Andrew S. Tanenbaum

IT & Computer - DXXIX

Andrew S. Tanenbaum

PREFACE The world has changed a great deal since the first edition of this book ap- appeared in 1992. Computer networks and distributed systems of all kinds have become very common. Small children now roam the Internet, where previously only computer professionals went. As a consequence, this book has changed a great deal, too. The most obvious change is that the first edition was about half on single- processor operating systems and half on distributed sysiems. 1 chose that format in 1991 because few universities then had courses on distributed systems and whatever students learned about distributed systems had to be put into the operat- operating sysiems course, for which this book was intended. Now most universities have a separate course on distributed systems, so it is not necessary to try to com- combine the two subjects into one course and one book. This book is intended for a first course on operating systems, and as such focuses mostly on traditional single-processor systems. 1 have coauthored two other books on operating systems. This leads lo two possible course sequences. Practically-oriented sequence; 1. Operating Systems Design and Implementation by Tanenbaum and Woodhull 2. Distributed Systems by Tanenbaum and Van Steen Traditional sequence: 1. Modern Operating Systems by Tanenbaum 2. Distributed Systems by Tanenbaum and Van Steen

XXIV PREFACE The former sequence uses M1N1X and the students are expected to experiment with MINIX in an accompanying laboratory supplementing the first course. The latter sequence'does not use M1N1X. Instead, some small simulators are available that can be used for student exercises during a first course using this book. These simulators can be found starting on the author's Web page: www.cs.vu.nl/-ast/by clicking on Software and supplementary material for my books . In addition to the major change of switching the emphasis to single-processor operating systems in this book, other major changes include the addition of entire chapters on computer security, multimedia operating systems, and Windows 2000, all important and timely topics. In addition, a new and unique chapter on operat- operating system design has been added. Another new feature is that many chapters now have a section on research about the topic of the chapter. This is intended to introduce the reader lo modern work in processes, memory management, and so on. These sections have numerous references to the current research literature for the interested reader. In addition, Chapter 13 has many

introductory and tutorial references. Finally, numerous topics have been added to this book or heavily revised. These topics include: graphical user intefaces, multiprocessor operating systems, power management for laptops, trusted systems, viruses, network terminals, CD- ROM file systems, mutexes, RAID, soft timers, stable storage, fair-share schedul- scheduling, and new paging algorithms- Many new problems have been added and old ones updated. The total number of problems now exceeds 450. A solutions manual is available to professors using this book in a course. They can obtain a copy from their local Prentice Hall representative. In addition, over 250 new references to the current literature have been added to bring the book up to date Despite the removal of more than 400 pages of old material, the book has increased in size due to the large amount of new material added. While the book is still suitable for a one-semester or two-quarter course, it is probably too long for a one-quarter or one-trimester course at most universities. For this reason, the book has been designed in a modular way. Any course on operating system should cover chapters ! through 6. This is basic material that every student show know. If additional time is available, additional chapters can be covered. Each ot them assumes the reader has finished chapters 1 through 6, but Chaps. 7 through 12 are each self contained, .so any desired subset can be used and in any order, depending on the interests of the instructor In the author's opinion, Chaps 7 through 12 are much more interesting than the earlier ones. Instructors should tell their students that they have to eat their broccoli before they can have the douhle chocolate fudge cake dessert I would like to thank the following people for their help in reviewing parts of the manuscript: Rida Bazzi, Riccardo Bettati, Felipe Cabrera, Richard Chapman John Connely, John Dickinson, John Elliott, Deborah Frincke, Chandana Gamage, Robbert Geist, David Golds, Jim Griffioen, Gary Harkin, Frans Kaashoek, Muk-

PREFACE XXV kai Krishnamoorthy, Monica Lam, Jussi Leiwo, Herb Mayer. Kirk McKusick, Evi Nemeth, Bill Potvin, Prasant Shenoy, Thomas, Skinner, Xian-He Sun, William Terry, Robbert Van Renege, and Maarten van Steen. Jamie Hanrahan, Mark Russinovich, and Dave Solomon were enormou.sly knowledgeable about Win- Windows 2000 and very helpful. Special thanks go to Al Woodhull for valuable reviews and thinking of many new end-of-chapter problems My students were also helpful wilh comments and feedback, especially Staas de Jong, Jan de Vos, Niels Drost, David Fokkema, Auke f-olkerts, Peter Groene- wegen, Wilto lbes, Stefan Jansen, Jeroen Ketema, Joen

Mulder, Irwin Oppenheim, Stef Post. Umar Rehman, Daniel Rijkhof, Maarten Sander, Maunts van der Schee, Rik van der Stoel, Mark van Dnet, Dennis van Veen, and Thomas Barbara and Marvin are still wonderful, as usual, each in a unique way Finally, last but not least, 1 would like to thank Suzanne for her love and patience, not to mention alt the druiven and kersen, which have replaced ihe sinasappelsap Andrew S. Tanenbaum

CONTENTS PREFACE 1 INTRODUCTION 1.1. WHAT IS AN OPERATING SYSTEM1' 3 1.1.1. The Operating Sysiem as an Extended Mach 1.1.2. The Operaiing Sysiem a.s a Resource Manag. 1.2. HISTORY OF OPERATING SYSTEMS 6 1.2.1. The First Genera.ion A945-55) 6 1.2.2. The Second Generation A955-65) 7 1.2.3. The Third Generation A965-1980) 9 1.2.4. The Fourth Generaiion A980-Present) 13 1.2.5. Ontogeny Recapitulates Phylogeny 16 1.3. THE OPERATING SYSTEM ZOO 18 1.3.1. Mainframe Operating Systems 18 t .3.2. Server Operating Systems 19 1.3.3. Multiprocessor Operating Systems 19 1.3.4. Personal Computer Operating Sysiems 19 1.3.5. Real-Time Operating Systems 19 1.3.6. Embedded Operating Sysiems 20 1.3.7. Sman Card Operating Sysiems 20

CONTENTS COMPUTER HARDWARE REVIEW 20 1.4.1. Processors 21 1.4.2. Memory 23 1.4.3. I/O Devices 28 I 4.4. Buses 3t OPERATING SYSTEM CONCEPTS 34 1.5.1. Processes 34 1.5 2. Deadlocks 36 1.5 3. Memory Managemeni 37 1.5.4. Input/Ouiput 38 1.5.5. Files 38 1.5.6. Security 41 1.5.7. The Shell 41 1.5.8. Recycling of Concepts 43 SYSTEM CALLS 44 1.6.1. Sysiem Calls for Process Managemeni 4g t .6.2. System Calls for File Management 50 1.6.3. System Calls for Directory Management 51 1.6.4. Miscellaneous System Calls 53 1.6.5. The Windows Win32 API 53 1.7. OPERATING SYSTEM STRUCTURE 56 1.7.1. Monoliihic Systems 56 1.7.2. Layered Systems 57 1.7.3. Virtual Machines 59 1.7.4. Exokernets 61 1.7.5. Client-Server Model 61 1.8. RESEARCH ON OPERATING SYSTEMS 63 1.9. OUTLINE OF THE REST OF THIS BOOK 65 1.10. METRIC UNITS 66 1.11. SUMMARY 67

CONTENTS 'X 2 PROCESSES AND THREADS 71 2.1. PROCESSES 71 2.1.1. The Process Model 72 2.1.2. Process Creation 73 2.1.3- Process Termination 75 2.1.4 Process Hierarchies 76 2 1 5 Process Stales 77 2.1.6- Implementation of Processes 79 2.2 THREADS 81 2.2.1. The Thread Model

81 2.2.2. Thread Usage 85 2-2.3. Implementing Threads in User Space 90 2-2.4. Implementing Threads in the Kernel 93 2-2.5. Hybrid Implementations 94 2.2.6. Scheduler Activations 94 2.2.7. Pop-lip Threads 96 2.2.8. Making Single-Threaded Code Multithreaded 97 2.3. INTERPROCESS COMMUNICATION 100 2.3.1. Race Conditions 100 2.3.2- Critical Regions 102 2.3.3. Mutual Exclusion wiih Busy Waning 103 2.3.4. Sleep and Wakeup 108 '2.3.5. Semaphores 110 2.3.6. Mutexts 113 2.3.7. Monitors 115 2.3.8. Message Passing 119 2.3.9. Barriers 123 2.4. CLASSICAL 1PC PROBLEMS 124 2.4.1. The Dining Philosophers Problem 125 2.4.2. The Readers and Wri.ers Problem 128 2.4.3. The Sleeping Barber Problem 129 2.5. SCHEDULING 132 2.5.1. Introduction to Scheduling 132 2.5.2. Scheduling in Batch Systems 138 2.5 3. Scheduling in Inieraciive Systems 142 2.5.4. Scheduling in Real-Time Systems 148 2.5.5. Policy versus Mechanism 149 2.5.6. Thread Scheduling 150

x CONTENTS 2.6. RESEARCH ON PROCESSES AND THREADS 151 2.7. SUMMARY 152 3 DEADLOCKS 159 3.1- RESOURCES 160 3.1.1. Preemptable andNonpreempiable Resources 160 3.1.2. Resource Acquisition 161 3.2. INTRODUCTION TO DEADLOCKS 163 3.2.1. Conditions for Deadlock 164 3.2.2. Deadlock Modeling 164 3.3. THE OSTRICH ALGORITHM 167 3.4. DEADLOCK DETECTION AND RECOVERY 168 3.4.1. Deadlock Detecnon with One Resource ot Each Type 168 3.4.2. Deadlock Deiecnon with Multiple Resource of Each Type 171 3.4.3. Recovery from Deadlock 173 3.5. DEADLOCK AVOIDANCE 175 3.5.!. Resource Trajectories 175 3.5.2. Safe and Unsafe Stales 176 3.5.3. The Banker's Algorithm for a Single Resource 178 3.5.4. The Banker's Algorithm for Multiple Resources 179 3.6. DEADLOCK PREVENTION 180 3.6.1. Attacking (he Mutual Exclusion Condition 180 3.6.2. Attacking the Hold and Wait Condition 181 3.6.3. Attacking the No Preemption Condition 182 3.6.4. Attacking the Circular Wait Condition 182 3.7. OTHER ISSUES 183 3.7.1. Two-Phase Locking 183 3.7.2. Nonresource Deadlocks 184 3.7.3. Starvation 184 3.8. RESEARCH ON DEADLOCKS 185 3.9. SUMMARY 185

CONTENTS XI 4 MEMORY MANAGEMENT 189 4 I BASIC MEMORY MANAGEMENT 190 4.1.] Monoprogramming without Swapping or Paging 190 4.1.2. Multiprogramming with Fixed Partitions !9I 4.1.3. Modeling Multiprogramming 192 4.1.4 Analysis of Multiprogramming System

Performance 194 4.1.5. Relocation and Protection 194 4.2. SWAPPING !96 4.2.1. Memory Management with Bitmaps 199 4.2.2. Memory Management with Linked Lists 200 4.3. VIRTUAL MEMORY 202 4.3.1. Paging 202 4.3.2. Page Tables 205 4.3.3. TLBs—Translation Lookaside Buffers 211 4.3.4. inverted Page Tables 213 4.4. PAGE REPLACEMENT ALGORITHMS 214 4.4.1. The Optima! Page Replacement Algorithm 215 4.4.2. The Not Recently Used Page Replacement Algorithm 216 4.4.3. The First-In, First-Out 217 4.4.4. The Second Chance Page Replacement Algorithm 2]7 4.4.5. The Clock Page Replacement Algorithm 218 4.4.6. The Least Recently Used 218 4.4.7. Simulating LRU in Software 220 4.4.8. The Working Set Page Replacement Algorithm 222 4.4.9. The WSClock Page Replacement Algorithm 225 4.4.:. Summary of Page Replacement Algorithms 227 4.5. MODELING PAGE REPLACEMENT ALGORITHMS 228 4.5.1. Belady's Anomaly 229 4.5.2. Stack Algorithms 229 4.5.3. The Distance String 232 4.5.4. Predicting Page Fault Rates 233 4.6. DESIGN ISSUES FOR PAGING SYSTEMS 234 4.6.1. Local versus Global Allocation Policies 234 4.6.2. Load Control 236 4.6.3. Page Size 237 4.6.4. Separate Instruction and Data Spaces 239

XI] CONTENTS 4.6.5. Shared Pages 239 4.6.6. Cleaning Policy 241 4.6 7. Virtual Memory Interface 241 4 7 IMPLEMENTATION ISSUES 242 4.7.1. Operating System Involvement with Paging 242 4.7.2. Page Fault Handling 243 4.7.3. Instruction Backup 244 4.7.4. Locking Pages in Memory 246 4.7.5. Backing Store 246 4.7.6. Separation of Policy and Mechanism 247 4.8. SEGMENTATION 249 4.8.1. Implementation of Pure Segmentation 253 4.8.2. Segmentation with Paging. MULTICS 254 4.8.3. Segmentation with Paging: The Intel Pentium 257 4.9. RESEARCH ON MEMORY MANAGEMENT 262 4.10. SUMMARY 262 5 INPUT/OUTPUT 269 5.1 PRINCIPLES OF I/O HARDWARE 269 5.1.1. I/O Devices 270 5.1.2. Device Controllers 271 5.1.3. Memory-Mapped I/O 272 5.1.4. Direct Memory Access 276 5.1.5. Interrupts Revisited 279 5.2. PRiNCIPLES OF I/O SOFTWARE 282 5.2.1. Goals ot the I/O Software 283 5.2.2. Programmed I/O 284 5 2.3. lnterrupt-Dnven I/O 286 5 2.4. I/O Using DMA 287 5.3. I/O SOFTWARE LAYERS 287 5.3.1. interrupt Handlers 287 5.3.2. Device Drivers 289

5-3.3. Device-Independent I/O Software 292 5.3.4. User-Space I/O Software 298 5.4. DISKS 300 5.4.1. Disk Hardware 300 5.4.2. Disk Formatting 315 5.4.3. Disk Arm Scheduling Algorithms 318 5.4.4. Error Handling 322 5.4.5.

Stable Storage 324 5.5. CLOCKS 327 5.5-1 Clock Hardware 328 5.5.2. Clock Software 329 5.5.3. Soft Timers 332 5.6. CHARACTER-ORIENTED TERMINALS 333 5.6.1. RS-232 Terminal Hardware 334 5.6.2. Input Software 336 5.6.3. Output Software 34! 5.7. GRAPHICAL USER INTERFACRS 342 5.7.1. Persona! Computer Keyboard, Mouse, and Display Hardware 343 5.7.2. Input Software 347 5.7.3. Output Software for Windows 347 5.8. NETWORK TERMINALS 355 5.8.1. The X Window System 356 5.8.2. The SUM Network Terminal 360 5.9. POWER MANAGEMENT 363 5.9.1. Hardware Issues 364 5.9.2. Operating System Issues 365 5.9.3. Degraded Operation 370 5.10. RESEARCH ON INPUT/OUTPUT 371 5.11. SUMMARY 372

XIV 6 FILE SYSTEMS 379 FILES 380 .1. FilcNai ing 380 ;ture 382 .3. File Types 383 .4. File Access 385 5. File Attributes, 386 .6. File Operations 387 .7. An Example Program Using File System Calls, 6.1.8. Memory-Mapped Files 391 6.2- DIRECTORIES 393 6.2.1- Single-Level Directory Systems 393 6.2.2. Two-level Directory Systems 394 6.2.3. Hierarchical Directory Systems 395 6.2.4. Path Names 395 6.2.5. Directory Operations, 398 6.3. FILE SYSTEM IMPLEMENTATION 399 6.3.1. File System Layout 399 6.3.2. Implementing Files 400 6.3.3. Implementing Directorie.s 405 6.3.4. Shared Files 408 6.3.5. Disk Space Management 410 6.3.6. File System Reliability 416 6.3.7. File System Performance 424 6.3.8. Log-Structured File Systems 428 6.4. EXAMPLE FILE SYSTEMS 430 6.4.1. CD-ROM File Systems 430 6.4.2. The CP/M File System 435 6.4.3. The MS-DOS File System 438 6.4.4. The Windows 98 File System 442 6.4.5. The UNIX V7 File System 445 6.5. RESEARCH ON FILE SYSTEMS 448 6.6 SUMMARY 448

CONTENTS XV 7 MULTIMEDIA OPERATING SYSTEMS 453 7.] INTRODUCTION TO MULTIMEDIA 454 7.2 MULTIMEDIA FILES 458 7.2.1. Audio Encoding 459 7.2.2. Video Encoding 461 7.3. VIDEO COMPRESSION 463 7.3.1. The JPEG Standard 464 7.3.2. The MPEG Standard 467 7.4. MULTIMEDIA PROCESS SCHEDULING 469 7.4.1. Scheduling Homogeneous Processes 469 7.4.2. Genera] Real-Time Scheduling 470 7.4.3. Rate Monotonic Scheduling 472 7.4.4. Earliest Deadline First Scheduling 473 7.5. MULTIMEDIA FILE SYSTEM PARADIGMS 475 7.5.1. VCR Control Functions 476 7.5.2. Near Video on Demand 478 7.5.3. Near Video on Demand with VCR Functions 479 7.6. FILE PLACEMENT 481

7.6.1. Placing a File on a Single Disk 481 7.6.2. Two Alternative File Organization Strategies 482 7.6.3. Placing Files for Near Video on Demand 486 7.6.4. Placing Multiple Files on a Single Disk 487 7.6.5. Placing Files on Multiple Disks 490 7.7. CACHING 492 7.7.1. Block Caching 492 7.7.2. File Caching 494 7.8. DISK SCHEDULING FOR MULTIMEDIA 494 7.8.1. Static Disk Scheduling 495 7.8.2. Dynamic Disk Scheduling 496 7.9. RESEARCH ON MULTIMEDIA 498 7.10. SUMMARY 499

XVI CONTENTS 8 MULTIPLE PROCESSOR SYSTEMS 503 8.!. MULTIPROCESSORS 506 8.1.!. Multiprocessor Hardware 506 8.1.2. Multiprocessor Operating System Types 513 8.1.3. Multiprocessor Synchronization 516 8.1.4. Multiprocessor Scheduling 521 tt.2. MULT1COMPUTERS 526 8.2.1. Multicomputer Hardware 527 8.2.2. Low-Level Communication Software. 531 8.2.3. User-Level Communication Software 534 8.2.4. Remote Procedure Call 537 8.2.5. Distributed Shared Memory 540 8.2.6. Multicomputer Scheduling 544 8.2.7. Load Balancing 545 8.3. DISTRIBUTED SYSTEMS 549 8.3.1. Network Hardware 55 I 8.3.2. Network Services and Protocols 553 8.3.3. Document-Based Middleware 558 8.3.4. File System-Based Middleware 559 8.3.5. Shared Object-Based Middleware 565 8.3.6. Coordmation-Based Middleware 572 8.4. RESEARCH ON MULTIPLE PROCESSOR SYSTEMS 577 8.5. SUMMARY 577 SECURITY 583 9.1. THE SECURITY ENVIRONMENT 584 9.1.1. Threats 584 9.1.2. Intruders 585 9.1.3. Accidental Data Loss 586 9.2. BASICS OF CRYPTOGRAPHY 587 9.2.1. Secret-Key Cryptography 588 9.2.2. Public-Key Cryptography 588

9.2.3. One-Way Functions 589 9.2.4. Digital Signatures 590 9 3. USER AUTHENTICATION 591 9.3.!. Authentication Using Passwords, 592 9.3.2. Authentication Using a Physical Object 601 9.3.3. Authentication Using Biometrics 603 9.3.4. Ccmniermeasures 606 9.4 ATTACKS FROM INSIDE THE SYSTEM 606 9.4.1. Trojan Horses 607 9.4.2. Login Spoofing 608 9.4.3. Logic Bombs 609 9.4.4. Trap Doors 6!0 9.4.5. Buffer Overflow 610 9.4.6. Genene Security Attacks 613 9.4.7. Famous Security Flaws, 6!4 9.4.8. Design Principles for Security 616 9.5. ATTACKS FROM OUTSIDE THE SYSTEM 6!7 9.5.1. Virus Damage Scenarios 618 9.5.2. How Viruses Work 619 9.5.3. How Viruses Spread 626 9.5.4. Antivirus, and Anti-Antivirus Techniques 628 9.5.5. The Internet Worm 635 9-5.6. Mobile Code 637 9.5.7. Java Security 642

9.6. PROTECTION MECHANISMS 645 9.6.1. Protection Domains 645 9.6.2. Access Control Lists 647 9.6.3. Capabilities 650 9.7. TRUSTED SYSTEMS 653 9.7.1. Trusted Computing Base 654 9.7.2. Formal Models of Secure Systems 655 9.7.3. Multilevel Security 657 9.7.4. Orange Book Security 659 9.7.5. Covert Channels 661 9 8. RESEARCH ON SECURITY 665 9.9. SUMMARY 666

XVIII CONTENTS 10 CASE STUDY 1: UNIX AND LINUX 671 0 1 HISTORY OF UNIX 672 10. .1. 10.1.2. 10. 10. 10. 10. 10. .3. .4. .5. .6. .7. UN1CS 672 PDP-11 UNIX Portable UNIX Berkeley UNIX Standard UNIX MIN1X 677 Linux 678 673 674 675 676 !0 2. OVERVIEW OF UNIX 681 10.2.1. UNIX Goals 681 10.2.2. Interfaces to UNIX 682 10.2.3. The UNIX Shell 683 10.2.4. UNIX Utility Programs 686 10.2.5. Kernel Structure 687 10.3. PROCESSES IN UNIX 690 10.3.1. Fundamental Concepts 690 10.3.2. Process Management System Calls in UNIX 692 10.3.3. Implementation of Processes in UNIX 699 10.3.4. Booting UNIX 708 10.4. MEMORY MANAGEMENT IN UNIX 710 10.4.1. Fundamental Concepts 71! 10.4.2. Memory Management System Calls in UNIX 714 10.4.3. Implementation of Memory Management in UNIX 715 10.5. INPUT/OUTPUT IN UNIX 723 10.5.1. Fundamental Concepts 724 10.5.2. Inpui/Output System Calls in UNIX 726 10.5.3. Implementation of Input/Output m UNIX 727 10.5.4. Streams 730 10.6. THE UNIX FILE SYSTEM 732 10.6.1. Fundamental Concepts 732 10.6.2. File System Calls in UNIX 736 10-6.3. Implementation of the UNIX File System 10.6.4. NFS: The Network File System 747

CONTENTS 10.7. SECURITY IN UNIX 753 10-7.1. Fundamental Concepts 753 10.7-2. Security System Calls in UNIX 755 10.7-3. Implementation of Security in UNIX 756 108 SUMMARY 757 11 CASE STUDY 2: WINDOWS 2000 763 11.I. HISTORY OF WINDOWS 2000 763 ll.I.K MS-DOS 763 11.1.2. Windows 95/98/Me 764 11.1.3. WindowsNT 765 11.1.4. Windows 2000 767 11.2. PROGRAMMING WINDOWS 2000 771 11.2.1. The Win32 Application Programming Interface 772 11.2.2. The Registry 774 11.3. SYSTEM STRUCTURE 778 11.3.1. Operating System Structure 778 11.3.2. Implementation of Objects 787 11.3.3. Environment Subsystems 792 11.4. PROCESSES AND THREADS IN WINDOWS 2000 796 11.4.1. Fundamental Concepts 796 11.4.2. Job, Process, Thread and Fiber Management API Calls 799 11.4.3. Implementation of Processes and Threads 802 11.4.4. MS-DOS

Emulation 809 11.4.5. Booting Windows 2000 820 11.5. MEMORY MANAGEMENT 811 11-5.1. Fundamental Concepts 812 11.5.2. Memory Management System Calls 816 11.5.3. Implementation of Memory Management 8l7 11.6. INPUT/OUTPUT IN WINDOWS 2000 824 1 1.6.1. Fundamental Concepts 824 11.6.2. Input/Output API Calls 825 11.6.3. Implementation of I/O 827 11.6.4. Device Drivers 827

CONTENTS 11.7. THE WINDOWS 2000 FILE SYSTEM 830 11.7.1. Fundamental Concepts 830 t 1.7 2. Rte System API Calls in Windows 2000 83 t 117.3. Implementation of the Windows 2000 File System 833 118. SECURITY IN WINDOWS 2000 844 11.8.1. Fundamental Concepts H45 11.8.2. Security API Calls 847 11.8.3. Implementation of Security 84* 11.9. CACHING IN WINDOWS 2000 849 tl 10. SUMMARY 851 12 OPERATING SYSTEM DESIGN 855 I2.I. THE NATURE OF THE DESIGN PROBLEM 856 12.1.1. Goals 856 12.1.2. Why.is i! Hard to Design an Operating Systems" 857 12 2. INTERFACE DESIGN 859 12.2.1. Guiding Principles 859 12.2.2. Paradigms 861 12.2.3. The System Call Interface 864 12.3 IMPLEMENTATION 867 12.3.1. System Structure 867 12.3.2. Mechanism versus Policy 870 12.3.3. Orthogonality 871 12.3.4. Naming 872 12.3.5. Binding Time 874 12.3.6. Static versus Dynamic Structures 875 12.3.7. Top-Down versus Bottom-Up Implementation 876 12.3.8. Useful Techniques 877 12.4. PERFORMANCE 882 12.4.1. Why are Operating Systems Slow? 882 12.4.2. What Should be Optimized? 883 12.43. Space-Time Trade-offs 884 12.4.4. Caching 887

12.4.5. Hints 888 12.4.6. Exploiting Locality 888 12.4.7. Optimize the Common Case 889 12.5. PROJECT MANAGEMENT 889 12,5.1 The Mythical Man Month 890 12.5.2. Team Structure 891 125.3. The Rote of Experience 893 12 5.4. No Silver Buttet 894 12.6. TRENDS tN OPERATING SYSTEM DESIGN 894 12.6.1. Large Address Space Operating Systems, 894 12.6 2. Networking 895 12.6.3. Parallel and Distributed Systems 896 12.6.4. Multimedia 896 12.6.5. Battery-Powered Computers 896 12.6.6. Embedded Systems 897 12.7. SUMMARY 897 13 READING LIST AND BIBLIOGRAPHY 13.1. SUGGESTIONS FOR FURTHER READING 901 .1. Introduction and General Works 902 .2. Processes and Threads 902 .3. Deadlocks 903 .4. Memory Management 9A3 .5. Input/Output 903 .6. File Systems 904 .7. Multimedia Operating Systems 905 .8. Multiple Processor

Systems 906 .9. Secunly 9(>7 .10. UNIX and Linux 908 .11. Windows 2000 909 .12. Design Principles 910 13.2 ALPHABETICAL BIBLIOGRAPHY 911 901 INDEX 935

1 INTRODUCTION A modem computer system consists of one or more processors ome main memory, disks, printers, a keyboard, a display, network interfaces, and other input/output devices. All in all, a complex system. Writing programs that keep track of all these components and use them correctly, let alone optimally, is an extremely difficult job. For this reason, computers are equipped with a layer of software called the operating system, whose job is to manage alt these devices and provide user programs with a simpler interface to the hardware. These sys- systems are the subject of this, book. The placement of the operating system is shown in Fig I -1. At the bottom is the hardware, which, in many cases, is itself composed of two or more levels (or layers). The lowest level contains physical devices, consisting of integrated cir- circuit chips, wires, power supplies, cathode ray tubes, and similar physical devices. How these are constructed and how they work are the provinces Of the electrical engineer. Next comes the microarchitecture level, in which the physical devices are grouped together to form functional units. Typically this level contains some reg- registers internal to the CPU (Central Processing Unit) and a data path containing an arithmetic logic unit. In each clock cycle, one or two operands are fetched from the registers and combined in the arithmelic logic unil (for example, by dddition or Boolean AND). The result is stored in one or more registers. On some machines, the operation of the data path is controlled by software, called the microprogram. On other machines, it is controlled directly by hardware circuits. 1

Banking system Cow*, reser Edi anon „„, Web Command inlerpreler Operating syslem no- Physical anguage iteclure devices The purpose of the data padi is to execute some set of instructions. Some of these can be carried out in one data path cycle; others may require multiple data path cycles. These instructions may use registers or other hardware facilities Together, the hardware and instructions visible to an assembly language program-programmer form the ISA (Instruction Set Architecture) level. This level is ofien called machine language. The machine language iypically has between 50 and 300 instructions, mostly for moving data around the machine, doing arithmetic, and comparing values. In this level, the input/output devices are

controlled by loading values into special device registers. For example, a disk can be commanded to read b> loading Ihe values of the disk address, main memory address, byte count, and direction (read or write) into its registers. In practice, many more parameters are needed, and the status returned by the drive afier an operation is highly complex Furthermore, for many I/O (Input/Output) devices, timing plays an important role in the program-programming. To hide this complexity, an operating system is provided. It consists ot a layer of software that (partially) hides the hardware and gives the programmer a more convenient set of instructions to work with. For example, read block from file is conceptually simpler than having to worrj about die details ot moving disk heads, waiting for them to settle down, and so on. On top of the operating system is the re^t of the system software Here we: find ihe command interpreter (shell), window systems, compilers, editors, and similar application-independent programs H is important 10 realize thai the>e programs are definitely not part of the operating system, even though ihey are typ- typically supplied by the computer manufacturer This is a crucial, hut subile, point The operaiing system is (usually) ihat portion ol the software ihat runs in kernel mode or supervisor mode. Ii is protected from user tampering by the hardware (ignoring for the moment some older or low-end microprocessors thai do nol have

hardware protection at all)- Compilers and editors run in user mode. It a user does not like a particular compiler, het is free to write his own if he so chooses; he is not free to write his own clock interrupt handier, which is part of the operat- operating system and is normally protected by hardware against attempts by users to modify it. This distinction, however, is sometimes blurred in embedded systems (which may not have kernel mode) or interpreted systems (such as Java-based operating systems that use interpretation, not hardware, to separate the components). Still, for traditional computers, the operating system is what runs in kernel mode. That said, in many systems there are programs that run in user mode but which help the operating syslem or perform privileged functions- For example, there is often a program that allows users to change theii passwords. This pro- program is not part of the operating system and does not run in kernel mode, but it clearly carries out a sensitive function and has to be protected in a special way. In some systems, this idea is carried to an extreme form, and pieces of whai is traditionally considered to be the operating system (such as the file system) run in user space. In such systems, it

is difficult to draw a clear boundary Everything running in kernel mode is clearly part of the operating system, but some prograns running outside it are arguably also part of it, or at least closely associated with it. Finally, above the system programs come the application programs. These programs are purchased or written by the users to solve their particular problems, such as word processing, spreadsheets, engineering calculations, or storing infor- information in a database. 1.1 WHAT IS AN OPERATING SYSTEM? Most computer users have had some experience with an operating system, but it is difficult to pin down precisely what an operating system is. Part of the prob- problem is that operating systems perform two basically unrelated functions, extending the machine and managing resources, and depending on who is doing the talking, you hear mostly about one function or the other. Let us now look at both. 1.1.1 The Operating System as an Extended Machine As mentioned earlier, the architecture (instruction set. memory organization, I/O, and bus structure) of most computers at the machine language level is primi- primitive and awkward to program, especially tor input/output. To make this point more concrete, let us briefly look at how floppy disk I/O is done using the NEC t "He" should be read as "he ar she" throughout the book

4 INTRODUCTION CHAP. 1 PD761) compatible controller chips used on inosi Intel-based personal computers. (Throughout this book we will use the terms "floppy disk" and "diskette'" inter- interchangeably.) The PD765 has 16 command*, each specified by loading between 1 and 9 bytes inio a device register These commands are for reading and writing data, moving the disk arm, and formatting tracks, as well as initializing, sensing, resetting, and recalibrating the controller anil tbe drives The most basic commands are read and write, each of whicb requires 13 parameters, packed into 9 bytes. These parameters specity such items a.s the address of [he disk block !o be read, the numher of sectors per (rack, the recording mode used on the physical medium, lhe miersecior gap spacing, and whai lo do with a deleted-data-address-mark. If you do not undersland thts mumbo jumbo, do no! worry, lhat is precisely (he poinl—i! is raiher esoteric. When lhe operation is completed, lhe controller chip reiurns 23 status and emir fields packed into 7 bytes. As if this were not enough, lhe floppy disk programmer must also be con- stanily aware of whether the motor is on or off If ihe motor is off, u must be turned on (with a long startup delay) before data can be read or written The motor cannot be left on too long, however, or ihe floppy disk will \«ar out The programmer is thus

forced to deal with tbe trade-off beiween long startup delays versus wearing out floppy disks (and losing ihe data on (hem). Without going into lhe real details, it should be clear thai (he average pro- programmer probably does not want to get too intnnaiely involved with the program- programming of floppy disks (or hard disks, which arc just as complex and quite dif- ferenl). Instead, what the programmer wants is a simple, high-level abstraction to deal with. In the case of disks, a typical abstraction would be tba(lhe dtsk con- contains a collection of named files. Each file can be opened tor leading or writing, lhen read or written, and finally closed. De(ails such as whether or not recording should use modified frequency modulation and what lhe current slate of (lie motor is should not appear in the abstraction presented lo (he user The program lhat hides the trulh about the hardware from the programmer and presents a nice, simple view of named files lhai can be read and written is, of course, the operating system. Just as lhe operating system shields (he programmer from ihe disk hardware and presents a simple file-oriented interface, it also con- conceals a lot of unpleasant business concerning interrupts, timers, memory manage- management, and other low-level features. In each case, the abstraction offered by the operating system is simpler and easier to use than that ottered hy the underlying hardware. !n this view, the function of lhe operating system is lo prcsenl lhe user with the equivalent of an extended machine or virtual machine that is easier to pro- program than the underlying hardware. How the operating system achieves this goal is a Jong story, which we will study in detail throughout this book Tu summarize it in a nutshelf, the operating system provides a variety or services that programs can ohtain using special instructions called system calls. We will examine some of the more common system talfs later in this chapter

SEC I I WHAT IS AN OPERATING SYSTEM ' 5 1.1.2 The Operating System as a Resource Manager The concepi ot lhe operating system as primarily providing its users wtih a convenient interface is a top-down view, A.n alternative, bottom-up, view holds lhat lhe operating system is there to manage all lhc pieces ot a complex system. Modem computer;, consist or processors, memories, timers, disks, mice, network interfaces, print-*;,, and a wide variety of other devices. In the alternative view, the job or lhe operating system is to provide lor an orderly and controlled alloca- allocation of lhe processors memories, and I/O devices among lhe various programs competing for them. Imagine what would happen if three programs running on some computer all

tried to print their output simultaneously on the same printer The first lew lines of printout might be from program i, the next few from program 2, then some rrom program 3, and so forth. The result would be chaos. The operating system can bring order to the potential chaos by buffering all the output destined for lhe primer on the disk. When one program is finished, the Operating system can then copy its ouiput from the disk file where it has been stored to the printer, while at the same time the other program can continue generating more output, oblivious to the tact that the output is not really going to the printer (yet) When a computer (or network) has multiple users, the need for managing and protecting the memory, I/O devices, and other resources is even greater, since the users might otherwise interfere with one another. In addition, users often need to share not only hardware, but information (files, databases, etc.) as well. In short, this view of the operating system holds that its primary task is to keep track i>f who is using which resource, to grant resource requests, to account for usage, and to mediate conflicting requests from different programs and users- Resource management includes multiplexing (sharing) lesources in two ways: m time and in space. When a resource is time multiplexed, different programs or users take turns using it. First one ot them gets to use the resource, then another, and so on. For example, with only one CPl) and multiple programs that warn to run on it. the operating system first allocates lhe CPU to ooe program, men after it has run long enough, another one gets to use the CPU, then another, and then eventually the first one again. Determining how the resource is time multi- multiplexed—who goes next and for how long—is the task of me operating system. Another example of time multiplexing is sharing the prititer When multiple print jobs are queued up for printing on a single printer, a decision has to be made about which one is to be printed next. The other kind ot multiplexing is space multiplexing Instead of the custo- customers taking turns;, each one gets part of the resource. For example, mam memory is normally divided up among several running programs, so each one can he resident at the same time (for example, in order to take turns using the CPU). Assuming there is enough memory to hold multiple programs, it is more efficient to hold several programs in memory at once rather than give one of them all of it,

multiplexed is :he (hard) disk. In m<m> svsrems a single disk can hold files flora manj user, at the same ltme. Allocating disk <paee and keeping irack of who is using which disk blocks is i tjpical operating system lesource management task. 1.2 HISTORY OF OPERATING SYSTEMS Operating s>sIlhis huvu been evolving ilmiugh the yen-, In the lollowinj sections we will bnefl> look at a few ot the highlights. Since operating systems have historically been closely tied tn the jichiiecmre ot the compuiei s on which Ihey run. we will look ai successive generations ot computers 10 see what theif operating systems were like This mapping of operating system gencianoliS io compulei generations is Lrude. but U docs provide some strutiurc where thec- would otherwise be none The firsi ime digual compuier was designed by the English mmheiiutic-ian Charles Babbage A792-1871) Although B.ibbage speni most ot his life and lor- tune trying to build his 'analytical engine." he never got it working properlv hecause u was purely mechanical, and the technology of his day could noi pro- produce the required wheels, gears, and cogs to llic high precision lhal he needed Needles to say. the analytical engine did not have an operating system As an interesting historical aside. Bahbagc re;ihzcd thai he would need software for his analytical engine, so lit- hned .i young woman named Adj Lovelace, who was the daughier of the tamed Bniish poel Lord Bjron. as lhe world's tirsl programmer The programming language Adac" is named afiei her 1.2.1 The First Generation A945-55) Vacuum Tubes and Plugboards A Her Babbage's unsuccessful efforts, liulc progress v,as made in consirudin.4 digital computers unli) Wi)rld Wai II Around the tiHc!-l94Os, Howard Aikcii at Harvard, John von Neam.inn a! the lnstiiule tor Advanced Sludy in PrUicetoii. J. Presper Eckert and William Mautfiley <t! the Uttn-ersiiy <il Penitsyhania. and Konrad Zuse in Germanv, among others, all succeeded in building calculating engines. The first ones used mechanical relays hut were eery slow, wiih cycle times, measured in seconds Relays were laier leplaced by vacuum tubes Iliese machines were enormous, tilling up entire roams with tens ot thousands of vacuum tubes, but they were still millions ot times slawer than even the cheapest personal computers available loday. In these early days, a single group ol people designed, huilt, programmed, operated, and maintained each machine. All programming was done in absolute

machine language, otten by wiring up plugboards to control Ihe machine's haste fundsFX1. Programming languages were unknown (even assembly language was unknown). Operating sysiems were unheard ot. The usual mode ol

operation was for ibe programmer to sign up for a block ot time on the signup sheet on the wall, then come down 10 the machine room, itisert his or her plugboaid into lhe com- puier, and spend ihe next tew hours hoping that none ol the 20,000 or so vacuum tubes would burn oui during the run Virtually all the problems were straightfor- straightforward numerical calculations, such as grinding oui tables of sines, cosines, and log- logarithm <; By the early 1950s, ihe routine had (mpioved somewhat with tile introduction of punched cards. 1! was now possible to write progiams on curd.s and read them in instead of using plugboards; otherwise, the procedure was the saiiie 1.2.2 The Second Generation A955-65) Transistors and Batch Systems The introduction of ihe transistor in the mid-1950s changed the picture radi- radically Computers became reliable enough that they could be- manufactured and .sold (o paying customers with the expectation that they would Continue to func- function long enough to get some useful work done. For the tir.st time, there wat, a clear separation between designers, builders, operators, programmers, and mainte- maintenance personnel. These machines, now called mainframes, were locked away in specially air conditioned computer rooms, with staffs ot professional operators to run them- Only big corporations or major government agencies or universities could afford the multimillion dollar price tag. To run a job lie.a program or sei of pro- programs), a programmer would first write the program on paper (in FORTRAN or assembler), then punch it on cards. He would then bring the curd deck down io Ihe input room and hand it to one of the opeiators and go drink coffee until the output was ready. When the computer finished whatever joh it was currently running, an opera- operator would go over to the printer and tear off the output and carry it over to the out- output room, so that the programmer could collect it later Then he would take one of the card decks that had been brought trom the input room and read it in. If the FORTRAN compiler was needed, the operator would have to get it trom a file cabinet and read it in. Much computer time was wasted while operators were walking around the machine room- Given the high cost of the equipment, it is not surprising that people quickly looked for ways to reduce the wasted time The solution generally adopted was the batch system. The idea behind it wa? to collect a tray full of robs in the input room and then read them onto a magnetic tape using a small (relatively) inexpen- inexpensive computer, such as the IBM 1401, which was very good at reading cards, copying tapes, and printing output, but not at all good at numerical calculations.

8 INTRODUCTION CHAP Oihcr. much moic expensive machines, siich as ihe IBM 7094. were usedfoi t real compnting. This situation is shown in Fit;. 1-2. Figure 1-2. An uirlj bi1Lh '.yslan (a) Prc 1401 Itfjdi ba*li of lobs onto Upi- (c) Ope 7094 Joes computing (e) Operator carries ■ @ 1401 print- After aboul an hour of collecting a baler* of |<>hs. Ihe lapc was rewocind and brought into the machine rc.i.m, where n wa> mountt-d cm a tape drive. Ttie opera- operator then loaded a .special program (the anceslor of Hidays operating sysiem). which read the first job Ironi tape and ran it The output wlw written onto a sec- second tape, instead of being printed. Aftei each job finished. Ihe operating system automatically read the next job from Ihe tape and began ronnmg H. Wlien I he whole hate the input tape with the next batch, and brought the output 1. ing off line (i.e.. not connected to the main computer) 1401 The net] i $JOB card, specifying the maximum run tune m minutes, th be charged, and the pioeramnier's name. Tiien can.e a $F0RTRAN curd, the operating system to load the FORTRAN compiler from the system was followed by the program to he compiled, and then a SLOAD card, the operating system to load the object |irdgraii> |iist compiled (Coinpi grams were often written i,n scratch tapes and liad to he li.aJed exj)licitl> came the $RUN caid, telling the operating systen, to run ihe program data ioliowing it Finalh, the $END caid maiked the end i,f the ji.h. The itive control cards were the fmerunners or nicdern |Ob eontrol languages a mand interpret ir pntit- nberto telims ■ipe Tt renting >d pro-) Next ■ith tne nd-ge i mputers engiueenng cakuiations. such as solving tile partial chf|erential equati often occur in physics and engineering They v,L-ic largely progianinieiJ TRAK and assembly language. Typical operaiing systems were FMS (the Monitor System) 3nd IBSYS, IBM's operating system for the 7094. ns that n FOR- F-ortr.in

HISTORY OF OPERATING SYSTEMS 1.2.3 The Third Generation A965-1980) ICs and Multiprogramming By ihe early 1960s, most computer manufacturers had two distinct, and tolaiiy incompatible, produci lines. On the one hand thcie were ihc wcird-tinemed. large-scale scientific computers, such as the 7094, which were used tor numerical calculations in science and engineering. On the other hand, there were the character-oriented, commercial computers, such as 'he 1401, which were widely used tor tape sorting and printing by hanks and insurance companies Developing and maintaining two completely different product lines was ai> expensive proposition for the manufacturers In addition, many new ci.n.puter customers initially needed a

small machine but later outgrew it and wanted a bigger machine that would run al! their old programs, hut faster IBM attempted 10 si.lve both of these problems ai a single stroke by introduc- introducing the Systetn/360. The 360 was a series ot sotiware-conipatibk.- machines rang- ranging from 1401-sized ti, much more powerful llian the 7094 The machines dif- differed (inly in piicc and performance (maximum memtiiv, processor speed. nu,Tihcr of I/O devices permitic-d, and so forth) Since all tl>e machines had ihL- same architecture and instruction set, programs wmiei, ior one n.aehme amliT run on all the others, at least in lhcory. Funherm<,re. ihe 3M> was designed to handle both scientific (i.e , numerical) and commercial computing Thus a single family of machines could satisfy the needs of all customers In subsequent years, IBM has ci.me out with compatihle successors to the- 160 line, using more nu.dern ic- cTinuT- ogy, known as the 370. 4300.10SO, and 1090 series

10 iNTkODLC I ION t-'UAP I Tlit- 360 was (he tusi majoi computer lino to use (small scale) Integrated Or- cuit*. <rCV. thus providing a major price/performance ;tdv;inlage o\er the second- generation machines. v\mdi were built up from indiwdu.il transistors, ll was an adopted by all lhe other majoi manufacturers The descendants of these machines are stili in use ,il computer centers today. Nowadays they are often used lor World Wide Weh site thai must pioccss thi.usarids cif requests j,er second. fhf greatest strength of the "one tamtl) ' idea was simultaneously its greatest weakness. The intention w.is thai all softwaie. including the operating system, OS/360 had to work on all i.iodel.s It had 10 ruti on small systems, whicn often lust replaced 1401s for copying aircis tu tape, and on very laigc systems, which often replaced 7094s lor doing weather forecasting and other heavy computing It had tu be good oti .systems with tew peripherals and on systems with nwuv pen- menis. Abo\eall, it had to be clTtcient for all elf these JtfteretitiK.es. There was no way that IBM (Or anybody else) UiUlci wine a piece Ol software to meet ad (hose conflicting requirements The result was an enormous and extraordinarily eotnplex operating system, piobalJy two to thiec orders ol magni- magnitude larger than FMS ll consisted or millions ot Imps of assembly language writ- written by thousands of programmers, ,md contained thousands upon (housaiids of bugs, which necessitated a continuous stream ol" new releases ui an attempt to correct then,. Each new iclease fixed some bugs and introduced new ones, so the numher ol bugs probabl; remained constant m time One ot the designers ot OS/360. Fred Biooks, subsequently wiole a witty and

incisive book (Brooks, 1996) describing his experiences with OS/lfn). While it would be impossible to summarize the hook heie. suffice i(to say that the cci\er shows a herd of prehistoric beasts siutk in ,i lai |iil. The cover ot Silherschat7 el al. B000) makes a similar point about tij.erating systems beina dmos.»irs. Despite its enormous size and problems. OS/3W) and the similai thud- getieratn>fl operatjng syst^ins produced by other comiiUler inaiiLiIucliirers actually sjtisfied most oi their customers, reasonably well- They also popularized seveial key techniques ahsent in seconci-getieiaiion ojieratmg systenis Probably the must important ot these was multiprogramming. On the 7094, when tlvc current job paused to wait for a tape or other i/O operation hi complex, the CPU simply sal idle until the I/O finished With heavily CPU-l.ound scientific calculations I/O is infrequent, so this wasted tune is not signitkani With conimeicial data pioccss- mg, the I/O wait time can often be- 80 or 90 peicent of the toi.il time, so something had to be done to avoid having lhe (expensivei CPV be idle so much different job in each petition, .is shown in Fig 1-4 While one job was waiting toi I/O lo complete, another job could be Using the <L PL) It enough)tibs could |,e held in main memory at once, the CpU could he kept busy nearly 100 percent of

SEC (2 HISTORY OF OPERATING SYSTFMS II the lime Having multiple jobs safely in memory a! once require-, special hardware to protect each job against snooping and mischief by the other ones, but (he 360 and other mird-generation systems were equipped with (his hardware. ability to omputcr Figure !»4- A nuifiiprogramniiug syMCiii \wrh rhrtv)vb<> ill mernor\. r major feature present in third-generation operating systems was the :ad jobs from cards onto the disk as soon as they were brought to tlie Then, whenever a running job finished, the operating syst p g j could load a new job from the disk into the now-empty partition and inn ii This technique is called spooling (from Simultaneous Peripheral Operation On t.ine) and was also used for output. With spooling, the T40)s were no longer needed, and much carrying of tapes disappeared. Although third-geneiation operating systems were wen suiic-d for big scic tific i pn.c still basically batch systems. Many programmers' pined for the first-generaturn days when they had the machine all to themselves for a few hours, so they could debug their programs quickly. With third-generation systems, the time between suhmit- ting a job and getting back the output was, often several hours, so a single mis- misplaced comma could cause a compilation to fail, and the programmer t<i waste half a day. This desire for

quick response time paved the way toi timesharing, a variant of multiprogramming, in which each u^ei has an online terminal In a timesharing system, if 20 users are logged in and 17 of them are thinking or talking 01 drinking coffee, the CPU can be allocated in turn to the three jobs that want service Since people debugging programs usually issue short commands (e.g. compile a five- page proceduret) rather than long ones (e.g., sort a mil lion-record file), the com- computer can provide fast, interactive service to a number of users and perhaps also work on big batch jobs in the background when the CPU is otherwise idle The first serious timesharing system. CTSS (Compatible Time Sharing System), w,ts developed at M.l.T. on a specially modified 7094 (Corbato et al . 1962). How- However, timesharing did not really become popular until the necessary protection hardware became widespread during the third generation. is "procedure." "subrouliiie." and " iterclungeabty ir

12 INTRODUCTION <-HAP I After tke success 01 thi- CT&S system. MIT. Bell Labh, am) Genei.il fclectnc (then a major computer m.mufactuier) decided to embark on the development of a •■a.mputcr utility." a machine that would suppoit hnmlieds (,f simultaneous need electric power. >on just stick a plug m Uie wall, and within reason, as much pown as you need will l,e there The designers or this system, known as Ml'L- TICS (>U'LTiplexed Information and Computing Ser\k-t). envisioned one hugt' machine fiioviding computing power tor everyone in ihe Boston area The idea fliat machines far more powerful than their GE-645 mainframe would he sold t(,r ,i thousand dollars b> thi- millions only 10 years later w<is pure science fiction. on a inahinc onlj slightly rnore powerful th.in an Intel 386-hasei1 PC, although it had much more I/O capacity This is not quite as cra^y its it sounds, since people- knew how to write small, efficient programs tu those days, a skill that has suhse- quenlly heen lost. There were many reason-, that MULTlCS did not t;ike over the world, not the least of which is that it was written in VIA. and the VIA compiler H'*is years tale and bare I v worked at alt when it finally arrived In addition. Ml L- ical engine in the nineteenth century To make a long story short. MULTlCS introduced many seminal ideas into the computer literature, but tunnng it into a serious [iroduet and a ma tor commeruat su^eess was a lot harder than anyone had r\pccreil. Bell Labs1 dropped out of the project, and General Elecinc quit the computer business .dlogeihei. However, M.I.T. persisted and eventually got MULTlCS working It was lihimaiety sold as a commercial product by the company that

honght Gt's computer business (Honeywell) and installed by aboul 80 major companies and universities world- worldwide While their numhert, were small. MLJt TICS users were fiercely toyat Gen- General Motors. Ford, and the (J S Nalionat Security Aecnc>. ti.r example, only sfiut down their MUl.TlCS iysien» '» the tate tWOs, 10 years ,ittci Ml'LTKS was released For the moment, the concept or a uompuici utility has hzzled out but it may well come back in the form of massive centralized internet servers to which reta- lively dumb user machines are attached, with mosl of the work happening on the big .servers. The motivation here is tikely to he that mosl peoptc do not wiint to administrate an increasingly complex and dnrcky computer system and would prefer to have thai work done by a team »f pujressionals woiking for the company running the server E-eommeree is already evolving in Ihrs direction, with various companies running e-mail^ on multiprocessor servers to which sinipte ctienl machines connect, very much in the spirit <)f the MULTlCS oV'simI1 Despite its lack of commercial success. MfLTlCS had a huge inJlni- ncc on subsequent operating systems It is1 desenhed in (Corbato el al.. £972. Corbato and Vyssolsky, 1965; Daley and Dennis,. 1968; Organick. 1972; and Saltzer, 1974). It

HISTORY OF OPERATING SYSTEMS 13 Weh site, vww. multn iam.o>g, with a great deal of informa- information about ihc system, ifs designed, and its users. Another major development during the third generation was the phenomenal growth of minicomputers, starting with the DEC PDP-1 in 1961. The PDP-i had only 4K of 18-bif words, but at S12(),(X)t> per machine (less than 5 percent of the price or J 7094). it sold like hotcakes. For certain kinds of nonnumencal work, it was ,ilm<iM as fjsi as (he 7094 and gave birth to a whole new industry. It was quickly followed by a series ot other PDPs (unlike IBM's family, all incompati- incompatible! culminating in the I'DP-| | One ol the computer scientist <*l Bell Labs who had worked on the MULTICS project. Ken Thompson, subsequently found a small pDP-7 minicomputer that no one was using and ^,ct out to write a stripped-down, one-user version of MULTICS. This work later developed into the UNIX operating system, which became popu- popular in the academic world, with government agencies, and with many companies. The history of UNIX has been told eisewheie (e.g., Salu.s, 1994). Part of that story will be given in Chap. 10. For now, suffice it to say. that because the source code was widely available, various organisations developed their own (incompati- (incompatible) versions, which

led to chads Two major versions developed. System V. from AT&T, and BSD, (Berkeley Software Distribution) Ironi the Universiiy of California at Berkeley. These had minor variants as well. To make ilpossihle to write programs that could run on any UNIX system. IEEE developed a standard for UNIX, called POSIX. that most versions of UNIX now support. POSIX defines a minimal system call interface that conformant UNIX system*, must support. In fact, some other operating systems now also support the POSIX interface. As an aside, it is worth meniioning that in 1987, the author released a small clone of UNIX, called M1NIX, for educational purposes. Functionally. MINIX is very similar to UNtX, including POSIX support A book describing Us internal operation and listing the source code in an appendix is also available (Tanenbaum and Woodhull, 1997) MINK is available for free (including all the source code) over the Internet at URL www.ci.vu.nl/~ast/mim\.hrml The desire for a free production (as opposed to educational) version ot MINlX led a Finnish student, Linus Top/aids, to write Linux This system was developed on MINIX and originally supported various MINIX features (eg . the MtNtX fite system) It has since been extended in many ways out still retains a laige amount of underlying slructure common to MINIX, and to UNIX (upon which toe former was based). Most of what will lie said about UNIX in this book thus applies to System V, BSD. MINIX, Linux, and other versions and clones of UNIX as well. 1.2.4 The Fourth Generation A980-Present) Persona! Computers With die development of LSI (Large Scale Integration) circuits, chips contain- containing thousands of transistors on a square centimeter ot silicon, the age of the per- personal computer dawned In terms of architecture, personal computers ^initially

 rii\l> ' tailed microcomputers i weie not all tb<<f dJ'tuent trom minicomputers of the PDP-11 class, but tn term;, ol price they eeitamly weic dittl-rent Whole the mini- minicomputer made ttpos'sibbforadepajdnent <n a company or iini\ersit> lo have it, own computei, the microprocessor cl.fp made H pos-ible for a single individual <o have bis m her own personal computer In (974, when Intel came out with the 8f)8(), the first genei;il-purpo--e 8-bit CPU, it wanted an operating system for the 8080. in part fo he able to test if. Intel ;,<.ked (."e of its consultants. Gary Kildall. t(i write one Kitdall and a friend first built a controller tortlie newly-released Sbug.irt Associates 8 inch floppy disk and hooked the floppy disk up fo the 8080, thus producing the first micr<ic<>tnputei wiib a dt.sk. Kildall (hen

wrote a disk-based operating system called CP/M (Con- (Control Program for Microcomputers) (or it Since Intel did not think that disk- b;tsed microcomputers, lidd much of a future, ^hen Kildall asked for the rights so CP/M. Intel granted fus request Kildall then formed J Loinp.iny. Digital Research, to further develop and sell CP/M In (977, Digital Research it wrote CP/M to m;ike if suitable tor running on (he many microcomputers using the K080, Zifoe Z80. an(l otfiet CPU cbips~ Many application programs were written to run on CP/M, allowing <t to complete!) dominate the world of microeompudng tor about 5 yens In trie early 1980s. IBM designed the IBM PC and looked around for software to run on if. People from IBM contacted Bill Gale* io license bis BASIC1 mtei- preter. They also asked him if he knew of an operating system fo run un the PC Gates suggested that |BM contact Digital Research, then the world"-, dominant operating systems company. Making ivbai was surely the worst business decision in recorded history, Kildall refused to tired with IBM, sending ,i subordinate instead. To make matters worse, bis lawyer even refused to sign IBM's nondis- nondisclosure agreement covenng (he no(-yei-anm>unced CC Consequently, IBM went back to Gates asking if be could provide them w tth an opcr.rtitig system When IBM came back, Ga(es realized tliat ;t loc.if eoniputei manufacturer, Seattle Computer Produce, bad a suUable operating system. DOS (Disk Operat- Operating System). He approached them and asked to buy it (allegedly for $50,000), which they readily ,tecep(ed. Ga(es then offered IBM a DOS/BASIC package, which IBM accepted. IBM wanted certain ino<l ill eat tons, --o Gates hired (he per- person who wrote DOS, Tim Paletson. as an employee of Gales fledgling company, Microsoft, to make (hem The revised system w,,s renamed MS-DOS (MicroSoft Disk Operating System) and quickly came io dominate ific IBM PC market \ key factor here was Gates' (in retrospect, extremely wisei decision to ,ell MS- DOS to compiler companies loi butidlnu' with then hardwaie. etunpared to Ki!(fa!l'!> atfeaip(to sell CP/M to end users- one at tt time (af least uiitiall>) By the fmie the IBM PC7AT carae out jn 1983 wr(h the Intel 80286 CPU, MS-DOS was fumly entrenched and CP/M v>as on its last leg-- MS DOS was Uter widely used on the 8(^86 and 80486 Although the initial version ot MS-IXJS w,is fairly pntnrtivc, suhsequenf versions included more advanced features, including

 many taken from UNIX (Microsoft was well aware ot UNIX, even selling a microcomputer version of

it called XENIX during ihc company's early years.) CP/M. MS-DOS, and otlicr opera! ing systems for early microcomputers were .til based on users typing in commands from ihe keyboard- That eveniuaHy Lhanged due (o research done by Doug Engelbart at Stanford Research Institute in the 1960s. Engelbart invented the GUI (Graphical User Interface), pronounced "gcntey." complete with windows, icons, menus, and mouse. These ideas were adopted hy researchers dt Xerox PARC and incorporated into machines they butlt. One day, Steve Johs, who co-invented the Apple computer in his garage, visiied PARC, saw a GUI, ;ind tnstandy realized its potential value, something Xerox management famously did not (Smith and Alexander, 1988). Jobs then embarked on building an Apple with a GUI This project led to the Lisa, which was too expensive and failed commercially. Jobs' second attempt, the Apple Macintosh, was a huge success, not only because it was much cheaper than the l.isd. hut also because it was user friendly, meaning that it was intended tor users who not only knew nothing about computers but furthermore had absolutely no intention whatsoever of learning. When Microsoft decided to build a success!>i to MS-DOS, it was strongly influenced by the success, ot Hie Macintosh It produced a GUI-based system called Windows, which originally ran on l(ip of MS-DOS U.e . It was more like ;t shell than a true Operating system) l;or aboi,t 10 years fn.m 1985 to t"J9\ Win- Windows was (us! a graplik at environment on top <>T MS-DOS However, starting in 1995 a freestanding version ot Window's, Windows 9<\ was released that incor- incorporated many operating system tentmes into it. using the uudeilyinj! MS-DOS sys- system only for booting and riiuriinu old MS-DOS programs lti 1098, a sfigh'fly modified version of [his system, called Windows 98 w.i, released Nevertheless, hotb Windows 05 and Windows 9$ stdl contain ,i large amount ot 16-bit Iniel .(ssetnbly language Another Microsolf oper.itint; svstcni is W'indovi.s NT i.NT stands for New Technology), whtcb is compatible with WukWv. 9S -tf .t ceit.nn level, bui a com- complete rewrite from Mraleb intetn.tlK If is ,t lull 12-fot system The lead designer lor Windows N'l was D.tvid CuUci. who w.ts Jlso one ol the designers oi the suit expected tliul the fir-,t veisuiit oIN I would kill off MS-DOS and all other ver- versions of Windows since it w;!S a v.istlv Mipcuor -^ysfcm, but it ti/^lcd Only with Windows NT 4.0 did it Imally lJIlIi on m .i biLr «a\. especially on coipor.Kc net- networks Version 5 of Wmdnws N I" was rcniifned Windows 2(XIO in eJily l'>s»y. It was intended to be [he sineessi>i tubnth Wiudi.v.-, 08 :ni(l Wmdows NT 4 (I. That Windows 98 called Windows Me (Millennium edition)

The other major contendei in trie pcisoiul computer woild is UNIX (and us vanon*. denvdttves) I'NIX is strongest oil woikstatums and other high-end com- computers, such as network servers It is especially populai on machines [lowered hy

16 tNTRODUCTlON rHAP ' liigh-pertonnance RISC' chips. On Peniiurn-based computers, Linux is becoming ,1 popular alternative to Windows for students and increasingly many corporate iwrs. (As an aside, throughout this bonk we will use ihe temi "Peniium" to mean thu Pentium t, tt, 1!!. <md 4) Although many UNIX users, especially experienced programmers, prefei a comnianJ-bdied m ten ace to a GUI, nearl> all UNIX systems support a windowing system called the- X Windows syMem produLed at M l.T This system handles the basic window management, allowing users io create, delete, move, and resize windows using a mouse Often a complete GUI. such as Motif, is available io mn on top of the X Windows system giving I'MX j look and reel something like ihc Macintosh or Microsoft Windows, for those i'NIX users who want such a thing. An interesting development thai began taking place during the mid-1980s is the giowih of networks of personal computers nmtung nelwork operaling sys- systems, and clislributed operating systems, i lanenbaum and Van Stceti. 2002) In ,i network operating system, the user-, arc aware of the existence ot multiple cm?i- putcis and can log in to remote machines and copy files irom one mjilune to another. Each machine runs its own local operating systeiii and has its own local user (or User.s). Network operating systems are not fundamentally different Irom single-processor operating sysiems They obviously need a nctwmk interface controllci login and lemote file access, but these addim.rj-. do not change the essential struc- structure of the operating system A distributed operating .system, in contrast, is oiic that appear* io its users as a traditional uniprocessor system, even though it 1-. actually composed of mult tele-teleprocessors. The users should not be aware ot where their piograms are being run or where iheir files are located; that should all be handled auiomaticall> anil effi- efficiently by the operating system True distributed operating systems require mote than lust adding a little cade to a uniprocessor operaiing system, because distributed and centralized systems differ in critical ways. Distributed systems lor example, often allow applications cessoi scheduling algorithms in order t(> optimise ihe amount ot parallelism Cormnunication delays within the netwuik often mean ihat these (ai)d other)

algorithms must run with incomplete, ouidated. or even nicoirett information operating system has complete information about ibe system state 1.2.5 Ontogeny Recapitulates Phytogeny Aftci Charles Darwin's book Tin- Oimw of the Spet-ies was published, the German 7oologT>,t Ernsi Haeckel siaied thar - Ontogcn> Recapitulates Phylo- geny." By this he meant that the development of an embryo (ontogeny) repeats

SEC. 1.2 HISTORY OFOPERATING SYSTEMS 17 (i.e recapitulates) the evolution of the species (phylogeny). In oiher words, after fertilization, a human egg goes through stages of being a. fish, a pig, and so on before turning into a human baby. Modem biologisrs regard this as a gross sim- simplification, but it still h.is a kernel of truth in it. Something analogous has happened in the computer industty. Each new species (mainframe, minicomputer, personal computer, embedded computer, smart card, etc) seems 10 go through the development thaf its ancestors did. The first mainframes were programmed entirely in assembly language. Even complex programs, like compilers and operating systems, were written in a.ssemhler. By the time minicomputers appeared on the scene. FORTRAN, COBOL, and other high-level languages were common on mainframes, but the new minicomputers were nevertheless programmed tn assembler (for lack of memory) When micro- microcomputers (early personal computers) were invented, they, too, vvere programmed in assembler, even though by then minicomputers were also programmed in high- level languages. Palmtop computers also started with assembly code hut quickly moved on to high-level languages (mi.stTy because the development work was done on bigger machines). The same is true for smart cards Now le u look per ng em Tl r m f m all had no protectio hard e nd n ppo f 1 p g m n° he n imple operating hi handl d n m 11 1 d d [oe m im Later they acqu red he h d e nd pe ns em pp h ndle m 1 pie pro- programs at o ce nd h I II me h n o p b I When m n omp rs fi ppe edh Ihdnpr nhd e and ran one m n 11> I ded p g m me n th gh n 1 ip m ng was well establ hed n he fr m rid b hen G du 11 th q ed pro- protection hdaredhb! n pog once The iirs-t microcomp we 1 p ble f nn na il n p n t e, but later acq ed he b 1 J mlp can Pin p J mil d it the Disk fr ppe el 1 g inf n e h m p m com- computers, and ond heleEe v, ddnohehd disks, but with he dven ffl h ROM h 11 n h b q 1 f When disks first appeared, primitive file systems sprung up. On the CDC 66(X), easily the most powerful mainframe in the World during much of the 1960s, the file sys- system consisted of users having the ability to create j file and then declare it to be

permanent, meaning it stayed on the disk even after the creating program exited To access such a file later, a program had to attach >t with a special command and give us password (supplied when the file was made permanent). In eifeet, theri; was a single directory shared by all users. 1(was up to the users to avoid file name conflicts. Early minicomputer iile systems had a single directory shared by ;i!l users and so did early microcomputer file system*, Virtual memory (the ability to run programs larger than the physical memory) had a similar development It first appeared in mainframes, minicomputers.

lH tIM'l RObt'(' IION t HAP 1 microcomputers and gradually worked IK vvj> down to smaller and smaller sys- systems. Networking had a similar history. In all cases, the software development was dictated by the Technology The firs! microcomputers, lor example, had sometliinji 'ike 4 KB of memory and no procecoon hardware. High-level languagc-s ami multiprogramming were simply too much for such a tiny system to handle. As the microcomputers evolved into modem personal computers, they acquired the necessary hardware and ihen the necessary software to handle more advanced featuies. It is likely ihat this development will continue tor years to come. Olher fields may also have this wheel of reincarnation, but in the computer industry it seems to spin faster 1.3 THE OPERATING SYSTEM ZOO All of this history and development has left iis with a wide variety <>f operat- operating systems, not all ot which are widely known in ihis section we will briefly touch upon seven ot them We will come b.ick to some ot these different kinds of systems latei in the book 1.3.1 Mainframe Operating Systems At the liigii und are ihe operating systems l<ir the miiintraincs. those loom- sized coinputeis still iouix.1 in major corporate daij centers These computers dis- distinguish themselves from personal computer in icinis oi Their i/O capacity \ mainframe with lfHO disks and thousands .it gigabytes ot il.u.i is not iimisual. a personal computer with these specifications vvonkl he odd indeed M.nntr.imcs are also making something (if a comeback j*. liigli-cnd Woh seuers. scrvi-rs for large-scale electronic eotiimcvee sites, and -crvers fm bnsiiiess-to-business tiati- Tlie operating sjstems loi mainframes ,nc he.nili oiieiucl tow.ud processing maiiyjobs.il once, mosi of which iieed iiixxligums ai.iount' <ii I/O Thc> tvplc,ilfy otter Ibiee kinds of seivi^es b.itcb. n.ins.Mion processing .inii tmicsbanng. ^> hatch systeni is one tbut processes i out me jobs w iili()iil jiiv >nttir.icli vc list i present Claims processing in an insurants L(iirip:m\ oi s.iies icpottiiit! u,t .i chuiii of stores is tvpiciilly ddiic in batch

mode 1i,inaction procVssinp %\stems fundfc large numheis ot small rcqtie>K. fov e\,n!!j>if check processmg a! <l i>,mk or aii- lint.- resen atidns. Eacfi unit of work is sm.ili. bill Ihe system miW handle bi)ii- dreds or thousands per sei.oiid Tiincliuniiii s>stems allow multiple vemme nscis lo runjohs on ihc computer at once, ^ncli a*1 ^-K'CIn>nu j l^ii? djiahcise 1 hese luna- lunations are closely ielated mainframe operating systems often perform all ol ibem A» example niiiinframe operating system is OS/WO. a descendant of OS/J60

SEC M THE OPERATiWi SYSTEM ZOO IV 1.3.2 Server Operating Systems One level down are the server operating systems They run on servers, which ,ire either very large personal computers, workstations or even mainframes. They serve multiple users, a< once over a network and allow the users to share hardware and software resource-,. Servers can provide print service, file service, or Web service Internet providers run many server niaehmes to support iheir customers and Web sites use servers to store the Web pages and handle the incoming requests. Typical server opening systems are UNIX and Windows 2000 Linux is alsd gaining ground for servers. 1.3.3 Multiprocessor Operating Systems An increasingly common way to get major-league computing power is to i_on- neet multiple CPUs into a single system Depending on precisely how they arc connected and what is shared, these systems aie called parallel computers, mitlti- computers. or multiprocessors. They need special (.peratnig systems, but often these are variations on the server operating systems, wiih special features for communication and connectivity 1.3.4 Personal Computer Operating Systems The next category is the personal computer operating system. Their job is to provide a good interface to a single user. They are widely used for word process- processing, spread.sheets, and Internet access. Common examples are Windows 98, Win- Windows 2000. the Macintosh operating system, and Linux. Personal computer operating systems are so widely known that probably little introduction is needed, in fact, many people are nut even aware thai othei kinds exist 1.3.5 Real-Time Operating Systems Another type of operating system is the real-time system. These systems aie characterized by having time as a key parameter Foi example, in industrtal proc- process control systems, real-time computers have to collect data about the production process and use it to control machines in the factory. Often there dre hard dead- deadlines that must be met. For example, if a car is moving down an assembly hue, certain actions must take place at certain instant., of time If a welding robot welds too early or too late,

the car will be ruined If trie action absolutely mmt occur at a certain moment (or wnhm a certain range), we li.ive a hard real-time system. Another kind of real-time system is a soft real-time system, in wtndi missing an occasional deadline is acceptable. Digital audio or multimedia systems fall m this category. VxWorks and QNX are well-known real-time operating systems.

20 1NTRODUC1 ION CHAP. I 1.3.6 Embedded Operating Systems Continuing on down to smaller and smaller svstcms. w tonic to patmt.p computers and embedded systems A palmtop computer or PDA (Personal Digi- Digital Assistant) is a small computer that tits in a ^hirt pocket and performs a sm.tll number of functions, ^ueli ;is Jn electronic address book and memo p.itl bmbed- thouglit of as computers nobilc t pbones These often h.Ke some characteristics ot real-time systems but also have size, meinorv. and powci re si fictions that m,ike tht'in special Examples of such opciJtiiic systems are PalmOS and Windows Ch (Consumer Electronic). 1.3.7 Smart Card Operating Systems The smallest operating systems run on smart tards, which are credit card- si/.ed devices containing a CPU chip. They have \ery severe processing powei and memory constraints. Some of them can hitndle only a single function, such as electronic payments, but others can handle multiple functions on the same smart card. Often these are proprietary systems. Some smart cards are Java oriented. What this means is that the ROM on the smart card holds an interprets tor the Java Virtual Machine (JVM) .lav.t applets (small programs) arc downloaded to the card an(l arc interpreted by the JVM interpreter. Some of these cards can handle multiple Java applets at the same time, leading to multiprogramming and the need to schedule item Resource management and protection also become an issue when two or more applets arc present at the same time These issues ntust be handled by the (usually extremely primitive) operating system present on the card. 1.4 COMPUTER HARDWARE REVIEW An operating system is intimately tied U> the hardware oi the coinputei it runs on. It extends the computer's instruction set and manages its resources To work, it must know a great deal about lhc hardwaie. at least, about how the hardware appears to the programmer Conceptually, a simple personal computer can be abstracied to ;i model resembling that of Fig |-5. The CPU. memory, and I/O devices are all connected by a system bus and communicate with one another Ovci it Modern personal computers haver a more complicated sCructure. involving multiple buses, which we will look at later For the time being, this model will be sufficient. In the follow- following

sections, we wilt briefly review these tomponenls and examine some of the hardware issues that are t>f concern to operating system designers.

COMPUTER HARDWARE REVIEW Monitor g [sa Figure t-5. Some of lhe components jj a simpiL' pcKonj] i-ompiiifj 1,4.1 Processors The bnm of the computer is the CPU It fetches instructions, from memory and executes them. The basic cycle of evciy CPU is, to fetch ihe first instruction from memory, decode it to determine its type and operands, execute it, and then tetch, decode, and execute subsequent instinct ions, in this way, pmgrains are ear- earned ou! Each CPU has, a specific set of instructions that it can execute Thus, a Pen- Pentium cannot execute SPARC programs and a SPARC cannot execute Pentium pro- programs. Because accessing memory to get an instruction or data word takes much longer than executing an instruction, all CPUs contain some registers inside to hold key variables and temporary results Thus the instruction set generally con- contains instructions to load a word from memory into a register, and store a word from a register into memory. Other instruclion.s combine two operands trom registers, memory, or boih into a result, such as adding twu words and storing the result in a register or in memory. In addition to the general registers used ti> hold variables and temporary results, most computers have several special registers that arc visible (o the pro- programmer. One of these is the program counter, which contains the memory address ot the next instruction (o be fetched. After (hat instruction has been fetched, the program counter is updated to point to its successor Another register is the stack pointer, which points to the top of the current stack in memory. The stack contains one frame for each procedure that ha.s been entered but not yet exited. A procedure's stack frame holds those input parame- parameters, local variables, and temporary variables that are not kept in registers. Yet another register is the PSW (Program Status Word). This register con- contains ihe condition code bits, which are set by comparison instructions, the CPU priority, the mode (user or kernel), and various other control bits. User programs

INTRODUCTION The PSW plays an important role in system calls and I/O. The operating system must be awaie of all the registers When time multi- multiplexing the CPU, the operating system will often stop the running piogram tu (re)stait another one Every time it stops a running program, the operating system must save all the registers so the)? can be restored when the. program

runs later To improve performance, CPU (tesigneis have long abandoned ihe .simple model of fetching, decoding, and executing one instruction at a time. Many modern CPUs have facilities for executing more than one instruction at the same time. For example, a CPU might have separate fetch, decode, and execute units, so that while it was executing instruction n, it cpuld also be decoding instruction ft + I and fetching instruction n + 2 Such an organization is called a pipeline and is illustrated in Fig |-6(a) for a pipeline with three stages Longer pipelines are common fn most pipeline designs, owe an (flsiruction has hcen fetched into the pipeline, it must he executed, even if the pieceding instruction was a condi- conditional branch that was taken. Pipelines cause compiler enters and operating sys- system writers great headaches because ihe> expose ihe uniiplcKifies of the underly- underlying machine to them Figure l-fi 1,11 A ihnrc sljjie pipeline (b> A. ■4ipei'it.<i]ar (TD Even more advanced ihan a pipeline design is a superscalar CPU. shown in Fig. l-6(b) In this design multiple execution units are present, tor example, one for integer arithmetic, one toi floating-point arithmetic, and out fur Boolean operations. Two or more instructions are fetched at once, decoded, and dumped into a holding buffer until (hey can be execmed As so<m as an execution unit is tree, it looks in the holding buffer to see if there is an instruction it con handle, and if so, it removes the instruction trom the buffer and executes it An implica- implication of this design is. that program instructions are often executed out of order For the most part, it is up to the hardware to make sure (he result produced is the same one a sequential implementation would have produced, but an anni.yin^ amount o(ihe complexity is foisted onto the operating system, as wo shall kee Most CPUs, except very .simple ones used in embedded systems, have (wo nn,des, kernel mode and user mi.de, as mentioned earlier. Usually a bit in the

sbC 14 COMPUTER HARDWARE RFV|EW 23 PSW controls the mode When running in kernel mode, the CPU can execute every instruction m its tnstruction set and use every feature of the hardware. The operating system runs in kernel mode, giving it access to the complete hardware In contrast, user programs run in user mode, which permits only a subset of ihe instructions, to be executed and a subset of the features to be accessed. Gen- Generally, all insiruciions involving I/O and memory protection are disallowed in user mode. Setting the PSW mode bit to kernel mode is also forbidden, or course. To obtain services from the operating system, a user program must make a system call, which

tr;ips into the kernel and invokes the operating system. The TRAP instruction swiiches from user mode to kernel mode and starts ihe operaiing system. When the work has been compieied. conirol is relumed to ihe user pro- program a(the insiruciion following ihe system call. We will explain the details of ihe sysiem call process later in this chapier. As a note on typography, we will use ihe lower case Helvenca foni 10 indicate sysiem calls in running text, like this read. li is worth noting ihai compuiers have iraps other ihan the instruction for exe- cuiing a sysiem call. Mosi of ihe oiher traps are caused by the hardware to warn of ten exceptional siiuaiion such as an anempi to divide hy 0 or a float mg~poin< underflow. In all cases the operaiing system gets conirol and musi decide what to do. Someiimes ihe program musi be lerminaied wcth an error. Oiher times, the error can be ignored (an underflowed number can he sei to 0). Penally, when ibe program has announced in advance thai it warns to handle ccnam kinds of condi- conditions, conirol can be passed back 10 the program to let it deal with ihe problem 1.4.2 Memory The second major component in any computer is the memory ideally, a memory should be extremely fast (faster tlian executing an oisiriicdon so ihe CPU is not held up by the memoiy). abundanily large, and dirt cheap. No current (ccb- nology satisfies all ot' ibese goals, so a different approach is uken. The memory The lop layer consists of the rcg<s!cis infernal 10 the CPU They are made ot ihe same maienal as the CPU and are ibus just as tasl <<s the CPU. Consequently, (here is no delay in accessing tliem The siotage capacity availahle ill them (s typ- typically 32 x 32-bits on a 12-tm CPU and 64 x 64-bits on a 64-bh CPU i_es>, than I KB in boih cases. Programs must manage the registers (i e . decide what to keep in ihem) Ihemselves, in software. Next comes the cache memory, which is mostly controlled by the hardware. Mam memory is divided up into cache lines, (ypically 64 byies. w<tn addresses 0 10 63 in cache line 0. addresses 64 to 127 in cache line |. and so on. The most heavily used cache lines are kept in a high-speed cache located inside or veiv close io the- CPU. When the program needs <0 read a memory word, ihe cache hardware checks to see if the line needed is in the cache. If it <s. ealled a cache

Marti rr Magne Magne emory cdisk ctape 64-512 MB 5-50 GB 20 100 GB e 1-7. A l> -i rouph Jf take requ tout 1 tin est is two ic pe ser clo< nail it c ;k ivei the cycles Q iche hit. the tcquest bus to the maul meiwin Cache Mis in Cache misses have to go (o memory, wi memory is limited in size due to its high cost Some machines have two or even ihrce levels ot cache, each one

slower and bigger than the one before it. Mam memory comes next This is the workhorse of the racnion system Main memory is often called RAM (Random Access Memory) Old timers sometimes call it core memory, hecause computers in the 1950s and l%0s used tiny magnetizable territe cores for main memory Currently, memories art; tens to hundreds of megabytes and growing rapidly All CPU requests that cannot he satisfied out of ilie cache go to main memory Next in the hierarchy is magnetic disk (hard disk). Disk storage is two older? of magnitude cheaper than RAM per hit and often iwo orders of magnitude largei as well. The only problem is that ihe time to randomly access data on it is close to three orders of magnitude slower This low speed is due to the taci that a disk is a mechanical device, as shown m Fig. I -8 A disk consists of one- or more metal platters that rotate at 541H. 7200. oi 10,800 ipm A mechanical arm pivots ovet [he platiers trom the corner, similar to (he pickup arm on an old 33 rpin phonograph for playing vinyl records !nform;t tion is written onto the disk in a series ot concentric circles At any aiveti arm position, each of the heads can read an aimulat region called a track. Together all ihe tracks for a given arm position form a cylinder Each track is divided into some number ot sectors, typically 512 bytes pel sector On modern disks, the outer cylinders contain more sectors than ihe innci ones. Moving the arm from one cylinder to the nexi one lakes aboui 1 msec Moving it to a random cylinder typically takes 5 msec io 10 msec, depending oi, the drive. Once the arm is on the correct irack. the drive must wan foi the needed sector to rotate under the head, an additional del,i> <>t 5 msec io 10 msec, depend- depending on the drive's rpm. Once the sector is undei the head, reading or writing occurs at a rate of 5 MB/sec on low-end disks io 160MB/sec on taster ones.

COMPUTER HARDWARE REVIEW Read/write head I' per surface) in of arm motion Tlie final layer in the memory hierarchy is magnetic tape. This medium is often used as a backup for desk storage and for holding very large data seis To access a tape, it must first be put into a tape reader, either by a person or a robot (automated tape handling is common at installations with huge databases). Then the iapc may have 10 be spooled forwarded to got to the requested block. All in all, this could lake minutes. The big plus of tape is that tt is exceedingly che;ip per bit and removable, which is jmportani for backup tapes that must be smred off-site in order to survive fires, floods euthquakts etc The memory hierarchy we have discussed is typical, but some installations do not have all the layers or have a few different ones (such as optical disk). Still,

in all ot them, as one goes down the hierarchy, the random access nine increases dramatically, the capacity increases equally dramatically, and the tost per bit drops enormously. Consequently, it is likely that memory hierarchies will be around for years to come. In addition to the kinds of memory d^cussed ahove. many computers have a small amount ot nonvolatile random aecess memory Unlike RAM, nonvolaiile memory does not lose its contents when the power is switched off. ROM (Read Only Memory) is programmed at the factory and cannot be changed afterward. It is fast and inexpensive. On some computers, tlie bootstrap loader used to siart the computer is contained in ROM. Also, some I/O cards coine with ROM for han- handling low-level device control. EEPROM (Electrically Erasable ROM) and flash RAM are also nonvola- nonvolatile, but in contrast to ROM can be erased and rewritten However, writing them takes orders of magmiude more time than writing RAM. so ihcy are used in the same way ROM is, only with the additional feature thai it is now possible to correct bugs in programs they hold by rewriting them in the field Yet another kind of memory is CMOS, which is volatile. Many computers use CMOS memory to hold the current lime and date. The CMOS memory and

 ihe cku-k cii-cuii ihai mcrcineuK the nine in it arc pcw^u'Cl b-. a small bancry. *" ihe lime is correctly updaled, even when ihc ,.mipul.-r i-- uHplu^cd. lhc CMOS installed bauery often lasis for several years Hr>we\ei. whon it begins to Ian. Ilk' compuier can appear to have Al/lieimer's disease. forger ring ihings ihai u lia\ known for years, like which hard disk 10 booi from. Let us now focus on main memory for a Inile while](is ohen desirahle to hofd niuftipfe programs- m memory at once. If one program is blocked waning W a disk read (o compfete, another program c;in use ihe CPU. giving a henei CP1 • unfizaiion. However, with two or more programs in main memory ai once, two prnbfems must be so/vecf' 1 How to proteci ihe programs from i.ne .mother and ihe kernel fmin ihem all. 2. How io handle refocation. Many solutions are possible Howevei. all ol them, invoke ^juippijiL- ihe CTl' with special hardware. The first probfem is obvious, but the second mie is a bti nioic snhifi- W'Ueii ,i program is compifed and finked, the conipilci ,uid finku no nm kiicrtv v.hi-ri' in physical memory it will be loaded when it is fveciitcd Kir tins reason ihc\ usu uHy assume it will stan at address H. and jus] put ihc |i,si iDstruetinii ilk-iv Sup pose th.it the first instruction Inches a HtiriHihin iKL-innr; a<ldtcss KUKKI Nov suppose ih,it

the enure program and dau jic huded sUrlma M jdcl.-css Sft.lHfi). When the first instruction exei-irtes. it will l,ul bfiau^' ll \wll rclcioiitf ri>e word at 10.000, liHc.td i.Mhu wi.id atM).<HXI F,, s(Hu- i!iK piolik-1,1 we rti-t-d In fiiht'i tdocatc the progru,ti at !(Md tune, hnduijr .ill nV aiklu-^s.- and i,todil> [i(g ihem The stniplfst si,lutn,ii l^. -.hown in f-'tg. 1 >)u) in iH;> Hum- *c si\- j Li,n, puter equipped wtth Iwirsjxxi.f! reijisiers, lhc hase regisier anJ H,i-Hniil rvgi-U'r (Please note ib.it in t!ii> h,t..k, numhers be»innnij; with (K air m hcxadcaiidil the C language conveimoii SiunlurK, iiiii(ibi-,>]u-y,(Hiitiy w.nil ,i k-iidui;! /,-it) ah in octal) When a program is win, the base lT.-istci is -.a h> fimi tn tlic st.in ill Us pto&ram it'Ktimd ihe l.nin regi.sk-i tcHs how. Kugc tltc ^mib^nd pi(>gr,r,n it-\i .n,d data are When an instruction u to be Ictchul, (h(hjulwarc checks [o \cf ,t llu program Counter is less ih.rn flic limit legistur .md ,t n is jdcK il tu thr \\t\p ic^is ter and sends tlie sum to intjnnn> ShinlsirK uln'it ihc pnmr.ni' w.ints id Kt.-Ii . data wore! (e.g. from address 10,000). tbc Wl^.uc .i.tiimuHCjIfv Akk th, tm, teiiis of the hase legislei (c y. . 50.000} to (ii.n dddicss arid st-nd- tlu- sum ih() (Kilt' to Ihe memory The base legisler makes it impossible li.r ,r pu^rain tn leierciicc any part i>r manory below if self Furthcnnoie the limit rcgi\tei nukes it unpos

Sbc i 4 COMPUTER HARDWARE RL-VfLW sible to reference any p;>rl ol"tnemor> abo*i' itst'i' Thus this scheme (he proiection and the relocajion problem dt I he cos' at two new re^ slighj increase in cycle liriK- (to perform (he limit check and addition). 27 > hc.lh y User-2 data User-) data User program Operating System The check <ind mapping result m, called a virtual address, n sical address. The device lhat perfom MMU (Memory Management Unit). It bul is logically between the CPU and the A more sophisticated MMU 15 iliu< MMU with two pairs ol base and limit re for the data. The program counter and al pair I and data references use pair 2. As multiple users share the same program w thing n<)t possihle with the first seheme registers are set as indicated hy ih gram 2 is running, they are set as ir Much more sophisticated MMUs e ing an address generated hy the pn address used hy the memory, called a phy- the check and mapping is called the s located 00 the CPU chip or ek>se ti, it. ■atcd m Fig.]-9<b). Here we have an isters, one lor the program text and nno ;>ther references to thu program text jsc Ldtisequenee. it is now possihle i<> have ii only (>rte cop> of K in niemoiy. soine- When program I is running, ihe roar iwi to the leh of Fig l-9(b) When pr<>- :cd by the arrows to the nght of the figure. We will study some ot ihem later in Ihis

28 iNtKOOU TtON CHAP f honk The thing to note here is tii.it m&iugme tile MML' must he ,m opeiatmg system itinction. since useis cannot he trusted to do K correctly Twn aspects ot the memory sysiem ii.Ke a maji.i effect rai performance Fiist.eacheshidethe^.it.vdy slo* speed ul menuir.v When:, i.ogu.m has bcii miming lur ,i w|iik-. (he cache is lull ol thai progrurn s cache hues, giving good performance However, when the operating system switches lrc,m one program to another, the cache returns lull <,i the I'iist pro"tam's cache lines The ones needed Liy the new, progiam must be loaded one at a time from phj.sudl memory This operation can be a major pel form a nee hit ,i it happens too often Second, when svwidung from one program to another, the MMU registers have to be changed hi Fig]-9(b). only four iegisters have to be- reset, which is not a prol'lem, but in real MMUs, many more legists h.ive to be reloaded either implicitly (,i- dynamically, as needed Either way, it takes time The m,,ral of die storj is that switching trom one program (o another, called a L-oniext swiieh. i<. an 1.4.3 I/O Devices devices also interact heavily with the operatm? system As we saw in Fi;; 1^, I/O devkus generally consist <if two parts, a controller diid the device itsell The controller K a chip or a set of chips i>n a plug-in hoard that phvsicarly controls the device, ft accepts commands trom the operating system, for example, to rc;ur dai;t froin the device, and carries them oui tn many cases, the actual control of the device is \ery complicate^ and detailed, so it is the J(,b of the controller to piehcnt a Mmplei interface @ the operating system For example, a disk contiolier might accept a command to read sector I 1.206 from disk 2 The controller then has <o convert this linear sector number to a cylinder, sector, and head. This u.nversiun may be complicaled by the tact that outer cylinders have more sectors than inner ones and that Some had sectois have heen remapped onto other ones Then the controller has td detenu me which Cylinder the disk arm is on atid giVL- it a sequence or pulse.s to move in o* out the requisite number of cylinders It has, to wait until the proper sectoi has rotated under the head anrf then start readtng and storing the bits a.s they c<,me olf the drive, removing the preamble and computing the checksum 1-mally, it has to assemhle the incoming bits into words and store them in memory To d<> all this work, controllers often contain small embedded computers that are programmed to do their work. The other piece is the actual device itself Devices have tairlv simple inter- interfaces, hoth hecause they c;tnnoi do much and io make them standaid The latter is needed so that any IDF] disk controller can handle ,,n> IDE disk, for example IDE stands for Integrated Drive Electronics and is ihe standard

type of disk on Penliums and some other computers Since the actual dcv'ice interface is hidden

SEC I 4 COMPUTER HARDWARE REVIEW 29 hehind the controller, all (hat the operJtmg system sees is the interface to the con- troHei which may he quite dilterent from the interface to the device Because each type of controller is different, different software is needed to cont.oi each one. The software that talks to a controller, giving it commands and accepting responses, is called a device driver Each controller manufacturer has to supply a driver tor each operating system it supports. Thus a scanner may come- witn drivers tor Windows 98. Windows 2001), and UMX. for example. I'o be umd, (he driver has to he put into the operating system so il can ron in kernel mode. Theoretically, drivers can run outside (he kernel, but tew current systems support this possibility because it requires the ability to allow a user- space driver i0 be ahU- to access the dev;ce in a controlled way, a feature rarely supported There art; three ways the driver can he put into the kernel. The first way is to relink the kernel with the new driver and then rehoot the system. Many UNIX systems work like this,. The second way is to make an entry in an operating system l'ile telling it that it needs the driver and then reboot the system. At boot tune, the operating system goes and finds the drivers it needs and loads them Windows works this way. The Ihird way is tor the operating syslent to be ahle to accept new drivers while running and install them on-Ihe-Hy wiihout tlte need lo rebool. This way used to he rare btir is becoming mttch more common now. Hot pluggable devices, such as USB and IBEh 1 W4 devices (discussed helow i always need dynamically loaded drivers. rivery controller hus ii sniall number of registers tlial are used to comniunicalc Wirh it For example, a minimal disk controller might have registers for specify-specifying die disk address, memory address, sector count, and direction (read or write) To activate the controller, the driver gets a command from the operating system, then translates it into the appropriate values to wrire inio (he device regisrers. On some computers, the device registers are mapped into the operating system's address space, so diey can be redd and written like ordinary memory words. On such computers, no special I/O instructions are needed and user pro- programs can be kept away from the hardware by not pulling these memory ad- dre.s.ses within their reach (e.g., by using hase and limit registers). On other com- computers, the device registers are put in a special I/O port space, with each register having a port address. ()n these

machines, special IN and OUT instructions are available in kernel mode to allow drivers lo read and write the registers The former scheme eliminates ihe need for special I/O instructions bul uses Up some of the address space. The latter uses no address space but requires special instruc- instructions Both systems are Widely u.sed. Input and output can be done in three different ways In the simplest method, a user program issues a system call, which the kernel then translates into a pro- procedure call to ihe appropriate driver. The driver then .starts the I/O and ■mis jn a iighi loop continuously polling the device to see if il Ls done (usually there is some bit thai indicates ihat the device is still busy). When ihe I/O has completed, ihe driver puts ihe daia where ihey are needed (it any), and reiums. The operating

30 INTRODUCTION CHAP I syslem llien relum* control Io Ihc caller. This method is called busy wailing and lias ihc disadvantage of lying up ihe CPU periling ihe device until rt K finished (he second rnerhod is for Ihc dnver ro start (he device and ask il ro give jn interrupt when i(is finished- Ar lh.it poinl Ihe driver relume, fhe operating s\s- lem ihen blocks rhe caller if need be and looks for (,mer work ro do When the toirtrotler deiecls ihe end of (he transfer, it generates an interrupt to signal com- completion hilcrnipls aie very importanl in operalmp systems, so let us examine the idea closely In Fig. !- !O(aj we see a ihrec-stcp process for I/O In step I. I hie ll ll h b d[i Th dnver tel troller tbe nu ciiip u accept ashcrK troller which y g g tarts rbe deuce. When the compiler hat, linished reading or writing mber or bytes it has been Sold to transfer, it signals the interrupt controller sing certain bus lines in step 2 If ihe interrupt controller is prepared to the interrupt (which it may not be if it is busy with a higher priority one), ,i a pin on the CPU chip informing it m step 3 hi step 4, the interrupt eo i- puts the number of the device on the bus so the CPU can rcad it and know dev^e has just finished (many devices may be ninnine at tlie same hme). 1 / Nexl ms 11 interrupt V 2. Dispatch ~— Once the CPU has decided to take the interrupt, the program counler and PSW are typically then pushed onto the curt em stack and the CPU switched mti, kernel mode. The device number may lie used as an index into part of memory t» find the address of the interrupt handler for this device This pan oi' memory)■> called the interrupt vector. Once the [ntenupt handler (part of the driver for the interrupting device) has started, it removes the stacked program counter and PSW finished, rt returns to the previously-running user program to the first instruction that was not yet executed. These steps are shown rn Fig l-10(b).

SEC t 4 COMPUTER HARDWARE REVIhW 31 The third method for doing I/O makes use of a special DMA (Direct Memory Access) ch,P that can control (he flow of biis between memory and some controller without constant CPU intervention. The CPU sets up the DMA chip, felling it how. many bytes 10 transfer, the device and memory addresses 'involved, and the direction, and lets it go When [he DMA chrp is done, it causes an mierrupt. which is handled as described above. DMA and I/O hardware in general \vrll be discussed in more detail in Chap. 5 Interrupt can often happen at highly inconvenient moments, for example, while another imerrupt handler is running For (his reason, (he CPU has a way to disable interrupts and then reenable them later. While interrupts are disabled, any devices that finish continue to assert their inttitrupi srgnals, but ihe CPU is not interrop(ed until interrupts are enabled again If multiple devices finish while in(errupts are d[.sabled. the interrupt controller decides which one io le! through first, usually based on static priorities assigned (o eiich device The highest prior- priority device wins 1.4.4 Buses The organi?ation ot Fig. 1 -5 was used ori minicomputers tor years and also on the original IBM PC. However, as processors and memories got faster, iht ability ot a. single bus (and certainly the IBM PC busi to handle all the traffic was strained to the breaking point. Something h;td U. give. As a result, additional buses were added, both for faster I/O devices and for CPU to memon- traffic As a consequence of this evolution, a large Pentium system currentK looks some- something itkeFig. 1-tl. This system has eight buses (cache, local, memory. PCI, SCSI, USB. IDE. and ISA), each with a different transfer rate and function. The operating system must be aware of all of them for configuration and management. The two main buses are the onginal IBM PC ISA (Industry Standard Architecture) bus and its successor, the PCI (Peripheral Component Interconnect) bus. The ISA bus. which was originally the IBM PC/AT bus, runs at 8.33 MHz and can transfer 2 bytes ittonve, for a maximum speed of 16.67 MB/see. |t fs fnclitded for backward compatibility with old and slow I/O cards. The PCI bus was invented by Intel as a successor to the ISA bus. It can run at 66 MHi and tian<-fei & bytes at a irme, for a data rate of 528 MB/sec Most high-speed I/O devices use the PCI bus now. Even some non-Intel computers use the PCI bus due to the large number of I/O cards available for tt. In this configuration, the CPU talks to the PCI bridge chip over the local bus. and the PCI bridge chip talks to the memory o\er it dedicated memory bus. often running at 100 MHz. Pentium system, have a level-1 cache on chip Jnd a much larger level-2 cache of(chip, connected to the CPU b> the cache bus In

addition, this system contains three specialized buses, IDE, USB. and SCSI. The IDE bus is for attaching peripheral devices sueh ;is disks and CD-

INTRODUCTION Ftffire 1-11. The si ROMs to the system. The IDE bus ,s an outgrowth of the disk controller mUafan: on the PC/AT and k now standard on nearly all Pentium-based systems for the hard disk and often the CD-ROM. The USB (Universal Serial Bus) was invented to attach all the slow I/O devices, such as [he keyboard and mouse, to the computei It uses a small foui- wire connector, (wo of winch supply electrical power to ihe USB devices USB is d centralized bus in which a root device polls ihe I/O devices every 1 n.sec to see if they have any traffic It can handle an aggregate load ot 1 5 MBZs.ec All the USB devices share a single USB device drivei, making it unnecessary to install a new driver for each new USB device Consequently, USB devices can be added to the computer without the need to reboot The SCSI (Small Computer Svstem Interface) bus rs a high-pcihwnianee bus intended for fast disks, scanners, and other devices needing cnnsiderahlc bandwidth. It can run at up to 160 MBZsec It hah been present on Macintosh sys- systems since they were invented and is also popular on UNIX and some Intel-based systems Yet another bus (not shown in Fig !-M) ,s IEEE 1394. Some-times it i- called HireWirc, although sfnetly Speaking, FireWiie is the name Apple uses foi its implementation of 1394. Like USB, IEEE 1394 is hit serial but is designed tot

SfcC 1.4 COMPUTER HARDWARE REVIEW 33 packet transfers at speeds up to 50 MB/scc, making it useful for connecting digital camcorders and similar multimedia devices to a computer. Unlike USB, lEEb 1194 does not have a central controiier SCSI ami IEEE 1394 face competition from a faster version of USB being developed To work in an environment such as that of Fig 1-1 I. the operating system has 10 know what is out there and configure it. This requirement led Intel and Micro- hoU to design a PC system called plug and play, based on a similar concep< first implemented in the Apple Macintosh. Before plug and play, each I/O card had a fixed interrupt requesi level and fixed addres.ses for its I/O registers. For example, the keyboard was interrupt 1 and used I/O addresses 0x60 to 0x64. the floppy disk controller was interrupt 6 and used I/O addresses 0x3F0 to 0x31-7. and the printer was interrupt 7 and used I/O addresses 0x378 to 0*37A, and so on. So far, so good- The trouble came when the user bought a sound card and a modem card and Roth happened to Use, say, interrupt 4. They would conflict and would not

work together. The solution was to include DIP switches or jumpers on every I/O card and instruct the user to please set them to select an interrupt levei and I/O device addresses that did not conflict with any others in the usef's system. Teenagers who devoted their lives to the intricacies of the PC hardware could sometimes do this without making errors. Unfortunately, nobody else could- lead- leading to chaos What plug and play does is have lhe system automatically collect information about the I/O devices, centrally assign interrupt levels and I/O addresses, and then tell each card what its numbers are. Very briefly, that works j;> follows on ihe Pentium. Every Pentium contains a parentboard (formerly called a modierboard before political correctness hit ihe computer industry). On the parentboard is a program called the system BIOS (Basic Input Output System, The BIOS con- contains low-level I/O software, including procedures to read the ke>board. write to the screen, and do disk I/O, among other things. Nowadays, it is held in a flash RAM, which is nonvolatile but which can be updated by the operating sysiem when bugs are found in ihe BIOS When the computer is booted, the BIOS is started. It first checks to see huw much RAM is installed and whether the keyboard and other basic devices are instaiied and responding correctly. It starts out by scanning ibe ISA <ind PCI buses to detect all the devices attached to them Some of ihese devices are typi- typically legacy (i.e., designed before plug and play was invented) and have fixed interrupt levels and I/O addresses (possibly set hy switches or jumpcis <>n the I/O card, but not modifiable by the operating system) These devices are recorded. The plug and play devices are also recorded If ihe devices preseiil arc differenl from when the system was last booted, |hu new devices are configured The BIOS then determines the boo! device by trying a iist of devices stored in tlie CMOS memory. The user can change ihis list by eniermg a BIOS configura- configuration program just after booting. Typically, an ai tempt is made to b<,<,t from ihe floppy disk. If that fails the CD-ROM is tried. If neither a floppy nor a CD-ROM

;i par 34 iNrRODllTlON ('HAt* ' is preseni. the system is booted Horn the iiaid disk The lust sector fiom the booi device is read into memory and executed. Tlu.s sector contains a program that normally examines ihe partition table ai ihe- end of the boot sector to deiermine itKin is active Then a secondary hooi loader is icad in fioin that parti- loader reads m ihe operating system from the active partition and sl.tfts il The opciating system then queries, ihL- BIOS to get the t tmiiguraticHi mforina- tioii For each device, n checks tu

see H 11 has tlie device driver. If noi. it asks, the usc-i to msert a floppy disk or CD-ROM coniaming the dnver (supplied by tfie device's manufacture^ Once ii has all the device drivers, ihe operating sysiem |o;ids them into the kernel. Then it initializes its iable\. creates whaievei baek- aiDinid processes are needed, and starts up a login program or GUI on each leniu- iial. Ai least, ihis is the way it is supposed (O v^ork In real lire, pluy and j>Ia> is frequently so unreliable thai many people call it plug ,md pray. 1.5 OPERATING SYSTEM CONCEPTS All operating systems have certain l.asic i.(incepts such as processes, memory, and files th;it are central to understanding them. In the following seein>ns. we will look at some of these basic concept;, ever so briefly, as an introduction We will come back to each of them in great dclait Liter in this book, 'lo illustraie these concept*, we will use examples from time to nine, generally drawn from UNIX 1.5.1 Processes A key concept in all operating sysiems is the- process. A process is basically a piogr.im in execution Associated with each process is its address space, a fisi of memory locations from some minimuiii (usually (n to some maximum- whiLh the process can read and write The address space contains the executable program, the program's daia. anci its stack. Also associated wuh each piwess is some set ot registers, including the program counter, stuck pointer, and other hardware registers, and all the other information needed to run the program We will come hack to the process concept in much more deiail in Chap 2. but f(»r the time being, the easiest way to get a good intuitive feel for a process is 10 think about timesharing systems Periodically, the operating system deudes, lo stop running one process and start running another, for example, because tlie fust one fun hid more truw its share of CPU time in the pasi seeond. When a process is suspended temporarily like this, it mils' later be restarted in exactly the same state it riad when it was stopped This means tliat all information about the process must ix- explicitly saved somewhere during [he suspension For example, the piocess may have several files open for reading at once Associated

SfcO. OPERATING SYSIfcM CONCEPTS 35 with each ol these files is a pointei giving tire current position A c . the number cif Ihc byte or record to be iead next) When a process is temporarily suspended, all these pointers must be s<ived so thai a read cull executed after (hi; process is re- restarted will read the proper data. In many operating systems, all the information about each proeess, other ihan the contents of its own address space, is stored in an

operating system table called the process table, which is an array (or linked lis'1) of structures, one t<ir each process currently m existence Thus,, a ^.suspended) process consists of its address space, usually called the core image (in honor uf ihe magnetic core memories used in days of yore), <ind its puieess1 table entry, which contains its regisieis, among other things The key process management system calls are those dealing with the oreancin and lermniatKin ot processes. Consider a typical example A process oiled ihi: command interpreter or shell read,, commands from a terminal. The user has just typed a command requesting that a program be compiled The shell must now create a new process that will run ihe compiler When that pioeess has fin- finished the compilation, it executes a s\sttm L ill to terminate itself. 1| a process can create one or more other processes (referred to as child processes) and these processes in tun) can create cliild processes, we t(inckly arrive at the process tiee structure of Fig 1-12. Related processes that art? cooperating to get some job d<,ne often need to communicate with one another and synchrdnize their activities. This communication is called interprocess com- communication, and will be addressed in detail in Chap. 1 ■e 1-12. A proixs : child priiu Other process system calls are availahle unused memory), wait tor a child process tc .... with a different one. Occasionally, there is a need to convey inf(,n is not sitting around waiting for this information ■ating with another process on a differe reqm mate, and ovei iory (or release lay its program ion to a running process that iii example, a process thai is ,'omputer does so hy sending messages to the remote process over a computer network To guard againsi the possihility that a message or its reply is lo.st, the sender may request ihat its own operating system notify it after a specified number of seconds, so that u cm retransmit the message if no acknowledgement has heen received yet After set- ling this timer, the program may continue doing other work.

36 INTHODl CT1ON CllA>- I When |hc specified number cil seconds has elapsed, the oper-Jhug sysleni sends an alarm signal to the process The signal causes the pi'i^ess to tem- temporarily suspend whalever it was doing, save its registers un the stack. ,uid slart lUnnintl a special signal handling procedure, fen example, lu retransinil a presum- presumably lost message. When the signal handlei is done, the running]irocess is re- restarted in ihe state it was in just before |he signal Signals die die software .rn.ilog «t haidwiare interrupts and can be generated b> a variety of causes m addition tt-> Miners expiring.

Many traps detected by li.irdwnrc, .such us executing an illegal instruction 01 using an invalid address, are also converted in|o sign.ils to the guilty process Bach person auihonzed 10 use a system is assigned ;i UID (User IDenuILa- tioni hy the system adminisuator Every process started h;is the UID o\ ihe per- person who started it. A child process has the same HID as !|s parent Users can be members of groups, eacli of which has a G1D [Group IDentiflcationi One UID. called the suptruser (in UNIX), has special power and ma\ vidlalc- many ot the protection tules. In large inscalUtiims. only the system admirnstraior knows the password needed to become superuser. but muny ot the ordinary Users (especially students) devote considerable effort n, trying to find naws m ihe sys- system that allow them to become superuser without the password We will study processes, interprocess ciimmiinicanon, and related issues m Chap. 2. 1.5.2 Deadlocks When twt, or more processes are interacting, they can swnetivnes get Them- Themselves into a stalemate situation they cannot get out ot Such a situation is c.dkd a deadlock. Deadlocks can be^t lie introduced with ,i real-world example eseryone is t.im- iliar with, deadlock in traffic Consider the situation of Fiji l-13ia) Here- four huses are approaching an intersection Behind each one ;ire more huses (not shown) With a little bit of bad luck, the first tour could all arrive at the intersec- intersection simultaneously, leading lo the situation ot Fig- l-13(t», in which they are deadlocked becau.se none ot them can go forward Each one i.s backing one <,t the others. They cannot go backward due t(> other buses behind them. Theie is isi_i easy way out. Processes m a computer can experience an analogous situation in which they cannot make any progress For example, im.ijj.ine a compmer with a tape dri\e and CD-recorder. Now imagine that two processes each need to produce- u CD- ROM from data on a tape. Process I request and is granted the rape drive Nexi proce.s.= 2 requests and is granted the CD-recordar Then process f requests the CD-recorder and is suspended until process 2 leturns it. finally, process 2 re- the i;ipe drive and is also suspended because process ! already has it. Here quest

SFC is OPERATING SYSTEM CONCEPTS Figure 1-13. (a) A poieniial deadlock <b> An jciual deadlock we have a deadlock from which ibere is no escape. We will siudy deadlocks and wha! can he done about Ihem in delail in Cbap. 3. 1.5.3 Memory Management Every computer has some main memory lhat it uses to bold executing pro- programs. In a very simple operating system, only one program at a time is in memory. To run a second program, Ibe first

one bas to be removed and ihe second one placed in memory. More sophisticated operating systems allow mulliple programs' !o be in memory al the same time. To keep Ihem from interfering with one anolher fnnd wilh (he operating system), some kind of protection mechanism U needed. While Ibis mechanism has to be in Ihe hardware, n us controlled by Ihe (iperaling system. Tbe above viewpoin! is concerned wilh managing and protecting Ibe computer's main memory. A different, but equally importani memory-related issue, js managing this address space of the processes. Normally, each process has some sel of addresses it can use, typically running irtiin 0 up (o some maximum. In the simples! case, the maximum amoun! of address space a process has is less man Ihe main memory. In Ibis way, a process can fill up its address space and there will be enough room in main memory lo hold it all However, on many computers addresses are 32 or 64 bils. giving an address space of 212 or 2M byles, lespectively Wbat happens li a process has mare address space than die computer has main memory and the process wants ta use it all? In tbe first computers, such a process was just out of luck Nowadays, a technique called virtual memory exists, in wbich the operating system keeps par! of |he address space in main memory and part on disi. and shuttles pieces back and forth between them as needed. This important operating .syslem function, and otber memory management-re I a ted functions will be covered in Chap. 4.

38 INTRODUCTION CHAR I 1.5.4 Input/Output All computers have physical devices for acquiring input and producing output. After all, what good would a computer be if the users could not tell it what to do and could not get the results after If did the work requested. Many kinds of input and output deviees exist, including keyboards, monitors, printers, and so on. It is up to the operating system to manage these devices. Consequently, every operating system has an I/O subsystem for managing Us I/O devices. Some of the I/O software is device independent, that is, applies to many or all I/O devices equally well. Other parts of it, such as deviee drivers, are specific to particular I/O devices. In Chap. 5 we will have a look at I/O software. 1.5.5 Files Another key concept supported by virtually all operating systems is (he file system. As noted before, a major function of the operating system is, to hide the peculiarities of the disks and other I/O devices and present the programmer with a niee, clean abstract model of device-independent files. System calls are obviously needed to create files, remove files, read files, and

write files. Before a tile can be read, it must be located on the disk and opened, and after it has been read it should be closed, so calls are provided to do these things. To provide a place to keep files, most operating systems have thi: concept ot a directory as a way of grouping files together A sludent, for example, might have one directory for each course he is taking (for the program* needed for that course), anotfier directory for his electronic mail, and still another directory tor his World Wide Web home page. System calls are then needed to create and remove directories. Calls are also provided to put an existing file in a diruclory. and to remove a file from a directory Directory entries may be either files or cither directories This model also gives rise to a hierarchy—the file system—as shown in Fig. 1-14. The process and file hierarchies hoth are organised as trees, but the similarity slops (here. Process hierarchies usually are not very deep (more ihan three levels i unusual), whereas file hierarchies are commonly four, five, < deep. Process hierarchies are typically shori-lived. gen most, whereas the directory hierarchy may exist tor year_s. Ownership and protec- protection also differ for processes and files. Typically, only a parent process may con- control or even access a child process, but mechanisms nearly always exist hi allow files and directories to be read by a wider group than just the own< Every file within the directory hierarchy can be specified b; name from the top of the directory hierarchy, the root directory Such ahsolute path names consist of the list of directories that must be traversed Irom die root directory to get to the file, with slashes separating the components, in Fig 1-14, n more levels / minutes

SEC ,.5 OPERATING SYSTEM CONCEPTS Root directory Figure m4. A file sysit sicy departm the path for file CSWJ is /Faculty/Prof.Brown/Courses/CSlOl. The leading slash indicates that the path is absolute, that is, starting at the root directory. As an aside, in MS-DOS and Windows, the backslash {\) character is used as the separa- separator instead of the slash (/) character, so the file path given above would be written as \Faculty\Prof.Brown\Courses\CS101. Throughoul this book we will generally use the UNIX convention for paths. At every instant, each process has a current working directory, in which path names not beginning with a slash are looked for. As an example, in Fig. 1-14, if /Faculty/Prof.Brown were the working directory, then use of the path name Courses/CSWl would yield the same file as the absolute path name given above. Processes can change their working directory by issuing a syslem call specifying the new working directory.

Before a file can be read or written, it musl he opened, at which time the per- permissions are checked. If the access is permitted, the system returns a small integer called a file descriptor to use in subsequent operations. If Ihe access is prohibited, an error code is relumed. Another important concept in UNIX is Ihe mounted file system. Nearly all personal computers have one or more floppy disk drives into which floppy disks can be inserted and removed, To provide an elegant way to deal with removable

40 INTRODUCTION ^HAI media (including CD-ROMs 1. UNIX allows the lite system on a floppy disk t< attached to the main tree Consider the situation c,t Fig l-15<a). Before mount call, the root file system, on the hard disk, and a second tile system, i floppy dbk. are separate ,ind unrelated. Figure 1-15. (a) Bbiore niounimg, Che nies on drive 0 jre not dLces^lhlc ibi Alter mount ing, they jre p^rl ol Cht fileiiierjnh> However, the file system on Ihe floppy cannoi be used, hceause there is no way to specify path names on it. UNIX does not allow path names to be prefixed by a drive name or number; that would be precisely the kind (if device dependence tha! operating systems ought to eliminate. Instead, the mourn system call allows the file system on the floppy to be attached to the root file sysiein wherever the program wants it to be. In Fig. 1-I5(b) the tile system on the floppy has been mounted on directory b, thus allowing access to files fbfx and /b/v. It directory b had contained any files they would not be accessible while the floppy was mounted, since fb would refer to the rool directory of Ihe floppy. (No! being able to access these files i.s not as serious as it a! first seems: tile systems are nearly always mourned on empty directories.) If a system contains multiple hard disks, they can all be mounted into a single tree as well. Another important conc-ept in UNIX is the special file. Special files are pro- provided in order io make I/O devices look like tiles. That way, tfiey can be read and wntten using the same system calls as are used for reading and writing files. Two kinds of special files exist: block special files and character special files. Block special files are used !o model devices that consist of a collection of randomly addressable blocks, such as disks. By opening a block special file and reading, say, block 4, a program can directly access the fourth block on the device, without regard to the structure of the file syslem contained on it. Similarly, character spe- special files are used to model printers, modems, and other devices that accept or out- output a character stream. By convention, the special files lire kept in the /dev direc- directory. For example, /devflp might be the line printer. The last feature we

will discuss in this overview is one that relates to both processes and files: pipes. A pipe is a sort of pseodofile that can be used to con- connect two processes, as shown in Fig. 1-16. If processes A and B wish to talk using

SEC 1.5 OPERATING SYSTEM CONCEPTS 41 a pipe, they must sot it up in advance. When process, A wants to send dala lo process. 6. i! writes on Ihe pipe as though i! were an output file. Process B can read the data by reading from the pipe as though i! were an inpu! file. Thus, com- munication belween processes in UNIX looks very much like ordinary file reads and wriles. Stronger yet, the only way a process can discover that the output file il is writing on is no! really a file, bul a pipe, is by making a special system call. File syslems are very important. We will have much more to say about them in Chap-6 and also in Chaps. lOandll. 1.5.6 Security Computers contain large amounts of information that users often want to keep confidential. This information may include electronic mail, business plans, tax returns, and much more. It is up to the operating system to manage the system security so that files, for example, are only accessible to authorized users. As a simple example, just to get an idea of how security can work, consider UNIX. Files in UNIX are protected by assigningeach one a 9-bit binary protection code. The protection code consists of three 3-bit fields, one for the owner, one for other members of (he owner's group (users are divided into groups by the system administrator), and one for everyone else. Each field has a bit for read access, a bit for write access, and a bit for execute access. These 3 bits are known as the rwx bits. For example, the protection code rwxr-x-x means that the owner can read, write, or execute ihe file, other group members can read or execute (but not write) the file, and everyone else can execute (but not read or write) ihe file. For a directory, jc indicates search permission. A dash means that Ihe corresponding permission is absent. In addition to file protection, there are many other security issues. Protecting the system from unwanted intruders, both human and nonhuman (e.g., viruses) is one of them. We will look at various security issues in Chap. 9. 1.5.7 The Shell The operating system is the code that carries out the system calls. Editors, compilers, assemblers, linkers, and command interpreter!, definitely are noi part of ihe operating system, even though they are important and useful. At the risk of confusing things somewhat, in this section we will look briefly at the UNIX com-

42 INTRODUCTION CHAP. I mand interpreter, called the shell. Although it is

not part of the operating system, it makes heavy use of many operating system features and thus serves as a good example of how the system calls can be used. It is also the primary interface between a user sitting at his terminal and Ihe operating system, unless the user is using a graphical user interface- Many shells exist, including ift, csh, ksh, and bash. All of them support trie functionality descnbed below, which derives from Ihe original shell (sh). When any user logs in, a shell is started up. The shell has the terminal as standard input and standard output It starts out by typing (he prompt, a character such as a dollar sign, which tells Ihe user lhat the shell is waiting to accept a com- command. If ihe user now types for example, the shell creates a child process and runs Ihe date program as the child. While the child process is running, the shell waits for it to terminate. When the child finishes, the shell types the prompt again and tries to read the next input line. The user can specify that standard output be redirected to a file, for example, date >fi1e Similarly, standard input can be redirected, as in sort <file1 >file2 which invokes the sort program with input taken from file I and output sent to file2. The output of one program can be used as Ihe input for another program hy connecting them with a pipe. Thus cat filei file2 fits3 t son >/dev/lp invokes the cat program to concarenate three files and t,end the output to sort to arrange all the lines in alphabetical order. The output of sort u redirected to the file /dev/Ip, typically the printer. If a user puts an ampersand after a command, the shell does not wait for it to complete. Instead it just gives a prompt immediately Consequently, cat (ile1 file2 file3 I son >/dev/lp & starts up the sort as a background job, allowing the user to continue working nor- normally while ihe sort is going on The shell has a number of other interesting features, which we do not have space to discuss here Most books on UNIX dis- discuss ihe shell at some length (e.g., Kemighan and Pike, 1984; Koehan and Wood. 1990; Medinets, 1999. Newham and Rosenblatt, 1998: and Robbins, 1999).

SEC. 1.5 OPERATING SYSTEM CONCEPTS 4.3 1.5.8 Recycling of Concepts Computer science, like many fields, is largely technology driven. The reason (he ancient Romans lacked cars is not that (hey liked walking so much. I< is because they did not know bow io build cars. Personal computers exist not because millions of people had some long pent-up desire io own a computer, but because it is now possible to manufacture them cheaply. We often forgei how much technology affects our view of systems and it is worth reflecting on this point from time 'o time. In particular, it frequently happens

that a change in technology renders some idea obsolete and ii quickly vanishes. However, anoiher change in technology could revive it again. This is especially (rue when (he change has to do with the relative performance of different parts of we system. For example, when CPUs became much faster wan memories, caches became important to speed up we "slow" memory. If new memory technology some day makes memories much faster than CPUs, caches will vanish. And if a new CPU fechnology makes (hem faster than memories again, caches will reappear. In biology, extinction is for- forever, but in computer science, it is sometimes only for a few years. As a consequence of this impermanence, in this book we will from time to time look at "obsolete" concepts, that is, ideas thai are not optimal with current technology. However, changes in the technology may bring back some of the so- called "obsolete concepts." For (his reason, it is important to understand why a concept is obsolete and what changes in the environment mighl bring it back again. To make this point clearer, let us consider a few examples. Early computers had hardwired instruction sets. The instructions were executed directly by hardware and could not be changed. Then came microprogramming, in which art underlying interpreter carried out the instructions in software. Hardwired execu- execution became obsolete. Then RISC computers were invented, and microprogram- microprogramming (i.e., interpreted execution) became obsolete because direct execution was faster. Now we are seeing (he resurgence of interpretation in me form of Java applets lhat are sent over the Internet and interpreted upon arrival. Execution speed is not always crucial because network delays are so great 'hat Ihey tend to dominate- But that could change, too, some day Early operating systems allocated files on (he disk by just placing (hem in contiguous sectors, one after another. Although this scheme was easy to imple- implement, it was not flexible because when a file grew, (here was not enough room to store it any more. Thus the concept of contiguously allocated files was discarded as obsolete. Until CD-ROMs came around. There (he problem of growing files did not exist. All of a sudden, the simplicity of contiguous file allocation was seen as a great idea and CD-ROM file systems are now based on it. As our final idea, consider dynamic linking. The MULTICS system was designed to run day and night without ever stopping. To fix bugs in software, it

 was necessary to have a way to replace library procedures while they were being used. The concept ot dynamic linking was

invented ftir this purpose. After Ml'L- T1CS died, the concept was forgotten for a while. However, it was rediscoveied when modern operating systems needed a way to allow many programs 10 share the saine library procedures without having their own private copies (because graphics libraries had grown so large) Most systems now support some form of dynamic linking once again The list goes on. but these examples should make the point; an idea that is obsolete today may he the star of the party tomorrow. Technology is not the only factor thjt drives systems and software Econom- Economics plays a big role too. In the 1960s and 1970s, most terminals were mechanical printing terminals or 25 x 80 character-oriented CRTs rather than bitmap graphics terminals. This choice was not a question of technology. Bit-map graphics termi- terminals were in use helore I960 It is just that they cost many tens of thousands ol dollars each. Only when the price came down enormousl> could people (other than the military) think of dedicating one terminal to an indiv idual user 1.6 SYSTEM CALLS The interface between the operating system and the user programs is defined by the set of .system calls that the operating system provides. To really understand what operating systems do. we must examine this interface closely The system calls available in the interface vary from operating system to operating system (although the underlying concepts tend to be similar). We are thus forced to make a choice between A1 vague generalities ("operat- ("operating systems have system calls tor reading files") and B) some specific system ("UNIX has a read system call with three parameters- one to specify tht; rile, one to tell where the data are to be put, and one to tell how many bytes to read" >. We have chosen the latter approach Its more work that way. but it gives more insight into what operating systems really do. Although this discussion specifically refers to POS1X (International Standard 9945-11. hence also to UNIX, System V. BSD. Linux. MINIX, ele . most other modern operating systems have system calls that perform the same functions, even if the details differ Since the actual mechanics of issuing a system cal! are highly machine dependent and often must be expressed in assembly code, a procedure lihiary is provided to make it possible to make system calls from C programs and often from other languages as well It is useful to keep the following in mind Any smgle-CPU computer can exe- cuic only one instruction at a time. It a process is running n user program in user mode and needs a system service, such as readitit! data from a file it has tn exe- The operating system then figures out what tbe calling process wants hy inspect- inspecting the parameters. Then u carries out the- system call and

returns control to the

<$IC) f, SYSTEM CALLS 45 instruction following the system call. In a sense, making a system call is like making a special kind of procedure call, only system calls enter the kernel and procedure calls do not. To make the system call mechanism clearer, let us take a quick look at the read system call. As mentioned above, it has three parameters: the first one speci- specifying the file, the second one pointing to the buffer, and the third one giving the numher of bytes to read. Like nearly all system calls, tt is invoked from C pro- programs hy calling a library procedure with the same name as the system call: read. A call from a C program might look like this: count = read(fd, buffer, nbytes); The system call (and the lihrary procedure) return the numher of bytes actually read in count. This value is normally the same as nbytes, but may be smaller, if, for example, end-of-file is encountered while reading If the system call cannot be carried out, either due to an invalid parameter or a disk error, count is set to -1, and the error number is put in a global variable, errno. Programs should always check the results of a system call to see if an error occurred. System calls are performed in a series of steps To make this concept clearer, let us examine the read call discussed above. In preparation for calling the read library procedure, which actually makes the read system call, the calling program first pushes the parameters onto the stack, as shown in steps 1-3 in Fig. 1-17. C and C++ compilers push the parameters onto the stack in reverse order for histori- historical reasons (having to do with making the first parameter to printf, the format string, appear on top ot the stack). The first and third parameters are called by value, but the second parameter is passed by reference, meaning that the address of the buffer (indicated by &) is passed, not the contents of the buffer. Then comes the actual call to the library procedure (step 4). This instruction is the nor- normal procedure call instruction used to call all procedures The library procedure, possibly written in assembly language, typically puts the system call number in a place where the oper itiiu > stem expects it such as a register (step 5). Then it executes a TRAP instruction to switch from user mode to kernel mode and start execution at a fixed address within the kernel (step 6). The kernel code that starts examines the system call numher and then dispatches to the correct system call handler, usually via a table of pointers to system call handlers indexed on system call number (step 7). At that point the system call handler runs (step 8). Once the system call handler lias completed its work, control may be returned

to the user-space library procedure at the instruction following the TRAP instruction (step 9). This procedure then returns to the user program in the usual way procedure calls return (step 10). To finish the job, the user program has to clean up the stack, as it does after any procedure call (step II). Assuming the stack grows downward, as it often does, the compiled code increments the stack pointer exactly enough to remove

Figure 1-17. The 11 steps m making the system call readfld. buffer, nbytes) the parameters pushed before the call to read. The program is now free to do whatever it wants to do next. In step 9 above, we said "may be returned to the user-space library procedure ..." for good reason. The system call may block the caller, preventing it from continuing. For example, if it is trying to read from the keyboard and nothing has been typed yet, the caller has to be blocked. In this case, the operating system will look around to see if some other process can be run next. Later, when the desired input is available, this process will get the attention of the system and steps 9-11 will occur. In the following sections, we will examine some of the most heavily used POSJX system calls, or more specifically, the library procedures that make those system calls. POSIX has about 100 procedure calls. Some of the most important ones are listed in Fig. 1-18, grouped for convenience in four categories. In the text we will briefly examine each call to see what it does To a large extent, the services offered by these calls determine most of what the operating system has to do, since the resource management on personal computers is minimal (at least compared to big machines with multiple users) The services include things like creating and terminating processes, creating, deleting, reading, and writing files, managing directories, and performing input and output.

SYSTEM CALLS Process management ! pid = waitpidjpid, &siatloc, options) ! s = execve(name, argv, ©nvironp) _ __ j exrt(status) _ __ cJ??te aJ**® P'-0_?eS5. identrcaUo the parent Wait tor a child to terminate Replace a process' core image Termmate process execution and return status Cail id = open(lrie how ...) s = close(id) n = reed(fd, buffer, nbytes) n =-wrife(fd, buffer, nbytes) position = Iseekffd offset, whence) s = staf(name, &buf) File management Description Open a JHe ior readmg, writing or both Cbse an open Me Read dafa from a file into a buffer Wrife data from a buffer into a file Move the file pointer Get a file's status information Directory and tile system management i = link(namei, name2) : unllnkfname) 3unt(speclai, name, Hag)

Create a new directory an empty directory Creale a new entry, name2, pointing to namel j Remove a directory entry Mount a file system Unmount a file systei Call s = cbdirfdtrname) s = cbmod(name, mode) a = ki!!(prd, signal) aeconds = t!me(«seconds) Miscellaneous Description Chenge the working directory Change a file's protection bits Send a signal to a process Get the elapsed time since Jan 1,1970 Figure 1-18. Some of ihe major POSIX system calls The return tode j- is - an error has occurred The reiura codes are as follows, pidis a process \d,jd file descriptor, n is a byte count, position is an offset within ihe f i* the etapsed time The parameieri are exptamed in the texi i finds As an aside, it is worth pointing out that the mapping ot" POSIX procedure calls onto system calls is not one-to-one. The POSIX standard specifies a number of procedures that a conformant system must supply, but it does not specify

48 INTRODUCTION CHAP I whether they are system calls, library calls, or something else. It a procedure can be carried out wilhoul invoking a system cad (i.e., without trapping to the kernel), il will usually be done in user space for reasons of performance. However, most ol the POSIX procedures do invoke system calls, usually with one procedure map- mapping directly onto one system call. Ill a few cases, especially where several required procedures arc only minor variations of one another, one system call han- handles more than one library call. 1.6.1 System Calls for Process Management The firsl group of calls in Fig. 1-18 deals wtlh process management. Fork is a good pldL-e to stari ihe discussion. Fork is the only way to create a new process in UNIX. Il creates an exacl duplicate of the original process, including all the file descriptors, registers—everything. After the fork, the original process and the copy (the parent and child) go their separate ways. Alf the variables have identi- identical values at the time of the fork, but since ihe parent's data are copied to create the child, subsequent changes in one of them do not arfeel the other one (The program text, which is unchangeable, is shared between parent and child.) The fork call returns a value, which is zero in the child and equal lo the child's process identifier or P1D in the parent. Using the returned P1D, the two processes can see which one is the parent process and which one Is the child process. In most cases, after a fork, the child will need to execute differenl code from the parent. Consider the case of the shell. It reads a command from the terminal, forks off a child process, wails for the child to execute the command, and then reads the next command when the child terminates. To wait for the child to fin- finish, the parent executes a waitpid

system call, which just wails until the child ter- terminates (any child if more than one exists). Waitpid can wait for a specific child, or for any old child by setting the first parameter to -1. When waitpid completes. the address pointed to by the second parameter, statloc, will be set to the child's exit status (normal or abnormal termination and exil value). Various options are also provided, specified by the third parameter Now consider how fork is used by the shell. When a command is typed, the shell forks off a new process. This- child process must execute the user command. It does this by using the execve system call, which causes its entire core image to be replaced by the file named in its first parameter (Actually, the system call itself is exec, but several different library procedures call it with different parame- parameters and slightly different names. We will treat Ihese as system calls here.) A highly simplified shell illustrating the use of fork, waitpid, and execve is shown in Fig. 1-19. In the most general case, execve has three parameters: the name of the file to be executed, a pointer to the argument array, and a pointer to Ihe environment array. These will be described shortly. Various library routines, including exeel, execv, execle, and execve, are provided to allow the parameters to be omitted or

SYSTEM CALLS #defineTRUE! while (TRUE) { /* repeat forever •/ type_prompt(). -" display prompt on the screef read_command(command. parameters); /• read input from terminal •/ if (fork( )(-Q){ /r *°rk off child process */ /* Parent code •/ waitpid{-1, Sstatus, 0); /• wait for child to exit */ }else{ /* Child code. •/ execve{command, parameters. 0); /• execute command */ :li Throughout the book, TRUt t1. assumed specified in various ways. Throughout this book we wTfl use the name exec to represent the system call invoked by all of these. Let us consider the case of a command such as cp Filet file2 used to copy file} to file2. After the sheff has forked, the child process locates and executes the file cp and passes to it the names of the source and target fifes. The main program of cp (and main program of mou olher C programs) con- contains the declaration mainfargc, argv, envp} where argc is a count of the number of items on the command line, including the program name. For the example above, argc is 3. The second parameter, argv, is a pointer lo an array. Element i o(Ihat array is a pointer to the j-th string on the command line. In our example, argv\Q\ would point to Ihe string "cp'", argv[\] would point to the string "fiiei" and argv\2) would point to the string ■'file2". The third parameter of main, envp, is a pointer to the environment, an array of

strings containing assignments of the form name = value used lo pass information such as the terminal type and home directory name lo a program. In Fig. I -19, no environment is passed to the child, so the third parameter of execve is a zero. If exec seems complicated, do not despair; it is (semantically) the most com- complex of all the POS1X system calls. All the other ones are much simpler As an example of a simple one, consider exit, which processes should use when they art finished executing. It has one parameter, the exit slatus @ to 255), which is returned to the parent via stathc in the waitpid system call.

50 INTRODUCTION CHAP Processes tn UNIX" have ihetr memory divided up tnlo ihrce segments the text segment (i.e , the program code), the data segment (i.e.. the variables), and the Stack segment. The data segment grov»s upward and the slack grows downward, as shown in Fig 1-20 Between them is a gap of unused address space. The stack grows into ihe gap automatically, as needed, but expansion of the data segment is done explicitly by using a system call. brk. which specifies the new address where the data segment is to end. This call, however, is nut defined by ihe POSIX stan- standard, since programmers aro- encouraged to use the malhic library procedure for dynamically allocating storage, and Ihe underlying implementation of malloc was not thought to be a suitable subject for standardization since few programmers use it directly 1.6.2 System Calls for File Management Many system calls relate to the file system In this section we will look at calls that operate on individual tiles; in ihu tiexi one we will examine those that involve directories or the file system as a wbole. To read or write a file, the file must llrst be opened using open. This call specifies the file name to be opened, either as an absolute path name or relative to the working directory, and a code or O-RDONLY, O. WRONLY, or 0_Wm. meaning open for reading, writing, or both To create a new Hie. O_ CHEAT is used. The file descriptor returned can then be used for reading or writing After- Afterward, the file can be closed by close, which makes the file descriptor available fur reuse on a subsequent open The most heavily used calls are undoubtedly read and write. We saw read earlier. Write has the same parameters. Although most programs read and wnti; Hies sequentially, tor some applica- applications programs need to be able to access any pan of a Hie at random. Associated with each file is a pointer that indicates the current position in the file When reading (writing) sequentially, it normally points to ihe next byte to be read (writ- (written). The

Iseek call changes the value of the position pointer, so that subsequent calls to read or write can begin anywhere m the file

SEC 1.6 SYSTEM CALLS 51 Lseek has three parameters: the first is the file descriptor for the file, the second Ts a file position, and the third tells whether the file position is relative to the beginning of the file, the current position, or the end of the file. The value returned by Iseek is the absolute position in the file after changing the pointer. For each file, UNIX keeps track of the file mode (regular file, special file, directory, and so on), size, time of last modification, and other information. Pro- Programs can ask to see this information via the stat system call. The first parameter specifies the file to be inspected; the second one is a pointer to a structure where the information is to be put, 1.6.3 System Calls for Directory Management In this section we will look at some system calls that relate more to directories or the file system as a whole, rather than just to one specific file as in the previous section. The first two calls, mkdir and rmdlr, create and remove empty directories, respectively. The next call is link. Its purpose is to allow the same file to appear under two or more names, often in different directories. A typical use is to allow several members of the same programming team to share a common file, with each of them having the file appear in his own directory, possibly under different names. Sharing a file is not the same as giving every team member a private copy, because having a shared file means that changes that any member of the team makes are instantly visible to the other members—there is only one file When copies are made of a file, subsequent changes made to one copy do not affect the other ones To see how link works, consider the situation of Fig. J-2J(a). Here are two users, ast and jim, each having their own directories with some files. If ast now executes a program containing the system call linkCVusr/ilm/memo", "/usr/ast/note"); the file memo inyim's directory is now entered into ast\ directory under the name note. Thereafter, /usr/jim/memo and /usr/ast/note refer to the same file. As an aside, whether user directories are kept in /usr, /user, /home, or somewhere else is simply a decision made by the local system administrator Understanding how link works will probably make it clearer what it does. Every file in UNIX has a unique number, its i-number, that identifies it. This i- number is an index inio a table of i-nodes, one per file, telling who owns the file, where its disk blocks are, and so on. A directory is simply a file containing a set of (i-number, ASCII name) pairs. In the first versions of UNIX, each directory entry was 16 bytes—2 bytes for the i-number

and 14 bytes for the name. Now a more complicated structure is needed to support long file names, but conceptually a directory is siill a set of (i-number, ASCII name) pairs. In Fig. 1-21, mail has \- number 16, and so on. What link does is simply create a new directory entry with a (possibly newj name, using the i-number of an existing file. In Fig. l-2)(b), two

INTRODUCTION 16 81 40 mail games 31 38 bin memo progl iusr/ast /usr/jim 16 81 40 70 matl games note 31 59 38 bin Ic progi (hi The same directories after linking. entries have ihe same t-number G0) and thus refer to the same file. It" either one is later removed, using Ihe unlink system call, the other one remains If both are removed, UNIX sees that no entries to the file exist (a field in the i-nodc keeps track of the number of direclory entries poinllnglo Ihe tile), so the file is removed from the disk. As we have mentioned earlier, the mount system call allows iwo file systems to be merged inlo one. A common situation is lo hdve Ihe root file system con- containing Ihe binary (executable) versions of Ihe common commands and other heavily used files, on a hard dfde. The user can ihen insert a floppy disk with files lo be read into Ihe floppy disk drive. By executing Ihe mount system call, the floppy disk file system can he attached to the root file system, as shown in Fig I -22 A typical statemeni in C to perform ihe mounl is mount{'7dev/fd0", 7mnt", 0), where the first parameter is the name of a block special file tot drive 0. Ihe second parameter is the place in ihe Hee where il is lo he mounted, and Ihe third parame- parameter tells whether the file system is to be mounted read-write or read-only. Figure After ihe mount call, a file on drive 0 can be accessed by just using ils path from Ihe root directory or Ihe working directory, withoul regard jo which drive il is on. In fact, second, third, and fourth drives can also be mounted anywhere in the tree. The mount call makes il possible to integrate removable media into a

is on Although this evample involves floppy disks, hard disks or portions of hard disks, (oflen called partitions or minor devices) can also be mounted this way When a file system is no longer needed, it can be unmounted with the urnounr sys- 1.6.4 Miscellaneous System Calls A variety of oiher system calk exist a.s well. We will look at just four of them here The chdir call changes the current working direclory. Afier the call chdirf/iisr/ast/tesr"), an open on the file ayz will open /u.\r/(ist/teit/-*-\7. The concept of a working directory eliminates the need for typing (long* absolute path names all the time. In UNIX

every file has a mode used for protection. The mode includes the read-write-execute bits for the owner, group, and others- The chmod system call makes it possible to change the mode 01 a file. For example, to make a tile read-readonly by everyone except the owner, one could execute chmodfiile". 0644), The kill sysiem call is [he way users and user processes send signals, jf a process is prepared to catch a particular signal, then when it arrives, a signal handler is run. If the process is not prepared to handle a signal, then its arrival kills the process (hence the name ot [he cull). POSlX defines several procedures for dealing with time. For example, time just returns the current time in seconds, with 0 corresponding io Jan 1. 1970 ai midnighl (just as the clay was starting, not ending) On computers with 32-bit words, the maximum value time can return is 2" - 1 seconds (assuming an unsigned integer is used). This \alue corresponds to a little over H6 years Thus in the year 2106. 32-bit UNJX systems will go berserk, imitating the famous Y2K problem. If you currently have a 32-bit UNIX system, you are ads'ised to trade it in for a 64-bit one sometime before the year 21(N 1.6.5 The Windows Win32 API So tar we have focused primarily on UNIX Now it is time to look briefly at Windows. Windows and UNIX differ m a fundamental way in their respective programming models. A UNIX program consists of code that does something or other, making system calls to have certain services performed. In contrast, a Win- Windows program is normally event driven The main program waits for some event io happen, then calls a procedure to handle it. Typical events are keys being struck, the mouse being moved, a mouse button being pushed, or a floppy disk inserted Handlers are then called to process the event, update the screen and

54 INTRODUCTION CHAP. I update the internal program state. All in all. this, leads, to a somewhat different style of programming than with UNIX, but since the focus of this book is on operannng system function and struciure, these different programming models will not concern us much more. Of course, Windows also has system calls. With UNIX, there is almost a 1- to-i relationship between the system calls (e.g , read) and the library procedures (e.g.. read) used to invoke the system calls. In other words, for each system call, there is. roughly one library procedure that is called to invoke it, as indicated in Fig. l-i7. Furthermore, POS1X has only about 100procedure calls. With Windows, the situation is radically different. To start with, the library calls and the actual system calls are highly decoupled- Microsoft has defined a

set of procedures, called the Win32 API (Application Program Interface) that pro- programmers are expected to use to get operating system services. This interface is (partially) supported on all versions of Windows since Windows 95. By decou- decoupling the interface from the actual system calls, Microsoft retains the ability to change the actual system calls in time (even from release to release) without invalidaiing existing programs. Whai aciuatiy constituies Win32 is also slightly ambiguous since Windows 2000 has many new calls ihat were not previously available. In this, section, Win32 means the interface supported by all versions of Window The number of Win32 API calls is extremely large, numbering in the thousands. Furthermore, while many of them do invoke system calls, a substan- substantial number are carried out entirely in user space. As a consequence, with Win- Windows it is impossible to see whai is a system call (i.e., performed by (he kemet) and what is simply a user-space library call. In fact, what is a system call in one ver ion of Windows may be done in user spare in a different version, and vice versa When we discuss the Windows system calls in this book, we will use the Win32 procedures (where appropriate) since Microsoft guarantees that these will be stable over time. But it is worth remembering that not all of them are true sys- system calls (i.e., traps to the kernel). Another complicalion is that in UNIX, the GUI (e.g.. X Windows and Motif) runs entirely in user space, so the only system calls needed for writing on the screen are write and a few other minor ones Of course, there are a large number of calls to X Windows and the GUI, but these are not system tails in any sense In contrast, the Win32 API has a huge number of calls for managing win- windows, geometric Figures, text, fonts, scrollbars, dialog boxes, menus, and other features of the GUI. To the extent that the graphics subsystem runs m the kernel (true on some versions of Windows but not on ail), these are system calls; othei- wise they are just library calls. Should we discuss these calls in this book or nol? Since they are not really related to the function of an operating system, we have decided not to, even though they may be carried out by ihe kernel. Readers interested in the Win32 API should consult one of the many books on the subject, for example (Hart, 1997; Rector and Newcomer, 1997: and Simon, 1997,.

5KC I (i SYSIKM CALLS Even introducing all tile Win32 AP! calls hwt is oui ol the C|ues|i will rotnct ourselves to those culls that ruughlv correspond in tlie runct; lhe UNIX calls listed in Fig 1-1% The.se arc lisied in Fig 1-23. Win32 _ j Descrlpt sateProcess Fcreate a new process rmdir I Re moveDi rectory

Remove an empty directory |_i(nK_ i (none) | Win32 does not support link I unlink Destroy an Win32 does Win32 does Change the current working directory Win32 does not support security (although NT Win32 does not support signals Get thecurrent time Let us now briefly go through the list ol Fig 1-23 CreateProcess creates a new process. It does the combined work ot fork and execve in UNIX, ll has many parameters specifying the properties of the newly created process. Windows does not have a process hierarchy as UNIX does so (here is no concept of a parent proc- process and a child process After a process is created, the creator and createe are equals. WaitForSingleObject is used to wait for an event Many possihle evems can he wailed for. If the parameter specifics a process, then lhe caller waits fur lhe specified process to exit, which is done using ExitProcess The next six calK operate on files and arc hmclionalK similar to their IjNfX counterparts although they differ in lhe parameters and details Slifl. files tan bti opened, closed, read, and written preny much as in UNfX The SetFilePointer and GetFileAttributesEx calls sel the file position and gel some of lhe file attributes.

56 INTRODUCTION CUM1 I Windows has directories and they are created with Create Directory and RemoveDirectory. respectively. There is, -ilso j noiion of a current directory, sel by SetCurrentDirectory. The current time is acquired using GetLocalTime The Win32 interface does not have links to files, mounted file systems, secu- security or signals, so the calls corresponding lo lhe UNIX ones do noi exisi Of course. Win32 has a huge numher of other calls that UNIX does not have. espe- especially for managing the GUI. And Windows 2000 has an elaborate security sys- teiii and also supports file links One last note about Win32 is perhap, worth making Win32 ,s not a lerrihly uniform or cOnsisterU interlace. The main culprit here was lhc need lo be hack- ward compatible with the previous F-bit inierfaee used in Windows 3." 1.7 OPERATING SYSTEM STRUCTURE Now lhat we have seen what opersting systems look like on the outside (i.e.. lhe programmer's inierface), ii is time to take a look inside. In the following sec- sections, we will examine five different structures thai have been iried. in order lo gel some idea of the spectmm of possibilities. These are by no means exhauslive. hul they give an idea of some designs lhat have been iried in practice. The five designs are monolithic sysiems, layered sysiems. virtual machines. L-xokernels. and clieni-server systems. 1.7.1 Monolithic Systems By far the mosi common organization, this approach might weli be suhtitled "The Big Mess." The

stmcture is lhat there is no stmciure. The operating system is wniien as a collection of procedures, each of which can call any of the oihei ones whenever it needs lo. When ihi<: technique is used, each procedure in the system has a well-defined inierface in lerms of parameters and results, and each one is free to call any oilier one. if lhc latter provides some useful compulation lhat the former needs. To construct lhe actual object program of the operating system when this approach is used, one first compiles all lhe individual proceduies, or files coniain ing lhe procedures, and lhen binds lhem alt logelher imo a single objed file using lhe sysiem linker. In lerms of information hiding, lhere is essentially none—e very procedure is visible lo every oilier procedure (as opposed lo a siruclure containing modules or packages, in which much of the informaiion is hidden away inside modules, and only the officially designated entry points can be called from out- outside lhe module). Even in monolithic sysienu,, however, ii is possible to have a\ least a litile stmciure. The services (system calk) provided by lhe operating system arc requested by putling the parameters in a well-defined place (e.g., on lhe slack) and

SEC. 1 7 OPERATING SYSTFM STRUCTURE 57 lhen execulmg a Iran instrudion. This msirudion switche.s the machine from user mode to kernel mode and lraitsfers control 10 the operating system, shown as step 6 in Fig. f-17. The operating system lhen fetches the parameters and determines which system call is lo he carried out. After that, ii indexes inio a table lhat ton- lains in sloi k a poimer to lhe procedure thai tarries out system call k (step 7 m Fig. 1-17). This organization hiiggesls a basic struclure for lhe opersting system. 1. A main program lhai invokes lhe requested service procedure. 2. A sei of service procedures that carry out the sysiem calls. 3. A set of miliiy procedures ihst help the service procedures. In (his model, for each sysiem call there is one service procedure thai lakes care of iL The utility procedures do things lhat are needed by several service pro- procedures, such as fetching data from user programs. This division of the pro- procedures into three layers is shown in Fig. 1-24. Figure 1-24. A simple stractunng model for a monoliihic system 1.7.2 Layered Systems A generalization of the approach of Fig. 1-24 is to organize lhe operating .sys- .system as a hierarchy of (ayers, each one constmcled upon the one below if. The firsi sysiem conslmcled in this way was the THE system built at the Technische Hogeschool Eindhoven in the Netherlands, by E. W. Dijkstra ((968) and his stu- students. The THE system

was a simple batch sysiem for a Dutch computer, the Electroiogica X8, which had 32K of 27-bit words (bits were expensive back then). The system had 6 (ayers, as shown in Fig. 1-25. Layer 0 dead with allocation of the processor, swiiching between processes when interrupts occurred or timers expired. Above layer 0, the system consisted of sequential processes, each of

58 INTRODUCTION CHAP. I which could be programmed without having to v.t>rry about the tact that multiple processes were running on a single processor In other words, layer 0 provided the basic multiprogramming of the CPU. Function Input/output n 2 joperator-pro Memory antf flrum management Processor allocation and multiprogramming Figure t-25. Structure of the THE operating ■-ytem Layer 1 did the memory management It allocated space for processes in main memory and on a 512K word dmm used tor holding parts of processes (pages) for which there was no room m main memory Above layer 1. processes did not have to worry about whether they were in memory or on the drum: the layer] software took care ot making sure pages were brought into memory when- whenever they were needed. Layer 2 handled communication between each process and the operator con- console. Above this layer each process effectively had its own operator consols. Layer 3 took care of managing the I/O devices and buffering the information streams to and from them. Above layer 3 each process could deal with abstract I/O devices with nice properties, instead of real devices with many peculiarities Layer 4 was where the user programs were found They did not have to worry about process, memory, console, or I/O management The system operator proc- process was located m layer 5 A further generalization of the layering concept was present Tn the MULT1CS system Instead of layers, MULTICS was described as having a series of concen- concentric rings, wTth the inner ones being more privileged than the outer ones (which is effectively the same thing). When a procedure in an outer nng wanted to call a procedure in an inner ring, it had to make the equivalent of a system call, that is. a TRAP instmction whose parameters were carefully checked for validity before allowing the call to proceed. Although the entire operating system was part of the address space of each user process in MULTICS, the hardware made it possible to designate individual procedures (memory segments, actually) as protected againsi reading, writing, or executing. Whereas the THE layering scheme was really only a design aid, because all the parts of the system were ultimately linked together into a single object pro- program, in MULTrcS. the ring mechanism

was very much present at run time and enforced by the hardware. The advantage of the ring mechanism is that it can easily be extended to stmcture user subsystems. For example, a professor could

OPERATING SYSTEM STRUCTURE tie a program to tesi and grade student prograr :h the student programs running in ring n + 1 s and mn this program in ring n, that they could not change their 1.7.3 Virtual Machines The initial releases of OS/360 were strictly batch systems. Nevertheless, many 360 users wanted to have timesharing, so various groups, both inside and outside IBM decided to write timesharing systems for it. The official IBM timesharing system, TSS/360, was delivered late, and when it finally arrived it was so big and slow that few sites converted to rt. It was eventually abandoned after its develop- development had consumed some $50 million (Graham, 1970). But a group at IBM's Scientific Center in Cambridge, Massachusetts, produced a radically different sys- system that IBM eventually accepted as a product, and which is now widely used on its remaining mainframes. This system, originally called CP/CMS and later renamed VM/370 (Seawrrght and MacKinnon, 1979), was based on an astule observation: a timesharing system provides (I) multiprogramming and B) an extended machine with a more con- convenient interface than the bare hardware. The essence of VM/370 is to completely separate these two functions. The heart of the system, known as the virtual machine monitor, runs on the bare hardware and does the multiprogramming, providing not one. but several vir- virtual machines to the next layer up, as shown in Fig. I -26. However, unlike all other operating systems, these virtual machines are not extended machines, with files and other nice features. Instead, they are exact copies of the bare hardware, including kernel/user mode, I/O, interrupts, and everything else the real machine ruceons here — Trap here — -\ CMS Jl 370 CMS VM/370 lare hardwa r CMS t- e - System - Trap he ca"s re |h CMS ■ue hardware, each one can Figure 1-26. The structure of VM/370 \ Because each virtual machine rs identical to the run any operating system that will run directly on the bare hardware. Different virmal machines can, and frequently do, run different operating systems. Some run one of the descendants of OS/360 for batch or transaction processing, while other ones run a single-user, interactive system called CMS (Conversational Monitor System) for interactive timesharing users.

 When a CMS program executes, a system call,

the call is trapped to rhe operat- operating system in its own virtual machine, not to Vm/370. just as it would if it were running on a real machine instead of a virtual one. CMS then issues the norm.il hardware I/O instruction-, for reading its virtual disk or whatever is needed ro carry out the call. These I/O instructions are trapped by VM/370. which then per- performs them as part of its simulation of the real hardware. By completely separat- separating the functions of multiprogramming and providing an extended machine, each of ihe pieces can be much simpler, more flexible, and easier to maintain. The idea of a virtual machine is heavily used nowadays in a different context: running old MS-DOS programs on a Pentium (or other 32-bit Intel CPU). When designing the Pentium and its software, both Intel and Microsoft realized that there would be a big demand for running old software on new hardware For this reason, Intel provided a virtual 8086 mode on the Pentium. In this mode, tbe machine acts like an 8086 (which is identical to an 8088 from a software point of view), including 16-bit addressing with a I-MB limit. This mode is used by Windows and other operating systems for running MS- DOS programs. These programs are started up m virtual 8086 mode. As long as they execute normal instructions, they run on the bare hardware. However, when a program tries to trap to the operating system to make a system call, ur tries tu do protected I/O directly, a trap to the virtual machine monitor occurs. Twci variants on this design are possible In the first one, MS-DOS itself is loaded into the virtual 8086's address space, so the virtual machine monitor just reflects the trap back to MS-DOS, just as would happen on a real 8086, When MS-DOS later tries to do the I/O itself, that operation is caught and carried oul by the virtual machine monitor. In the other variant, the virtual machine monitor just catches ihe first trap and does the I/O itselt, since it knows what all the MS-DOS system tails are and thus knows what each trap is supposed to do. This variant is less pure than the first une, since it only emulates MS-DOS correctly, and not other operating sy.stcms, as the first one does. On the other hand, it rs much faster, since it saves the trouhle of starling up MS-DOS to Jo the I/O. A further disadvantage of actually running MS-DOS m virtual 8086 mode is that MS-DOS fiddles arouiid with the interrupt enable/disable bit quite a lot. all of which must be- emulated at considtiable cost It is worth noting that neither of these approaches arc really the same as VM/37U, since the machine being emulated is not a full Pentiuni. but only an 8A86. With the VM/37U system, it is possible to run VM/370. it sell, m the virtual machine. With the Pentium, it is not possible to run, say. Windows in the virtual 8086 because no version ot

Windows runs on an 8086; a 286 is the minimum tor even the oldest version, and 286 emulation is not provided (let alone Pentium emulation). However, by modifying the Windows binary slightly, this emulation is possible and even available in commercial products Another area where virtual machines are used, but m a somewhat different way. is for running Java programs. When Sun Microsystems invented the Java

SEC 17 OPERATING SYSTEM STRUCTURh 61 programming language, it also invented a virtual machine (i.e . a computer aichi- tccture) called the JVM (Java Virtual Machine). The Java compiler produces code for JVM, which then typically is executed by a software JVM interpreter. The advantage of \hi* approach is thai the JVM code can be shipped over the Internet to any compuier that has a JVM interpreter and run there. If the compiler had produced SPARC or Pentium binary programs, for example, they could not have been shipped and run anywhere as easily. (Of course, Sun could have pro- produced a compiler that produced SPARC binaries and then distributed a SPARC interpreter, bul JVM is a much simpler architecture to interpret.) Another advan tage of using JVM]S that if the interpreter rs implemented properly, which is not completely trivial, incoming JVM programs can be checked for safety and then executed in a protected environment so they cannot steal data or do any damage. 1.7.4 Exokernels With VM/370. each user process gets an exact copy of the actual computer With virtual 8086 mode on the Pentium, each user process gets an exact copy of a different computer. Going one step further, researchers at M I.T have built a sys- system that gives each user a clone of the actual computer, but with a suhset of the resources (Engler et al , 1995). Thus one virtual machine might get disk blocks 0 to 1023, the next one might get blocks 1024 to 2047. and so on At the bottom layer, running in kernel mode, is a program called the exoker- nel. Its job is to allocate resources to virtual machines and then check attempts to use them to make sure no machine is trying to use somebody else's resources. Each user-level virtual machine can run its own operating >ystem, a.s on VM/370 and the Pentium virtual 8086s, except that each one is restricted to using only the resources it has asked tor and been allocated The advantage of the exokemel scheme isthat it saves a layer ot mapping. In the other designs, each virtual machine thinks it has its own disk, with blocks run- running from 0 to some maximum, so the virtual machine monitor must maintain tables to remap disk addresses (and all other resources) With the exokernel. this remapping is not

needed The exokemel need only keep track ot which virtual machine has been assigned which resource. This method still has ihe advanrjge of separating the multiprogramming (in the exokernel) from the user operating system code (in user space), but with less overhead, since all rhc exokemel has to do is keep the virtual machines out of each urtier's hair 1.7.5 Client-Server Model VM/370 gains much in simplicity by moving a large part ot the traditional operating system code (implementing [he extended machine) into a higher layer. CMS. Nevertheless, Vm/370 itself is still a complex program because simulating a

62 INTRODUCTION CHAP. <t (hat simple (especially if you want number of virtual 370s rn therr entirety i. to do it reasonably efficiently). A trend in modem operating systems is to take the idea of moving code up into higher layers even further and remove as much as possible from kernel mode, leaving a minimal microkernel. The usual approach is to implement most of the operating system in user processes. To request a service, such as reading a block of a file, a user process (now known as the client process) sends the request to a server process, which then does the work and sends back the answer. > User mode Cllenl Client server server Microkernel \ Memory Figure 1-27. The c I leni-server model In this model, shown in Fig. 1-27, all the kernel does is handle the communi- communication between clients and servers. By splitting the operating system up into parts, each of which only handles one facet of the system, such as file service, process service, terminal service, or memory service, each part becomes small and manageable. Furthermore, because all the servers run as user-mode proc- processes, and not in kernel mode, they do not have direct access to the hardware. As lay crash. dap b ! * h age le ok me h n o use H rverby idis- send- h ndted locally o server appens in on a both consequence, if a bug in the file server is triggered, the frle s but this will not usually bring the whole machine dt Another advantage of the cllent-server mode tnbuted systems (see Fig. 1-28). If a client mm n ing it messages, the client need not know whether the r in its own machine, or whether it was sen o remote machine. As far as the client is concerned, the cases: a request was sent and a reply came back. The picture painted above of a kernel that handles only the transport of mes- messages from clients to servers and back is not completely realistic. Some operating system functions (such as loading commands into the physical I/O device regis- registers) are difficult, rf not impossible, to do from user-space programs.

There are two ways of dealing with this problem. One way is to have some critical server processes (e.g., I/O device drivers) actually run in kernel mode, with complete access to all the hardware, but still communicate with otber processes using the normal message mechanism. The other way is to build a minimal amount of mechanism into the kernel but leave the polky decisions up to servers in user space (Levin et a)., 1975). For

OPERATING SYSTEM STRUCTURE Figure 1-28. The client-server mode) in a distnbuied sysiem example, tbe kerne' might recognize that a message sent to a certain special address means to take the contents of that message and load it into the I/O device registers for some disk, to start a disk read. In this example, the kernel would not even inspect the bytes in the message to see if they were valid or meaningful; it would just blindly copy them into the disk's device registers. (Obviously, some scheme for limiting such messages to authorized processes only must be used.) The split between mechanism and policy is an important concept; it occurs again and again in operating systems in various contexts. 1.8 RESEARCH ON OPERATING SYSTEMS Computer science is a rapidly advancing field and it is hard to predict where it is going. Researchers at universities and industrial research lab-s are constantly thinking up new ideas, some of which go nowhere but some of which become the cornerstone of future products and have massive impact on the industry and users. Telling which is which turns out to be easier to do in hindsight than in real time. Separating the wheat from the chaff is especially difficult because it often takes 20-30 years from idea to impact. For example, when President Eisenhower set up the Dept. oi" Defense's Advanced Research Projects Agency (ARPA) m 1958, he was trying to keep the Army from killing the Navy and the Air Force over the Pentagon's research bud- budget. He was not trying to invent the Internet. But one of the things ARPA did was fund some university research on the then-obscure concept of packet switch- switching, which quickly led to the first experimental packet-switched network, the ARPANET. It went live in 1969. Before long, other ARPA-funded research net- networks were connected to the ARPANET, and the Internet was bom. The Internet was then happily used by academic researchers tor sending email to each other for 20 years In the early 1990s, Tim Bemers-Lee invented the World Wide Web at the CERN research lab m Geneva and Marc Andreesen wrote a graphical browser for ii at the University of Illinois. All of a sudden the Internet was full of chatting teenagers. President Eisenhower is probably rolling over in his

grave.

64 INTRODUCTION CHAP 1 Research m operating systems has also led to dramatic changes m practical systems As we discussed earlier, the first commercial computer systems were all batch systems, until MJ.T. invented interactive timesharing in the early 1960s. Computers were all text-based until Doug Engelbart invented the mouse and the graphical user interface at Stanford Research Institute m the late 1960s. Who knows what will come next1 In this section and m comparable sections throughout the book, we will take a brief look at some of the research in operating systems that has taken place during the piist 5 to 10 years, just to give a flavor of what might be (in the horizon. This introduction is certainly not comprehensive and is based largely on papers that have been published in the lop research journals and conferences because these ideas have at least survived a rigorous peer review process in order to get pub- published. Mom of the papers cited in the research sections were published hy either ACM. the IEEE Computer Society, or USENlX and are available over the Inter- Internet to (student) members of these organizations. For more information about these organizations and their digital libraries, see ACM http://www.acm.org IEEE Computer Society http://www.computer.org USENlX http://www.usenix.org Virtually all operating systems researchers realize that current operating sys- systems are massive, inflexible, unreliable- insecure, and loaded with bugs, certain ones more than others (names withheld here to protect the guilty). Consequently, there is a lot of research on how to build flexible and dependable systems. Much of the research concerns microkernel systems. These systems have a minimal kernel, so there is a reasonable chance they can be made reliable and be debugged. They are also flexible because much of the real operating system runs as user-mode processes, and can thus be replaced or adapted easily, possibly even during execution. Typically, all the microkernel does is handle low-level resource management and message passing between the user processes. The first generation microkernels, such at; Amoeba (Taneubaum et a!.. 1990). Chorus (Rozier et al., 1988), Mach (Accetta et al., 1986). and V (Chertton, 1988), proved that these systems could be built and made to work. The second genera- generation is trying to prove that they can not only work, but with high performance as well (Ford et al., 1996; Hartig et al., 1997; Liedtke 1995. 1996; Rawson 1997; and Zubert et al., 1999). Based on published measurements, it appears that this goal has been achieved. Much

kernel research is focused nowadays on building extensible operating systems. These are typically microkernel systems with the ability to extend or customize them in some direction. Some examples are Fluke (Ford et al.. 1997). Paramecium (Van Doom et aL, 1995), SPIN (Bershad et al., 1995b), and Vino (Seltzer et al., 1996). Some researchers are also looking at how to extend existing

SFC t 8 RbSbARCH ON OPERATING SYSTEMS 65 systems (Ghormley et at.. tsJ9«). Many of these systems allow users to add their nwn code to the kernel, w hich brings up the obvious problem of how to allow user extensions in a secure way. Techniques include interpreting the extensions, res- restricting them to code sandboxes, using type-safe languages, and Code signing (Grimm jind Bershad. 1997: and Small and Selt/.er. 1998). Druschd ct al A997) present a dissenting view, saying that too much effort is going into security for user-extendahte systems In their view, researchers should figure out which extensions are useful and then just make those a normal part of the kernel, without the ability to ha\e users extend the kernel on the fly. Although one approach to eliminating bloated, buggy, umeliable operating system altogether. Tliis approach k being taken by the group of Kaaslioek at M l.T m their Exokeniel research. Here the idea is to have a thin layer of software running on the bare metal, whose only juh is to securely allocate the hardware- resources among the users. For example, it must decide who gets to use which part of the disk and where incoming network packets should be delivered. Everything else is up to user-level processes, making it possihle to hut Id hpth general-purpose and highly-specialized operating systems (Rngler and Kaashoek, 1995; Englcr et al., 1995; and Kaashoek et al . 1997) 1.9 OUTLINE OF THE REST OF THIS BOOK We have now completed our introduction and hird's-eyc view of the operating system. It is time to get down to the details. Chapter 2 is about processes. It discusses their properties and how they communicate with one another. It also gives a number of detailed examples of how interprocess communication woiks and how to avoid some of the pitfalls. Chapter 3 is about deadlocks. We briefly showed what deadlocks arc in tlii;, chapter, but there js much more to say Ways to prevent or avoid them are div- cussed. In Chap. 4 we will study memory management in detail The important topic of virtual memory will be examined, along with closely related concepts such as paging and segmentation. Input/Output is covered m Chap. 5. The concepts of device independence ami device dependence will be looked at. Several important

devices, including disks, keyboards, and displays, will be used as examples. Then, in Chap. 6, we come to the all-important topic of file systems To a considerable extent, what the user sees is largely the file system We will look at both the file system interface and the file system implementation. At this point we will have completed our study of the basic principles of single-CPU operating systems. However, there is more to say. especially about advanced topics. In Chap. 7. we examine multimedia systems, which have a num-

66 INTRODUCTION CHAP. 1 her of properties and requirements that differ from conventional operating sys- systems. Among other iiems, scheduling and the file system are affected by the nature of multimedia. Another advanced topic is multiple processor systems, including multiprocessors, parallel computer1-, and distributed systems. These subjects are covered in Chap. 8. A hugely important subject is operating system security, which is covered m Chap 9. Among ihe topics discussed in this chapier are threats (e.g., viruses and worms), protection mechanisms, and security models- Next we have some case studies of real operaiing systems. These are UNIX (Chap. 10) and Windows 2000 (Chap. 11). The book concludes with some thoughts about operating system design in Chap. 12. 1.10 METRIC UNITS To avoid aiiy confusion, it is worth stating explicitly that in this book, as m computer cience in general, meiric units are used instead of traditional English units (the furlong stone-fortnight system). The principal metric prefixes are listed in Fig 1 29 The prefixes are typically abbreviated by their first letters, with ihe units greater than 1 capitalized. Thus a 1-TB database occupies 1012 bytes of torage and a 100 psec (or 100 ps) clock ticks every 10"'° seconds. Since mtlli and micro both begin with the letter "m," a choiee had to be made Normally, m is for rnilli and "(T' (the Greek letter mu) is ror micro. "Expttclt _ ~ "Tprelto ', Ex~p.~| Explicit^ ^_ __ I Prefix ! .... — _, QQO [Kto 1,000,000 , Mega 1,000,0o"o"^ooo oo'o" Tera .' _ _ l 000,000.000000,000 PalaJ 1,000,000,000.000.000,000| Exa_ ^qooIooo1oop.ooo1ooo_ooo,oqo I zetta 1,000 000,000,000,000,000,000,000 j_Yotta Figure 1-29- The prinupa! meinL prefixes It is also worth pointing out that tor measuring memory sizes, in common industry practice, ihe units have slightly diftereni meanings There Kilo means 2'° A024) rather than 1()' A000) because memories are always' a power of two. Thus a 1-KB memory contains 1024 hyics, not 1000 bytes. Similarly, a 1-MB memory contains 2* A.048.576) hyies and a 1-GB memory contains 21" A,073,741,824) byles However, a 1-Kbps communication line

transmits 1000 bits per second and a 10-Mbps LAN runs at 10,000,000 bits/sec because t/iese speeds are not powers ot two- Unfortunately, many people lend to mix up these

SEC 1.10 METRIC UNITS 67 iwo systems, especially for disk sizes. To avoid ambiguity, in this book, we will use the symbols KB, MB, and GB for 210, 2:o, and 230 bytes respectively, and ihe symbols Kbps, Mbps, and Gbps for 101,106 and 109 bits/sec, respectively. 1.11 SUMMARY Operating systems can be viewed from two viewpoints: resource managers, and extended machines In the resource manager view, the operating system's job is to manage the different parts of the system efficiently. In the extended machine view, the job of the system (s to provide the users with a virtual machine that is more convenient to use than the actual machine. Operating systems have a long history, starting from the days when they replaced the operator, to modem multiprogramming systems. Highlights include early batch systems, multiprogramming systems, and personal computer systems. Since operating systems interact closely with the hardware, some knowledge of computer hardware is useful to understanding them. Computers are built up of processors, memories, and I/O devices. These parts are connected hy buses. The basic concepts on which all operating systems are built are process(.■•>. memory management, I/O management, the file system, and security, Rach of these will be treated in a subsequent chapter The heart of any operating system is ihe sei of system calls ihal n can handle. These lell whai the operating system really does. For UNIX, we have looked ai four groups of sysiem calls. The first group of system calls relates lo process creation and termination. The second group is for reading and writing files,. The third group is for directory management The fourth group contains, miscellane- miscellaneous calls. Operating systems can be structured in several ways The most common ones are as a monolithic system, a hierarchy of layers, a virtual machine system, an exokemel, or using the client-server model 1. 2, 3. 4. What; Whflli What j as a si; On eai CPU (ire ihe two mam ^ multiprogrami s spooling'1 Do indard feature jr ■]y computers, c j e iherc was multiprogramming? i functions, ol an ■ning? you Ihink Ihal ; i Ihe future? :very byte of di no DMA). Wh ;lly handled by ifie janiiation have lor

68 INTRODUCTION CHAP 1 5. Why was, timesharing nol widespread on sec 01 Id-generation computers ' 6. The lamily of computers idea was introduced

in the 1960s with the IBM Sj'sleni/360 7 One redson GUIs were initially slow Hi he jduplcd was tlic cost of the hardware ' iic-cdc-d id support then How muit udeo RAM is iicc-dcd to support <i 15 line x SO low chattier monochrome texi screen" llow much lor .1 1024 x 768 pixel 24-liii color hitinap'* Whai was Ihe cost of tins RAM ai 1980 fmcc-S (S^/KB)' How much is 8 Which (ifllie following msli"uctions should be allowed (inlv in kerntl moilc (a) Disable all interrupts (b) Read llie time-of-d,\y clock. (e) Set !he lime-of-day dock. Id) Change the memory map 9. Lisi some differences between personal computer opcnilinj! systems and iiidiiifrJi'ie operdiing sysiems. 10. A computer has a pipeline with tour Sldgcs Kdrii Siajre tdkes ilie same lime 10 do its 11. An altrl reviewer noiit.es a consiSlent spelling eri'ii m the niEuiuSviipt 01 jh opci':ilin^ sysiems lexlbocik lhat is dbout to go to press llie book lids aptiruximatcly 7(J() f>.iyes. c<iclt \vilh 50 Ijnes oi K0 eharacltrs each, llosv long will it lakt 10 t'lccirsinicatly sc:tn the K-xl for the ease ol ft^ masler copy being in e.ich d! !hc lesels o! iiidiicuj ,it Fiji 1 -7 ' For]Al^]iial sltirage methods^ (_\>nsid*r Ihjl the access liuic ^i^-cn is ^i chur-acier. tor disk devices assume Hie liinc is pet blcick dl 10Z4 cli.trjiclers. ami foi lJ]>c speed as disk access 12» In P]g 1 ~9» the MMt^ couiiiiiies Jlie jjivdiiiiui^ t\ uIli<iIt iiddic^s io ttic FitYiit icslstvi causJEig a fault if it is loo lar^e An alfcn]ali\o ^lesigii w^>uld be to lu'sl add the virtual address !o the base registei and Ihc^n compaie tlie icsiilt !o tlic (physical) asldress in Hit-limil register Are Ihe two iiKtliods ingnalK equu jlciil' Ate Ihev etiui\afctil in |>cr- 13" Wljcn a user program makes a system \Tall io ic<ul ar ^^ritf ii disk tile, u p]i">\idt-'s uii uidualuin ot which lilc it wauls, it pomlei U) ihe d,ila butlt'i .HiJ ilie Clilliu C ^nitUil is Ihen transferred to the operating systt-ni. wlik.li calls Hie appioiin.itc diuei SupiiOM' no d.ita tor il| What about llie east of vnilinj; to the disk'1 Need lilt c;illtr he block- my awaiting completion i>f the disk Iranslvi" 14- \V'h,i(is the kc> diftcreiice between a trjp and .in mteiriipl ' and is loaded at address -iOJIOO What \alucs clii tlie bine and limit legislcr i>et oceout"- mg Io the sehc-me described in ihe- tcsC

CHAP 1 PROBLEMS 16. Why is the- pmccss table ncedcir in a liinc-sturing 17. Is there any reason why you migh! want U> mt,unl a file syslem on a nonempty direc- toiy1' if so. what is it' 18. Foi \-aeh of [he following system calls, give a u>ii(iition th.it causes it to tjjl- iork. exec mid untink. 1"». Can the counl = write(td, buffer, nbytes); cjll leluin any \alue in iomii other lhaii nh\tes ' it so. why > 20. A die whose flic deseiiplur is fil contains tlie followiiig

secjuwnc ol bytes 1. 1-4. 1,5, <) 7, 6. 5, 3. 5 The following system culls Jre made lseek(ftt, 3, SEEK_SET), read(fd, Sbuffer, 4), uherc Hie Iseek call makes a seek Ip byte 1 ot Hie Hie What does buffst .imunn at lor ibu read lias complelcd' 21. What is the essential dilrercnee bclwcen <i hlock special file and a eb.ir.iaci special call itself us ejllcd lead Is iJ essential thai binb ^>1 tbese b.i\i> tbt sumc u.niie ' If not. wbich One is mure imp<)riant? 23. The \- Ucni-server iiiosiel is popular in distuhuied systems Can it also he used m a 24. To a programmer, a sySiem call looks like aus other tail to a library piotedmc K it impctrt.ini ihsil a progr.HTiiiier know whicb library J^rOteduici resuil in ss stein cills.' Llnder what circimisliinces iind wh\ > 25. Figure 1-21 shows thai a number or liNJX sjslem tails. Haw no Win 12 Ai'l cqui\- 26. Here are sdhic questions lor prueticing unit tomersions (a) How long is .i midyear in seconds > (c) How man\ bytes an- there in a I-TB memorj ' (d) The mass'of the e.uth is 6IMXI yotiagmms What is tli.it in kilograms' 27. Wri|e ;i sbeii that i*, simiiar to Fig i-|9 but contains emiugh todt thai input and outpul. pipes, and background |obs 28. it you have a personal UNIX-like system (Linux. MiMX. Free BSD. en) thai you carl saicly cnish ;ind reboot, wfite a sbell seripi thnt Jtlcnipts U"

INTRODUCTION I number or child processes and observe what happens. Before running the nt. type sync lo the shell Lo flush Ihe file system butters to disk to avoid ruin- 29. C\amine and !ry lo interpret the contents of d 1'NIX-like or Windows directory with a tool like Ihe UNIX od program or the MS-DOS DEBUG program Hint How you do this will depend upon what Lhe OS allows One trick that may work is 10 treate d directory on a floppy disk with one operating system jnd then read Lhe ww dis-k dal.t using a different operating system that allows such access

PROCESSES AND THREADS We ,ire nmv about in embark nn a detailed stud) ot how operating systems -ire designed and constructed The most central coiicepl in any operating system is the proem: an abstraction fit a running piogram- K\er>!biiig else binges on this concept, alid II is impo'tan! that Ibc operating system designer (and student) have ,i Iborougb understanding of what a process is as early as possible 2.1 PROCESSES All modern computers can do several Ibings al Ibc same it me While- runmiig a user program, a computer can also be reading troni ^ disk and outpLitiini? Text \o a screen or pnntei. In a multiprogramming system, the CPU ulso switches from program io program,

running each tor lens or hundreds ol nulliseconds While, strictly speaking, at any inslani of time, the CPL' ts running only (>ne prugiain. in the course of 1 second, il nia^ work on several program... thus giving Ihc users Ihe illusion ot parallelism. Sometimes people speak or pseud o parallel ism m this context, lo contrast it wilh tlie true hardware parallelism ot multiprocessor sys- systems (which have two or more CPUs sharing the aiiie physical memory) Keep- Keeping track of multiple, parallel activities is hard tor people- to do. Therefoie, oper- operating system designers over the years have evoived a conceptual model (sequen- (sequential processes) thai makes parallelism easier to deal with. That model, its uses, and some of its consequences form the subject of this, chapter.

72 PROCESSES AND THREADS CHAP 2 2.1.1 The Process Model in this model, all the runnaWe software on the computer, sometimes including the operating system, is organized into a numher of sequential processes or just processes for short. A process is just an executing program, mcltiding the curren] values ot the program counter, registers, and variables. Conceptually, each proe- think aboii to process, but to understand the system. i| i> much ea-. :tion of processes running in (pseudo) parallel, than to iapid switching hack and forth is called multiprogramming, as we saw in Chap. 1 In Fig. 2-| (a) we see a computer multiprogramming tour programs in memory. In Fig. 2-1 (b) we see four processes, each with its own How of control (i.e.. its own logical program counter), and each one runnitig independently of the process runs, its logical program counter is loaded into the tea] program counter fo the time being, the physical progiaru counter is saved in I F 2l() see that , ts lgcal prog finished for the t g, py pg g pogram counter in memory. In Fig. 2-l(c) we see that long enough time interval, all the processes have made piogress, tant only one process is actually running. t finis for t the process' logical prog viewed ove l but at any g r program counters Figure 2-1. (a) Multiprogramming ot four programs (b) Ctmuptual model tit four Independent, sequentl.it processes (t.) Onfy one program is active at once. With the CPU switching back and iortb among the processes, the rate at which a process performs its computation will not be uniform and probably not even reproducible if the same processes are run again. Thus, processes must not be programmed with built-in assumptions about tuning. Consider, lor example, an I/O process that starts a streamer tape to restore backed up files, executes an idle loop 10,000 limes to let it get up to speed, and then issues a command to read the first record. If the CPU decides

to switch fo another process during the idle loop, the tape process might not run again until after the first record was already past the read head. When a process has critical real-time requirements like this,

SEC 2 1 PROCESSES 73 that is, paiticular events mini occur within a spccihed number of milliseconds, special measures must be taken to ensure that they do occur Normally, huwever. most processes are not affected fry rhe underlying niultiprogrartunittg ot the CPU or ihe relative speeds or different processes The difference between a process, and a program is subtle, but crucial An analogy m<ikc help here. Consider a culinary-minded computer scientist who is baking a birthday cake ror hi.s daughter He has a birthday cake recipe and a kitchen well stocked with all the input: flour, eggs, sugar, extract of vanilla, and so un In this analogy, the recipe is the program (i e., an algorithm expressed in some suitable notation), the computer scientist is the processor (CPU), and the cake ingredients are the input data. The process is the activity consisting of our baker reading the recipe, fetching the ingredients, and baking the cake. Now imagine that the computer scientist's son comes running in crying, say- saying that he has been stung by a bee. The computer scientist records where he was in the recipe (the state of the current process is saved), gets out a first aid book, and fregins following the directions in it Here we see the processor freihg switched from one process (fraking) to a higher-priority process (administeniig medical care), each having a different program (recipe versus first aid frook) When the bee sting has been taken care ot, the computer scientist goes back to his cake, continuing at the point where he left off The key tdea here i.s that a process is an activity of some kind It has a pro-program, input, output, and a state. A single processor may be shared among several processes, with some scheduling algorithm being used to determine when to stop work on one process and service a different one 2.1.2 Process Creation Operating systems m.-ed some way to make sure ail ihe necessary processes exist. In very simple systems, or in systems designed for running only a single application (e.g., the controller in a microwave oven), it may be po-ssifrle to have all the processes that will ever be needed be presem when the system comes up. In general-purpose systems, howevei, some way is needed to create and terminate processes as needed during operation. We will now look at some of the issues There are four principal events that cause processes to he created 1. System initialization 2. Execution of a process creation system call fry :i running process 3 A user request to create a new

process. 4 Initiation of abatchJoh. When an operating system is booted, typically several processes are created. Some of these are foreground processes, that is, processes that interact with

74 PROCESSES ANDTHKbADS CH4.P 2 (human) users and perform work for tliem Others are background processes, which are not associated with particular users, but instead have some specific lunctton. For example, one background process may be designed to accept incoming email, sleeping most of the day bui suddenly springing to lire when email arrives. Another background process may be designed to accept incoming 10 service the request. Processes th;it stay in the background to handle some activity such as email. Web pages, news, printing, and so on are called daemons Large systems commonly have dozens of them In UNIX, the /?< program can be use(i to'list the running processes In Windows 95/98/Me. typing CTRL-ALT- DEL once shows what's running. In Windows 2000, the task manager is used. In addition to the processes created at boot time, new processes can be created afterward as well Often a mnning process will issue system calb to cruute one or mote new processes tn help it do its job. Creating new processes is particular!} useful wl,en the work to be done can easily be formulated to terms of se\cml related, but otherwise independent interacting piocesses. 1-or example, it a laige amount or data is being fetched u\er a network t»r subsequent processing, it tnay butter while a second process removes the data items and processes them On a multiprocessor, ;illow,ny each process to fuu on it dillcicnt CPU m.iy also makc- the job go faster. In interactive systems, users tan start a progiani by lyptng a command or (double) clicking an icon. Taking either ot these actions starts a new process and runs the selected program in it. In command-based UNIX systems running X Win- Windows, the new process takes over the window m v.hnh n was started. In Micro- Microsoft Windows, when a process is started it do*>*. not hus'c n window, but it can create one (or more) and most do In hoth *.>steml>- users may h.i\e multiple win- windows open at once, each running some proems Using the mouse, ilie user tan needed The last situation in which protesse-, tire created applies only to ilic hJich sys- systems found on large mainframes. Hen- users can submit baich jobs to the system (possibly remotely). When the opciating svstem docidfcs th.it U has the resources tt) run another joh, it cieatcs a new pmcess iind runs the next job trijiii the in[)Ut queue in it. Technically, in all these cases, a new process is created b\ having an existing process execute a process creation

system call That process may he a itinning user process, a system process imolvcd from the keyboard or mouse, or a batch manager process What that process does]S execute a system call to create ihe new process Tin-, system call tells the operating system to create a sew process and indicates, dimctly oi indirectly, which progiam to iiin in it In UNjX. there is only one system call to create a new process, fork. Ihi.s call creates an exact clone oi the calling process. After the fork, the two processes, the

SEC 2 1 PROCESSES 75 parent and the child, have the same memory image, the same environment strings, and the same open fires That is all there is Usually, the child process then exe- executes execve or a similar system call to change its memory image and run a new program For example, when a u.ser types a command, say. sort, to the shell, the sheH forks off a child process and the child executes sort The reason for this two-step process is to allow, the child to manipulate its file descriptors after the- fork but before the execve to accomplish redirection of standard input, standard output, and standard error. In Windows, in contrast, a single Win32 function call. Create Process, han- handles both process creation and loading the correct program into the new process. This call has 10 parameicrs. which include the program to be executed, the com- command line parameters to teed th.it program, various security attributes, hits that control whether open files are inherited, prioritv mtormanon. a specification of the window to he created for the process (if any), and a pointer to a structure in which information about the newly created process is returned to the caller In addition to CreateProce5S. Win32 has about 100 other functions tor managing and synchronizing processes and related topics. In both UNIX and Windows, after a process is created, both the parent dud child have their own distinct dddi*ess spaces If either process changes a word in]ts address, space, the cliangt is not visible to the other process. In UNIX, the child's initial address space is a copy of the parent's, but there are two distinct address spaces involved; no writable memorj is .shared (some UNIX implementa- implementations share the program texj between the two since thai cannot be modified) 1) is. however, possible for a newly created process to share- some of it1- creator's Other resources, such as open tiles In Windows, the pare-nt\ and child's addpes1- spaces arc different from the start. 2.1.3 Process Termination After a process has been created, it starts running and does whatever its job js Howc-ver. nothing lasts forever, not even processes. Sooner or later the new proc- process will terminate, usually due to

one of the following conditions 1. Normal exit (voluntary). 2. Error exit (voluntary! 3. Fatai error (involuntary). 4. Killed by another process (involuntary) Most processes terminate because they have done their ivork When ii com- compiler has compiled the program given to it. the compiler eteeutes a system calf lo teil the operating system that tt is finished. This call is exit tn UNIX and ExitPro- cess tn Windows. Screen-oriented programs afso support voluntary termination.

76 PROCESSES AND THRFADS CHAP 2 Word processor., internet browsers and similar programs always liave an icon or menu item ihat the user can click to tell the hioeess to rtmove .my tc-mpoiary tiles it lias open and then terminate. The second reason ior lermination is th.it the process disaivws a fatal emir. For example, if a user types ihc command cc too c to coinpile the program ftio (and no such tile exists, the compilei simply exits. Screen-oriented interactive processes generally do not exit when given had parameters. Instead ihey pop up a dialog hox and ask the user to try .igain The third reason for termination is an error caused hy the process, often due to a program bug. Kxamples include executing an illegal instruction, referencing nonexistent memory, or dividing by zero- In some sysiems (e g . UNIX), a process can tell the operating system that it wishes to handle certain errors itself, in which case the process is signaled (interrupted) instead of terminated when one of the errors occurs. The fourth reason ;i process might terminate is that a process executes a sys- system call telling the operating system to kill some oiher process In UNIX 'his call is kill. The corresponding Win32 function is Terminate Process. In both cases, the killer must have the necessary authorization to do in the killee In some systems, when a process terminates, either voluntarily or otheiwise, .ill processes it c-reated are immediately killed as well. Neither UNIX nor Windows works this w.iy. how- 2.1.4 process Hierarchies In some systems, when a process creates anathoi process, the p<iieiit piocess and child process continue to be associated in certain ways. The child piocess can itself create more processes, forming a process hierarchy. Note that unlike plants and animals that use sexual reproduction, A process has onK one parent (but zero, one. two, or more children) In UNIX, a process, and all of its children and further descendants together form a process group When a user sends ,i signal from iho keyboard, ths.- signal is delivered to all memburs of |he process gmup currently associated with the key-keyboard (usually all acnvc processes thai were created in the current

window) Indi- Individually, each process can catch the mgnai. ignore the sign.it. or take the default action, which js to be killed by ihe signal As another example of where the process hierarchy pljjs a n'le, let us look at how UNIX initializes iiself when il is started A special process, called inn, is present in the boot image- When it starts running, it reads a file telling h(iw man) terminals there are. Then it forks off one n^w piocess pei terminal. These proc- processes wait for someone to log in, If a login is successful, the login process exe- executes a shell to accepl commands These commands may start up more processes.

SEC. 2 I PROCFSSES 77 and so forth Thus, alt the processes in the whole system belong to a single tree, with ma at th« root. In contrast, Windows does not have any concept ol a process hierarchy. All processes art equal. The only place whe.c- there is something like a process hieraichv is that when a process is created, the parenl is given a special ioken (called a handle) that it can use to control the- child However, it is free to pa^s this token to sonic Other pioccss, thus invalidating the hierarchy. Procc-sses in UNIX cannot disinherit theii children 2.1.5 Process States Although each process is an independent entity, with its own program counter and Lnlernal state, processes oflen need to interact with other processes One process may generate some output th;u anolher process uses as input In tht shell command cat chapieM chapter2 chapter3 I grep Iree ihe first process, running cuk concatenates and outputs three tiles. The second process- running grep, selects all lines containing the word "Irec " Depending on Ihe relative speeds of the two processes (which depends on both ihe relative com- complexity of the programs and how much CPU lime each one has had), it may hap- happen that grep is ready to run, but there is no input waiting ror it It must th^n block until some input is available. When a process h|ocks. ,1 does .so because logically it cannot continue, typi- typically hecause it is waiting for input that is not yei availahle It is also possihle for a process that is conceptually ready and able to run to be stopped because Hie operating system has decided to allocate the CPU to another process for a \vhile. These two conditions are completely different In the first case, the suspension is inherent in the problem (you cannot process the user's command line until il h,is been typed). In the second case, it is a technicality of the system (not enough CPUs to give each process its own private processor). In Ftg 2-2 we see a state diagram showing the three states a process may be in I Running (actually using |hc CPU at that instant). 2. Ready (runnable;

temporarily stopped to lei another process run) 3. Blocked (unable to run until some external event happens). Logically, the first two stales are similar In both cases the- process- is willing to run. only in ihe second one. there is temporarily no CPU available tor it The third state]s different from the first two in that the process cannot run. even if the CPU has nolhing else to do Four transitions are possihle among theso |hree states, as shown Transition 1 occurs when a process discovers that it cannot conOnue. In some systems the

PROCLSibS AND THRbADS 2 Scheduler picks another pi 3 Scheduler p>eks this proce 4 input becomes, dvailabie Mgure2-2. process niuM execute a system call, such as block or pause, to get mm blocked ■-t.ite In uthcr systems, including l.NIX. when a process re;ids tiom a pipe oi spe- special file (e.g.. a terminal) and there is no input available. |he process is automati- automatically blocked. Transiiions 2 and 3 ore caused by the process scheduler. ;i part til ihe oper.it- ing system, without the process even knowing about them Transition 2 occurs when fht^ scheduler decides tluit the running process has run long enough. ,ind 't is time to let another process h;ive some CPU time Transition ~l occurs when all the other processes have had then fair share- and it is tune- for the first piocess U> get the CPU to run again. The subject of scheduling, that is, deciding which process should run when and tor how long, is an impoitant one, »o will look at it later in this chapter. Many algorithms have been devised to tiy to balame the competing demands or efficiency for the system as .i whole and forties's to individual processes. We will study some of them tatc-r in this chapter Transition 4 occurs when the external event for which a process was waiting (sue-h as the arrival of some input) happens If no othei process ii> running at that instant, transitton 3 will be triggered and the process will start running. Otherwise it may have to wait in ready state for a little while until the CPU is available and r to think about what is going Using the process model, it becomes m mands typed in by a usei Other processes are part of the system and liaudb tasks such as carrying out requests for file services or managing the details of running a disk oi a tape drive When a disk interrupt occurs, the system makes a decision to stop running the current process and run the disk process, which was blocked waiting for that interrupt, Thu.s. iiistead or ihinking about interrupts, we can thijik about user processes,, duk processes, terminal processes, and so on, winch block when they are waiting kir something to happen Whc-n the disk has btcn scad oi the character typed, the process waiting foi it is unblocked ,ind is eligible to nm again. This view

gives rise to the' modcf shown in Fig 2-3. Here the lowest fevd or the operating system is tlie scheduler, with a variety of piocessos uii tiip oi it All ihe interrupt handling and details of actually starting and stopping pioces.ses iire hidden away in what is here called the scheduler, which i.s actually not much

:ode The rest of the opt-rat] real svitems are as mcetv sta y system is utcel} tured as this, how ctured m process foim f-e 2.1.6 Implementation of Processes To implement the process modtl. the operating ss">tem maintains a tablt (<jn array of structures), called the process table, with one entry per process (Some authors call these entries process control blocks.) This ent]y contains informa- information about the process' state, its program counter, staci pointer, memory alloca- allocation, the status: of its open fifes, its accounting and .scheduling information, and everything else about the process that must be saved when tht- piocess is switched from running to ready or blocked state so that it can be rcstai ted later as if it had never been stopped. Figure 2-4 shows some of the more important f]e(ds in a typicaf system The ficld-s in the first column reiatc to process management The other two columns relate to memory management and fife management, respectively It .should be noted that precisely which fieids the process tabie has is highly .lystem dependent, but thi.s figure gives a general idea of the kinds of information needed. Now that we have looked at the process table, it]s possibic to explain a little more about how [he illusion of muitipie sc-quentiai processes is maintained on a machine with one CPU and many I/O device Associated with L-ach I/O device class (e.g.. floppy disk.s, hard disks, tuners, [emunafsl K a location (often near the bottom of memory) called the interrupt vector It contains the address of the interrupt scnice proceduie. Suppose that user proeev. 3 k running when ti d]sk interrupt occurs. User pioeess Vs progiam counter, program ■-tarns word, and possJl specified in the disk interrupt on. it is up to the software, in hardware. The computer then jumps to the vector. That is all the hardware does. Frui particular, the interrupt service procedure Ail interrupts start by saving the regime]-,, often in the pioces.s table cmry fof the current process. Then the information pushed onto the stack by the interrupt is removed and the stack poinier is set to point to a [emporary .stack used by ihv-

PRCX'hSSES AND THRFADS 1 Process management Registers Program status word Stack pointer Memory management Poinlertolexlsegmenl Pointer

to flala segment Pointer to stack segment j File management Root direclory ■ Working direciory j File descriptors ', UserJD I Group ID Parent process . Process group i S,gnais \ Time when process slarted j CPU lime used Children's CPU time- Time of nexl alarm i Figure 24. Sonic of (he JieliK tu ,1 m>n,il pt.«-ess ubJc eitl,\ process handler. Actions suth as saving rhc legislers and setting the sUck pojnier cannol even be expressed in high-level languages such ;\s C. ■>(> they are per- performed by a small a.ssembfy [anguage routine-, usually the siinie cme tor alf inrer- rupts since ihe work of saving ihe registers is identical- no niatrer whal the uiusc- of ihe inierrupi is. When (his routine is fiiiished.il calls .1 C piccediire (tide, the rci ol rhc- uork for this specific inierrupi lype. (We assume ihe operating system i-. v,rirten in C. ihe usual choice for all real operating sysltms) When it has done its [oh. posMbly making some process now ready. Ihe seheduier is called kt see \^ho to iuii next Afler thai, coniroi is passed back 10 Ihe assembly language code it, litfid up the registers and memory map for ihe now-uirrenr process and stair rt iunnUJg Intci- rupi handdng and scheduling are sumiuarized in l-ig 2-5. it ^ woiih ncitmg that ihe deiaiis vary somewhat from sy^lem tci svstem. n~Hardware slacks program counrer etc i 2. Hardware loads new program counter from ffiterruprveclor. 3. Assembly language procedure saves registers 4 Assembly language procedure sels up new stack 5 C imercupt service runs (typically reads and buffers mput) 6 Scheduler decides which process is id run next 7. C procedure returns lo ihe assembly code. I 8. Assembly language procedure starts up new current process

SEC 2] PROCESSES ol 2.2 THREADS In traditional opening systems, each process has an address space and a sin- single thread of control. In fact, [hat is almost the definition of a process. Neverthe- Nevertheless, there are frequently situations in whicli it is desirable Ki have multiple llia-ad1* ot control in the same address space running in t|iiasJ-parallel, as though they were separate processes (except for the- shared address space) fn the follow- following section's we will discuss ihese situations and their implk ahons 2.2.1 The Thread Model The process model as we havi.- discussed it thus tar is based on two indepen- independent concepts' resource grouping and execution Sometimes it e» useful to separate them, this is where threads come in. one way of looking at .i process is that il is way to group reiated resources together. A process has an address space containing program ie*i and data, as well as other resources These resource may inciude open fifes, child

processes, pending alarms, signal handlers, accounting information, and more B^ pulling them together in (he foim of a piocess. they can be managed more eastJy The other concept a process has is a thread of execution, usually shortened in just thread. The thread has a program counter ih.il keeps irack ot which jnstiuc- has a slack, which contains ihe execution hisiury. with une Ininie lor each pro^ cedure called but noi yet returned from Although a thread must execute in some process, the thread and its procc-s.s art.- different concept's ,ind can be treated separately. Processes are used 10 group resources together, threads are Ihe entities scheduled for execution on the CPU. Wbat threads, add ki the process model is lo allow multiple executions to t.iku place in ihe same process environment, lo a large degree independent of sine another. Having muliiple threads running m parallel in one process is analognus Ki having muitiple processes- running m parallel m one computer, in the former case, the threads share an address space, open files, and olher lesources in the latter case, processes share physical memory, disks, printer-, and iit her resources Because threads have some of ihe properties of processc. they are ■.oinetimes called lightweight processes. The lerm multithreading is also used to describe the situation of allowing multiple threads m the same process. hi Ftg. 2-6(a) we see three traditional processes. Each process has its own address space and a single thread of control in contrast, in Fig 2-6tb) we see a single process with ihree thn;ad« of control Although m both cases we-havc-lhree threads, in Fig. 2-6(a) e.ich of them operates m a different address space, when-as in Fig. 2-6(b) all three nf them share ihe same address space When a multithreaded process is run on a stngfc-CPU system, the ihread.s take turns running. In Fig. 2-l.we saw how multiprogramming of processes works.

PROCESSES ANDIHKMtiS By switching back and forth among muiltple processes, the system £ives the illu- illusion of separate sequential processes running in paraiiei. Muffithreading works [he same way. The CPU switches rapidly back and forth among ihe threads pro- providing the iFiuk-ion Ihai ihe threads are running in parallel, albeit on a slower CPU than the real one. With three com pule-bound threads in a process, the threads would appear to be running in parallel, each one on a CPU with one-third the speed of the reai CPU. Different threads in a process are npt quite as independent as dilferent processes. All threads have exactly the same address space, which means that they aiso share the same global variables Since every thread can access every memory address within the process' address space, one thread can read, write.

<* even completeiy wipe out another thread's stack. Thcie is no protection between threads because (I) it ts impossible, and B) t(should not bo necessary Unlike different processes, which may he from different users and which may be hostile to one another, a process is always owned bv a single user, who has presumably crealed multiple threads so that they can corporate, not fight. In addition to shar- sharing an address space, all the thieads share lite same set of open files, child processes, alarms, and signals, etc. as shown in Fig 2-7 Thus the urganuatton of Fig. 2-6(a) would he ust-d when the three processes are essentially unrelated, whereas Fig. 2-6(b) would be appropriate when the three threads arc actually part of the same jdh and are actively and closely cooperating with caeh othci The items m the first column are process properties, nut thread properties For example, if one thread opens a tile, that file i> \isihle to the other threads in the process and they can read and write it. I lus is logical since the process is tbe Unit of resource management, not the thread If each thread had its own address space, open files, pending alarms, and so on. it would be a separate proces-s. What we are trying to achieve with the thread concept ls the ability for multiple threads

per process items Per thread items I Address space Program coumer i Giobai variables t Registers Open Ides Stack Child processes State Signals and signal handlers ! Accounltng information __ of execution to share j sei of resources so thuy can wort, together etoseU to per- perform some usk t.tke a traditional process (t e . a protest W]th only om- thread), a ihre^d can be in any one of several stales running, hlockod, re.idy, or terminated A iuni)mj3 thread currently has the CPU and is acti\e A blocked thread is waiting lor some event to unblock il- For example, when a thread performs a system call to rtdd from the keyboard, it K blocked until input is typed A thre.it! can hlock waiting for some exiemal event lo happen or for some other thread to utihlock il. A re.tdj ihread is scheduled |o run and will as soon as its tuni cornes up The transitions between ihread states are Ihe same as the transitions between process states .incl are illustrated in Fig. 2-2. Il is important to realize that each thread 1ms tts own stack, as shown in Fig. 2-8. Each thread's stack contains one frame lor each procedure called bui not yet relumed from. This frame contains the procedure's local variables and ihc- reium address to use when the procedure call has finished. For example, if pro- procedure X calls procedure Y and this, one calls procedure Z. while Z is executing Ihe frames for X, Yr and Z will nil be on the slack Fach ihread will generally call dif- different procedures and a

ihus a different execution history. This is why :s ihread needs Us own stack. When multithreading is present, processes normally start wilh a single thread preseni. This thread has the ability to create new threads hy calling a library pro- procedure, for example, ihread-creaie. A parameter to thread—('reate lypically specifies the name of a procedure for the new thread lo run, It is noi necessary (or even possible) to specify anything about ihe new thread's address space since il automatically runs in Ihe- address space of the crealtug tl,rcad. Sometimes threads are hierarchical, with a parent-child relabonshtp. but often no such relationship exists, wjih all ihreads being equal. With or without a hietarchical rehnionship, the creating thread is usually returned a thtead identifier thai n.unes ihe n.uw thread. When a ihread has finished it-, work, il i:an exit by cailing a (ibrary procedure, say, thread_e\it d then vanishes and is no longer scheduiable. in some thread

PROCESSES AND THREADS Thread 2 Thread 3's slack one thread can v Mgure 2-8. Each ihread ha it for a (speci ihread 10 exit by caffing a procedure for exampFe. thread_wdtt. This proceduic blocks the caFFing thrt?«id unil 1 a (specific) ihread has exited In this regard, ihread creation and termination is veiy much (ike process creation and termination, wuh approximately the s.ime opiioru as weil. Another common ihread caff is thread_vie!tI, which arrows a ihread to voFuu- laniy give up the CPU lo (et another thread run Such a cafl is important because there is no ciock imerrupi lo actually enforce timesharing as ihere is wiih processes. Thus it is important for threads to be poftie and vofununiy surrender the CPU from time to time to give other threads a chance to run Other calls aiiow one thread to wail for another thread to finish some work, tor a ihread lo announce that it has finished some work, and so on. Whjfe threads are ofien usefuF, ihey also introduce a number of complications into ihe programming model To start with, consider the effects of the l'N(X fork system ca((. (f the parent process has mu(tip(e threads, should the child aisn have them" if noi. the process may noi function properly, since al(of (hem niay be essential. However, if the child process gets as many threads as the parent, what hap- happens if a ihread in the parent was bfocked on a read ca((, say. from the keyboard'* Are two threads now hiocked on the keyboard, one m the parent and otic in the- child? When a (me is typed, do both ihieads get a copy o(ii1 On(y ihe parent' Oniy the ctuid^ The same problem exists wiih open network connections. Another cfass of problems is, refated lo the fact ihat threads share many dat.i structures. What happens if one ihread cfoses a riic

whife another one is sliff read- reading from u? Suppose that one ihread nonces thai there is too Iniie memory anii starts aiiocaimg more memory. Part way through, a thread swticd occurs, and the new ihread aiso notices lhal there is too [iiiic memory and also siarts ailocaling

more memory. Memory will probybly be aflocated twice These problems can be solved with some errori, bui carefuf ih<xighl and design are needed lo make muf- Ulhreaded programs work correctly. 2.2,2 Thread Usage Having described what threads are. n is now lime io expfain wh\ in\one wjnls them The nyin reason for having threads is lhai m many appiicaiions. mnllipie aeiivines are going on ai onct;. Some of thest. ran block fiom time io lime By decomposing such an applicauon into muiupie sequential threads thai run in quasi-parallel, ihe programming mode! hecon ls sin pier We liavfc seen ihis argument herore. (t is preciseiy ihe argument tor having processes-. Instead of thmking aboui imerrupis. timers, and contexi switches, we can think aooui parallel processes. Only now wilh threads we add a new clement: the ability for the parallel eniiliei to share an address space and all or its data among themselves. This' ahility is essential for certain applications, which is why having multiple processes (with their separate address spaces) wjll not work resource*; attached to iheni. ihey are easier lo crealc and dcsiroy than processes. In inanj sjsleim. creating a thread goes 11H limes taster than wealing a process When the number of threads needed changes dynamical I> and rapidly, ihis pro- property is useful. A third reason for having ihreads is also a performance argument. Threads yield no performance gam when aii of ihem are CPU hound, bui when there is siihs-ianual computing and also substantial 1/CX having threads allows these activi- activities 10 overlap, thus speeding up the application Finally, threads are useful on sysicms with multiple CPUs- where real paral- parallelism is possible. We will come back lo [his issue in Chap 8. It is probably easiest to see why threads are useful by giving some concrete examples. As a first example, consider a word processor. Most word processors display the document being created on the screen formatied exactly as it will appear on the printed page. In particular, all the line breaks and pdge breaks are in their correct and final position so the user can inspect them and change the document if need be (e g . io eliminate widows and orphans —incomplete lop ,ind bottom line-s on a page, which are considered es,theticully unpleasing) Suppose that [be uslt is writing a book From Iht; author's point of view, it is easiest to keep the entire book as a single

file to inakt- it easier to search lor topics, perform global substitutions, and so on Alternatively, each chapter niighi be a separate file Howevei, having every sec-tion aiid subsection as a separate file is a real nuisance when globul changes have* io he rtia.de to the entire book since then hundreds of tiles have [o be individually edited For example, if pro- proposed standard xxxx is approved just beiore ihe book goes [o press, all occurrences of ■'Draft Standard xxxx" have to be changed [o "Siandard xxxx"" at

86 PROCESSES AND THREADS CHAP 2 ihe lasi minuie. If ihe entire book is one Tile, lypically a single command can do ail the ^ubsiiiutions In contrast, if the book is spread over 10A hies, each One inust be edited separately Now consider what happens when ihe user stiddeiilj deletes one seniciice from page I of an 800-page documem. Alier checking ihe changed page- to make sure i| is correci, the user now wants lo make ;ino|hir change oil page 6(K) and lypes in a command telling the word proce-soi lo go to thai page (possibly by searching for a phrase occurring only there). The word processor is now forced io reformai ihe enure book up to page 600 on ihe spoi because it does mil know what ihe first line of page 600 will be unul il has processed all lite previous pages There may be a substantial delay before page 600 can lie displayed, leading lo an unhappy u^er Threads can help here Suppose thai the word processor i. wniien as a two- ihreaded program. One Ihjead interacts wilh the usti jnd ihe oiher handles ivlbr- malling in the background As soon as ihe sentence is deletc-d Irtini page I. |he interaciive ihread tells the re format] ing thread to reformat the whule hook Meanwhile, ihe interactive ihread continues lo listen to ihe keyboard and iruiuse and responds io simple commands like scrolling page 1 while ihe nther thread i>. compuiing madly in the background. Wilh a Iiule luck, the reformatting will he completed before ihe user isks lo see pige 6A0. so ii can be displayed instanily. While we are ai u wh\ noi add a third ihread" Many word processor's have a feature of automaiicalK saving Ihe enure file to disk every few minutes lo pioieci the user against losing a da\ s work in ihe event nl a program crash, sysiem crush, or power failure. The third ihre id t,w handle ihe disk backups without interfering with ihe oiher iwo. The situation with three threads is shown in Fig. 2-9 Figure i-9. \ word proccs

SFC 2 2 TIIRbADS 87 A tht program were single-threaded, then whenever a drst. backup started, commands from the keyboard and mouM- would be

ignored until the backup was finished The user would perceive this as sluggish performance. Alicrnatnelv. keyboard and mouse events couid interrupt the disk backup, allowing good perfor- performance but leading to a complex interrupt-driven programming model. With three threads, ihc programming model k much simpler. The first thread just mtoracb with il,c user The second thread reformais the document when told tu. The third thread writes the conients of RAM to disk periodically It should be clear that having three separate processes would not work here because all three threads need to operate on the docunienl By having thiee threads instead of three processes, they share a common memory and thus all have access to the documeni being edited. An analogous srtuadon exists with many other rnternctive program-!. For example, an electronic spreadsheet is a program thai allows .i ttser to maintain a matrix, some of whose elements are data provided by the user. Other elements are compuied hased on the input data using potentially complex formulas When a user changes one elen>ent, many other elements may have to be recomputed By having a background thread do the recomputjtion, the interactive thread can allow the user to make additional changes while the cotnpuiation is going on. Similarly, a third thread can handle periodic backups to disk on its own Now consider yet another example of where threads are useful- a server fur a World Wide Web site. Requests tor pages come in and tht requested page is sent back to the client. At most Web Sites, some pages are more commonly accessed than other pages For example. Sony's home page is accessed far more than a paye deep in the tree containing ihe technical specifications of some particular camcorder. Web servers use this fact to improve fierfoi mance by maintaining a collection of heavily used pages, in main memory to eliminate the need to go to disk to get them. Such a collection is called a cache and is used in many other context-, as well. Om way to organize the Web server is shovMf id F-tg. 2-10(a) Hore i.ne thread, the dispatcher, reads incoming requests foi wnrk from the network Alter examining the requesr. ir chooses an idle (i e . blocked) worker thread uiid hands it the request, possibly by writing a pointer to the message into ,i special word associated with each thread. The dispatcher then wakes up the sleeping worker, moving it from blocked state- to ready state When the worker wakes up. it checks to set: if the request can he satisfied from the Web page cache, to which all threads have access li not, it starts a read operation to gel the page trom the- disk and blocks until the disk operation com- completes. When tnt thread hlocks on the disk operation, another thread is chosen to run. possibly the dispatcher, in order to acquire- more work. 01

possibly another worker that is now ready to i un This model allows ihe server to be written a^ a collection ot sequential threads The dispatcher's progrnm consists of an infinite loop ior getting i± work

PROCESSES AND THREADS Web server process Figure 2-10. A multuhrcjded Wch server request and handing it ofl to a worker Each worker's aide consists of an infinite loop consisting of accepting a request from ihe dispatcher and checking the Web cache to see if the page is presenr. If so. il is relumed 10 the client and the workei blocks wailing for a new request. If nor. it gels the page- from ihe disk, rciutns n to the client, and blocks waiting for a new request. A rough outline of the code is given in Fig 2-1 i. Here. ,\s in ihe rest ul rhis book. TRUE is assumed to be the constant I Also, bu/ and page are structures appropriate for holding a work request and a Wtb page, respectively. while (TRUE) { get_next request(Sbuf); handoff_work(&buf), while (TRUE) { wat tor work(Sbut) lookjor_page_m_cache(&buf, Spagej, if (page. not_m cache(Spage)) read pagejrom disk(Sbuf, Spagej, return page(Spage);) (b) Consider how the Web serv possibility is to have it operat server gels a request t\ imine the next one While waiting for the disk, uld be written in the absence oi threads. One a single thread. The mum loop of the Wch nd l imes it out to completion hefore getting idle and does not process g i d any other incoming requests. If the Web server is running dedicated

SEC 2 2 THREADS machine, as u commonly ihe ease, the CPU is simply idle white ihe Web server is waiting tor the disk The net result is that many tewcr requests/scc can be pro- processed Thus threads gain considerable performance, but each thread is pro- programmed sequentially, in the usual way So far we ba\e seen two possible designs: a multithreaded Web server and a Mnglc-threaded Web server Suppose that threads are not available but the system designers hnd the performance loss due to single threading unacceptable If a ncfUblockmg version of the read system call is available, a third approach is possi- possible When a request comes in. the one and only thread examines it. If it can be satisfied fmm the cache-, fine, hut if not. a nonblocking disk operation is started. The ser\er records the state of Ihe current request in a table and iben goes and <-Tcts the next event. Tbe next event tiidy cither be a rtxjuest for new work or d icply from the disk about u previous operation If it is new work, tbat work is started. If it is a reply from ihe disk, the relevant information is fetched from the table and the reply

processed. With nonbloekmg disk f/O. a reply probably will have to take the form ot a signal or interrupt In this design, ihe "sequential process" model thai we had in the first two. cases is lost Tbe stale i|f Ibe computation must be explicitly saved and reared in tbe table every time the server switches Irom working on one request tu another. In effect, we are .simuialing Ibe threads and Iheir stacks Ibe hard way. A design tike tbis in which each computation has a saved stale and there exists some set of events that can occur lo change Ibe slate is catted a finite-state machine. This concept is widely u>>ed tbrougboui computer science It sboutd now be clear what threads have In otter. They niake il possible tij relam Ibe idea of sequenliai processes that make biockmg system cails (e.g . for disk I/O) and .still achieve parallelism. Blocking system calls make programming easier and parallelism improves performance Tbe single-threaded server retains the ease of blocking system calls but gjve-s up performance. Tbe tbinf approach achieves, high performance through parallelism hut uses iionblocking calls and interrupts, and is thus is hard to program These models arc summarized in Fig. 2-12. j Model Threads i_S|ngle-threaded procet Characteristics __ Parallelism blocking system c No parallelism, blocking systei Parallelism, nonblocking syste Figure 2-12. Three ways lo const ma a sener A Ibird example where Ibreads are useful is in applicat very farge amounts (if data. The normal approach is to re process it. and then write it out again The problem here i: system calls are available, Ihe process blocks while data a that must process :n a biock of data, at if only blocking ommg in and data

 : are goinjr out. Having the CPU go idle when there is kits <>1 Jttmpuimji to An is iearly wasteful and should be aroided if possible. Threads offer a solution. The process amid be structuietf with an inpui [bread, a processing thread, and an output thread The input thread icads data into in input buffer. The processing thread takes data out of the m|)u! buffer, processes them, and puts the results in an outpui bufttr. The output buffer writes these results back to disk In tht,s way. input, output, and processing can att dc going on at the same iime. Of course. lh>s model only v»orks it a system call blocks only the calling thread, not the entire process. 2.2.3 Implementing Threads in User Space There are two main ways to implement a threads package in user Space and in the kernel. The choice is moderately controversial, and a hybrid nnplenienta- iion is also possible. We will now describe these methods, along with their advantages and tii sad van! ages The

first method is to pui the threads package entirely in user space The ker- kernel knows nothing about them As far as the kernel is eonceriied. it is managing ordinary, single-threaded processes. The tirst. and most obvious, advantage is that a user-level threads package can be implemented on an operating system thai does not support threads All operating systems used to fall mio ilns category, and even now some still do. All ot these implementations have ihe same nenoriLl siriicture, wh]ch is illus- illustrated in Fig. 2-13(a). The ihreads run on iop ot a run-time system, winch is a col- collection of procedures thai manage ihreads We Iuvl- seen ftiur <jl" these already thread ^create. thread_e\tt, thrcad_wiiit, and thread, vteid, hui usually ihert are more When threads are managed in user space, each process needs its (>wn private thread table to keep track ot ihe ihreads in thai process. This iable k [in.ilonous io the kernel's process table, except thai it keeps track i>nly i|V the per-thread pro- properties such the eac-h thread's program counter, siack pointer, registers. -,tale. etc. The iliread table is managed by ihe run-time system When a thread is moved tu ready state or hlocked state, the information needed to restart it is stored in the thread table, exactly the s,ime way as ilic- kernel stores in Turin ati an ,,hout processes in ihe process iable. Wl,en a thread docs something thai mai cause ti to becnnie blocked locally, lor example, waiting hir another thread in its prt,cess io complete smnv w.uk. <! calls a run-lime sysiem procedure. This procedure checks M see il the threat! must he put into hlocked stale. If so. it siorc ihe thread's registers (i e.. its own) in the thread table, looks m the table for a icaJj thread io run. and reloads the machine registers w-ith ihe new thread's sav'td values As soon as the siack pointer and program counter have been switched, fhe new thread comes io fire

Process Thread / / \ / another one fo load them all- (he entire thread switch ction Doin an be done m a handful t)f ast an order of magnitude g faster lhan trapping to the kerne] and is a strong argument in favor oj user-kvel threads packages However, (here k one key difference with processes. When a thread is fin- finished running for the moment, tor example, vihen it talk thread_weld, the code Of thread-\-ield can save the thread's information in the thread table itself f-urth- eniiore. it can then calf the thread scheduler fo pick another ihreaO to inn. The procedure thaf saves the thread's state and the scheduler are just local procedures, so invoking them is much more efficient lhan making a kernel call. Among other issues, no trap is needed, no context switch is needed, fhe memory cache need not be flushed, and so on. This makes thread

scheduling very last. User-level threads also have other advantages. They allow each process to have its own customised scheduling algorithm For some applications, for exam- example, those with a garbage collector thread, not having to worry about a thread being stopped at an inconvenient moment is a plus. Thev also scale better, since kernel threads invariably require borne table spact; und stack ;,pace in the kernel, which can be a problem if there are a very large number of threads Despite then better performance. User-level threads packages have some major problems. First among these is the probk-m of how blocking system calls are implemented. Suppose that a thread reads trom ilie koytxyird before any keys have been bit. Letting the tliread actually make the system call is unacceptable, since this will stop all the threads One ot the main goals of having threads in the first place was to allow each one Jo use blocking calls, but to prevent one hlocked

92 PROCESSES AND TMRt;ADS CHAP 2 (bread from affecting the others. With blocking system calls, it is bard to sec how this goal can be achieved readily. The system calls could all be changed lo be nonlocking (e.g.. a read on the keyboard would just return 0 byles if no etuiacters weie already buttered), but requiring changes lo the operating system k unattractive. Besides, one of the arguments for usei-Jevel threads was piecisely that they could run with existing operating systems In addition, changing the semantics of lead will require clinnges lo many usei programs. Another alternative is passible in the" even! thai it is possible lo tell in advance allows Ibe culler lo lell wbeiber a prospeclivc read will bliick. When this call is present- Ibe library procedure read can be replaced wilb a new one Ibat firs! does a select call and then only does Ibt read call if il is safe (i e, will nol block) ll Ibe read call will block, tbe call is nol made Inslcad, anolber thread is iun The next time Ibe run-lime syslem get* control. i(can check again lo see it (be read is now safe This approach requires iewnlin£ parts of the system call library, is inefficient and inelegant, but (here is liitle choice The code placed around (hi- Somewbal analogous (o (be problem of blocking system calls n, the problem of page taulls We will siudy (hese in Chap 4 1 or the moment, it is sufficient to say that computers can be set up in such a wjj thai not all of tbe program is m main memory at once If tbe program calls (ir jumps to an instruction thai is no(m memory, a page fault occurs and the operating sy'i,tc-sn will go and get the- nnssinj! instruction (and its neighbors) rroni disk. Tins is called a page fault. The process is blocked while the necessary instruction is being located and

re;id in. If a thread cause- a page fault, the kernel, noi even knowing aboui ihe existence ol'ihreads naturally blocks the entire process until (he disk I/O is complete. e\en (hough olher Ihreads mighl be runnable. Another problem with user-level Ihread packages is that if a Ihread siarts run- running, no other (hread in (ha(process will e\ei run unluss Hit tiisl Ihread wilun lanly gives up Ihe CPU Wiihm a single process. Iht-ic arc no clock intomipis making i(impossible to schedule processes round-robm fashion dakma turns) Unless a (hread enters Ihe run-lime sysieni of us own irec will, ihe schcdulei will never get a chance. One possible solution lo the problem of threads running forever is to ha\e the run-hnie system request a clock sigi'a] (interrupt] once a second to give it contiol bul this, too, ts crude and messy to program. Periodic clock mtemipts at .i hiehci frequency arc not always possible, and even i]" they are. the total overhc-ad may be substantial Furthermore, a thread might also need a clock interrupt, interfering wtlh the run-time system"), use or the clock. Another, and probably tht most devaslatnig ai^umenl against user-level threads, is that programmers generally want thieads precisely in applications where the threads block often, as. for example, in a mullithmaded Web server

SFC 2 2 THRfcADS w These threads are constantly making syslcni calls. Once a trap has occurred to the w\ !■ jrk for the kernel to switch threads]]' the old one has blocked, and having the kernel do this eliminates Ihe need tor constantly making select sysiem calls lhat check to see if read syslem calls are safe. For applications that are essentially entiiely CPU bound and rarely block, what is the point or having threads at all" No one would seriously propose computing Ihe first » prime numbers or playing chess using threads because there is rioihuig to be gained bv doing il that way 2,2,4 Implementing Threads In the Kernel Now let us consoler having the kernel know- about and manage the threads No run-nme system is needed in each, as shown in Fig. 2-13(bl. Also, there is no thread table in each process Instead, the kernel has a thread table that keeps twek of all the threads in the system. When a thread wants to create a new tbre.id or destroy an existing thread, it makes a kernel will, which then does the creation or destruction hy updating the kernel thread table The kernel's thread tahle holds each thread's registers, state, and other infor- information. The information is the same us with user-level threads, but it ts now in the kernel instead of in user space (inside the run-time system). This information is a subset of Ihe information that traditional kernels maintain about each of theii Single-Ihreacted processes, that ts, the process state. In

addition, the kernel also maintains (he traditional process, table to keep track of processes. All calls lhat might block a thread are implemented as system calls, at consid- considerably greater cost than a tall to a run-lime system procedure. When a thread blocks. Ihe kernel, at its, opiion, can nin either another thread from the same proc- process (if one is ready), or a thread from a different process With user-level threads, the run-time system keeps running threads from its own process until the kernel takes the CPU away from it (or there are no ready threads left to run). Due to the relatively greater cost of creating and destroying threads in the ker- kernel, Some systems take an environmentally correct approach and recycle their threads. When a thread is destroyed, it is marked its not runnable. but its kernel data structures are not otherwise nffecteil. Later, whon a new thread must be created, an old ihread is reactivated, saving some overhead. Thread recycling ts also possible for user-level ihrends. but since the ihread management overbe,id is much smaller, Ihere is less incentive to do Ihis Kernel Ihreads do not require any new. nonlocking system calls In addition, if one Ihread in a process causes a pajjt; laull. the kernel can easily check lo see if Ihe pri>cess has any olher runnable threads, and if so, run one of them while wail- wailing for the required page lo be brought in from the disk. Their main dtsadvanlage is that the cos! of a syslem can is subslanlial, so if thread operations (crealion. ler- miniiiion. clc.) are common, much more overhead will be incurred.

94 PROCESSES AND THREAPS CHAP. 2 2.2.5 Hybrid Implementations Various, ways have been invesligaied to Iry to combine the advantages nf asM-lcvel Ihreads with kernel-level Ihreads One way is use kernel-level threads and then mulliplcx user-lev?] Ihreads onto some or all ot the kernel ihreads. as shown in rig. 2-14. Multiple Figure 2-14. Multiplexing (isvr-level IhrcatK unlo kerntl Il"m4 threap In this design, the kernel is aware of only Lhc kernel-level Ihu-ads and schedules those. Some of those Ihreads may have multiple user-level ihreads mul- liplexed on lop of them These user-level Ihreads are created, des!ri>ved. and scheduled jus! like user-level threads in a process that runs on an operating system wilhou! multithreading capability. In this model, each kernel-level thread has some se! of user-level threads thai lake iurns usiny H 2.2,6 Scheduler Activations Various researchers have altempted to combine Ihe advantage ot usei thieads igood performance) wiih Ihe advantage ot kernel threads (not having lo use .1 lo! of Iridc^ !o make things work) Below we svtU describe one such approacl) de\ ■ isedby Anderson et

al. A992). called scheduler activations Relaied woik is dis- discussed by Edleret al.(1988) and Scon ctal. 11990). The goals of the scheduler activation work are lo Inimit the funclionalny (it kernel threads, but with ihe beiter performance and greater flexibility usually associated with threads packages implemented in user space In particular. u«ui ihreads should noi have lo make .special nonbloc-kmg system calls or check id advance if n is safe i0 make certain system calls Nevertheless, when a thread blocks on a system call or on a page fault, n should be possible to run other threads within the same process, if any are ready

SCC 2 2 THR1-ADS ¥S Efficiency is achieved by avoiding unnecessary (mnsttiuns between user and kernel space. If a ihrcad blocks waiting tor another thread lo do something, for example, there is no reason to involve the kernel, ihus saving the- overhead of the kernel-user transition. The user-space run-time system cjii block ihe synchroniz- synchronizing thread and schedule a new one by itself When scheduler activations are used, ihe kernel assigns ;i certain number ot virtual processors to each process and lets the (usei-space(run-tune system allo- allocate threads to processors. This mechanism c-an :ilso be used on a multiprocessor where ihe virtual processors may be real CPUs. The number ot virtu.il processors allocated 10 a process is initially one, but the process c-an ask tor more and can also return processors it no longei needs The kernel cm also lake hack viriual processors already allocated in ordei 10 assign them lo other, more needy, processes. The basic idea that makes this scheme work is thai when the kernel knows thai a ihrcad has blocked (e.g.. by its having executed a blocking system call or caused ;i page tauli), the kernel notifies ihe process' run-nme sysielu. passing as parameters on the stack the number of the thread in question and a description ol the event that occurred The notification happens by liaMng the kernel activate the run-time system at a known starting address, roughly analogous to a signal m UNIX. This mechanism i« called ;m upcall Once activated like this the run-time s>stem tan reschedule its threads, typi- typically by marking the current thre.id a.. hlocked and (akutg another Ihread Irom Ihe ready list, setting up ils registers and icstaning il, Latci, when the kernc-l learns that the original thread can run again le g the ptpt- it was trying to lead trom now contains data, or the page it faulted ovei has been bronghl m from disk), it makes anotbci upcall to the rim-tinu s\siem to mfonn il al lb,s evenl The lun-lime sys- system, at its own discretion, can either reslai I the bilked Iba-ad jimncdialdv, ai put it on the ready list to be

mn laler When a baidware intenupi occurs while a user thread is running, ihe ntier- rupled CPU switches into kernel mode]| (be intcmipt is caused b\ jn even) not of mteiest to the interrupted process, sudt ,is w>iii]>letion til' .mnHier process' I/O. when the mtermpt handler has finished, il puts ihe intenupied tbieud back in the state it was in belore the interrupt. It. howevej, ihe piqeess is niteresied in tlie mtenupt, such as the <.\rnv:il of a page needed In one ol the process' threads the intfirupietl threat! is not restarted lnste.ul. tlie intcinipied ibrcsid is suspended and the run-time sysiein started t)ii that virtual CPl.\ with tbt state ol ilie inter- interrupted thread i>n the st.ick If is iben up to ibe run-nvnc swein 10 decide which thread to schedule on that CPU- the interrupted one. the ncvly ready one. or vtnne third choice. An objection to scheduler activations is ibe lunilarnenial reliance on npealls, a concept ibat violates ihe structure inherent in any layered system Normally, layer n otters certain services that layer n + 1 can call on. hut layer ii may not call pro- procedures in layer n + 1. Upcalls do not follow this fundamental principle.

96 PROCESSES AND THREADS CHAP 2 2.2.7 Pop-Up Threads Threads are frequently useful in distributed systems An important example is how incoming messages, for example requests tor service, are handled. The nadi- tionai approach is to have a process or thread that is blocked on a receive system call waiting tor an incoming message. When a message arrives, it accepts The message and processes it. However, a completely different approach ir> also possible, in which the arrival ot a message causes the system to create a new thread to handle the mes- sage Such a thread is called a pop-up thread and is illustrated in Fig 2-15. A key advantage ot pop-up threads is that since they are brand new. they do not have any histoiy—registers, stack, etc. that must he restored. Each one starts oui fresh and each one rs identical to all the others This, make*- it possible to create such d thread quickly. The new thread is given the incoming message to process. The result ot using pop-up threads is that the latency between message arrival and ibn start of processing can be made very short Figure 2.15. Cre ves (b) Aher the it \ thread when j menage Some advance planning is needed when pop-up threads .ire used For pie. in which process does the thread run? If the system supports threads r in the kernel's context, the thread may run there (which is why we have not the kernel in Fig. 2-15). Having the pop-up thread run m kernel space is i easier and faster than putting jt in user space Also, a pop-up thread hi isually kernel

SEC 2 2 THREADS 97 space can easily access all the kernel's tahles and the I/O devices, which ma> be needed for interrupt processing. On the other hand, a buggy kernel thread can do more damage than a buggy user thread. For example, if it runs too long and there is no way to preempt it, incoming data may be lost 2.2.8 Making Single Threaded Code Multithreaded Many existing programs were wrilien for single-threaded processes'. Conven- Convening these to multithreading is much trickier than it may at first appear Below we will examine just a few of the pitfalls. As a stan. the code of a thread normally consists ol multiple procedures. \Ust like a process. These may have local variables, global variables, and procedure parameters. Local vanahle1. and parameters do not cause any trouble, but van- ahles that are global to a thread but not global to the entire program do. These are variables that are global in the sense that many procedures within the thread use them (a1- they might use any global variable), but other threads should logically leave them alone As an example, consider the errno variable maintained by UNIX. When a process (or a thread) makes a system call that fails, the error code is put into errno. In Fig. 2-16. thread 1 executes the system call access to find out if it has permission !o access a certain file. The Operating system returns the answer in the global variable errno. After control has returned to thread 1. but before it has a chance to read errno, the scheduler decides that thread I has had enough CPU time for the moment and decides to switt-h to thread 2 Thread 2 executes an Open call that falls, which causes errno to be overwritten and thread l's access code to be lost forever When thread 1 siarts up later, it W]ll read the wrong value and behave incorrectly. i Errno inspected Figure 2-16. Condice.

98 PROCESSES ANO THRbADS CHAP 2 Various solutions to this problem are possible. One is to prohibit global vari- variables altogethci. However worthy thls ideal may be, it conflicts with much exist- existing software Another is to assign each thread its o*i\ private global variables, as shown in Pig. 2-17 In this way, each thread has its own private copy of errno and other global variable;,- so conflicts ate avoided. In efffeci. this decision creates a new scoping level, variables visible to all the procedures of a thread, in addition to the existing scoping levels of variables visible only to one procedure and vanahles visible everywhere in the piogtam. Accessing ihe private global variables is a bit tricky, however si net- most pro- programming languages have ,i way of expressing local variables and global vari- variables, hut not intermediate forms It is possible to allocate a chunk or memor> for the

globals and pass it to each procedure in the thread, ,is an extra parainetc. While hardly an elegant solution, it works Alternatively, new lihraiy procedures Lan be intiDdnced to create, sei, and read these thread-wide global variables The hi si eall might look like-this Create g!oba!(b fpt It allocate t g I i area re^er d t tl I only tht e II g th d I a global v b! th tie not conflict tl th t t lid/ tpt, un the ht-ap or in a special storage the I N mailer whtrc the stor.ige is allocated. t th 1 bal variable If another thread c reaics Two t II -e n d t other for r dnglhn [I g set_g!oba!("bufptr", &buf); arables I ing hke

SEC 2: THREADS ^ will do I! stores ihe value of a pointer in the storage location previously created by the call to create__global. To read a global variable, the call might look like bufptr - read, global("bufptr"); It returns the address stored in the global variable, so its data can be accessed. The next problem turning a single-threaded piogram into a multithreaded pro- program i_s that many library procedures are not reentrant. That is, they were not designed to have a second call made to any given procedure while a previous call has not yet finished. For example, sending a message over the network may well be programmed to assemble the message in a fixed buffer within the library, then to trap lo the kernel to send ft- What happens if one thread has assembled m mes- message in ihe butter, then a clock interrupt forces a switch to a second thread tliat nnmedi.ite]} overwrites the buffer with us uwii message? SirmhiHy, memory allocation procedures, such as nialhc in UNIX, maintain ciucial tables about memory usage, for ex,,mple. a linked l,st of available chunks or memoiy While malloc is busy updating these lists, they may temporarily bo in an inconsistent state, witb pointers that point nowhere. If a thread switch occurs while the tables are inconsistent and a new call comes" in from a dilferent thread, an invalid pointer may be u.sed, leading to a program crash. Fixing all these prob- problems properly effectively means rewriting the entire library. A different solution is 10 provide each procedure with ajacket that sets a bii to mark the library as in use Any attempt for another thread to use a lihrary pro- procedure while a previous call has not yet completed is blocked Although ihis approach can be niade to work, it greatly eliminates potential parallelism. Next, consider signals. Some signals are logically thread specific, whereas others are not. For example, if a thread calls alarm.]t makes sense for the result- resulting signal to go to the thread (hat made the call. However, when threads .ire implemented entirely in user space, the kernel does not even know about threads and can hardly direct the signal to

the right one. An additional complnation occurs if a process may only have one alar in at & time pending and several threads call alarm independently. Otber signals, such as keyboard interrupt, are not ihread specific Who should catch them? One designated thread1' All 'he threads' A newly treated pop-up thread? Furthermore, what happens if ont thread changes ihe signal htindbrs without telling other threads? And what happen* if one thread wants 10 catch a particular signal (say. the user hitting CTRL-C), and another thread wants this sig- signal to terminate the process' This situation can arise if one or more threads run standard library procedures .ind others are user-written Cltarly. these wishes aje incompatible. In general, signals are difficult enough to manage in ;i single- ihreaded environment Going to a multithreaded envmjnment does not make One last problem introduced by threads is slack management. In many sys- systems, when a process' stack overflows, the kernel |ust provides that process with

100 PROCESSES AND THREADS CHAP 2 more stack automatically. When a process has multiple thicads. it must also h;ive multiple stacks. If the kernel is not aware ot all these stacks, n cannot grow ib«m automatically upon slack fault In fact, u may not even realize that a memory' fault is related to stack growth. These problems are certainly not insurmountable, but they do show that just troducmg threads into an existing system without a fairly substantial system ,^design is not going to work at all. The semaniics of .system calls may have to be redefined and libraries have to be rewritten, at the very least And all of the.se things must be done in such a way as, to remain backward compatible with exUt- — programs for the Inciting case of a process with only one thread. For addi- lal information about threads, see (Hauser ct al., 1993; and Marsh et al , 1991). 2.3 INTERPROCESS COMMUNICATION Processes frequently need to communicate with othei processes For exam- example, in a shell pipeline, the output of the first process must be passed to the second process, and so on down the line. Thus ihere is a need toi communication between processes, preferably in a well-snudured way not using interrupts. In the following sections we will look at some of the issues related to this Interpro- Interprocess Coiranunlcation or IPC Very briefly, there are three issues here The first was alluded to above, how one process can pass information to another The second hat. to do with making sure two or more processes do not get into each other's way when engaging in critical activities (suppose two processes each try to grab the last I MB iji memory). The third

concerns proper sequencing when dependencies are present: if process A produces data and process B prints them. B has to wail until A Iws produced some data before starting to print We will examine all three ol ibtse issues starting in the nexi seetion. It is also important to mention that two ot ibese issues apply equally well lo threads. The first one—passing information_ts easy for threads since they shaie a common address space (threads in different address spaces that need to com- communicate fall under the heading of communicating processes') However, the other two—keeping out of each other's hair and proper sequencing—apply equally well to threads. The same problems exist and the same solutions appK Below we will discuss the problem in the context ot processes, bm please keep in mind that the same problems and solutions also apply to threads 2.3.1 Race Conditions In some operating systems, processes that arc working together may share some common storage that each one can iead and write The shared storage may be in main memory (poswbty in a kernel data structure) or it may be a shared file.

SLC 2 3 INTERPROCESS COMMUNICATION 101 the location of the shared memory docs not change ihe nature of the communica- communication or the problems that arise To see how interprocess communication works, in practice, let us consider a -simple but common example, a print spooler. When J process wants, to pnnt a hie. it enters the tile name in a special spooler directory Another process, the printer daemon, periodically checks to see if there are any files to be printed, and it there ate, ii prints them and then removes thar names fniiri the directory. Imagine that our spooler directory has a very large number of slots, numbered 0. 1, 2. ., , each one capable of holding ,i file name. Also imagine that there are Iwo shared variables, out, which points to the nexl file to be printed, and in, ^hit-h points to the next fret: slot m the directory. These two variables might well be kept on a two-word file available to all processes At a certain instant, sluts 0 to 3 are empty (the files have already been printed) and slots 4 to 6 are full (with the names of files queued for printing). More or less Simultaneously, processes A and B decide they want to queue a file for printing This situation is shown in Fig. 2- 18 progc piog n m Figure 2-18. Tw(In jurisdictions where Murphy's lawt (S applicable, ihc following mtgh! hap- happen. Process A reads in and stores the value. 7, in a local variable called next-free slot. Jus! then a clock imerrup! occurs and fhe CPU decides fha! pioc- ess A has tun long enough, so if switches lo process B Piocess B also reads in. and also gels a 7. It loo stores

it in in local variable next-free-\lot At this instant both processes think that the next available slot is 7 Process B now continues to run. It stores the name of its file in slot 7 and updstes in to be an 8. Then it goe.s off and does other things Eventually, process A runs again, starting from the place tt loft off It looks at next-free slot, finds a 7 there, and writes Us file name in slot 7. erasing the name + tf sumeihfUg can go wrong II wi!t

102 PROCESSES AND THRHADS CHAP 2 that process B just put there. Then it compi^ next_jree_s!oi + 1. whtcli us 8, and »ets in to 8. The spooler directory is now internally consistent, so the printer daemon will not notice anything wrong, but process B will never receive any out- output. User B will hang around the printer room for years, wistfully hopmg for out- output that never tomes. Situations like this., where two or more processes are read-reading or writing some shared data and the final result depends on who runs precisely when, are called race conditions. Debugging programs containing race condi- conditions is no fun at all. The results ot most test runs are hue. but once in a rare- while something weird and unexplained happens 2.3.2 Critical Regions How do we avoid race conditions'* The key to preventing trouble here and in many other situations involving shared memory, shared files, and shared every- everything else is to find some way lo prohibit more than one process from reading and writing the shared data at the same time. Put in other words, what we need is mutual exclusion, that is, some way of making sure that if one process is using a shared variable or file, the other processes will be excluded from doing the same thing. The difficulty above occurred because process B started using ()ne of the shared variables before process A was finished with it. The choice of appropriate primitive operations for achieving mutual exclusion is a major design issue in any operating system, and a subject lhat we will examine in great detail in the follow- following seci ions. The problem of avoiding race conditions can also be formulated in an abstract way. Pan of the time, a process is busy doing internal computations and other things that do not lead to race conditions. However, sometimes a process have to access shared memory or files, or doing other critical things that can lead to races That part of the program where Ihe shared memory is accessed is called the criti- critical region or critical section. If we could arrange matters such that no two proc- processes were ever in their critical regions at the same time, we could avoid races. Although this requirement avoids race conditions, this is not sufficient for having parallel processes cooperate correctly and

efficiently using shared data. We need four conditions to hold to have a good solution: 1. No two processes may be simultaneously inside their critical regions. 2. No assumptions may be made about speeds or the number of CPUs. 3. No process running outside its critical region may block other processes. 4. No process should have to wait forever to enter its critical region. In an abstract sense, the behavior that we want is shown in Fig. 2-19. Here process A enters its critical region at time 7",. A little later, at time T2 process B attempts to enter its critical region but fails because another process is already in

SEC. 2 3 INTERPROCESS COMMUNICATION 103 its critical region and we allow only one at .1 time. Consequently. B is temporarily suspended until time r3 when A leaves its critical region, allowing B to enter immediately. Eventually B leaves (at TA) and we are back to the original situation with no processes in their critical regions region 2.3.3 Mutual Exclusion with Busy Waiting In this section we will examine various proposals for achieving mutual exclu- exclusion, so that while one process is busy updating .shared memory in its critical region, no other process will enter its critical region and cause trouble. Disabling Interrupts The simplest solution is to have each process disable all interrupts just utter entering its critical region and re-enable them just before leaving it. With inter- interrupts disabled, no clock interrupts can occur. The CPU is only switched from process to process as a result of clock or other interrupts, after all, and with inter- interrupts turned off the CPU will not be switched to another process Thus, once a process has disabled interrupts, it can examine and update the shared memory without fear that any other process will intervene. This approach is generally unattractive because it is unwise to give user processes the power to turn off interrupts Suppose that one of them did it, and never turned them on again? That could be the ond of the system. Furthermore, if the system is a multiprocessor, with two or more CPUs, disabling interrupts affects only the CPU that executed the disable instruction. The other ones will continue running and can access the shared memory

104 PROCESSES AND THREADS CHAP. 2 On the other hand, it is frequently convenient tor the kernel iiself to disable interrupt for a few instructions While it is updating variables or lisis. If an inter- interrupt occurred while the list of ready process, for example, was m an inconsistent

state, race condntons could occur. The conclusion is: disabling imerrupis is ofien a useful technique within the operating system itself but k no! appropriate as a general mutual exclusion mechanism for user processes. Lock Variables As a second attempt, let us look for a sottware solution Consider having a single, shared (lock) variable, ininally 0. When a process wants to enter its criti- critical region, it first test;, the lock. If the lock is 0, the process sets n to 1 and emers the critical region If the lock is already 1, the process just waits until it becomes 0. Thus. a 0 means that no process h in its critical region, and a 1 means that some process is in Us critical region. Unfortunately, this idea contains exacily the same fatal flaw that we saw in the spooler direciory. Suppose that one process reads' the lock and sees thai it is 0. Before it can set the lock to 1, another process is scheduled, runs, and sets ihe lock to]. When ihe first process runs again, it will also set the lock to 1, and two processes will be in their critical regions at the same time Now you might think ihat we could get around this problem by first reading out the lock value, then checking it again just before storing into it. but that really does not help The race now occurs if the second process modifies the lock just after the first process has finished its second check. Strict Alternation A third approach to the mutual exclusion problem is shown in Fig. 2-20. This program fragment, like nearly all the oihers in this book, is written in C. C was chosen here becau.se real operating systems dre virtually always wntien in C (oi occasionally C++), but hardly ever in languages like Java, Modula 3, or Pascal. C is powerful, efficient, and prediciable, characteristics critical for writing operating systems. Java, for example, is not predictable because]t might run out of storage at a critical moment and need lo invoke the garbage collector at a most inoppor- inopportune time. This cannot happen in C because there is no garbage collection in C A quantitative comparison of C, C++. Java, and four other languages]S given in (Prechelt, 2000). In Fig. 2-20, the integer variable turn, initially 0, keeps track of whose turn it is to enter the critical region and examine <ir update the shared memory. Initially. process 0 inspects turn, finds u to be 0. and enters its critical region. Process 1 also finds it to be 0 and therefore sits in a tight loop continually testing turn to see when it becomes 1. Continuously testing a variable until some value appears is called busy waiting, it should usually be avoided, since it wastes CPU time.

 while (TRUE) { while (TRUE) { while (lum != 0) /* loop */; while (turn '= 1) /* loop * critical_region();

cntical_region(); turn = 1; turn = 0; noncnHcal_regton(), noncntica!_region{); } } (a) (b) Only when there is a reasonable expectation that the wait will be short is busy waiting used. A lock thai uses busy waiting is called a spin lock When process 0 leaves the critical region, it sets turn to 1, to allow process 1 to enter its critical region. Suppose that process 1 finishes its critical region quickly, so both processes are in their noncnncal regions, with mm set to 0. Now process 0 executes its whole loop quickly, exiting its critical region and setting turn to 1. At this point turn is 1 and both processes are executing in their noricnti- cal regions Suddenly, process 0 finishes its noncntical region and goes back to the top of its loop. Unfortunately, it is not permitted to enter its critical region now, because turn is 1 and process 1 is busy with its noncritical region. It hangs in its while loop until process 1 sets turn to 0. Put differently, taking turns is not a good idea when one of the processes is much slower than the other. This situation violates condition 3 set out above: process 0 is hcing blocked by a process not in its critical region. Going back to the spooler directory discussed above, if we now associate the critical region with reading and writing the .spooler directory, process 0 would not be allowed to print another file because process 1 was doing something else. In fact, this solution requires that the two processes stnetly alternate in enter- entering their critical regions, for example, in spooling files. Neither one would be permitted to spool two in a row. While this algorithm does avoid al! races, it is not really a serious candidate as a solution because it violates condition 3 Peterson's Solution By combining the idea of taking turns with the idea of lock variables and warning variables, a Dutch mathematician, T. Dekicer, was the first one to devise a software solution to the mutual exclusion problem that docs not require strict alternation. For a discussion of Dekkers algorithm, see (Dijkstra. 1965). In J98I, G.L. Peterson discovered a much simpler way to achieve muiuai exclusion, thus rendering Dekker's solution obsolete. Peterson's algorithm is shown in Fig. 2-21. This algorithm consists of two procedures written in ANSI C,

106 PROCESSES AND THRbAPS CHAP. 2 which means that function prototypes should be supplied for all the functions defined and used. However, to save space. %ve will not show the prototypes in this or subsequent examples. #define FALSE 0 #defineTRUE 1 #deflne N 2 mt mterestedfN]; void enter_region(mt process); mt other; other = 1 - process; /* number of processes */ /* whose turn is if */ /* all values initially 0 (FALSE) */ /*

process is 0 or 1 */ /* number of the other process */ /* the opposite of process */ mterested[process] - TRUE; /* show that you are interested */ turn = process; /* set flag */ while (turn == process && intefested|other] == TRUE) /* null statement * void leave_region(int process) /* process; who is leaving */ (mterestedfprocess] - FALSE, /* indicate departure from cntical region */ Figure 2-21. Peiei ualo Before using the shared variables (i.e., before entering its critical region), each process, calls enter .region with its own process number. 0 or 1, as parame- parameter. This call will cause it to wait, if need be. until it is safe to enter. After it has finished with the shared variables, the process calls leave ^region to indicate that it is done and to allow the other process to enter, if it so desires Let us see how this solution works. Initially neither process is in its critical region. Now process 0 calls enter _regum. It indicates its interest by setting its array element and sets turn to 0. Since process I i.s not interested, enter_region returns immediately, if process I now calls enter _region, it will hang there until mterested\Q] goes to FALSE, an event that only happens v.hen process 0 calls leave_region to exit the critical region. Now consider the case that both processes call enter^reyion almost simul- simultaneously. Both will store their process number in turn- Whichever store is done last is the one that counts; the first one is overwritten and lost. Suppose that proc- process I stores last, so turn is 1. When both processes come to the while statement, process 0 executes it zero times and enters its critical region. Process 1 loops and does not enter its critical region until process, 0 exits its critical region.

SEC 2 1 INTERPROCESS COMMUNICATION H" The TSL Instruction Now let us look at a proposal that requires a little help from the hardware. Many computers, especially those designed with multiple processors in mind, TSL PX.LOCK (Test and Set Lock) that works as follows It reads the contents of the memory word lock into register RX and then stores a nonzero value aE the memory address lock. The operations of reading the word and storing into it are guaranteed to be indivisible—no other processor can access the memory word until the instruction is finished. The CPU executing the TSL instruction locks the memory bus to prohibit other CPUs from accessing memory until it is done To use the TSL instruction, we will use a shared variable, lock, to coordinate access to shared memory. When lock is, 0. any process may set il to I using the TSL instruction and then read or write the shared memory. When it is dune, the process sets lock back toO using an ordinary move instruction.

How can this instruction be used to prevent two processes from simultane- simultaneously entering their critical regions'? Thu solution is given in Fig 2-22. There a four-instruction subroutine in a fictitious (but typical) assembly language is shown. The first instruction copies the old value of lock to the register and then sets lock to I. Then the old value is compared with 0. If it is nonzero, the lock was already set, so the program just gtjes back to the beginning and tests it again. Sooner or later it will become 0 (when the process currently in its critical region is done with its critical region), and the subroutine returns, with the lock set. Clear- Clearing the lock is simple. The program just stores a 0 in lock. No special instruc- instructions are needed. enter.region: TSL REGISTER.LOCK i copy lock to register and set lock to 1 CMP REG1STER,#O I was lock Zero" JNE enter, region I if it was non zero, lock was set, so loop RETI return to caller; critical region entered leave _ region: MOVE LOCK,#0 I slore a 0 in lock RET I return to caller Figure 2-22. F.nienng ami leaving a iriiicjl region using the TSL ilistruciiiui One solution io the critical region problem is now straightforward. Before entering its critical region, a process calls enter_region, which does busy waiting until the lock is free; then it acquires ihe lock and returns. After the critical region

 the process calls leave .region, which stores d 0 in lock As witn all solutions based on crincal regions, the processes must call eutei _region and leave _region at the correct times for the method to work. It a process cheats, the mutual exclu- exclusion will fail. 2.3.4 Sleep and Wakeup Both Peierson's solution and the solution using TSL are correct, but both have the deteci of requiring husy waning In essence, what these solulions do is this when a process wants to enter its critical region, it checks to see it the entry is allowed It it is not, the process just sits in a tight loop waiting until it is Not only does this approach waste CPU time, but it can also have unexpected effecis. Consider a compuier with two processes, //. with high priority and L. with low priority. The scheduling rules are such lhai H is run whenever u is in ready state. At a certain niomeni, with L in Us critical region. H becomes ready lo run (e.g.. an I/O operation completes), fj now begins busy waiting, but since L is never scheduled while H is running, I never gels the chance to leave its critical region, so H loops forever This situation is sometimes referred to as the priority inversion problem. Now let us look at some interprocess communication primitives that block instead of wasting CPU lime when ihey are not allowed to enter their critical regions. One of the

simplest is the pair steep and wakeup. Steep is a system call that causes the caller to block, that is. be suspended mini another process wakes n up. The wakeup call has one parameter, the process lo be awakened. Alterna- lively, both steep and wakeup each have one parameler. a memory address used to match up steeps with wakeups. The Producer-Consumer Problem As an example of how these primitives can be used, let us consider the producer-consumer problem (also known as ihe bounded-buffer problem) Two processes share a common, fixed-size buffer One of them, the producer, puts information into the buffer, and the other one, the consumer, takes it out (It is ,ilso possible to generalise the problem to have in producers and n consumers, but we will only consider the case of one producer and one consumer bccau.se this assumption simplifies the solutions). Trouble arises when the producer wants to pui a new item in the buffer, but it is already full The solution is tor the producer to go to sleep, to be awakened when the consumer has removed one or more items Similarly, it the consumer wants to remove an item from the buffer and sees that the buffer is empty, it goes to sleep unhl the producer puts something in the huffer and wakes it up This approach sounds simple enough, but it leads to the same kinds of race conditions we saw earlier with the spooler directory To keep track of the number

SfcC 2 } fNJ'ERPROCESS COMMUNICATION 109 of items in the buffer, we will need a vanabfe. count ff ihe maximum number of items the buffer can hofd is N, the producer's aide wifl first test to see it count is A'. If it is. the producer witl go to .sleep, it it is. not. the pioducer will add an item and increment count The consumer's code is simitar first test count to sec if it is () If it is. go to sleepV \i it is nonzero, remove an item and decrement the counter Each of the processes also tests tt> see if the other should bo awakened, and it so, wakes it up. The code for both producei and consumer is shown in Fig 2-23. #defmeN 100 int count = 0 void producer(void) ml item. while (TRUE) { item = produce_item{); if (count == N) sleep{); insert _ item (item); count = count + 1; if {count == 1) wakeup(consumer), /* number of slots in the buffer */ /* number of items in the buffer */ /* repeal forever */ /* generate next item */ /* if buffer is fulf, go to sleep •/ /•put item in buffer*/ /* increment count of items in buffet /• was buffer empty? */ void consurner(void) { int item, white (TRUE) { /* repeal forever */ if {count == 0) sieep(); /* if buffer is empty, got lo s(i item = remove _item(); /• take item out of buffer */ count = count - 1; /* decrement count of items if {counl == N - 1) wakeup (producer); /* was buffer

futt? •/ consume_item(ilem)- /* print item */] re 2-23. The produce er problem u.ilh a latul To express system calls such as steep and wakeup in C. we will show them as calls to library routines. They are not part ot the standard C tibrary but presum- presumably would be available on any system that actually had these system calls. The procedures insert-item and remove ^item^ which are not shown, handle the book- bookkeeping of putting items into the buffer and taking items out of the buffer.

HO PROCESSES AND THREADS CHAP 2 Now let us get back to the race condition. It can occur because access to count is unconstrained. The following situation could posstbly occur. The buffet is empty and the consumer has just read count to see if it is 0 At that instant, the scheduler decides to stop running the consumer temporarily and start running the producer. The producer tnserts an item in the buffer, increments count, and notices that it is now I. Reasoning that count was just 0, and thus the consumer must be steeping, the producer calfs wakeup to wake the consumer up. Unfortunately, the consumer is not yet fogicaffy asleep, so the wakeup signaf is fust. When the consumer next runs, it will test the vafue of count it previously read, find it to be I), and go to sleep. Sooner or fater the producer will filf up the buffer and 3lso go to sleep. Both will sleep forever. The essence of the problem here is that a wakeup sent to a process that is not (yet) steeping is lost. If it were not lost, everything would work. A quick fix is to modify the rules to add a wakeup waiting bit to the picture. When a wakeup is sent to a process that is stiff awake, this bit is set. Later, when the process tries to go to sfeep, if the wakeup waiting bit is on, it will be turned off. but the process will stay awake. The wakeup waiting bit is a piggy bank for wakeup signafs. While ihe wakeup waiting bit saves the day in thts simple exampfe. it is easy to construct examples with fhree or more processes in which one wakeup wailing bit is insufficient. We could make another patch and add a second wakeup wait- waiting bif, or maybe 8 or 32 of them, buf in pnnctple the probfem is still there, 2.3.5 Semaphores This was the situation in 1965, when E. W DtjkstraA965| suggested using an integer variable to count the number of wakeups saved for future use. In his pro- posaf, a new variable type, called a semaphore, was introduced. A semaphore could have the value 0, indicating that no wakeups were saved, or some positive vafue if one or more wakeups were pending. Dijksira proposed having two operations, down and up (generalizations of sleep and wakeup, respectively). The down operation on a semaphore checks to see if the value is greater than 0. if so. it

decrements the value (i.e., uses up one stored wakeup) and just continues. If the value is 0, the process is put to sleep without completing the down for the moment. Checking the value, changing it, and possibly going to sleep, is all done as a single, indivisible atomic action. It is guaranteed that once a semaphore operation has started, no other process tan access the semaphore until the operation has completed of blocked. This atomi city is absolutely essential to solving synchronization problems and avoiding rate conditions. The up operation increments the value of the semaphore addressed. If one or more processes were sleeping on that semaphore, unable to complete an earlier down operation, one of them is chosen by the system (e.g., at random) and is allowed to complete iu down. Thus, after an up on a semaphore with processes

SEC 2.3 INTERPROCESS COMMUNICATION 111 sleeping on it the semaphore will stilt be 0. but there wilt be one fewer process sleeping on it. The operation of incrementing the semaphore and waking up one process is also indivisible. No process ever block* doing an up. just as no process ever blocks doing a wakeup in the earlier modet As an aside, in Dijkstra's original paper, he used the names P and V instead of down and up, respectively, but since these have no mnemonic significance to peo- people who Jo not speak Dutch (and only marginjf significance to those who do), we witl use the terms down and up instead. These were first introduced in Algol 68. Solving the Producer-Consumer Problem using Semaphores Semaphores solve the lost-wakeup puiblem, as shown in Fig. 2-24 It is essential that they be implemented in an indivisible way The normal way is to implement up and down as system calls, with the operating system briefly disa- disabling atl interrupts while it is testing the semaphore, updating it. and putting the process to sleep, if necessary. As all of ihese actions take only a few instructions, no harm is done in disabling interrupts If multiple CPUs are being used, each semaphore should be protected by a lock variable, with the TSL instruction used to make sure that only one CPU at a titiie examines the semaphore. Be sure you understand that using TSL to prevent several CPUs from accessing the semaphore at the same lime is quite different from busy waiting by the pioducer or consumer waiting for the other to empty or fill the butter The semaphore operation will only take a few microseconds, whereas the producer or consuinei might take .irbi- tranly long. This solution uses three semaphores one called full for counting the number of slots that are full, one called empty tor counting the number of slots

thai are empty, and one called muiex to make sure the producer arid consumer do not w.te the buffer at the same time. Full is initially 0. empty is initially equal to the number of slots in the buffer, and mutex is initially 1. Semaphores that are ini- initialized to 1 and used by two or more processes to ensure that only one of them n enter t t 1 ey \t the rrn. time are called binary semaphores. It each process does a down rust before entering its critical region ant! an up just tte le \nc t m t 1 e\clu n j, ntced. N th t e h\ e j, w nterp t s communication primitive <tt our dispo- 1 let u £, b ck n lwk t the it rrupl sequence of Fig. 2-5 again In a sys. tern using semaphores, the natural way to hide interrupts is to have a semaphore, initially set to 0, associated with each I/O device. Just after starting an I/O device, the managing process does a down on the associated semaphore, thus blocking immediately. When the interrupt comes in. the interrupt handler then does an up on the associated semaphore, which makes the relevant process ready to run again. In this model, step 5 in Fig. 2-5 consists ot doing an up on the device \ semaphore, so that in step 6 the scheduler will be ahle to run the device manager. Of course, if several processes are now ready, the scheduler may choose to run an

PROCESSES AND THREADS #defmeN 100 typedef int semaphore; semaphore mutex =t 1; semaphore empty = N; semaphore futl = 0; void producer(void) number of slots in the buffer */ semaphores are a special kind of tnt */ controls access to critical region */ counts empty buffer stots */ counts fuff buffer slots*/ lie (TRUE) { item = produce_item{); down{&empty); down(&mufex); inserts Item(item), up{&mutex); up(&fult), /* TRUE Is the constant 1 */ /* generate something to put in buffer /* decrement empty count */ /• enter cntlcat region •/ /* put new item In buffer */ /* teave crificat region •/ /* increment count of futt slots */ oid consumer(vo!d) int item; while (TRUE) (down{&fu(t); down(Smutex); item = remove Jtem(); up{&rnutex); up(&empty); consume _item{item) ■] /• infinite loop */ /* decrement futt count */ /* enter critical region •/ /* take ifem from buffer */ /* teave cntlcat region •/ /* increment count of empty stots •/ /* do something wifh fhe item */ Figure 2-24. The producer roblem using selnjphor e of the algorithms used even more important process next. We will loo for scheduling later on in this chapter. In the example of Fig 2-24. we have actually used semaphores in two dif- different ways. This difference is important enough to make explicit. The mutex semaphore is used for mutual exclusion It ts designed lo guarantee that onl> one process at a time will be reading or Writing the buffer and the

associated variables. This mutual exclusion is required to prevent chaos. We will study mutual exclu- exclusion and how to achieve it more in the next section.

SEC. 2 3 The other INTERPROCESS COMMUNICATION 113 use ot semaphores is for synchronization. The full and empty semaphores are needed lo guarantee lhal certain evenl sequences do or do no! occur In ihis case, Iftey ensure thai Ihe producer stops running when the buffer is full, and Ihe consumer stops running when it is emply. This use is difterenl from mulual exclusion. 2.3.6 Mutexes When the semaphore's abilily to counl is not needed, a simplified version of the semaphore, called a mutex, if. sometimes used. Mutexes arc good only for managing mutual exclusion lo some shared resource or piece of code. They are easy and efficient lo implement, which makes them especially useful in thread packages that are implemented entirely in user space. A mutex is a variable that can be in one of two states: unlocked or locked. Consequently, only 1 bil is required lo represenl it, but in practice an integer <>flen is used, with 0 meaning unlocked and alt olher values meaning locked. Two pro- procedures are used with mulexes. When a ihread (or process) needs access lo a crit- critical region. It calls mutex Jock. If the mutex is cunenl unlocked {meaning thai the critical region is available), the call succeeds and Ihe calling thread is- free to enter the cntical region. On the other hand, if the mutex is already locked, the calling thread is blocked until the thread in the critical region is finished and calls mutex ..unlock. If multi- multiple threads are blocked on the mutex, one of them is chosen at random and allowed to acquire the lock. Because mutexes are so simple, they can easily be implemented in user space if a TSL instruction is available. The code for mutex Jock and mutex^unhek for use with a user-level threads package are shown in Pig. 2-25 mutexJock: I copy mutex to register and set mutex to 1 I was mutex zero" I if it was zero, mutex was unlocked, so retun I mulex is busy; schedule another ihread I try again latei TSL REGtSTER,MUTEX CMP REGISTER #0 JZEok CALL thread _yield JMP ok: RET In n to caller; critical region entered mutex_ unlock: MOVE MUTEX,#0 I store aOm mutex RET I return to caller Figure 2-25. Implementation tif mutex lock <ind itulex_unhx.k The code of mutex-lock is similar to the code of enter^regitm of Pig. 2-2; but with a crucial difference. When enter^region fails to enter the critical region it keeps testing the lock repeatedly (busy waiting). Eventually, the clock run-' ou

 and some other process is scheduled to lun Sooner or later the process holding the lock gets to run and releases it. With threads, the situation is different because there is no clock that stops threads that have run loo long Consequently, a thread that tries to acquire a lock by busy waiting will loop forever and never acquire the lock because it never allows any other thread tn run and release the lock. That is where the difference hetween enter teuton and muicxJock comes m. When the later fails to acquire a i.ick. it calls thuwl- vwld to give up the CPU lo another thread. Consequently ihere is no busy wailing When the thiend runs the next time, it tests the lock again. Since thread vield is |ust a call lo the thread scheduler in u.ser space, it is very tast. As a consequence, neithei muteX-lock nor muie_unlock requires any kernel calls. Using them, user-level threads can synchronize entirely m user space using procedures that require only a handful ot" instructions. The mutex system that we have described above is a bare bones set of calls With all software, there is always a demand tor more features, and synchroniza- synchronization primitives are no exception. For example, sometimes a thread package offers a call muiex_irylock that eilher acquires the lock or returns a eode for failure, but does not block. This call gives Ihe thread thu flexibility to decide what to do next if there are alternatives to just wailing Up until now there is an issue that we have glossed over lightly l>ut which i> worth at least making explicit W;th a user-space threads package there ts Ho problem with multiple threads having access to the same mutex since all ihe threads operate in a common address space However, with most of the earlier solutions, such as Peterson's algonlhm and semaphores, there is an unspoken assumption that multiple processes have access, to at leasi some shared memory, perhaps only one word, but something. If processes have disjoint addiess spaces, as we have eonsislently said, how can ihey share the mni van.ibie in Peteison\ algorithm, or semaphores or a common bulfer? There are Iwo answers, Firsl, some of the shared daia strtielures, such js the semaphores, can be stored in the kernel and only accessed via system calk This approach ehminaies the prohlem. Second, mosl modern operaling systems (including UNIX and Windows) offer a way for processes to share some portion o! their address space with other processes. In ihis way. butlers and other data struc- turet, can be shared. In ihe worsl case, th.it nothing else is possible, a shared file can be used If iwo or more processes share mosl or all of their address spaces, the distjne- lion belween processes and ihreads becomes somewhat blurred bill is nevertheless present.

Tw() processes lhat share a common address space still have different open files, alarm timers, and olher per-process properlies. whereas ihe threads within a single process share them. And U is always true lhal multiple processes sharing a common address space never have ihe efficiency of user-level threads since the kernel is deeply involved in Iheir managemenl.

SEC 2 3 INTERPROCESS COMMUNICATION 11$ 2.3.7 Monitors Wilb semaphores interprocess communication looks easy, righl? Forget it. Look closely at ihe order of (he downs before inserting or removing items from Ihe buffer in Fig. 2-24. Suppose lhat the (wo downs in the producer's code were reversed in order, so mutex was decremented before empty instead of after >1 If (he buffer were completely full, (he producer would block, with mutex sel to 0. Consequenliy, Ihe nexl lime Ihe consumer Ined lo access (he buffer, il would do a down on mutex. now t), and block loo. Bolh processes would May blocked forever and no more wofk would ever be done. This unfortunale silualion is called a deadlock. We will study deadlocks in detail in Chap. 3. This problem is pointed out to show how careful you musl be when using semaphores. One subtle error and everything comes to a grinding halt. It is like programming in assembly language, only worse, because the errors are race con- conditions, deadlocks, and olher forms of unpredictable and irrcproducible behavior. To make il easier lo write correct programs, Hoare 11974) and Bnnch Hansen A975) proposed a higher-level synchronization primitive called a monitor. Their proposals differed slightly, as described below. A monitor is a collection of pro- procedures, variables, and data structures that are all grouped together in a special kind ot module or package. Processes may call the procedures in a monitor when- whenever Ihey wanl lo, bul they cannot directly access the monitor's internal data structures from procedures declared outside the monitor. Figure 2-26 illustrates a monitor written in an imaginary language. Pidgin Pascal. monitor example integeri; condition!; procedure produreii): end: procedure comumeri.); end: end monitor; Figure 2-26, A mpnlior. Monitors have an important property that makes them useful for achieving mutual exclusion: only one process can be active in a monitOf at any instant. Monitors are a programming language construct, so the compiler knows they are special and can handle calls to monitor procedures differently from other pro- procedure calls. Typically, when a process calls a monitor procedure, the first few

116 PRWHSSFS AND THREADS CHAP 2 instructions ,if the procedure will chc-ck lo sec if an> other process is currently active wilhin (he moniloi. tf so. Hie calling process will be suspended unlii (he olher process has left (he monitor. If no olher process is using (he monitor. Hie calling process may enler. 11 is up to the compiler Id implement the mutual exclusion on monitor entries, but a common way is to use a mutex or binary semaphore. Because the compiler, not the progiiimmei, is arranging for the mutual exclusion, it is much less likely that something will go wrong. In any event, the person writing ihe monitor does not have to be aware of how the compiler arranges for mutual exclusion It is sur- ficienl to know that by turning all the critical regions into monilor procedures, no two processes will ever execule their crilicai regions at Ihe same time Although monitors provide ail easy way to achieve multial exclusion, as we have seen above, that U, not enough. We also need a way for process lo block when they cannot proceed. In the producer-consumer problem, it is easy enough to pul all the tests for bulfer-full and buffer-cniply in monitor procedures, but how should the producer block when il finds the buffer full" The solution lies in ihe introduction of condition variables, along with two operations on them, wait and signal. When a monitor procedure discovers that it cannot continue {e.g., the producer tinds the buffer tull), it does a wait on some condition variable, say, full. This action causes the calling process to block. It also allows another process that had been pieviously prohibited from entering the monilor to enter now. This other process, for example, the consumer, can wake up iis sleeping partner by doing a signal on the condition variable that its partner is waiting on. To avoid having two active processes in the momtoi at the same lime, we need a rule telling what happens after a signal. Hoare proposed letting the newly awak- awakened process run, suspending the other one. Bnnch Hansen proposed finessing the problem by requiring that a process doing a signal must exit the monitor immediately. In other wofds, a signal statement may appear only as the final statement in a monitor procedure We will use Brinch Hansen\ proposal because it is conceptually simpler and is also easier to implement. If a signal is done on a condition variable on which several processes are waiting, only one of them, determined by the system scheduler, is revived As an aside, there is also a third solution, not proposed by either Hoare or Brinch Hansen. This is to let the signaler continue to run and allow the waiting process to start running only after the signaler has exited the monitor. Condition variables aie not counters. They do not accumulate signals for later use the way semaphores do.

Thus if a condition variable ;s signaled with no one waiting on it, the signal it, lost forever. In other words, the wait must come before the signal. This rule makes the implementation much simpler. In practice it is not a problem because it is easy to keep track of the state of eacli process with vari- variables, if need be. A process that might otherwise do a signal can see that this operation is not necessary by looking at the variables.

SEC 23 INTERPROCESS COMMUNICATION 117 A skeleton ot" the producer-consumer problem with monitors is given in Pig. 2-27 in an imaginary language, Pidgin Pascal. The advantage of using Pidgin Pascal here is that u is pure and simple and follows the Hoare/Bnncti Hansen model exactly. monitor Pniclu. condition integers procedur begin Vti Iff, end: function begin ifn "full empn.r Hint - ,V then ■Htm - I then ■ vnAVful'h I: signal (cjn/ reinuve integer, >unt - 0 then wait(e»i/?f' n = N- | then signal(/n/') end monitor; procedure producer, begin wbUe true do begin ProducfrConiumer-inserttut end end; procedure consumer- begin while true do begin item — PrttducerCon.su fiter ft end Figure 2-27. An outline of ihe producer-consumer problem w^ih Only one jnonjior protedure ai a lime i<; active The bulter ha, N slots.

118 PROCESSES AND THREADS CHAP. 2 You may be ihinking lhai ihe operations wail and signal look similar 10 sleep and wakeup, which we saw earlier had fatal race conditions. They are very simi- similar, but with one crucial difference: sleep and wakeup failed because while one process was trying to go to sleep, the other one was irying to wake it up. With monitors, that cannot happen. The automatic mutual exclusion on monitor pro- procedures guarantees that if. say. the producer inside a monitor procedure discovers that the buffer is full, it will be able to complete the wait operation without having lo worry about the possibility that the scheduler may switch to the consumer just before the wait completes The consumer *iH ne>i even be let into the monitor ai all until the wail is finished and the producer has been marked as no longer run- nable. Although Pidgin Pascal is an imaginary language, some real programming languages also support monitors, alihough no! always in ihe form designed by Hoare and Brinch Hansen- One sudi language is Java. Java is an object-on en ted language that supports user-tevet threads and also allows methods (procedures) to be grouped together into classes. By adding the keyword synchronized to a method declaration, Java guarantees that once any thread has started executing that method, no other thread will be allowed 10

start executing any other syn- synchronized method in that class. A solution to the producer-consumer problem using monitors in Java is given in Fig. 2-28. The solution consists of four classes. The outer class. Producert'on- sumer, creates and starts two threads, p and c The second and third classes, producer and consumer, respectively, contain the code for the producer and consuconsumer. Finally, the class our^monitor, is the monitor It contains two synchronized threads that are used for actually inserting items into the shared buffer and taking them out. Unlike in the previous examples, we have finally shown the full code of insert and remove here. The producer and consumer threads are functionally identical to their counter- counterparts in al! our previous examples. The producer has an infinite loop generating data and putting it into the common buffer. The consumer has an equally infinite loop taking data out of the common buffer and doing some fun thing with it. The interesting pan of this program is the class our^nvmitor, which contains the buffer, the administration variables, and two synchronized methods. When the producer is active inside insert, it knows for sure that the consumer cannot be active inside remove, making it safe to update the variables and the buffer without fear of race conditions The variable count keeps track of how many items are in the buffer. It can take on any value from 0 through and including N - I The variable lo is the index of the buffer slot where the next item >s to be] etched. Similarly, hi is the index of the buffer slot where the next item is to be placed. It is permitted that lo ~ht, which means either that 0 items or N items are in (he buffer. The value of count tells which case holds. Synchronized methods in Java differ from classical monitors in an essential way: Java does not have condition variables. Instead, it offers two procedures.

SPr 2 1 INTERPROCESS COMMUNICATION 119 w<itt and iiottt\ that are the efluivafent of \lci'p and \\akeitp except that when they are used inside synchronized methods, they are not subject to race conditions. In theory, the method watt can be interrupted, which is what the code surrounding <l is all aboul. Java requires that the exception handling be made explicit. For our purposes, jus! imagine thai ga^to^sleep is the way lo go lo sleep. By making the mutual exclusion of critical regions automatic, monitors make parallel programming much less error-prone lhan wilti semaphores. Still, Ihey loo have some drawbacks. It is nol for nothing ihal our two examples of monitors were in Pidgin Pascal and Java instead of C. as are the other examples in this book As we said earlier, monitors are a programming language concept. The

compiler must recognise them and arrange tor the imitual exclusion somehow. C, Pascal, and most olher languages do not have monitois. sii it is unreasonable to expect their compilers to enforce any mutujil exclusion rules. In tact, how could the compiler even know which procedures were in moniiors and which were not? These same languages do not have semyphores either, t>ul adding semaphores is easy: All you need to do is add two short assembly code routines* to the library to issue the up and down system calls. The compilers do noi even have to know that they exiKt. Of course, the operating systems have to know about the sema- semaphores, but at least if you have a semaphore-based operating system, you can still write the user programs for it in C or C++ (or even assembly language it you are masochistic enough). With monitors, you need a language that has them huilt in Another problem with monitors, and ako with semaphores, is that thc>' wore designed for solving the mutual exclusion problem on one or more CPUs that all have access to a common memory. By putting the semaphores in the shared memory and protecting them with TSL instructions, we fan avoid races. When we go to a distributed system consisting of multiple CPUs, each wiih us own private memory, connected by a local area network, these primitives become inapplica- inapplicable. The conclusion is that semaphores are too low level and monitors are not usahle except in a tew programming languages. Also, none of the priorities pio- vide for information exchange between machines. Something else in needed. 2.3.8 Message Passing That something else is message passing This method of interprocess com- communication uses two primitives, send and receive, which, like semaphores and unlike monitors, are system calls rather than language constructs As such, they can easily be put into library procedures, sucli as sendfdestination, Smessage); receive{source, Smessage}; The former call sends a message to a given destination and the latter ■

10 PROCESSES AND THREADS public class Produce rConsumer { static finafini N s= 100, //constant giving the butter size staijc producer p = new producer]); // insinuate a new producer thread static our_rtiomtor mon => new our_monitor!), // .nsiannate a new monitor public static void mainlStnng args[l) I p siart(), // start the producer ihread c.sts/i(j, // start lhe consumer thread) staifc class producer extends Thread (public void run(} \ll run method contains the threed code ini item, white Itrue) I // producer loop Item =■ produce.. item(): mon insert] Jem).) 1 private ini produce -item() I ■ ! //

actuary produce while (true) I //consumt (tern = mon remove)), consume_uem (item). static cfaas our_mon"or (// ihls Is a monitor private mi bufferf] - new int(N], private int couni = 0. Io = 0, hi = 0, // counters and Indices public synchronized void insert(ini vaf) (If (couni ==N) go_io_sieepl), II if Ihe buffer Is fuP, go to slee; butter [hi) = vai; II insert an Item fnio the buffer hi = (hi 11) % N, II slot to piaoe next Hem in couni = couni + 1, // one more Hern in ihe buffer now If (count * 1) notify!). // If consumer was Sleeping, wake it) public synchronized int remove(j I if (couni".- 0) go_to_sieep(1, // if ihe buffer Is empty, go to s vai = Duffer [io{, // letch an Item from the buffer io = (io + 1) % N, //slot to letch next Jerri froin count» count — 1. // one lew items in the buffer tf (count»» N — 1) notlfy(jh tt >f producer was sleeping, wake H l irivaievoidgo_to_sieep()I trylwaiti),! catch (inierru pled Exception enc)]),) Hgure 2-28. A solution io [he producer-consumer problem in Java

SEC 2 3 INTERPROCESS COMMUNICATION 121 a message from a given source (or from ANY, it Ihe receiver does not care)- If no message ts available, ihc receiver can block until one arrives Alternatively, it can return immediately wiih an error code. Design Issues for Message Passing Systems Message passing systems have many challenging problems and design issues thai do not arise with semaphores or monitor1;, especially if the communicating processes are on different machtnes connected by a neiwork. For example, mes- messages can be !osi by ihe neiwork. To guard agamsl lost messages, the sender and receiver can agree thai as soon as a message has been recetved, the receiver will send hack a special acknowledgement message. If the sender has not recetved the acknowledgement within a certain time interval, tt retransmits the message. Now consider whai happens if the message itself i.s received correctly, but the acknowledgement is lost The sender will retransmit the message, so the receiver wifl get it twice. It is essential that the receiver be ahle to distinguish a new mes- message from the retransmission of an old one. Usually, this prohlem is solved hy putting consecuiive sequence numhers in each original message. It' the receiver gets a message bearing the same sequence numher as the previous message, it knows that the message is a duplicate that can be ignored. Successfully commun- communicating in the face of unreliable message passing is a major part of the study of computer networks. For more information, see (Tanenhaum, 1996) Message systems also have to deal with the question of how processes are named, so that the process specified in a send or receive call is unambiguous.

Authentication is also an issue in message systems; how can the client tell that he is communicating with the real file server, and not with an imposter? At the other end of ihe spectrum, there are also design issues thai are impor- lant when the sender and receiver are on the same machine. One ofthe.se is per- performance. Copying messages from one process to another is always slower than doing a semaphore operation or entering a monitor. Much work has gone into making message passing efficient. Chenton A984), for example, suggested limit- limiting message size to whai will fit in the machine's registers, and then doing mes- message passing using the registers. The Producer-Consumer Problem with Message Passing Now lei us see how the producer-consumer problem can be solved with mes- message passing and no shared memory. A solution is given in Fig. 2-29. We assume thai all messages are the same size and thai messages sent but not yet recetved are buffered automatically by the operating system. In this solution, a lotal of /Vmes- /Vmessages is used, analogous to the N slots in a shared memory buffer. The consumer

122 PROCESSES AND THREADS CHAP. 2 starts out hy sending jV empty message has an item to give to the consumer. ,t full one In this way, the total number oi in tittie. so they can be stored If Ihe produo full, waning for the consurt ; back If ihe con: ks faster than ihi ihe prods ilie producer. Whenever ihe producer s ari empty messiige and sends hack a essages in the system remains constant ount of memory known in advance, onsumcr. all Ihe messages will end up r will he blocked- wailing fbr an empty tei, (hen the reverse happens-, dll thy messages will be empties wailing for the producei 10 till them up. the consumer will he blocked, waiting for a full message. * number of slots in the buffer * void producer(void) message m; /„ message buffer */ wnile (TRUE} { item = produce __ item (}; /, generaie something to puf in buffer receive)consumer, Sm); /* wait for an empty to arrive */ build .message(Sm, item); * construct a message to send */ send(consumer, Sm}; * send jtem to consumer */ void consumer(void) int item, i; message m; for (i =0; i < IsT 1++} send{producer, ■ while (TRUE) { receivefproducer. Sm}; item = extract. item(Sm); sendfproducer, Sm}; consume item {item); Sm), /• . /* send N empties */ gel message containing item extract item from message */ send back empty reply */ do something wilh the item */ Figure 2-29. The prudincr-mnsunier piohlem with Vriess^e- Many variants are possible with message passing For starters, [et us look at how messages are addressed One way is to assign each process a unique address and have messages he addressed to

processes. A different way is to invent a new data structure, called a mailbox. A mailbox is a place to buffer a certain number of messages, typieafly specified when the mailhox is created. When mailhoxes

Sf-X 2 3 fNTERPROCESS COMMUNtCATfON 123 are used, the address parameters in the send and receive calls are mailboxes, noi processes. When a process tries to send io a mailbox thai is fun, it us suspended untif a message is removed from that mailbox, making room for a new one. For the producer-consumer problem, boih the producer and consumer would creaie maifboxes large enough io hold N messages. The producer would send messages containing data to the consumer's mailbox, and ihe consumer would send empiy messages to the producer's mailbox. When mailboxes are used, the buffering mechanism is clear: the destination mailbox holds messages (hat have been sent io the destination process but have not yet been accepted. The oiher exireme from having mailboxes is to eliminaie all huffering. When ihis approach is followed, if ihe send is done hetore ihe receive, the sending proc-process is blocked until ihe receive happens, at which time the message can be copied directly from the sender io ihe receiver, with no intermediate huffering. Simi- Similarly, if the receive jj, done first, the receiver is blocked until a send happens. This strategy is often known as a rendezvous, li is easier io impfemeni ihan a buffered message scheme but is less flexible since ihe sender and receiver arc- forced to run in [ockstep. Message passing is commonly used m parallel programming systems. One well-known mes-sage-passing system, for example, is MPl (Message-Passing Interface). It is widely used for scientific computing. For more information about it, see for example (Gropp et al., 1994; and Snir ei al., 1996). 2.3.9 Barriers Our last synchronization mechanism is intended for groups of processes raiher than two-process producer-consumer type situations. Some applications are divided into phases and have ihe rule thai no process may proceed into the nexi phase until all processes are ready to proceed io ihe next phase. This behavior may be achieved by placing a barrier at the end of each phase. When a process reaches the barrier, it is blocked until all processes have reached ihe harrier. The operation of a barrier is illustrated in Fig. 2-30. In Fig. 2-30<a) we see four processes approaching a barrier. What ihis means is that they are just computing and have not reached the end of ihe current phase yet. After a whife. the firsi process finishes af! the computing required of it during the first phase. It then executes the barrier primitive, generally hy calling a library procedure.

The process is ihen suspended. A filtfe fater, a second and then a third process finish the first phase and also execute the barrier primitive. This situation is iffusiraied in Fig. 2-30<b). Finaffy, when the fast process. C. hits the barrier, aff ihe processes are released, as shown in Fig. 2-30(c). As an example of a problem requiring harriers, consider a typical relaxation problem in physics or engineering. There is lypicaffy a malrix lhai contains some initial values. The vafues might represem temperatures al various points on a

PROCtSSES AND THREADS © © © Figure 2-30. Us. ot processes but ore bio barrier, all ofiheni art s approathing .1 h.imi-i sheel of metal. The idea might be io calculate how long it lakes tor ihe effeci of a flame placed ai one corner 10 propagate throughout ihe sheet Starting with ihe current values, a trans formation is applied io the mairix io get the second version of ihe maim, for example, hy applying ihe laws of ihermo dynamics io see whai all ihe lemperaiures are AT later Then ihe processes is repeated over and over, giving ihe temperatures al ihe sample points as a function of time as ihe sheet heats up. The algorithm thus produces a series of mairices over ttme. Now imagine ihai ihe mainx is very large (say, 1 million hy 1 million), so thai parallel processes are needed (possibly on a muliiprocessor) io speed up the calcu-calculation. Differenl processes work on different parts of ihe matrix, calculating the new matrix efement-s from ihe old ones according io the ldws of physics. How- However, no process may siart on iteration n + 1 until Iteration n is complete, ihai is, unli! alf processes have finished iheir current work The way io achieve this goal is to program each process io execuie a barrier operation afier it has finished its part of ihe curreni iieraiton. When all of iheni are done, the new matrix (the input to ihe nexi iteration) will be finished, and all processes will be simultaneously released io start ihe nexi iteration. 2.4 CLASSICAL IPC PROBLEMS The operating systems liiera widely discussed and analyzed u following seciions we will exam are is full of iniere* ,inga variety ofsyi le three of ihe beiH iitig prohlems thai have been chronizaiion melhods. In ihe r-known problems.

SEC 2 4 CLASSICAL IPC PROBLEMS 125 2.4.1 The Dining Philosophers Problem In 1965, Dijkstra posed and solved a synchronization problem he called the dining philosophers problem. Since thai time, everyone inventing yet anoiher synchronizalion primitive has fell obligated to demonstrate how wonderful Ihc new primitive is hy showing how elegantly il solves the dining

philosophers prob- problem The problem can be stated quite simply as follows. Five philosophers are seated around a circular table. Each philosopher ha.s a plate of spaghetti. The spaghetti is so slippery lhat a philosopher needs iwo forks to eat it. Between each pair or plates is one fork. The layout of the iahle is illustrated in Fig. 2-31. Figure 2-31. Lunch lime in the Philosophy Depanmen! The life of a philosopher consists of alternate periods of eaiing and thinking. (This is something of an abstraction, even tor philosophers, bui ihe other activities are irrelevant here.) When a philosopher gels hungry, she tries io acquire her left e at a time i ith d If fl ii iwo forks. The key question is: Can you write a program for each philosopher that does what it is sup- supposed to do and ne requirement is right fork, one at a time, in either order. If successful in acquiring two for eais for a while, then puts down the forks, and continues to think. The 1 :stion is: Can you write a program for each philosopher that does what u is si posed io do and never gels stuck" {II has been pointed out lhat the iwo-fonc requirement is somewhat artificial; perhaps we should switch from Italian food to Chinese food, substituting rice for spaghetti and chopsticks for forks.| Figure 2-32 shows the obvious solution. The procedure lake-fork wails until the .specified fork is available and then seizes 11, Unfortunately, the obvious solu- solution is wrong. Suppose thai all five philosophers take their left forks simultane- simultaneously. None will be able to lake their nghi forks, and ihere will be a deadlock We could modify the program so thai after taking ihe left fork, the program checks io see if the nght fork is available, ff u is not, the philosopher puts down

6 PROCESSES AND THREADS CHAP : #def.ne N 5 '* number of philosophers */ voW Phiiosopherlint.) '* '■ PH-iosopher number, from 0 to 4 ♦/ while {TRUE} { , ■ , tl^ink(y I* philosopher is thrnkmg */ iake_foik(iV '* take left fork *' fakelfork{(i+1) % N): /* iake right fork; % is modulo operator- / eat( )■ /* yum-yum, spaghetti */ put j'ork(i)- ;* Put teft fork back °" the rable * Put_fork((i+1) % N); /* Put right fork back on the fable */ Figure 2-.12. A nonsoluiion 10 ihe dining philosophers problem ihc left one, wails for some time, and then repeals ihe whole process This propo- .sal too, fails, although for a different reason. With a lit lie bit of bad luck, all the philosophers could start the algorithm simultaneously, picking up iheir left forks, seeing ihai iheir righi forks were not available, putting down their lett forks, wail- wailing, picking up their left forks again simultaneously, and so on, forever. A situa- situation like this, in which all the programs continue to run indefinitely but fail

to make any progress is called starvation. (It is called starvation even when the problem does not occur in an Italian or a Chinese restaurant.) Now you might think, "If the philosophers would just wait a random time instead of the same lime after failing to acquire the right-hand fork, the chance thai everything would continue in lockstep for even an hour is very small." This observation is true, and in nearly ail applications trying again later is not a prob- send a packet at the same time, each one waits a random time and tries again; in practice this solution works fine. However, in a few applications one would prefer a solution that always works and cannot fail due to an unlikely series of random numbers. Think about safety control in a nuclear power plant. One improvement to Fig. 2-32 that has no deadlock and no starvation is to protect the five statements following the call to think by a binary semaphore. Before starting to acquire forks, a philosopher would do a down on mulex. After replacing the forks, she would do an up on mutex. From a theoretical viewpoint, this solution is adequate. From a practical one, it has a performance bug: only one philosopher can be eating at any instant. With five forks available, we should be able to allow two philosophers to eat at the same time. The solution presented in Fig. 2-33 is deadlock-free and allows the maximum parallelism for an arbitrary number of philosophers. It uses an array, state, to keep track of whether a philosopher is eating, thinking, or hungry (trying to acquire forks), A philosopher may move only into eating state if neither neighbor

CLASSICAL IPC PROBLEMS #define N 5 #define LEFT (t+l #define RIGHT A+ #deflne THINKING 0 #deflne HUNGRY 1 #define EATING 2 typedef Int semaphore; Int state[N]; semaphore mutex = 1; semaphore s[N]; void phllosopherfint I) while (TRUE) (thmk(); take_forks(i); eat(); put_forks(i); /* number of philosophers */ /* number of i's left neighbor */ /* number of I's right neighbor */ /* philosopher is thinking */ /* philosopher is trying to gel forks */ /* philosopher is eating •/ /* semaphores are a special kind of int */ /* array lo keep track of everyone's state */ /* mutual exclusion for critical regions */ /* one semaphore per philosopher */ /* i: philosopher number, from 0 to N-1 */ /* repeat forever */ /* philosopher is thinking */ /* acquire two forks or block */ /* yum-yum, spaghetti */ /• pul bolh forks back on table */ void take_forks(int i) (down(&mutex); state[i] = HUNGRY; test(i); up (& mutex); down(&s[i]); /* i: philosopher number, from 0 to N-1 */ /* enter critical region */ /* record fact that philosopher I is hungry */ /* try 10 acquire

2 forks */ /* exit critical region */ /* block if forks were not acquired */ down(&mutex); state[>] = THINKING; test(LEFT); lest(RIGHT); up (& mutex); /• i: philosopher number, from 0 to N-1 */ /* enter critical region */ /* philosopher has finished eating */ /* see if left neighbor can now eat */ /* see if right neighbor can now eat */ /• exii critical region */ void test(i) if (state[il == HUNGRY && stale[LEFT] != EATING state[l] = EATING; up(&s[i]); /* i: philosopher number, from 0 to N-1 */ ;lale[RIGHT] != EATING) | Figure 2-33. A whilton to ihe drntng philosopher problem

128 PROCESSES AND THREADS CHAP 2 is eating. Philosopher f s neighbors are defined by the macros LEFT and RIGHT. In other words, if (is 2. LEFTfr I and RIGHT is 3 The program uses an array of semaphores, one per philosopher, so hungry philosophers can block if rhe needed forks are busy. Note that each process runs the procedure philosopher as its main code, but the other procedures, take_Jorks, put_torks, and lest are ordinary procedures and not separate processes. 2.4.2 The Readers and Writers Problem The dining philosophers problem is useful for modeling processes that are competing for exclusive access to a limited number of [esources. such as I/O devices Another famous problem is ihe readers and wriiers problem (Courtois et al., 1971), which models access to a database. Imagine, for example, an anline reservation system, with many competing processes wishing to read and write it. ft is acceptable to have multiple processes reading the database at the same time, but if one process is updating (writing) the database, no other processes may have access to the database, not even readers. The question is how do you program the readers and the writers'1 One solution is shown in Fig 2-34 in this solution, the first reader to gei access to the datahase does a down on the semaphore db. Subsequent readers merely increment a counier. n As readers leave, they decrement the counter and the last one out dues an up on the semaphore, allowing a blocked writer, if rhere is one, to get in. The solution presented here implicitly contains a subtle decision tha' is worth commenting on. Suppose that while a reader is using the database, another reader comes along. Since having two readers ai the same timo is not a problem, ihe second reader is admitted. A third and subsequeni readers can aNo be admitted it they come along. Now suppose that a writer comes along. The writer cartnoi be admitted io the database, since writers must have exclusive access, so the writer is suspended Later, additional readers show up. As long as at least one reader is still active, subsequent readers are admitted. As a consequence of

this strategy, as long as there is a steady supply of readers, they will all get in as soon as they arrive. The writer will be kept suspended until no reader is present. If a new reader arrives, say, every 2 seconds, and each reader takes 5 seconds io do its work, ihe wriier will never get in. To prevent ihis siiuation, the program could be written slightly differently, when a reader arrives and a writer is waiting, the reader is suspended behind the writer instead of being admitted immediately In this way. a writer has to wan Tor readers that were active when it arrived to finish bui docs not have to wait for readers that came along after it The disadvantage of this solution r.s that ii achieves less concurrency and thus lower performance. Courtois et al. present a solution that gives priority to writers. For details, we refer you to the paper.

CLASSICAL IPC PROBLEMS typedef int semaphore; semaphore mutex = 1, semaphore db = 1 int re = 0; void reader(void) I while (TRUE) { down(&mutex); re = re + 1; if {rc== i)down(&db); up{&mirtex>; read_data_base(); down{&mutex); re = re - 1; if (rc==O) up(&db); up(&mutex); use_data_read(); /* use your imagination */ /* controls access to 're' */ /* controls access to the database */ /* # of processes reading or wanting to */ '* repeat forever */ '* get exclusive access to 're' */ '* one reader more now •/ '* if this is the first reader... */ '* release exclusive access to 're' */ '* access the daia */ /* get exclusive access to 're' */ '* one reader fewer now */ '* if this is the last reader... */ '* release exclusive access to 're' */ /* noncritical region */ while (TRUE) (think_up_data{); down(&db); write_data_base{); up(&db). /* repeat forever ■/ /* noncriiical region */ /* get exclusive access */ /* update the data */ /* release exclusive access Figure 2-34. A solution 2.4.3 The Sleeping Barber Problem Another classical IPC problem takes place in a barber shop. The barber shop has one barber, one barber chair, and n chairs for waiting customers, if any, to sit on. If there are no customers present, the barber sits down in the barber chair and falls asleep, as illustrated in Fig. 2-35. When a customer arrives, he has to wake up the sleeping barber. If additional customers arrive while the barber is cutting a customers hair, they either sit down (if there are empty chairs) or leave the shop (if all chairs are full}. The problem is to program the barber and the customers

 without getting into race conditions. This problem is similar to various queueing situations, such as a

multipenxin helpdesk with a computerized call waiting syv lem for holding a limited number of incoming calls. Figure 2-35. The -Jeeping barhsir Our solution uses three semaphores: cu.homers, wh mers {excluding the customer in the barher chair, who is number of barbers @ or 11 who are idle, waiting for cusi is used for mutual exclusion- We also need a variable, w the waiting customers, it is essentially a copy of cus h counts waiting custo- ot waiting), barbers, the mers. and mute\. which iring, which also counts a. The reason for hav- having waning is that theie is no way to read the current value of a semaphore In this solution, a customer entering the shop has to count the numher or waiting cus- customers, if it is less than the number or chairs, he stays; otherwise, he leaves. Our solution is shown in Fig. 2-36 When Ihe barber shows up for w,»rk in the morning, he executes the procedure barber, causing him to block on the sema- semaphore customers because it is initially 0. The barber then goes to sleep, as. shown in Fig. 2-35- He stays asleep until the first customer show*, up.

CLASSICAL IPC PROBLEMS #define CHAIRS 5 typedef int semaphore; semaphore customers = 0; semaphore barbers = 0; semaphore mutex = 1, int waiting = 0; void barber(void) I white (TRUE) | down{&customers); down(&mutex), waiting = waiting - 1; up(&barbers); up(&mutex), cut_hair(); /* # chairs for waiting customers */ /* use your imagination */ /* # of customers waiting for service */ /* # of barbers waiting for customers */ /* for mutual exclusion */ /* customers are waiting {not being cut) */ /* go to sleep if # of customers is 0 */ /* acquire access to "waiting' */ /* decrement count of waiting customers /* one barber is now ready to cut hair */ /* release 'waiting' */ /* cut hair (outside critical region) */ dcustomer{void) down(&muiex)' if (waiting < CHAIRS) { waiting = waiting + 1, up{&customers); up{&mutex); down(&barbers); get_haireut();) else { up(&mutex), } /* enter critical region */ /* if there are no free chairs, leave */ /* increment count of waiting customers */ /* wake up barber If necessary */ /* release access to "waiting' */ /* go to sleep If # of free barbers Is 0 */ A be seated and be serviced */ /* shop Is full; do not wai */ Figure 2-36. A <.oluuo K sleeping barber pri>htenl When a customer arrives, he executes lusfomer, starting by acquiring mulex to enter a critical region. If another customer enters shortly thereafter, the second one will not be able to do anything until the first one hat, released mutex. The cus- customer then checks to .see if [he number of waiting customers is less than the number of chairs, if not, he releases mutex and leaves without a haircut. if there

is an available chair, the customer increments the integer variable, waiting. Then he does an up on the semaphore customers, thus waking up the

132 PROCESSES AND THREADS CHAP 2 barber. At this point, the customer and barber are both awake. When the custo- customer releases mutex, the barber grabs it, does some housekeeping, and begins the haircut. When the haircut is over, the customer exits the procedure and leaves the shop Unlike our earlier examples, there is no loop for the customer because each one gets only one haircut. The barber loops, however, to try to get the next custo- customer. If one is present, another haircut is given. If not, the barber goes to sleep. As an aside, it is worth pointing out that although the readers and writers and sleeping barber problems do not involve data transfer, they are still belong to the area of IPC because they involve synchronization between multiple processes. 2.5 SCHEDULING When a computer is muitiprogrammed, it frequently has multiple processes competing for the CPU at the same time This situation occurs whenever tw.i or more processes are simultaneously in the ready state. If only one CPU is a\ail- able, a choice has to be made which process to run next The part of the operating system that makes the choice is called the scheduler and the algorithm it uses is called the scheduling algorithm. These topics form the subject matter of the tol- lowing sections. Many of the same issues that apply to process scheduling also apply to thread scheduling, although some are different Initially we will focus on process scheduling. Later On we will explicitly look at thread scheduling. 2.5.1 Introduction to Scheduling Back in the old days of batch systems with input in the form of card images on a magnetic tape, the scheduling algorithm was simple; just run the next job on (he tape. With timesharing systems, the scheduling algorithm became more com- complex because there were generally multiple users waiting for service. Some main- mainframes still combine batch and timesharing .service, requiring the scheduler to decide whether a batch |ob or an interactive user at a terminal should go next. 'As an aside, a batch job may be a request to run multiple program;, in succession, but for this section, we will just assume it is a request to run a single program.) Because CPU time is a scarce resource on these machines, a good scheduler can make a big difference in perceived performance and user satisfaction. Conse- Consequently, a great deal of work has gone into devising clever and efficient schedul- scheduling algorithms. With the advent of personal computers, the situation changed in two ways. First, most of the time there is

only one active process. A user entering a docu- document on a word processor ts unlikely to be simultaneously compiling a program in the background. When the user types a command to the word processor, the

SEC 2.5 SCHEDULING 133 scheduler does not have to do much work to figure oul which process to run—the word processor is the only candidate. Second, computers have goUent so much taster over the years that the CPU is raiely a scarce resource any more. Most programs for personal computers are limited by the rate jt which the user can present input (by typing or clicking), not by the rale the CPU can process it. Even compilations, a major sink of CPU cycles in the past, take jum a few seconds at most nowadays Even when two pro- programs are actually running at once, such as a word processor and a spreadsheet, it hardly matters which goes first since the user is probably waiting for both of them to finish. As a consequence, scheduling does not matter much on simple PCs. [Of course, there arc applications that practically eat the CPU alive, rendering one hour ot high-re.solution video may require industrial-strength image processing on each of 108.000 trames in NTSC (90,000 in PALI, but these applications are the exception rather than the rule.I When we turn to high-end networked workstations and servers, the situation changes. Heru multiple processes often do compete for the CPU, so scheduling matters again. For example, when the CPU has to choose between running a process that updates the screen after a user has closed a window and running a procc-ss that sends out queued email, it makes a huge difference in the perceived respon.se. if closing the window were to take 2 sec while the email was being sent, the u.ser would probably regard the system as extremely sluggish, whereas having the email delayed by 2 sec would not even be noticed, in this case, proc- process scheduling matters very much. In addition to picking the right process to run, the scheduler also has to worry about making efficient use of the CPU because process switching is expensive. To start with, a switch from user mode to kernel mode must occur. Then the state of the current process must be saved, including storing its registers in the process lable s,o they can be reloaded later In many systems, the memory map (e.g., memory reference btts in the page table) must be saved as well. Next a new proc- e must be selected by running the scheduling algorithm. After that, the MMU must be reloaded with the memory map of the new process. Finally, the new process muM be started, in addition to all that, the process switch usually invali- invalidates the entire memory cache, forcing it to be

dynamically reloaded irotn the main memory twtce (upon entering the kernel and upon leaving it). All in all, doing loo many process s,wnchei» per second can chew Up a substantial amount of CPU lime, so camion is advised. Process Behavior Nearly all processes ahernaie bursts of computing wiih idisk) I/O requests us Shown in Fig. 2-37. Typically lhe CPU runs for a while wiihoui stopping, lhcn a system call is made to read from a file or write to a file. When the system call completes, the CPU computes again until ii needs more data or has to write more

134 PROCESSES AND THREADS CHAP 2 data, and so on. Note that some I/O activities coum as computing. For example, when (he CPU copies bits to a video RAM to update the screen, it is computing, not doing I/O. because the CPU is- in use. I/O in this sense is when a process enters the hlocked state waiting tor an external device to complete its work. Figure 2-37. Bursls ACPLI-boundproces e wilh period1- at wailing tor t/O The important thing to notice about Fig. 2-37 is that some processes, such as the one in Fig. 2-37{a}, spend most of their time computing, while others, such as the one in Fig 2-37(b). spend most of their time waiting for I/O. The former are called compute -bound; the latter are called I/O-bound. Compute-bound processes, typically have long CPU bursts and thus infrequent I/O waits, whereas 1/O-bound processes, have short CPU bursts and thus frequent I/O waits. Note that the key factor is the length of the CPU bursi. not the length vt the I/O burst. 1/O- bound processes are I/O bound because they do not compute much between I/O y requests, not because they have especially long I/O requ time to read a disk block no maiter how much or how little time the data after they arrive. It is worth noting that as CPUs get faster, processes tend bound. This effect occurs because CPUs are improving much As a consequence, the scheduling of I/O-bound processes is li b i portant subject in the futu process wants to run, it should get and keep the disk busy. When to Schedule takes the s akes to process get more 1/O- ter than disks, kely to become a f /O y The hasic idea here is that if an 1/O-bound chance quickly sc it can issue its disk requesi A key issue related tu scheduling is when ki make scheduling decisions U ns out that there are a variety of situations in which scheduling is needed. Firsi. en a new process is created, a decision needs tti be made whether to run the

-SEC 2 i SCHEDULING '35 parent process or the child process. Since hoth processes are in ready state, it is a normal scheduling decision and it can go

either way, that is, the scheduler can leg- legitimately choose to run enher tht parent of the child next. Second, a scheduling decision must be made when a process exits. That proc- process can no longer run (since it no longer exists), so some other process must be chosen from the set of ready processes If no process is ready, a system-supplied idle process is normally rui) Third, when a process blocks on I/O, on a semaphore, or for some other rea- on another process has to be selected to run Sometimes the reason for blocking may play a role in the choice. For example, if A is an important process and it is waiting f(i B t txit its triiiu I region letting B run next will allow it to exit its' critical region and thus let A continue. The trouble, however, is that the scheduler enenlly does not have the necessary information to take this dependency into account Fourth, when an I/O interrupt occurs, a scheduling decision may be made. If the interrupt came from an I/O device that has now completed its work, some process that was blocked waiting for the I/O may now be ready to run. It is up to the scheduler to decide if the newly ready process should be riiii. if the process that was running at the time of the interrupt should continue running. 01 it some third process should run If a hardware clock provides, periodic interrupts at 50 Hz, 60 Hz, or some other frequency, a scheduling decision can be made at each clock interrupt or at every fr-th clock interrupt Scheduling algorithms can be divided into two categories with respect to how they deal with clock interrupts. A nonpreemptive scheduling algorithm picks a process to run and then just lets it ron until ii blocks (either on I/O or wailing for another process) or until it voluntarily releases the CPU- Even if it runs tor hours, H will not be forceably suspended. In effect, no scheduling decisions are made during clock interrupts. After clock interrupt proc- processing has been completed, the process that was running before the interrupt is always resumed. In contrast, a preemptive scheduling algorithm picks a process and lets it run for a maximum of some fixed time. If it is .still running at the end of the time interval, it is suspended and the scheduler picks another process to run Of one is available). Doing preemptive scheduling requires having a clock interrupt occur at the end of the time interval to give conirol of the CPU hack to the scheduler If no clock is available, nonpreemptive scheduling js the only option. Categories of Scheduling Algorithms Not surprisingly, in different environments different scheduling algorithms are needed. This situation arises because different application areas (and different kinds of operating systems) have different goals. In other words, what the sched-

136 PROCESSES AND THREADS CHAP. ? uler should optimize tor is not the same in all systems.. Three environments worth distinguishing are I. Batch. 2 Interactive 3 Real time. In batch systems, there are no users impatiently waiting at their terminal1; for a quick response. Consequently, nonpreemptive algorithms, or preemptive algo- algorithms with long time periods for each process are often acceptable. This' approach reduces process switches and thus improves performance. In an environment with interactive users, preemption is essential to keep one process from hogging the CPU and denying service to the others Even it no process intentionally ran forever, due to a program bug, one process miglit shut out all the others indefinitely. Preemption is needed to prevent this behavior In systems with real-time constraints, preemption is, oddl> enough, sometimes not needed because the processes know that they may not run for long periods of time and usually do their work and block quickly. The difference with interactive systems is that real-time systems run only programs thai are intended to further the application at hand Interactive systems ate general purpose and may run arbi- arbitrary programs that are not cooperative or even malicious. Scheduling Algorithm Goals In order to design a scheduling algorithm, u is necessary to have some idea i>f what a good algorithm should do Some goals depend on the environment (batch, interactive, or real time), but there are also some that are desirable in nil cases Some goals are listed in Hg. 2-38. We will discuss these in turn below Under all circumstances, fairness is important. Comparable processes should get comparable service. Giving one process much more CPU tune than an equiv- equivalent one is not fair. Of course, different categories ot processes may be treated very differently. Think of safety control and doing die payroll at a nuclear reactor's computer center. Somewhat related lo fairness is enforcing the system's policies. If die local policy is that safety control processes get to run whenever they want to. even it it means the payroll is 30 sec late, the scheduler hat, to make sure this poiicv is enforced- Anothei general goal is keeping all parts ot the ■,> stem busy when p(issitile. If the CPU and all the I/O devices can be kep(running alt the lime, more wnrk gen, done per second than if some of the components are idle In a batch '.y.icni, lor example, the scheduler has control of which jobs are brought into memory (o run. Having lome CPU-bound processes and some 1/O-bound processes in memory together is a better idea than first loading and running all the CPU-bound jobs and

All systems Fairness - giving each process a lair share ol the CPU Policy enlorcemeni -seeing that slaled policy is earned out Balance - keeping all parts ol the system busy Batch systems Throughput - maximize (Obs per hour Turnaround hme - minimize time between submission and lerminaUon CPU utilization - keep the CPU busy all the lime Interactive systems Response Hme - respond lo requests quickly Proportionality - meel users' expectations Real-time systems Meeting dead&nes - avoid losing daia Predictability ■ avoid quaflty degradaiion in multimedia systems Figure 2-38. Some goalt of ihe vchediirnij; algorithm under differsni araimsunces. then when ihey are finished loading and running ail ihe 1/O-bound jobs. If the latter strategy is used, when the CPU-bound processes, are running, they will fight for the CPU and the disk wtll be idle Later, when the I/O-hound jobs come in. they will tight for the disk and the CPU will be idle Betier to keep tbe whole sys- system running at once by a careful mix ot processes. The managers of large computer centers lhat run many batch jobs typically look at three metrics to see how well their systems are performing: throughput, turnaround time, and CPU utilization. Throughput is tlie number tit jobs per hour that the system completes. All things considered, finishing 50 jobs per houf is beuer than finishing 40 jobs per bour Turnaround time is the statistically average time from the moment that a baich job is submitted until tbe moment if is completed. It measures how long the average user has to wait tor the output- Here the rule is: Small is Beautiful. A scheduling algorithm that maximizes throughpui may not necessarily minimjze turnaround time. For example, given a mix of short jobs and long jobs. a scheduler that always ran short lobs ;md never ran long lobs nnaht achieve an excellent throughput (many short jobs per hour) but at the expense oi a terrible turnaround time for the long jobs. If short jcihs kept arriving at a stead) rate, the long jobs might never run, milking ilw mean turnaround tune inHrtUe while achieving a high throughput CPU utilization is also an issue with batch systems heciuse tin ihe big main- mainframes where batch systems run, ihe CPU is still a major expense. Thus compute center managers teel guilty when it is not running all the time Actually though, this is noi such a good metric. What really matters is how many jobs per hour

 come out of the system (throughput) and how long 11 lakes lo gel a job back (tur- (turnaround nine). Using CPU utilization as a metric is like rating cars based on how many times per hour the engine lurns over. For interactive systems, especially timesharing

sysieins and servers, different goals apply. The most important one is 10 minimize response time, ihai is ihe time between issuing a command and gelling ihe result. On a personal computer where a hackground process is running (for example, reading and storing email from ihe network), a user request to start a program or open a file should lake pre- precedence over the background work Having all interactive requests go first will be perceived as good service. A somewhat related issue is what might be called proportionality. Users have- an mheicnt (but often incorrect) idea of how long things should take. When a lequest that is perceived as complex takes a long time, users accept that, but when a request that i.s perceived as simple takes a long time, Users gel irritated. For example, if clicking on a icon ihat calls up an internet provider using an ana- analog modem takes 45 seconds to establish a connection, the user Will probably accept that a.s a fact of life. On the other hand, if clicking on an icon that breaks the connection takes 45 seconds, the u.ser will probably be swearing a blue streak by ihe 30-.sec mark and frothing at the mouth by 45 sec This behavior is due to the common user perception thai placing a phone call and getting a connection is .supposed to take a lot longer than just hanging up In some cases [such as this one), the scheduler cannot do anything about tht response time, but in other cases it can, especially when the delay is due to a poor choice of process order. Real-time systems have different properties than interactive systems, and thus different scheduling goals They are characterized by having deadlines that must or al least should be met. For example, if a computer is controlling a device that produces data at a regular rate, failure to run the data-collection process on time may result m lost data. Thus the foremost need in a real-time system is meeting all (or most) deadlines. In some real-time systems, especially those involving multimedia, predictabil- predictability is important. Missing an occasional deadline is not fatal, but if the audio proc- process runs too erratically, ihe sound qualily will deteriorate rapidly Video is also an issue, but the ear is much more sensitive to jitter than the eye. To avoid this prob- problem process scheduling must be highly predictable and regular- We will siudy balch and interaelive scheduling algorithms in (his chapter hut defer mosi of our study of real-time scheduling until we come to multimedia operaling systems in Chap. 7. 2.5.2 Scheduling in Batch Systems It is now time to turn Irom general scheduling issues to specific scheduling algorithms. In this section we will look at algorithms used in balch systems. In the following ones we will examine interactive and real-time systems. It is worth

StC. 2.5 SCHEDULING '->' pointing oul that some algorithms are used in both batch and liiteraelive systems. We will study these later. Here we will focus on algorithms [hat are only suitable in balch systems. First-Come First-Served Probably [he simplest of all scheduling algorithms is nonprecmptive first-come first-served. Wilh this algorithm, processes are assigned the CPU in the order [hey request it. Basically, there is a single queue of ready processes. When [he first job enters the system from the outside in the morning, it is started immediately and allowed to run as long as it wants to. As other jobs come in, they are put onto the end of the queue. When [he running process blocks, the firsl process on the queue is run nexl. When a blocked process becomes ready, like a newly arrived job, it is put on the end of ihe queue. The great strength of this algorithm is that il is easy to understand and equally easy lo program. It is also fair in (he same sense [hat allocating scarce sports or concert tickets to people who are willing to stand on line starting a[2 A.M. is fair. With this algorithm, a single linked lisl keeps track of all ready processes. Pick- Picking a process to run jusl requires removing one from the front of the queue. Adding a new job or unblocked process just requires attaching it to (he end of the queue. What could be simpler? Unfortunately, first-come first-served also has a powerful disadvantage. Sup- Suppose tha(there is one compute-bound process that runs for 1 sec at a time arid many l/O-bound processes that use liltle CPU time but each have to perform 1000 disk reads to complete. The compute-bound process runs tor 1 sec, then it reads a disk block. All the I/O processes now run and start disk reads. When the compute-bound process gets its disk block, it runs for another 1 sec. followed by all Ihe l/O-bound processes in quick succes ion The net resull is tha(each l/O-bound proves gel [o read 1 block pel second and will take 1000 sec to finish W»h a scheduling algorithm that preempted the compute-bound process every 10 msec, the I/O-boun<i processes would finish in 10 sec instead of 1000 sec. and without slowing down the compute-bound process very much. Shortest Job First Now let us look at another nonpreemptive batch algorithm that assumes the run times are known in advance. In an insurance company, for example, people can predict quite accurately how long it will take to rnn a batch of 1000 claims, since similar work is done every day. When several equally important jobs are sitting in (he input queue waiting to be started, the scheduler picks the shortest job first. Look at Fig. 2-39. Here we find four jobs A, B, C, and D with run times of 8, 4, 4, and 4 minutes, respectively- By running them in that order, the

140 PROCESSES AND THREADS CHAP 2 turnaround time tor A is 8 minutes, for If l< 12 minutes, tor C is 16 minutes, and for D is 20 minutes lor an average of 14 minutes A C 0 B C D A Figure 2-W. An example of shortest job fir1.! ".chedulmg (d) Running four |obs m \tic original i)rdcr |b) Running them in shortest |ob first order Now let us consider running these four jobs; using .shortest job first, as shown in Fig. 2-3*«b). The turnaround times are now 4, 8, 12. and 20 minutes for an average of 11 minutes. Shortest job first is provably optimal. Consider the case of four jobs, with ran times of a. b, <, and d, respectively The first job finishes ;it time a. the second finishes at time a + b. and so on. The mean turnaround lime is Da + 1b + 2c + d)/A. It is clear that a contributes more to the average than the other times, so it should be the shortest job, with b next, then c, and finally d as the longest as. it affects only its own turnaround time The same argument applies equally well to any number of jobs. It is worth pointing out that shonest job fust is only optimal when all the jobs are available simultaneously. As a counterexample, consider five jobs. A through E. with run times of 2. 4. I, I. and I, respectively Their arrival times are 0. 0. 3, 3, and 3. Initially, only A or B can be chosen, since the other three jobs have not arrived yet. Using shortest job first we will run the jobs in ihe order A. B, C, D, E, for an avefage wail of 4.6. However, running (hem in ihe order B, C. P. E. A has an average wait of 4.4, Shortest Remaining Time Next A preemptive version of shortest job first is shortest remaining time next. W(th this algorithm, the scheduler always chooses ihe process whose remaining run lime is ihe shortest Again here, the run time has to be known m advance When a new job arrives, its total nine is compared to (he current process' remain- remaining time. It tbe new job needs less lime to finish lhan (he t:urrenl process, the current process is suspended and (be new job siarted This scheme allows new short jobs lo get good service Three-Level Scheduling From a certain perspective, batch systems allow scheduling at three dirrerenl levels, as illustrated in Fig. 2-40. As jobs arrive at the system, ihey are initially placed in on input queue stored on the disk. The admission scheduler decides

SEC. 2.5 SCHEDULING 141 which jobs 10 admit io the sysfem. The others are kept in (he inpui queue until they are selected. A typieal algorithm for admission control might be to look for a mis of coinpiiie-bound jobs and I/Obound jobs. Alternatively, short }obs could be admiiieti quickly where;is longer jobs would have io wail. The admission scheduler is free to hold some

jobs in the input queue and admit jobs that arrive later if i o cr Figure 2-40. 7hn-t-le\e Once a job ba.s been adnulled to and il can contenci for the CPU. Ho1 of processes is so large that th«)me ol' tbe pn .ystein. a process can be created for il . it migh] well bappen tbat the number not enough room for all of them in memory. have to be swapped out to disk The second level of scheduling is deciding which processes should be kept i which ones kept on disk. We will call this scheduler the memory scheduler, since it determines which processes are kept in memory and which on the disk. This deciston has to be reviewed frequently to allow the processes on disk to get some service. However, since bringing a process in from disk is expensive. the review probably should not happen more often than once per second, mayhe less often. If Ihe contents of main memory are shuffled too often, a large amount of disk bandwidth will be wasted, slowing down file I/O. To optimize system performance as a whole, the memory scheduler might want to carefully decide how many processes it wants in memory, called the degree of multiprogramming, and what kind of processes. If it has information about which processes are compute bound and which are I/O bound, it can try to keep a inixot these process types m memory- As a very crude approximation, if a certain class of process computes about 21)% of the time, keeping five of them around is roughly the right number to keep the CPU busy We will look at a slightly better multiprogramming model in Chap. 4

142 PROCESSES AND THREADS CHAP. 2 To make its dec if. ion*,, the memory scheduler periodically review;, each proc- process on disk to decide whether or not to bring it into memory. Among the criteria that il can use to make us, decision arc the following ones: 1. How long has it been since the process was swapped in or out'.' 2. How much CPU time has ihe process had recently'.' 3. How big is the process? (Small ones do not gel m the way.) 4 How important is the process9 The third level of scheduling is actually picking one of the ready processes in mam memory to run next. Often this is called the CPU scheduler and is the one people usually mean when they talk about the -scheduler." Any suitable algo- algorithm can be used here, either preemptive or nonpreemptive. These include the ones described above as well as a number of algorithms to be described in the 2.5.3 Scheduling in Interactive Systems We will now look at some algorithms that can be used in interactive systems. All of these can also be used as the CPU scheduler in batch systems as well While three-level scheduling is not possible here, two-level

scheduling (memory scheduler and CPU scheduler} is possible and common. Below we will toeus oil the CPU scheduler. Round-Robin Scheduling Now let us, look ai some specific scheduling algorithms. One of the oldest, simplest, fairest, and most widely used algorithms is round robin. Each process is assigned a lime interval, called its quantum, which u is allowed to run. If the process is still running at the end of the quantum, the CPU is preempted and given to another process- If the process has blocked or finished before the quantum has elapsed, the CPU swiiching is done when the process blocks, of course. Round fobin is easy to implement. All the scheduler needs lo do is maintain a list of runnable processes, as shown in Fig. 2-4I(a). When the process uses up its quan- quantum, it is put on the end of the list, as shown m Fig. 2-41 (b). The only interesting issue with round robin is the length of ihe quantum Switching from one process to another requires a certain amouni of time for doing the administration—saving and loading registers and memory maps, updating var- various tables and lists, flushing and reloading ihe memory cache, etc Suppose that this process switch or context switch, as it is sometimes called, takes I msec, including switching memory maps, flushing and reloading ihe cache, eic. Also suppose lhat the quantum is sei ai 4 msec. With these parameters, after doing 4

Figure 2-41. Rnuiiil-nihin scheduling (a) The !isi of runuablc processes, (b] The list ill runiidhle processes alter B iii.es up us quantum msec of useful work, the CPU will have to spend I msec on process switching. Twenty percent of the CPU time will be wasted on administrative overhead. Clearly this is too much. To improve- the CPU efficiency, we could set the quantum to, say, 100 msec Now the wasted time is only I percent But consider what happens on a timeshar- timesharing system if ten interactive users hit the carriage return key at roughly the same trme. Ten processes will be put on the list of runnable processes. If the CPU is idle, the first one will start immediately, the second one may nor start until 100 mt-ec Iafer, and so on. The unlucky Iasl one may have to wail I sec betore getnng a chance, assuming all the others use their full quanta. Most users will perceive a 1 -sec response to a short command as sluggish Another factor is thai if the quantum is set longer lhan the mean CPU bursi, preemption will rarely happen. Instead, most processes will perform a hlocking operation before ihe quantum runs out, causing a process swiich. Eliminating preemption improves performance because process switches then only happen when they are logically necessary, that is. when a process blocks

and cannot con- continue. The conclusion can be formulated as follows- setting the quantum too short causes too many process switches and lowers the CPU efficiency, hut setting it too long may cause poor response to short interactive requests. A quantum around 20-50 msec is olten a reasonable compromise. Priority Scheduling Round robin scheduling makes the implicit assumption that all processes are equally important. Frequently, the people who own and operate multiuser com- computers have different ideas on that subject. At ,i university, the pecking order may be deans first, then professors, secretaries,, janitors, and finally .students. The need to take external factors into account leads to priority scheduling. The basic idea is straightforward: eacli process is assigned a priority, and the runnable process with the highest priority is allowed to run. Even on a PC with a single owner, there may be multiple processes, some more important than others For example, a daemon process sending electronii

144 PROCESSES AND THREADS CHAP 2 mail in the background should be assigned a tower pfiority than a process display- displaying a video film on the screen in real lime To prevent high-prionly processes from running indefinitely, the schedule! may decrease the priority of the currently running process, at each clock nek (i.e., at each clock interrupt)- If this action causes its priority lo drop below that of the next highest process, a pioces-. switch occurs. Alternatively, cadi process may be assigned a maximum time quantum that it is allowed to run When this quantum is used up, the next highest priority process is given a chance to run Priorities can be assigned to processes statically or dynamically. On a mili- military computer, piocesses started hy generals might begin at priority t(X). processes started by colonels at 90, majors at SO. captains at 70. lieutenants at 60. and so on. Alternatively, at a commercial computer center- high-priority jobs might cost 100 dollars an hour, medium priority 75 dollars an hour, and low priority 50 dollars an hour. The UNIX System has a command, nice, which allows a user to volunlarily reduce the priority of his process, in order to bo nice to tbe other users. Nobody ever uses n. Priorities can-also be assigned dynamically hy the system lo achieve certain system goals. For example, some processes are highly I/O bound and spend inost of their time wailing for I/O to complete- Whenever such a process wants the CPU, it should be. given the CPU immediately, to let it start its next I/O request, which can then proceed in parallel with another process actually compuUm* Makmg the I/O-bound.process wait a long time for the CPU will

just mean having it around occupying memory for an unnecessarily long time. A simple algorithm for giving good service to I/O-bound processes i.s to set the pnonty to \/f, where / is the fraction of the last quantum that a process used. A process that used only 1 msec of its 50 msec quantum would get priority 50. while a process lhat ran 25 msec hefdre hlocktng w,rald get priority 2. and a process that used tin: whole quantum would get priority 1, It is often convenient to group processes into priority classes and use priority scheduling among the classes but round-rohin scheduling withm each class Fig- Figure 2-42 shows a system with four priority classes. The scheduling algorithm is as follows as long as there are runnahle processes, m priority class 4. |ust run each one tor one quantum, rouud-rotim fashion, and ne\c-r bother with lower pnonty classes. It priority class 4 is empty, then run die class 3 processes round robin. If classes A and 3 are both empty, then run class 2 round robin, and so on If priori- priorities are not adjusted occasionally. lower pnonty classes may all starve to dejth Multiple Queues One of the earliest priority schedulers was in CTSS (Cofbato et al., 1962) CTSS had the problem that pioces-. switching was very slow hecause the 7A94 could hold only one process m memory. Each switch meant swapping the current process to disk and reading in a new one from disk. The CTSS designers quickjj

SCHEDULING Runabfe processes HZHZHU > -n-cH>n -D Figure 2-42. AM.hedul4(igdlgonlhni realised thirt it was mom-efficient no give CPU-bound.processes a large quantum once m a while, rather than giving.them small quanta frequency (to reduce swap- swapping). On (he other hand, giving all processes a large quantum would' mean poor response ttme, as we have already seen Then solution was to set up prtority class.es. PtocehSes in the highest class were run tor one quantum. Processes in the neKt.hjghest'Class were run for .two quanta. Processes in the next class were ran tot fouri quanta; and so on. Wiiqnever a process used up all the quania allo- allocated to it. il wwt moved down one Glass As an,example, confer a process thai needed i» compute continuously for 100 quanta. Il would initially: ibe. given one quantum, then swapped nut. Next time 11 would gel two quanta before being swapped out On succeeding runs it would gei 4, 8, 16. 32, and 64 quanta, although it would h;ivc us>ed only 37 oi the final 64 quanta to complete its work. Only 7 swaps would be needed (including the initial load) instead of 100 with a pure round-robin algonihm. Furthermore, as the procesb sank' deeper and deeper into the priority queues, u would- be rnn less and less

frequently, saving the CPU tor short, interactive processes. The following policy was adopted to prevent a process thai needed lo run ior a long time when it lirst started bui became interactive latei. from being punished forever. Whenever a carnage return was typed al a terminal, the process belong- belonging io that terminal was moved to the highest priority class, on the assumption that it was aboul to become interaeiive One fine day. some user wiih a heavily CPU- bound process discovered ihat just Mt'ing ai iru; terminal and lyping carriage returns at random every few second', did wonders ioi Ins response time. He told all his friends. Moral of ihe story getting it right in practice is much hafder than gelling it nghi in principle. Many otber algorithms have been used tor assigning processes lo priority classes For example, the influential XDS 940 system {Umpson. 1968). huilt ai Berkeley, had tour priority classes, called terminal, I/O, shon quantum, and long quantum When a process that was waiting for terminal input was finally awak- awakened, il went into the highest priority class {terminal). When a process wailing tor a disk block became re;idy, it went into the second class. When a process was still

 running when its quantum ran oul, it was initially placed in the third class. How- However if a process used up its quantum too many times in a row without blocking for terminal or other I/O, it was moved down to the bottom queue. Many other systems uj,e something similar to favor interactive users and processes over back- background ones. Shorlesl Process Nexl Becau.se shortest job first always produces the minimum average response time for batch systems, it would be nice if it could be used for interactive processes as well To a certain extent, it can be Interactive processes generally follow the paltem of wait for command, execute command, wait for command, execute command, and so on If we regard the execution of each command as a separate "job," then we could minimize overall response time by running (he shortest one first. The only problem is figuring out which of the currently run- nable processes is the shonest one. One approach is to make estimate-, based on past behavior and run the process with the shonest estimated running time. Suppose that the estimated lime per command for some terminal is To. Now suppose its next run is measured to be 7",. We could update our estimate by taking a weighted sum of these two numbers, that is, a7+A -aO"t. Through the choice of a we can decide to have (he estimation process forget old runs quickly, or remember

them for a long time. With a = 1/2, we get successive estimates of To. To/2 + T,/2, 7V4 + 7V4 + 7V2. Ta/S + Tt/& + T-,/4 + T3/2 After three new runs, the weight of 7\, in the new estimate has dropped to 1/8. The technique of estimating the next value in a series by taking the weighted average of the current measured value and the previous estimate is sometimes called aging. It is applicable to many situations where a prediction must be made based on previous values. Aging is especially easy to implement when a = 1/2. All that is needed is to add the new value to the current esiimate and divide the sum by 2 (by shifting it righi 1 bit). Guaranteed Scheduling A completely different approach to scheduling is to make leal promises to the users about performance and then live up to them. One promise that is realistic to make and easy to live up to is this: If there are n users logged in while you art- working, you will receive about 1/h of the CPU power. Similarly, on a single user system with n processes running, all things being equal, each one should ger]/« of the CPU cycles. To make good on this promise, the system must keep track of how much CR' each process has had since its creation. It then computes the amount of CPU each

SEC 2.5 SCHEDULING 147 one is entitled to, namely the lime since creation divided by n. Since (he amount of CPU time eadi process has actually had is also known, it is straightforward to cumpute the ratio of actual CPU time consumed to CPU time entitled. A ratiu of 0.5 means lhai a process has only had half of what it should have had, and a ratia of 2.0 means thai a process has had twice as much as tt was entitled to. The algo- algorithm is then to run the process with the lowest raiio until its ratio has moved above its closest competitor Lottery Scheduling While making promises to the users and then living up to Ihem is a line idea, it is difficult to implement. However, anoiher algorithm can be used to give simi- similarly predictable results with a much simpler implementation. It is called lottery scheduling (Waldspurger and Weihl. 1994). The basic idea is to give processes lottery tickets for various sysiem resources, such as CPU time. Whenever a scheduling decision has to be made, a lottery ticket is chosen ai random, and ihe process holding that ticket gels the resource. When applied to CPU scheduling, the system might hold a lottery 50 limes a second, with each winner getting 20 msec of CPU time as a prize. To paraphrase George Orwell: "All processes are equal, but some processes are more equal." More important processes can be given extra ticketb, to increase their odds of winning. If there are 100 tickets outstanding,

and one process holds 20 of them, it will have a 20 percent chance of winning each lottery, fn the long run, it will get about 20 percent of the CPU. fn contrast to a priority scheduler, where it is very hard to state what having a priority of 40 actually means, here the rufe is clear: a process hofding a fraction /"of the tickets will get about a fraction / of Ihe resource in question. Lottery scheduling has several interesting propenies. For example, if a new process shows up and is granted some tickets, at the very next lotiery it will have a chance of winning in proportion to the number or' tickets it holds. In other words, lottery schedufing is highly responsive. Cooperating processes may exchange lickels if they wish. For example, when a client process sends a message to a server process and then blocks, it may give all of ils lickeLs lo Ihe server, to mcrease Ihe chance of the server running next. When the server is finished, it returns the tickets so (he client can mn again. In fact, in (he absence of clients, servers need no tickets a(all. Lottery scheduling can be used |o solve problems lhat are difficult to handle with other methods. One example is a video server in which several processes are feeding video streams to their clients, hut al different frame rates. Suppose thai Ihe processes, need frames al 10, 20, and 25 frames/sc-c. By allocating these processes 10, 20, and 25 tickets, respectively, they will automatically divide (he CPU in approximaiely ihe correct proportion, that is. 10 : 20 : 25.

148 PROCESSES AND THRBADS CHAP. 2 Fair-Share Scheduling So far we have assumed lhal each process is scheduled on Kb own. without regard to who its owner is. As a result, \i user 1 starts up 9 processes and user 2 Marls up 1 process, with round robio or equal prii>ritie$, user 1 will get 90^ Ot the CPU and user 2 wilf get onfy 10% of it. To prevent (his situation, some systems take into account who owns a process before scheduling it. In thk model, each user is allocated some fraction of the CPU and the scheduler picks processes in such a way as to enforce it. Thus it two users have each been promised 50<7r of the CPU, they will each get that, no matter how many processes they have in existence As an example, consider a system with two users, each ot which has been promised 50<7e of the CPU. User 1 has four proceRses. A. B, C, and D, and user 2 him Only 1 process. E. It round-rohiri scheduling is used, a possible scheduling sequence that meets all the constraints is this one: AEBECEDEAE BLCEDE On the other hand, if user 1 is entitled to twice as much CPU time as usci 2, we might get A B ECDE AB ECDE... Numerous other possibilities exist, of course, and can be exploited,

depending cm what the notion of fairness is. ISA Scheduling in Real-Time Systems A real-time system is one in which time plays an essential role Typically, one or more physical devices external to the computer generate stimuli, and the computer must react appropriately to them within a fixed amount of rime. For example, the computer in a compact disc player gels the bits as they come off the dnve and must convert them into music within a very light rime interval- It the calculation takes too long, the music will sound peculiar. Other real -time systems are patient monitoring in ;i hospital intensive-care unit, the autopilot in an aircrait, and robot control in an automated factory In all these cases, having the right answer but having it too late is often juht as bad as not having it at all. Real-time systems are generally categorized as hard real time, meaning there are absolute deadlines that must be met, or else, and soft real time, meaning that missing an occasional deadline is undesirable, but nevertheless tolerable In boih cases, real-time behavior is achieved by dividing the progmm into a number i>r processes, each of whose behavior is, predictable :md known in advance. These processes are generally short lived and can run to completion in well under a second. When an external event is detected, it is the job of the scheduler to schedule the processes in such a way that all deadline*, are met.

Str. 2J, SCHEDULING 1« The events lhat a real-time system may have to respond to can be further oaieeorized as periodic {occurring at regular intervals) or aperiodic (occurring unpi'edictably). A system may haw to respond to multiple periodic event (.(reams. Depenchng on how much time each event requires for processing, it may not even be possible to handle tbeni all For example if there are i» periodic events and event / occurs with period P, and requires C, seconds of CPU time to handle each cveni. then the load can only be handled il A real-nme system thai meets ihis criteria is said lo be schedulabie. As an example, consider a soft real-time system with ihree periodic events, with periods of 100, 2D0, and 300 msec, respectively. If these events require 50, 30, and 100 msec of CPU time per event, respectively, the sy.stem is schedulable because fj.5+ 0.1?+ 0.2 < 1. If a fourth evem with a period of j sec is added, the system will remain schedulable a* long as this event does not need more ihan 150 nis,ec of CPU time per event. Implicit in this calculation is the assumpt<on that ihe context-*, witching overhead is so small that it can be ignored Real-time scheduling algorithms can he static or dynamic The former make theif scheduling decisions before the lysiem starts

running. The latter make their scheduling decisions at run nine. Static scheduling only works when ihere is per- perfect information available in advance aboui the work needed to be done and ihe deadlines that have to be met. Dynamic scheduling algorithms do no! have these restrictions. We Will defer our study of specific algorithm', until we treai real- realtime multimedia systems in Chap. 7. 2.5.5 Policy versus Mechanism Up until now, we have tacitly assumed thai all the processes in the system belong to different users and are thus competing tor the CPU. While this is often true, sometimes it happens lhat t>ne process has man) children running under its control. For example, a database management system process may have many children. Each chitd might be working o" a different request, or each one iiwght have some specific function to perform (query parsing, disk aeoe&s, ere.) Jt is entirely possible that the mam process has an excellent idea or whjch of it<> chjl- dren are the most important (or time critical) and which the least. Unfortunately, none of the schedulers discussed above accept any input from user processes aboui scheduling decisions. As a result, ihe scheduler rarely makes [he best choice. The volution io this problem is to separate the scheduling mechanism from the scheduling policy. What this means is ihai the scheduling algorithm ih parameterized m some way, but ihe parameters can be filled in by user processes Let us consider the database example once again. Suppose ihat ihe kernel uses ,i

150 PROCESSES AND THREADS CHAP 2 prioniy scheduling algorithm but provides a system call by which a process c;in set (and change) the priorities of its children In this way the parent can control in deiail how its children are scheduled, even though it itself does no! do [he scheduling. Here the mechanism is in the kerne] but policy is sei by a user proc- 2.5.6 Thread Scheduling When several processes each have muliiple ihreads. we have two levels of parallelism preseni: processes and ihreads Scheduling m such sysiems differs substantially depending on whether user-leve! threads or kernel-level threads (or both) are supported. Let us consider user-fevel ihread*. first Since the kernel is no! aware of the existence of ihreads, it operaies as1 it always does, picking a process, say, A. and giving A control for its quantum. The thread scheduler inside A decides which ihread io run, say AI. Since there are no clock interrupis lo muliiprogram threads, this thread may continue running as long as it wants to. If ii uses up 'he process' eniire quantum, the kernel will select another process io run. When ihe process A

finally runs again, ihread AI \vjlj resume running, li will continue to consume all of A's time until it is finis'hed. However, its antisocial hehavior will not affect oiher processes. They will get whatever the scheduler considers their appropriate share, no matter what is going on inside process A. Now consider the case that A's (hreads have relatively little work to do per CPU burst, for example, 5 msec of work within a 50-msec quantum. Conse- Consequently, each one runs for a little while, then yields the CPU back to the thread scheduler. This mighi lead to the sequence At. A2, A3, At. A2, A3, At. A2, AS, At, before the kernel swiiches to process fi This situation is illustrated in Fig. 2- 43(a). The scheduling algorjihm used by the run-time system can be any of the ones described above. In practice, round-robin scheduling and priority scheduling are most common. The only constrain! is ihe absence of a clock to interrupt a thread that has run too long. Now consider the situation wilh kernel-level threads. Here the kcroel picks a particular thread to run. It doe*, not have to take into account which process the thread belongs to, but it can if it wants to. The thread is given a quantum and is torceabiy suspended if it exceeds the quantum. With a 5tl-m«,ec quantum hut threads that block after 5 msec, the thread order for some period of 30 msec might be Al. fi/, .42, fi2, A3. B3. someihing not possible with these parameters and user-level threads. This situation is partially depicted in Fig. 2-43(b). A major difference between user-Level threads and kerncl-lcveL threads is the performance. Doing a thread switch wilh user-level threads takes a handful of machine instructions. With kemel-fevel threads it requires a full context s.witch, changing the memory map, and invalidating the cache, which is' several orders of

magnitude slower. On the oiher block on I/O does not suspend tht Since the kernel knows that > process 6 Is more expensi land, with kernel-level threads, having a thread entire process as ii does with user-level threads, ^itching from a thread in process A to a thread ung a second thread in process A (due io having to change the memory map and having the memory cache spoiled), ii can take [Jus information into account when making a decision. For example, given two threads thai are otherwise equally important, with one of them belonging to the same process as a thread that just blocked and one belonging to a different process, preference could be given to the former Another important factor is that user-leve) threads can employ an application-specific thread scheduler. Consider, for example, the Web server of Fig. 2-10. Suppose that a worker thread has just blocked and the dispatcher thread and two worker

threads are ready. Who should run next? The run-time system, knowing what all the threads do, can easily pick the dispatcher to run nexi, so it can start another worker running This strategy maximizes the amount of parallel-parallelism in an environment where workers frequently block on disk I/O. With kernel- level threads, the kernel would never know what each thread did (although ihey could be assigned different priorities). In general, however, application-specific thread schedulers can tune an application better than the kernel can. 2.6 RESEARCH ON PROCESSES AND THREADS In Chap. 1, we looked at some of the current research i ucture. In this and subsequent chapters we will look at moi search, starting with processes. A*, will become clear in time t>perating system narrowly focused some subjects are

132 PROCESSES ANWTMREADS CHAP 2 much more set (led than olhea. Most of the research tends (o be on (he new topics, rather (han ones that have been around for decades. The concept of a process is an example of Something (hat is weH sottted. Almost *very system Tias some notion of B proeass as a contfiiner for grouping together related resources, such as an address space, threads, open tiles, protection permissions, etc Different, Systems do the grouping *Jig-h<ly differently, but these are >]<S engineering dMtewuces The basic idea is-JiGfl very controversial any more and t'tere i\ little new research on tbe strict. Threads are a newer id«a tbart processes, so there is stili somfl research going on a&out them. Hauser & al. A993) looked at how teaJ programs- actually use thread!! and—came, ^p "Jith f(J different paradigms tor thread usage Thread scheduhMg^iroih.unipreiueBsftr and multiprocess«j^"iS'S(iH a topic near and dear to tW -head xrf'fcomei'esearelTers (Bfumofe and- Letserson, 1994; Buchanan and Chien, i997,'CorbafcSn ct al.. 2000; Chandra e(af., 20(K); Duda and Chenton, 1999; Fo«1^9n*8tltaFlai -t9*»6!'emd Pttrou'et VA.\ 1999). However, few actual sys-system designervaft wafkfng around all day wringing tfieir hands for lack of a detent thread scheduling algorithm, so it appears ifiis type of research is mere iti ynizatwn »nd m«|ua1 esclusfon. In mm «i»«artor aH i| was worfli, w> there is nol jTe is tends lo be focuses on perfbr- nuoide (e^,(^iesdihev Wiftpxndi&S&p (Swdtiting Nynchmnl2;l"0n eirors (Savage et afo, <t999|JjOEioo*%iilgoM'Jcitcapteiiti.t«w wayEFfTdi and Carver. 1996: Thwro, 2fiKM^iF»ifflnjp«ISvrftPO6*Xi;(«mfi*«l«M (breads packages are still ibeing prixluced o,1 t9f»4t>flffdlMfter, 1999}. 2.7 SUMMARY To hide the effects of interrupts, operating systems provide a conceptual model

consisting of sequential processes running m parallel Processes can be created juirHerJOiaated dynamically. F,ach process has us own address space. For some applications it is useful to have multiple threads ot control within a single process. These threads are scheduled independently and each one has its own stack, but aH the threads m a process share a common address space Threads can be implemented in user space or m ihc Kernel Processes can communicate with each other using interprocess tomniuuica- tion primitives, such as semaphores, monitors. Or messages. These primitives are used to ensure that no two processes are ever in their critical regions ,u ine same time, a situation that lead.s w ehaos. A process can be running, mnnable. or blottksd an* cm change state when it or another process executes one of the inter- interprocess communication primitives, Interthread communication is similar.

SEC. 2 7 SUMMARY 153 Interprocess communication primitives can he used m solve nuch problems as ihe producer-consumer, dimng philosophers, reader-writer, and sleeping barber Even with (hese pnmnives. care has to be taken to avoid errors and deadlocks. Many scheduling algorithms are known. Some of these are primarily used for hatch systems, such a<, shortesl job first. Others are common in bom batch sys- systems and interactive systems. These include round robin, priorfly scheduling, multilevel queues, guaranteed scheduling, lottery scheduling, and fair-share scheduling. Some system-, make a clean separatum between the scheduling mechanism and the scheduling policy, which allows users to have control of the •vcheduliiiB aleoriihm. PROBLEMS 1. In Fig 2-2, three proofs states are shown. In theory, wiih three slates, lhere could be six. transition^ two out or. each state. However, only (our transitions are shown. \re there any circumstances in which either or both of ihe missing transitions mighi 2. Suppose that ^ou were to desigri an advanced computer architecture that did process, switching |n hardware, insieail of having interrupt* What information would the CPU ncecP Describe how the hardware pr.ict^s switching might work 3. On dll currerit compulers, at least part ot the interrupt handlers are written in .issembly language Why? 4. When an interrupt or >i system call lrjnsfers control to the optrdting system, a kernel stack area separate Iron) ihe slack of. the iniemjpteJ process is generally used. Why ' 5. In the text it w-ts stated that ihe model ot Fig 2-6(aj was Dot suited to a lUc server using J cache in memory Why not11 Could each process have us own cache * 6. In Fig 2-7 the register sei is listed as a pcr-thre<i(i rather than a per-process item 7. If a mulmhreaded pmc^s forks, a problem occurs it the s-

hild gef. copies- <»i all the parent's ihrcad.s Suppose that ,)nL- of llie unginal thieads was waiting for keyb,).ird inpui. Ntiw iwo threads are waiting for kcyboLinl inpui, one m eacli process. Doc this problem ever oeeur in ^ingle-lhreaded processes1 8. In Fig 2-11), a multithreaded Web ^rver is shown It llie only way to read troin a file is the normal blocking read system call, do you think user-level thieads or kernel-level threads .ire being uscd for the Web server1 Wiry ' all. smcc lherc is no periodic; clock interrupt n m.iy never get the CPL back 10. Can a thread ever be preeiiipied by a clock niterrupi" It so under what lic- cumsiances1' If not. wl,y noi1'

154 PROCESSES AND THREADS CHAP 2 11. In this problem you arc to compare reading a file using a smglc-ihreaded file serve, and a multithreaded server. It takes 15 msec to get a request for wi?rk. dispatch it. and tin the rest ol lhe necessary processing, assuming lhat the daw needed are in llie block cache. If a disk operation is needed, as is the case one-third of the time, an additional 75 msec is required, during which time the thread sleeps. How many requests/see can [he server handle if it is single lhreaded? If il is ii)ultiihre<tded" 12. In Ihe text, we described a mullithreaded Web server, showing why it is better than a single-ihreaded server and a finite-state machine server. Are there any circumstances in which a single-threaded server might be better' Give an example 13. In the discussion on global variables in Ihreads. we used i procedure create global to allocaic storage for a pointer to the variable, rather than the variable i|sell Is th,s essentiil or ujuld the procedures work wnh the value*; themselves just as well' 14. Consider a system in which threads are implemented entirely in user space, with ihe run-time vsiem geiting a clock interrupi once a second Suppose ihat a clock inter-interrupt occurs while sonic thread is execuiing in the run-time system What problem might occur? Can you suggesi a way to solve n ' 15. Suppose that an operaiing system docs not have .inyihmg like Ihe select system call to see in advance if it is safe 10 read from a file, pipe, or device, bui it dues allow alarm clocks io be set thai interrupt blocked sysiem calls. Is ii possible io implemem a lhreads package in user space under lhese eondiiions'1 Discuss 16. Can Ihe priorii}* inversion problem discussed in Sec. 2 3.4 happen witli user-level threads" Why or why noi'> 17. In a system with threads, is there one siack per lhread or one suck per process when user-level threads are used' Whai about when kernel-level threads are used? Explain. 18. Whai is a race condition9 19. When a computer is being developed, n is usually firsi

simulated by a program lhai runs one instruction at a time. Even muluprocessor? arc simulated siricily sequemially like this Is ii possible for a race condiiion io <>ccur when there are no srmulianctius events l,keth,s? 20. Does the busy wailing solution using the tum variable (Fig 2-20) work whcii lhciwo processes are running on a shared-memory multiprocessor, thai is. iwu CPUs, sharing a common memory" 21. Does Peterson's solution io the mutual exclusion problem shown in Fig. 2-21 work when process scheduling is preemptive' How aboul when n is nonpreemptive? 22. Consider & computer thai does noi have a TSL instruction bui docs have an insiruclioii io swap ihe contenis of a regisier and a memory word tn a jingle indivisible aciiiin Can ihai be used to write a rotiiine eiiier_re%(on such ,i< the oni." found in Fie "-^2'> 23. Give a skeich ot how an opcrannng sysiem ihat c.in disable mtcrrupis could unpIemiMii 24. Shuw how counting semaphores (i e , semaphores ihat can hold an arbitrary value) can be implemented using only binary semaphores and ordinary machine instructions

them'1 Why or why not > 26. In S«. 2 3 4, a situation wiih a high-priority process. H. and a low-priority process, L w^s described, which led to « looping forever. Does ihe same problem occur if lound-robin scheduling is used msiead of priority scheduling? Discuss. 27. Can iwo threads in the same process synchronize using a kernel semaphore if the ihre.ids arc implememed by the kernel? Whai if they are implemented in user space' Assume that no threads in any other processes have access to ihe semaphore Discuss 28. Synchrunization within inoniiors uses condition variables and iwo special operations, wait and signal A more genera! form uf synchroni/.ation would be u> have a stngle primitive, wailunlil, (hai had an arbtirary Boolean predicate at> parameter Thus, one cuuld say, for example, waituntil x < 0 or i +;<n The signal pnmiuvc would no longer be needed This scheme K clearly more general than that of Hoarc or Brinch Hanscn. but ti ts noi used. Why not? Him Think aboui 2V. A fast food restaurant has four kinds of employees (I) order lakers, who lake custo- customers orders. B) cooks who prepare Ihe food: C) packaging specialists, who siuff ihe food into bags, and D) cashiers, who give Ihe bags to customers and take ihetr money Each employee can be regarded as a commumealing sequential process Wha(form i|l interprocess communication do (hey use1 Relate this mode! to processes in UNIX 30. Suppose thai we have a message-passing sysiem using mailboxos When sending to a Instead, n gels an error code back. The process responds to the error code by jusl Irv-

Irving again, over and over, until it succeeds. Does (his scheme lead to race conditions'1 31. In ihe soluiion io the dining philosophers problem (Fig. 2-20). why is ihe staie variable st:i to HUNGRY m Ihe procedure hike^fork^ 32. Consider ihe procedure pui_fnrk\ in Fig. 2-20 Suppose iha(the variable state\i\ was se((o THINKING after (he (wo calls to ten. ra(hcr than before. Hnw would (his change ailed ihe solution0 33. The readers and wriicrt problem can be lorrnula(ed in several ways with regard tn which caiegory of processes, can be started when. Carefully describe three differeni variations of (he problem, each one favoring (or no(tavoring) tiime category m processes. For each variation, specify what happens when a reader or ,i writer becomes ready (o access the dalahase, and whai happens when a process is finished using the database 34. The CDC 6600 computers could handle up to 10 I/O processes simultancous-K uskijj an interesting lorm ol round-robin scheduling called processor sharing A process ,ind (he overhead was zero I' a process needed '/' sec to complete in the absence nf

156 PROCFSSF.S ANRTHRFAD.S CHAP 2 compel tit used with (i 35. Round-mbm scheduler* normally maintain a list ot all runnable processes, with ejeh protest occurring exactly once ill the list. What wmild happen tl a process .iccun-cd (vi ice tn ihc Itsl > Can you think ol any revolt |,>r allowing this') 36. Can a measure of whether a prices* is. 1'kdy to be CPU bound oi I/O bound bo deter- determined by analyzing source code' How «in (his be deu-numed at run lime" fun when it blocks. Give ii situation where this could be used and cxphun how 38. ^^ casu re m en ts 'if *i ecus in system have shdV^n thai (he s> era^e process runs lor 9 lime T before blocking on I/O A process swildi requires a time S. which is effeaivelj the CPU efficiency for each ot ihe follow ing (b) Q> ! U)S<Q<T <d) Q = s (e) Q nearly 0 39. Fivejobsare waiting to be run. Their expL.cied run times arc 4, b. 3. 5. and X lnv.ha(iirdcr should ihey ht run to minimize average ivs'pon-,e time' (Your answer will depend on X) 40. Five butch jobs A through E, arrive at a computer centci ai almost the same time They have estimated running hniei' of 10. 6. 2. 4. and 8 minutes. Their (exicrnyHy determined) priorities are 3. 5, 2. I, and 4. respectively, with 5 being the highest pdiir- iiy. For each of the following scheduling algortilmis. determine the mean process iui naround (ime. Ignore process iwtiching mcrhend (.U Round vohm. (bl Priornj scheduling. (c) Firsi-comc. llrsi-served (fun in order 10. 6. 2, 4, Si. (d) Shortest job fl™ For (a), assume (hat (he system is muldproj-'ftnnmeil. and th.il tacit joh gels i^ lair share of (he CPU

For fb) through (d) actinic that unlj one job ,it ,i time runs, until it finishes All jobs Lire completely CPU bound 41. A process running on CTSS needs 30 quanta to complete Hnw many times musi n be swapped in, including Ihc very first time (before it hjs run Jt all)' 42. Cun you think o\ a vvay to save ihe CTSS pficirny .ysiem Ir'im being tooled by run- dom carnage returns' 43. The aging algorithm with a = 1/2 is heiny used to prwiia run limes The invyious (nur runs, irnm nldest 10 mum reteni, JrC 40. 20, 40. and 15 msec Whjt is ihc prediction nl the nest rime'1

fHAP 2 PROBLEMS 157 4d. A soli red-time system has tour periodic events with periods o\ 50, 100. 200. and 2^0 msec each Suppose lhdl Ihe tour events require «, 20, 10. and x msec ui CPU lime. (Cpcctively What is the largest value of x 1c,r which the .system ,s schcdulable1 46. Consider a systc-m in which a is desired io separate policy and mechanism for ihe scheduling of kernel ihrf.ids Propose a me.ms ol achieving (hts goal. 47. Write a i.ltell script (h.ii produces a Tile ol sequential numbers by reading the last tiiiiiihcr it] (he file, adding I m n. and then appending n m Ihe file. Run one instance til ihe script in 'he background and one *n the foreground, each accessing the sairie lilt: How long does, tt lake before a lace condition maiiifesls Uself Wh;tt is the trui- Ct*l rtbion" Mo*liiy iht? scripl Ui prevcm the race (hint use In file tile lock mlocJ, thedJta tile). 48. Assume ihat ynu have .tn nperjimg syUem lhat provides, semaphores Implement a message i.yst^*n- Write the procedures for sending and receiving messages 49. Solve the dining philosophers problem usifi^j mimitors instead o\ semaphores. 50. Suppose lhai a un*vers*iy wants io show oil how politically correct it is by applving ihe U.S. Supreme Court's "Sepanitc bill equal is inherentb unequal" docinne Io uendfr as well as race, ending us long-sianding practice ot gender-segregaied baih- roijnii. t~)n ciimpus. H(*wever. as a cwncesskin io lradinon. tt decrees that when a wjinian is m a bathrooni. oiher woitien may enter, but no men, and vice vcrs.a A s*gn wild a sliding marker on (he door of each bathroom mdicaic<, whtch ofihrec possible . Eraplj • Men prescm tn youi tavoriie piogrammtng language, wr«e iht followiitg procedures: may use whi*it?ver counrcrs jnd synchronizal'on Iechn*ques you like 51- Rcwrtie the program ni Fig 2-20 to handle more than mu processes 52. Write a producer-consumer problem that Uses threads and shares a common butter However, do not Use semaphores or <tny oiher synchronizemn prtmttives (o guard the shared daia slruciures Just lei each thread access them when n wants to Use sleep and wakeup to handle ihe

full and empiy condition1' See how long it Likes lor ,i fatal r^ce condition to occur For example, you tmght ha\e the producer Prjnt t number once m a while D(i not prtnt more than nnc number every minute bcoiusc- the I/O could alfcei the race condilions 53- A process can b£ pit into a rounti-robin q^Jtnc n^^rc than once io Pive it <i higher prjor- itv Running multiple instances Oi *\ program cacti workmt? iin i dilfeienl par! (il a data pool uun have Ihe s.ime effect Firsl untc a ptogram that tests a hsi of numbers forprtmalify Then devise j meihod to allo* multiple msiances of the program to run

58 PROCESSES AND THRfcADS CHAP. 2 ji ooce in such a «ay iha(no (wo instances of ihe program will wurk on the same number. Can you rn fact gel ihrougb (he Irsi lasier by running muliiple copies, ot lhe program1/ Note that your results will depend upon what else your computer is doing; on a personal rampoier running only instances nt th's priigram you would not expeel an tmprovemeni, bin on a sysiem wfih other processes, you should be able to grab a bigger thare of the CPU (his way

DEADLOCKS Computer systems are full of resources that can only be used by one process at a time. Common examples include printers. iape drives, and slots in the system's internal tables. Having two processes simultaneously writing to the printer leads to gibberish. Having two processes using the same file system table siot will invariably lead to a corrupted fiie system Consequently, all operating systems have the ability 10 (temporarily) grant a process exclusive access to cer- certain resources. For many applications, a process needs exclusive access to not one resource, but several- Suppose, for example, iwo processes each warn to record a scanned document on a CD. Process A requests permission to use the scanner and ts granted it. Process B is programmed differently and requests the CD recorder first and is also granted it. Now A asks for the CD recorder, bui the request is denied until B releases it. Unfortunately, instead of releasing ihe CD recorder B asks ft.r the scanner. At this poinr both processes are blocked and will remain so forever. This situation is called a deadlock. Deadlocks can also occur across machines. For example, many uffiees have a loeai area network with many computers connected to it. Often devices such as scanners, CD recorders, printers, and tape drives are connected 10 the network as shared resources, available to any user on any machine. It these devices can be reserved remotely (i.e., from the user's home machine), the same kind of dead- deadlocks can occur as

described above. More complicated situations can cause deadlocks involving three, four, or more devices and users-

16D DEADLOCKS CHAP. 3 Deadlocks can occur in a variety of situations besides ret|ui!Siing dedicated i/O devices. In a databa.se system, for example, a program may have to lock several records it is using, to avoid race conditions. If process A locks record Rl and process B locks record R2, and then each process tries to luck ihe other tine's record, wo also have a deadlock. Thus deadlocks can occur on hardware resources or on software resources. In this chapter, we will look at deadlocks more closely, see how they arise, and study some ways of preventing or avoiding tiiem. Although this material is abou! deadlocks in the context of operating systems, they also occur tn database systems and many other contexts in computer science, so this material is, actually applicable to a wide variety of multiprocess systems. A great deal has been writ- written about deadlocks. Two bibliographies on the subject have appeared in Operat- Operating Systems Review and should be consulted for references (Newton, 1979: and Zobei, 1983). Although these bibliographies are old, most of the work (,n deadlocks was done well before 1980, so they are still useful 3.1 RESOURCES Deadlocks can occur when processes have been granted exclusive access to devices, files, and so rorth. To make the discussion of deadlocks as general as possible, we will refer to the objects granted as resources A resource can be a hardware device (e.g., a tape drive) or a piece tir information (e.g.. ,i locked record in a database). A computer will normally have many different resources that can be acquired. For some resources, several identical instances may be available, such as three tape drives. When several copies of a resource are avail- available, any one of them can be used to satisfy any request foi ibe resource. In short- a resource is anything that can be used by only a single process at any instant of 3.1.1 Preemptahle and Nonpreemptahle Resources Resources come in two types: preemptable and nonprecinptable. A preempt- able resource is one thai can be taken away from the process owning it with no ill effects Memory is an example of a preemptatHe resource. Consider, for exam- example, a system with 32 MB of user memory, one printer, and two 32-MB processes that each want to print something. Process A requests and gets the printer, then starts to compute the values to print. Before it has finished with the computation, it exceeds its t'me quantum and is swapped out. Process B now runs and tries, unsuccessfully, to acquire the primer. Poicn- tially, we now have a deadlock

situation, because A has the printer and B has the

SEC 3.1 RESOURCES 161 memory tnd neither can proceed wifhout ihe resource held by tbe other. For- Fortunately, ,t is passible to preurnpt (take away} the memory from B by swapping it nut and swapping A in Now A can run, do its printing, and ihen release the printer No deadlock occurs. A nonpreemplable resource, in contrast, is one ihat cannot be taken away rrom its curreni owner without causing the computation to fail. If a process h;is begun to bum a CD-ROM, suddenly taking rhe CD recorder away from it and giv- giving 11 io another process will result ,n a garbled CD CD recorders are not preemptaWe at an arbitrary moment- In general, deadlocks involve nonpreemptable resources. Potential deadlocks that involve preemptable resources can usually be resolved by reallocating resources from one process to another. Thus our treatment will focus on non- nonpreemptable resources. The sequence of events required to use a resource is given below in an ab- abstract for.n. 1. Request the resource. 2. Use the resource. 3. Release the resource. if tbe resource is not available when it is requested, ihe requesting process is forced to wait. In some operating systems, the process is automatically blocked when a resource request fails, and awakened when it becomes available, in other systems, tbe request fails with an error code, and |t is up to the calling process to wait a lUtle while and try agatn. A process whose resource request has just been denied will normally sit in a tight loop requesting the resource, then sleeping, then trying again. Although this process is not blocked, for all intents and purposes, it is as good as blocked, because it cannot do any useful work. In our further treatment, we will assume that when a process is denied a resource request, it is put to leep. The exact nature of requesting a resource is highly system dependent. In some systems, a request system call is provided to allow processes to explicitly ask for resources. In others, the only resources that ihe operating system knows about are special tiles that only one process can have open at a lime. These are opened by the usual open call, if me file is already in use. tbe caller is Hocked uniil its curreni owner closes it 3,1,2 Resource Acquisition For some kinds of resources, such as records in a database system, it is up Io ihe user processes to manage resource usage themselves. One possible way of allowing user managemuni of resources is to associate a semaphore with each resource. These semaphore!, are all initialized to i. Mutexes ean be used eqanlly

162 DEADLOCKS well. The three steps lis(ed above are then implemented i pbore to acquire the resource. us(ng tbe resource, and final to release it These steps nre shown in Fig. 3-l(ii}. CHAP. 3 ■ an up on ihe resource lypedef rnt semaphore, semaphore resource 1: raid process _A(void) { down(&resouree _ 1), use-resource -1(), up(&resource 1); typedef in(semaphore; semaphore resource. 1; semaphore resource. 2, void process-A(void) (down(&resource. 1); down(&resource _ 2), use_both_resources() up(&resource _2); yp(&resource_ 1 j; } (b) ;es. They can be acquired o resources are needed, they Figure 3.1. Usinj! J tciiiiiphore to proicti iiMmries (< Sometimes processes need two or more re.sou sequentially, as shown in Fig. 3-Kb) if more than !■ are just acquired one after another. So far, so good. As long as only one process is involved, everything works fine Of course, with only one process, tbere is no need to formally acquire resources, since there is' no competition for ibem. Now le(us Consider a siiualion with two processes, A and B, and (wo resources. Two scenarios are depicted in Fig. 3-2. in Fig. 3-2(a), both processes ask for the resources in ibe same order in Fig 3-2(b), they a*,k for them in a dif- different order. This difference may seem minor, bul it is not, in Fig. 3-2(a), one of the processes will acquire the first resource before the Oiber one. Thai process will Iben successfully acquire Ibe second resource and d0 its work. If the other process allenipis io acquire resource! I before il lias been released, the other process will simply block unlil it becomes available. in Fig. 3-2(b), the uluation is' different. It migh(happen (bat on^ of ibe processes acquires both resources and effectively blocks oul the o(ber process until it is done. However, il mighl also happen that process A acqu,res resource 1 and process B acquires resource 2. Each one will now block when trying the acquire the other one. Neither process will ever run again. This situation is a deadlock. Here we see bow wbal appears @ be a minor difference in coding slyle— which resource to acquire first—turns out to make tbe difference between the pro- program working and [he program failing tn a bard-(o-detect way. Because deadlocks can occur so easily, a lot of research has gone in(o ways (o deal ibem. This chapter discusses deadlocks tn detail and what can be done about them.

INTRODUCTION TO DEADLOCKS typedef int semaphore; semaphore resource__1; semaphore resource-1, semaphore resource__2, semaphore resource. 2; void process-A(void) (void process^A(void) { down(&resource 1); down(&resource-1); downf&resoiirce 2); down(&reSource 2); use-both

__resources(); use_botb_resources(); up(&resource^2); up(&resource, 2); up(&resource_1); up(&resource-1); I I void process-B(void) (void process 3(void) (down(Sresource_1); down(& resource _2); down(Sresource_2); down(&resource_1|; use_both_resources(); use_both_resources{); up(&resource_2|; up(&resource_1); up(&resource_1)- up(&resource 2); I I Figure 3-2. !a) Deadlock-I^eoode (b) Cede wllh n potential deadlock. 3.2 INTRODUCTION TO DEADLOCKS Deadlock can be defined formally as follows: A set of processes is deadlocked if each pmcew in the \et L\ wailing /or an event that only another process in the fet cun ctiuse. Because alf the processes are waiting, none or them will ever cause any of the events thai could wake up any of the other members, of Ihc sel, and all Ibe processes continue to wall forever. For this model, we assume |baj processes have only a single thread and thai (here are no interrupts possible to wake up a blocked process. The no-!n(errup(s condition is needed to prevent an oihcrwise deadlocked process from being awakened by. say, an alarm, and then causing evenis that release other processes in the sei. In mosi cases, the event (hat each process is waiting for is the release ol some resource currently possessed by another member of Ihe set. In other words, each member of the set of deadlocked processes is waning for a resource that is owned by a deadlocked process. None of the processes can run, none of them can lelease any resources, and none at them can be awakened. The number of processes and the number and kind of resources possessed arid re(jues(ed are ummportani. This result holds for any kind of resource, including bo(h hardware and software.

3.2.1 Conditions for Deadlock Cotfman et al. A971) showed that four conditions must bold for there to be a deadlock: I. Mutual exclusion condition Each lesource (S e(ther currently as signed to exacily one process or is available. 2 Hold and wait condition Processes currently holding resources granted earlier can request new resources 3 No preempiion condition. Resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding ibem. 4. Circular wait condition. There must be a circular chain or mo oi more processes, each of which is waiting for a resource held bv (be nexi member of the chain. Alt tour of (bese condi(ions musi be presem for a deadlock (o occur. If one ot them is absem. no deadlock is possible. 1(is worth noiing tba(each condition lel.nes to >i policy that a sys(em can have or noi bave. Can a given resource be

assigned (o more (ban one process .it once' Can a process hold a resource and ask for another1" Can resources be preemp(ed'1 Can circular waits exisi? Laier on we wilt see how deadlocks can be attacked by trying io negate some of these conditions. 3.2.2 Deadlock Modeling Holt A972} showed bow these four conditions can be modeled using direUed graphs'. The graphs bave ,wo kinds of nudes; processes, shown as circles, and resources, shown as squares. An arc- from a resource node (square) (o a process node (circle} means ihat the resource has previously been requested by. graced io, and is currently held by that process. In Fig. 3-3(a), resource R is currently assigned io process A- An arc from a process to a resource means thai Die piocess is currently blocked wailing for thai resource. In Fig. 3-3(b). process B is waning for resource S. In Fig. 3-3(c} we see a deadlock: process C is waning tor resource T. which is currendy held by process D, Process D (s noi abom io release resource 7 because ii is wailing for.resource V, held by C. Bo(b processes w(H wan lorever A cycle in the graph means that there is a deadlock involving the processes and resources rn the cyde (assuming ihat ihere k one resource of each kind). In ihts example, ihe cycle is C-T-D-U-C. Now let us look at an example of how resource graphs can be used Imagine thai we have ihrae processes, A, B, and C. and ihree resources, R. S. and T. The

INiRODLCTION TO DEADLOCKS 165 requests and releases or the three jxocesses are given in Fig. 3-4(<t)-(c) The operating system is tree to run any unblocked process at any instant, so it could decide U) run A until A finished all its work, iben run B 10 completion, and finally run C This order(ng does not lead (o any deadlocks (because there is no competition for resource1;} bin it also has no parallelism at all In addition to requesting and releasing resources, processes compuie and do I/O Whcti the processes are run sequentially, ihere is no possibility (hai while one process is waning for I/O. anoiher can use ihe CPU Thus running ihe processes strictly sequentially may no(he optimal. On ihe oiher hand if none of ihe processes do any I/O a< all. shor- shortest job first is better ihan round robin, so ontlcr some circumstances running ull processes sequentially inay be the best way. Let us now suppose that the processes do ho(h I/O and computing, so that round robin m a reasonable scheduling algonihw The iesource requests mighi occur m the order oi Fig. 3-4(d). If these six requests arc carried out in that order, the six resulting resource graphs are shown m 1'ig ?-4(e)-(j) After request 4 has been made, A blocks waiting lor .V ^ shown in l-jg 3-4(h) In ,hc next ,wo steps B

and C also block, ultimately leading (o a cycle and the deadlock of Fig l-4(j r However, as we hate already mentioned, the operating -ystem is not required to run the processes in any special order In particular, il granting a particular request might lead to deadlock. (he ©penning system can simply suspend ihe proc- process without granting the request d c, jusi "ot schedule the process) until it is sirle in Fig 3-4. if the operating systezn knew about the impending deadlock, it could suspend B instead of granting it .S' By running iitily A and C. we would get thu requests and releases ot hig 3-4(k) instead or Fig. 3-4(d] This sequence leads 10 the resource graphs of Hg 3-4(l)-(q). which do not lead to deadlock Al^er step (u.), procc-ss B can be gian(etl 5 because A is linisbed anil C has every(hing it needs tven it B should eventually block when requesting 7", no deadlock can occui B will nisi w;ni uii(i1 (. is tinished Later in this chaptei we vvil] study a deluiled algorithm ror making a]luca(ion decisions that do not lead to deadlock. For the moment, the point lo understand is

DEADLOCKS ®@© ®@© ®@© 000 0 00 0®© ©0© ©0©^ 000 000 00 0y ©®© ®@© l 11 000 0B0 0 ®®0l 00© 0®® Figure 3-4. Ane>amplo of hov. deaduxk dcluis and how ,t i,,,i be 4 mi tied

IN I ROPl'CTION TO OEADLOC Ki 167 that tesource graphs are a tool that lei us see il a given request/release sequence lends to deadlock. We |usl cair> oul the requests and iclcascs step by step, and lifter every step check tlie graph in see if it contains any cycles. II so, we have <i deadlock; it not, ihere is no deadlock. Although our treatment ot resource graphs has been foi tlie ease of a single resource ol each type, resource graphs can also be generalized to handle multiple resources of the same type (Holt, 1972). 1 Just ignoic the problem altogether Maybe if you ignore it. it wili 2 Detection and recovery Let deadlocks occur, detect them, and take 3 Dynaniic avoidance by eareiul resouice allocation 4 Prevention, bv structurally negating one ot the four conditions neces sary to cause d deadlock We will examine ench or these methods in mm in the next lour sections 3.3 THE OSTRICH ALGORITHM The simple*,! approach is the oslrich algonthiii stick)i>ui head tn the .and and pretend there is no problem at all'i Different people lead to this strategy u) ditfeicnt ways Mathematicians fmd it totally unacceptable and say that deadlocks must be prevented at all costs- Engineers dsk how often the probleiii is expected, how often the system crashes loi other reasons, and ho\v

serious a deadlock is- If deadlocks occur on (he aveiage once every live yeais. but system crashes due to hardware lailures, compiler errors anj opei.uiii11 svsteiii hugs occur onec a week, most engineers would not be willing to pay a large penalty in performance or convenience to eliminate deadlocks To make this contrast more specific, most operating systems potentially suller from deadlocks that are not even detected, let alone authmaticallv broken The the process fable Thus pioeess table slots are (mite resources It a fork (ails because the table is lull, a reasonable appioych (or the program doing tlie fork is to wait a random time and try again Now suppose that a LNIX system has 100 process sluts Ten programs are running, each of which needs to create 12 (sub(processes After each process has

168 DEADLOCKS CHAP 3 created 9 processes, the 10 original processes and the 90 new processes have exhausted the Cable. Eacli ot the 10 oiiginal processes now sits in an endless loop forking and failing—a deadlock. The probability of [his happening if. minuscule, but it could happen Should we abandon processes and the fork call to eliminate the problem1' The maximum number of open files is similarly restricted by the size of the i- nodc table, so a similar problem occurs when il fills up. Swap space on the dUk is another htinted resource In fact, almost every table in the operating system represents a finite resource Should we abolish all of these because it might hap- happen that a collection of n processes might each claim \/u (if the total, and then each try to claim another one? Mosl operating systems, including UNIX and Windows-, just ignore the prob- deadlocks could be eliminated for iree. there would nut be much discussion The problem is (hat the price is high, mostly in terms ol putting inconvenient restric- restrictions OR processes, as we will see shortlv Ihus wt1 die iaced w*th an unpleasant trade-ott between convenience and corrceiness, arld a great deal <>t discussion about which is more important, and to whom, (,'ndei these conditions, general solutions are hard to find 3.4 DEADLOCK DETECTION AND RECOVERY A second technique is detection and recovery. When [his technique is usoi, the system does not attempt to prevent deadlocks rrom occurring Instead, it lets ihem occur, tries to detect when this happens. ,md then takes some at (ion to recover after the fact. In this section we will look at some oi ihc ways deadlocks can be delected and som,' of [he ways recovery hoi,, (hem can be handled 3.4.1 Deadlock Detection with One Resource of Each Type Let us begin with the simplest ej^e. on)> one resource of each lype exisis Such a system might ha\L- one scanner,

one CD recorder, one plotter, and one tape drive, bul no more than one ot each class ot resouice In other words, we .ire excluding systems with two printers for the moment. We will treat them laL-t. using a different method Fig. 3-3. II this graph contains one or more jycles a deadlock exists. Any proc- process that is part ot a cycle is deadlocked It no c;des exist, the system is not deadlocked. As an example of a more complex svslem than the ones we have looked ul so lai, consiilei a system with seven processes. A ihough G, and six resources. R

SHC 3 4 Oh ADLOCK DETECT ION AND RECOVERY 169 through W The staie uf which resources are cuncntty owned and winch ones are currently being requested is as follows I. Process ,4 holds R and wjnls S 2 Process B holds nothing but wants T 3. Process C holds nothing but wants S. 4 Process O holds (J and wanis ,? and T ^ Process £* holds Tand wants V. 6. Process F holds Wand wants S. 7. Process C holds V and wants 6'. The question is: 'is this system deadlocked, ami it so. which processes are involved'.'" To answer this question, we can construct the resource graph of Fig. 3-5(a). This graph contains one cycle, which can be s^en by visual inspection. The cycle is .shown m Fig. 3-5(b). From thK cycle, we can see that processes D. E. and G are all deadlocked. Processes A, O, and F are not deadlocked because ,V can be allocated io any one of them, which then finishes and returns it Then the other iwo can take ii in turn and also cornpleie Although it is relatively simple lo pick oul the deadlocked processes by eye from a simpie graph, Tor use in aciual systems we need a formal algorithm toi detecting deadlocks. Many algorithms for detecting cycles in directed graphs are known. Below we will give a simple one that inspects a graph and terminates data structure. L. a list of nodes. During ihe algorithm, arcs will be maiked to indicate that they have already been inspected, to prevent repeated inspections

UJi(i]JoS|i! 3i|) do]s put! apAo oqi J3Aoosip n^\ ]inod qsiqM l'; "U 'O 77 'O 'A '3 '1 'ffl ^ "l 7 srepdn pm; ; spid ^m ami] puooas aqj, a °' ^pc.uppq pur pua prap r <>] amen o*\ <,- >)Did 3«]l Mioip (luopiiEj) p a^pui jsnui oa\ mi>n [a -,j •/> vi ~3 j -y] = 7 ami] ipiqM IE 'a oi lag sm imm sdjb gmogjno MO|[(i| ni .muDiK'.i a* ff uiojj 3]t; uit:gc ibis a* os -sdois X|>)ninb 'ooi 'qaiBss sdois (iiqi'J ■nf ^ i,i] Ds -Aipuiqai; si s.npou .iq) 3ui^M<ud jo aapjo oqx (n}f-f -oi.j -ip,O si qdpjS 3ji]uo 3i|) -..ip'Hi iji? JU) sp|Oq Oaadtud sii|] ji ^3[3A3 *ue una Mu ssop apou UUJ.in.-> sqj uiojj o|qpq^i'M qdEjSqns ai|] 'iaqjjnj o3

iouuit.i pun nip oi si.-»iM4^.irq H j] spou sndi\3j(l ^qi 01 s^pun^pRq 11 'opou usmS ^ut; U s'ljt* 5l|] |jis slsucq\j]i 41 op^i r puiioj ^pq i' u^i|] "p^-iojuno^u^ \ptroi|tj su spou t- 01 iot(['-31UD3 J0^?]i 41 11 110 i|.^jE3^ Kiy-iiidsp r ^oop pup. -aai] r w| sadnq]i Jcijm jo iooi oqi sp -Tiitil 111 -ap'iu q,ipa d^itj si saop uJi|)ijo3|P siip ip 3i(l si ,ip«u s»im j| X da]s 0] oS puc -spou lusjin^ aq] 3110 irifl a>{BUi ■ouo siqj aJojoq jsnf jiiaiin.i sca\ inq] suo 3i|l >i roqi 'opou snoi,\ -jjrf sip oi ^.req oS pun 11 5 \om.iy -pun peop p ppqopa.: mou 3ACq sm 9 1. dais 01 03 pirp. spou 1U3UHD a\su nqi 01 H wojioj usqx]i ^-ii'iu pup mopuRi iu aip 3uio3ino]>.as|jpuiun up ^oid 5, g cbis o] h8 -iou ji \ dsis oj oil \>s j| sail1 5uio^]no p.^ii>tiuni "iup iijp .iiaqi ji sss 'apou uoaiS .iq] unu-j f. scilpiiiui.131 iiiiiiiuoSiP sq] pup G in p3)si[| a(.iX3 r sumiuo? qdu-ig 341 - saop li || -s'aiun owj 7 «' Mcaddu won apou ?i[i ji aas 01 spaip pun / jo pu^ nqi 01 ?pou majjn.i am ppv t pa^icuiun sc sare sqi i(i? ajpuriisgp pui: -isii ^diiu .up o] 7 3zqpinu] z apou ^uii.ib]S 3q] se iNl i]jiM sdM* s 8ui*on«J) sqiuiiojiad-miwaamiii.V'spouMaiKiiOd] •poiimd- sp scis.s 3uimo(|'\! 31,1 int. SuiXum Aq s.nmsdo Liiqiuoif|n ai|i dVHJ SlM.)O'iaV:lC! Oil

DEAD! OCK DETECTION AND RhCOVfvRY 3.4.2 Deadlock Detection with Multiple Resource of Each Type Whim multiple copies of some ol the resources exist, a different approach d d ddlk W ll bd lth f de ge It ii' A eded to nerally. At an be the drives are request i deadlock am E, rejourc. y instant, som available res assigned, A , ion;z « pu>. ot class i (will be 0. .-esses. P] 1 < i <M1) ources tire the turn throu gh E is the i ass.gn. ;nt all ed , oc» txisli inJa tion Lei the n ing n»oun re no! avai matrix, a umber .e veil lable nd R. of Let ■ oi apu held by process i. Similarly, R,, is the number oi instances ot rtso wanis. These four data structures are shown m Fig. 1-6. Current allocation mat™ |~c,, cl2 cl3 c,ltl Figure 3-6. TIk i,mr d.u,, sirimiirc ntftidl bs Hw ilcidUnk Jt-Uttum .ilgonihin An impoitant invariant holds for these tuui data structures In particular. In other woids. if we add up ail the instances of the resource j that have been

172 DbADLOCKS CHAP 1 allocated and to this adit all the instances thai are available, the result is (he number of instances of that resource class that exist. The deadlock detection algoiithm is based on comparing sectors. Let is define the relation A < B »n two vectors A and B to mean that each element of A is jevi than or equal to the corresponding element of B. Mathematically. A <B holds if and only if A, < B, tor I < / < m. Each process is initially said lo he

unmarked. As the algorithm progresses-, processes will he marked, indicating that they arc able to complete and are thus nut deadlocked Whtn the algorithm terminates, any unmarked processes- are known to be deadlocked. The deadlock detection algorithm can now be given, as follows. I. Look tor an unmarked process. P,. for which the Mh row of R ,s less than or etjual to A 2 If such a process is lound, add the i-th row of C to A, mark the proc- process, and go back to step I ■ 3 If no such process exists, the algorithm terminates. When the algorithm finishes, all the unmarked pioccses, if any. are deadlocked What the algorithm is doing in step I ts looking for a process that can be run to completion Such a process is characterized as having resource demands that can be met by the currently available resources The seletted process is then ran until it finishes, at which time it returns the resources it i^ holding to the pool of available resources. It is then marked as completed II all ihe processes are ulti- ultimately able to run, none of them an? deadlocked. If some <>l them e,ui never run run the processes in any feasible order), the icsult is always the same As an example of how the deadlock detection algorithm works, consider Fig. 3-7. Here we have three processes and loin resource classes, which we ha\e arbitrarily labeled tape drives, plotters, scanner, and CD-ROM drive Process I has one scanner. Process 2 has two tape drive and a CD-ROM drive Piote.ss 1 has a plotter and two scanners Eath process needs additional resources, as shown by Che R matrix . To run the deadlock detection algotitlim, we look for a process whose resource request can be satisfied. Ihe first one canuor be satishcd because theie is no CD-ROM drive available, fhe second cannot be satisfied either, because there is no scanner free. Fortunately, the thud one can be satisfied, ^o process 1 runs and eventually returns all us rcsotirt e,s, gmng A = B 2 2 0) A! this poiiii process 2 can run and return Us resources, gisins A = D 2 2 l| Now the remaining process can run- There is no deadlock m the system.

DEADLOCK DETECTION 4ND RRCOVLRY Hgur :on.-idcr £ n of 1-ig 3-7 Suppose that j tape drives and the plotter. ,tern is deadlocked question of when to look for process 2 needs- a CD-ROM drive as well a Nijne of the requests tan be satisfied, so ihe . Now that we know how to detect deadlo them comes up. One possibility is to chi/ck every lime a resource request is made Tins is certain to detect them as early as possible, but it is potentially expensive in terms of CPU lime An alternative siiaiegy is to chetk i-\ery k minutes, or perhaps only when the CPU utilization has dropped below some threshold. The

reason for considering the CPU utilization is thai if enough processes are deadlocked, there will be few runnable processes, and the CPU will often be idle 3.4.3 Recaverj from Deadlock Suppose that our deadlock detei deadlock- What next' Some way i r and j>ef the sysiem going None of them are especially .ittrac Recovery through Preemption current owner and give it to another process. In many cases, manual mu may be required, especially in batcii pioces*,ing operating systems mi For example, to lake a Idser printer a\va> from its owner, the operator luei all Ihe sheeis already printed and put them m a pile Then Ihc proces suspended (marked as not runnable). At this point the printer can be as'

174 DEADLOCKS CHAP. I aiiolhei pioccss. When thai process finishes. the pile of primed sheets can be put back in the piniter's output tra\ and the original process restarted. il, aiid then give il back wuhoul the process noticing it is highly dependent on the Choosing the process is> suspend depends largely on which ones haw resources thai can easily be taken back Recover* through Rollhack they can arrange to have processes check pain ted periodically Checkpoint ing a process means that its state is written to a file su that it can he restarted later The checkpoint contains* not onlv the memorv image, but also the resource state, thai is. which iesources are currently assigned to the process. To be most effective, new checkpoints should not overwrite old ones but should be written to new hies, so as the process executes, a whole sequence of checkpoint fibs are accumulated When a deadlock is detected, it is easy to see which resomces are needed To do the recovery, a process that owns a needed resource is rolled back to a point in time before it acquired some other resource b> starting one of iK earlier cheek- points. All the work done s'ince the checkpoint is lost (e.g.. output printed since the checkpoint must be discarded, since it will be punted again). |n effect, the process is reset to an earlier moment when it did not have the resource, which is now assigned to one of the deadlocked processes. If the restarted process tries to acquire the resource; again, it will have to wait until it becomes available. Recovery through Killing Process The crudest, but simplest way to break a deadlock is to kill one or tnoie processes. One possibiht) is to kill a process in the cycle With a little luck, the other processes will be ab]e to continue If this does not help, it can be ielated until the cycle is broken. Alternatively, a process not in the cycle can be cho-.cn as the victim in older to re/ease its resources In this appioach. the process to be killed is caretu|l\ chosen because it is holding

resouices that some piocess in the iyclr needs, f-o'i example, one process might hold a printer and wdnt a plotter, with another process holding a plotter and wanting a printer. These- two are deadlocked. A third proc- process may hold another identical punter and anothet identical plottei and he luppik running Killing the third process will ielcase these resouices and brc;,k the deadlock involving the first two Where possible, it is best to kill a piocess that can be rerun Iroiu the hegm ning with no ill effects For example, a compilation can a/way* be rerun because

Si:C ^4 DHADLOCK DETECTION AND RECOVERY 175 at] it does is read a -ource (ile and produce an ob|ect tile. If it i*, killed part way through, the first run has no influence on the second run. On the other hand. .1 process that updates a database cannot always he run a second time safely II the process ad,ls I to some record in the- database running it once, killing it. and then uinning it again will add 7 to the record, which is 3.5 DEADLOCK AVOIDANCE In the discussion ot deadlock detection, we tacitjy assumed that when d pri>c- Css asks tor resources, it asks for them all at once ulie R matri* of Hg. 3-6). In ni<)St systems, however, resources are requested one at a time. The system must be able to decide whether granting a iesource is sate or ntit and only make [he qualified ye(,—we can avoid deadlocks- hut only if certain information is- avail- available in advance. In this- section we examine ways to avoid deadlock by careful 3.5.1 Resource Trajectories The main algorithms for doing deadlock avoidance are based on the concept of sale states. Before describing the algorithms, we will ni.rke a slight digression to look at the concepi of safety m a graphic and easy-to-uuder-tand way Although (he graphical approach does not translate directly inio a usable algo- algorithm, it gives a good intuitive feel foi the nature ol the problem In Fig 3-8 we sec a model lor dealing with iwo processes and two resources, for example, a printer and a plotter. The horizontal a\is represents the nunibei ol instructions executed by process ,4. The vertical axis represents the number ol instructions executed b; piocess B. At /, A requests a printer, at /2 i> needs a plotter The printer and pldltei are leleuhed at /? and /4. lespcaiveK l'a>ceSs B needs the plotter from /; to /; and the printer from /,, to !K fivcry point ;ii the diagram represents a [oint staie of the two processes Ini- Initially, the state is at p, with neither process lu\iiii* executed any inductions If the schedulei chooses to iun A lust, we get to the point q. in which A has executed sonie number of instructions, but B has cxetuifd none. At point q the trajectory becomes

vertical, indicating that the schedulei has chosen to run B. With a siuule processor, all paths must be horizontal 01 vertical, nevef diagonal, hurihcmiore. motion is always td the north or east, nevei to the south or west (processes cannot run backward). When A crosses the /1 line on the path Irom > u> ^ it requests and is grained the printxi When B reaches point /, it lequests (he plotter-

DEADLOCKS The regions that are shaded are especially interesting The region with lines The mutual exclusion ruk' makes it impossible to enter this region. .Similarly, the regn,n shaded the other way represents both processes having the plotter, and is equally impossible. If the system ever enters the box hounded by /, and !-, on the sides and / oi 11 and If,. At this point, A is requesting the plotter and B is requesting the printei. and both are aJready assigned. The entiie box is unsafe and must not bL- entered. At point t the only safe thing to do is run process .4 until il gets to /j. Beyond that, any trajectory to u will do The important thing to see here is at point / B is requesting a resource. The system must decide whether to grant it or not If the grant is made, the system will enter an unsafe region and eventually deadlock. To tivoid the deadlock, B should he suspended until A has requested and release^ !he plotter 3.5.2 Safe and Unsafe States The deadlock avoidance algorithms thai we w,‖ study use the in R A state is said to he safe il order in which every prOLess, c t is noi deadlocked and ther i run to completion even it this concept by an example using t,ne resource In Irig 3-9(al we ha\e a stale in which A has ~\ instances of the resource but may need as many as 9 eventually B currently has 2 and may need 4 altogether, late,-. Similarly, C also has 2 but may

DEADLOCK AVOIDANCE need an additional 5. A [oial ot 10 instances «f the resource e resources alread> allocated, there are 3 still free. 177 Has B Has Max B Has 0 - B 0 B Has 0 Figure 3-9. DemiinsiTjiion lh.il ihe ^Ulc it\ (a) is ^afo The state of Fig 3-9<a) is sate because there exists a sequence of allocations that allows all processes to compleie. Namely, the schedule! could simply run B exclusively, until it asked fur and got two more instances of [he resource, leading to the state of Fig. 3-9(b). When B completes, we get the state of Fig. 3-l>(c). Then the scheduler can run C. leading eventually io Fig. 3-9<d) When C com- completes, we get Fig. 3-(>{e). Now A can gel the six instances ot the resource it needs and also complete. Thus Ihe state of Fis. l-9(a) iN safe because the system, by carelul scheduling, can avoid deadlock No'

requests and gets anotlu that is guaranteed to work'1 Let us try The scheduler tor all its resources, as shown in Fig. 3-10(c rce. giving Fig 3-lO(b) Can we find a sequence n B until it a^ked A B C 3 2 2 9 4 7 A B C 4 2 2 9 4 7 A B C 4 4 2 9 4 7 A B C 4 _ 2 9 _ 7 Figure 3-10. Dem trjtuiii that the still e ge[the siiuali Eventually, B complies, and we get the situation of Fig. V IO{d) At this1 point we aru stuuk We onlv have four instances of the resource tree, and each of the active processes; need1; five. There is no sequence that guarantees completion. Thus the allocation decision [hat moved the system frum Fig. 3-IO(a) to Fi^ 3- |0(b) went from a safe state to an unsafe state Running .1 or C ne\t starting at Fig 3-IO(b) does not work either In retrospect, A\ request should not have been granted \\ is worth noting that an uns^i^e stale is noi a deadlocked state Slarliii!? ai

any more, allowing C to complete and avuidmg deadlock altogether. ThiK the difference between a safe Male and an unsuie slate is that from a safe state the sys- system r-;m ounrtmree that ail processes will liiiish; from an unsale state, no such 3.5.3 The Banker's Algorithm for a Single Resource A scheduling algorithm that can avoid deadlocks is due in Dijksira {]9(v>) and it, known as the banker's algorithm and is .in extension of the deadlock detection algorithm given in Sec 3 4.1. It is modeled on thi- way a small-town hanker might deal with a group of customers lo whom he has granted lines of credit What the algorithm does k check to see if granting the request leads to an unsafe state. If it does, (he request is denied. If granting the request leads to a safe stale, it is carried out. In Fig. 3-11 (a) we see lour customers, A, B. C and D, each of whom has been granted a certain number of credit units (e.g.. I unit is 1 K dollars) The banker knows that not all customers will need their maximum credit immedi- immediately, so he has reserved only [0 units rather than 22 to sen/ice thein l|n this analogy, customers are piocesses, units are, say, tape drives, and the banker is ihe operating system.) B C D 0 0 0 JL A 7 A a c D 1 1 2 A 6 5 A A 1 1 B 1 2 C 2 □ | 4 6 5 4 7 The customers go ahout their respective businesses, making loan icquests from lime to time (i.e., asking for rcsouicesl. \\ a ccitam ninuicut. the situation is can delay any requests except C\. thus letting (' finish and rele.is,- all lour of his resources. With four units in hand, the banket can lei cither O 01 B have Un- Unnecessary units, and so on Consider what would happen ii a requesi from B Jor one more mm were granted in l:ig 3-1 l(b) We would have situation Fig 3-1 l(e|. wind, ,s iu,sale. If all the customers suddenly .isked for then maximum loans, the bankei could noi

satisfy any or them, and we would have .1 deadlock An unsafe stale Joes noi haw to lead to deadlock, since a customer mighi noi need the entire credit line available but the banker cannot count on ihis behavioi The banker's algorithm considers each request as it <>ccuis, and see il granting 11 le.ids to a safe state. If it does, the request is granted; otherwise, it is postponed

stc ^5 DEADLOCK AVOIDANCE 17V until later. To sec il .1 staie i.s safc, the banker checks tu see il he has enough Source1; i<» satisfy some customer If so, those loans are assumed to be repaid, .nid the customer novt closest to the limit is checked, .ind so 011 It all loans can 3.5.4 The Banker's Algorithm for Multiple Resources The hunker'., . ure 3-12 shows ho A B C D E 3 0 1 1 0 0 1 1 1 0 1 0 1 0 0 1 0 0 1 D A B C D E 1 D 3 0 2 t 1 I 0 1 0 1 0 1 1 0 2 0 0 D E =F342| P = E322) A = A020) Resources assigned Resources still needed Figure 3-12, fhe banker's .ilgondun with mutnpltf rcsouriL's In Fig. 3-12 we see two matrices. The one on the led shows h each resource are currently assigned to each of ihe tive processes. T the right show's how many resources each process still needs in order These matrices are just C and R from Fig-. 3-6 As in the single re processes musl state their total resource needs before executing, so thi.i can compute the right-hand matrix at each instant. The three vectors at the right of the figure show the existing reso possessed resources, P, and the available resources. A, respectively. see that the system has six tape drives, three plotters, four primers, a urces, t, the l;rom E we nd two CD-ROM drives. Of these, five tape drives- three plotters, two printers, and two CD- ROM drives arc currently assigned. Thi.s fact can be seen by adding up the tour resource columns in the left-hand matrix. The available resource vector is simply the- difference between what the system has ,md what is Lurrently in use The algorithm I'nr checking to see if a state k safe can now be stated 1. Look tor a row, R. whose unmet resomce needs are uti smaller than Ot equal to A. |f no suuh row exists, the system will eventually deadlock .since no process can run to completion. 2. Assume the pioces:> of the row chosen requests <t[l the resources it needs (which is guaranteed to bu possible) and finishes. Mark that piocess as terminated and add all its resources to the A vector.

180 DEADLOCKS CHAP 3 1. Repeal steps I ;incl 2 until either all processes arc marked terminated, in which case the initial state was safe, or until a deadlock occurs, u) which case it was not. It sever.!] processes' are eligible to

be chosen in step |, it dues not matter whidi one is -.elected' the pool of available resources either yets larger, or at worst, stays Now lei ik get back to the example of Fig 3-12. The current state is safe. Suppose that process ti now requests a printer This request cm be granted because the resulting state is slill sate (process Dean finish, and then processes A (ii E, followed by the rest) Now imagine that after giving B one of the two icmaming printers. L wants the last printer (IrantKiy that ietjuest would icduee the vector of uvtillable resources to A0 0 0), which leads to deadlock Clearly £"s request must be deterred for a while. The banker's algorithm was first puhhshed by Diikstra in 1965 Suite th.it tune, nearly every book on operating systems has described it in detail Innumer- Innumerable papers have been written about various aspects of it Unfortunately, few. authors have had the audacity to pomi out that although m theory the algorithm is wonderful, in practice it is essentially useless because processes' rarely know in advance whai their maximum resource needs w,‖ be In addition, the number <>f processes is not fixed, hut dynamically vary in js as new users log in aiid out Furthermore, resources that were thought to be available ,an suddenly vanish (tape drives can break). Thus in piactice. few, if any. existing systems use the banker's algorithm for avoiding deadlocks. 3.6 DEADLOCK PREVENTION Having seen that deadlock avoidance is essentially impossible, because it requires information about future requests, which is not known, how do real sys- systems avoid deadlock" The answer m to go b.iek to the four conditions stated by Coffman et al. I ll>7l) i,> see it they can provide a clue. If we can ensure that -it least one of these conditions is never satisfied, then deadlocks will be structuially impossible (Havender. 1968) 3.6.1 Attacking the Mutual Exclusion Condition First let us attack the mutual exclusion condition If no resource were ever assigned exclusively to a single process, we would never have deadlocks How- However, it is equally clear that allowing two processes to wrile on the printer at the

ShC 3 6 DEADLOCK PRbVbNTION 181 same time will lead to chaos. By spooling printer output, several processes can generate output at the same time In this model, the only process thai actually requcMs the physical primer it, lh<; printer daemon Since the daemon never requests any othu, resources,, we can eliminate- deadlock fAr ihc printer Unfortunately, not all devices can be spooled (the process table does nut lend itselt well to bemg spooled) Furthermore, competition for di*k space lor spool- spooling can itself lead to

deadlock. What would happen it two processes c;ich filled up hall ot the available spooling space with output and neither wah finished pio- duc-iiif! output"](the daemon was programmed to begin pnnting even before all the ourput was spooled, the printer might lie idle if an output process decided to wait se\eral hours after the first burst of output 1'or this reason, daemons are nor- normally programmed to print only after (he complete output file is available In this case we have two processes that have each finished part, hut not all. of their out- on the disk Nevertheless, Ihere is a germ of an l{iea here that is frequently applicable Avoid assigning a resource when that is not absolutely necessary, and try to make sure that as few processes as possihle ma; actually elatm the resource 3.6.2 Attacking the Hold and Wait Condition The second of (he conditions stated h; Cullruari et al. looks slightly more promising. If we i/ati prevent processes that hold resources froin waiting lor more resources, we can eliminate deadlocks. One way to achieve this goal is to require all processes to request all their resources before starting execution If e\eiything is available, the process will be allocate^ whatever it needs and cars run to com- completion If one or more resources are busy, iiothing will be allocated and the proc- An immediate problem with this approach is that many processes do not know how many re-sources they will need until the) have started running In tact, if they knew, (he banker's algorithm could be used. Another problem is that n.'sourecs will not be used optimally with (his approach Take. ,is an example, a process that reads data from an input tape, analyzes it fof aii bom, and then writes ail output tape as well as p|ottitig the icsults Ifafi resources inu»,l be reqLiesteo in advance the process will tic up the output tape drive and the plotter for an hour Nevertheless, some mainframe hatch systems require the usei to list all ihe resources on the first line ot each joh 'l"he system then acquires all rcsouices immediately and keeps them until the joh finishes While this method puts a bur- burden on the programmer and wastes resources, it does prevent deadlocks A slightly different way to break the hold-and-wmt condition is (o require a currently holds, Then it tries to get everything it needs ali al once.

 3.6.3 Attacking the No Preemption Condition Attacking ihe third condition (no preemption) is even less promising than attacking the second one If a process has been assigned the printer and is m the middle ol printing its oulput, forcibly lakmg away the printer because a needed plotter is not available a, tricky at best and impossible at worsi. 3.6.4

Attacking the Circular Wait Condition Only one condition is lell The circular wail can be eliminated in several wavs One v^ av is siniplv io have a rule s*Winc that a piqccss is entitled only to a single resource at any moment If ir needs ,i second one, a must release ihe hrst one. For a process thai needs io copy a huge- file from a tape to a primer, this res- restriction is unacceptable Another way to avoid the circular wail is to provide i\ global numbering ot all the resources, m shown m Fig 3-13(a) Now the rule is this processes can request resources whenever ihey want io, but all requests must be made in numer- numerical order. A process may request first a printer and then a tape drive, but it may not request first a plotter and then a printer. „™s=„«,. © ® 2 aannar (I 5 CD Ron Figure With ihis rule, the resource allocation graph can never have cycles Let us see why this is true for the case of two processes, in Fig 3-|l(h) We can gel a deadlock only if A requests resource; and 8 requests resource i. Assuming i and; arc distinct resources', they will have different numbers. II (> /, then A is not allowed to request y becau.se that is lower than what il already has H / < j, then B is noi allowed to request i because thai is lower than whai il already has hither way, deadlock is impossible. With multiple processes, the same logic holds- At every instant, one of the assigned resources will be highest. The process holding that resource w(ll nevi-r ask for a resource already assigned. It will either finish, or at worst, requesl even higher numbered resources, all of which are available Eventually, it will fim-.li and free its resources At this point, some other process v,ill hold the highest resource and can also finish In short, there exists a scenario in which all proc- processes finish, so no deadlock is present.

S£C ^ t DEADLOCK PREVENTION Io3 be acquired in strictly increasing sequence and merely insist that no process request a resource lower than what it is already holding. If a process initially so there is no reason to prohibit it (rom now requesting resource I. Although numerically ordering the resources1 eliminates the problem of deadlocks it may be impossible to find an ordering that satisfies everyone When the resources include process table slots, disk spooler space, locked database records, and other abstract resources, the number <j(potential lesources and dif- teient uses may be so large th.it no ordering could possibly work. The various approaches to deadlock prevention are summarised in Fig 1-14 [_ Condition _ Approach j _Mutual exclusion ^Spool everything I I Hold and wait ^ Request all resources imtiaJly_ No preemption Take resources away , I Circular wait _ ' Order resources

numerically , Figure 3-14, SurnjiMQ ot jppro*ithc<> lo deadlock prcvennon 3.7 OTHER ISSUES In this; seciion we will discuss .1 few miscellaneous issues related to deadlocks These include two-pha^ locking, non resource deadlocks, and starva- 3.7.1 Two-Phase Locking Although both avoidance and prevention are not temhly promising in ihe gen- general case, tor specific applications, many excellent special-purpose algorithms are known. As an example, in many database systems, an operation thai occurs fre- frequently 1(, requeKfing locks, on several records and then updating all ihe locked records When multiple processes, arc running al the same time, [here 1- a real danger of deadlock. The approach often Used is called two-phase locking In the W phase, the process trie*, to lock all the records it needs, one at a lime It it succeeds, it beems the second phase, performing its updates and releasing the locks. No leal work ... done in the first phase. If during the first phase, some record is needed that n, already locked, the process |ust releases all its locks and slarts the first phase all over. In a certain

184 DEADLOCKS CHAP 1 seiwe, this approach is similar to requesting all the resources needed in advance, or at least before anything irreversible is done In some versions of two-phase locking, there is no release and restart it a lock is encountered during the hrsl phase. In these versions, deadlock can occur However, this strategy is not applicable in general. In real-time systems and process control systems, tor example, il is not acceptable to |iisi terminate a proc- process partway through because a resource is not Livaitable and start all over again. Neither is it acceptable [o Mart over it the process has read or written messages to the network, updated files, or anything else that cannot be safely repeated. The algorithm works only in those situauons where the programmer has very carefully arranged things so that ihe program can be stopped at any point during the first phase and lestarted Many applications cannot be siruciured this way 3.7.2 Nonresource Deadlocks All of our work so tar has concentrated on resource deadlocks One- process, wants something [hat another process" has and must wait until the firsi one gives it up. Deadlocks can also occur in other situations, however, including those not involving resources at all. For example, it can happen thai two processes deadlock each waiting for (he other one to do something. This often happens with semaphores. In Chap. 1 we saw examples in which a process had to do a down on iwn semaphores, lypically mutex and another one |f these are done in the wrong ()rder, deadlock can result. 3.7.3 Starvation A problem closely related

to deadlock is starvation. In a dynamic system, requests for resources happen all the time Some polnv is needed m make a deci- decision aboui who gets which resource when This policy, although seemingly lea- sonable, may lead to some processes never getting service even though they ire not deadlocked. As an example, consider allocation of the printer Imagine ihat the system uses some kind of algorithm io ensure [hat allocating the printer does not lead t,, deadlock. Now suppose that several processes all want it at once Which one should get il? One possible allocation algorithm is to give it to the process with the smallest file to print (assuming this information is available). This approach inaurni/e-, the number of happy customers and seems fair. Now consider what happens in a free, the system will look around and choose the process" wiih the shortest file It there is a constant stream of processes with short files, the process with the huge

SEC 3.7 OTHER ISSUES 185 file will never be allocated the printer, ll will simply sttive to death (be post- postponed indefinitely, even though it is not blocked). tion pohcy With this approach, the process waning the longest gets served next In due course of time, any given process will eventually hecome the oldest and 3.8 RESEARCH ON DEADLOCKS It ever there was a subject that was investigated mercilessly duru,g the early days of operating systems, it was deadlocks The ieason for this is th.ii deadlock detection is d nice little graph theory problem thai unc mathematically-inclined graduate student can gei his jaws around and chew on for 1 or 4 years All kinds i)f algorithms were devised, each one more exotic ,rnd less practical than the pre- previous one- Essentially, all this research has died out, with only a very occasional new paper appearing (e g., Karacali et al . 2000). When an operating system wants to do deadlock detection or prevention, winch tev, of them do. they use one of the methods discussed in this chapter. There is still a little research on distnhuied deadlock detection, however. We will not treat that here because (I) it is outside the scope of this book, and >2) none of it is even remotely practical in re.il systems. |ts mam function seems to be keeping otherwise unemployed graph theorists off ihc streets 3.9 SUMMARY Deadlock is a potential problem in any operating system It occn^ when a group of processes each have been granted exclusive access to some resources, and each one wants yet another resource that belongs to another process m the group. AI] of them are blocked and none will ever run again Deadlock can be avoided by keeping tmt.k ol which states .ire safe and which are unsafe. A sale state k one

hi which there exists a sequence ol events tluit The banker s algorithm avoids deadlock b\ not gr^rnmg a request if that request will put the system ,n an unsafe state. Deadlock tan be structurally prevented b> building the system m Nuch a waj that it can never occur by design For example, by allowing a process to hold only broken. Deadlock can also be prevented by numbering all the resources. ,i;id making processes request them in -.trictlv increasing order. Starvation can be avoided by a first-come, first-served allocation policy

Ifj6 DEADLOCKS CHAP 3 PROBLEMS 1, Give un example ol a deadlock taken from polities 2. Students working at individual PCs in a computer laboratory send their files to be printed bv a server which spools the files on ,1s lutti disk. Under v.hji conditions, inj> a deadlock occur if the disk space for the print spool i- limited' How may the deadlock be avoided1' 1 In [he preceding question which resources .iiv? p^cctnptaole And which .ire nonpretmpt- able' 4. In Fig Vl the resources an? relumed in the reverse ordui ot their acqdismon Would giving them back in the oilier order be just as good'1 lhat structurally violate the mode! we have used ol resource u.sagcJ It .so, j^ive -in 6. The discussion of the ostrich algorithm mentions the possibility of process rable slots 7. Consider Fig. 3-4. Suppose that in step (ol C requested S instead of requesting K Would this lead to deadlock? Suppose thai it icquested both S and ff1' 8. At a crossroads wnh STOP sijzns on all iour approaches, the rule is ih^t each driver yields the right o\ way to the driver on his right This ride is not adequate when four vehicles arrive simultaneously. Fortunately, humans are stunelimes capable ol .icting more intelligeiilly than computers and the problem is usually resolved when one driver signals the driver to his left to jio ahead Can vou diav^ Jn anak>gV between ihis behavior and any ol the wa^s ol recovering trom deadlock described m ,Si?c ^4 ^ Wnv is a problem wnh such j simple soluiion in tho huinan vvoi Id so ditf 1 icult u> jppl\ to j computer system' 9. Suppose thai in Fig 1-6 Ci(+ Rl; > EJ for some i What imphattions dues this have for all Ihe processes linishing without deadlock'.' 10. All the trajectories in l:ig. 3-8 arc hori7ont.il t>r vertic.il Cm von enMsion anj cir- circumstances in which diagonal trajectories wert also possible ' 11. Can the icsource trajectory schema ol Fig 3 8 ako be uNed U, illusiraic the piohlem ot deadlocks with three processes and three resources' II so how <_an this be done' U not. why not'1 12. In theory, resource trajeotorv ^raplis ccmld bt' Used to uvoid deadlocks Us cl^s ^i problem with actually doing [his safe state cr an unsafe Oiie" What it ihe u-quest came lrom C msicad o(/>> 14. Can

a system he in a .state that is neither deadlocked iior safe1 If sd, <*ivc an cxafnplc II in it, prove lhat all states nru either deadlocked ot sale

CHAP PROBLEMS 187 miuin of two resources, Is deadlock poss.hle.' Fxplau, your atiNwei. 16. Consider the previous problem again, bul mm wilh p processes each needing j nuv imusn of m resources and a total of r resource', available What condition must hold io make rhc ■.ysleni deailloUc free' 17. Suppose that process .4 m Fig 3-12 requests the tost tape dme Docs (his action lead io a deadlock'1 IS. A computer lias six (apt drives, with n processes competing tor them hadi process 19. | lie banker's algorithm is being run mi a system with /rt resource closes and n processes In the limn of large ,n and n. thtr number ol" operations Ihai must be pei- liumed I,i check a sldie fur wfetj is proportional io m"nh Whjt .ire the «ilues of u .tnd b ' 2A. A system has fouf proccs&cs and iive aMoeatiible resources The curreni allOLalio'i -ind TTiaximum needs *ire as follows" Allot ated Maximum Available Process A 10 2 11 I I 2 I } 00 x M Process B 2 0 110 222H Process C I I 0 I 0 2 13 10 Process D Milt) 11221 What is the sn^all^sl value t>l x for which this is is sale state? 21. A distributed system usmg iruilboxcs has two IPC primitives, send and receive The Idtier primitive specifies a process to receive from and blocks ji no message from th<it There are no .shared resources, hut processes need io conimuncoate frc^ueml^ dbuuf other matters K deadioctc possible? Discuss 22. Two processes, A and S, each need three records, 1, 2. and 3, in a daiabase. If A asks lor them m the order 1.2,3. dnd B asks tor them m the Same order, deadlock is not possible However, il B asks for them in the order 3, 2. I, then deadlock is possible With Ihree resources, ihere .ire 3-' ^r 6 possible tombinations each process can request the icsourccs What fraction nt all the combinations are guaranteed to be deadlock free-1 23. Now reconsider the above problem, but using iw,i-phasc locking Will that chminan.' the potential for deadlock1 Does il have ally other undesirable characteiistics, hou- evcr? If so. which ones' 24. in an electronic lunds transfer system, there an: hundreds ol identical processes th it work as lo'lows Path process reads an input line specifying ;in amount of money, the p.irallc/. (here is a very real danger thai has my locked .itcoum a it will be unahle Jo thai avoids deadlocks Po not release an account record until you have completed the

188 DEADLOCKS CHAP 3 transactions (]n oilier words. solutions thai lock one account and then release ii immediately il the other is locked are not

allowed) 25. One wjy to prevent deadlocks is ((. eliminate ihc h(ild-and-y>au condition hi the text i[was proposed Ihdl before asking for a new resource, a process must first release whatever resources n already holds (assuming that is possible! However doiny; so introduces the danger ihat it mav gel ihc new resource but lose some ol the existing 26. \ computer sutnLC student assigned to work oil deadkuks thinks of the itillowiny hnllidnt wa> to eliminate deadlocks When .1 process lequcMs a ies(iurce, it specifics .i time limit II die process bkxks because the lesouree i- not available ,, timer is l{ v*>u were the processor, whiit gratis womC1 you j^is'e tru^ pioposal arid \\hy 27. Cinderella Jnd the Prince arc getting divorced To div,du then property, thev have agreed on the following algorithm. Every rnornmg, wcl, urn- may send j letter io the other's lawyer requeiting one item nt properi> Since il taki's ,i da> tor letters In be delivered, they have agreed that if both discover thai they have rcqucsied ihe same item on the same day. the nexl day they will send a letier canceling the request Amonj! [heir propcrt) is their dog. Woofer. Woofer's doghouse, their canary Tweeter- and Tweeter*,, cage '1'he animals love their houses, mi i(has been agreed ihat any divi- division of property separating an animal Imni its house is invalid, requiring the whole division to start over from scratcli. Both Cinderella and the Prime desperately want Woofer So They can go on (separate) vacations, each spouse lias programmed a per- personal computer to handle the negotiation Wburt they come batk from \atalion. the computers arc still negotianng. Wh> ' U deadlock possible ' Is starvation possible' Discuss. 28. A student majoring in anthropology ami minoring in computer slichcc has embarked on a research project to sec j| African baboons can be taught about deadlocks He locates a deep canyon and fastens* a rope across it. so the baboons Liiri ert'ss hand- handover-hand. Several haboons can cross at ihe same tuiie. piovided that they' <iil all get nnlo ihe rt,pe di the simie time, a deadlock will a-suh (the baboons will get snick in the middle) because n is impossible lor one habooii n> chmh over another enc v.\\\]e suspended over ihe canyon If a baboon wants to cross the canyon, he iinisr cheek io sce that no other babiHin is currently crossing in the opposite direction Write a pro- program using semaphores thai avoids deadlock Do not \\i>(ty aboui ;r series ot easiward moving baboons holding up the westward iw-'.ng baboons indefinitely 29. Kcpeat the previous problem, but now auml starvation When a baboon that wants to unti] the rope is empiv. but no more wesiward moving baboons are allowed to stcirl until at least one baboon has crossed the other way. 30. Program a simulation of the banker's algorithm Your progr.im should

cycle through each oi ihe bank clients asking loi a reqiicsi ,md jvaliutintr uheiher it i-. \,\i'e oi unsale Output a log of requesis tind decisions io a lilt

MEMORY MANAGEMENT Memory is an important resource that must be carefully managed. While the average home computer nowadays1 has a thousand times as much memory as the IBM 7094. the largest computer in the world in ihc early 1960s, programs arc get- getting bigger faster than memories. To paraphrase Parkinson's Jaw, "Programs expand to till the memory availahle to hold them.'" In this chapter we will study how operating systems manage memory. Ideally, what every programmer would like is an infinitely large, infinitely fast memory that is also nonvolatile, that is. does not lose its contents when the electric power tails. While we are at it. why not also ask for it to be inexpensive, too'* Unfortunately technology does no! provide such memories Consequently, most compulers have a memory hierarchy, with a small amount of very last, expensive, volatile cache memory, tens of megabytes ot medium-speed, medium- price, volatile main memory (lxAM) and tetis or hundreds of Gigabytes oi slow, cheap, nonvolatile disk storage. Et is the job of the operating system to cooidmate how the.se memories are used The pjn ot the operating system that manages the metnoiy hierarchy is called the memory manager. Its |ob is to keep tiack of which parts ot memory are in use and whieh parts are not in use, to allocate memory to processes when they need it and deallocate it when they are done, and to manage swapping between mam memoiy and disk when mam memory is loo small to hold all the processes In this chapter we will investigate a number ot different memory management schemes, ranging from very simple to highly sophisticated We will start at the

190 MEMORY MANAGEMENT CHAI' 4 beginning and look firs! at the simplest possible memory management system and then gradually progress to more and more elaborate one;. As we pointed out in Chap- I. hMory lends to repeat itself in the computer world While the simplest memory management schemes are no longci useJ on desktop computers, they arc still used in some palmtop, embedded, and smart card 4.1 BASIC MEMORY MANAGEMENT Memory management systems can be divided into two classes, itiose that move processes back and forth beiween mam memory and disk during cxeculion study them firM. Later in the chapter v»e will examine swapping and paginu Throughout this chapter the re.ider should keep in mind that swapping and paune are largely artifacts caused by the l&ck of sulficient mum memory i h Id

|| iIil programs at once. If main memory evt?i gets so large that then.- is mKei u hit it, the arguments m tavor ot one kind yf memory management schein me hor may become obsolete On the other hand, at, mentioned above, software -;eems to be growing dui faster than memory, so efficient memory m.inagemeni ma\ always be needed. In the 1980s, there were many universities that ran a nmeshariiig system with do/ens of (more-or-le^s satisfied) users on a 4 MB VAX Now Microsoft recommends having at least 64 MB for a single-usei Window., 2000 system. The trend toward multimedia puts even more demands, on memory, so good memoiy management is probably going to be needed for the next decide at least. 4.1.1 Monoprogramming without Swapping or Paging The simplest possible memory management M'heme ti to run |Usl one program ut a time, sharing the memory between that progiani and the operating systetn. Three variations on this theme are shown in Fig 4-1 The operating system may be af the bottom of memory in RAM (Random Access Memory), as shown in Fig 4-l(a), or it may be m ROM (Read-Only Memory) at the top ot memory, as shown in Fig. 4-1 (b), or the device drivers may be at (he top ol memorj in a ROM and the rest or'lhe system in RAM down below, ,is shown in Fig. 4-l(c) The first model was formerly used on mainframes and minicomputers but is rarely used any more. The second model is used on tonie palmtop computers ,md embedded systems The third model w.is used by early personal computers (e g., running MS-DOS), where the porlion of the system in the ROM is called tht; BIOS (Basic Input Output System). When the system is organized m this way, only one process at a time can be running As soon as the user types a command, the operating s-ystem copies [he

BASIC MEMORY MANAGbMP.NT Ope rail ng ROW Operating RAM requested program from disk to memory and executes it When the pioeess fin- finishes, the operating system displays a prompt chaiacter and waits tor a nev, com- command When it receives the e-onunand, it loads a new program into HU'iiion, overwriting the first one. 4.1.2 Multiprogramming with Fixed Partitions Except on simple embedded systems. monopiogiaiutTting. !s hardly used any Having multiple proces.ses running a[once means that when one proi'ess is blocked waiting for I/O to finish, another onu can use the CPU Thus multipro- multiprogramming increases Ihe CPU utili?ation. Neiwuik seivers always have the ability to run multiple processes (for different clients) at Ihe same time bur most diem (i.e., desktop) machines also have this ability nowaday The easiest way to achieve multiprogiammnie is --imply to dis'ide

memory up into n (possibly unequal) partitions. This partitioning can. for example, be done manually when the system is started up When a joh arrives, it can be put into the input queue tor the smallest partition large enough to hold it. .Since the partitions are fixed in [his scheme, any space m a partition not used by a |<>b is lost. In Fig 4-2(a) we see how this system ot lixed partitions and separate input queues looks The disadvantage of sorting the incoming jobs into sepaiate queues become* apparent when the queue tor a large partition is empty but the queue !or a small partition is full, as is the <.ase tor partitions 1 and 3 in l;ig 4-2<a). Here small jobs have to wail to ge[into memory, even though plenty of memory i» free An altei- partition becomes free, the job closest to the tront of [he queue [hat fits in it could be loaded into the empty partition and run. Since it is undesirahle to waste a large

MEMORY MANAGEMENT DO- D- Figure 4-2. (a) Fix partition (b) Fixed s with sepsr.ile input quc lhu ingle jnpm queue partition on a small job, a different strategy is to search the whole input queue whenever a partition become*, free and pick the largest |oh that fits- Note that the latter algorithm discriminates against small jobs as being unworthy ol having a whole partition, whereas usually it is desirable to give the smallest iobs (otten interactive jobs) the best service, not the worst One way out is to have at least one small partition mound Such a partition ill allow small jobs to run withoui having to allocate a large partition lor them Another approach is to have a rule stating that a job that is eligible to run ma ot tie skipped over more than k times. Each time it it. skipped over, ii gels on oint When it has acquired it ponts it may not be kipped agan lay irnmg and p not tie skipped over more than k ti i pp point. When it has acquired it points, it may not be skipped again. This system, with fixed partitions set up by the operator in the riHirnmg an not changed thereafter, was used by OS/360 on large IBM mainframes for man years. It was called MFT (Multiprogramming with a Fixed number of Tasks o OS/MFT). It is simple to understand and equally snnple to implemen' •«~™-« jobs are queued unnl a suitable partition is available '-■-*■ ' — loaded into that partition and run until operating Systems, support this model. 4.1.3 Modeling Multiprogramming When multiprogramming is used, the CPU utilisation can he improved Crudely put, il the average process computes only 20 percent of the lime it is sit- sitting in memory, with five .processes in memory at once, the CPU should be busy liable, at which tir nates Nowadays, lew, it any.

SEC 4.1 BASIC MEMORY MANAGEMENT 193 all [he lime This model is unrealistically optimistic, however, since it assumes [hat all five processes will never be waning for I/O at the sarru; lime. A better model is to look at CPU usage from a probabilistic viewpoint. Sup- Suppose lhat a process spends a fraction p of its time waiting for I/O 10 complete. With n processes m memory at once, the probability that a|] i, processes are wait- waiting for I/O (in which ease the CPU will be idle) is p". The CPU utilization is then given by the formula CPU utilization = I-/'" Figure 4-3 shows the CPU utihzatic degree of multiprogramming. I 20 s a funciion of n, which is caJled ihe Figure 4-3, CPU ul,n From the figure it is clear that if processes spend SO percent ol their time waiting for I/O, at leasi 10 processes must be in memory at once to get the CPU waslc below |0 percent. When you realize [hai an interactive process waiting tur a user to type something at a terminal is in I/O wait state, it should be clear that I/O wait times of 80 percent and more are not unusual. But even in batch sys- systems, processes doing a lot of disk I/O will ofien have this percentage or more. For the sake of complete accuracy, it should be pointed out that the proba- probabilistic model jusl described I 11 n In phcitly assumes that all)i processes are independ n n h u eptahle for a system with five processes in menu h h run n nd waiting. But with a single CPU, we cannot have h p run ng so a process becom- becoming ready while the CPU is h > II I Tl he processes are not independent. A more aceur n d 1 n b d ng queuemg iheorv, but the point we are maki - I p n g I p esses Use the CPU when it would be otherwise dl — f II I I en il ihe tiue curves uf Fig 4-3 are slightly differ Even though the model f F 4 3 d d ^vertheless be used to make specific, althougl pp n r d on b it CPU performance.

MEMORY MANAGEMFNT Suppose, for example, that a computer has 32 MB or memory, with the opera ■ '«■" -.1--.L.. „, taking up 4 MB These s g raising [he CPU utilization 10 83 percent In will raise the throughput by 38 percent ing yet another 16 MB would only mcrcase C'PU utilization Ironi 83 pcr~ Adding yet another 16 MB would only mcrcase C'PU utilization Ironi 83 pcr~ cent to ^3 percent, thus rinsing the throughput by only another 12 percent Using this model the computer's owner might decide that the first addition is a good investment but th.it the second is not 4.1.4 Analysis of Multiprogramming System Performance The model discussed above can also be used to analyze batch systems. SI"- III I' IL."IUI ^ I "I ill I1—"SI ilf IIIIUUIO IIJ [H [III HI even m the absence ol competition for the CPU From 10:00 a M to 10:10 a M ,

job I is all hv ,tselt in memory and gets 2 minutes of work done. When job 2 arm'es at ifVlO AM, the CPU utilization increases from 0.20 to 0 36. due to the higher degree uf multiprogramming (see Fig. 4-U However, with mund-rohm scheduling, each job gets half ul the"cpU. so each job gels 0.18 minutes of CPU work done for each minute ,t is in memory. Notice that the addition of a second job costs the first job only |!) percent ot ik performance It goes from getting 0.20 CPU minutes per minute ol re;,] time to getting 0.18 CPU minutes per minute of real time At 10 15 am the third job anives At this puint job I has received IM minutes of CPU and job 2 h.is hud 0 9 minutes of C'PU With thiee-way multipro- multiprogramming each job gets' O.|6 minutes of CPU time per minute of real nine, js shown in Hg 4 4(b> Kom I0-I5AM to 10:2()\M each of the three jobs tjets 0 8 minutes ot CPU time At 1020 A M a lourtli job arrives. Fig 4-4|Ci shows the complete sequence of events 4.1.5 Relocation and Protection Multiprogramming introduces two essential problems that must be solved — relocation and protection. Look at Fig. 4-2 From the figure it is clear that dif- different jobs will be run at different addresses. When a program is linked (i e., the

BASIC MEMORY MANAGEMENT 1 2 3 4 10 00 10 10 10 15 10 20 4 3 2 2 CPU idle CPU busy CPU/process an 20 20 64 36 10 t>1 4y 16 .41 16 2 0 i .9 1 .8 Job 2 starts 1 9 1 3 1 8 ! 3 3 . 3 . 3 ^JobUmishes 9 1 1 i 1 9 ii i 9 1 H 7 1 I I I Figure 4-4, m) Arm aland wk requirements of four jots <b) CPU uu.liz.ti inn .md Iinish The numbers *ibove the hon/orira] lines sho\^ how ninth <7P{~ lime, main program, user-written procedures, and library procedure'; are combined into a single address space), ihe linker must know at what address the program will begin in memory. For example, suppose that the first instruction is a call to a procedure at abso- absolute address 100 within the binary file produced by the linker. If this program is loaded in partition i (a! address I00K). that instruction will jump to absolute address 100. which is inside the operating system. What is needed is a call to I00K + 100. If the program is loaded into partition 2. it must be carried out as a call to 200K + 100, and "mi on This problem is known as the relocation problem One possible solution is' to actually modily ihe instructions as the program is loaded inio memory. Programs loaded into partition | have HH)K added to each address, programs loaded into partition 2 have 200K added to addresses, and so forth- To perform relocation during loading like this, the linker musi include m [he binary program a list or bitmap telling which program words are

addresses to be relocated and which are opcode-., constants, or other items that must not be relocated. OS/MPT worked this way. Relocation during loading doe*, not solve the proieclion problem. A malicious program can always construct a new instruciion andjuinp to it. Because programs in this system use absolute memory addresses rather than addresses relative io a

MEMORY MANAGEMENT i,r writes any word in memory. In multiuser systems, it is highly undesirable to let processes read and write memory belonging to other users. The solution that IBM chose for protecting the 360 was to divide memory into blocks of 2-KB bytes and assign a 4-hit protection code to each block. The PSW (Program Status Word) contained a 4-bit key. The 36A hardware trapped any atiempi by a running process to access memory whose protection code differed trom the PSW key. Since only the operating system could change the protection codes and key, user processes were prevented from interfering with one another and with the operating system itself. An alternative solution [o both the relocation and protection problems is Us equip the machine with two special hardware registers, called the base and limit registers When a process is scheduled, ihe base register is loaded with the address of the start of us partition, and the limit register is loaded with the length ot the partiiion. Every memory address generated automatically has the base register contents added to it before being sent to memory. Thus if tlic base regis- register contains the value I00K, a CALL 100 instruction is effectively turned into a CALL 1()OK+ 100 instruction, without the instruction itself being modified. Addresses are also checked against the limit register to make sure that ihey do noi aitempt to address memory outside the current partition. The hardwaie protects the base and limit registers to prevent user programs from modifying them A disadvaniage of this scheme is the need to perform an addition and a com- comparison on every memory reference. Comparisons can he done fast, but additions are slow due to carry propagation time unless special addition circus .ire used The CDC 6600—the world's first supercomputer—used this scheme. The Intel 8088 CPU used for ihe original IBM PC used u. weaker version of this scheme—base registers, but no limit registers Few computers use il any more though. 4.2 SWAPPING With a batch system, organizing memory into fixed partition^, is simple and effective. Each j(tb is loaded into a partition when it gels lo the head ol the queue memory to keep the CPU busy all the time, there- is no reason to use anything more complicated. With timesharing systems w graphically oriented peisonal

computers, the situation is different. Sometimes there is noi enough main memory to ho|d all ihe currently active processes, \o excess processes must be kept on disk and brought in to run dynamically. Two general approaches to memory management can be used, depending (m pan) on the available hardware. The simplest strategy, called swapping, consists

SWAPPING 197 of bringing in each process in its entirely, running it for a while, [hen putting it back Un the disk. The other strategy, called virtual memory, allows programs to run even when they are only partially in main memory. Below we will study swapping: in Sec. 4.3 we will examine virtual memory. The operation of a swapping system is illustrated m Fig. 4-5. Initially only process A is m memory Then processes 6 and C are created or swapped in from disk. |n Fig 4-5(d> A \f. swapped out to disk- Then D comes in and fi goes out. Finally A conies in again Since A is new at a different location, addresses con- contained m it must be relocated, either by software when it is swapped in or (more likely) by hardware during program execution. Operating | Qpeoting Opeiattng | Operating]ea\c u The shaded regions 4re unused nicmor\ The main difference between the fixed partitions of Fig. 4-2 and the variable partitions of Fig. 4-, ,s that the number, location, and size of the partitions vary dynamically jn the latter as processes come and go, whereas ihey are fixed in the former The flexibility of not being tied to a fixed number of partitions that may be loo large or loo small improves memory utilization, but i(also complicates allocating and deallocating memory, as well as, keeping track of it. When swapping creates multiple holes in memory, it is possible to combine them all inio one big one by moving all the processes downward as far as possi- possible This technique is known as memory compaction. It is usually not done because it requires a lot of CPU time. For example, on a 256-MB machine that can copy 4 bytes in 40 nsec, it takes aboul 2.7 s~ec to compact all of memory. A point that is worth making concerns how much memory should be allocated for a process when it is created or swapped jn If processes are created with a fixed size that never changes, then ihe allocation is simple- ihe operating system allocates exactly what is needed, no more and no less.

198 MEMORY MANAGEMENT CHAP 4 li, howevei, presses' data segments can grow, tor example, by dynamically allocating memory from a heap, as m many piogramming languages, a problem ot-curs whenever a process lues lo grow If J hole is adjacent to the process, n can be allocated and the process

allowed to grow into the hole On the other hand, if the process is adjacent to another process, the growing process will either have to be moved to a hole in memory large enough ft,i ii. or one or moiu piocesses will have to lie swapped out to cieate a large enough hole If a piocess cannot glow in memory and the swap area on the disk K full, the process will have io wait or be killed. II it is expected that most processes will grow as they run, u is probabl; .i good idea to allocate a little extra memory whenever a process is swapped in oi iiiovud, to reduce the overhead associated with moving or swapping processes [hat no longer fit in their allocated memory, Howuver. when sapping processes to the extra memory as well. In Fig. 4-6(a) we see a memory configuration in whkh space for growth has been allocated lo two processes. > Room for growth Figure 4-6. I If processes can have two growing segments, for example, the data segment being used as a heap for variables that are dynamically allocated and released and a stack segment for the normal local variables and return addresses, an alternative arrangement suggests itself, namely that of Fig. 4-6<b). In this figure we see th.il each process illustrated has a stack at the top of us allocated memory that is grow- growing downward, and a data segment just beyond the program text that u, growing

upward The memor> between them can be- uicd for cither segment. It it rims out, cilher the piocess will have to be moved to a hole with enough space, swapped out yf memory until a large enough hole can be created, or killed. 4.2.1 Memory Management with Bitmaps When memory is assigned dynamically, the operating system inusi manage it. In general terms theie are two ways to keep track of memory usage- bitmaps and free lists- In this section and the next one we will look at these two methods in small a .1 few words and perhaps, as large as several kilobytes. Correipijiiding to each allocation unit is a bil m ihe bitmap, which is 0 if the unil is free and I it n is occupied (Or vice versa). Figure 4-7 shows part ol memory and the corresponding bitmap Figure 4-7. (j) A pjn ol men marks show the memory Mo, jre ULce (h) Tht toricspondin ie. The smaller the llocatio :mory The size of the allocation unit is an nnpottanl dt allocation unit, the larger the bitmap. However, even ..mi un mimauun u... small as 4 byies, 32 but- of memory will require t)nly I bit of ihe map A merr of 32n bits will use n map bits, so the bitmap will take up only |/33 of memory u the allocation unit i.s chosen large, the bitmap will be smaller, but appreciable memory may be wasted in the last unil of the process if the process size is noi an exact multiple of the allocation unit. A biimap

provides a simple way to keep track of memory words in a fixed amount of memory because the size of Ihe bitmap depends only on the size of memory and the size of the allocation unit The main problem with it is thai when

MEMORY MANAGEMENT CHAP 4 it has been decided to bring a k unit process into memory, the memory manager must search the bitmap to find a run of k consecutive 0 bits m the map Searching a bitmap for a run of a given length is a slow operation (because the run may straddle word boundaries in the map); this is an argument agamst bitmaps 4.2.2 Memory Management with Linked Lists Another way of keeping track of memory n to maintain a linked list of allo- allocated and free memory segments, where a segment is either a process or a hole between two processes. The memory of Fig. 4~7(a) is represented in Fig 4-7(c) as a linked list of segments. Each entry m the list specifies a hole (H) or process (P), the address at which it starts, the length, and a pointer to the next eniry In this example, the segment list is kept sorted by address. Sorting this way has the advantage that when a process terminates or is swapped oui. updating the list is straightforward. A terminating process normally has two neighbors (except when it is at the very top or bottom of memory). These may be either processes or holes, leading to the four combinations of Fig. 4-8. In Fig. 4-8(a) updating the list requires replacing a P by an H. In Fig. 4-8{b) and Fig. 4-8{c), two entries arc coalesced into one, and the list becomes one entry shorter In Fig- 4-8<d), three entries are merged and two items are removed from the list. Since the process table slot for the terminating process will normally point to the list entry ior the process itself, it may be more convenient to have the list as a double-linked list, rather than the single-linked list of Fig. 4-7(c) This structure makes it easier r<> find the previous entry and to see if i a) », <") A * X B * YM, X j B Figure 4-8. Four neighbor corrttnnaiions for fhe icrminaung process. X When the processes and ho|e« are kept on a list sorted by address, several algorithms can be used to allocate memory for a new|y created process (or an existing process being swapped m from disk). We assume that the memory manager knows how much memory to allocate The simplest algorithm is first fit. The memory manager scans' along the list of segments until it finds a hole that is big enough. The hole is then broken up into two pieces, one for the process and one for the unused memory, except jn the statistically unlikely case of an exact fit. First fit is a fast algorithm because it searches as little as possible.

SfcC 4 2 SWAPPING 2**1 A minor variation or first fit is next fit. It works Che same way as firM fit, except that it keeps track of where u is whenever it finds a suitable hole. The next time it is called to find a hole, it stans searching the list from the place where it left oft last time, instead of always at the beginning, as first fit does. Simulations by Bays A977) show that next fit gives slightly worse performance than first fit, Another well-known algorithm is best fit. Best fit searches the entire list and takes the smallest hole that is adequate. Rather than breaking up a big hole that might be needed later, best fit tries to find a hole that is close to the actual size needed As an example of first fit and best fit, consider Fig. 4-7 again. If a block of size 2 is needed, first fit will allocate the hole at 5, but best fit will allocate the hole at 18. Best fit is slower than first fit because it must search the entire list every time it is called. Somewhat surprisingly, it also results in more wasted memory than first fit or next fit because it tends to fi‖ up memory with tiny, useless holes. First fit generates larger holes on the average. To get aruund the problem of breaking up nearly exact matches into a process and a tiny hole, one could think about worst fit, that is. always take the largest available hole, w that the hole broken off will he big enough to be useful. Simu- Simulation has shown that worst fit is not a very good idea either. All four algorithms can be speeded up by maintaining separate lists for processes and holes. In lhis way, all of them devote their full energy to inspecting holes, not processes The inevitable price that is paid for this speedup on alloca- allocation is the additional complexity and slowdown when deallocating memory, since a freed segment has to be removed from the process list and inserted into the hole list. If distinct lists are maintained for processes and hole*,, the hole hst may be kept sorted on size, to make best fit faster When best fit searches a list of holes from smallest to largesi, as soon as it finds aho|e that fits, u knows that the hole us the smallest one that will do the job, hence the best fit. No further searching is needed, as it is with the single list scheme. With a hole list sorted by size, first fit and best fit are equally fast, and next fit is pointless. When the holes are kept on separate lists from the processes, a small optimi- optimization is possible. Instead of having a separate set of data structures for maintain- maintaining the hole list, as is done in Fig. 4-7(c), the holes themselves can be used. The first word of each hole could be the hole size, and the second word a pointer to the following entry. The nodes of the list of Fig, 4-7(c), which require three words and one bit (P/H), are no longer needed Yet another allocation algorithm is quick fit, which maintains separate lists for some of the more common sizes

requested. For example, it might have a tabie with « entries, in which the first entry is a pointer io the head of a list of 4-KB holes, the second entry is a pointer to a list of 8-KB holes, the third entry a pointer to 12-KB holes, and so on. Holes of say, 21 KB, could either be put on the 20-KB

202 MEMORY MANAGEMENT <nHAP 4 list or on a special list ol odd-sized holes With quick fit. finding a hole or the required size is extremely fast, bui il has the same disadvantage as all schemes that sort by hole size, namely, when a prices'- terminates or is swapped out, find- finding us neighhors to sec if a merge is possible is expensive If merging is not done, memorv will auiuklv tracment into a large number of small holes into which m, 4.3 VIRTUAL MEMORY Many years ago people wvre first confronied with programs that were too big to lil in the available memory The solution usually adopted was 10 split the pro- program into pieces, tailed overlays. Overlay 0 would start running first When it plex, allowing multiple overlays in memory at once. The overlays were kept t)n the disk and swapped in and out of memory b> the operating system, dynamically, a*, needed. Although the actual work of swapping overlays in and out was done by the <;ysienu the work of splitting the program into pieces had to be done by the pro- programmer. Splitting up large programs; into small, modular pieces was time con- consuming and boring. It did not lake long before someone thought of a way to iuiii the whole job over to the computer. The method that was devised (Fothenngham, 1961) ha^, come to be known as virtual memory. The basic idea behind virtual memory is that the combined size of the program, data, and ^lack may exceed the amount of physical memory avail- available for it. The operating system keeps those parts of thi* program currently in use in mam memory, and the rest on Ihe disk h>f example, a 16-MB program can run on a 4-MB machine by carefully choosing which 4 MB to keep in memory al each instant, with pieces of the program being swapped between disk and memory as needed. Virtual memory can also work in a multiprogramming system. With hits and of itself to be brought in. jt Is waiting for I/O and cannot run, so the CPU can be 4.3.1 Paging Most virtual niemoiy systems use a technique called paging, winch we will now describe. On any computer, there exists a set of memory addresses that pro- programs can produce. When a program uses an instruction like MOV REG,1000 it docs this to copy the contents of memory address 1000 to REG (or vice versa.

S]-(-] 1 VIRTUAL MEMORY depending on the computer) Addresses can be ger registers, segment registers- and other wjjs Figure 4-9. The position and iimcum ot ihi- MMU Heic ihc MM1' is shown as being a parl ill the CPU chip became n cmnwnly is nowadays How(S\cr. logically n could be a ^sparaitf chip Jiitl w.is in years £tine bt These prog ram-gene rated addresses arc called virtual addresses and Form [he wrtual address space On computers without virtual memory, ihe virtual address is put directly onto the memory bus and causes the physical memory word with ihe same address to be read or written. When virtual memory is used, the virtual addresses do not go directly to the memory bus. Instead, they go to un MMli (Memory Management Unit) that maps the virtual addresses onto the physical memory addresses as illustrated in Fig. 4-9. A very simple example of how this mapping works it- shown in fig 4-li) In this example, we have a computer that can generate 16-bit addresses, from 0 up to 64K. These are ihe virtual addresses This computer, however, has only M KB of physical memory, so although 64-KB programs can be written, they cannot be loaded into memory' in their entirety and run A complete copv of a program's core image, up to 64 KB, must be present on the disk, however, so that pieces caw be brought in as needed. The virtual address space is divided up into units" called pages. The corre- corresponding units In the physical memory are called page frames The pages and page frames are always the same size. |n this example they are 4 KB, but page sizes; from 512 bytes to 64 KB have been used m real systems. With 64 KB of virtual address space and 32 KB of physical memory, we gel 16 virtual pages and 8 page frames. Transfers between RAM and disk are always in units of a page. When the program tries lo access address 0, for example, using the instruction MOV REG.O virtual address 0 ,s sent to the MMU. The MMU sees (hat this address falls

MEMORY MANAGEMENT 60K-64K 56K-60K 5SK-56K 4BK-52K 44K-48K 40K-44K 36K-40K 32K-36K 2BK-32K 24K-28K 20K-24K 16K-20K 12K 16K 8K-12K 4K-BK 0K-4K X X X X 7 X 5 X X X 3 4 0 e 1 2 \\ V \ v/ 28K-32K 24K-2BK 20K 24K 16K-20K 12K |6K 8K-12K 4K-8K k 0K-4K Figure 4-10. The rcl.mon beiwoen dresses is given by ihe page uble itlual a<| <lre nemory J in page 0 @ to 4095), which according to its, mapping is page frame 2 (8192 to 12287). It ihus transforms the address to 8192 and outputs address 8192 onto the bus. The memory knows nothing at all about the MMU and just sees a request foi reading or writing address 8192, which it honors

Thus, the MMU has effective!} mapped all virtual addresses between 0 and 4095 onto physical addresses 8192 to 12287. Similarly, an instruction MOV REG.8192 is effectively transformed into MOV n£G,24576 because viftua! address 8I92 is in virtuai page 2 and this page is mapped onto physical page frame 6 (physical addresses 24576 to 28671). As a third eSampte. virtual address 20500 is 20 bytes from ihe start of virtual page 5 (virtual addresses 20480 to 24575) and maps onto physical address 12288 + 20= 12308 By itself, this ability to map the |6 virtual pages onto anv of the eight page frames by setting the MMU'i map appropriately does not solve the problem that the virtual address space is larger than the physical memory. Since we have only

 eight physical page frames, only eight of (he virtual pages in Fig. 4-10 are mapped onto physical memory. The other-, shown an a cross in the ligure, are not mapped In the actual hardware, a Present/absent bit keeps track of which pages are physically present in memory. What happens if the program tries to use an unmapped page, tor example, hy using the instruction MOV REG.32780 which is byte 12 within virtual page 8 (starting at 32768) ? The MMU notices that the page is unmapped (indicated by a cross in the figure) and causes the CPU to trap to the operating system. This trap is called a page fault. The operating sys- system picks a little-used page frame and writes its contents back to the disk It then fetches the page just referenced into the page frame just freed, changes the map, and restarts the trapped instruction. For example, if the operating system decided to evict page frame 1, it would load virtual page 8 at physical address 4K and make (wo changes to the MMU map. First, it would mark virtual page I's entry as unmapped, to trap any future accesses to virtual addresses between 4K and 8K. Then it would replace the cro s in virtual page 8's entry with a I, so that when the trapped instruction is re- execufed, it will map virtual address 32780 onto physical address 4108 Now let us look inside the MMU to see how jt works and why we have chosen to use a page size that is a power of 2. In Fig. 4-11 we hee an example of a virtual address, 8196 (OOlOOOOOOOOOOlOO m binary), being mapped using the MMU map of Fig. 4-10. The incoming 16-bii virtual address is, split into a 4-bit page number and a 12-bit offset. With 4 bits for the page number, we can have 16 pages, and with 12 bits for the offset, we can address all 4()96 bytes within a page The page number is used as an index inio the page table, yielding the number of the page frame corresponding to that virtual page. If the

Present/absent bit is 0, a trap lo ihe operating system is caused If the bit is I, the page frame number found in the page table is copied to ihe high-order 3 hits of the output register, along with the 12-bit offset, which is copied unmodified from the incoming virtual address. Together they form a j5-bit physical address. The output register is then put onto the memory bus as the physical memory address. 4.3.2 Page Tables In the simplest cas s as we have just described it. The virt number (high-order bits) and an offset (lo g In the simplest case, the mapping of virtual addresses onto physical a is as we have just described it. The virtual address is split into a virt number (highorder bits) and an offset (lowode bts) For example w ddresses page

MEMORY MANAfiEMFNT |o|o|i|ojo|o)olo|o|o|o|o|o|i|o)o| t Outgoing B4590| Figure 4-11. The jn al ope.a, I the MMU with 16 4-KB p. The virtual page number is used as an index into the page table to lind ihc entry for lhat virtual page. From the page table entry, the page Irame number (ii any) is found. The page (rame number is attached to the high-Order end o(the offset, replacing the virtual page number, to lorm a physical address that can be sent to ihe memory. The purpose of the page table is to map virtual pages onto page Irames Mathematically speaking, the page table is a function, with the virtual page number as argument and the physical frame number as result. Using the result ol this function, the virtual page field in a virtual address can be replaced by a page frame field, thus forming a physical memory address Despite this simple description, two major issues must be faced: 1. The page table can be extremely large. 2. The mapping must be fast,

Sbt 4 3 VIRTUAL MEMORY 207 The firsl point follows dom the fact that modern computers use vinual addresses or at least 32 hits. With, say. a 4-KB page size, a 32-bit address space has I mil- million pages, and a 64-bit address space has more than you waul to contemplate. With I million pages- in the virtual addiess space, the page table must have 1 mil- million entries. And remember that each process needs its own page table (because it has it- own virtual address space). The second point is a consequence of the tact that the virtual-to-physical map- mapping must be done on every memory reference A typical instruction has an instruction word ind often a memory operand as well. Consequently, it is neees- instiuction takes sav 4 n,e^ tht page tahk lookup must be done in under I itsec to ivoid becoming a iiiaior bottleneck The nt-Ld tor large tist paje mipptne is a significant constraini on the wa> computers are

built. Although the problem is most serious with rop-nf-lhe-line pnee/pertormance ratio are critical. In this section and the following ones. v»e will look at page table design in detail and show a number ol haidware solutions that have been uj-eci in actual computers Tht simplest desian i it least t<->ni_eplu dl\ ) ts to have a single page table (on- indcxed b> virtual page number, as shown in Fig 4-11. When a process is started up, the operating system loads the register's with the process1 page table, taken from a copy kept in main memory. During process execution, no more memory references are needed lor the page table. The advantages ol this method are th.n it is straightforward and requires no memory relereni.es during mapping A disad- disadvantage is thai it is potentially expensive (it the page table is largej. Having to load the full page table at every context (.witch hurts performance. At the other extreme, the page table can be entirely in mam mentor) Ml the hardware needs then is a single legister that points to the Mart of the page table This design allows the memory map to be changed at a context switch bv reload- reloading one register Of course, it has the disadvantage of requiring one or more memory references to read page table entries cluring the execution ol each instiLk- lion. Tor this reason, this approach is rarely used in its, most pure torni. but belov> we will study some variations that have much betier performance Multilevel Page Tables To get around the problem of having u> stoic huge page tables in mernoiy all the time, many computers use a multilevel page table. A simple example" is shown in Fig. 4-12. |n Fig. 4-l2{a) we have a 12-bit virtual address that is parti- partitioned into a 10-bit PTI field, a 10-bit PTI field, and a 12-bit Offset field Since offsets are 12 bin,, pages are 4 KB. and there are a total of 220 of (hem.

208 MEMORY MANAGEMENT CHAP 4 BrtS 10 PT1 10 PT2 12 Offset | / lf]M H t 4 D1 "J_ 1 ^—" 1023 6 5 4 3 2 1 0 Second-level page tables I ; : ; — ^ - m —^ -^ —^~ -^ -*- —+~ —*- Page table tor the top 4Mof memory • »■ _^ pages —»- - > (b) Figure 4-12. (a) A 32-bit address with Iwo page table fields (b) Two-level page tables. The secret to the multilevel page table method is to avoid keeping all the page tables in memory all the time. In particular, those that are not needed should not be kept around. Suppose, for example, that a process needs 12 megabytes, the bottom 4 megabytes of memory for program text, the next 4 megabytes for data, and ihe top 4 megabytes for the stack. In between the top of the data and the bot- bottom of the stack is a gigantic hole that is not used. In Fig. 4-12(b) we see how (he two-level page table works in

this example. On the left we have the top-level page table, with 1024 entries, corresponding to

SEC 4 3 VIRTUAL MFMORY' 209 (he]O-bit PTI field. When a virtual address is, presenied to the MMU, it first extracts the pTI field and uses this value as an index into Ihe top-level page lab\e. Each of these 1024 entries represents 4M because the entire 4-gigabyte (i.e.. 32- bit) virtual address space has been chopped into chunks of 1024 bytes. The entry locaied by indexing into the top-level page table yields the address or the page frame number of a second-level page table. Entry 0 of the top-level page table points to the page table for the program text, entry I points to the page table for the data, and entry 1023 points to the page table for the Stack. The other (shaded) entries are mtf u^ed. The PT2 field is now used as an index into the selected second-level page table to find the page frame number for \he page itself. As an example, consider the 32-bit virtual address 0x00403004 D.206.5% decimal), which is 12,292 bytes into the data. This virtual address corresponds to PT I - 1, PT2 = 2, and Offset =4. The MMU first uses PTI to index into the top-level page (able and obtain entry I, which corresponds to addresses 4M to 8M It then uses PT2 to index into the second-level page table just found and extract entry 3, which corresponds to addresses 12288 to 16383 within its 4M chunk d e., absolute addresses 4,206,592 to 4,210,687). This entry contains the page frame number of the page containing virtual address, 0x00403004. If that page is not in memory, (he Present/absent bit in the page table entry will be zero, causing a page fault. If the page is in memory, the page frame number taken from the second-level page table is combined wiih the offset D) to construct a physical address. This address is put on the bus and sent to memory. The interesting thing to note about Fig. 4-12 k that although the address space contains over a million pages, only four page tables are actually needed: the top- level table, and the second-level tables for 0 to 4M, 4m to 8M, and the top 4M. The Present/absent bits in 1021 entries ot the top-leve\ page table are set to 0, forcing a page fault if they are ever accessed Should this occur, the operating system will notice that the process is trying to reference memory that it is not sup- supposed to and will take appropriate action, such as sending it a signal or killing tt. In this example we have chosen round numbers for the various sizes and have picked PTI equal to PT2 but in actual practice other values are also possible, of course. The two-level page table system of Fig. 4-12 can be expanded to three, four, or more levels.

Additional levels give more flexibility, but it is doubtful that the additional complexity is worth it beyond three levels Structure of a Page Table Entry Let us now turn from the structure of the page tables in the large, to the details of a single page table entry. The exact layout of an entry is highly machine dependent, but the kind of information present is roughly the same from machine to machine. In Fig. 4-13 we give a sample page table entry- The size varies from computer to computer, but 32 bits is a common size. The most important field \>

210 MKMORY MANAGEMENT CHAP. 4 the Page fmiw numb,; After all, the goal of the page mapping is to locale this value- Next to it we have the Present/absent hit. If this bit is I. the entry is valid and can be used. II it is 0. the virtual page to which Ifie entry belongs is not currently in memory Accessing a page table entry with this bit set to 0 causes a page fault Caching figure 4-13^ A lypit.il p.i^c lable cnlit The Protection bits tell what kinds of access are permitted. In [he simplest form, this field contains' I bit, wilh 0 for read/write and I for read only. A more sophisticated arrangement is h.iving 3 bits, one hit each for enabling reading, writ- writing, and executing the page. The Modified and Referenced bus keep track of page usage. When a pasc is written to, the hardware automatically sets the Modified hit This bit is of value when the operating system decides to reclaim a pace liaine If the page in it has been modified (i.e., is "dirty"), it must be written buck to the disk II it has noi been modified (i.e . is- "clean"), il can just he abandoned. s!ucc the disk copy is still valid. The bit is sometimes called the dirty bit, since it reflects the page's stale. The Roieii'iiccd bit is set whenever a pa^e is releienc^d, eithci ior leading or writing. Its value is to help the operating system choose a page to evict svhen ,i page fault occurs. Pages that .ue not being used are bettei candidates than pages that are, and this hit plays an important role in several nt the page ieplacl-iiieiu algorithms thai we will study laiei in this chapter. Finally, the last bil allows caching to be dis.ihled for the page Tins feature is important for pages thai map onto device legis'ters rathei ih.iu memoiy If the operating system is sitling in a tight loop waiting for some I/O device to respond word from the device, and nof use an old cached copy. With this bit, caching can be turned olr. Machines that have a separate I/O space and do not use memory mapped I/O do not need [his bit Note that the disk addiess used to hold the page when it is noi in memory is not part of the page table. The reason is simple The page table holds onlv that information the hardware needs to

translate a virtual address to a physical address. Information the operating system needs to handle page faults is Kept in software tables inside the operating system. The hardware does not need it.

SEC 4 3 VIRTUA] MEMORY 211 4.3.3 TLBs—Translation Lookaside Butters In most paging scheme.;, the page tables are kept in memory, due to llieir large si?e Potentially, this design has an enormous impact on performance Con- Consider, lor example, an instruction thai copies one register to another. In the absence ol paging, this instruction makes only one memory reference, to fetch the instruction Wich paging, additional memory references will be needed to access ilw page table Since execution speed is generally limited by the rate the CPU can yet instructions and daU out of the memory, having to make two page table refer- references per memory refeience reduces performance by 2/3 Under these conditions1. Computer designers have known ahout this problem for years and have come ii|i with a solution. Their solution is based on the observation that niost programs tend to make a large number of references to a small number of pages, and not the other way around. Thus only a small fraction o(the page table entries are heavily read; the rest arc barely used at all The solution that has been devised is to equip computers with a small hardware device for mapping virtual addresses to physical addresses, without going through the page table. The device, called a TLB (Translation Lookaside Buffer) or sometime- an associative memory, is illustrated m Fig 4-14. It is usually inside the MMU 4nd consists of a small number of entries, eight in this example, but rarely mure than 64. Each entry contains information about one page, including the virtual page number, a bit that is set when the page is modi- modified, the protection aide (read/write/execute permissions|. and the physical page frame in which the paye is located. These fields have a one-to- one correspon- correspondence with the fields in the page table. Another bil indicates whether the entry is . Valid Virtual page Modified j Protection Page frame Figure 4-14. A Tl B to speed "P paging An example that might generate the TLB of Fig. 4-14 is a process in a loop that spans virtual pages 19, 20, and 21, so these TLB entries have protection codes

212 MEMORY MANAGEMENT CHAP 4 (or leading and executing. The main data currently being used (say. an army being processed) are on pages 129 and 130 Page 140 contains the indices ^ed m the array calculations. Finally, the stack is on pages 860 and 861 Let us now see how the TLB functions. When a

virtual address i-s presented to the MMU for translation, the hardware first checks to sec if its virtual page number i<s present in the TLB by comparing u to all the entries simultaneously (i e., in parallel) If a valid match is found and the access does not violate thepio- tection bits, the page frame is taken directly from the TLB, without going to the page table. If the virtual page number is present in the TLB but the instruction is trying to wriie on a read-only page, a protecnon fault is generated, the same way a> it would be from the page table itself The interesting case is what happens when the virtual page number is not in (he TLB. The MMU detects the miss and does an ordinary page table lookup It then evicts one of the entries from the TLB and replaces n with the page table entry just looked up. Thus if that page is used again soon, the second time it will result in a hit rather than a miss. When an entry is purged from the TLB, the modified bit is copied back into the page table entry in memory. The other values are already there. When the TLB is loaded from the page table, all the fields are taken from memory. Software TLB Management Up until now, we have assumed that every machine with paged virtual mem- memory has page tables recognized by the hardware, plus a TLB In this design, TLB management and handling TLB faults are done entirely by the MMU hardware. Traps to the operating system occur only wheii a page is not in memory. In the past, this assumption was true. However, many modem R|SC machines, including the SPARC. MIPS, Alpha, and HP PA. do near]) all of this page management in software. On these machine*, the TLB entries are esphcitly loaded by the operating system. When a TLB miss occurs, instead of the MMU just going to the page tables to find and fetch the needed page reference, it just generates a TLB fault and tosses the problem into the lap of the operating system. The system must find the page, remove an entry from the TLB, enter the new one, and restart the instruction that faulted. And, of course, an of this must be done in a handful of instructions because TLB misses occur much more frequently than page faults. Surprisingly enough, if the TLB is reasonably large (say, 64 entries) to reduce the miss rate, software management of the TLB turns out to be acceptably effi- efficient. The main gain here is a much simpler MMU. which frees up a considerable amount of area on the CPU chip for caches and other features that can improve performance. Software TLB management is discussed by Uhhg et a!. A994) Various strategies have been developed to improve performance on machines that do TLB management in software. One approach attacks both reducing TLB

VIRTUAL MKMORY 213 tal, 1994) misses and reducing the co.st of a TLB miss when it does occi To reduce TLB misses, sometimes the operating system can use its intuition to figure out which pages are likely to be used next and to preload entries for them in the TLB. For example, when a client process sends a message to a server process on the saim; machine, it is very likely that the server will have to run soon. Knowing this, while piocessing the trap to do the send, the system can also check ti, see where the scivei's code, data, and stack pages are and map them m before they can cause TLB faults. The normal way to process a TLB iims. whether in hardware or m software, is to go to the page table .md perform the index,ng operations to locate the page lefcrenced The prDblcn, with doing this search in software is that the pages hold- holding the page table m<i\ m,t be in the TLB, which will cause additional TLB faults during the processing. These faults can be ieduced by maintaining a large (e.g. 4-KB) software cache ot TLB entries in a fixed location whose page i.s always kept m the TLB. By first checking the software cache, the operating system can substantially reduce Tl ,B misses 4.3.4 Inverted Page Tables Traditional page tables ot the type described so far require one entry per vir- xed by virtual page number. If the address space milli s per page, then over num. the page table size is prohably doable. ?ecome more common, the situalio now 2M bytes, with 4-KB pages, i entry is 8 bytes, the table p page table be at least 4 tual page, ... -.., _.. „ consists of V1 bytes, with 4096 byte entries are needed. As a bare minimi megabytes. On larger systems, this si; However, as 64-bit computers be drastically. If the address space is m gigabytes, Tying up 30 million gigabytes just for the page table is not doable, r now and not for years to come, if ever Consequently, a different solution is needed for 64-bit paged virtual address spaces. One such solution is the inverted page table In this design, there is one entry per page frame in real memory, rather than one entry per page ot virtual address space. For example, with 64-bit virtual addresses, a 4-KB page, and 256 MB of RAM, an inverted page table only requires 65,536 entnet,. The entry keeps track of which (process, virtual page) is located m the page frame Although inverted page tables save vast amounts of space, at least when the virtual address space is much larger than the physical memory, they have a seri- serious downside: virtual-to-physica] translation becomes much harder. When proc- process n references virtual page p. the hardware can no longer find the physical page by using p as an index into the page table Instead, it must search the entire inverted page table for an entry (>>• p). Furthermore, this search mua be done on every memory reference-,

no! just on page faults. Searching a 64K table on every memory reference is not the way to make your machine blindingly fast,

MPMORY MANAGHMENl CHAP Inverted page tables are currently used on some IRM and Hewlett-Packard workstations and will become more common as 64-hit machines become wide- widespread. Other approaches to handling largia virtual memories can be found in (Huefc and Hay*,. 1993; Talluri and Hill. 1994, and Talluri et aU 1995). 4.4 PAGE REPLACEMENT ALGORITHMS When a page fauli occurs, the operating system has to choose a page to re- remove from memory to make room for the page thai ha? to be brought m If the page to be removed has been modified while m memory, u must he rewritten to the disk to bring the di*,k copy up to date |f. however, the page has not been changed (e.g., it contains program text), the disk c».,py is already up to date, so no rewrite is needed. The page to be read m jusl overwrites the page being evicted While it would be possible to pick a random page to evict at each pagu fault system performance is much better if a page that is not heavily used is chosen. It

SEC 4 4 PAGE REPLACE MFNT ALGORI THMS 215 a heavily used page k removed, it will probably h.ive to be brought back in quickly, resulting in extra overhead. Much work has been done on the subject of page replacement algorithms, both theoretical and experimental. Below we will describe some o(the most important algorithms. It is worth noting that the problem of "page replacement" nccim> in other areas of computer design as well. For example, most computers have one or more memory caches consistmg of recently used 32-byte or 64-byte memory blocks. When the cache is full, some block has t<> be chosen tor removal. This problem is precisely the same as page replacement except on a shorter time scale (it has to be done in a few nanoseconds-, not milliseconds as with page replacement). The rea- reason lor the shorter time scale is that cache block misses arc satisfied from main memory, which has no seek time and no rotational latency A second example is in a Web server. The server can keep a certain numbei of heavily us,ed Web pages in its, memory cache. However, when the memory cache is full and a new page is referenced, a decision has to be made which Web page to evict. The considerations are similar to pages of virtual memory, except for the fact that the Web pages are never modified in the cache, so there is always a fresh copy on disk. In a virtual memory system, pages in mam memory may be either

clean or dirty. 4.4-1 The Optimal Page Replacement Algorithm The best possible page replacemenl algorithm is easy to describe bul impossi- impossible to implement. It goes like this At the moment that a page fault occurs, -time set of pages is in memory. One of these pages will be reiereiiccd on the very next instruction (the page containing that instruction). Other pages may nut be refer- referenced until 10, 100, or perhaps 1000 instructions later Bach page can be labeled with the number uf instructions that w,‖ be executed belore that page is lirst leferenced. The optimal page algorithm simply sa;s that the page with (he highest label should be removed. If one page will not be Used tor 8 million instructions and another pagt will not be Usiid lor 6 million instructions removing [be former pushes the page fault that will letch it back as tai into the future as po.wib;e. Computers-, like people, try to put off unpleasant events lor as long as they can. The only problem with this algorithm k that i[is unrealizable At the time of the page fault, Ihc operating system has no way of knowing when each of the pages will be referenced ncn (We saw a similar situation earlier with the shor- shortest joh first scheduling algorithm—how e,in the system tell which |ob ts shor- shortest'1) StilL by running a progmui on a simulator and keeping track of all page references, it is possible to implement optimal page replacement on the second run by using the page reference information collected timing the firsi run In this way it is possible to compare the performance of realizable algorithms wnh the best possible one. If an operating system achieves a performance ot. say.

 only I percent worse than the optimal algorithm, effort spent in looking lor a better algorithm will yield at most a I percent improvement To avoid any possible contusion, it should be made clear thai this log of page references refers only to the one program just measured and then with only one specific input. The page replacement algorithm derived from it is thus specific to that one program and input data Although this method is useful for evaluating page replacement algorithms, it is of no use in practical systems. Below we will study algorithms that arc useful on real systems. 4.4.2 The Not Recently Used Page Replacement Algorithm In oider to allow the operating system to collect useful statistics about which pages are being used and which ones are not. most computers with virtual mem- memory have two status bits associated with each page. R is set whenever the page is referenced (read or written). M is set when the page is written to (i.e , modified). The bits are contained m each page table entry, as

shown in Fig 4-13 It is impor- important to realize that these bits must be updated on every memory reference, so it is essential that they be set by the hardware Once a bit has been set to I, it stays 1 until the operating system resets it to 0 in software If the hardware does not have these bits, they can be simulated as follows. When a process is started up, all of its page table entries are markc,! as not in memory. As soon as any page is referenced, a page fault wi]| occur. The operat- operating system then sets- the R bit (in ifs internal tables), changes the page table entry to point to the correct page, with mode READ ONLY, and restarts the instruction. If the page is' subsequently written on, another page fault will occur, allowing the operating system to set the M bit and change the page's mode to READ/WRITE The R and M bits can be used to build a simple paging algorithm as follows. When a process is started up, both page bits for all its pages are set to 0 by the operating system. Periodically (e,g , on each clock interrupt], the R bit is cleared, to distinguish pages that have not been referenced recently from those that have been. When a page fault occurs, the operating system inspects all the pages and divides them into four categories based on the current values of their R and M bits-. Class 0; not referenced, not modified. Class I; not referenced, modified. Class 2: referenced, not modified Class 3: referenced, modified. Although class I pages seem, at first glance, impossible, they occur when a class 3 page has iu R bit cleared by a clock interrupt. Clock interrupts do not clear the M bit because this information i-; needed to know whether the page has to be rewritten to disk or not. Clearing R but not M leads to a class I page.

SEC. 4 4 PAGE REPLACEMENT ALGORITHMS 217 The NRU (Not Recently Used) algorithm removes a page at random from the lowest numbered nonempty class Implicit in this algorithm is that it is better to remove a modified page that has not been referenced in at least one clock tick (typically 2() msec] than a clean page that is in heavy use. The mam attraction of NRU is that it is easy to understand, moderately efficient to implement, and gives a performance that, while certainly not optimal, may be adequate 4.4.3 The First-ln, First-Out (FIFO) Page Replacement Algorithm Another low-overhead paging algorithm is the FIFO (First-ln, First-Out) jlgonLhm To illustrate how this works, consider a supermarket that has enough shelve*, io display exactly k different products One day, some company intro- introduces a new convenience food—instant, freeze-dried, organic yogurt that can be reconstituted in a microwave oven. It is an immediate success, so our finite

supermarket has to get nd of one old product m order to stock it. One possibility is to find Ihe product that the supermarket has been stocking the longest (i.e., something it began selling 120 years ago) and get rid of it on the grounds that no one is interested any more. In effect, the supermarket maintains a linked list of all the products it currently sells in the order they were introduced. The new one goes on the back of the list; the one at the front of the list is dropped. As a page replacement algorithm, the same idea is applicable The operating system maintains a list of all pages currently in memory, with the page at the head of the list the oldest one and the page at the tail the most recent arrival. On a page fault, the page at the head is removed and the new page added to the tail of the list. When applied to stores, FIFO might remove mustache wax, but it might also remove flour, salt, or butter. When applied to computers the same problem arises. For (his reason, FIFO in its pure form is rarely used 4.4.4 The Second Chance Page Replacement Algorithm A simple modification to FIFO that avoids the problem of throwing out a heavily used page is to inspect the R bit of the oldest page If it is 0, the page is both old and unused, so it is replaced immediately. If the R bit is 1, the bit is cleared, the page is put onto the end of the list of pages, and its load time is updated as though it hadjust arrived in memory. Then the search continues The operation of this algorithm, called second chance, is shown in Fig. 4-|6. En Fig. 4-16(a) we see pages A through H kepi on a linked list and sorted by the time they arrived in memory Suppose that a page fault occurs at time 20. The oldest page is A, which arrived ai time 0, when the process staned. If A has the R bit cleared, it is evicted from memory, either by being written to the disk (if it is dirty), or just abandoned (if H is clean). On the other hand, if the R bit is set, A is put onto the end of the

218 MEMORY MANAGEMENT Page loaded first 3 7 8 12 14 15 5 18 20 _ pew|y |oa(le(j page Figure 4-16. Operanori ol second chance (a) Pages sorted in FIFO order (b) Page list if a page fault occurs ai tune 20 and A has ils R bit set. The number* above I he pages are (heir loading times list iind its "load time" is reset to the current time B0). The R bit is also cleared. The search for a suitable page continues with B. What second chance is doing is looking for an old page [hat has not been referenced in ihe previous clock interval. If all the pages have been referenced, .second chance degenerates into pure FIFO. Specifically, imagine that all the pages in Fig. 4-16(a) have their R bits ,set. One by one, the operating system moves the pages to the end of the list,

clearing the R bit each ume it appends a page to the end of the list. Eventually, it comes back lo page A, which now has its. Kbit cleared. At this point A ,s evicted. Thus the algonthm always terminates. 4.4.5 The Clock Page Replacement Algorithm Although second chance is a reasonable algorithm, it is unnecessarily ineifl- iHent because it is constantly moving page-, around on its list. A heller approach is to keep all the page frames, on a circular list in the form of a clock, as shown in Fig. 4-17 A hand points to the oldest page. When a page fauli occurs, the page being pointed to by the hand is inspected. If its R bit is" 0, Ihe page is evicted, the new page is inserted into the clock in us place, and the hand is advanced one posilion If R is I, it is cleared and ihe hand is advanced to ihe next page. This process i* repeated until a page is found with R=0. Not surprisingly, this algorithm is called clock. It differs irom second chance only in the implementation. 4.4.6 The Least Recently Used (LRU) Page Replacement Algorithm A good approximation to the optimal algorithm ^s based on the observation that pages lhat have been heavily used in the lasi few instructions will probably he heavily used again in the next few. Conversely, pages that have not been used' for ages will probably remain unused for a long time. This idea suggests a realizable

SEC 4 4 PAGF-REPLACEMENT ALGORITHMS 219 E 0 E H HI - ' ' When a page fa h h pg ^ me page the hand is Qy^ r—I pointing to is ,nspeded S 0 The action taken depends U-^ on ihe R bit a ihe R bit R = 0 Evict Ihe page r—j r = 1 Cleat R and advance 0 Figure 4-17. The dock page repl.it.emem algnnihrn algorithm: when a page fault occurs, throw out the page (hat has been unused lor the longest time This strategy k called LRU (Least Recently Used) paging Although I.RU is theoretically realizable, it k not cheap. To fully implement LRU. it is necessary to maintain a linked list of all pages in memory, with the most recently used page at [he, frotil and the least recently used page at the rear The difficulty is chat the hst must be updated on every memory reference Find- Finding a page in the list, deleting it, and then moving it to the front is a very time con- consuming operation, even m hardware (assuming that such hardware could be built) However, there are other ways to implement LRU with special hardware Let us consider the simplest way first. This method requires equipping the hardware with a 64-bit counter, C, that is automatically incremented after each instruction. Furthermore, each page table entry must also have a field large enough to contain the counter. After each memory reference, the current value of C is stored in the page table entry for the page

just referenced. When a page faull occurs, the operating system examines a|] the counters in the page table to find the lowest one. That page is the least recently used. Now let us look at a second hardware LRU algorithm. For a machine with n page frames, the LRU hardware can maintain a matrix of " x « bits, initially all zero. Whenever page frame k is referenced, the hardware first sets all the bits of row it to 1, then sets all the bits of column k to 0. At any instant, the row whose binary value is lowest is the least recently used, the raw whose value is next lowest is next least recently used, and ,so forth. The workings of ihis algorithm are given in Fig. 4-18 for four page frames and page references in the order 0 12 3 2 10 3 2 3 After page 0 is reterenced, we have the situation of Fig, 4- I8{a). After page I is reference, we have the situation of Fig- 4- |8{b), and so forth.

MEMORY MANAGEMENT 0 n 0 Page 1 2 , 0 0 0 , o 0 0 1 0 0 0 a 0 1 0 0 Page 1 2 0 0 0 0 1 ! 0 0 3 1 1 0 0 0 0 1 1 0 Page 0 I 0 0 1 0 0 0 o 3 1 1 1 0 0 0 1 1 1 Pa 0 0 1 t ge 2 0 0 0 I 3 0 I 0 0 0 0 0 ' 1 1 Pago 1 a 3 0 t 1 1) 0 0 0 0 | 1 Ci | 0 0 , 1 1 0 0 0 0 0 1 D 0 0 1 1 0 0 0 0 D 1 0 0 0 1 1 0 0 1 1 1 0 0 0 0 1 I 0 0 1 1 1 0 1 0 0 0 0 0 0 1 1 1] 0 0 | 0 1 j 0 1 | 0 0 0 I 0 0 0 1 1 10 0 I 0 1 I 0 1 1 t 0 0 0 0 4.4.7 Simulating LRU in Software Although both of the previous LRU algorithm ■■-, if any, machines havr 'k:- k—J—— — -u" syst hardw possibi software counter associatei able in principle, h rtltllUUgJ] 1XJII1 U| UIC |7ltVlUUN L.KU rtlgUlllllllS |UC Icilll/_<IU1C III JJllimpiC, :w, if any, machines have this hardware, so they are of little use to the operating astern designer who is making a system for a machine ihai does n,it have this ardware. Instead, a solution that can be implemenied in software is needed. One assibiliiy is called the NFU (Not Frequently Used) algorithm it requires a tfiwrnrp rmintPr nctiV^inrpri wllh pqrVi naup lnifi-nlfv /prA Ai pni h t L-n V inrpr The main problem with NFU is that it in a multipass compiler, pages that were heavni a high count well into later passes. In fact, if [execution time of all the passes, the pages containing nay always have lower counts than the pass I page lple ii ha\e the longest :ode for subsequent Consequently, the man uic rigmuiu&c on Figure 4-19 illustrates how the modified algorithm, kn Suppose that after the first clock tick the R bits for pages 0 /n as aging, works.

SEC. 4 4 PAGE REPLACEMRNT ALGORITHMS 221 0, 1. 0. I, and |, respectively (page 0 is I, page 1 is 0, page 2 is I, etc.). In other words, between tick 0 and rick 1, pages 0, 2, 4, and 5 were referenced, setting their ft

bits to I, while the other ones remain 0. After the six corresponding counters have been shifted and the R bii inserted ai the left, they have the values shown in Hg. 4-l9(a) The four remaining columns show the six counters after the next four clock tick*, A bus for i1 R blister pages 0-5, pages 0-5, clock nek 3 , clock lick 4 .|0|o|o|i|n|: [srrpT°T°T°1 latto LRU ,n software Shown are six k ticks are represented b> (a) 10 (e) When a page fault occurs, the page whot,e counter is Che lowest is removed. It is clear that a page that has not been referenced tor, say. four clock ticks will have four leading zeros in its counter and thus will have a lower value than a counter that has. not been referenced for three clock ticks. This algorithm differs from LRU in two ways. Consider pages 3 and 5 in Fig.4-I9(e), Neither has been referenced for two clock ticks; both were refer- referenced in the tick prior to that. According to LRU, if a page must be replaced, we should choose one of these two. The trouble is, we do not know which of these two was referenced last in the interval between tick 1 and tick 2. By recording only one bit per time interval, we have lost ihe ability to distinguish references early m the clock interval from those occurring later. All we can do is remove page 3, because page 5 was also referenced two ticks earlier and page 3 was not. The second difference between LRU and aging is that in aging the counters have a finite number of biis, 8 bits in this example. Suppose that two pages each have a counter value of 0. All we can do is pick one of ihem at random, [n

 icility, it may well be that one of the pages was last referenced 9 treks, ago ami the other was last referenced 1000 neks ago. We have no way of seeing that In practice, however, 8 bits is generally enough it a clock tick is around 20 msec II a page has not been leferenced in 160 msec, it probably is not that important 4.4.8 The Working Set Page Replacement Algorithm In the purest form of paging, processes are started up with none or their pages in memory As soon as the CPU tries to fetch the first instruction, it gets a page fault, causing the operating system to hnng Hi the pa i contaimn the fir.st instruc- instruction Other page faults for global variables and the stdiA usuillv follow quickly After a while, the process has most of the pages it needs and settles down to run with relatively few page faults. This strategy is called demand paging because pages are loaded only on demand, not in advance. Or course, it is easy enough to write a test program that systematically reads all the pages in a large address space, causing so many page faults that there is not enough memory to hold them all Fortunately,

most processes do not work this way. They exhibit a locality of reference, meaning that during any pha^e of exe- execution, the process references only a relanvely small fraction or its pages. Each pass of a multipass compiler, for example, references only a fraction oi' all the pages, and a different fraction at that. The set of pages lhat a process is currently Using is called its working set (Denning, 1968a; Denning, 1980). If the entire working set is in memory, the process will run without causing many faults until it moves into another execution phase (e.g., the next pass of the compiler) If the available memory ;s too small lo hold the entire working sec, the process will cause many page faults and run slowly since executing an instruction takes a tew nanoseconds and reading in a page from the disk typically takes 10 milliseconds At a rate of one or iwo instructions per 10 milliseconds, it will take ages to finish. A program causing page faults every few instructions is said to be thrashing (Denning. 1968b). In a multiprogramming system, processes are frequently moved 10 disk (i.e., all their pages are removed from memory) to Jet other processes have a turn at the CPU. The question arise-, of what to do when a process is brought back in again Technically, nothing need be done. The process will just cause page faults until its working set has been loaded. The problem is thai having 20, 100, or even liKX) page faults every time a process is loaded is slow, and it also wastes considerable CPU time, since it takes the operating system a few milliseconds of CPU time to process a page fault. Therefore, many paging systems try to keep track of each process' working set and make sure that it is in memory before letting the process run This approach is called the working set model (Denning, 1970). It is designed to greatly reduce the page fault rate. Loading the pages before letting processes run is also called prepaging. Note that the working set changes over time.

SEC 4 4 PAGE REPLACEMENT ALGORITHMS 223 It has been long known ihiit most programs du not reference their address pages A memory reference may fetch an instruction, it may fetch data, or it may store data. At any instant of time, i, there exists a set consisting ol all the pages used by the k most recent memory references'. This set, w(k, /). is the working sei Because the k = I most recent references must have used all the pages used by the A > 1 most recent references, and possibly others, w(k\ I) is a moiiotovu- eally nondcereasing function of A. The limit ol n (A, t) as k becomes large is tmite because a program cannut reference more pages th.m its address space contains, and tew programs' will use every single pajr£ Figure 4-20 depicts the size of the

working sei as a function of A king s The fact that most programs randomly access1 ,i s that ihis ,sel changes slowly m lime explains the initial then the slow ri.se for large k. For example, a progra occupying two pages using data on four pages, may refer 1000 instructions, bui the mosi recent reference to some other page may be a mil- million instructions earlier, dimiig the initialization phase. Due to this asymptotic all nui ipici n that i >ren<- e nber se ol s exi alls ill" pages. the curve fining a ix pages e . but and IO<)p very beha\ . the c To put it differently, there exists a wide range o(k values lor which the working sei is, unchanged. Because the working stt varies slowly with time, it is possible to make a reasonable guess as to which pages will be needed when the program is restarted on the basis of its working set when it was last stopped Prepaging con- consists of loading these pages before the process is allowed to run again To implement the working sel model, it is necessary for the operating system td keep track of which pages are in the working sel Having this information also immediately leads to a possible page replacement algorithm: when ,i page fault occurs, find a page not in the working set and evict it. To implement such an algorithm, we need a precise way of determining which pages are in the working set and which are not at any given moment in time

224 MEMORY MANAGEMENT CHAP. 4 As we mentioned above, the working set is the set of pages used m the * most recent memory references (some authors use the k most recent page references, but the choice is arbitrary) To implemem any working set algorithm, some value of k must be chosen in advance. Once some value has been selected, after every memory reference, the se(of pages used by the previous k memory references is uniquely determined. Of course, having an operational definition oi the working set does not mean that there is an efficient way to monitor it in real time, during program execution. One could imagine a shift register of lenglh k, with every memory reference shift- shifting the register left one position and inserting the most recently referenced page number on the right. The set of all it page numbers in the shift register would be Ihe working set. In theory, at a page fault, the contents oi the shift register could be read out and sorted. Duplicate pages could then be removed. The result would be ihe working set. However, maintaining the shift register and processing it at a page fault would both be prohibitively expensive, so this technique is never used. Instead, various approximations are used One commonly used approximation is to drop the idea

of counting back k memory references and use execution time instead For example, instead of defining the working set as those pages used dur- during the previous 10 million memory references, we can define it as the set of pages used during the past 100 msec of execution time. In practice, such a defini- definition is just as good and much easier to use Note that for each process, only its own execution time counts. Thus if a process starts running at time T anci has had 40 msec of CPU time at real time T + 100 msec, for working set purposes, its time is 40 msec. The amount of CPU time a process has actually Used has since it started is often called its current virtual time with this approximation, the working set of a process is the set of pages it has referenced during the past T seconds of virtual time. Now let us look at a page replacement algorithm based on the working set. The basic idea is to find a page that is not in the woikmg set and evjct it. In Ftg. 4-2i we see a portion of a page table for some machine Because only pages that are in memory are considered as candidates lor eviction, pages that are absent from memory are ignored by this algorithm Each entry contains (at least) two items of information: the approximate time the page was last used and the R (Referenced) bit. The empty white rectangle symbolizes the other fields not needed for thts algorithm, such as the page frame number, the protection bits, and the M (Modified) bit. The algorithm works as follows The hardware is assumed to set the R and M bits, as we have discussed before. Similarly, a periodic clock interrupt is assumed lo cause software to run that clears the Referenced bit on every clock tick On every page fault, the page table is scanned to look tor a suitable page to evict. As each entry is processed, the R bit is examined If it is 1, the current virtual time is written into the Time of last use field in the page table, indicating that the page was in use at the time the fault occurred Since the page has been referenced

PAGE REPLACEMENT ALGORITHMS i((R==O Figure 4-21, The ^ork-ing %ei algorithm during the current clock tick, it is clearly in the working set and is not a candidate If R is 0, the page has not been referenced during the current clock nek and may be a candidate for removal. To see whether or not it should he removed, its age, that is, the current virtual time minus its Time of last use is computed and compared to T. If the age is greater than t, the page is no longer in the working set It is reclaimed and the new page loaded here The scan continues updating the remaining entries, however However, if R is 0 but the age is less than or equal lo X, ihe page is still in the working set. The page is

temporarily spared, but the page with the greatest age (smallest value of Time of las! use) is noted If the entire tahle is scanned without finding a candidate to evict, that means that alt pages are m the working set. In that case, if one or more pages with R = i) were found, the one with the greatest age is evicted In the worst case, al! pages have been relerenced during the current clock tick (and ihus a|l have R = 1). so one is chosen at random for remo- removal, preferably a clean page, if one exists. 4.4.9 The WSCIock Page Replacement Algorithm The basic working set algorithm is cumbersome since the entire page table has to be scanned at each page fault until a suitable candidate is located. An improved algorithm, that is based on the clock algorithm but also uses tlie woik- ing set information is called WSCIock (Carr and Hennessey, 1981). Due to its simplicity of implementation and good performance, it is widely used in practice.

226 MEMORY MANAGEMFNT ("HAP 4 The dat.i structuic needed is a uicul.ii list of [)Jg^ frames, as in the clock algorithm, and ,ts shi,\vn in Hg 4-22(a) luniall>. [liis list is empty. When the hist page is loaded, it is added to the list As more pages are added, they go into the list 10 form lt ring Rath entr> contains the lime of last use field from the basic working set algorithm, ds well a* (lie ft bit (shown) and the M bit 'not shown I |~2204] Currenl virtual time [——| f figure 4-22, 0|xr.iinni t.l Hie WSCL.a .ilsmnhm (.,i and A-1 aise Jn cxi-nrie ,.l whji h.,ppc1K *!,, „ « ^ | K i Jn(hA1 glu iln ,x,lll|)lt, ltf K ="|1_ As With the clock algonthm. at each page fault the page pomied K, b> the hand is examined first If the R bit is set to I, the p.ige has been used dunng \hc

S[;C. 4 4 I^C.E Rtl'LAC'fcMENl ALGORITHMS 227 current tick so it is no! an ideal candidate to remove The ft bit is then sei to 0, the hand advanced to the next page, and the algorithm repeated for (hat page. The state after this sequence of events is shown in Fig. 4-22(bl Now consider what happens if the page pointed to has R = 0. as shown hi Fie 4-22(O If the age is greater than T and the page is clean, i(is not in the working set and a valid copy exists on the disk The page frame is simply claimed and ihc new page put there. ;is shown in Fie 4-22(d) On the other hand, if the page is dirty, it cannot he claimed immediately since no valid copy is present on disk. To avoid a process switch, the write lo disk is scheduled, but (he hand is advanced and the algorithm continue*, with the next page After all, there might be an old. clean page further down the iine that can be used immediately In principle, ali pages might

be scheduled for disk I/O on one cycle around the clock. To reduce disk traffic, a limii might be set. allowing a niatiiiiiiin of /; pages to be written back Once this limit has been reached, no new wuies are scheduled. What happens if the hand comes all jIil* way around fo its starting point'1 There are two cases to distinguish. 1 At least one write has been scheduled 2 No writes have been scheduled. In the former case, the hand just keeps moving, looking lor a cleaii page Since its page will be marked as clean. The first clean page encounteied is evicted. This page is not necessarily the first write scheduled because the disk diner may reorder writes in order to optimize disk performance In the latter case, all page", arc m the working set, otherwise at least one wnte would have been scheduled. Lacking additional information, the simplest thing to do is claim any clean page and use it. The location of a clean page could be kept track of during the sweep. If no clean pages exist, then the current page is chosen and written back to disk. 4.4.10 Summary of Page Replacement Algorithms We have now looked at a variety of page replacement algorithms. In this sec- lion we will briefly summarize them. The list of algorithms discussed is given in Fig. 4-23. The optimal algorithm replaces the page referenced last among the current pages. Unfortunately, there is no way to determine which page will be last, so in practice this algorithm cannot be used. Jt is useful as a benchmark againsi which other algorithms can be measured, however The NRU algorithm divides pages into four classes depending on the state at Che R and M bits. A random page from the lowest numbered class is chosen This algorithm is easy to implement, hut it is very crude. Better ones exist.

MEMORY MANAGEMENT CHA Algorithm^ I Comment ' Optimal | Not implementable^butuseftil as^a benchmark^ NRU (Not Recently Used) T Very crude ' FIFO (Firsl-ln, First-Out) Might throw out important pages Second chance _ Big improvement over FIFO ^lock j Realistic _l | LRU (Least Recently Used) \ Excellent, but difficult to implement exactly I NFU (Not Frequently Used) j Fairiy crude approximation to_LRU _ ' Aging __ ___ i Efficient algorithm that approximates LRU well | Working set ~ I' 5omewhat expensive to implement I WSClock ~f QOT(I efficient algorithm Figure 4-13. Page replaC-inem algorithms Uiscusied ui Ihe lex. FIFO keeps track of the order pages were loaded into memory by keeping (hem \n a linked Itst. Removing the oldest page then becomes trivial, but that page might still be in use, so FIFO is a bad choice. Second chance is a modification to FIFO (ha!

checks if ,i page is in use before removing it. If it is, (he page is spared. This modification greatly improi.es the performance. Clock is simply a different implementation of second chance It has the same performance properties, but takes a tittle less time to execute the algorithm LRU is an excellent algorithm, but it cannot be implemented without special hardware If this hardware is not available, it cannot be used NFU is <i crude attempt to approximate LRU. (t is not very good. However, aging is a much better approximation to LRU and can be implemented efficiently It is a good choice. The last two algorithms use the working set. The working set algorithm is reasonable performance, but it is somewhat expensive to implement WSClock is a variant that not only gives good performance but i« also efficient to implement All in all, the two best algorithms are aging and WSClock. They .ire based on LRU and the working set, respectively. Both give good paging performance and can be implemented efficiently. A few other algorithms exist, but these two .tre probably the most important m practice. 4.5 MODELING PAGE REPLACEMENT ALGORITHMS Over (he yean,, some work has been done on modeling page replace algorithm!; from a theoretical perspective. In this section we will discuss sor these ideas, just to see how the modeling process works

ShC 4 ^ MODELING PAGF- RfcPLAC hMEN f ALGORITHMS 229 4.5.1 Belady's Anomaly lmuiti\ely, il might seem that Lhe more page dames the memory has, the fewer page faults d program wrtl get Surprisingly enough, this is not always the case tidudy et al. A^69) disco\ered a counterexample, in which FIFO caused more page iaults with iour page frames lhan with three Thts strange situation has become known as Belady's anomaly It is illustrated in Fig 4-24 for a program with five i-irtual pages, numbered from 0 to 4 The pages are referenced in lhe (I 1 2 T 0 I 4 0 I 2 14 how with three page fram n Fig 4-24(a) w, aused In Fig 4 i total of nine page faults surpage frames 0 1 0 2 1 0 3 2 1 0 3 2 1 0 3 4 1 0 4 1 0 4 1 0 p 4 1 3 2 4 3 2 4 Youngest page |~~ r Oldest page j t 0 1 0 2 1 0 3 2 1 0 3 2 1 0 3 2 1 0 4 3 2 '. 0 4 3 2 1 0 4 3 2 1 0 4 3 2 1 0 4 3 2 1 Figure 4-24. BeUiv' with loui page frames 4.5.2 Stack Algorithms Many rescurehers in computer science were dumbfounded by Belady's 3no- majy and began investigating it This work ltd to the development of a whule theory ot paging algorithms and their properties While most of this work is beyond the scope of this book, we will gi\e a bnet introduction below. For more details see (Maekawa et at.. 1987) All of tms work begins with the observation that every process generates a sequence ot memory references as

i! runs. Each memory reference corresponds to

230 MEMORY MAN WjRMENT <"H\P 4 a specific vinuai page Tims conceptually, a process' memory access can he characterized bv an (ordered) list ot page numbers This list is called the refer- reference siring, and plays a cental role in the theory For simplicity, in the rcst of this section we will consider only ihe case ot a machine with one process, so each machine has a single, deterministic reference siring (wilh multiple processes, we the multiprogramming). A paging system can be characterized by three items: 1 The reference string of the executing pmeess 2 1 he page replacement algoiilhin 3. The number of pago frames available in memory, wi. M is divided into two pints. The top part, with m entries, contains all the pages that aa currently in memory The bottom part, with n • m pages, contains all the pages that have been referenced once but have been paged out and are not oittmtiy in memory. Initially, M is the empty set. since no pages have been referenced and no pages are in memory As execution begins, the process begins emitting (he pages in (he reference stnne one at a time. As each one comes out, the interprelei checks to see if the pagt is m memory (i.e.. in the top part ot M) If i(is not, a page fault occurs. If there is an empty slot in memory (i.e., the top part ot M contains tewei than m eniuesi the pat,e is loaded and entered in the top part of M. This situation arises only at the start of execution.]f memory is full (i.e.. the top part of M contains m entries), the page replacement algorithm is invoked to remote a page from iiii-rn- ory. In (he model, what happens is thai one page is moved from the top pail of M to the bottom part, and the needed page entered into the top part. In addition, die top part and the bottom pan may be separately rearranged To make the operation ot the inierpretei clearer, let us look at a concrete example using LRU page replacement. The virtual address space has eight page.s and the physical memory has four page frames At the top of Fig 4-25 we have a reference string consisting of (he 24 pages: 021354637473 355311172 34 1 Under the reference string, we have 25 columns of 8 items each The first column, which is empty, reflects (he state of M before execution begins Each successive column shows M afier one page has been emitted by the reference and processed by the paging algorithm. The heavy outline denotes the top of M, that is. the first four slots, which correspond io page frames in memory. Pages inside the heavy box are in memory, and pages below it have been paged oui to disk.

ShC. 4 5 MODELING PAGF REPLACEMENT ALGORITHMS 231 Reference stnna 02135463747335531 117134- ft °l '\ 4 1 Ji T 1 T 4 -j- o" 7 T -p Ji ", u T 0 0 6 v ■5" — e 6 7 6 — b V 1 b T Fijsure4-25. The The first page in the reterence string is 0. so it is entered in the top ot mem- memory, as shown m the second column. The second page is 2. so it is entered at the top of the third column. This action causes 0 to niu\e down In this example, a newly loaded page is always entered at the top. and everything else moved down, as needed Each of the first seven pages in the reference string causes a page fault. The first four can be handled without removing a page, but starting with the reference to page 5, loading a new page requires removing an old page The second reference to page 3 does not cause a page iault, because \ is already m memory. Nevertheless, the interpreter removes it horn wheie it was and puts it on the top, as shown. The process continues foi a while, until page 5 is referenced. This page is moved from bottom part ot M to the top part (i e , it is loaded mio memory from disk). Whenever a page is referenced that is not within the heavy box. a page fault occuis. as indicated by the P\ below the matrix. Let us now briefly summarize some oi the properties of this model I- irst. when a page is referenced, it is always mo\ed to the top entry in M Second, it the page referenced was already m M, all pages abo\e it move down one position. A transition from within the box to outside of it corresponds lo a page being evicted from memory. Third, pages that wen.- below the referenced page are not moved. In this mannei. the contents of M exactly represent the contents of the LRU algorithm. Although this example uses LRU. the model works equally well with other algorithms. In particular, there is one ^lass ot algorithms thai is especially interesiing. algorithms that have the property where m varies over the page frames aiid r is an index into ihe reterence string. What this says is (hat the set of pages included m the top part of M for a memory

232 MEMORY MANAtiEMfcN ry ref CHAP wnh m page Irames at , ory with m + 1 page frames. In other words, if w page frame and rc-execute the process, at every po. pages thai were present in (he first run Juded in Af tor amen i- st' memory size by one ._ r_... . ig the execution, (ill the ilso presi-nt in the second run. along with one additional page From examination ot Fig. 4-25 and a little thought about how U works, il should be clear thai LRU has this property. Some other algorithms (e g . optimal page replacement) also have it. but FIFO does not Algorithms (hat have this pro- property are called stack algorithms.

These algorithms do not suffer from Belady's anomaly and are thus much loved by virtual memory theorists 4.5.3 The Distance String For stack algorithms, it is often convenient to represent the reference string in a more abstract way (han the actual page numbers A page reference will he henceforth denoted by the distance from ihe top of the .stack where the referenced page was located For example, the reference to page 1 in the last column of Fig. 4-25 is a reference to a page at a distance 3 from the top ot the stack (because page I was in third place before the reference) Pages that have not yet been referenced and thus are not yef on the stack (i.e . not yet in M) are said to be at a distance «. The distance string for Fig. 4-25 it, given at the bottom of the figure. Note thai the distance string depends not only on the reference string, but ako on the paging algoriihm. With the same original reference string, a different pag- paging algorithm would make different choices about which pages to evict As a result, a differenl sequence of stacks arises The statistical properties of the distance string have a big impact on the per- performance of the algorithm. In Fig. 4-26(a) we see the probability density function for the entries in a (ficticious) distance string, d Most of the entries in the string are between I and k. With a niemory of A: page frames, tew page faults occur. ntiast, in Fig- 4-26(b). the refer a large number of page faults is virtual pages. Having a progra y to give the program as many page fram like this is just bad luck.

SEC. 4.5 MODELING PAGE REPLACEMENT ALGORITHMS 4.5.4 Predicting Page Fault Rates One ol the nice properties of the distance string is that it c predict the number of page faults (hat will occur with me We will demonstrate how this computation can be used to f different size One o t i pp g predict the number of page faults (hat will occur with memories of different sizes. We will demonstrate how this computation can be made based on the example of Fig. 4-25. The goal is to make one past, over the distance siring and, from the information collected, to be able to predict how many page faults the process would have in memories with 1, 2, 3, ... n page frames, where n is the number of virtual pages in the process' address space. The algonthm starts by scanning the distance siring, page by page. It keeps track of the number of times I occurs, the number ot limes 2 occurs, and so on. Let C, be the number of occurrences of (. For the distance string of Fig. 4-25, the C vector is illustrated in Fig 4-27(a). In ihis example, it happens four limes that the page referenced is already on lop of the stack. Three times the reference is to (he

next-to-ihe-top page, and so forth. Let (\ be the number of times » occurs in the distance string. C,s C3 = C4 = C5 = CB = c,= 4 2 i * 2 2 1 F3= 17 F,= 16 F4* 12 F6= 10 F7= B F.= 6 ^ c -* C + c otp age according to the iornmla The value of Fm is the number of page (aults that wi|l occur with [he given dis- distance string and m page frames. For the distance string of Fig 4-25. Fig 4-27(b| gives (he F vector. For example, F, is 20. meaning that with a memory holding only I page frame, out of the 24 references m the string, all gei page faults excepi the four that are the same as the previous page reference.

234 MFMORY MANAObMbNI CHAP 4 To '■ee why this foimula works, gn back to the heavy box in big 4-_. .et m be the number of page frames in the top pan n| M A page iault occurs an> lime in dement of the distance siring is m + 1 or more The summation in the formula .ibove adds up the number of times such elements ocutr This model can be used In make other predictions as well (Maekawa et al . 1987) 4.6 DESIGN ISSUES FOR PAGING SYSTEMS In the preuous sections we have explained how paging works and have given ;i lew (il the basic page replacement algorithms and shown how to model them. But knowing the bare mechanics is nut enough to design a system, you have to know a lot more to make it work well- |l is like the diilerence between knowing how to move the rook, knight, bishop, and othei piecvs in chess1, and being a good player In the following secnons. we will look at other issues that operating sys- system designers itiust consider carefully in ordoi to get good performance from ,i paging system. 4.6.1 Lotal versus Global Allocation Policies in the preceding sections we have discussed sevi.-i.il algorithms tor choosing a page to replace when a lault occurs. A ma|of issue associated with this choice (which we have carefully swept under the rug until now) is how memory should be allocated among the competing runnable processes- Take a look at Fig. 4-28(a). In this figure, three processes. A. B, and C, make up the sel of runnable processes. Suppose A gets a page fault Should the page replacement algorithm try to find the least recently used page considering only the Si\ pages currently allocated to A, or should it consider all the pages in meinory' If it looks only at A\ pages, the page with the lowest age value is A5. so i\c get the situation ol Fig 4-28(b) On the other hand, it the page with the lowest aye value is removed without regard to whose page it is. page B3 will be chosen and we will get the situation of Fig. 4-28(c) The algorithm ol Fig. 4-2S(b, is said to be a local page replacement algorithm, whereas that of Fig 4-2S(c) is said to be ;i global algorithm Local algorithms effectively correspond

to allocating every process a fixed fraction ot the memory Global algorithms dynamically allocate page frames among the runnable processes Thus the number of page framts tssi.ncd to each process vanes in time In general, global algorithms work better, especially when the working set size ean vary over the lifetime of a process. It a local algorithm is used and the working set grows, thrashing will result, even if there are plenty of Iree page frames If the working set shrinks, local algorithms waste memory. If a global algorithm is used, the system must continually decide; how many page frames lo assign to each process. One way is to monitor the working set size as indicated by

DliSIGN ISSUES i-OR PAGING SYSThMS the aging bits, but this approach does not necess working set may change si^e in microseconds, whe measure spread over a numbei ot clock neks rily prevent thrashing. The as the aging bits aie a ciude processes. One way is to periodically determine the number of running processes and allocate each process an equal share Thus with 12.416 availahle fi e . uoii- operating system) page frames and 10 processes, each process gets 1241 frames The remaining 6 go into a pool to he used when page faults, occur. Although this method seems fair, it makes little sense to give equal share-, of the memory to a 10-KB process and a 300-KB process. Instead, pages can be allocated in proportion to each process" total si?e, with a 300-KB process getting 30 times the allotment of a 10-KB process. Il is probably wise to give each proc- process some minimum number, so it can run. no matter how small it is On some machines, for example, a single two-operand instruction may need a.s many as six pages because the instruction itself, the source operand, and ihe destination operand may all straddle page boundaries. With an allocation of only live pages, programs containing such instructions cannot execute at all It a global algorithm is1 used, it may he possible 10 stari each process up with some number of pages proportional to the- process' si/e, but the allocation has to be updated dynamically as the processes run. One way to manage the allocation is to use the PFF (Page Fault Frequency) algorithm It iells when to increase or decrease a process' page allocation bui says nothing about which page to replace on a fault. It just controls the sue of the allocation set.

 Ford large class of page replacement algorithms,, including LRU. it is known that the fault rate decreases as more pages are assigned, as we diseased above This is the

assumption behind PFF. This properly is illustrated in Fig. 4-29. Figure 4-29. Page I ^signed Measuring the page fault rate is straighttorward: just count the number of faults per second, possibly taking a running mean over past seconds as well One easy way to do this is to add the present second's' value tu the currenf running mean and divide by two The dashed line marked A corresponds to a page fault raie (hat is unacceptably high, so the faulting process is given more page trames io reduce the fault rate. The dashed line marked B corresponds to a page fault raie so low that it can be concluded that the process lias too much memory. In this case page frames may be taken away from it. Thus, PFF tries to keep the paging rate for each process within acceptable bounds It is important to note that some page replacement algorithms can work with either a local replacement policy or a global one For example. FIFO can replace the oldest page in all ot memory (global algorithm) or the oldest page owned by the current process (local algorithm) Similarly- LRU or some approximation to it can replace the least recently page in all of memory (global algorithm) or the least recently used page owned by the current process (local algorithm). The choice of local versus global is independent of [he algorithm in some cases On the other hand, for other page replacement algorithms, only a local stra- strategy makes sense. In particular, the working set and WSClock algorithms refer to some specific process and must be applied in that context. There really is no working set for the machine as a whole and trying to use the union ot all the working -sets would lose the locality property and not work well. 4.6.2 Load Control Even with the best pa; of page frames to proces.su ever the combined workin ■eptacement algorithm and optimal global allocation t can happen that the system thrashes. In tact, when- ets of all processes exceed the capacity of memory.

SRC 4 6 DFSIGN ISSUES HOR PAGING SYSTEMS 237 thrashing can he expected One symptom of this situation is lhat the PFF algo- algorithm indicates that some processes need more memory but no processes need less needing it without hurting some other processes- The only real solution is to tem- temporarily get rid of some processes The way to reduce the number of processes competing for memory is to swap some or them to the disk and free up all the pages they are holding. For example, one process can be swapped to disk and its page frames divided up among other processes lhat are thrashing If (he thrashing stops, the system can run for a while this way. It it does not Mop, another process has to be swapped out, and so on, uniil the thrashing stops.

Thus even with paging, swapping is .still needed, only now swapping is used t<> reduce potential demand for memory, rather than to reclaim blocks of il for immediate use Swapping processes out to relieve (he load on memory is reminiscent of two- level scheduling, in which some processes are put on disk and a short-ierm scheduler is used to schedule the remaining processes. Clearly, the two ideas can be combined, with just enough processes swapped out to make the page-fault rate acceptable. Periodically, some processes are brought in from disk and other ones are swapped out there. However, another factor to consider is the degree of multiprogramming As we saw in Fig. 4-4, when the number of processes in main memory ts too low the CPU may be idle for substantial periods of time This consideration argues for considering not only process size and paging rate when deciding which process to swap out, but also its characteristics, such as whether il is CPU bound or I/O bound, and what characteristics the remaining processes have as well. 4.6.3 Page Size The page size is often a parameter that can be chosen by the operating system. Even if the hardware has been designed with, for example, 512-byie pages, the operating system can easily regard pages 0 and I. 2 and 3. 4 and 5, and so on, as 1-KB pages by always allocating two consecutive 512-byte page frames for them Determining the besl page size requires balancing several competing factors As a result, there is no overall optimum. To start with, there are two factors lhat argue for a small page size. A randomly chosen text, data, or slack segmeni will not fill an integral number of pages. On the average, half of the final page will be empty. The extra space in that page is wasted. This wastage is called internal fragmentation. With n segments in memory and a page size of p bytes, np/2 bytes will be wasted on internal fragmentation This reasoning argues for a small page .size. Another argument for a small page size become;, apparent if we think about a program consisting of eight sequential phases of 4 KB each. With a 32-KB page

 (HA1' 4 si/e, the program must be allocated M KB all ihe lime. With a 16-KB page sizt-. li nteds only 16 KB. With a page size ol 4 KB or smaller, it requires onl) 4 KB at any instant In general, a large page size will cause more unused progiani to be in memory than a small page size On the other hand, small pages mean that programs will need many pages, hence a large page table. A ^2-KB program needs only tour 8-KB pages, but 64 with niosl of ihe time being for the seek and rotational delay, so that tianstwnng J small page lakes almost a1- much time as transferring a large page li might

take h4 x | () msec to load 64 i I 2-byte pages, but only 4x12 msec to load foui 8-KB pages On some machines, the page table must be loaded into hardware rccisteis e\eij tune the CPU switches from one process t<> another On these- machines having a small page size means that the Lime requited to load the page repisteis gets lotigei as the page si/e gets smaller. Furthermore, the space occupied by the page table increases as the page size decreases This last point can be analyzed mathematically Let the- ;iveiage process size be ^ bytes and ihe pace size lie /> bytes Furthermore, assume that eacli n<ige entry requires e bytes. The approximate- number ot pages needed per pioiess is then .\/p, occupying \e/p bytes ot page table «paee The v.asred memory in ilie last page of the process due to internal fragmentation i^ p/2 Thus, the total overhead due to the page table and the iniemal fragmentation loss is t^iveri by the sum ot these two terms' overhead = \e/p + p/2 The first term (page table size) is large when the pyge si/.e is small The second term (internal fragmentation) is large whei) the page size is largt- The optimum mu.sl lie somewhere in between. By taking the first derivative with respect to/? and equating it to zero, we get the equation -se/p2 + 1/2 = 0 From this equation we can derive a fomjula that gives the optuyiuni page size (considering only memory wasted in fragmentation and page table size). The result is: p = \l\f For ,\ = 1MB and e = 8 bytes per page table entry, the optimum page size is 4 KB Commercially available computers have used page sizes ranging from 513 bytes to 64 KB. A typical value used to I KB, but nowaday,, 4 KB or 8 KB are more common. As memories get larger, the page size tends to get larger as well (bui not linearly). Quadrupling the RAM .size rarely even doubles the page size.

Sh(I 6 HI SlCr- ISSUE? FOR PAGING S\ STFMS 239 4.6.4 Separate Instruction and Data Spates Most computers have a single address space that holds both programs and data, as shown in Fig- 4-3(J(a). if this address space is laige enough, everything works fine. However, it is often loo small, forcing progiammers to stand on their heads io fiL everything into ihe address space Singte address Data 1 space — } Unused page Fij-ui 4-30. (j)One address One solution, pioneered on the A6-bii) PDP-I 1, is to have separate address spaces for instructions (program text) and data. These are tailed I-space and D-Space, respectively. Each address space runs trom 0 to some maximum, typically 2l(l - I or 232 - 1. In Fig. 4-30(b) we see both ot them The linker must know when separate I- and D-spaces are being used because when they are. the data Lire relocated to virtual address 0 instead of starting after the

program In a computer with ihis design, both addrcs.s spaces tan be paged, indepen- independently from one another Each one has its own page table, with its own mapping of virtual pages to physical page frames When the hardware wants to fetch an instruction, it knows that it must use I-space and the l-space page lable Simi- Similarly, references to data must go through the IXspau; page table. Other than this, distinction, having separate [- and D-space^ docs not introduce an^ special com- complications and it does double the available addiess space 4.6.5 Shared Pages Another design issue is sharing. In d large multiprogramming system, it is common for several users to be running the same program at the same time. It is clearly more efficieni to share the page.s, to avoid having two copies of the same page in memory at the same time. One problem is that not all pages are sharable. In particular, pages that are read-only, such as piogram iL-xt. can be shared, but data pages cannot. If separate I- and D-spaces are supported, ii is relatively straightforward to share programs by having two or more processes use the same page table for thetr

240 MEMORY MANAGEMENT CHAP 4 |-spaee but different page tables for their D-spaces Typically iti an implementa- implementation thai supports sharing in this way, page tables are data structures independent of thL- process table. Each process then has two pointers in its process table' one lo the l-sp;iec page table and one to the D-space page table, as shown in Fig. 4-31. When the scheduler chooses a process to run, it uses these pointers to locate the appropriate- page tables and sets up the MMU using them Even without separate I- and D-spaces, processes can share programs (or sometimes, libraries), but the mechanism is more complicated program sharing [la pjge (able code, a problem occurs with the When two or more processes share some code, a problem occurs wit shared pages. Suppose that processes A and B are both running the editor and sharing its pages. If the scheduler decides to remove A from memory, evicting all its pages and filling Ihe empty page frames with some other program -w. ill cause B 10 generate a large number of page faults to bring them back in again. . . arly, when A teirrnnate, pages are still in use so that their disk space ing all (he page tables to see if a page is shared .... agai .,, ■,, be able to discover that (he not be freed by accident Search illy too e it of

SEC 4 t> D1S1GN" ISSUES FOR PAGING SYS ["EMS 241 of page*. Thus no copying oi pages is done at fork lime. However, all the data pages\ue

mapped into both processes as RhAD ONLY * As long as both process just read then data, without modifying it. this situa- situation of iKe read-onb protection causes- a trap to the operating system. A copy is then made ot the page so that each process now has its own private copy. Both copies are now set to READ-WRITE so subsequent write* to either copy proceed without trapping. This strategy means that tho.se pages that are never written (including all the program pages! need not be copied Only the data pages that are actually written need be copied. This approach, called copy on write, improves performance by reducing copying 4.6.6 Cleaning Polity Paging works- best when there are plenty of tree page frames Chat can be clarmed as page faults occur If every page frame is full, and furthermore modi- modified, before a new page can be brought m, an old page must first be written to disk. To insure a plentiful supply ot free page frames, many paging systems have a background process, called the paging daemon, that sleeps most of the time but is awakened periodically to inspect the state of memory. If too few page frames are free, the paging daemon begins selecting pages to evict using the chosen page replacement algorithm If these pages have been modified since being loaded they are written lo disk In any even!, the previous contenis ot the page are remembered In the event one of the evicted pages is needed again before its irame has been overwritten, it can be reclaimed by removing it from the pool ol free page frames Keeping a supply ot page frames around yields better performance than using 3II of memory and then trying to find a frame at the moment it is needed At the very least, the paging daemon ensures that all the free frames are clean, so they need not be writ- written to disk in a big hurry when they are required One way to implement this cleaning policy is with a two-handed clock. The (ront hand is controlled by (he paging daemon When it points to a dirty page, that page it written back to disk and the front hand advanced. When it points to a clean page, it is jusi advanced The back hand is used for page replacement, as in (he standard clock algonthm. Only now. the probability of the back hand hitting a clean page is increased due to the work of the paging daemon 4.6.7 Virtual Memory Interface Up until now. Our whole discussion has assumed that virtui 'grammers. that is. all they see is a large virtual address i small(er) physical memory Wuh many systems, (hat red systems, programmers have some control over the

briefly look at a tew ot these. One reason for giving programmers control over (heir memor; map is to allow (wo or more processes to share the same memory. If programmers can name regions of their memoty, it may be possible for one process to gue another h f ■ ti ' n o tl at nroce s can also map it in With iwo (oi morel processes shilling the >,ame pages, high bandwidth sharing becomes possible—one process wrrtcs into the shared memorj and another one reads from Sharing ot pages, can also be used to nuplement a high-performance mes- message-passing system. Noinully, when messages are passed, the data are copied from one address space to Qnothei. at considerable cost. 11 processes can contiol then page map. ,1 message can be passed by having the sending process unmap the page(s) containing the message, and the receiving process mapping them in. Here only the page names have to be copied, instead ot all the data Yet anther advanced memory management technique is distributed shared memory (Feeley et a!.. 1995. Li, 1986; Li and Hudak. \9m: and Zekauskas et a[.. 1994). The idea here is to allow multiple processes over a network to shaie a set of pages, possibly, but not necessarily, as a single shared linear address spate. When a process references a page that is not currently mapped in. it gels a page fault The page fault handler, which may be in the kernel or in user space, then locates the machine holding the page and sends it a mewaee asking it to unmap the page and send it over the network. When the page arrives, it is mapped in and m more detail in Chap. 8.

4.7 IMPLEMENTATION ISSUES

Impleinenters of virtual memo'y systems have to make choices among the major theoreucal algonthrm such as second chance versus aging, local versus glo- global page allocation, and demand paging versus prepaging. Bui they also have to be aware of a number ot practical implementation issues as well In this section we will take a look at a few of the more common pioblems and some solutions.

4.7.1 Operating System Involvement with Paging

There are four times when the operating system lias work to do relating to paging: process creation time, process execution time, page fault time, and proc- process termination time. We will now briefly examine each ot these to sec what has lo be done. When a new process is created in a paging system, the operating system has to determine how large the program and data will be (initially) and create a page

[able toi it Space has to be allocated m memory tor the page table and it has to be initialized- The page table need not be redden! when the process is swapped ,,ut hut has to be m

memory when the process- is running In addition. space lid- to be allocated in the swap area on disk so that when a page is swapped out. it has somewhere li> go The swap area also has to be initialised with program text and data so that when tlie new process starts getting page faults, the pages can be brought in from disk Some systems p;ige the program text directly iron) the cic- utwble tile, thus sav'itii; disk space and initialisation time Finally, information about the page table and swap area on disk must bo recorded in the process table When a process is scheduled lor execution, the MMU has to be reset for the new pioeess and ihe 1 L.B flushed, to get rid ol traces (if the previously executing pf,)(.ess The new proeeSls" page table lias to be made current, usually by copying it or a pointer to it to some hardware rtgister(s) Optionally, some oi all or the process' pages tan be brought into memorv to ieduce the number of pape faults When a page fault occurs, the operating system has to read out hardware registers to determine which virtual address caused the lault. (toiti this wtciriria' (ion. it must compute which p.'ige is needed and locate thai page on disk It must then find an available page frame u> put the new page, evicting some t>kl page it need be Then it must read the needed page into the page frame Finally, it must back up the program counter to have it porrtt (o the 'aullmp instruction jnd let thai instruction execute again. When a process exits, the operating system must release Us pa^e table, if- pages, and the disk spact- that the pages occupy when they are on disf. It some ot the pages are shared with other processes, the pages in memoiy and on disk ..m only be released when the last process iNitig them has terminated. 4.7.2 Page Fault Handling We are finally hi a position Lo describe whai happens on a page lault in some detail. The sequence ot events is as follows 1. The hardware rraps to the kernel, saving the program counter on the stack- On most machines, some infomiation about the state oi the current instruction is saved m special CPU register 2. An assembly code routine is started to save the general registers and other volatile information, to keep the operating system from des- destroying it. This routine calls the dperaung system as a procedure 3- The operating system discovers that a pasic fault has occurred, and tries tc, discover which virtual page is needed Often one ol the hardware registers contains this information.]i not. the operating system must retrieve the program counter, tetch the instruction, and parse it in software to figure out what it was doing when the fault hit.

the fault is known, the s> stem checks to see ii this address is valid and ihe protection consistent with the access. If not. the process is sent a signal or killed |f the address is valid and no protection fault lias occurred, the system checks lo see it a page frame is free It no trames are free, the page replacement algorithm is run to select a victim 5. If the page frame selected is, dirty, Hie page is scheduled for transfer lo the disk, and a context switch takes place, suspending the laultmg process and letting another one run until the disk transfer has com- completed. In any event, the frame is marked as busy to prevent it from being used for another purpose. 6. As soon as the page frame is clean (cither immediately or after il is written to disk), the operating system looks up the disk address where the needed page is. and schedules a disk operation to bring it in. White the page is being loaded, the faulting process is ulill suspended and another user process is run. if one is available. 7. When the disk interrupt indicates that the page has arrived, the page tables are updated to reflect its position, and the frame is marked as being in normal state 8. The faulting instruction is backed iip to the stale it had when it began and the program counter is reset to point to that instruction. 9. The faulting process rs scheduled, and the operating system returns to the assembly language routine that called it. 10. This routine reloads the registers' and other .state information and re- returns to user space to continue execution, as if no fault had occurred 4.7.3 Instruction Backup When a program references a page that is not in memory, the instruction c ing the fault is stopped part way through and a trap to (he operating system occ After the operating system has letched the page needed, it must restart the inst tion causing the trap. This is easier said than d n causing the trap. This is easier said than done To see the nature of (his problem at rts worst, consider a CPU that has instruc- instrucns with two addresses, such as the Motorola 680x0. wfdely used in embedded ystems. The instruction MOV.L#6(A1),2(A0) 6 bytes, for example (see Fig. 4-32). In order to restart the instruction, the op- oprating system must determine where the first byte of the instruction is. The value

ShC 4 7 IMPLEMENTATION ISSUES 245 ut the program counter at the lime ot Uit trap depends on which operand faulted and how the CPU's microcode has been implemented h -16 Bits MOVE e 2 *■ Opcode First op Second In Fig. 4-32, we have an instruction starting at address 1000 that makes three memory references: the instruction word itself, and two offsets for the operands. Depending on which of these three memory references caused the

page fault, the program counter mighi be 1000, 1002. or 1004 at the time of the fault. It is fre- frequently impossible for the operating system to determine unambiguously where the instruction began. If the program counter is 1002 at the time of the fault, the operating system has no way of telling whether the word in 1002 is1 a memory address associated with an instruction at 1000 (e.g.. the location of an operand), or an instruction opcode. Bad as this problem may be, it could have been worse Some 680x0 address- addressing modes use autoincrementing, which means that a side effect of executing the instruction is to increment one or more registers. Instructions that use autoincre- ment mode can also fault. Depending on the details of (he microcode, the incre- increment may be done before the memory reference, in which case the operating sys- system must decrement the register in software before restarting the instruction. Or. the autoincrement may be done after the memory reference, in which case it will not have been done at the time of the trap and must not be undone by the operat- operating system. Autodecremem mode also exists and causes a similar problem. The precise details of whether aistoincrements and auto decrements have or have not been done before the corresponding memory references may differ from instruc- instruction to instruction and from CPU model to CPU modtl Fortunately, on some machines the CPU designers provide a solution, usually in the form of a hidden internal register into which the program counter is copied just hefore each instruction is executed. These machines may also have a second register lellmg which registers have already been auto incremented or autodecre mented. and by how much. Given this information, the operating system can unambiguously undo all the effects of the faulting instruction so it can be started ill over again. If this information is not available, the operating system has \o ump through hoops to figure out what happened and how to repair it. It is a though the hardware designers were unable to .solve the problem, so they threw up their hands' and told the operating system writers to deal with it. Nice guy;

Although wi- have not discussed I/O much in [his chapter, the tact lhat a com- computer h;is virtual memory does not mean that I/O is absent. N ulual menioiv and I/O interact in subtle ways. Consider a process lhat has just issued a s>siem call to read trom some file or device into a bulfer withrn its address space While wjiitui° ior the I/O lo complete the process is suspended and another process is allowed U, run This othei process acts a page fault it the

pdginy algonlhtu is global, ihere is a sniall. hut non/eK>. chance that tli<_" [/() device is currently in the process ot doing a DMA uansfer to lhat page, le-moMiia it uull cause pan <>t the d.ita to he wnttcn m thebuffci wheie they bclonu problem is lo lock pases engaged in I/O in memory so that they \m|I n»l be removed Locking a page is often called pinning it m memory. Amnher solution is to do all I/O to kernL-l buffers and then copy the data to user pages later. 4.7.5 Backing Store selected for removal. We have not said mucli jhout where on the disk it is put when it is paged out Let us now describe some of rhe issues related lo di-k management. The simplct algorithm foi allocating p.ige space on the disk is to have a spe- special swap area on the disk. When the system is bouted, this area is empty, and is represented in memory as a single entry giving its origin and si/e When the fust process is. started, a chunk of the swap area the si^e uf the fust ihuccss is reserved and ihe remaining are.f reduced by tliat amount. As new processes are sorted, they are assigned chunks of (he .swap area equal m (,!«.■ to (heir core images. As they finish, their disk space k (reed The swap area is managed as a list of free chunks. Associated with each process is ihe disk jdtiress ol its swap area, kept in (he process tjble Calculating the address, to wnff a p.ige lo becomes simple- just add the offset of the page within the virtual address space to ihe .start o(the swap aiea. However, before a process can start, the swap area niusi be initialized. One way is to copy the entire process image to the s'wap area, so that it can be brought in as needed. The other is tr> load the entire process in memory aiid let n be paced out as needed. However, this simple model has a problem- processes can nicicase in si^e times grow_ and the stack can always grow Consetjiiently, n may be better io reserve separate swap areas for the text, data, and stack and allow each of these areas to consisi of more than one chunk on the disk.

|MFLE\U;M"A DON IS.SL t-S The clher extreme is to allocate nothing in advance and allocate disk space roi each paae when il is swapped out and deallocate tt when it it- swapped back in In this way. processes in memory do not tie up any swap space. The disadvantage i4. that a disk addiess is needed in memory to keep track of each p,)ge on disk. In oihcr words, theie must a table per process telling for each page on disk wheie rt is Tlie two alternatives arc shown in l;ig 4-33. In Fig. 4-33(a), a page- table with eisjht pages is iHustuted Pages 0. *. 4. and 6 are in mam memorv Pages 1.2. 5. and 7 jje on disk The swai^ *irea on disk is as large as the process virtual address space

(eight pages) vmIIi each page having a fixed location to which it is written when tt i\ evicted from mat a memoij. Calcu- Calculation of this address inquires knowing ohly wheru the process' paging aiea begins, sinei: pages arc stored in it contiguously in order of their \inual p;lge number. Pages that are in meinor; always have a shadow copy on disk, but this copy may be out ot date it the page has been modified since being loaded In Fig. 4-33(b|. p;igeh do not have fixed addresses on disk When a page is swapped out, Jn empty disk page is chosen on the tl> and ihe dKk map (which has room for one ihsk address per virtual page) is updated accordingly P;,gcs th;it aie in memory have no copy on disk Their entries in the disk map contain an invalid disk address or a bit maiking them as not m use. 4.7.6 Separation of Pulley and Mechanism An important tool fin n the pohcy from the mecri management by having mo; , c,f any system is to separate can be applied to memory ,er run as a user-level process.

248 MEMORY MANAGEMENT CHAP ■ Such a separation waK first done in Mach (Young ei a|, 1987) and in MINiX (Tanenbaum, 1987). The discussion below is loosely based on Mach. A simpie example of how policy and mechanism can be separated is shown in Fig. 4-34. Here the memory management system is divided into three parts 1 A |ow-|evel MMU handler 2 A page fault handler that is part of the kernel. 3. An external pager running in user space. All the details of how the MMU works are encapsulated in the MMU handler, which is machine-dependent code and has to be rewritten for each new platform the operating system if, ported to. The page-fault handler is machine independent code and contains most of the mechanism for paging. The policy is largely deter- determined by the external pager, which runs as a user process. 3 Reauestpage ^~-r^ 2 Need*./* 'Page! / / Fault ^ [handler (ExlB,nalY i pager I r /5 Here V is page / « 4 Page """ '—V MMU > v. / Figure 4-34. Page fault handling wiih an exiemal pager When a process starts up, the external pager is notified in order to set up the process page map and allocate backing store on the disk if need be As the proc- process runs, it may map new objects into its address space, so the external pager is again notified. Once (he process starts running, it may get a page fault The fault handler fig- figures out which virtual page is needed and sends a message to the external pager, telling what the prohlem is. The exiemal pager then reads ihe needed page in from the disk and copies it to a portion of us own address space Then it tells the fault handler where the page U. The fault handler then unmaps the page from the

external pager's address space and asks the MMU handler !o put it into the user's address space at the right place. Then the user process can he restarted.

SEC 4 7 IMPLEMENTATION ISSUES 249 This implementation leaves open where the page replacement algorithm is pm Ii would be cleanest to have it in the external pager, but there are some prob- problems wnh this approach Principal among these is that the external pager does not have access to the R and M btts of all (he pages These bits play a roie m many of the paging algorithms Thus either some mechanism is needed to pass this infor- information up to the external pager, or the page replacement algorithm must go in the kernel In the latter case, the fault handler tells the external pager which page it has selected for eviction and provides the data, either by mapping it into the exter- external pager's address space or including it in a message. Either way. the external pager writes the data to di!,k The main advantage of this implementation is more modular code and greater flexibility The main disadvantage is the extra overhead of crossing the user- kemel boundary several times and the overhead of the various messages being sent between the pieces of the system. Ai the moment, ihe subject is highly con- controversial, but as computers get faster and faster, and the software gets more and more complex, m the long run sacrificing some performance for more reliable software will probably be acceptable 10 most implemented- 4.8 SEGMENTATION The virtual memory discussed so far is one-dimensional because ihe virtual addresses go from 0 to some maximum address, one address after another. For many problems, having two or more separate virtual address spaces may be much better than having only one. For example, a compiler has many tables ihat are built up as compilation proceeds, possibly including t. The source text being saved for the printed listing (on batch systems). 2. The symbol table, containing the names and attributes of variables. 3. The table containing all Ihe integer and floating-point constants used. 4 The parse tree, containing the syntactic analysis of the program. 5. The stack used for procedure calls within the compiler. Each of the first four tables grows continuously as compilation proceeds. The last one gruws and shrinks in unpredictable ways during compilation. In a one- dimensional memory, these five tables would have to be allocated contiguous chunks of virtual address space, as in Fig. 4-35. Consider what happens if a program has an exceptionally large number of variables but a normal amount of everything else The chunk of address space allocated for the symbol table may fill up. but

there may be lots of room in me other tables. The compiler could, of course, simply issue a message saying that the compilation cannot continue due to too many variables, but doing so does not seem very sporting when unused space is left in the other tables.

250 MF.MORY MANAGF.Mr-.NT CHAP. 4 Virtual address space Call stack Free Address space [' SpBce cllrrent|y being allocated to the « Parse tree f ^d the a^e treMe parse tree I ' Constant labk * I' Symbol table has burrped into the source text table Figure 4-35. In a une-dinicnsional address spucc with growing tables, one laMc may biiTnp mtu arnrther. Another possibility is to play Robin Hood, taking space from the tables with an excess of room and giving it to the tables with little room. This shuffling can be done, but it is analogous to managing one's own overlays—a nuisance at best and a great deal of tedious, unrewarding work at worst. What is really needed is a way of freeing the programmer frum having to manage the expanding and contracting tables, in the same way that virtual mem- memory eliminates the worry of organizing the program into overlays. A straightforward and extremely general solution is to provide the machine with many completely independent address spaces, called segments. Each seg- segment consists of a linear sequence of addresses, from 0 to some maximum The length of each segment may be anything frotn 0 to the maximum allowed. Dif- Different segments may. and usually do, have different lengths. Moreover, segment lengths may change during execution. The length of a stack segment may be increased whenever something is pushed onto the stack and decreased whenever something is popped off the stack. Because each segment constitutes a separate address <ipace. different seg- segments can grow or shrink independently, without affecting each other. If a stack in a certain segment needs more address space to grow, it can have it. because there is nothing else in its address space to bump into. Of course, a segment can fill up but segments are usually very large, so this occurrence is rare. To specify an address in this segmented or two-dimensional memory, the program must sup- supply a two-part address, a segment number, and an address within the segment.

SEGMFNTATION Figure 4-36 illustrate,, a segmented memory being used fur [he compilt discussed earlier. Five independent segments are shown here Hgure4-36. A sej!Lnented JepcndenllV ot tht? ujher lahh We emphasize that a segrr ware ot and uses as a logical enti rray, or a stack, or a collect i< of si

ogical entiiy. which (he prograr em might contai inablcs. but i if doe: a mixiure of different types. A segmented memory has other advantages besides simplifying the handling of data structures that are growing or shrinking. If each procedure occupies a separate segment, with address 0 as its starting address, the linking up c.f pro- procedures compiled separately is greatly simplified. After all the procedures that constitute a program have been compiled and linked up. a procedure call to (he procedure in segment n will use the two-part address («. Ol to address word 0 (the entry point). If the procedure in segment n is subsequently modified and recompiled, no oiher procedures need be changed (because no starting addresses have been modi- modified), even if the new version is larger than the old one, Wiih a onc-dimensiona! memory, the procedures are packed tightly next to each other, with no address space^between them. Consequently, changing one procedure's size can affect the starting address of oiher, unrelated procedures This, in turn, requires modifying all procedures that call any of the moved procedures, in order ((> incorporate their new siarting addresses. If a program contains hundreds ot procedures, this proc- process can be costly. Segmentation alsci facilitates sharing procedures or dala between several processes. A common example is the shared library. Modem workstations (hat

MEMORY MANAGEMENT librar p y y pg egmented system, [he graphical library ..-an be put in a segment and shared by multiple processes, eliminating the need tor having it in every process' address spate. While it is ako possible U, have shared in pure paging systems, it is much more complicated In effect, these .terns do it by simulannng segmentation Because each segment forms a logical entity of which the programmei is piled into nearly every progra libram different kinds nf prelection. A procedure segment can he specified as execute only, prohibiting attempts to read from it or store into it A floating-point array can be specified as read/write but not execute, and attempts to jump to it will be caught. Such protection is helpful in catching programming errors You should try to understand why protection makes sense in a segmented memory but not in a one-dimensional paged memory In a segmented memory tain a procedure and a stack, for example, but cine or the other Since each seg- segment contains only one typa of object, the segment can have the protection appropriate for that pamcular type. Paging and segmentation are compared in Fig. 4-37. Consideration Need the programmer be aware that this technique is being used' How many linear address spaces are there" Can the total address

space excaed the size of physicei memory' Can procedures and data be distinguished and separately protected? Can tables whose size fluctuates be accommodated easily? Is sharing of procedures Why was this technique invented? Paging No 1 Yes No No No To get a large linear address space without more physical memory Segmentation Yes Many Yes Yes Yes Yes To attow programs and data to be broken independent address spaces and to aid Figure 4-37. Comparison ol paging and se

SRC ■ SEGMENTATION 253 The contents of a page are, in a sense, accidental. The programmer is unaware of the fact (hat paging is even occurring. Although putting a few bits in each entry of the page table to specify the access allowed would be possible, to utilize this feature the programmer would have to keep track of where in his address space the page boundaries were. That is precisely the sort of administra- administration (hat paging was invented to eliminate. Because ihe user of a segmented memory has the illusion ihat all segments are in main memory all the time—that is. he can address them as though they were—he can protect each segment separately, without having to be concerned with the administration of overlaying them. 4.8.1 Implementation of Pure Segmentation The implementation of segmentation differs from paging in an essential way: pages are fixed size and segments are not. Figure 4-38(a) shows an example of physical memory initially containing five segments. Now consider what happens if segment I i*, evicted and segment 7, which is smaller, is put in its place. We arrive at the memory configuration of Fig. 4-3Sfb>. Between segment 7 and seg- segment 2 is an unused area—that is. a hole. Then segment 4 is replaced by segment 5, as in Fig. 4-38(c), and segment 3 is replaced by segment 6. as in Fig. 4-38(d). After the system has been running for a while, memory will be divided up into a number of chunks, some containing segments and some containing holes. This phenomenon, called checkerboarding or external fragmentation, wastes mem- memory in the holes. It can be dealt with by compaction, as <,hown in Fig. 4-38fe). Segment 4 (TK| Segment 3 (BK) Segment 2 EK| Segment 1 FK) Segment 0 Segment 4 <7K) Segment 3 (BK) Segments EK) Segment 7 EKj Segment 0 Segment 5 DK1 Segment 3 OK) Segment 2 |5K) Segment 7 <5K) Segment 0 <4K) Segment 5 I4KI Segment 6 Segment 2 EK) WW//> Segment 7 <SK) Segment 0 DK) pi! Segment 5 DK1 DKI Segment 2 EK) Segment 1 <5K) Segment 0 <4K) Figure 4-38. (a)-(d) Development of checkerboarding (e) Rem checkerboarding by compaction

254 MLMORY MANAGEMENT CHAP 4 4.8.2 Segmentation with Paging: MULTICS it the segments are large, it may be inconvenient, or even impossible, to keep them in main memory m their entirety. This leads to the idea of paging them, so that only those pages thai aie actually needed have to be around. Several signifi- significant systems have supported paged segments. In this section we will describe the first oner MU[,TICS In the next one we will discuss a more recent one. the Intel Pentium MULTICS ran on the Honeywell 6000 machines and their descendants and provided each program with a virtual memory of up to 2 segments (more than 250,000), each of which could be up to 65,536 C6-bit) woids long. To implement this, the MULTICS designers chose to treat each segment as a virtual memory and to page u. combining the advantages of paging (uniform page size and not having to keep the whole segment m memory if only part of it is being used) with the advantages ot .segmentation (ease of programming, modularity, protection, and sharing). Each MULTICS program has a segment table, with one desenptor per seg- segment. Since there are potentially more than a quarter of a million entries rn the table, the segment table is itself a segment and is paged A segment desenptor contains an indication of whether the segment ts in main memory or not. [f any part of the segment is in memory, the segment is considered to be in nieniory. and its page table will be in memory. If the segment is in memory, its desenptor con- contains an 18-bit pointer to its page table [see Fig. 4-39{a)]. Because physical addresses are 24 bits and pages are aligned on 64-byte boundaries (implying thai the low-order 6 bits ot page addresses are 000000), only 18 bits are needed in the descriptor to store a page table address The descriptor <ilso contains the segment size, the protection bits, and a few other items. Figure 4-39(b) illustrates a M1JL- TICS segment desenptor. The address ot ihc segment in secondary memory is not in the segment descriptor but in another table used by the segment fault handier Each segment is an ordinary virtual address space and is paged in ihe same way as the nonsegmented paged memory described earlier m this chapter The normal page size is 1024 words (although a few small segments used by MUL- MULTICS itself are not paged or are paged in units of 64 word*, to save physical mem- memory). An address in MULTICS consists of two parts, the segment and the address within the segment. The address within the segment's further divided into a page number and a word within the page, as shown in Fig. 4-40. When a memory reference occurs, the following algorithm is carried out. 1. The segment number is used to find the segment descriptor. 2. A check is

made to see if the segment's page table is m memory If the page table is in memory, it is located. If it is not, a segment fault occurs. If there is a protection violation, a fault (trap) occurs.

SEGMENTATION — — 36 btts i qm s e m e qm e Se m e qm e Se m e - — criptor cuptOr .cnptor Cfiptor >cnptor cnpior J. Page lable lor segrnent -Tl 1 = 64 words 0 = segmeni is paged 1 = segment is not page! Protection brts Figure 4-3!l. The Milt TIC <■ \ [rtual menu io the page tables <bl A se^irieru dcscrij» 3. The page table entry for the requested virtual page is examined, if the page is not in memory, a page fault occurs It i(is rn memory, the main memory address of the start of the page t^ extracted from the page table entry. 4. The offset is added to the page origin to give the main memory address where the word is located. 5 The read or store finally lakes place.

MEMORY MANAGEMENT number the page Figure 4-40. A 34-bit MU-TICS ^ nl dddrc riptoi rocess is illustrated in Fig. 4-4]. For simplicity, the fact that the segment is itself paged has been omitted. What really happens is that a register (the descriptor base register), is used to locate the descriptor segment's page table, which, in turn, points to the pages of the descriptor segment. Once the descriptor for the needed segment has been found, the addressing proceeds as shown in Fig. 4-41. MULTICS virtuat address Page number onset |Descriptor \ \(Page^- =age frame: kt.Wcrd&r Figure 4-41. Cor a two-part MLLTICS addie. As you have no doubt guessed by now, if the preceding algorithm were actu- actually carried out by the operating system on every instruction, programs would not mn very fast. In reality, the MULTICS hardware contains a 16-word high-speed TLB that can search all its entries m parallel for a given key. It is illustrated in Fig. 4-42. When an address is presented to the computer, the addressing hardware first checks to see if the virtual address is in the TLB If so. it gets the page frame number directly from the TLB and forms the actual address of the referenced word without having to look in the descriptor segment or page table.

SEGMENTATION Segment number 4 6 12 2 2 page 1 0 3 -, 2 Page 7 2 1 0 12 Prowfton Read/write Read oniy Read/write Execute oniy Execute only Age 13 10 2 7 g I 1 1 1 0 1 1 Figure 4-42. A simplified version of the MULTICS TLB. The e page sizes makes the actual TLB more complicated The addresses of the 16 mt Programs whose working set i brium with the addresses of thi run

efficiemly. If thi st recently referenced pages are kept in the TLB. > smaller than the TLB size will come to equili- entire working set in the TLB and therefore will ot in (he TLB, (he descriptor and page tables are actually referenced to find the page frame address, and the TLB is updated I include this page, the least recently used page being thrown out. The age field keeps track of which entry is the least recently used. The reason thai a TLB is used is for comparing the segment and page number of ali the entries in parallel. 4.8.3 Segmentation with Paging: The Intel Pentium In many ways, the virtual memory on the Pentium resembles MULTICS, including (he presence of both segmentation and paging- Whereas MULTICS has 256K independent segments, each up to 64K 36-hJt words, the Pentium has 16K independent segments, each holding up to 1 billion 32-bit words. Although (here are fewer segments, the larger segment size is far more important, as few pro- programs need more than 1000 segments, but many programs need large segments. The heart of the Pentium virtual memory consists of two tables, (he LDT (Local Descriptor Table) and the GDT (Global Descriptor Table). Each pro- program has its own LDT, but there is a single GDT, shared by all the programs on the computer The LDT describes segments local to each program, including its code, data, stack, and so on, whereas the GDT describes system segments, includ- including the operating system itself To access a segment, a Pentium program first loads a selector for (hat segment into one of the machine's six segment registers. During execution, the CS register holds the selector for the code segment and the DS register holds the selector for

MEMORY MANAGEMENT the data segment The othe, segment register a 16-bit number, as shown in Fig 4-43 CHAP Each selector i ip it used ,egnient register, the At the time a seleuor is loaded into a segment register, the corresponding descriptor is fetched from the LDT or GDT and stored in microprogram registers, so it can be accessed quickly. A descriptor consists of 8 bytes, including the segment's base address. si?e. and other information, as depicted irt Fiji. 4-44. is in pages j~1 S9 v v 1 S&QrriGnl is present in memcy Privilege tevel@-3) 0 System Figure 4-44. Pum,nm uxle > The format of the .selector :i>pmr Data ighttv s been cleverly chosen to make locating the descriptor easy. First cither the LDT or GDT is selected, based on selector bit 2, Then the selector i»; copied to an internal scratch register, and the 3 low-arder bits set to 0. Finally, the address of ejiher the LDT or GDT table is added to it. to give a direct pointer to the descriptor For example, ^elector 72

refers to entry 9 m the GDT. which is located at address GDT + 72 Let us trace the steps by which a (selector, offset) pajr is converted to a physi- physical address. As ;>oon as the microprogram knows which segment register is being

ShC. ;8 SEGV1ENTA1ION Used, it can find [he complete descriptor correspondnuj to that selec l 0) tly paged out. nap occurs It ilieri checks to ,see if tlie offset is hey or id (he end ot the segment, in which se a irap also occurs Logically, there should simply be a 32-bil field in the scriptor mvirig the size ot the segment, but there are only 20 bits available, so a lleiem scheme is used. It the Gbir (Gianuianty) field is 0, the Limit field is the act segment si^e. up to 1 MB It" it is 1- the- Limn field gives the segment si/.e in i segments up to 2 2 bytes ling that the segment the offset is in range, the Per adds the 32-bit Bine field in the descriptor to the oitset to form what is inear address, as shown in Fig. 4-45. The Base field is broken up into three pieces and spread all over the descriptor for compatibility with the 286, in which the B,M is on|> 24 bits In effect, the Base field allows each segment to start at an arbitrary place within (he 32-bit lmeai address space. Kigure 4-4S. C It paging is disabled (by a bit in a global control register), the linear address is interpreted as ihe physical address and ,sent (o the memory tor the read or wntc. Thus with paging disabled we have a pure segmentauon scheme, with each segment's base address given in its descriptor. Segments are permitted to overlap, incidentally, probably because it would be too much trouble and take too much time to verify thai they were all disjoint. On (he other hand, it paging is enabled, the linear address is interpreted as a virtual address and mapped onto ihe physical address using page tables, pretty much as in our earlier examples. The only real complication is that with a 32-bit virtual address and a 4-KB page, a segment might coniain I million pages, so a two-level mapping is used 10 reduce ihe page table size for small segments,. Each running program has a page directory consisting ot 1024 32-bit entries. It is located at an address pointed to by a global register- Each entry in this

 directory points to a page table also containing 1024 32-bit entries. The page table entries point to page frames. The scheme is shown in Fig. 4-46. Dir Page Offsei Page directory . . ■MB A Directory entry \ page iBbte Figure 4-46. Mappm T ^y Page (D) ar dddre aOle 1 m Page la We entry points Word selected In Fig. 4-46(a) we see a hnear address divided into three fields. On, Pas>e, and Offset. The Dir field

is used to index into the page directory to locate j pointer to the proper page table. Then the Page field is used as an index into the page table to find (he physical address of the page frame Finally, Offset is added to the address of (he page frame to get (he physical address of the byte or word needed. The page table entries are 32 hits each, 20 of which contain a page frame number. The remaining bits contain access and dirty bits, set by the hardware for the benefit of the operating system, protection bits, and other utility bits Each page table has entries for 1024 4-KB page frames, so a single page table handles 4 megabytes of memory. A segment shorter than 4M will have a page directory with a single entry, a pointer to its one and only page table. IA this way, the overhead for short segments is only two pages, instead of the million page<> that would be needed in a one-level page table. To avoid making repeated references to memory, the Pentium, like MULTirs has a small TLB that directly maps (he most recently used Dir-Page combina- combinations onto the physical address of the page frame. Only when the current combi- combination is not present in the TLB is the mechanism ot Fig. 4-46 actually carried oui and the TLB updated. As long as TLB misses are rare, perfoi

ShC 4N SEGMENTATION 261 I[is also worth noting thai it some application does not need segmentation but is content With a single, payed. ^2-bit address space, jh.it model is possible. All the segment registers *-dn be set up with the same selector, whose descriptor has Base = 0 and Limit set to the niaxiinum The instruction offset will then be the linear address, with only ,i single address space used—in effect, normal paging only one that used the lull power of the Intel MMU architecture ing goals ot implement ing pure paging, pure segmentation, and paged segments, while at the same time being compatible- with the 286. and doing all ot this effi- aenily, ihe resulting design is surprisingly simple and clean Although we have coveted the complete architecture ot the Pentium viitual memory, albeit briefly, it is worth saying a few words about protection, since this subject is intimately related to the virtual memory. Just ds the viMu^il memory scheme is closely modeled on Mb'LTlCS. so is the protection system The Pen- Pentium supports tour protection levels wrtli level () being the most privileged and level 3 the least These are shown m Fi£j 4-47 At each instant, a runniijg pio- gram Ls at a certain letxl. indicated by a 2-nit field m its PSW Each segment in the system also has a level As long ui, a program restricts itself to using segments at its own level, every- everything works fine Attempts to access data ar ,i higher

level art- permitted Attempts to access data at a lower level are illegal and cause traps Attempts to call procedures at a different level (higher or lower] are allowed, but in a carefully selector instead ol an addtess. This selector designates a descriptor called a call

262 MRMORY MANAGEMENT CHAP 4 gate, whidi gives the address of the procedure to be called Thus it is not possible to jump into the middle of an arbitrary «xie segment at a different level Only official entry points may be used. The concepts of protection levels and call gates were pioneered in mULTICS. where they were v iewed as protection rings. A typical ut,c for this mechanism is suggested in Fig. 4-47. At level 0. we find the kernel of the operating system, which handles I/O, memory management, itnd other critical matters At level I, the system call handler is present User pro- grams may call procedures here to have system calls carried out, but only a specific and protected list of procedures maj be called Level 2 contains library procedure--, possibly shared among many running programs User programs' may call these procedures and read their data, but they may not modify them Finally, user programs run at level 3. which has the least protection Traps and interrupts usc a mechanism similar to the call gates Thev, too. reference descriptors, rather than absolute addresses, and these desXiiptors point to specific procedures to be executed. The Type field in Fig. 4-44 distinguishes between code segments, data segments, and the various kinds of gates. 4.9 RESEARCH ON MEMORY MANAGEMENT Memory management, especially paging algorithms, was once a fruitful area tor research, but most of that seems to have largely died oft, at least for general- purpose systems. Most rea! systems tend to use \ome variation on clock, because it is easy to implement and relatively effective. One recent exception, however, is a redesign of the 4 4 BSD virtual memory system (Cranor and Parulkar. 1999). There is still research gotng on concerning paging m special-purpose and newer kinds of systems though. Some of this work relates to allowing uslt processes to field their own page faults and handle their own memory manage- management, possibly in an application-specific way (Engler et a!.. 1995) One area where applications may need to do their own paging in Special ways is mul- multimedia, so some research has looked at this (Hand. 1999). Another area tlut has some special requirements is hand-held personal communicators (Abutaleb and Li. 1997; and Wan and Lin. 1997). One final area is systems with 64-brt address spaces shared by many processes (Tallun et al.. 1995). 4.10 SUMMARY In this

chapter we have examined memory management We saw that the simplest systems do not swap or page at all Once a program is loaded mio mem- memory, it remains there until it finishes Some operating systems allow only one process at a time m memory, while others support multiprogramming.

SEC. 4.10 SUMMARY 263 The next step up if- swapping When swapping is used, (he system can handle more processes than it has room for in memory. Processes for which there is no room arc swapped out to ihe disk Free space in memory and on disk can be kept track of with a biimap or a hole list Modern computer often have some iorm ot virtual memory In (he simplest form, each process' address space is divided up into uniform sized blocks called pages, which can be placed into any available page frame in memory. There are many page replacemeni algorithms: two of the beik'r algorithms are aging and WS Clock. Paging systems can be modeled bv abstracting the page reference siring from the program and using the same reference string with different algorithms. These models can be used to make some predictions about paging behavior Td make paging systems work well, choosing an algorithm is not enough; attention to issues such as determining the working set, memory allocation policy, and page si?e are required. Segmentation helps in handling data structures that change size during execu- execution and simplifies linking and sharing, h also facilitates providing different pro- protection tor different segments Sometimes segmentation and paging arc combined to provide a two-dimensional virtual memory. The MULTICS system and the iniel Pentium support segmentation and paging These programs are idle waiting for I/O half the i is wasted? 2. In Fig. 4-21 we saw an example of how multiple jobs can run in parallel and finish 10 minutes of CPU timeL start simultaneously How long will the last one take us complete if they run sequentially? How long if ihe.v run in parallel'1 Assume 501 I/O 3. A swapping system eliminates holes by compaction Assuming a random distribution of many holes anil many data segments and a time to read or write a 32-btt memory word of 10 nsec, about how long does it take to compact |2K MB ' For simplicity, assume that word 0 is part of a hole and that the highest word in memory contains valid daw. 4. in this problem you arc to compare the storage needed io keep track of free memory using a bitmap versus using a linked list The 128-MB memory is altoeaied in unit-, ot

MEMORY MANAGEMENT ■^-hlT ■nrs1 me ■jge and holes. is require f « dfi ;h 64 KB a 16-bit >r each m Also length. ethwd" and Wh' me that ea a 16-

bil n. ich one is cli ex] bet node m the 1 -node field. tcr1 inked Us, Hov, ma t needs a iiy bytes ™.. rr...BJJ - . - ory order- [0 KB. 4 KB. 20 KB. |K KB, 7 KB. 9 KB, |2KB.and 15 KB Which (a) 12 KB [b> 10 KB 6. What is the difference between a physical address and a virtual address'* 7. For each of the following decimal virtual ,-iddresses. compute the virtual page number .ind offset for a 4-KB page and for an 8 KB page. 20000, ^2768. 60000 8. Using the page table ot Fig 4-10. give the physical address corresponding lo e;,ch of the following virtual addresses' (b) 4100 9. The Intel K086 processor does not support Virtual memory Nevertheless, some com- companies previously s-old systems that contained an unmodified K086 CPU .md do pay- paying. Make an educated guess as to hi™ they did it Him Think about the logidl loca- location of the MMU 10. The amount ot disk space trui musi be available tor page storage is related lo the max- maximum number of processes, n, the number of bytes in the virtual address s|j;tce. \, and the number ot bytes of RAM, r Give an expression foi the worst case disk space requirements How realistic is ihis arnouni1 11. Jt an instruction takes K! nsec and a page liujli lakes qin additional it nsec, gi<_c a liir- 12. A machine has a 32-bn address space and an K-KB page 1 he page table is entirely w hardware, with one 32-bit word per entry When a process starts, the page table k copied to the hardware from memory, at one word even' 100 nsec. It each process runs lor 100 msec (including the time to load ihe pajie iahlc), whut fraction ot the CPL time is devoted to loading the page tables'* split into a 9-bit top-level page table field, an 11 -bit second-level page table field, and an offset. How large are the pages and how many are there in the address space' 14. Suppose that a 32-bit viriual address is broken up into four fields, a, l>. t. ,md (/ The ftrsi three are used for i three-level page table sjsiem The fourth field, d <■. ihc offset. Does the number of pages depend on the sizi^ of all four fields'* If not, which

CHAP 4 PROBLEMS 265 15. \ computer has ^2-bit virtual addrcss.es and 4-KB pages The program and data together ht m the lowest page (O-40Q5) The st.ick tits m the highest page How tnany entries are needed Ln the page table if ttadiii,>na] (oiie-|e\el) paging is used? Hov, many page table entries arc needed loi (vio lecel paging, with 10 bits in each part ' 16. Below is an execution irace of a program liagmcnt tor a compute! with 5t2-byie pjges. The program is located at .uidress 1A20, nnd us stack pointer is at 8192 (the siack glows (owaid 0). Cjive the page lelercnce string generated bv this program bdeh instruction ocLiipies 4 livies (I vioid) including' immediate constants

Both Lo.id word 6144 into registei 0 Push icgtsierOt.nto the stack Call a procedure at 5120, stacking the return address C ompare the actual parameter to the immediate constant 4 Jump il equal to ^152 17. A computer whose processes have |O24 pages in then address spaces keeps its page tables in memory The overhead required fL»r ivjdin^ a word Irom the page table is 5 n,sec To reduce this overhead, the compiler has a TLB. which holds 12 [virtual page, physical page Irame) pairs, and can do j look up in I nsec Whai hit r.ite i1- needed io 18. The TLB on the VAX dues not contain an R bu Wh> ' 19. How can the asspciaticc memorv d^ciic needed lor a TLB be implemented in 20. A machine has 48-bit virtual addresses ,tnd ^2-bit physical addiesscs. Pages aie H KB How many entries arc needed for the page table ' 21. A computer with m S-KR ^ge. a 256-KB mam mirmor>, and a 64-CiB \ irtual address space uses an inverted pagi: table (o implement us uriual iuemot> How. big should the hash table be (o eiisuie a mean |;,^h chain length of k'ss than I ' Assume that the hash table si?c is a p!>ic'cr of mo 22. A student m a compiler design course [Eposes to the professor .1 ptojcn ot wruiiij! a compiler that will produce a list ot page relercnces that can be used io miplcincni the optimal page replacement algorithm. Is this possible ' Whv c»i why hot'* Is there .iny- thing that could be done io improve pa^uii* efficn-'ne> at run nine ' 23. If FIFO page replacement is used with tour paae frames .ind eight pages, how many nall> empty i Now repeat this problem Kir I RU 25. A small computer has (our page trames. Ai the tirsl clock nek, ih<_- K bits are 01 I I (page- 0 ts 0, the icst arc 1} At subsequent Jock ticks, the olucs ate 1011, iOlO, 1101, 0010, KtlO. 1100 and IKH1 It rlie ag,ng algontlim is Used with an 8 bil counter, give the values ot the foui counters alter ihe last tick

266 MLMORY MANAGEMENT /ill CHAP ■ 26. Suppose ihaiT = 400 in Fig. 4-21. Which p 27. In (he wSCkxk algorithm uf Fig. 4-22(ci, ihe hand points, lo i page with R = 0. If tT4(io, will this- page bo removed? What ahoui if T = 10IK)? 28 How lone doc*, it take to load a 64-K.B program frohl a disk whose average stek tnno ' if H) msec, whose rotation time is 10 msec, and whose [racks hold 32 KB UM<>r ■* 2-KB page si (b) for a 4-KB pjgc si randomly around (he disk and the number of cylinders o pages being on ihe same cylinder is negligible shown below (the 10 2H5 (a) Which page will NRfl replacu^ (b) Which page will FIFO replace' (c) Which page will I-RU replace'1 (d) Which page wilUecmd chance replace1 30. One of Ihe first timesharing machines, the PDP-I. had a nicnior\ of 4K 18-btt words. It held one

process at a tune in memory. When ihe scheduler decided to run another procesSi the process in memory was wruten [o *i paging drum, with 4K 18-bii words any word, rather than only ai word 0 Why do yol 31. A computer provides each process wuh 6S.5^6 iiippose tilts dru pagesi of" 16,386 bytes, and a stack <.i£c of 1*5,870 bvl space' [f the page si-te were 12 byles. could contain parts of two diffcrenl segments i^ proj:rans fit in the addrc neinber lhat a page may it' 33. 1 one process and read-write fiit the other' Why iir .M. It has been observed thai the number of insnucti directly pn>po1i"ual Lo the number of page tra available memory Ls doubled, the mean interval Suppose thai a normiil instruction 200] ii e (i.t : to handle the I. ns executed hemccn page faults i' ieh allocated to a program If the ;twcen page faults is also doubled bul if a page fault victurs u lake*. Program takes 60 sec to run. duorij?

CHAP 4 PROBLEMS 267 35. -\ group ol operating system designer lor the Ptuga) Compuier Company are thinking about ways or reducing the amount t'f backing More needed in their new operating sys- tem The head guiu has |iM suggested not bothering Hi sa^e ihc program test in the swap area ai nil, but just page it in directly from the binary file whenevw it is needed conditions- if any. doc, a « ork toi the daia > 36. A machine language instruction In load a M-ba word mio a registci contains the 12- liit .iddrtss ot the woid id be loaded What is the ina-diiium numbei ol page laulls lilts Which one occurs, in paging systems" Wind) one occurs in systems using pure scg- desenptor must be looked up, thc-n ihc page descriptor L>ocs the Tl,B also work ihis way. with two le\eK of lookup" 39. Plot a histogram and calculate the mean and median ol the si/es of executable bmaiy and dll tiles: on a UN|X system look at all executable hies Hi /bm. /nsi/hin. and the optimal page si7e for this computer just considering the code (not dataf Consider internal fragmentation and page ta^le size making some reasonable assumption yhiH]i the size of a page Table enir> Assume that all programs are equally likely in be run and thus should be weighted equally. 40. Small programs lor MS-DOS can be compiled as COM files These lilcs lire always loaded at address 0x100 in a single mcinwj segment that is Used foi code. data. ,ind slack Instrucltons that trdiiMcr control of execution, such as JMP and CALL, or iliai access iwiic data Irom fixed addresses hace the addresses compiled into the obfccl code "^rite a program thiii can relocate such a profirain nlc to run starting a\ an jirbi~ trary address Your program must si_an rhrough codu looking lor object codes loi insiruciions that refer to fixed memory addicsses. then modily those addresses

that point to memory rotations wiihin the ranpc to be reloCaicd You can lind the ob|cct codes in an assembly language pnigramn>ing text Note iliat doing this perlectly words may has'e values ih.u mmiic instrutiion object codes 41. Virile a program thai sfmulatcs a pat^ntg s^ste'n At iftc st;irt oi the proITianih the user should be asked to choose a page replacement algorithm, choosing from FIK3. I.Rtt, and at least one other On each cycle, read \hc nunibci ol the-referenced page from a file- Generate 3 JjMitig similar 111 Fig 4-2S, except rotated W degrees so thai each new page reference increases the length of the niitput hie by iinc 'me 42. Write a program that models the distance strtnu algorithm described ill the icxt The input 10 (he program is a list ol page references (contained in a (tie), plus ihe number of page frames of physical memory available If possible, use trace data from real programs instead ol randomly-generaled page relerences The program should m.un- Lain the sttiek o* pages, analogous to Fig- 4^5 At each page fault, a procedure should

ME-.MOrY MANAGEMENT

INPUT/OUTPUT One of ihe main functions of an operating system is to control all the computer's I/O (Input/Output) devices. It must issue commands lo the devices, catch interrupts, and handle errors- H should also provrde an interface hetwecn ihe devices and the rest of the system that is simple and easy to use To die extent possible, the interface should be the same for all devices (desiee independence). The I/O code represents a significant fraction of the total operating system. How the operating system manages I/O is The subject of this chapter This chapter is organized as follows. First we will look at some of (he princi- principles of I/O hardware, and then we will look ai I/O software in general. I/O software can be structured m layers, with each layer having a well-defined task to perform. We will look at these layers to sec what they do ami how (hey fit together. Following that introduction, we will look at several I/O devices in detail disks, clocks, keyboards, and displays. For each device we will look at iK hardware and software Finally, we will consider power management. 5.1 PRINCIPLES OF I/O HARDWARE Different people look at I/O hardware m different ways Electrical engineers look at it in terms of chips, wires, power supplies, motors, and all the other physi- physical components that make up the hardware. Programmers look at the interface

hack In this book we art' con- tcpned with programming I/O devices not designing, building', or maintaining them, so our intercs! will be restricted to how the hardware is piogiatnnu-d. not liow il woiks inside Nevertheless, the programming of many I/O devices is olien vill provide a lutle general background on I/O hardwJie ,Ls it relate-, to program- programming. It may be regarded as a review and expansion ot the introductory material in See I 4 5.1.1 I/O Devices I/O devices can be roughl) divided rnto two categories, block devices and character devices. A block device is one that stores infoimation in Iixed-si7,e blocks, each one with its own address.. Common block si7es range from 512 bytes to ^2.768 bytes. The essential property of a block device is that it is possible to read or write each block independently of all [he other ones. Disks are [he most common block devices If you look closely, the boundary between devices that jre block addressable and ihose that are not is not well defined. Everyone agrees that a dask is a block addressable device because no matter where the arm currently is1, it is always pos- possible to seek to another cylinder and then wait for the required block to iolate under the head. Now consider a tape drive used for making disk backups Tapes contain a sequence of blocks. If the tape drive is givtn a command to read block A', it can always rewind die tape and go forward until it tomes to block N This operation is analogous lo a disk doing a seek, except that it takes much longer. Also, it may or may not be possible to rewrite one block in ihe middle of a tape. Even if it were possible to uhe tapes as random access block devices, that is .stretching the poini somewhat they are normally not used that way The other type of I/O device it, the character device. \ character device delivers or accepts a stream of characters, without regard to any block structure [t is not addressable and does no! have any seek operation Printers, network interfaces, mice (for pointing), rats (for psychology lab experiments), and most other devices lhat are not disk-like can be seen as character devices This classification scheme is not perfect Some devices just do not fit in Clocks, for example, are not block addressable- N()r do t|ie> generate or accept character streams. All they do is cause mteirupts at well-defined intervals Meniory-niapped screens do not fit the model well either Still, the iikk|c-1 ol block and character devices is general enough that it can be used as ., basis toi making some of the operaiing system software dealing with I/O device indepen dent The file system, roi example, deals just with abstract block devices and leaves the device-dependent part to lower-lcve! software.

:- =; i PKINCU'I hS or I/O HARDWARE 271 I/O devices cover a huge uinge in speeds, whitii puh considerable prc-isuie on sohwaii- to pcrfurm well ova man> orders ot magnitude in data tatev hg 5- !,,.,,.. ,i,^ ,i:-,,;> mt,-. nf ..(iinr (ommoit (IcML-cs Most Af tlioe devko tend to | Device Data rate |~Keyboard _ io bytes/sec 1 Mouse _ , lOObytes/sec i 56K modem 7 KB'sec ' i Telephone channel j 8 K_a_set: 1 Dual ISDN lmes_ _ , 16 KB/sec_ ^ Laser printer 100 KB/sec I Scanner 400 KB/sec_ 1 Classic Ethernet 1 25 MB/sec j rijS_B(Uniyersal_Serial_BuS) __ f 1_5 MB/sec [Digital camcorder T 4 MB/sec I IDE^disk__^~ _ " 5MBAsec |~40x CD-ROM t 6 MB/sec | Fast Ethernet _ 12 5 MB/sec i [ISA bus^ _ _i6 7 MB/sec_ _, I JlDE (ATA-2) disk _ j 16.7 MB/sec \ FtreWire (IEEE 1394) _50 MB/se(l J fxGA~Monitor ' _ 60 MB/sec [SONET OC-12 network ~[_ 78 MB/sec SCSI Ultra 2 disk 80 MB/sec [Gigabit Ethernet 125 MB'sec ! lUttnum tape _ _ 320 MB/sec _ J [PCI bus ^_ T 528 MBjsec __ ■ | Sun Gigaplane XB backplane i 20 GB/sec 5.1.2 Device Controllers I/O units typically consis(ot a mechanical component and an electronic com- component It is often possible to separate the two portions to provide a more modular and general design. The electronic component is called the device ton (roller or adapter On personal computers, it often lakes thu form of a printed circuit card

272 INPUT/OUTPUT CHAP 5 that can be inserted mto an expansion slot The mechanical component is the device itself Thts arrangement is shown in Fife 1-5 The controller tard usually has a connecioi on it. rnto which a cable leading to the device itself can be plugged Many controllers can handle two, four, or even eight identical devices If the interface between the controller and device is a standard interface, either an official ANSI. lEfcE. or ISO standard or a de facto one, ihen companies can make contiollers or devices (hat fit that interface Many companies, tor example, make disk drives, thai match (he IDF or SCSI interface. The interface beiween the controller and the device is often a very low-level interface A disk, for example, might be formatted with 256 sectors ot 512 bytes per track. What actually comes off the drive, however, is a serial bit stream, Mail- Mailing with a preamble, then the 40% bits in a sector, and finally a checksum, also called an Error-Correct ing Code (ECC). The preamble ih written when (he disk is formatted and contains (he cylinder and sector number, the sector srze. arid similar data, as well as synchronization informal ion. The controller's job is to convert the serial bit stream into a block of bytes and perform any error correction necessary. The block of bytes

is typically first assembled, bit by bit, in a buffer inside the controller After its checksum h;is been veufied and the block declared to be error tree, it can then be copied to matn memory The controller for a monitor also works as a bit serial device at an equally low level. It reads bylcs containing the characters to be displayed from memory and generates the signals used to modulate the CRT beam to cause if to write on the screen. The controller also generates the signals for making the CRT beam do a horizontal retrace after it has finished a scan line, as well as the signals for mak- making it do a vertical retrace after the entire screen has been scanned. II it weie not for the CRT controller, the operating system programmer would have to explicitly program (he analog scanning of the tube With the contiollei, the operating sys- system initializes the controller with a few parameter, such as the number of charac- characters or pixels per line and number of lines per scieen. and lets the controller lake care of actually driving the beam 5.1.3 Memory-Mapped I/O Each controller has a few registers that aie used tor communicating with the CPU. By writing into these registers, the operating system can command the device to deliver data, accept data, switch itself on or off. oi otheiwise perform some action. By reading from these registers, the operating system c.in learn what the device's state is. whether it is prepared to accept a new command, and so on In addition to the control registers, many devices have a data buffer fhat the operating system can read and write For example, a common way for computers to display pixels on the screen is to have a video RAM. which is basically just ;i data buffer, available for programs or the operating system (o write into.

PRINCIPLES OF I/O HARDWARE The issue thus arises of how the CPU communicates with the control registers and ihe device data buffers Two alternatives exist- In the first approach, each control regisier is assigned an I/O port number, an 8- or 16-bit integer. Using a -nnni'il I K~\ illrini^tllMI <. ll^h li IN REG.PORT, the CPU can read in c REG. Similarly, using OUT PORT.REG r PORT and ; the CPU can wr ers, including n sors, worked (hi.. ... In this scheme, the address spaces f in Fig 5-2{a). The instructions IN R0r4 MOV R0,4 ontents ol REG to a control register. Most early comput- itidinframes. such as the IBM 360 ant! all of its succes- nory and I/O are different, as shown are completely different in this design The former reads the contents of I/O port 4 and puts it in RO whereas the latter reads the contents of memon, word 4 and puts it in RO. The 4s in these examples thus refer to different and unrelated is spa Two addr 1=1 Figure 5-2. (a) .Separate I/O .ind memor) sp.ii.-c (h)

Mcinorv-mapped I/O (O Hybrid The second approach, introduced with the PDP-I 1. is to map all the control registers into the memory space, as shown in Fig 5-2fb) Each control register i-. is called memory-mapped I/O. Usually, the assigned addresses are at the lop of the address space. A hybrid scheme, with memory-mapped I/O data bufters and

274 INPUT/Oli TPl'T CHAP i separate I/O ports for the eoiitiol registers is shown m Fig. 5-2<c). 'I he Pentium uses this archiicciure. with addresses 640K in 1M being reserved tor device data builers m IBM PC compatibles, in addition to I/O potf 0 thiough G4K How do these schemes work? In all cases when the f'Pll wants to read a woid. either tiom memoiy 01 trom an I/O poit. it puts the address n needs on Hie mt-morv space, rhe memor> responds, to the request II" it is I/O space, the I/O device respond- to the rcquesl If ihere is onlv memoiv space [.is in Fig. 5-2(bi], every memory mociule and every I/O deuce comparts the address lincs to tlie range of addresses that it services. It the addiess falls in its range, it responds to the request. Since- no address is ever assigned to both memory and an I/O device, there ts no ambiguity and no conflict The two schemes loi addressing the controllers have different Miengths and special I/O instructions are needed to read <ind write the device control registers, access 10 them requires the use of assembly code sinec there is no wa) to execute an]N or OUT instruction in C oi C++. Calling such a procedure adds overhead to controlling I/O In contrast, with memory-mapped I/O, device control lejjisteis variables. Thus wiih memory-mapped 1/0. a I/O device driver can he vwitten entirely in C. Without memory-mapped I/O. some assembly code is needed Second, with memory-mapped I/O. no special protection mechanism is needed to keep user processes from performing I/O. 'Ml the operating system has to do is refrain from putting that portion <>! the addiess space contartimg the con- control registers in any user's sutual address space Better yet. i! each device has us control registers on a different page ot the address space, the operating system tan give a user control over specific devices but not others by simply including the desired pages in its page I able Such a scheme can allow different device drivers to be placed in different address spaces, not only reducing keiiiel si^e but also keeping one driver from interfiling wiih others Third, with memory-mapped I/O. every uistmction that c.iu reference memoiv can also reference control registers Foi example, it ihere is an instruction. TEST, that tests- a memory word tor 0. it can also be used to test a Control register lor 0. which might be the signal that the device-

is idle and can accept a new command The assembly language code might look like tins LOOP TEST PORT 4 // check if port 4 is 0 BED READY // tf tl is 0 go to ready BRANCH LOOP // otherwise continue testing READY If memory-mapped I/O is not present, the control register must first be read into the CPU, then tes'Ied. requiring two instructions instead of one In the case of the

PRINCIPLES OF I/O HARDWARE 275 loop given Above, a fourth instruction hds to be added, slightly slowing down the responsiveness* of detecting an idle device In computer design, practically everything involves irade-offs. and thai is (he casf he-re too. Memory-mapped I/O also has its disadvantages First, mosi com- computers nowadays have some form of caching of memory words Caching a device control register would be disastrous. Consider the assembly code loop given above in the presence (,f caching. The first reference to PORT 4 would cause it to be cached. Subsequent references would just lake the value from the cache and not even ask the device Then when the device finally became reads, the software would have no way ot finding it out. Iiistead, the loop would go on forever- To prevent this situation with memory-mapped I/O, the hardware has to be equipped with the ability to selectively disable caching, for example, on a per page basis This feature adds extra complexify to both the hardware and the operating system, which has to manage the selective caching Second, if there is only one address space, then all memory modules and all I/O devices must examine all memory references to sec which one** to respond to. If the computer has a single bus. as in Fig. 5-3(a), having everyone look at every address is straightforward. Figure 5-3. (However, the trend in modern personal computers is to have a dedicated high-speed memory bus as shown in Fig. 5-3(b). a propcrt> also found in main- mainframes, incidentally. This bus if. tailored foi optimize memory performance. v>i(h no compromises for the sake of slow I/O deuces Pentium systems even have ihree exiernal buses (memory. PCI, ISA), as shown m Fig 1-1 l. The (rouble with having a separate memory bus oil memory-mapped machines is that (he I/O devices have no way of seeing memory addresses its they go by on Ihe memory bus, so they have no way of responding. Again, special measures have (O be taken to make memory-mapped I/O work on a system with multiple buses One possibiliiy is to first send all memory references to (he memory |f

276]NPUT/OU'IPUT CHAP 5 the memory fails to respond, (hcn the CPU tries the other buses This design can be made to work but requires additional

hardware complexity A second possible design is to put a snooping device on (he memory bus to pass all addresses presented to potentially interested I/O devices The problem here- is thai I/O device may not be able 10 process requests at the speed the A third possible design, which is (he one used on (he Pentium configuration of hg. l-l I, is to filter addresses in the PCI bridge chip Th.s chip contains range registers (hat are preloaded a(boot time For example, 640K to 1M could be marked as a nonmemory range Addresses that tall within one of the ranges marked as nonmemory sire lorwarded onto the PCI bus instead ot to memory. The dihadvantage of this scheme is the need for figuring out at boot time which memory addresses are not really memory addresses. Thus each scheme has argu- rricnts for iind against it. &o compromises and trade-offs are inevitable 5.1.4 Direct Memory Access (DMA) No matter whether a CPLr does or doe* not have memoiy-mapped I/O. il needs io address (he device controllers to exchange- data with them. The CPU can request daia from an I/O controller one byte at a tune but doing so wastes the CPU's time, so a different scheme, called DMA (Direct Memory Access) is often used The operating system can only use DMA](the hardware has a DMA controller, winch most .systems do. Sometimes this eoiitioller is integrated into disk controllers and other controllers, but such a design requires a separate DM\ controller lor each dtvice More commonly, a single DMA controller is available (e.g.. on ihe parentboard| for regulating transfers to multiple devices, often cnn- No matter where it is physically located, the DMA contioller lias access to (hi: system bus independeni ot the CPU, as shown in Fig. 5-4 It (.unljins several registers that can be written and read by the CPU These include a uienioiy address register, a byte couut registei. and one or more contiol registers The con- control registers specify the I/O port to use, the direction ot the transfer (reading Irom the I/O device or writing to the I/O device), the traiisler unit (byte at a time or word at a time), and the number of bytes to transfer in one burst To explain how DMA works, let us first look at how tlisk reads occur when DMA is n()t used. First the controller reads the block (one oi more sectors) from the drive serially, bit by hit. umtl the entire block is in the controller's intorn.il buffer Next, it computes the checksum to verrt> that no read error* have occurred. Then the controller causes an interrupt. When the operating system starts running, it can read the disk block from the controller's butter a byte or a word at a nine by executing ;i loop, with each iteration reading one byte or word from a controller device register and storing it in main memory.

PRINCIPLES OF I/O HARDWARE Figure 5-4. Opcratuiii ol ,\ I)VH lunsier When DMA is used, the procedure rs ditteient First the CPU programs the DMA controller by setting its registers sn it knows what to transfer where (step I from the disk into its lutcrnci! bulfcr and verify the checksum When ^alid. (iaia are m (he disk controller's buffer, DMA can begin The DMA controller inmates' (he transfer by issuing a read reques' iiver [he bus1 to the disk controller (step 2) This read request looks' like any other read request, and the disk contro]lei does not know or care whether it came from the CPU or from a DMA controllei. Typical 1>. the memory address to wiilc ro is on ihe bus' address lines so when the disk eomiollei fetches the next word troni its internal bufter, it knows where to write u The write to memoiy is another st;in- darci bus cycle (step 3) When the write is complete, the disk controller sends an acknowledgement signal to the disk controller, a|so over the bu.s (slep 4) The DMA controller then increments the memory address to use and decrements the byte count. I' the bvte count is still greater than (). steps 2 through 4 art' repeated until the count reaches () At that (ime, the DMA controller interrupts the CPU to let it know that the transfei is now complete. When the iterating system starts up. K does not have to copy the disk block to memory, it is alieady there. DMA controllers ^ury considerably in their sophistication Tlio simples! ones handle one transfer &{ a time, as described above More complex ones can be pro- programmed to handle multiple transfers at once Such controllers have multiple sets of registers internally, one for each channel 1 he CPU starts by loading each set of registers with the relevant parameters feu its transfei Each tiansfei must use a different device controller After ejicfi word is transtcned (steps 2 through 4) m Fig 5-4, (he DMA controller decides which device to service nexf It may be set up to use a round-robin algorithm, or it may h;ivc a priority scheme design in lavor some devices over others. Multiple requests to different device controller

278 INPUT/OUTPUT CHAP 5 may be pending at the same time, provided thai there is an unambiguous v,ay m tell (he acknowledgements apart Often a different acknowledgement lint? uti the bus is used for each DMA channel for this reason. M;my buses can operate in two modes- word-at-a-time mode and block mode. Some DMA controllers can also operate in either mode. In the former mode, the ope i r the transfer >rd and gets u. It the CPU also wants the bus, it has m wait The n ism k called cycle stealing because the device controller sneaks in and steals an occasional bus cycle from the CPU once in a

while, delaying it slightly. In block mode, the DMA controller tells the device io acquire the bus. issue a series of transfers, then release the bus. This ronn ot operation is called burst mode. It is more efficient than cycle stealing because acquiring the bus takes tinic and multi- multiple words can be transferred for the pnce of one bus acquisition The down side io burst mode is that il can block the CPU and other devices tor a substantial period of time if a long burst is being transferred. In ihe model we have been discussing, sometimes called fly-by mode, (lie DMA controller tells the device controller to transfer ihe data directly to main memory. An alternative mode that some DMA controllers use is to have ihe device controller send the word to the DMA controller, which then issues a second bus request to write the word to wherever it is supposed to go. This scheme requires an extra bus cycle per word transferred, but is more flexible in that it can also perform device-io-de\ ite copies and even memory-to-memory copies (by first issuing a read to memon and then issuing a wriit: 10 memory at a different address). Mom DMA controllers use phy ical rru.moiy iddresses for iheir transfers. Using physical addresses requires the opentmc system to convert ihe virtual address ot the intended memory buf|er into a phwcal address and write th(s phy- physical address into ihe DMA controller s iddres register. An alternative scheme used in a few DMA controllers is to write virtual addresses, inio the DMA con- controller instead. Then the DMA coniroller must use the MMU io have the virtual- to-physical translation done. Only m the case thai the MMU is part of ihe memory (possible, but rare) rather lhan part of the CPU, can virtual addresses be put on the bus. We mentioned earlier thai the disk first reads data into its internal buffer before DMA can start. You may be wondering why ihe controller does nol just Store the bytes in main memory as soon as it gets (hem from ihe disk In other words, why does it need an internal buffer" There are two reasons. First, by doing internal buffering, the disk controller can verify the checksum before stall- stalling a transfer. Jf the checksum i.s incorrect, an error is signaled and no transfei is done. The second reason is that once a d'sk transfer h;is started, the bits keep aniv- ing from the disk at a constant rate, whether ihe contioHer k ready for them or not. If the controller tned to write data directly 10 memory, it would have to go

 mer the system bus lor each word transfcried If the bus were hus> due to some nrher device using it W £ . m hurst mode), the controller would ha^e to watt It end up storing quite .1 lev,

words yud h.nui^a lot of" administration to do as well WVn the block is, buffered internally, the bus is not needed until the- DMA begin*., so the design ol the eontroller is much simpler beauisc the DMA transfer to memory is not trine critical (Some older coutioHers did. in tact, go directly to ■f) Not all compute e DMA. The limiting factor is not the speed of the I/O device) If there is no other w lo do. having the (fast) CH) wait lor the (slow) DMA controller to finish less Also, getting rid of the DMA controller and having the CPL' do all in software saves money, important on low-end (embedded) computers 5.1.5 Interrupts Revisited We briefly introduced interrupts in Sec I 4,3, but there is more to a typical personal computer system, the interrupt structure is as shown At the hardware level, interrupts work as follows. When ;in I/O dev ished the work given to it. it causes an interrupt (assuming that inte been enabled by the Operating system). It does thi< by asserting a Mg line that ii has1 been assigned. This signal is detected by the interru chip on the paremboard. which then decides what to do rk for it is point- pointhe woik |-ig 5-5 e has Jin- p un a bus cmtrollei if no other interrupts are pending, the interrupt controller processes the i rupt immediately. It another one is in piogress, or another device han m simultaneous request on a higher-priority interrupt request line on the bus

280 INPUT/OUTPUT CHAP 1 device is just ignored foi the momeni. In this case it conlinues to assert an inter- interrupt signal on the bus until it is serviced by the CPU To handle the interrupt, the controller puts ;i number on the address lines snecilvmi! which device wants attention and asserts a signal that interrupts the CPU The interrupt signal causes the CPU to stop what it is doinji and stilt doing something else. The number on the address lines is used as an index inio a table called the interrupt vector to fetch a new program counter This program counter points to the start of the corresponding interrupt service procedure Typi- ciillv traps and intenupts use the same mechanisrn trom this point on. and fre- frequently share the same interrupt vector The location of the interrupt vector can be hardwired into the machine or it can tie anywhere m memory, with a CPU regKter (loaded by the operating system) pointing to its origin Shortly after it starts running, the interrupt service procedure acknowledges the interrupt by writing a certain value to one of the interrupt controllers I/O ports This acknowledgement tells the controller that it is free to issue Another interrupt By having the CPU delay this acknowledgement until i[is ready to han- handle the next interrupt, race

conditions involving multiple almost simultaneous interrupts can be avoided As an aside, some (older) computers do not have ,, cen- centralized interrupt controller chip, so each device controller requests ils own inter- rupts The hardware always sa\es certain information before starting the servree procedure. Which information is saved and wliere it is saved varies greatly from CPU to CPU. As a bare minimum, the piogram counter must be saved, sn the interrupted process can be restarted At the other extreme, all the visible registers and a large number of internal registers may be sased as well. registers that the operating system can read tHit as needed A problem with this approach is that then the interrupt controller cannot be acknowledged until all potentially relevant information has been read out. lest a second interrupt over- overwrite the internal register saving the state This strategy leads to long dead limes when interrupts are disabled and possibly lost interrupts and lost data Consequently, most CPUs save the information on tlie stack However, this approach, too. has problems. To start with whose slack? If the current stack is used, it may well be a user process stack The stack pointer may not even he legal, which would cause a fatal error when the hardware tried to write some words at it Also, it might point to the end of a page. After several memory writes, the page boundary might be exceeded and a page fault generated. Having a page fault occur during the hardware interrupt processing creates ;i bigger prob- problem: where to save the state to handle the page fault? If the kernel stack is used, there is a much belter chance' of the stack pointer being legal and pointing to a pinned page. However, switching into kernel mode may require changing MMU contexts and will probably invalidate most or all of

SEC 5 I PRINCIPLHS OF I/O HARDWARE 281 the cache and TLB. Reloading all of ,hese. statically or dynamically will increase [he time to process an interrupt and thus waste CPU lime. Another problem k caused by the faci that niost modem CPUs are heavily pipelined and often superscalar (internal I > parallel). In older systems, after each instruction was finished executing, the microprogram or hardware checked to sec if there was an inicrrnpt pending. If so. the program counter and PSW were pushed onto the stack and the interrupt sequence begun. After the interrupt handler ran. the revere process iook place, wifh the old PSW and program counter popped from the stack and the previous process continued. This model makes the implicit assumption thai if an interrupt occurs jusi atler some instruction, all the instructions up to and including that instruction have been executed

completely, and no instructions after it have executed at al!. On older machines, this assumption was always valid. On modern ones it may not he. For starters, consider the pipeline model ot Fig l-6(a) What happens rf an interrupt occurs while the pipeline is full (the usual case)'* Many instructions are in vanous stages or' execution. When the interrupt occurs, the value of the pro- program counter may not reflect the correct boundary between executed instructions and nonexecoted instructions. More likely, it reflects the address of the next instruction to be fetched and pushed into the pipeline rather than (he address ot the instruction that just was processed by the execution unit. As a consequence, there may be a well-defined boundary between instructions that have actually executed and those that have not, but the hardware may not know what it is. Consequently, when the operating system must return from an interrupt, it cannot just start filling the pipeline from the address contained in the program counter. It must figure out what the last executed instruction was. often a complex task that may require analyzing the state of the machine. Although this situation is bad. interrupts on a superscalar machine, such as that of Fig, |-6(b) are far wor^e. Because instructions may execute out ot order, there may be no well-defined boundary between the executed and nonenecuted instructions. It may well be that instructions I. 2. 3. 5, and 8 have executed, but instructions 4. 6. 7, 9. 10. and beyond have not Furthermore, (he program counter may now be pointing to instruction 9, [0, or 11 An interrupt that leaves the machine in a well-defined state is called a precise Interrupt (Walker and Cragon. |995) Such an interrupt has four properties. 1. The PC (Program Counter) ,s saved in a known place 2. AH instructions before the one pointed to by the PC have fully executed. 3. No instruction beyond the one pointed to by the PC has been executed. 4. The execution state of the instruction pointed toby the PC is known. Note that there is no prohibition on instructions beyond the one pointed to by the PC from starting. It is just that any changes they make to registers or memory

282 INPL I/OL TI'Ill <n[IAP 5 pointed 10 has been executed It is also permuted that it iias ool been tMtutcd However, it muM be clear wlneli use applies. Often, if ihe interrupt is an I'O interrupt, the inuructuui will not yet h.ivc Mailed Howaci, it Ihe interrupl is caused the fault so if can he restarted later An interrupt [hat does not mm these rcqutienieuis is called an imprecise interrupt and makes hie extremely unpleasant t,n (he operating system writer, who now has to liguie out vshat has happened <ind what sttH lias to happen.

onto the stack to give the operating system the possibility <,t figuring out ivhjl was going on. Saving a larsie amount of information tt> memory on every uitcTrupt makes interrupts slow and recovery even worse. This leads lo (he ironic situation of having very fast superscalar CPUs sometimes being unsuitable for real-ntiie work due to slow interrupts. Some computers are designed so that some kinds o| imerrupts and traps are precise and others are not F"or example, having I/O interrupts be precise but traps due to fatal programming errors be imprecise is not so bad since no attempt need be made to restart the running process Some machines have a bit that can be set to force all interrupts lo be precise. The downside of setting this bit is that it forces the CPU to carefully log everything it is domy and maintain shadow copies of registers so "it can generate a precise interrupt at any instant All this overhead has a major impact on performance Some superscalar machines, such as the Pentium Pro and all of its s-uecessois. have precise interrupts to allow old 386, 486. and Pentium I programs to woik correctly (superscalar was introduced in (he Pentium PrA. [he Pentium I just h,,d two pipelines). The price paid Tor precise mtemipts is extremely complex inter- interrupt logic within the CPU to make sure that when the interrupt controller signals that it wants to cause an interrupt, all instructions up to same point are allowed to finish and none beyond (hat point are allowed to have any noticeable effect on the machine state Here the price is paid nol in time, but in chip area and in complex- complexity of ihe design. If precise interrupts were nol required for backward compatibil- compatibility purposes, this chip area would be available for larger on-chip caches, making the CPU faster. On the other hand, imprecise interrupts make the operatrng sys- tern lar more complicated and slower, so it is hard to tel] which appro'ich is reyIIv 5.2 PRINCIPLES OF I/O SOFTWARE l~ct u*> now turn awav Iroin the I/O hardware and look at tbe I/O soflv First we will look at ihe goals of ihe I/O software and (hen at the different v I/O can be done from the point of view of the operating system.

SBC s 2 PRINClPI £S OF I/O SOFTWARE 28.5 5.2.1 Goals of the I/O Software A key concept m the design of I/O software is. known as device indepen- independence. What it means i*- that it should be possible (O wuie programs that ^jll access an\ I/O des'ice without having to specify the des'ice in advance For exam- ply, a program that read- a file as input should be able to redd a file on a floppy disk, on ;i hard disk, or on a CD-ROM, without having to modify the program lor e.ich different device- Similarly, one should be able

to type a command such as sort <input >output and have il work with input coming from a floppy disk, an IDE disk, a SCSI disk, or the keyboard, and the nuiput going to an> kind of disk or the screen. It is up to (he operating system 10 take care of the problems caused by the fact that these devices really are different and require very different command sequences to read Closely related to device independence is the goal of uniform naming. The name ol a file or a device should simply be a string or an integer and noi depend on the device in any way In UNIX, all disks can be integrated in the- file system hierarchy in arbitrary ways .so the user need not be aware ot winch name corresponds to which device For example, a floppy disk can be mounted on top of the directory /usr/axr/backup so thai copying a File to /uyr/ast/hackup/mondi'V copies ihe file to ihe floppy disk. In this way. all files and devices are addressed the same way: by a paih name Another important issue for I/O software is error handling. In general, errors should be handled as close to the hardware as possible. If the controller discovers a read error, u should try to correct the error itself if it can. If it cannot, then (he device driver should handle it, perhaps by just trying to read the block again Many errors are transient, such as read errors caused by specks of dust on the read head, and will go away if (he operation is repealed. Only if the lower layers are not able to deal with the problem should (he upper layers be told about tt. In many cases, error recovery can be done transparently a! a low level without the upper levels even knowing abou! the error Still another key issue is synchronous (blocking) versus asynchronous (intemipt-driven) transfers. Most physical I/O is asynchronous—the CPU starts the transfer and goes off to do something el.se unul the iiuerrupt arrives User pro- programs are much easier to write if the I/O operations are blocking—atier a read system call the program is automaiically suspended until the data are available in the buffer. 1(is up to (he operating system to make operations lhat arc actually interrupt-driven look blocking to (he user piogram.s. Anofher issue for the I/O software is buttering Often dala that come off a device cannot he stored directly in its iinal destination For exarnple vvhtn a packei comes in oft the network, the operating system does not know where 10 pul i! until it has stored the packet somewhere and examined it. Also, some devices

 have severe rcal-Urne constraints (lor example, thgilal audio devices). m> Ihe data buffer K filled from the rate at which it h emptied, in orilei to avoid buffer undci- runs Buffering invokes

c<.vnsider;ih|e copying and tiften h;is ;i majoi itripact on I/O peiformaiKe The final concept thai we will mention here is sharable versus dedicated devices Scime I/O devices, such as disks, can be used by many users at ihe sitme lime No problems arc caused l>v multiple users having open files on Lhc s.ime disk ai this same time Other devices, such ,k tape drive-,, have 10 be dedi^aied to a single i.esl'1 until that us^'r is finished 1 ht'n another tiser ctin have the tape drive willdeririiLly iitii work Introducing dedicated (unshared) dcvn.es also introduces a variety t>t problems, sUfh as deadlocks Again, the operating system must be able to handle hotn shaied and dedicated devices in a way th.il avoids problems 5.2.2 Programmed I/O There ,,re three fundamentally different w.iys tiiat I/O can be performed In this seenon we wi|i look at the firs! one (piogrammed I/O) In the ne\t two sec- lions *c will examine the others iinterrupi-dmeu I/O .nut I/O using DMA) The simplest form t,f I/O is t<i have the CPU do all the \vt,ik This nu-tliod is ealled programmed I/O It is simplest to illustrate programmed I/O b> hvans ol an example Consider a User piocess that wants t.) print the eight-eliaraeie! stnnji "ABCDEFG1I" on the printer. It fiist assemble* Ihe stnnj; in ,i butler in uset space, as shown in [s\£ 5- 6(a). page 1 The usei process then acquires the punioi tor wming by making a system call open it 1! the printer is currently in use by another pioeess. (his calf will tail

SFC i 2 PRINCIPLES OF I/O SOF1 WARE 285 and return an error code or will block until the printer is available, depending on the operating system and the parameters of (he call. Once rt has the printer, the user process makes a syMem call telling the operating system tt> print the siring on rhe printer The operating system then (usually) copies the buffer with the siring to an array, say, p. in kernel -.pace, where it is more easily accessed (because the kernel may h,)ve to change (he memory map to get ai user space) It then checks to see ,f the printer is currently available. If not. it waits until rt is available. As soort as ihe printer is available, the operaiing system copies the first character to the printer's datn register, in this example using memory-mapped I/O. This action activates the printer The character may noi appear yet because some printers buffer a line or 4 page befoie printing anything. In Fig. 5-6(b), however, we see that (he first character has been printed and that the system has marked the "B"' as the nexi character to be printed. As soon as i(has copied the first character to the printer, the operating system checks to see if the printer is ready to accept another one Generally, the printer has a second register, which

gives its status The act of writing to the daia register causes the status to become not ready. When the printer controller has processed the current character, it indicates rts availability by setting some bit in its status register or putting some value in it. At this point the operating system waiis for ihe printer to become ready again. When that happens, it prints the next character, as shown in Fig. 5-6(e) This loop continues until the entire string hus been printed. Then coiiirol returns to the user The actions followed by (he operating system are summarized in Fig 5-7. First the data are copied to the kernel Then the operating system enters a light loop outputting [he charaeiers one at a time. The essential aspect of programmed I/O. clearly illustrated in this figure, is that after outputting a character, the CPU continuously polls the device to see if it is ready to accept another one. Tins behavior is often called polling or busy waiting copy_trom_user(buffer, p, count); /* p is Ihe kernel bufer +/ lor (i = 0, i < count, i++) { /+ loop on every character */ while (*printer_status_reg r= READY), /- loop until ready -/ *pnnter_data register = p[i]: /* outpul one character -/ return _to_user(); figure 5-7. Writing <i slung L> the primer using programme*! I/O Programmed I/O i.s simple but has the disadvantage of lying up the CPU full tune until all the I/O is done If the time to ■'print" a character is very short (because all the primer is doing is copying (he nevv character to an internal buffer), then

 busy waiting is tine Also, in an embedded system, where the CPU has nothing else to do, birsy waiting is reasonable. However, in more complex systems, where ihe CPU ha* other work to do. busy waiting is inefficreni A bcticr I/O method is needed. 5.2.3 Interrupt-Driven I/O Now lei us consider ihe case of pnnlrng on a printer that does not butler char- characters but prints each one a.\ it arrives If the printer can print, say 100 characters/sec, each character takes II) msec to pnnt This means (hat after every charactei ^ written to the printer's data register, the CPU will sit in an idle loop for 10 msec waiting to be allowed to output the next character. This is more than enough time to do a context switch and run some other process for the]0 msec that would otherwise be wasted. The way to allow the CPU to do something else while waiting lor the printer to become ready is to use interrupts. Wlien the system call to print the string is made, the buffer is copied to kernel space, as we showed earlier, and the first character is copied to the primer as soon a.s it is willing to accept a character At that point the CPU calls the scheduler and sonic other process is run The process that asked

for the string to be printed is blocked until the entire siring has printed. The work done on the system call is shown jn Fig 5-8(a). copy_from_user(bufter, p, count); if (count -= 0) { enable_interrupts(); unblock _user(), while (*printer status_reg i= READY) . } else { •pnnter_data_register-p[0]; "pnnter dala register = p[i]; schedule^); count = count - 1, i = [+1; I acknowledge interrupt!), return _from_interrupt(); Figure 5-8. Wnunt a Jung m Hie primer u-ing iniernipi-drivcn I/O l.u Code When the printer has printed the character and is prepared to accept the nc\i one. it generates an interrupt This interrupt .stops the current process and sa\es its state. Then the printer interrupt service procedure is run A crude version ol this code is shown in Fig. 5-8{b) If there are no more characters to print, the interrupt handler takes some action to unblock the user Otherwise, it outputs the next character, acknowledges the interrupt, and returns to the process that was running just before the interrupt, which continues from where it left off.

SEC 5 2 PRINCIPLES OF I/O SOFTWARE 287 5.2.4 I/O Using DMA An obvious disadvantage of interrupt-dnven I/O is that an interrupt occurs on every character. Interrupis take time, so thts scheme wastes a certain amount of CPU time- A solution is. to use DMA. Here the idea is to lei the DMA controller feed the characters to the printer one at time, without (he CPU being bothered. In essence. DMA is programmed I/O. Only with the DMA controller doing ail the work, instead of the main CPU. An outline of the code is given in Fig. 5-9 copy_1rom_user(buffer, p, count); acknowledge_mterrupt(). ser_up_DMA_controller)): unblock _user(); schedulerj); return _trom_ interrupt), Figure 5-9. Printing a siring using DMA u) Code eXLtniint when ihu print sysiem c*ill is mads (b) Inlctrupt service pun-ednre The big win with DMA is reducing ihc number ot interrupts from one per character to one per buffer printed. If there are many character*, and interrupts are slow, this can be a major improvement. On the other hand, (he DMA controller is usually much slower than the main CPU. Tfr the DMA controller is not capable of driving the device at full speed, or the CPU usually has nothing to do anyway while waiting for the DMA interrupt, ihen interrupi-dnven I/O or even pro-programmed I/O may be belter 5.3 VO SOFTWARE LAYERS I/O software is typically organized in four l.iyers, ^ shown in Fig. 5-10 Each layer has a well-defined function to perform and a well-defined interlace to the adjacent layers. The functionality and interfaces differ from system to system, so the discussion that follows, which examines all the byeis starting ai the hottom, is not specific

to one machine 5.3.1 Interrupt Handlers While programmed I/O is occasionally uselul. for most I/O. interrupts are an unpleasant fact of life and cannot be avoided. They should he hidden away, deep in the bowels of the opeiating system, so thai a_s little ot the operating system as possible knows about them The best way to hide diem is to have the driver start- starting an I/O operaiion block until the VO has completed and the interrupt occurs. The driver can block itself by doing a down on a semaphore. ,i wait on a condition variable, a receive on a message, or something similar, for example.

INPUT/OUTPUT User-level I/O software Device-independent operating system software Interrupt handlers Figure 5-10. Layers ot the I/O software system When the interrupt happens, ihe interrupt procedure does whatever it has lo in order to handle the interrupt. Then it can unblock the driver thai started it In some cases it will just complete up on a semaphore. In others it will do a signal on a condition variable in a monitor. In sn|l others, it will send a message to the blocked driver. In all cases the net effect ol ihc interrupt will be that a driver that was previously blocked will now be able 10 run. This model works best if dmers are structured as kernel processes, wnh their own stales, stacks, and program Ol course, reality is nol quite so simple Piocessmg an interrupt is not just a matter of taking the mterrupl. doing an up on some semaphore, and then executing an iRET instruction to return from the interrupt to the previous process There ,s a great deal more work involved for the operating system We will now give an outline of this work as a series of steps thai must be performed in software after the hardware interrupt has completed ll should be noted that the details .ire very system dependent, so some ot the steps listed below may not be needed on a par- particular machine and sleps not listed may be required Also, the sieps thai do occur may be in a different order on some machines |. Save any registers (including the PSW) ihat have not already been saved by the interrupt hardware 2 Set up a context for the interrupt service procedure. Doing this may involve setting up the TLB, MMU and a page table. 3. Sel up a slack tor ihe interrupt icivice piocedure 4. Acknowledge the interrupt controller. If there is no centralized inter- interrupt controller, reenable interrupts 5 Copy the registers from where they were saved (possibk some stack) to the process table ■act information from e process table 6 Run the interrupt se the interrupting devi e procedure It will e ontroller's registers.

SEC 5. i 1/0 SOFTWARE LAYERS 289 7. Choose which process to run next It the interrupt has caused some high-priority process that was blocked to become ready, it may be chosen to run now 8. Set up the MMU context for the process to run next. Some TLB set up may also be needed. 9 Load the new process' registers, including its PSW. |0 Start running the new process. As can be seen, interrupt processing is fai from trivial. It also takes a consider- considerable number of CPU instructions, especially on machines m which virtual memory is prcscni and page tables have to be sei up or the slate of the MMU stored (e g., the fi and M bits) On some machines Ihe TLB and CPU cache may also have to be managed when switching between user and kernel modes, which takes additional machine cycles 5.3.2 Device Drivers Earlier in this chapter we looked at what device controllers do We saw that each controller has some device registers used to give it commands or some device registers used (o read out its status or both. The number of device icgistcrs and the nature of the commands vary radically from device to device. For exam- example, a mouse driver has to accept information from the mouse telling how far it has moved and which buttons are currently depressed In contrast, a disk driver ha.s to know about sectors, tracks, cylinders, heads, arm motion, motor drives, head settling times, and all the other mechanics of making the disk work propeily. As a consequence, each I/O device attached to a computer needs some device-specific code for controlling it This code, called the device driver, is generally written by the device's manufacturer and delivered along with the device. Since each operating system needs its own drivers, device manufacturers commonly supply drivets for several popular operaimg systems Each device driver normally handles one device type, or at mosl, one das'- of closely related devices. For example, a SCSI disk driver can usually handle mul- multiple SCSI disks of diflerent sizes and differeni speeds, and perhaps a SCSI CD- ROM as well. On the other hand, a mouse and joystick are so different that dif- different drivers are usually required. However, there is no technical restriction on having one device driver control multiple unrelated dev]ces. It is just not a good idea. In order to access the device's hardware, meaning the controller's icgisters, the device driver normally has to be part of the operating system kernel, at [cast with current architectures Actually, it would be possible to construct drivers that

290 INPUT/OUTPUT CHAP. 5 ran in user space, with system calls for reading and writing the device registers. In fact, this design would be a good idea,

since it would isolate the kernel from the drivers and the drivers from each other. Doing this would eliminate a major source of system trashes—buggy drivers that interfere with the kernel in one way or another. However, since current operating systems, expect drivers to run in the kernel, thai is the mode! we will consider here. Since the designers of every operating system know thai pieces of code (drivers) written by outsiders will be installed in it. it needs to have an architec- architecture that allows such installation. This means having a well-defined model of what a driver docs and how it interacts with the rest of the operating system. Device drivers are normally positioned below the rest of the operating system, as illustrated in Fig. i-11. User process Printer controfer) |camcorder controller! |CD-R0M controller Kernel space Hardware Operating systems usually classify dnvers into one of a small number of cat- categories. The most common categories are the block devices, such as disks, which

SEC 5 3 1/" SOfTWARh LAYERS 2V1 contain multiple data blocks that can be addressed independently, and the charac- character devices, such as keyboards and printc-rs. which generate or accept a stream ol characters. Most operating systems define a standard interface that all block driven, must support and a second standard interface that all character drivers must suppou These interfaces consist of a number of procedures thai the rest of the operating system can call to get th<3 driver to do work for it. Typical procedures are those to read a block (block device) or write a character string (character device) In some systems, the operating system i1- a single binary program that contains d‖ of the drivers thai it will need compiled inio it. This scheme was the norm for years with UNIX systems because they were run by computer centers and I/O devices rarelv changed If a new device was added, the system administrator sim- ply recompiled the kernel with the new driver to build a new binary. With the advent of personal computers, with thcii myriad of I/O devices, this model no longer worked Few users are capable of recompiling or relinking the kernel, even if thuy have the source code or object modules, which is not always the case. Instead, operating systems, starting with MS-DOS, went over lo a model in which drivers were dynamically loaded into the system during execution. Dif- Different sysiems handle loading drivers in different ways. A device driver has several functions The most ohvious one is to accupl abstract read and write requests from the device-independent software above it and see thai they are earned out But there are

also a few other functions they must perform. For example, the driver must initialize the device, it needed. It may also need to manage its power requirements and log events. Many device drivers have a similar general structure- A typical driver starts oul by checking the input parameters to see it they are valid. It not, an error is returned. It they are valid, a translation trom abstract to concrete terms may be needed. For a disk driver, this may mean convening a linear block number into the head, track, sector, and cylinder numbers for the disk's geometry Next the driver may check if the device is currently in use It it is\ the request will be queued for later processing. If the device is idle, the hardware status will be examined to see if the request can be handled now. It may be necessary to switch the device on or start a motor before transfers can be begun Once the device is on and ready to go. the actual conirol can begin Controlling the device means issuing a sequence ol commands to it The driver is the place where the command sequence is deiermined. depending on what has to be done. After the driver knows which commands it is going to issue, it starts writing them into the controllers device registers. After writing eych command to the controller, it may be necessary to check to see if the controller accepted the command and is prepared to accept the next one. This sequence con- continues until all the commands have been issued Some contiollers can be given a linked list of commands (in memory) and to|d to read and process them a|| by itself without further help from the operating system.

292 [NPUT/OUTPUT After t example writing ■ needed, : CHAP apply, in ; commands have been issued, one of two situat [he device driver must wait until the controller does some work lor H, itself until the interrupt comes in to unblock it In other cases, how- operation finishes without delay, so the driver need not block As mi he latter situation, scrolling the screen in character mode requires |ust w bytes into the controllers registers. No mechanical motion is ie entire operation can be completed in nanoseconds, rormer case, the blocked driver will be awakened by the interrupt. In the latter case, it will never go to sleep. Either way. after the operation has been completed, the driver must check for errors If everything is all right, the driver may have data to pass to the device-independent software (e.g., a block jus! read). Finally, it returns some status information for error reporting back to us caller If any other requests are queued, one of them can now be selected and started It nothing is queued, the driver blocks waiting for the next request. This simple model is

only a rough approximation to reality Many factors make the code much more complicated. For one thing, an I/O device may com- complete while a driver is running, interrupting the driver. The interrupt may cause a device driver to run. In fact, it may cause the current driver to run For eximple, while the network driver is processing an incoming patku another packet may arrive. Consequently, drivers have to be reentrant meaning thit a running driver has to expect that it will be called a second time before the lirst t ill has com- completed. In a hot pluggable system, devices can be ulded or temo\ed whiit the com- computer is running. As a result, while a driver is hus\ rt iding from some de\ n_e, the system may inform it that the user has suddenly removed thit de\ice from the sys- system. Not only must the current I/O transfer bt iborted without d mat,int, my ker- kernel data structures, but any pending requests for the now-vanished device must also be gracefully removed from the system and their callers given the bad news. Furthermore, the unexpected addition of new devices may cause the kernel to jug- juggle resources (e.g., interrupt request lines) taking "Id ones iwa\ trom the driver and giving it new ones in their place. Drivers are not allowed to make system calls but the\ orten need to inteiact with the rest of the kernel. Usually, calls to certain kernel procedures are permit- permitted. For example, there are usually calls to allocate and deallocate hardwired pages of memory for use as buffers. Other useful calls are needed to manage the MMU. timers, the DMA controller, the interrupt controller, and so on 5.3.3 Device-Independenl I/O Software Although some of the I/O software is device specific, other parts ot it are device independent. The exact boundary between the dnvers and the device- independent software is system (and device) dependent, because <,ume functu ar uld be done i y a dev ( ) penden, becaue ,ume functions ce-independent way may actually be done in the

I/O SOFTWARE LAYERS are typi- typically done in the device-independent software 1 Uniform interfacing lor c(evic£ drivers . Buffeting _j ! Error reporting . Allocating_and releasing dedicated devicesj Providing a device-independent block size_ ! figure 5-12. Funuitinsol ihe deuie-independeni I/O Mitt ware The basic function of the device-independent software is to perform the I/O function4- that are common to all devices dud to provide a uniform interface to the user-le\el software Below we will look at the above issues in more detail Uniform Interfacing for Device Drivers A major issue in an operating system is how to make all I/O devices and drivers look more-or-less

the same If disks, printers, keyboards, etc, arc all interfaced in different ways, every time a new device comes along, the operating system must be modified for the new device. Having to hack on the operating system for each new device is not a good idea. rest of the operating system. In Fig. 5-13(a) we illustrate a situation in which ^ach device driver has a different interface to the operating system. What this means is that the driver functions available for the system to call differ from driver to driver. It might also mean that the kernel functions that the driver needs also differ from driver to driver Taken togethei. it means that interfacing each new driver requires a lot of new programming effort. In contrast, in Fig 5-13(b). we show a different design in which all drivers have the same interface. Now it becomes much easier to plug in a new driver, providing it conforms to the dn\er interface. It also means that driver writers know what is expected of them (e.g.. what functions they must provide and what kernel functions they may call), [n pi act ice, not all devices aie absolutely identi- identical, but usually there are only a small number of device types and even these are generally almost the same For example, even block and character devices have many functions in common Another aspect of having a uniform interface is how I/O devices are named The device-independent software takes care ot mapping symbolic device names onto the proper driver. Foe example, in 1 MX a device name, such as /dev/dnkO, uniquely specifies the i-node for a special file, and this i-node contains ihe major device number, which is used to locate the appropriate driver. The i-node ako

294 INPUT/OUTPUT CHAP. 5 nn,JI_ i\r I _ ln_O jig ■ i i n j Disk driver Printer driver Keyboard driver Disk driver Printer driver Keyboard driver (a) (b) Figure 5-13. (a) Wilhiiui a <siaodan] driver interlace, (b) With a suruliird Jii\cr interface. contains the minor device number, which is passed as a parameter to the driver in order to specify the unit to be read or written. All devices have major and minor numbers, and all drivers are accessed by using the major device number to select the driver. Closely related to naming is protection. How does the system prevent users from accessing devices thai they are not entitled to access? In both UNIX tmd Windows 2000 devices appears in the file system as named objects, which means that the usual protection rules for files also apply to I/O devices. The system administrator can then set the proper permissions for each device. Buffering Buffering is also an issue, hoth for block and character devices for a variety of reasons. To see

one of them, consider a process that wants to read data from a modem. One possible strategy for dealing with the incoming characters is to have ihe user process do a read system call and block wailing for one character. Each arriving character causes an interrupt. The interrupt service procedure hands the character l« the user process and unblocks it After putting the character some where. Ihe process reads another character and blocks again. This model is indi- indicated in Fig. 5-14{a). The trouble with this way of doing business is that the user process has to be started up for every incoming character. Allowing a process to run many times for short runs is inefficient, so this design is not a good one. An improvement is shown in Fig. 5-l4(b). Here the user process provides an ^-character buffer in user space and does a read of /i characters. The interrupt ser- service procedure puls incoming characters m this buffer until it fills up. Then it

wake up ihe user process. This scheme is f.tr more efftuent than the previous one. but it, too, has a drawback, what happens if the buffer is paged out when a charadei arrives0 The buffer could be locked m memory, but if many processes start locking pages in mc-mon,. the pool ot available page will shrink and perfoi- manee will degrade. Yet another approach is to create a buffer inside the kernel and have the mtei- rupt handler put the characters there, as shown in Fig 5-l4(c) When this buffer is full, the page with the u^er buffer is brought in, if needed, and the buffer copied there in one operation. This scheme is far more efficient However, even this scheme suffers, from a problem- What happens to charac- characters that arrive while the page with the user buffer is being brought in from the disk' Since the buffer is fulL there is, no place to put them A way out is to have a second kernel buffer After the first buffer fills up. but betoru it has been emp- emptied, (he second one is used, as shown in Fig. 5-|4(d). When the second buffer fills up, it is available to be copied to the user (asbumitig the user has asked for it) While the second buffer is being copied to user space, the first one can be used tot new characters. In this way, the two buffers take turns- while one is being copied to user space, the other is accumulating iiev. input A buttering scheme like this is called double buffering. Buffering is also important on output Consider, tor example, how outpui i.s done to the modem without buffering using the model of Fig 5-14(h). The user process executes a wrtle system call to output n characters The system has two choices at this point, [t can block the usei until all the characters have been writ- written, but this could take a very long time over a slow telephone line. It

could .iko release the user immediately and do the I/O while the usci computes some more, but this leads to an even worse problem: how does the user process know that the output ha.s been completed and it can reuse the buffer" The system could generate

296 INPUT/OUTPUT CHAP. 5 a signal or software interrupt, but that style of programming is difficult and prone to race conditions. A much better solution is for the kernel to copy the data to j kernel buffer, analogous in Fig. 5-l4(c) (but the other way), and unblock the caller immediately. Now it does not matter when the actual I/O has been completed The user i« free to reuse the buffer the instant it is unblocked. Buffering is a widely-used technique, but it has a downside as well If data gel buffered too many times, performance suffers. Consider, for example, the net- network of Fig. 5- [5. Here a user does a system call to write to the network The kernel copies the packet to a kernel buffer to allow the user to proceed immedi- immediately (step I). Figure SOS. Networkm. i packet When the driver is called, it copies the packet to the controller for output (step 2). The reason it does not output to the wire directly from kernel memory is that once a packet transmission has been started, it must continue at a uniform speed. The driver cannot guarantee that it can get to memory at a uniform speed because DMA channels and other I/O devices may be stealing many cycles Failing to get a word on time would ruin the packet. By buffering the packet inside the con- controller, this problem is avoided. After the packet has been copied to the controller's internal buffer, it is copied out onto the network (step 3). Bits arrive at the receiver shortly after being sent, so just after the last bit has been sent, that bit arrives at the receiver, where the packet has been buffered in the controller Next the packet is copied to the receiver's kernel buffer (step 4) Finally, it is copied to the receiving process' buffer (step 5). Usually, the receiver then sends back an acknowledgement When the sender gets the acknowledgement, it is free to send the next packet However, it should be clear that all this copying is going to <,tow down the transmission rate considerably because all the steps must happen sequentially.

SEC. 5 3 I/O SOFTWARE LAYERS *97 Error Reporting Errors are far more common m the context of I/O than in other contexts When they occur, the operating system must handle them as beKt it can. Many errors are device-specific and must be handled by the appropnate driver, but the framework for

error handling is device independent. One class of I/O errors <tre programming errors. These occur when a process asks tor something impossible, such as writing to an input device (keyboard, mouse, scanner, etc) or reading from an output device (primer, plotter, etc.). Other errors include providing an invalid buffer address <ir other parameter, and specifying an invalid device (e.g., disk ? when the system has only two disks). The action to take on these errors is straightforward: just report back an error code to the caller. Another class of errors is1 the class or actual I/O errors, for example, trying to write a disk block thai has been damaged or trying to read from a camcorder that has been switched off. In these circumstances, it is up ti> the driver to determine what to do. If the driver does not know what to do. it may pass the problem back up to device-independent software. Whai this software does depends on the environment and the nature of the error. If it is a simple read error and there is an interactive user available, it may display a dialog box asking the user what to do. The options may include retrying a certain number of times, ignoring the error, or killing the calling process. If there is no user available, probably the only real option is to have the system call fail with an error code However, some errors cannot be handled this way. For example, a critical data structure, such as the root directory or free block list, may have been des- destroyed. In this case, the system may have to display an error message and ter- AHocating and Releasing Dedicated Devices Some devices, such as CD-ROM recorders, can be used only by a single proc- process at any given moment. It is up to the operating system to examine requests for device usage and accept or reject them, depending on whethei the requested device is available or not. A simple way to handle these requests is to require processes to perform opens on the special files for devjces directly. If the device is unavailable, the open fails. Closing such a dedicated device then releases it. An alternative approach is to have special mechanisms for requesting and releasing dedicated di-viees An attempt to acquire a device that is not available blocks the caller instead of failing. Blocked processes are put on a queue. Sooner or later, the requested device becomes available and the first process on the queue is allowed to acquire it and continue execution.

)evke-Independent Block Size Diffeiem disks may have different sector sizes, [t is up to the device- ndependem software to hide this fact and provide a unifonn block size to higher ayers. till example, by treating several sectors as -i single logical block In this vay. the higher layers

only ilcal with abstract devices that all Use ihe same logical)lock size, independent of the physical sector size. Similarly, some character lev ices deliver their data one byte at a time {eg., modems), while otheis deltver hens ,11 laiger units (eg., network interfaces) Thes^ differences may also be iiddcn S.3.4 User-Space I/O Software Although most of ihe I/O software is within the operating sjsienu a small por- portion ot it consists ol libraries linked together with user programs, and even whcile piograms running (,ut,side the kernel System calls- including the I/O system calls, aie normally made by ltbraiy procedures. When a C program contains the call count = wrileffd, buffer, nbyles). the library procedure nnte will be linked with the program and contained in the binary program present in memory at run time. The collection of all these ltbr;ir> procedures is clearly part of the I/O system While these procedures do little more than put thetr parameters in the appropriate place for the system call, there are other I/O procedures that actuall> do real work In particular, formatting of input and output is done by library pro- procedures. One example from C is printfi which takes j format string and possibly some variables as input, builds an ASCI] siring, and then calls write to output the string. As an example of prinff, consider the statement pnntffThe square of %3d is %6d\n", i. i*i), It formats a string consisting of the 14-character stnng '■The square of " followed by the value i as a 3-char,ieter stung, then the 4-L-|iaracrer string " is ". then r2 as six characters, and finally a line feed- An example of a similar procedure for input is stunt which leads input and i-tores it into variables described in a format suing using the same syntax a.±pnmf The standard I/O library contains a number of procedures that involve I/O and all run as part of user programs Not all user-level I/O software consists of library procedures. Another impor- important category is the spooling .system. Spooling 1S d way of dealing with dedicated I/O devices in a multiprogramming system. Consider a typieal Spooled device: a printer. Although ii would be technically easy to let any user process open the chaiacter special file for the printer, suppose a process opened it and then did nothing for hours. No other process could print anything.

1/0 SOFI'WMIE LAYERS 299 Instead what is done is to create a special process, called a daemon, and a sne^al directory, called a spooling directory. To print a file, a process first gen- generates the entire file to be printed and puts it in the spooling directory. It is up to the daemon, which is the only process having permission to use the printer's spe- special file, to print the

tiles in the directory By protecting the special file against direct use by users, the problem oT havjng someone keeping it open unnecessarily lonj! is eliminated. Spooling is not only used for printers It is dlso used in other stations. For example, file transfer over a network often uses a network daemon To send a file somewhere, a user puts it in a network spooling directory. Later on, the network daemon takes it out ;ind transmits it- One particular use of spooled file transmis- transmission is the USENET News system. Th,s network consists of millions of machines around (he world communicating using [he Internet- Thousands of news groups exiM on many topics To post a news message, the user invokes a news program, which accepts the message TO be posted and then deposits it m a spooling direc- directory tor transmission to other machines laier. The entire news system runs outside the operating system Figure 5-16 summaries the I/O system, showing all the layers and the princi- principal functions of each layer. Starting at the bottom, the layers are the hardware, interrupt handlers, device drivers, dev1ce-mdependeni software, and finally the user processes ' Devrce-independenl I/O reply I/O functions Make I/O call, format I/O. spooling Naming, protection, blocking, buffering, allocator Set up device registers check status Wabs up dwer when I/O completed Perform I/O operation Figure 5-16. Layers ol ihe I/O system and ibe mam tuuUiohs of each Ijyci The arrows in Fig. 5-16 s,how the flow of control. When a user program tries to read a block from a file, for example, the operating system is invoked to carry out the call. The device-independent software looks for it in the buffer cache, for example. If (he needed block ,.s not there, it calls the device driver to issue-trie request to the hardware to go get it from the disk. The process is then blocked until the disk operation has been completed-

|QQ iNPLT/OUTPlJ F CHAP ■? When the disk is finished, the hardware generates an interrupt. The interrupt randier is run to discover what hat. happened, that is. which device wants atten- ion right nnw It then extracts the status from this device and wakes up [he sieep- ng process to finish off the I/O reques! and let the user process continue 5.4 DISKS Now we will begin studying some real I/O devices We will begin with disks. \fter [hat we will examine clocks. ke>boards and display. 5.4.1 Disk Hardware Disks come in a variety of types. The most common ones are the magnetic disks (hard disks and floppy disks). They are characterized by the tact that leads and writer are equally fast, which makes them ideal us secondary memory (pag- (paging, file

.systems, etc.) Arrays of the.se disks are sometimes used to provide highly-reliable storage. For distribution of programs, data, and movies, various kinds of optical disks (CD-ROMs. CD-Recordablcs. and DVDs) are also impoi-tant. In the following sections we will first describe the haidware arid then the software for these devices Magnetic Disks Magnetic disks are organized into cylinders^ each one containing as many tracks as there are heads stacked vertically The tracks are disided into sectors, with the number of sectors aiound the circumference typically being 8 io 32 on floppy disks, and up to several hundred on hard disks The number of heads varies' from 1 to about 16 Some magnetic disks have little electronics and just deliver a simple scnal bit stream. On these disks, the controller does most oi the work. On othei disks, m particular, IDE (Integrated Drive Electronics) disks, the drive itself contains a microcontroller that does some work and allows the real controller to issue a set of higher-level commands. A device feature that has important implications foi the disk driver is ihe pos- possibility ot a controller doing seeks on two or more drives at the same time These are known as overlapped seeks While the controller and software are waiting for a seek to complete on one drive, the controller can initiate a seek on another drive. Many controllers eati a|so read or write on one drive while seeking on one or more other drives, but a floppy disk controller cannot read or write on two fjrrv^s a! trie same tirne. (Keading or writms requires the controller to move bits on a microsecond time scale, so one transfer uses up most of its computing power.) The situation is difterent for hard disks with integrated controllers, and in

,i system with m»re than one ot these hard drives, they can operate simultaneously, at least to the extent of transferring between the disk and the controller's buffer meniorv Only one transter between the controller and the main memory is possi- possible at once, however The ability to perform two or more operations at the same time cjd reduce the average access time considerably Figure 5-17 compares parameters or the standard storage medium for the ori- original ffiM PC with parameter1, of a modem hard disk to show how much disks have changed in the past two decades. It is interesting to note that not all parame- parameters have improved as much. Average seek time is seven times better, transfer late is I3(K1 times, better, while capacity is up by a factor of 50.000. This pattern hah to do with relatively gradual improvements in the moving parts, but much higher bit densities on the recording surfaces. j Parameter iNumber of cylinders Tracks per cylinder Sectors per track Sectors

per disk Bytes per sector , Disk capacity . Seek time (adjacent cylinders) j Seek time (average case) [Rotation time Hmotor stop/start time ^_ | Time to transfer 1 sector Figure 5-17. Disk parameters for j Western DigiuJ WD tSKXI hard One thing to be aware of in looking at the specifications of modern hard disks i.s that the geometry specified, and used by the driver software, may be different than the physical format On older disks, the number of sectors per track was the same for all cylinders. Modern disks are divided into zones with more sectors on the outer zones than the inner ones. Fig. 5-[8(aj illustrates a tiny disk with two zones. The outer zone ha.s 32 sectors per track: the inner one has |6 sectors per track. A real disk, such as the WD 18300. often has 16 ? oneK, with the number ot sectors increasing by about 4% per zone as one goes out from the innermost zone to the outermost /one. To hide the details ol how many sectors each track has. most modem disks have a virtual geometry that is presented to the operating system. The software is instructed to act as though there are x cylinders, v heads, and z sectors per track. The controller then remaps a request for (\. >. ?) onto the real cylinder, head, and

INPUT/OUTPI'T *1 di«k of Fig 5-|8<a> is. sho ^rs jnty the published arranj Figure 5-18, (a) Physical geu letry for this disk. ;ctor. A possible virtual geometry for the physi i Fig. 5-1 g(b). In both cases ihe disk has 192 set lent is different than the real one. For Pentium-based computers, the maximum % lues tor these three parime ■rs are often F5535. 16, and 63), due to the need to beb ckward compitible with le limiiations of Ihe original IBM PC On this m chine 16 4 nd 6 bit fields 'ere used to specify these numbers, with cylinders nd sectors tiumbered starting I I and heads numbered siarting at 0. With these parameters and 5B bytes pt.r ;ctor, the largest possible disk is 3|.5 GB. To t,et round this limit mam di ks ow support a system called logical block addressing in which disk sectors ire ist numbered consecutively starting a! 0, without regard to the disk geometry, LAID CPU performance has been increasing exponentially over the past decade, jughly doubling every 18 months. Not so with disk performance. In the 1970s, verage seek times on minicomputer disks, were 50 to |00 msec Now seek times re slightly under 10 msec. In most technical industries (say, automobiles or avia- on), a facior of 5 to 10 performance improvement in two decades would be lajor news, but in the computer industry n is an embarrassment. Thus ihe gap etween CPU performance and disk performance has become much larger over As we have seen, parallel processing is being used more and more to speed up "PU

performance. It has occurred to various people over the years that parallel

sr.c ^4 disks 303 I/O might be a good idea loo. In Iheir 1988 paper, Patterson et al. suggested six specific disk organizations that could be used to improve disk performance, relia- reliability, or both (Patterson et al.. |988). These ideas were quickly adopted by industry and have led to a new class of I/O device called a RAID. Patierson et ul defined RAID as Redundant Array of Inexpensive Disks, but industry redefined the [to be '"Independent" rather than 'inexpensive" (maybe so they could use expensive disks'?) Since a villain was also needed (as in RISC versus CISC, also due to Patterson), the bad guy here wds the SLED (Single Large Expensive Disk) The basic idea behind a RAID u; to install a box tull of disks next lo the com- computer, typically a large server, replace the disk controller card with a RAID con- controller, copy the data over to the RAID, and then continue normal operation. In other words, a RAID should look like a SLED to the operating system but have better performance and teller reliability. Since SCSI disks have good perfor- performance, low price, and the ability lo have up to 7 drives on a single controller A5 for wide SCSI), it is natural that most RAlDs consist of a RAID SCS[controller plus a box of SCSI disks that appear to the operating system as a single large disk In this way, no software changes are required to use the RA[D, a big selling point for many system administrators In addition to appearing like a single disk to the software, all RAlDs have ihc property that (he data are distributed over the drives, to allow parallel operation Several different schemes for doing this were defined by Patterson et al., and they are now known as RAID level 0 through RAID level 5. [n addition, there are a few other minor levels that we will not discuss The term "level" is something of a misnomer since there is n0 hierarchy involved; there are simply six different organizations possible RAID level 0 is illustrated in Fig. 5-19(a). It consists ,,f viewing the virtual single diKk simulated by the RA[D as being divided up into strips of k seciors each, with sectors 0 to i - 1 being strip 0. sectors k to 2k - 1 as strip [. and so on For k = I. each strip is a sector, for *: - 2 a strip is two sectors, etc The RAID level 0 organization writes consecutive strips over the drives in lound-romn fashion, as depicted m Fig. 5- [9(a) for a RAID with four disk drives Distributing data over multiple drives like this is called striping for example, if the software issues a command to read a dat<i block consisting of four consecutive strips start- starting at a slop boundary, the RAID controller will break this command up into four separate commands, one for

each of the four disk.s. and have them operate m parallel. Thus we have parallel I/O without the software knowing about it. RAID level 0 works best with large requests, the bigger the better. If a re- request is larger lhan the number of drives time*; the strip size, some drives will get multiple requests, so that when they finish the first requesl they start ihe second one. It is up to the controller to split the request up and feed the propei commands to the proper disks in the right sequence and then assemble the results in memory correctly. Performance is excellent and the implementation is straightforward.

@4 INPUT/OUTPUT CHAP. 5 RAID level 0 works worst with operating si.stums th;it habituall> ask lor data me sector at a time The iesults will be toned, but there ts no parallelism and lence no performance gain Another disadvantage of this organization is that the liability is potentially worse than having a SUED If a RAID (.(insists of lour lisks. each with a mean time to failure of 20.000 hours, about once every 5000 lours d duve will fail and all the data will be completely lost A SLED with a near nmu to f.iiluie of 20.000 hours would be four times more reliable Because io redundancy is present in this design, it is not really a true RAID The next option, RAID level I, shown in fit:. *>-\9(b). is a true RAID It Juphcates all the disks, so there art; four primary disks and four backup disks On l write. e\ery strip i.s written twice. On a read, either copy can be used, distnbut- ng the load over more drives Consequently, write perToiTnance is no better than foi a single drive, but read perfoimance can be up to jwice as good Fault toler- ince is excellent: if a drive crashes, the copy is simply used instead Recovery Unlike levels 0 and 1. which work with strips of sectors. R<\ID level 2 works on a word basis, possibly even a byte basis Imagine splitting each byte ot the sin- single virtual disk into a pair of 4-bit nibbles, flien adding a Hamming code to each one to form a 7-bit word, of which bits |. 2, and 4 were panty bits. Further ima- imagine that the seven drives of Fig 5- [9(C) were synchronized in terms ot arm posi- position and rotational position Then it would be possible to writs; the 7-bit Hamming coded word over the seven drives, one bit per drive. The Thinking Machines' CM-2 computer used this scheme, taking 32-bit data words and adding 6 parity bits to form a 38-bit Hamming word, plus an extra bit for word panty, and spread each word over 39 disk drives. The total throughput was immense, because in one sector time it could write 32 sectors worth of data Also, losing one drive did not cause problem'., because loss of a drive amounted to losing I bit in each 39-bit word read, something the Hamming

code could han- handle on the fly On the down side, this scheme requires all the drives to be rotationally syn- synchronized, and it only makes sense with a substantial number at drives (even with 32 data drives and 6 parity dnves. the overhead is [^ percent), h also asks a lot of the controller, since if must do a Hamming checksum every bit time. RAID level 3 is a simplified version of RAID level 2. It is illustrated in Fig. 5-I9(d), Here a single panty bi(is computed for each data word and written to a partly drive. As in RAID level 2, the drives must be exactly synchroni?ed. since individual data words are spread ov^r multiple dnves. At first thought, it might appear that a single parity bit gives only error detec- detection, not error correction. For the case of random undetected errors, this observa- observation is true However, for the case of a drive crashing, it provides fu‖[1-bit error correction since the position of the bad bit is known If a drive crashes, the con- controller just pretends that a‖ its bits are 0s. If a word has a parity error, the bit from the dead drive must have been a I. so it js corrected. Although both RAID levels

SRC. 5.4 DISKS 305 RAID level 2 RAID level 3 RAID level 4 RAID level 5 Figure S-19. RAID levels 0 through 5. Backup and paniy drives are shown shaded.

306 INPUT/OUTPU I <-HAP ^ 2 and 3 oflcr ver> h'gh data rdles. lhe number of separate I/O requests per second they can handle is no better than for a single drive. RAID levels 4 and 5 work with strips again, not individual words with parity, and do not require synchronized drives. RAID level 4 [see Fig 5-I9(e)l is like RAID level 0, with a stnp-tor-strip partly written onto an extra drive. For L-xam- ple it each str,p is t bytes long, all the strips are EXCLUSIVE ORed loKCthci. resulting in a parity strip k bytes long. [| a drive crashes, the lost b>te<. can be recomputed fiom lhe paiity drive. This design protects against the loss of a drive but performs poorly lor small updates. If one sector is changed, it is necessary to read all the drives in older to recalculate the parity, which must then be rewritten. Alternatively, it can read the old user data and the o|d parity data and recompute the new paiity troni them Even with this optimization, a ..mail update requires two reads and two writes. As a consequence of the heavy load on the parily drive, it may become a bottleneck This bottleneck is eliminated in RAID level 5 by distributing the par- parity bits uniformly over all (he drives, round robin fashion, as shown in Fig. 5- 19A). However, in the event ot a drive crash, reconstructing the

contents of the tailed drive is a complex process CD-ROMs in recen! years, optical (as opposed to magnetic) disks have become available They have much fughei recoiding densities than conventional magnetic disks Optical disks were originally developed for recording television programs, bui they can be put to more esthetic use as computer storage devices Due to ihen potentially enormous capacity, optical disks have been the sub|eet of a great deal of research and have gone through an incredibly rapid evolution First-generation optical disks were invented by the Dutch electtomcs conglomerate Philips, for holding movies. They were 30 cm across and marketed under the name LaserVision. but they did not catch Oil. except in Japan, In |9gO, Philips, together with Sony, developed the CD (Compact Dise). which rapidly replaced Hie 33 1/1-rpni vinyl record for music (except among con- connoisseurs, who still pierei vinyl) The precise technical details for the CD were published in dt\ ofticial international Standard (IS 10149). popularly called the Red Book, due to the color of its cover. (International Standards arc issued by the International Organization for Standardization, which is the international counter- counterpart of national standards groups like ANSI. DIN. etc. Each one has an IS number.) The point of publishing the disk and drive specifications as an Interna- International Standard is to allow CDs from different music publishers anil players from different electronic- manufacturers to work together- All CDs are I 20 mm across and I 2 mm thick, with a |5-mm hole in the middle. The audio CD was the first successful mass market digital storage medium. They are supposed to last [00 years. Please cheek hack in 2080 for an update on how well the first batch did.

StC 5 4 DISKS 307 A CD is prepared by using a high-powei infiarwi laser to burn 0.8-niicron dia- diameter hales in a coated glass masier disk From this master, a mold is made, with bumps where !he laser holes were, fnto this mold, molten polycarbonate resin is injected to form a CD with the same pattern of holes as the glass master. Then a very thin layer of reflective aluminum is deposited on the polycarbonate, topped by a protective lacquer and finally a label. The depressions in the polycarbonate siibsirate are called pits; the unburned areas between the pits are called lands. When played back, a low-power laser diode shines infrared light with t wavelength of 0.78 micron on the put, and lands as ihey stream by. Ttie laser is on the polycarbonaie side, so the pits stick out toward the laser as bumps in (he otherwise flat surface. Because the pits have a height of one-quarter the wavelength of me Us,er light,

light reflecting off a pit is half a wavelength out ot phase with light reflecting off the surrounding surface As a result, the iwo parts interfere destructively and return less light to the player's photodetecior than light bouncing off a land. This is how ihe player tells a pit from a land. Although K might seem simpler to use a pit to record a 0 and a land to record a I. it is more reliable to use a pit/land or land/pit transition for a I and its absence as a 0. so this scheme is used The pits and lands are written in a single continuous .spiral starting near the hoie and working (>ut a distance of 32 mm toward the edge. The spiral makes 22,188 revolutions around the disk (about 600 per mm) If unwound, it would be 5.6 km long. The spira) is illustrated in Fig. 5-20. Spiral groove Figure 5-20. Recording birueture of a compact <W or CD-ROM To make the music play at a uniform rate, it is necessary for the pits and lands stream by at a constant linear velocity. Consequently the rotation rate of the

308 INPUT/OUTPUT CHAP. 5 CD mus! be continuously reduced a-s ihe reading head moves from the inside oi the CD to the outside. At the inside, the rotation rare is 530 rpm to achieve the desired streaming rate of 120 cm/sec; at the outside it has to drop to 200 rpm to give the same linear velocity at the head A constant linear velocity drive is quiti.- different than a magnetic disk drive, which operates at a constant angular velocity, independent of where the head is currently positioned Aho, 530 rpm is a far cry from the 3600 to 7200 rpm that most magnetic disks whirl at. In 1984, Philips and Sony realized the potential for using CDs to store com puter data, so they published the Yellow Book defining a precise standard tor what are now called CD-ROMs (Compact Disc - Read Only Memory) To pig- piggyback on the by-then already substantial audio CD market, CD-ROMs were to be the same physical size as audio CDs, mechanically and optically compatible with them, and produced using ihe same polycarbonate injection molding ma- machines. The consequences of this decision were not only that slow variable-speed motors were required, but also that the manufacturing cost of a CD-ROM would be well under one dollar in moderate volume. What the Yellow Book defined was the formatting of the computer data, li also improved the error-correcting abilities of the system, an essential step because although music lovers do not mind losing a bit here and there, computer lovers tend to be Very Picky about that. The basic format of a. CD-ROM consists of encoding every byte in a 14-bit symbol. As we saw above, 14 bus is. enough to Hamming encode an S-bit byte with 2 bits left over In fact, a more powerful encoding system is used. The

l4-to-8 mapping for reading is done in hardware b> table lookup. At the next level up, a group of 42 consecutive symbols forms a 588-bit frame. Each frame holds 192 data bits B4 bytes). The remaining 396 bits are used for error correction and c-ontrol. So far, this scheme is identical for audio CDs and CD-ROMs. What the Yellow Book adds is the grouping of 98 frames into a CD-ROM sector, as shown in Fig. 5-21. Every CD-ROM sector begins with a N-byte preamble, the first 12 of which are OOFFFFFFFFFFFFFFFFFFOO (hexade- (hexadecimal), to allow the player to recognize the start of a CD-ROM sector. The next 3 bytes coniain the sector number, needed because seeking on a CD-ROM with its single daia spiral is much more difficult than on a magnetic disk with its uniform concentric tracks. To seek, the software in the drive calculates approximately where to go, moves the head there, and then starts hunting around for a preamble to see how good us guess was. The last byte of the preamble is the mode The Yellow Book defines two modes Mode I uses the layout of Fig. 5-21 with a 16-byte preamble, 2048 data bytes, and a 2gS-byte error-correcting code (a crossinterleaved Reed-Solomon code). Mode 2 combines the data and ECC fields into a 2336-byte data field for those applications that do not need (or cannot afford the time to perform) error correction, such as audio and video. Note that to provide excellent reliability, three separate error-correcting schemes are used:

42 Symbols make 1 Figure 5-21. Logical djia layout on ,\ CD-ROM within a symbol, within a trame, and within a CD-ROM sector Smgle-bit errors are corrected at the lowest ievel. short burst errors are corrected at the frame level, and any residual errors are caught at the sector level. The puce paid tor this relia- reliability is that it takes 98 frames of 588 bits G203 bytes) to carry a single 2048-byie payload, an efficiency of only 28 percent. Single-speed CD-ROM drives operate at 75 sectors/sec, which gives a data rate of E3.600 bytesAec m mode I and 175,200 byte^/sec in mode 2. Double- speed drives are twice as fast, and so on up to the highest speed. Thus, a 40x drive can deliver dala at a rate of 40 x 151,600 bytes/sec, assuming that the drive inter-interface, bus, and operating system can all handle this data rate. A standard audio CD has room for 74 minutes of music, which, if used for mode 1 daia. gives a capa- capacity of 681,984,000 bytes. This figure is usually reported as 650 MB because | MB is 21{i bytes A,048.576 bytes), not 1,000,(X>0 bytes. Note that even a 32x CD-ROM drive D.915.200 bytes/sec) is no match for a fast SCSI-2 magnetic disk drive ai 10 MB^sec. even though many CD-ROM

drives use the SCSI interface (|DE CD-ROM drives also exist) When you realise that the seek time is usually several hundred milliseconds, it should be dear that CD-ROM drives are not in the same performance category as magnetic disk drives, despite their large capacity In 1986. Philips struck again with the Green Book, adding graphics and the ability to interleave audio, video and data in the same sector, a feature essential for multimedia CD-ROMs. The last piece of [he CD-ROM puzzle is the tile system. To make it possible to use the same CD-ROM on different computers, agreement wat, needed on CD-ROM file systems. To gel ihis agreemeni, representatives of many computer companies met at Lake Tahoe in the High Sierras on the California-Nevada boun- boundary and devised a file system that they called High Sierra. It later evolved into an International Standard (IS 9660). It has three levels. Level I uses file names of up to 8 characters optionally followed by an extension of up to 3 characters (the

310 INPUT/OUTPI I <-IIAl' i MS-DOS rile naming convention) File names ina> contain only uppei uise let- letters digits, and the underscore. Directories may be nested up to eight deep, but directory iumes may not contain extensions Level I requires all files to be con- contiguous, which ,s nw a problem on a medium written only once Any CD-ROM conformant to IS 9660 level I can be read using MS-DOS. <m Apple computer, a UNIX computer, or just -ibout any oiher computer. CD-ROM publishers regard this property as being a big plus. IS 9660 level 2 allows names up to XL characters, and level 3 allots noncon- noncontiguous Hies The Rock Ridge extensions (whimsically named after the town ,n thts Gene Wilder film Blazing Saddles) allow very long names (tor UNIX). UIDs, OlDs, and symbolic links, but CD-ROMs not conforming to level I will not be readable on all computers- CD-ROMs have become extremely popular for publishing games, mo\ius, encylopedias. aliases, and reference works ot all kinds. Most commercial software now comes on CD-ROMs Their combination of large capacity and low manufacturing cost makes them well suited to innumerable applications CD-Recordables Initially, the equipment needed to produce a master CD-ROM (or audio CD, for that matter) was extremely expensive But as usuaf m the computer industry, nothing stays expensive for long. By |he mni 1990s, CD recorders no bigger than a CD player were a common peripheral available in most computer stores. These devices were still different from magnetic disks because once written, CD-ROMs could not be erased Nevertheless, they quickly found a niche as a

backup medium tor large hard disks and also allowed individuals or startup companies to manufacture their own small-ron CD-ROMs or make master*, for delivery to high-volume commercial CD duplication plants. These drives are known as CD- Rs (CD-Recordables). Physically. CD-RS Mart with 120-mm polycarbonate blanks that are like CD- ROMs, except that they contain a 0.6-mm wide groove to guide the laser for writ- writing. The groove has a sinusoidal excursion of 03 mm at a frequency of exactly 22.05 kHz to provide continuous feedback so the rotation speed can be accurately monitored and adjusted if need be CD-Rs look like regular CD-ROMs, except that they are gold colored on top instead of silver colored. The gold color conies from the use of real gold instead Of aluminum for the reflective layer. Unlike silver CDs, which have physical depressions An them, on CD-Rs the differing reflectivity of pits and lands has to be simulated. This is done by adding a layer of dye between the polycarbonate and the reflective gold layei, as shown in Fig. 5- 22. Two kinds of dye are used: cyanine, which is green, and pthalocyanme. which is a yellowish orange Chemists can argue endlessly aboul which one is better. These dyes are similar to those used in photography, which explains why Eastman Kodak and Fuji are major manufacturers of blank CD-Rs.

II II Polycatbon Direction Dve veiacauer) Qold layer If layer Substrate 1 i i-t-Dark spot in the dye layer burned ~~ by laser when wtiting Figure 5-22. Cross Wctiun of u CD R disk arid User (not 1.1 scale) A sil\er C D-ROM has a similar ^Iructnre, except without the dve. layer and wiih a pitted aluminum layer instead ill a gold layer In its initial state, the dye layer ,s transparent and lets the laser light past, through and reflect oft the gold layer. To write, the CD-R laser is turned Up to high power (8-| 6 mW) When the beam hits ,i spot of dye. it heats up. breaking a chemical bond. This change to the molecular structure creates a dark spot. When read back (at 0.5 raW), the photodetector sees a difference between the dark spots where the dye has been hit and transpaient areas svhert it is. intact. This differ- difference is interpreted jis the difference between pits and lands, even when read back on a regular CD-ROM reader or even on an audio CD playei No new kind of CD could hold Up its head with pride without a colored book so CD-R has the Orange Book, published in |l>89. This document defines CD-R and also a new format, CD-ROM XA, which allows CD-Rs to be written incre- incrementally, a few sectors today, a few tomorrow, and ;, few nexi month A group of consecutive

seciois' written at once is called a CD-ROM track One of the first uses of CD-R was for the Kodak PhotoCD. [n this system the customer brings a roll of exposed film and his old PhotoCD to the photo processor and gels back the same PhotoCD with the new pictures added after the old one-,. The new b;itch. which is created by scanning m the negatives is written onto the PhotoCD as a separate CD-ROM track. Incremental writing was needed because when this product was introduced, the CD-R blanks were loo expensive tti provide a new one for evory film ro‖. However, incremental writing creates a new problem, Prior to tho Orange Book, all CD-ROMs had a single VTOC (Volume Table of Contents) at the

312 IMHT/Ol I'PUl <-'HAP S start lhat scheme doc-s m,l work with mcicmenlal d c. multilraek) write [<ie Oiaune Books solution ,s to give each CD ROM irack its own VTOC The tile*. listed in tile VTOC tan include some or all ol the hies Itoin previous tracks. Alter ihe CD-R is inseited into the drive, the operating system searches through all the CD-ROM tracks io locate the most recent VTOC which gives the current m the Liirrciii VTOC, it is possible to give the illusion ihat files have been deleted Tracks cm be grouped into sessions, leading to multisession CD-ROMs Stan- Standard .mdio CD players cannot handle multisession CDs snve they expect a single VTOC at Ihe stait F,ich liack has to be vwilien in a single continuous operation without stopping enough, to deli^ei it on tnnc. ff the files to be copied are spread all over the hard disk, the seek times ma; cause the data stream to the CD-R to dry up and tause a buffer underrun. A buffo underrun results m producing a nice shiny (but some- somewhat expensive) coaster for your dnnks. 01 a 120 mm gold-colored trisbee CD-R software usually offers the option of collecting all the input files into a single con- contiguous 650-MB CD-ROM image prior to burning the CD-R, but this process typi- typically doubles the effective writing lime, requires 650 MB of free disk spaie. dad still does not protect against hard disks that panic and decide to do a thermal recaJibrario,, when they get too hot CD-R makes it possible for individuals and companies to easily copy CD- ROMs (and audio CDs), generally in violation of the publisher's copyright. Several schemes have been devised to make such piracy harder and to make it dif- difficult to read a CD-ROM using anything other than the pubhshei \ software. One of them involves recording all the fi|e lengths on the CD-ROM as multigigabyte. thwarting any attempts io copy the files to hard disk using standard copying software The true lengths are embedded in the

publisher's software or hidden (possibly encrypted) on the CD-ROM in an unexpected place. Another scheme uses intentionally incorrect ECCs in selected sectors, in the expectation that CD copying software will "fix" the errors. The application software checks the ECCS itself, refusing to work if they are correct Using nonstandard gaps between the tracks and other physical ""defects"-are also possibilities CD-Rewritable Although people are used to other write-once media sueh as paper and photo- photographic film, there is a demand for a rewritable CD-ROM. One technology now available is CD-RW (CD-Re Writable), which uses the same size media as CD- R. However, instead of eyanine or pthalocvanine dye, CR-RW uses an alloy of silver, indium, antimony, and tellurium for the recording layer. This alloy has two stable states: crystalline and amorphous, with different reflectivities.

SFC 14 DISKS 313 CD-RW drives use lasers with three different powers At high power, the laser melts the alloy, convening it from the high-reflectivity cr>siallme state to the low-reflectivity amoiphom, state to represent a pit. At medium power, the alloy melts and reforms m its natural crystalline state to become a land again At low power, the state of the material is sensed (for reading), but no phase transition The leason CD-RW has not replaced CD-R is thai the CD-RW blanks are much m,:ie expensive than the CR-R blanks. Also, for applications consisting cr hjtking up hard disks, the iact that once wntien. a CD-R cannot be accidentally erased is a big plus DVD The basic CD/CD-ROM format has been around since 1980. The technology has improved since then, so higher-capacity optical disks are now economically feasible and there u, great demand for them. Hollywood would dearly love to replace analog video tapes by digital disks, since disks have a higher quality, are cheaper to manufacture, last longer, take up less shelf space in video stores, and do not have to be rewound. The consumer electronics companies are looking for a new blockbuster product, and many computer companies want to add mul-multimedia features to (heir software This combination of technology and demand by three immensely rich and powerful industries has led to DVD, originally an acronym for Digital Video Disk, but now officially Digital Versatile Disk DVDs use the same general design as CDs, with 120-mm injection-molded polycarbonate disks containing pits and lands that are illuminated by a laser diode and read by a photodetector What is new is the u«: of 1. Smaller pits @.4 microns versus 0.8 microns for CDs) 2. A tighter spiral @.74 microns between tracks versus | .6 microns for CDs). 3. A red

laser (at 0.65 microns versus 0.78 microns for CDs], Together, these improvements raise the capacity sevenfold, to 4.7-GB A lx DVD drive operates at 1.4 MB/$ec (versus [50 KB/sec for CDs). Unfortunately the switch to the red lasers used in supermarkets means that DVD players will require a second laser or fancy conversion optics to be able to read existing CDs and CD-ROMs, something not all of them may provide. Also, reading CD-Rs and CD-RWs on a DVD drive may not be possible. Is 4.7 GB enough? Maybe. Using MPEG-2 compression (standardized m IS 13346), a 4.7 GB DVD disk can hold 133 minutes of full-screen, full-motion video at high resolution G20x480), as well as soundtracks in up to eight languages and subtitles in 32 more. About 92 percent of all the movies Holly- Hollywood has ever made are under 133 minutes. Nevertheless, some applications such

as multimedia games or reference works may need more, and Hollywood would like to put mulriple movies on the same disk, so four formats have been defined: 1. Single-sided, single-layer D.7 GB). 2. Single-sided, dual-layer (8.5 GB). 3. Double-sided, single-layer (9.4 GB). 4. Double-sided, dual-layer (I? GB) Why so many formats? In a word: politics. Philips and Son> wanted single- sided, dual-layer disks for the high capacity version, but Toshiba and Time Warner wanted double-sided, single-layer disks. Philips and Sony did not think people would be willing to turn the disks over, and Time Warner did not believe putting two layers on one side could be made to work. The compromise, all com- combinations, but the market will determine which ones survive. The dual layering technology has a reflective layer at the bottom, topped with a semirefleaive layer. Depending on where the laser is focused, it bounces off one layer or the other. The lower layer needs slightly larger pits and lands io be read reliably, so its capacity is slightly smaller than the upper layer's. Double-sided disks are made by taking two 0.6-mm single-sided disks and gluing them together back to back. To make the thicknesses of all versions the same, a single-sided disk consists of a 0.6-mm disk bonded to a blank substrate (or perhaps in the future, one consisting of 133 minutes of advertising, m the hope that people wiii be curious as to what is down there). The structure of the double-sided, dual-layer disk is illustrated in Fig. 5-23 Polycarbonate substrate 1 Polycarbonate substrate 2 Fipire 5-23. A double-sided. d,wl layer DVD d,vk DVD was devised by a consortium of |0 consumer electrwncs companies, seven of them Japanese, in close cooperation with the major Hollywood studios (some of which are owned by the Japanese

electronics companies in the consor- consortium). The compater and telecommunications industries were not invited to the

DISKS 315 mcnic and the resulting focus was on using DVD tor movie rental and sales ,hows Far example standard features include real-time skipping of dirty scenes to allow parents to turn <t film raied NC17 into one safe foi toddlers), six-channel ,ound. and support for Pan-and-Scan The latter feature allows' the DVD player to dynamically decide how to crop the left and right edges off movies (whose ividth:height r;itio is 3:2) to fit on current television sets (whose aspect ratio is 4.3). Another item the computer industry probably would not have thought of is an intentional incompatibility between disks intended for the Unued States and disks intended for Europe and yet other standards for other continents. Hollywood demanded this "feature" because new films are always released first m the United States and then shipped to Europe when the videos come out in the United States. The idea was to make sure European video stores could not buy videos in the LJ.S too early, thereby reducing new movies' European theater ytlcs. [f Hollywood had been running the computer industry, we would have had 3.5-mch floppy disks in the United States and 9-l-iti floppy disks in Europe. 5.4.2 Disk Formatting A hard disk consists of a stack of aluminum, alloy, or glass platters 5.25 inch or 3-5 inch in diameter (or even smaller on notebook computers). On each plattet h deposited a Ihin magnetizable metal oxide After manufacturing, there is no information whatsoever on the disk. Before the disk can be used, each platter must receive a low-level formal done by software. The format consists of a series of concentric tracks, each con- containing some number of sectors, with short gaps between the sectors. The formal of a sector is shown in Fig 5-24. Figure 5-24. A disk seaor The preamble starts with a certain bit pattern that allows the hardware to recognize the start of the sector. It also contains the cylinder and sector numbers and some other information. The size of the data portion is determined by the low-level formatting program. Most disks use 512-byte sectors. The ECC field contains redundant information that can he used to recover from read errors. The size and content of this field varies from manufacturer to manufacturer, depending on how must disk space the designer is willing to give up for higher reliability and how complex an ECC code the controller can handle. A |6-byte ECC field is not unusual. Furthermore, all hard disks have some number of spare sectors allocated to be used to replace sectors with a manufacturing defect.

3J6 INPUT/OUTPUT CHAP ^ The position of sector 0 on each track is offset from the previous track when the low-level format is laid down. This offset, called cylinder skew, is d(,ne to improve performance The idea is to allow the disk to read multiple tracks in one continuous operation without losing data The nature of the problem can be seen by looking at Fig. 5-|8(a). Suppose that a request needs 18 sectors starting at sec- sector 0 on the innermost track. Reading the first 16 sectors takes one disk rotation, but a seek is needed to move outward one track to get the 17th sector. By Che itme the head has moved one track, sector 0 has rotated past the head so an entire rota- rotation is needed until it comes by again That problem is eliminated by offsetting the sectors a.s shown in Fig. 5-25. 'igureS.25. An !c>linder-.kev The amount of cylinder skew depends on the drive geometry. For example, a 10.000-rpm. drive rotates in 6 msec. If a irack con wins 300 sectors, a new sector passes under the head every 20 psec. If the track-to-track seek time is 800 [isec, 40 sectors will pass by during the seek, so the cylinder skew should be 40 sectors, rather than the three sectors shown in Fig. 5-25. [t is worth mentioning that switching between heads also takes a finite time, so there i^ head skew as well as cylinder skew, but head skew is not very large. As a result of the low-level formatting, disk capacity js reduced, depending on the sizes of the preamble, intersector gap, and ECC. as well as the number of

SLC. 5.4 DISKS 317 spaie sectors reserved. Otren the formatted capacity is 20% lower than the unfor matted capacity The spare sectors do not count toward the formatted capacity, so all disks of a given type have exactly the same capacity when shipped indepen dent oi how many bad sectors they actually have (if the number of bad sectors exceeds the number of spares, the drive will be rejected and not shipped). There is considerable confusion about disk capacity because some manufac- manufacturers advertised the unformatted capacity to make their drives look larger than they really are For example, consider a drive whose unformatted capacity k 20 x l(f bytes. This might be sold as a 20-GB disk. However, after formatting, perhaps only 2W - 17.2 X |0g bytes are available for data. To add to the confu- confusion, the operating system will probably report this capacity as. 16.0 GB, not 17.2 GB because software consider* I GB to be 230 A,073,741,824) bytes, not I09 A.000.000.000) bytes. To make things worse, in the data communication world, I Gbps means 1,000,000.000 bits/sec because the prefix giga really does mean 109 (a kilometer is 1000 meters, not 1024 meter.1., after all). Only

with memory and disk sizes do kilo, mega, giga, and tera mean 210, 22U, 2^°, and 240, respectively. Formatting also affects performance. If a |0,000-rpm disk has 300 sectors per track of 512 bytes each, it takes 6 msec to read the 153,600 bytes on a track for a data rale of 25.600,000 bytes/sec or 24.4 MB/sec. It is not possible to go taster than this, no matter what kind of interface :s present, even if it a SCSI interface at 80 MBfcec or [60 MB/sec Actually reading continuously at this rate requires a large buffer in the con- controller. Consider, for example, a controller with a one-sector buffer that has been given a command to read two consecutive sectors. After reading the first >.ector from the disk and doing the ECC calculation, the data must be transferred to main memory. While this transfer is taking place, the next sector will fly by the head. When the copy to memory is complete, the controller will have to wait almost an entire rotation time for the second sector to come around again- This problem can be eliminated by numbering the sectors in an interleaved fashion when formatting the disk. In Fig. 5-26(a), we see the usual numbering pattern (ignoring cylinder skew here). In Fig. 5-26(b). we see single interleav- interleaving, which gives the controller some breathing space between consecutive sectors in order to copy the buffer to main memory. If the copying process i!, very dow. the double interleaving of Fig. 5-27(cj may be needed. If the controller has a buffer of only one sector, it does not matter whether the copying from the buffer to main memory ii done by the controllei, the main CPU. or a DMA chip; jr. still takes some time. To avoid the need for inter- interleaving, the controller should be able to buffer an entire track Many modern con- controllers can do this After low-level formatting is completed, the disk is partitioned. Logically, each partition is like a separate disk. On the Pentium and most other computers, sector 0 contains the master boot record, which contains some boot code plus

INPUT/OUT PUT Figure 5-26. < 'ing (b) Single m .] Double n the partition table at the end. The partition table gives the starling sector and size of each partition. On the Pentium, the partition table has room for four partitions. If a‖ of ihem aie for Windows, they will be called C. D:. E:, and F- and treated as Separate drive.. If three of them are for Window*, and one ,s for UNIX, then Win- Windows will call its partitions C:. D\ and E The first CD-ROM will then be F" To be ah|e to boot from the hard diik. one partition must be marked as active m the partition table. The final step in preparing a disk for us,e is to perform a high-level format of each partition (separately) This operation lays

down a boot block, the free storage administration (free list or bitmap], root directory, and yn empty file sys- system. It also puts a code in the partition table entry telling which file system is used in the partition because many operating systems support multiple incompati- incompatible file systems (for historical reasons). At this point the system can be booted. When the power is. turned on. the BIOS runs initially and then reads in the master boot record and jumps to it. This boot program then checks to sec which partition is active. Then it reads in the boot sector from that partition and runs it. The boot sector contains a small program that searches the root directory tor a certain program (either the operating system or a larger bootstrap loader). That program is loaded into memory and executed S.43 Disk Arm Scheduling Algorithms In this section we will look a First, consider how long it takes U determined by three factors; ;s related to disk drive te a disk block. The tir in general. : required is Seek time (the time to move the arm to the proper cylinder). Rotational delay (the time for the proper sector to rotate under tnt head). 3. Actual data transfer t For most disks,, the seek tim seek time can improve syste ; dominates the other two tim n performance .substantially.

SEC ^4 DISKS 319 It the disk driver accepts requests one at a time and carries them out in thai order, that is, KirM-Come, First-Served (FCFS). little can be done to optimize seek tune. However, another strategy is possible when the disk is heavily loaded. It is likely that while the arm is seeking on behalf of one request, other disk requests may be generated by other processes. Many disk drivers maintain a table, indexed by cylinder number, with all the pending requests for each cylinder chained together in a linked list headed by the table entries. Given ihis kind ol data structure, we can improve upon the first-come, lirst- served scheduling algorithm. To see how. consider an imaginary disk with 40 cylinders \ request comes m to read a block on cylinder I I While the seek to cylinder II is m progress, new requests come in for cylinders 1, 36, |6. 34, i), and 12. tn that order. They are entered into the table ot pending requests, with a separate linked list for each cylinder The requests are shown in Fig. 5-27 Initial Pending l>l II I I II M M»l I I M I I II i i i i i«i i*i i n Figure 5-27. Shortest Seek I iw (SSFi Ltisk scheduling ^uruhni When the current request (for cylinder 1 It is finished, ihe disk dn\er has a choice of which request to handle next. Using FCFS. it would go next to cylinder I, then to 36, and so on This algorithm would require arm moiions of |0, 35. 20. 18, 25, and 3. respeclively, tor a lotal of [I 1 cylinders Alternately, it could always handle

the closest lequest next, lo minimize seek time. Given the requests of Fig. 5-27. the .sequence is 12.9. Id. 1. 34. and 36. as shown as the jagged line it the bottom of Fig. 5-27 With this sequence, the arm motions are 1.3,7. |5. 33, and 2, for a total of 61 cylinders This algorithm. Shortest Seek First (SSF). cuts the total arm motion almost in half compaied to FCFS. Unfortunately, SSF has a problem. Suppose more requests keep coming in while thu requests of Fig. 5-27 are being processed. Foi example, it. after going to cylinder 16, a new request for cylinder 8 is present, that request will have priority over cylinder I. If a request for cylinder |3 then comes in, the ami will next go to 13, inslead of I- Wilh a heavily loaded disk, the arm will tend to stay in the mid- middle of the disk most of the time, so requests at either extreme will have to wait until a statistical fluctuation in the load causes there to be no requesis near the

 middle. Requests far from the middle may get poor service. The goals of minimal response time and fairness are in conflict here. Tall buildings also have to deal with this trade-off. The problem of scheduling an elevator in a tall building is similar to that of scheduling a disk arm Requests come in continuously calling the elevator to floors (cylinders) al random The computer running the elevator could easily keep track of the sequence in which customers pushed the call button and service them using FCFS. It could also use SSF However, most elevators use a different algorithm to reconcile the conflicting goals of efficiency and fairness They keep moving in the same direction until there are no more outstanding requests in that direction, then they switch direc- directions. This algorithm, known both in the disk world and the elevator world as the elevator algorithm, requires the software to maintain I bit: the current direction bit. UP or DOWN. When a request finishes, the disk or elevator driver checks the bit. ff it is UP, the arm or cabin is moved to the next highest pending request. If no requests are pending at higher positions, the direction bit is reversed. When the bit is set to DOWN, the move is to the next lowest requested position, if any Figure 5-28 shows the elevator algorithm using the same seven requests as Fig. 5-27, assuming the direction bit was initially UP The order in which the cylinders are serviced is |2. 16, 34, 36, 9, and 1, which yields arm motions or 1.4. IS, 2, 27, and 8, for a total of 60 cylinders. In this case the elevator algorithm is slightly better than SSF, although it is usually worse. One nice property that the elevator algorithm has is that given any collection of requests, the upper bound on the total motion is fixed: it is just twice the number of cylinders. position f

M I I I 1 I I Ix| |x|x| | | |x| I I I I || ITTTTTTrmfflTTI Figure 5-28. The elevator algorithm for scheduling duk requests A slight modification of this algorithm thai has a smaller variance in response times (Teory, 1972) is to always scan in the same direction. When the highest numbered cylinder with a pending request has been serviced, the arm goes to the lowest-numbered cylinder with a pending request and then continues moving in an

SEC 5 4 DISKS 321 upward direction. In effect, [he lowest-numbered cylinder is though! of js being just above the highest-numbered cylinder. Some disk controllers prov,de a way for the software to inspect the current sector number under the head. With such a controller, another optimization is possible If two or more requests for the same cylinder are pending, the driver c;in issue a request for the sector that will past, under the head next Note that when multiple tracks are present m a cylinder, consecutive requests can be for different tracks with no penalty. The controller can select any of its heads instantaneously, because head selection involves neither arm motion nor rotational delay If the disk has the property that the seek time is much faster than the rota- rotational delay, then a different optimization strategy should be used Pending requests should be sorted by sector number, and as a soon as ihe next sector is about to pass under the head, the arm should be zipped over to the right track to read or write it. With d modem haid disk, the seek and rotational delays so dominate perfor- performance that reading one or two sectors at a time is very inefficient. For this rea- reason, many disk controllers always read and cache multiple sectors, even when only one is requested. Typically any request to read a sector will cause thai sector and much or all Ihe rest of the currem track to be read, depending upon how much space is available in the controller's cache memory The disk described in Fig. 5-17 has a 2-MB or 4-MB cache, for example. The use of the cache 11 determined dynamically by the controller. In its simplest mode, the cache is divided into two sections, one for reads and one for writes If a subsequent read can be satisfied out of the controller's cache, it can return the requested daia immediately It is worth noting that the disk controller's cache is completely independent of the operating system's cache. The controller's cache usually holds blocks' that have noi actually been requested, but which were convenient the iead because they jusl happened to pass under the head as a side effect of some other read. In contrast, any cache maintained by the operating system will consist of blocks that were explicitly read and which the operating system

thinks might be needed again in the near future (e.g., a disk block holding a directory block). When several drives are present on the same controller, the operating should maintain a pending request table for each drive separaiely. Whenever any drive is idle, a seek should be issued to move its arm to the cylinder where it will be needed next (assuming the controller allows overlapped seeks) When the currem transfer finishes, a check can be made to see if any drives are positioned on the coireci cylinder. It' one i>r more are. the next transfer can be started on a drive that is already on the right cylinder- If none of the arras is in ihe righl place, the driver should issue a new seek on the drive that just completed a tranter and wait Until the next interrupt to see which arm gets to us destinaiion first It is important to realize ihat aJl of die above disk scheduling algorithms tacitly assume that the real disk geometry is the same as the virtual geometry. If it is not, then scheduling disk requests makes no sense because the operating system

322 INPlT/OI'TPUr C|)AP. 5 cannot really tell whether cylinder 40 or cylinder 2(H is closer to cylinder 39 On the other hand, if the disk controller can accept multiple outstanding requests, it can use these scheduling algorithms internally In that case, the algorithms are still valid, but one level down, inside the control lei. 5.4.4 Error Handling Disk manufuctuieis die constantly pushing the hunts of the technology by increasing linear bit densities A truck midway out on a 5 25-inch disk has a cir- circumference ot about 3(K> mm. If the track holds 300 sectors ot 512 bytes, the linear recording density ma> be aboui 5001) bils/mm taking into account t|ie fact that some space is lost to pieambles. ECCs. and mtersector gaps Recording 5@) bus/mm icuuires an extremely uniform substrate and a vei"y fine oxide coating Unfortunately, n is not possible to manufacture ,i disk to such specifications without defects As soon as manufacturing technology has improved lo the poini where it is possible to operate flawlessly at suc|i densities", disk designers will go to higher densities to .nciease the capacity. Doing so will probably lemtroduce defects. Manufacturing defects introduce bad sectors, that is. sectors that do not correctly read back the value just written to them. If the defect is very small, say, only a tew bits, it is possible to u« the bad sector and just let the ECC correct the errors every time If the defect is bigger, the error cannot be masked There are two general approaches, to bad blocks deal with them in the con- controller or deal with them in ihe operating system in the former approach, before the disk is shipped from the factory, it is tested and a list of bad sectors

is written onto the disk For each bad sector, one of the spares is substituted |or it. There are iwo ways, to do this substitution In Fig. 5-29(a). we .see a single disk track with 30 data sectors and twt> spares. Sector 7 is defective. What the controller can do is remap one of the spares as sector 7 as shown in Fig. 5-29|b). The other way is to shift all the sectors up one. as shown in Fig. 5-29(c). In both cases the controller has lo know which sector is which It can keep track of thi*, information through internal tables (one per track) or by iewritmg the piedmblei to give the remapped sector numbers, if the preambles are rewritten, the method of Fig 5-2%) is more work (because 23 preambles must be iewritteu) but ulti- Errors can also develop during normal operation after the drive has been installed. The first line of defense upon getting an error that the ECC cannot han- handle is to just try the read again. Some read errors are transient, that is. are caused by specks of dusl under the head and will jro away on a second attempt. If the controller notices that it is getting repealed errors on a ceitam sector, it can switch lo a spare before- the sector has died completely In this way, no data aie lost and [he operanng system and user do noi even notice the problem. Usually, the

b.id sector, (t) Shifting all the seciors to bypass ihc bad one method of Fig. 5-29(b) has to be used since the other sectors might now contain data. Using the method of Fig 5-29(c) would require nut only rewriting the preambles, but copying all the data as well Earlier we said there were two general approaches to handling cnors: handle them in the controller or in the operating system, [t the controller does not have the capability to transparently remap sectors as we have discussed, the operating system must do the same thing in software This means that it must first acquire a list of bad sectors, either by reading them from the disk, or simply testing the entire disk itself. Once it knows which sectors are bad, it can build remapping tables. If the operating system wants to use the approach of Fig 5-29@). it must shift the data in sectors 7 through 29 up one sector. If the operaiing system k handling the remapping, it musi make sure that bad sectors do not occur in any files and also do not occur in the free list or bitmap One way to do this is to create a secret file consisting of all the bad sectors, ft th(, file is not entered into the file system, users will not accidentally read it (or worse yet, tree it). However, there is still another problem: backups If the dis,k is backed up file by file, it is important that the backup utility not try to copy the bad block file To prevent this, the operating system has to hide the bad block file so well that even ;i

backup utility cannot find it. If the disk hs backed up sector by sector rather than file by file, it will be difficult, if not impossible, to prevent read errors during backup. The only hope is that the backup program has enough spiarts to give up after 10 failed reads and continue with the next sectoi. Bad sectors are not the only source of errors. Seek errors caused by mechani- mechanical problems in the arm also occur. The controller keeps track of the arm position internally. To perform a seek, it issues a series of pulses to the arm motor, one pulse per cylinder, to niove the arm to the new cylinder When the arm gets to its destination, the controller reads the actual cylinder number from the preamble of the next sector. 1 f the arm is in the wrong place, a seek error has occurred.

324 INPUT/OUTPl.T CHAP 5 Mom hard disk compilers correct seek errors automatically, bat most floppy controllers (including Ihe Pentium's) just set an erroi bit and leave the resi to the driver The driver handles this error by issuing a recalibrate command, to move the arm as tar out as it will go and reset the controller's internal idea of the current cylinder to 0. Usually this solves the problem. If it does not, the drive must be repaired As we have seen, the controller is really a specialized little computer, com- complete with software, variables, buffers, and occasionsl|y, bugs. Sometimes an unusual sequence of events such as an interrupt on one drive occurring simultane- simultaneously with a recalibrate command for another drive will tng^ei j bug and ciuse the ciintrollci to go into a loop or lose track of what it was doing. Controller designers usually plan for the worst and provide a pin on the chip which, when assigned, forces the controller to forget whatever it was doing and lesel itsel! [f all else tails, the disk driver can set. a bit io invoke this signal and reset the controller If that does not help, sill the driver can do is print a message and give up Recalibrating a disk makes a funny noise but otherwise normally is not dis- disturbing. However, there is one situation where recalibration is' a serious problem systems with real-time constraints. When a video is being played <iit .t hard disk, or files from a hard disk are being bucned onto a. CD-ROM, it is essential that the bits arrive from the hard disk at a uniform rate Undei these circumstances, recallbrations insert gaps into the bit stream and are therefore unacceptable Spe- Special drives, called AV disks (Audio Visual disks), which never recalibrate arc available for such applications. 5.4.5 Stable Storage As we have seen, disks sometimes make errors Good sectors can suddenly become bad sectors. Whole drives can die

unexpectedly RAIDS protect against a few seciors going bad »r even a drive falling out However, they do not protect against wiite errors laying down bad data in l[ie first place They ,tko do not pro- protect against crashes during writes corropting the original data without replacing them by newer data. For some applications, it is essential that data never be lost or corrupted, even in the face of disk and CPU errors Ideally, <i disk should simply work all the time with no errors. Unfortunately, that is not achievable What is achievable is a disk subsystem that has the following property, when a write is issued to it, the disk either correctly writes the data or it does nothing, leaving the existing data intact Such as system is called stable storage and is implemented in software (Lampoon and Sturgis. 1979). Below we will describe a slight variant of the original idea. Before describing the algorithm, it is important to havt- a clear model of the possioic errors ine model assumes that when a disK writes a block (one or moro sectors), either the write is correct or it is incorrect and this error can be detected on a subsequent read by examining the values of the ECC fields. In principle.

SEf 5 4 D[SKS 325 guaranteed error detection is never possible because with a, say, '6-byte tCC field guarding a 5 12-byte sector, there are 24II% data values and only 21 ECC values Thus if a block is. garbled during writing bul the ECC is net. there arc bil- billions upon billions of incorrect combinations that yield the same ECC It any of them occur, the error will not be detected. On the whole, the probability of lan- dom data having the proper 16-byie ECC is about 244. which is small enough thai we will tall it zero, even though it is really not The model also assumes that a correctly written sector can spontaneously go bad and become unreadable. However, the assumption is that such events are so rare that having the same sector go bad on a second (independent) drive during a reasonable time interval (e g.. I day) is small enough to ignore. disk write in progress at the moment of failure alsci stops., leading to incorrect data in one sector and an incorrect ECC that can later be detected. Under all these conditions, stable storage can be made \(Hty reliable in the sense of writes either working correctly or leaving the old data in place. Ot course, it does not protect against physical disasleis, such as an earthquake happening and the computer tai- tailing 100 meters into a fissure and landing in a pool of boiling magma H is tough to recover froin this condition in software. Stable storage uses a pair of identical disks wuh the corresponding blocks working together to form one error-tree block, [n the absence of errors, ihe

corresponding blocks on both drives are the same, tither one can be read to ^et the same result. To achTeve this goal, the following three operations arc defined" 1 Stable writes. A stable write consists of first writing the block on drive I. then reading il back to verify that it was written correctly. It it was not written correctly, the write and reread arc done again up to n time^; until they work. After n consecutive failures, the block is remapped onto a spare and the operation repeated until ir succeeds, no matter how many spares have to be tried After the write to drive 1 has succeeded, ihe corresponding block on dnve 2 is written and reread, repeatedly if need be, until it, too. finally succeeds In the absence of CPU crashes, when a stable write completes, the block has correctly been written onto both drives and verified on both of them. 2 Stable reads. A stable read first reads the block" from dnve I jf this yields an incorrect ECC, the read is tried again, up to n times. It all of these give bad ECCs. the corresponding block is read from drive 2. Given the tact that a successful stable write leaves two good copies of the block behind, and oui assumption that the probability ot the same block spontaneously going bad on both drives in a reason- reasonable time inteival is negligible, ;i stable read always succeeds.

326 INPUT/OUTPUT CHAP 5 3. Crash recovery. After a crash, a recovery program scans both di».ks comparing corresponding block* If a pair of blocks are both good and the same, nothing is done If one of them has an ECC error, the had block is overwritten with the corresponding good block- It a pair of blocks are both good but different, the block from drive | is writ- written onto drive 2 In the absence of CPU crashes, this scheme always works because siable writes always write two valid copies of every block and spontaneous errors are assumed never to occur on both corresponding blocks at the same time What about in the presence ot CPU crashes during stable writes' It depends on pre- precisely when the crash occur. There are five possibilities, as depicted in Fig. 5-30. ECC Figure 5-30. Analyse of the inHueiire of craves on vUble wnles. in Fig. 5-30(a), the CPU crash happens before either copy of the block is wnt- len. During recovery, neither wi[] be changed and the old value will continue to cxisi, which is allowed In Fig. 5-30(b), the CPU crashes during the write to drive I. destroying the contents of the block. However the recovery program delects (his emir and restores the block on drive I from drive 2. Thus the effect of the cni-sh js wiped out and the old state is fully restored. In Fig. 5-3O(c), the CPU crash happens after drive I is written but

before drive 2 is written. The point of no return has beer) passed here the recovery pro- program copies the block from drive 1 to drive 2 The write succeeds. Fig. 5-30(d) is like Fig. 5-3O(b): during recove-iy. the good block overwrites the bad block. Again, the final value of both Mocks is the new one. Finally, in Fig. 5-3O(e) the recovery program sees that both blocks are the same, so neither is changed and the write succeeds here too Various optimizations and improvements are possible to this scheme For starters', comparing ail the blocks pairwise after a crash is- doable, but expensive. A huge improvement is to keep track of which block was being written during a

puters have a small amount of nonvolatile RAM which is a special CMOS memory powered by a lithium battery. Such batteries last fur years, possibly even the whole life of ihe computer. Unlike mam memory, which is last after a crash, nonvolatile RAM is not lost after a crash. The time of day is normally kept here (and incremented by a Special circuit), which is why computers still know what time it is even after having been unplugged. Suppose thai a tew bytes of nonvolatile RAM are available for operating sys- system purposes. The stable write can put the number of the block it is about to update in nonvolatile RAM befoie starling the write After successfully complet- completing the stable wriie. the block number in nonvolatile RAM is overwritten with an invalid block number, tor example. -1. Under these conditions, after a Crash the recovery program can check the nonvolatile RAM to see it a stable write hap- happened lo be in progress during the crash, and it so, which block was being written when the crashed happened The two copies of the block can then be checked for correctness and consistency. If nonvolatile RAM is not available, it can be sitnulaied as follows At the start of a stable write, a fixed disk block on drive I is overwritten with the number of the block to be stably written. This block is then read back to verify it. After getting it correct, the corresponding block on drive 2 is written and verified. When the stable write completes correctly, both blocks arc overwritten with an invalid block number and verified. Again here, after a crash it is easy to deter- determine whether or not a stable write was in progress during the crash. Of couise. this technique requires eight extra disk operations to write a stable block, so it should be used exceedingly sparingly. One last point is worth making We assumed that only one spontaneous decay of a good block to a bad block happens per block pair per day. If enough days go by, the i>ther one might go bad too. Therefore, once a day a complete scan of both disks must be

done repairing any damage. That way, every morning both disks are always identical. Even if both blocks in a pair go bad within a period of a tew days, all errors are repaired correctly. 5.5 CLOCKS Clocks (also called timers) are essential to the operation of any multipro- grammed system for a variety of reasons. They maintain the time of day and prevent one process from monopolizing the CPU. among other things. The clock software can take the form of a device driver, even though a clock is neither a block device, like a disk, nor a character device, like a mouse. Our examination of clocks will follow the same pattern as in the previous section: first a took at clock hardware and then a look at the clock software.

328 INPUT/OUTPUT CHAP 5 S.S.1 Clock Hardware Two types ot clocks are commonly used in computers, and both are quite dif- different from the clocks and watches, used by people. The simpler clocks are tied to the 1 |0- or 220-volt power line and cause an interrupt on every voltage cycle, at 50 or 60 Hz These clocks used to dominate, but are rare nowadays. The other kind of clock is built out of three components: a crystal oscillator a counter, and a holding register, as shown in Fig. 5-31. When a piece ol quartz crystal k properly cut and mounted under tension, it can be mado to generate a periodic Signal of very high accuracy, typically in the range of several hundred megahertz, depending on the crystal chosen. Using electronics, this base signai can be multiplied by a small integer to get frequencies up to 1000 MHz or even more At least one such circuit is usually tound in any coniputer, providing a syn-chromzing signal to the computer's various circuits This signal is fed into the counter to make it count down to zero. When the counter gets to zero, it causes a CPU interrupt. HDh nummmEn folding register is used to Programmable clocks typically have mode, when the clock is, started, it copie counter and then decrements the counter ral modes of operation In one-shot value ot the holding register into the ich pulse from the crystal. When the again by the software. |n square-wave mode, after getting to /.am and causing tlie interrupt, the holding register is automatically copied into the counter, .ind the whole process is repeated again indefinitely Thes-e periodic interrupts are called clock ticks. The advaniage of the programmable clock is that its interrupt frequency can be controlled by software If a 500-MHz crystal is used, then the counter is pulsed every 2 nsec. With (unsigned) 32-bit rcgisiers. interrupts can be programmed to occin at intervals from 2 nsec to 8-6 sec. Programmable clock chips usually con- contain two or three independently programmable

clocks and have many other options as well (e g.. counting up instead of down, interrupts disabled, and more)

SbC 5 5 CLOCKS 329 turned ofi, most computers have1 a battery-powered backup clock, .implemented with the kind ot' low-power circuitry used in digital watches. The battery cluck can be read at startup, [f ihe backup clock k not present, the software may ask ihe user for the cunent date and time There is also a standard way for a networked system tD get the current time fruui ;i remote host |n am case the time is then translated into the number of clock ticks since |2 am UTC (Universal Coordi- Coordinated Time) (formerly known as Greenwich Mean Time) on Jan I. 1970 as UNIX does, or since some other benchmark The origin of time for Windows is Jan I. 1980. At every clock tick, the real time is incremented by one count. Usu- Usually utility programs are provided io manually set tlie system clock and the backup clock and to synchronize the iwo clocks. S.S.2 Clock Software All the clock hardware does is generate interrupts at. known intervals Every- Everything else involving tune must be done by the software, the clock driver. 'Ihe exact, duties of the clock driver vary among operating systems, hut usually include most of the following: 1 Maintaining the tjmc of day. 2. Preventing processes from running longn than they are allowed to. 3. Accounting for CPU Usaye. 4. Handling the alarm system call nude by user piocesses 5. Providing watchdog tuners for parts of the system itself 6. Doing profiling, monitoring, and statistics gathering The first clock function, maintaining the lime ot day (also called the real timel is- not difficult. It just requires incrementing a countei at each clock tick, as mentioned before The only thing to watch out for is- the number ot bits in the time-of-day counter- With a clock rate o) 60 H?. ,, ^2-bit counter will overflow in just over 2 years. Clearly the system cannot store the real nmc as tlie nutnbei of ticks since Jan. I. 1970 in 32 bits Three approaches can be taken to solve this problem. The first way is to use a 64-bit counter, although doing so makes maintaining the counter inoie expensive since it has to be done many times a second The second way is to maintain the time of day in seconds, rather than in ticks, using a subsidiaiy countei to count ticks until a whole second has been accumulated. Because 2'2 seconds is more than C6 years, this method will work until the twenty-second century. The third approach is to count in ticks, but to do that relative to the time the system was booted, rather than relative to a fixed external moment When the

330 INPUT/01 ITU CHAP *» cal- backup clock is read or file user types in the real time, ihe system boo eulated irom the current time-of-day value and stored in memory m any con- convenient form. Later, when the time of day is requested, the stored tune of day is added to the counter to get the current time of day All three approaches arc shown in Fig 5-32. Figure 5-32. Three ways li> nHinuiin tht: time ol dj\ The second clock function is preventing processes from running too long. Whenever a process k started, the scheduler initializes a counter to the \alue or that process quantum in cltKk ticks. At cverv clock intC'iTupt. the clock driver decrements the quantum counter hy | When u gets to zero, the clock iliiver e.ill.s ihe scheduler to se(up ;inot)icr process. The third clock function is doing CPU .iccounting. The most accurate wa_\ to do it is to start a second timer, distinct from the main system timer, whenever a process is started When th.it process, is stopped, the timer can be read out to tell how long the process has run. To do things right, the second timer should be saved when itn interrupt occurs and restored afterward pointer to the process table entry tor the currently running process in a global vari- variable At every clock tick, a field in the cunent process' entry is incremented In this way, every clock tick is "charged"' to the process running at the time of the tiek. A minor problem with (his siraiegy is thai if many interrupts occur during a process' run. it is Still charged for a full tick, evon though it did not get much work done. Properly accounting for the CPU during interrupts is too expensive and is rarely done. in many systems, j process can request that ths; operating system give it a warning <*fter a certa/n interval. The waniifig is usually d signal, interrupt, mes- message, or something similar. One application requiring such warnings' is network- networking, in which a packet not acknowledged within a certain iime interval must be retransmilted. Another application is computer-aided instruction, where a student not providing a response within a certain time is told ihe answer. If the clock driver had enough clocks. it could set a separate clock for each request. This nol being the case, it must simulate multiple virtual clocks wtin a single physical clock. One way is to maintain a table in which the signal time for

ShX 5 5 CLOCKS 331 all pending timer., is kept, as well as a variable giving the time ot tne next one. Whenever (he time ol day is updated, the driver checks »> see if the closest signal has occurred. If it has. the table is searched tor the next one to occur It many signals are expected, it is more c-fficient to simulate multiple el odes by chaining all the pending clock requests together, sorted on

time, in a linked list, as shown m Fig. 5-33. Each entry on the list tells how many clock ticks following the previous (.ne lo wait before causing a signal. In this example, signals are pend- pending for 4203. 4207. 4211. 42 15. and 4216. Figure 5-33. •= In Fig. 5-33. the next interrupt occurs in 1 ticks. On each tick. Next signal is decremented. When it gets to 0, the signal corresponding to the first item on ihe list is caused, and that item is removed from the list. Then NexT signal is set to ihe value in the entry now at [he head of the list, in this example. 4 Note thai during a clock interrupt, the clock driver has several things to do— increment the real time, decrement the quantum and check for 0. do CPU account- accounting, and decrement the alarm counter. However, each of these operaiions has been carefully arranged to be very fast because they have to be repeated many times a second. Parts of the operating system also need to set timers. These are called watch- watchdog timers. For example, floppy disk<; do not rotate when not m use, to avoid wear and tear on the medium and disk head When data are needed from a floppy disk, the motor must first be started. Only when the floppy disk is rotating at (all Speed can I/O begin. When a process attempts to read from an idle floppy disk, ihe floppy disk driver starts the motor and then sets a watchdog timer to cause an interrupt after a sufficiently long time interval (because there is no up-to-speed mtermpt from ihe floppy disk itself)- The mechanism used by the clock driver to handle watchdog timers is the same as for user signals. The <,nly difference is that when a tunei goes oft, instead of causing a signal, the clock driver calls a procedure supplied by the caller. The procedure is part of the caller's code. The called procedure can do whatever is necessary, even causing an interrupt, although within ihe kernel interrupts are often inconvenient and signals do not exist. That is why the watchdog mechanism is provided It is worth nothing that the watchdog mechanism works only when the clock driver and the procedure to be called are in the same address space.

 The last thing in our list i<- profiling Some operating systems provide a mechanism by which d user program can have the system build up a histogram o1 its program counter, so it can see where it is spending its time-. When profiling ■<; a possibility, at every tick the driver checks to see if the current process is being profiled, and if so. computer ihe bin number (a range of addresses) corresponding to the current program counter. It then increments that bin by one. This mechan- Tsm can also be used in profile the system itself S.S.3 Soft Timers Most computers have a second

programmable clock that can be set to cause linici interrupts at whatever rate a program needs- This tniier is in addition to the mam system timer whose functions were described above As long as the inter- specific purposes. The trouble arrives when the frequency of the application- specific timer is very high. Below we will briefly describe a software-based timer scheme thai works well under many eircuiiM.inces, even at high frequencies. The idea is due to Aron and Druschel A999) For more details, please sec their paper. Generally, there arc two ways to manage I/O: interrupts and polling Inter- Interrupts have low latency, that is. they happen immediately after the event itself with little or no delay On (he other hand, with modem CPCX interrupt*, have a sub- substantial overhead due to the need for context switching and their influence on the pipeline. TLB and cache The alternative to interrupts is to have the application poll for the event expected itself. Doing tills avoids interrupts, but there iruiv be substantial latency because an event may happen directly after a poll, in which case jt wjits almost a whole polling interval On th<i average, the latency is half the polling interval For some applications, neither the overhead ot interrupts nor the latency ot polling is acceptable. Consider, for example, a high-performance network such as Gigabit Ethernet. This network is capable ot accepting oi delivering a full-sue packe! every |2 usec. To run at optimal performance on output, one packet should be sent every |2 u^ec. One way to achieve this rate is to have ihc completion of a pac-kct transnns- sion cause an interrupt or to set the second tuner to interrupt every 12 usei. The problem is that this lnienupt has been meJsuied to take 4.45 u_sec on .i 31H MHz Pentium II (Aron and Druschel, |<?99) This overhead is barely better than that of computers in the 1970s On most minicomputers, for example, an interrupt took four bus cycles: to stack the program counter and PSW and to load a new program counter and PSW. Nowadays dealing with the pipeline, MMU. TLB, and cache, add a great deal to the overhead. These effects are likely to get woise rather than better in tune, thus canceling out faster clock rates Soft timers avoid interrupts. Instead, whenever the kernel is running for some other reason, jusl before it return^ to user mode it checks the real time clock

SEC. 5.5 CLOCKS 333 to see if a soft timer has expired If the timer has expired, the scheduled event (e.g., packet transmission or checking for an incoming packei) is performed, with no need to switch into kernel mode since the system is already there After t|te work has been performed, the soft tinier is

reset to go off again. All that has to be done k copy the current clock value to the- timer and add the timeout interval to it. Snti timers stand or fall with tlie rate at which kernel entries, are made for dthcr reasons These reasons include 1. Syuetn calls. 2 TLB misses 3. Page faults. 4. I/O interrupts 5. The CPU going idle To see how often thesi.' events happen, AiOti and Druschcl made measurements with several CPU loads, including a fully-loaded Web server, a Web server with a compute-hound backgiound job, playing real-time audio Iroin the Internet, and recompiling the UNIX kernel. The average entry late into the kernel varied from 2 u_t,ee to 18 u.sec. w1th about halt of these entries being s>stem ca)U. Thus to a first-order dpproximation. having a soft timer go off every 12 u_sec is doable, albeit with an occasional missed deadline For applications like sending paekels or polling tor incoming packets being 10 u_sec late from time fo time is better than having interrupts eat up 35* of the CPU. Of course, there will be periods when there are no system calls. TLB misses, or page faults, in which case no soft timers will go off. To put an upper bound i>n [f the application can live with uniy lO(X) packets/s-ec fur occasional intervals, then the combinaiion ot soft timers anil <t low-frequency hardware timer ma> he better than either pure inlerrupt-dnven I/O or pure polling 5.6 CHARACTER-ORIENTED TERMINALS tvery general~pmpo^e computer hds at least i^ne keyboard and one display (monitor or flat seieen) used to communicate with it. Although the keyboard and together, On mainframes, there are frequently many remote users, each with a device containing a keyboard and an attached display. These devices have hktdit-cally been called terminals We will continue lo use that term, even vvhen dN-cu«jng personal computers (mostly for lack at a better term).

334 INPUT/OUTPUT CHAP 5 Terminals coinc in many forms Three of the types most commonly encoun- encountered in practice nowadays are [. Standalone terminals with RS-232 serial interfaces tor use on mainframes. 2. Personal computer displays with graphical user inteifaccs 3 Network terminals Each type of terminal has its own ecologic will describe each of these types in turn. Fche. In the following sections 5.6.1 RS-232 Terminal Hardware RS-232 terminals are hardware devices containing both a keyboaid and a display and which communicate using a serial interface, one bit ill a time (see Fig- 5-34) These terminals use a 9-pin or 25-pin connector, of which one pin is used for transmitting data, one pin is for receiving data, and one pin is ground. The other pins are for various control functions, most of which are noi used. Lines

m which character are sent one bit at a tune (a,s opposed to 8 bits in paral- parallel the way primers are inierfaced to PCs.) are called serial lines. All modems' also use this interface. On UNIX, serial lines have names like M&v/ml and /dev/riy-2 On Windows ihey have names like COM] and COM2. To send a character over a serial line iA an RS-232 terminal or modem, the computer must transmit it I bit at a time, prefixed by ;i start bit. and followed by I or 2 stop bits to delimit the character. A parity bit which provides rudimentary error detection may ajso be inserted preceding the stop hits, although this is com- commonly required only for communication with mainframe sysk-ms Compute i RS-232 t< remote user t< are still commonly used in the mainframe world to allow a nutate with the mainframe, sometimes using a modem and a

ShC. 5.6 CHARACTER-ORIENTED TERMINALS 335 when Ihey are replaced by personal computers, thr PCs often simply emulate the old RS-232 terminals lo avoid having to change the mainframe software Thee terminals also used to dominate the minicomputer world. A great deal ol .software tor systems that grew up in this period are based on these terminals. For example, all UNIX systems support this kind of device However, even more important, many current UN[X systems (and other sys- systems) provide the option of creating a window consisting of some number of lines of text. Many programmers work almost exclusively in text mode in such win- windows, even on personal computers or high-end workstations These windows usu- usually simulate some RS-232 terminal (or the ANSI standard for this type of termi- terminal) so they can run the large existing software base that was written for .such ter- terminals. In ihe course ol the years, this software, such as the vi and emacs editors, ha.s become completely debugged and extremely stable, properties programmers value highly The keyboard and terminal software tor these terminal emulation windows is the same as for Ihe real terminals. Since these terminal emulators are in widespread use. Ihe software is stiil important, so we will describe it in the fol- following two sections'- RS-232 terminals aie character oriented What this means ts that the screen or window displays a certain number of lines of" text, each of a maximum size A lypical size is 25 lines of 80 characters each. While a few special characters .ire sometimes ;>upported, ihese terminals (and the emulators) are basically text only Since both computers and terminals work internally with whole characters but musl communicate over a serial line a bit at a time, chips have

been developed to do the character-to-seriiil and serial-to-character conversions. They are called UARTs (Universal Asynchronous Receiver Transmitters)- UARTs are attached to the computer by plugging RS-232 interface cards into the but, as illustrated in Fig. 5-34. On many computers, one or two serial ports are built into the parent- board. To display a character, the terminal driver writes the character to the interface card, where it is buffered and then shifted out over the serial line one bit at a time by the UART. For example, for an analog modem operating at 56.000 bps, ,t lakes just over 179 n_sec to send a character. As a result of this slow transmission rate, the driver generally outputs a character to the RS-232 card and blocks, waking for the interrupt generated by the interface when the character has been transmitted and the UART is able to accept another character. The UART can send and receive characters simultaneously. An interrupt is also generated when a character is received, and usually a small number of input characters can be buffered. The terminal driver must check a register when an interrupt is received to determine the cause of the interrupt. Some interface cards have a CPU and memory and can handfe multiple lines, taking over much of the I/O load from the mam CPU RS-232 terminals can be subdivided into (hree categories. The simplest ones are hardcopy (i.e.. printing) terminals. Characters typed on the keyboard arc

 transmitted to the computei. Characters sent by the unnputer an- printed on the paper These terminals are obsolete and rarely sci-n any more except as low-end Dumb CRT terminals work the same wa>. onk with a screen instead ol papt;r. fhese are often called "glass Itys" because they are functionally the same as pan> ihat pionceied m the computer lermitial business: '■tty" has come to mean any terminal) Glast, ny> arc also obsolete Intel hgent CRT terminals are in laci miniature, specialized computers 1 tic ■ have a CPU and memory and contain software, usually in ROM. From the operat- operating system's viewpoint, the main difference between a glass tty and an intelligent terminal is that the latter understands certain escape sequences. For example, by sending the ASCII F.SC character (OxlB), followed bj various other characters, it may be possible to move the cursor to any position on the screen, insert text in the middle of the screen, and so forth. Intelligent terminals arc the ones used in main- mainframe systems, and are the ones emulated by other operating systems It is their software that we will discuss below. 5.6.2 Input Software The keyboard and display are almost

independent devices, so we will treat them separately here. They ure not quite independent, however, since typed char- characters generally are displayed on the screen The basic job ot Hie keyboard driver ts to collect input hvm the keybo.ird and pass it to user programs when they read from the ici nnnal Two possible philoso- philosophies can be adopted for the driver. In the fust one, the driver's job is |ust to gets a raw sequence of ASCII codes (Giving u^ei programs the kes numbers is too primitive, as well as being highly machine dependent.} This philosophy is well suited to the nei!ds ol sophisticated screen editors such as t?tti(tt a , which 31 low tnc user to bind an at bitrai y action to anv chai ac ter or sequence of characters It does, howevci mean that if the usei ivpes d^tc instead of date and then corrects the error by typing three backspaces and ate. followed by a carriage return, the user program will be biveii til!] l ASCII code** tvped. js follows- Not ail programs want this much detail Often the\ just want the cclnected input, not the exact sequence of how it was produced This observation leads to the second philosophy: the driver handles all the mtnilme editing, and just delivers corrected lines to the u.ser programs. The first philosophy is characier-onented: the second one is line oriented. Originarllv tht?y were referred to as rsw mode «ind cooked mode, respectively. The POS1X standard uses the less-picturesque term

SEC. ^ 6 CHARACTER-ORIENTED TERMINALS 337 canonical mode to describe line-oriented mode Noncanonical mode is equiv- equivalent to raw mode, although many details of terminal behavior c.in be changed. POSIX-compatible systems provide several library functions that support selecting The first task of the keyboard driver is to collect characters If every ke>-stroke causes an interrupt, the driver can acquire the character during the inter-interrupt ll interrupts are turned into message by the low-|evel software, it is possi- possible to put the newly acquired character in the message. Alternatively, it can be put in a small buffei in niemorv and the message used to tell the driver that some- something has arrived The latter approach is actually saU-r if .1 message can be- sent only to a waiting process and there is some chance that the Keyboard driver might still be busy wit[i the previous character. [I the terminal is in canonical (cooked) mode, ch.iuicters must be stored until an entire line has been accumulated, because the user may subsequently decide to erase part of it Even if the terminal is in law mode, the program rna> not yet have requested input, so the characters must be buffered to allow type ahead (System designers who do not allow users to type tai

ahead ought to be tarred and feathered, or worse yet, be forced to use their own system) Two approaches to character buffering are common. In the first one, the driver contains a central pool ot buffers, each buffer holding perhaps 10 charac- other items, a pointer to the chain of buffers foi tnput collected from that terminal As more characters are typed, more buffers are acquired and hung on the chain When the characters are passed to a user program, the bulters aic removed ,ind put back in (he central pool. The other approach 1*, to do the buffering directly in the terminal dat;i struc- structure itself, wiih no central pool of buffers Siiice it is common ("or users to type a command thai will take a little while (say, recompiling and linking a lajge binary program) and then type a tew lines ahead, to be safe the driver should allocate something like 200 characters per terminal [n a large-scale timesharing system with 100 terminals, allocating 20K all the time for type ahead is clearly overkill, so a central buffer pool with space tor perhaps 5K is probably enough. On the other hand, a dedicated bujfer per ternnn.il makes iho drivei simpler (nc> linked list management) and is to be preferred on personal computers with nnly one key-keyboard. Figure 5-35 shows the difference between these two methods Although the keyboard and display are logically separate devices, many users have grown accustomed to .seeing the eharacturs they have jtist typed uppear on the screen. Some (older) terminals oblige by automatically displaying (in hardware) whatever has just been typed, which is not only y nuisance when pass- passwords are being entered but greatly limits the flexibility of sophisticated editors and other programs. Fortunately, with mos! terminals, nothing is automatically displayed when a key is struck. It is entirely up to the software in ihe computer to display ihe character, if desired. This process is called echoing

buffer pool ib) Figure 5-35. U) CentrJl bulfer pool ib! Deduced bufler lor oOi termni.il Echoing is complicated by the fact thai a program may be writing to trie screen while the user is typing. At the very least, the keyboard driver has to fig- figure out where to put the new input without it being overwrnten by program out- output. Echoing also gets complicated when more lhan HO characters have to be displayed on a screen wiih 80-character line1; (or some other number). Depending on the application, wrapping around to the next line may be appropriate. Some drivers just truncate tines to SO characters by throwing awjy all characters beyond column 80- Another problem is tab handling. It is usually up to the driver to compute where the cursor is currently

locaied, taking into account both output from pro- programs and owput from echoing, and compute- the proper number of spaL-es to be echoed. Now we come to the problem ot device equivalence. Logically, at the end of a line of text, one wants n carriage return, to move the cursor back n> column I. and a linefeed, to advance to the next line. Requiring users \o lype both .it the end of each line would not sell well (although some terminals have a key which gen- generates, both, with a 50 percent chance of doing so in tbe order that the software wants them) It h Up to the driver to convert whatever conies in to the standard internal tormat used by the operating <>y^tem If the standard form is just to store a linefeed (the UNIX convention), then car- carriage returns should be turned into linefeeds [f the internal format is to store both (the Windows convention), then the driver .should generate a linefeed when it gets

SEC. 5 6 CHARACTER-ORIENTED TERMINALS a linefeed No 339 latter what the a carriage return and a carriage reiurn when it gets internal convention, the terminal may require both ,1 linefeed and a carnage return to be echoed in order to get the screen updated properly. Since a large computer may well have a wide variety or different terminal1- connected to it, it is up to the keyboard driver to get all the different carriage return/linefeed combinations con- converted to the internal system standard and arrange for all echoing to be done right. When operating m canonical mode, a number of input characters have special meanings. Figure 5-3n shows all of the special characters required by POSlX The defaults are all control characters that Should no) conflict with text mpui or codes used by programs, but al[except the last two can be changed under progiam nent Comment Backspace one characti inme line being typed Interpret next character literally ^top output _Stattoutpul tnterrupi pjqcess (SIGINT) force core dump (SIGQUIT) End of tile _ I Carriage return (unchangeable^ Linefeed (i thai are handled spe The ERASE character allows the user to rub out the character just typed. It usually the backspace (CTRL-H). It is not added to the character queuL- bi instead removes the previous character from the queue. It should be echoed us sequence of three characters, backspace, space, and backspace, in order to rem<>\ the previous character trom the screen. If the previous character was a tab. t ing it depends on how it was processed when u was typed If it i expanded into spaces, some extra information is needed to del back Up. [f the tab itself is Stored m the input queue, it c:i entire line just output again. In most systems, backspacing will only erase charac- characters on the current hne It will not erase a carriage

return and back up into the previous line. When the user notices an error at the start ol the line being typed in, it is often convenient to erase the entire line and start again. The KILL charactei erase*, the entire Hue. Most systems make the erased |me vanish from the screen, but a few echo it plus a carriage return and linefeed because some users like to see the old immediately e how fai t<, n be removed .md the

340 INPUT/OUTPUT CHAP S line. Consequently, how to echo KILL is a matter ot taste. As with ERASE U is usually not possible to go further back than the current line. When a block of characters is killed, it may or may not be worth (he trouble for the driver to return buffer;; to the pool, if one is used. Sometimes the ERASE or KILL characters must be entered as ordinary data. The LNEXT character serves as an escape character. In UNIX CTRL-V is the default As an example, older UN[X systems often used the @ sign for KILL, but ihc- Internet mail system u^es addresses of the form hnda@cs.wcishingion.edu. Someone who feels more comfortable with older conventions might redefine KILL as, @- but then need to enter an @ sign literally to address email This can be dune by (yp(ng CTRL-V @. The CTRL-V itself can be entered literally by typ- typing CTRL-V CTRL-V After seeing a CTRL-V, the driver sets a flag saying that the next character is exempt from special processing The LNEXT character itsell is not entered in (he character queue. To allow users to .stop a screen image from scrolling out of view, control codes are provided to freeze the screen and restart it later In UNIX these .Me STOP. (CTRL-S) and START. (CTRL-Q), respectively They are not stored but <ire used to set and clear J flag in the terminal data structure Whenever ot.itpin !s attempled, the flag is inspected, if it is set. no output occurs Usually, echoing is also suppressed along with program output It is often necessary to kill a runaway program being debugged The INTR (DEI-) and QUIT (CTRLA) characters can be used for this purpose In UNIX. DE[- sends the SlGINT signal to all the processes siarted up from the terminal Implementing DEL can be quite tncky. The hard part is gutting the infoimation from the driver to the part of the system that handles signal*,, which, after .ill. Has not asked for this information. CTRL-\ is similar t» DHL, except that it sends the SlGQUlT signal, which forces a core dump it not caught Or ignored. When either of these keys is struck, the driver should echo a carnage return ami linefeed and discard all accumulated input to allow for a fresh start. The default vjlue toi INTR is often CTRL-C instead ot DEL, iinct? man> programs

use DE[inter- interchangeably with the backspace for editing Another special character k EOF (CTRL-D). which m UNIX causes ;in> pending read requests for ihe terminal to be satisfied with whatever is available m the buffer, even if the buffer is empty Typing CTRL-D at the stait of a line ises the p lerpre cnd-o|-file and causes most programs to act the same way as they \ seeing end-of-file on an input file. Some terminal drivers allow much fancier intrahne editing than we have ward or forward character*, or words, go to the beginning or end of the line being iype<l. inserting text in tlie middle of the line, and so tenth Adding all these func- functions to the terminal driver makes it much larger and, furthermore, is wasted when using fancy screen editors that work m raw mode anyway.

SEC. 5.6 CHARACTER-ORIENTED TERMINALS 341 S.6.3 Output Software Output is simpler ihan input. For the most part, the computer sends characters m the terminal and [hey are displayed there. Usually, a block of character*,, for example, a line, is written to the terminal in one system call The method that is commonly used for RS-232 terminals is to have output buffers associated with each terminal. The buffers can Lome from ihe same pool as [he input buffeis. or be dedicated as with input. When a programs writes to the terminal, the output is first copied to ihe buffer. Similarly, outpui from echoing U also copied to the buffer. After all the output has been copied to the burfer. the first character is out- output, and the driver goes to sleep When the interrupt comes in. the next character ?.s output, and so on Screen editors and many other sophisticated programs need to be able to update the screen in complex ways such as replacing one hue in the middle of the screen To accommodate this need, most terminals support a a series of toni- mands to move the cursor, insert and delete characters or lines at the cursor, etc. These commands are often called escape sequences. In the heyday o| the RS-232 terminal. Ihere were hundreds of terminal types, each with its own escape sequences. As a consequence, it was difficult to write software that worked on more ihan one terminal type One solution, which was introduced in Berkeley L'N|X, was a terminal data- database called termcap This software package defined a number of basit actions, such as moving the cursor io (ran, tt>luntn) To inovc the cursor to a particular location, the software, say, an editor, used a generic escape sequence which was" then converted to the actual escape sequence for the terminal being written to In this way. the editor worked on any terminal that had an entry m the tcrmcap data- database. Eventually, the

industry saw the nved toi standardization of the escape sequence, so an ANSI standard was developed A few or the values are shown in F.g. 5-37. Consider how these escape sequences might be used by a text editor. Suppose that the user types a command telling the editor to delete all of line 3 and then close up the gap between lines 2 and 4 The editor might send the following escape sequence over the serial line to the terminal: ESC[3,1HESC[OK ESC[t M (where ihe spaces are used above only io separate the symbols; they are not transmitted). This sequence moves' the cursor to the start of line 1, erases the entire |me, and then deletes the now-empty line, causing all the lines starting at 5 to move up I ime. Then what was line 4 becomes line 3: what was line 5 becomes line 4. and so on Analogous escape sequences can be u^ed io add text to the middle or the display. Words and be added or removed in a similar way.

INPUT/OUTPUT @ to end, 1 from start, 2 all) , 1 from start. 2 all) rt n chars at cursor I Enable rendition n@=normal, 4=bold, 5=blinking, 7=teverse) | Scroll the screen backward if the cursor is on the top line 5.7 GRAPHICAL USER INTERFACES PCs can use character-based interfaces. In fact, for years MS-DOS, which k character based, dominated the scene. However, nowadays most personal rom- ;ronyni GUI is pronounced a GUI (Graphical User Interface) The i "gooey.- The GUI was invented by Douglas Engelbart and his research group ai the Stanford Research Institute. It was then copied by researchers at Xerox PARC. One fine day, Steve Jobs, cofounder of Apple, was touring PARC and saw a GUI on a Xerox computer. This gave him the idea for a new computer, which became the Apple Lisa. The Lisa was too expensive and was a commercial failure, but its successor, the Macintosh, was a huge success. The Macintosh was the inspiration for Microsoft Windows and other GUI-based systems. A GUI has four essential elements, denoted by the characters WIMP These letters stand for Windows, Icon;,, Menus, and Pointing device, respectively. Win dows are rectangular blocks of .screen area used to run programs. Icons are little symbols that can be clicked on to cause some action to happen Menuti arc lists of actions from winch one can be chosen. Pmaily, a pointing device is a mou.se. trackball, or other hardware device used to move a cursor around the screen to The GUI software can be implemt UNIX systems, or in the operating system itself, Mther •level code, as is done in case in Windows, [n the

Spr s 7 GRAPHICAL USER INTERFACES 343 following sections we will

look at ihe hardware, input software, and output software associated with personal computer GUIs with an emphasis on Windows, bui the general ideas hold for othe,- GUIs as well. 5.7.1 Personal Computer Keyboard. Mouse, and Display Hardware Modern personal computers ah have a keyboard and a bit-onented memory- nupped display These components are an integral part of the computer itself. However, ihe keyboard and screen are completely .separated, each with their own driver. The keyboard may be interfaced via a serial port, a parallel port, or a USB port. On every key action ihe CPU is inierrupted, and the keyboard driver extracts the character typed by reading an I/O port. Everything else happens in software, mostly in the keyboard driver. On a Pentium, the keyboard contains an embedded microprocessor which communicates through a specialized serial port with a controller chip on the parentboard. An interrupt is generated whenever a key it. struck and also when one is released. Furtherm<ire. all that the keyboard hardware provides is the key number, not the ASCII code. When the A key is struck, the key code C0) is put in an I/O register It is up to the driver to determine whether it is lower case, upper case. CTRL-A. ALT-A, CTRL-ALT-A. or some other combination. Since the driver can tell which keys have been struck but not yet released (e g.. SHIFT), it has enough information to do the job. For example, the key sequence DEPRESS SHIFT. DEPRESS A, RELEASE A. RELEASE SHIFT indicates an upper case A. However, the key sequence DEPRESS SHIFT, DEPRESS A. RELEASE SHIFT, RELEASE A also indicates an upper case A. Although thiK keyboard inierface puts the full bur- burden on the software, it is extremely flexible. For example, user programs may be interested in whether a digit just typed came from the top row of keys oi the numeric key pad on the -side. In principle, the driver can provide this information. Most PCs have a mouse, or sometimes a trackball, which is just ;■ mou.se lying on its back. The most common type or mouse hsi.s a rubber ball inside that pro- protrudes1 through a hole in the bottom and rotates as the mouse is moved over a rough surface. As the ball rotates;, it rubs against rubber roller-; placed on orthogo- orthogonal shafts. Motion in the east-west direction causes the shaft parallel to the >-axjs to rotate; motion in the north-south direction causes the shaft parallel to the T-axis to rotate. Whenever the mouse has moved a certain minimum distance in either direction or a button is depressed or released, a message is sent io the computer. The minimum distance is about O.I mm (although it can he set in software), ome people call this unit a mickey. Mice (or occasionally, mouses) can have

344 INPUT/OUTPUT CHAP S one, two, or three buttons, depending on the-designers' estimate oi ihe users' intellectual ability ti> keep track of more than one button. The message to the computer contains thiee items' Ax, Ay. buttons. The first item is the change in x position since the last message Then comes, the change in v position since the last message Finally, the stains ot the button.s is included. The format <>t the message depends on the s>stem and the number of buttons the mouse has Usually, it lakes J bytes. Most mice report back a maximum of 40 timos/sec, so the mouse may have moved multiple mickeys since the last report Note that the mouse only indicates changes in position, not absolute position itself If the mouse is picked up and put down gently without causing the ball U. rotate, no messages will be sent. Some GUIs distinguish between single clicks and double clicks Of *i mouse button If two clicks are close enough in space (mickeys) and also close enough in lime (milliseconds). „ double click is signaled. The maximum for "close enough" is up to the software, with both parameters usually being usel sellable That is enough about the input hardware, let us now (urn to the display hard- hardware. Display devices can be divided into two categories. Vector graphics devices can accept and carry out commands such as drawing points, lines, geometric figures, and text [n contrast, raster graphics devices represent the output area as a rectangular grid of points called pixels, each of which has some grayscale value or some color In the early days or computing, vector graphics devices were common, but now plotters are the only vector graphics devices around. Everything else uses raster graphics, sometimes called bitmap graphics Raster graphics displays are implemented by a hardware device called ,i graphics adapter. A graphics adapter cont.nns a special memory called a video RAM. which forms part of the computer's address space atid is addressed by the CPU rhe same way as the rest of memory (see Fig. 5-38) The screen image is stored here m either character mode or bit mode In character mode, each byte (or 2 bytes) of video RAM contains one character to be displayed. In bitmap modi-, each pixel on the screen is represented separate!) in the video RAM. with I l>it per pixel for the simplest black and white display to 24 or mom bits pet pixel tor a high quality color display Also part of the graphics adapter is a chip called a video controller This chip pulls characters or bits out of the video RAM and generates the video signal used to drive the monitor. A monitor generates a beam of electrons that scans horizontally across the screen, painting lines on it T\pically the screen has 480 to 1024 lines from top to bottom, with 640 to | 200 pixels per l,ne The video eon trailer signal modulates

the electron beam, determining whether a given pixel will be light or dark. Color monitors have three beams, for red. green, and blue, which are independently modulated. Flat panel displays also use pixels m three colors, but how these displays work in detail is beyond the scope of ifm book Video controllers have two modes: character mode (used for simple text) and bn mode (for everything else). In character mode, the controllei might fit each

GRAPHICAL USER INTERFACES Graphics adapter video an trailer character in a box 9 pixels wide by 14 pixels high (including the space between characters), and have 25 lines of 80 characters The display would then have 350 scan lines of 720 pixels' each. Each of these frames is redrawn 60 to |0() limes a second to avoid flicker To display the text on the screen, the video controller might fetch the first 80 characters from the video RAM, generate 14 scan lines, fetch the next 80 charac- characters from the video RAM, generate the following |4 scan lines, and so on Alter- Alternatively, it could fetch each character once per scan line to eliminate the need for buffering in the controller. The 9-by-l4 bit patterns for ihe characters are kept in a ROM used by the video controller. (RAM may also be used to support custom fonts.) The ROM is addressed by a 12-bit address, 8 bits from the character code and 4 bits to specify a scan line The 8 bits in each byte of the ROM control 8 pixels; the 9th pixel between characters is always blank. Thus 14 x SO = 1120 memory references to the video RAM are needed per line of text on the screen. The same number of references are made to the character generator ROM In Pig. 5-39(a) we see a portion of the video RAM for a display operating in character mode. Each character on ihe screen of Fie ^ ^y<b> octupies two bytes in the RAM. The low-order character is the ASCII code for the character to be displayed. The high-order character is the attribute byte, which is used to specify the color, reverse video, blinking, and so on. A screen of 25 by 80 characters requires 4000 bytes of video RAM in this mode. Operating in bitmap mode uses the same principle, except that each pixei on the screen is individually controlled and individually represented by | or more bits in the video RAM. in the simplest configuration for a monochrome display, each screen pixel has a corresponding bit in the video RAM At the other extreme, each screen pixel is represented by a 24-bit number in the video RAM, with g bits each for the Red. Green, and Blue intensities. This RGB representation is used because red, green, and blue are the primary additive colors, from which all other colors can be constructed by summing various

intensities of these colors. Screen sizes vary, the most common being 640 x 480 (VGA), 800 x 600 (SVGA), 1024x768 (XGA). 1280 X [024, and 1600 x 1200. AH of these except

Vide ... xZ .. . xD xCxSxA |—160 characters H i-- 80 Characters ■-! |280x |024 are in the ratio of 4:3, which fits the aspect ratio of NTSC television sets and thus gives square pixels. 1280* 1024 should have been 1280x960, but the lure of 1024 was apparently to great too resist, even though it distorts the pix- pixels slightly and makes scaling to and from the other sizes harder. As. an example, a 768 x 1024 color display with 24 bits per pixel requires 1 25 MB of RAM just to hold the image. If the full screen is refreshed 75 times/sec, the video RAM must be capable of delivering data continuously at 169 MB/sec. To avoid having to manage such large screen images, some systems have the ability to trade off color resolution against image <,\ze In the simplest s,cheme, each pixel is represenied by an g-bit number. Rather than indicating a color, this value is an index inlo a table of 256 entries, each holding a 24-bit (red, green, blue) value. This table, called a color palette and often stored in hardware, allows ihe screen to contain an arbitrary 256 colors ai any instant. Changing. sa>. entry 7 in the color palette, changes the color of all the pixels in the imagt with a 7 as value. Using an g-bit color palette reduces the amount of space needed to store the screen image from 3 bytes per pixel to 1 byte per pixel. The price paid k coarser color resolution The OIF image compression scheme also works with j color palette like this It is also possible to use a color palette wiih 16 bits per pixel, [n this case the color palette contains 65.536 entries so that up to 65336 colors can be used at once. However, the saving in space is less since each pixel now requires 2 bytes in the video RAM. Also, if the color palette is kept m hardware (to avoid an expensive memory lookup on every pixel), more dedicated hardware is needed to store the color palette. It is also possible to handle 16-bit color by storing the RGB values as three 5- bii numbers, wiih ! bit left over (or to give green 6 bits, since the eye is more sen sitive to green than to red or blue), [n effect, this system is the same as 24-bit color, except with fewer shades, of each color available.

SEC 5 " GRAPHICAL USER INTERFACES 347 5.7.2 Input Software Once the keyboard driver has received the character, it must begin processing it. Since the keyboard delivers key numbers rather than the character codes used

by application software, the driver must convert between (he codes by using a table. Not all IBM "compatibles'" use standard key numbering, to if the driver wants to support these machines, it must map different keyboards with different tables. A simple approach is to compile a table that maps between the codes pro- provided by the keyboard and ASCII codes into the keyboard driver, but this is unsa- unsatisfactory lor users' or languages other than English. Keyboards are arranged dif- differently in different countries, and the ASCII character set is not adequate even for the majority of people in the Western Hemisphere, where speakers of Spanish, Portuguese, and French need accented characters and punctuation mark.; not used in English. To respond to the need for flexibility of keyboard layouts to provide for dif- different languages, many operating systems, provide for loadable keymaps or code pages, which make it possible to choose the mapping between keyboaid codes and codes delivered to the application, either when Ihe system is booted oi later 5.7.3 Output Software for Windows Output software for GUIs is, 2 massive topic. Many 1500-page books have been written about the Windows GUI alone (e.g , Petzold, 1999; Simon. 1997; and Rector and Newcomer. 1997). Clearly, in this seciion, we can only scratch the surface and present a tew of ihe underlying concepts. To make the discussion concrete, we will describe the Win32 API. which is supported by a‖ 32-bit ver- versions of Windows. The output software for other GUIs is roughly comparable in a general sense, but the details are very different. The baste iiem on the screen is a rectangular area called a window. A window's position and size are uniquely determined by giving the coordinates (in pixels) of two diagonally opposite comers. A window maj contain a title bar, a menu b;ir, a tool bar, a vertical scroll bar. and a horizontal scroll bar. A typical window is shown in Fig. 5-40 Note (hat the Windows coordinate Astern puts the origin in the upper leftliand corner and has \ increase downward, which is dif- different from the Cartesian coordinates used in mathematics' When a window is created, the parameters specify whether ihe window can be moved by the user, resided by the user, or scrolled (by dragging the thumb on the scroll bar) by the user. The main window produced by niosi programs can be moved, resized, and scrolled, which has enormou* consequences for the way Win- Windows programs are written. In particular, programs must he informed about changes io ihe size of their windows and must be prepared to redraw the contents of their windows at any time, even when they least expeci it.

 As a consequence. Windows programs are message oriented User actions involving the keyboard or mouse are captured by Windows and converted into messages to the program owning Ihe window being addressed. Each program has a message queue to which messages relating to all its windows are sent. The main loop of the program consists of fishing out the next message and processing it by calling an internal procedure for that message type. In some cases, Windows itself may call these procedures directly, bypassing the message queue. This model is quite different than the UNIX model of procedural code that makes sys- system calls to interact with the operating system. To make this programming model clearer, consider the example of Fig. 5-41. Here we see the skeleton of a main program for Windows. It is not complete and does no error checking, but it shows enough detail for our purposes. It starts by including a header file, windows.fi, which contains many macros, data types, con- constants, function prototypes, and other information needed by Windows programs. The main program starts with a declaration giving its name and parameters. The W1NAPI macro is an instruction to Ihe compiler to use a certain parameter

GRAPHIC \1- t'SFR INTERFACES ((include <windows h> int WINAPI WinMain(HINSTANCE hT HINSTANCE. hprev, char *SZCmd, int iCmdShow { WNDCLASS wndclass, /* class object for this window */ MSG msg, /* incoming messages are stored here */ HWND hwnd, /* handle (pointer) to the window object */ /* Initialize wndclass */ wndclass.IpfnWndProc = WndProc; /* tells which procedure to call */ wndclass IpszClassName = "Program name", /* Text for title bar */ wndclass hlcon = Loadlcon(NULL, IDI APPLICATION), /- load program icon */ wndclass.hCursor = LoadCurSor(NULL, IDC .ARROW); /* load mouse cursor */ RegisterCIass(&wnddass). /* tell Windows about wndclass */ hwnd = CreateWindow (..) /* allocate storage for the window */ ShowWindow(hwnd, iCmdShow), /* display the window on the screen */ UpdateWindow(hwnd); /* tell the window to pain! itself */ while (GetMessagef&msg, NULL, 0, 0)) { /* get message from queue */ TranslateMessage(Smsg), /* translate the message */ DispatchMessage(&msg); /* send msg to the appropriate procedure */ } return(msg wParam), } long CALLBACK WndProc(HWND hwnd, UINT message, UINT wParam, long IParam) { /* Declarations go here */ switch (message) { case WM.CREATE: . , return ; /* create window */ case

WM_PAINT. . , return .. , /* repaint contents of window */ case WM DESTROY. . return , /* destroy window */ return(DefWindowProc(hwnd, message, wParam, IParam));/* default */ figure 5-41. A skeleton (it j Window, iimhi pro(.,aii> passing convention and will not be of further umcern to us The first pai amctw. It, is an mstjnce handle ,md is used to identity the program Id the rest of the sys- tains objects (e g.. programs, files, and windows! that have some state and associ- associated code, called methods, thai operate on that state. Objects, are referred to using

350 INPUT/OUTPUT CHAP. 5 handles, and m this case, A identifies the program. The second parameter is present only for reasons of backward compatibility It ts. no longer used. The ihird parameter, \:Cmd, is a zero-terminated string containing ihe command line rhat started the program, even if it was not started from a command line. The fourth parameter. iCmdSkow. tells whether the program's initial window should occupy the entire screen, par! of the screen, or none of the screen (task bar only). This declaration illustrates a widely used Microsoft convention called Hun- Hungarian notation. The name is a pun on Polish notation, the postfix system invented by the Polish logician J. LukasieWic? tor representing algebraic formulas without using precedence or parentheses. Hungarian notation was invented by a Hungarian programmer at Microsoft. Charles Simonyi. and uses (he first few characters of an identifier to specify the type The allowed letters and types include c (character), w (word, now meaning an unsigned 16-bit integer), i C2-hit signed integer). I (long, also a 32-bit signed integer), s (string), sz (string ter- terminated by a zero byte), p (pointer), fn (function), and h (handle). Thus \;Cmd is a zero-terminated string and iCmdShmv is an integer, for example. Many pro-programmers believe that encoding the type in variable names this way has little value and makes Windows code exceptionally hard to read. Nothing analogous to this convention is present in UNIX, Every window must have an associated class object that defines its properties. In Fig. 5-41, that class object is wndclaa.. An object ot type WNDCLASS has 10 fields, four of which arc initialized in Fig. 5-41. In an actual program, the other six would be initialized as well. The most important field is ipfnWndProv. which is a long (i.e., 32-bit) pointer to the function that handles the messages, directed to this window. The other fields, initialized here tell which name and icon to use in the title bar, and which symbol to use for the mouse cursor. After wndclass has been initialized, RegisterClass is called to pass it to Win- Windows. In

particular, after this call Windows knows which procedure to call when various events occur that do not go through the message tjueue. The next call, CrealeWiitdow, allocates memory for the window's data structure and returns a handle for referencing it later. The program then makes two more calls tn a row, to put the window's outline on the screen, and finally fill it in completely. At this point we come to the program's mam loop, which consists of getting ,i message, having cenain translations done to it, and then passing li back to Win-Windows to have Windows invoke WndProc to process it. To answer the tjucstion of whether this whole mechanism could have been made simpler, the answer is yes. but it was done this way for historical reasons and we are now stuck with it. Following the main program is the procedure WndProc. which handles the various messages that can be sent to the window The use of CALLBACK here like WtNAPf above, specifies the calling sequence to use for parameters The tirsi parameter is the handle ol the window to use. The second parameter is the mes- message type. The third and fourth parameters can be used to provide additional information when needed.

SEC 5 7 GRAPHICAL L'St.R 1NTERHACES 351 Message types 1VM CREATE and IVM DESTROY are sent ai [he start and end of the program, respectively. They give the program the opportunity, for example, to allocate memory for data structures and then return it The third message type, WM-PAINT. is an instruction to the program to fill m [he window. It is noi only called when the window is first drawn, but often during program execution as well. In contrast to text-based systems, in Windows a pro- program cannot assume that whatever it draws on the screen will stay there until it removes it. Other windows can be dragged oil iop of this one, menus can be pulled down over it, dialog boxes and tool tips can cover par! ot it. and so on. When these items are removed, the window has to be redrawn The way Win- Windows tells a program to redraw a window is to send it a WM_PAlNT message. As a friendly gesture, it also provides information about what part of the window has been overwritten, in case it is easier to regenerate that part ot the window instead of redrawing the whole thing. There are two ways Windows can get a program to do something. One way is to post a message lo its message queue. This method is used for keyboard input, mouse input, and timers that have expired The other way, sending a message to Ihe window, involves having Windows directly call WnJProc itself This method is used lor all other events Since Windows is notified when a message is fully processed, it can refrain

from making a new call until the previous one is finished. In this1 way race conditions are avoided. There are many more message type.s. To avoid erratic behavior .should an unexpected message arrive, it is best to call DefWindowProc at the end of WndProt- to let the default handler take care of the other cases In summary, a Windows program normally creates one or more windows with a class object for each one. Associated with e;ich program is a message queue and a set of handler procedures. Ultimately, the program's behavior is driven by the incoming events, which are processed by the handler procedures. This is a veiy different model of the world than the more procedural view that UNIX takes. The actual drawing to the screen is handled by a package consisting of hun- hundreds ol procedures thai are bundled together to form the GDI (Graphics Device Interface). It can handle text and all kinds of graphics and is designed to be plat- platform and device independent. Before a program can draw (i.e , paint) in a win- window, it needs to acquire a device context which is an internal data structure con- containing properties of the window, such as the current font, text color, background color, etc. Most GDI calls use the device context, either tor drawing ot for getting or setting the properties. v anous ways exist to acquire the device context exist A si trip It? examrjje of its acquisition and use is hdc = GetDC(hwnd); TextOut(hrJc, x, y, psText. iLength); ReleaseDC(hwnd, hdcv

 The first statement gels a handle to a device- conicni. hdc. The second one use* the device context to write a line of text on the screen, specifying the (jr. v) coordi- coordinates of where the string starts, a pointer to the string itself, and its length. The third call releases the device context to indicate that ihe program is through draw- drawing for the moment. Note that hdc is used in a way analogous to a UNIX file descriptor. Also nole that ReleaseDC contains redundant information (the use of hdc uniquely specifies a window). The use of redundant information thai has no actual value is common in Windows. Anothei interesting note is that when hdc is acquired in this way. the program can only write in the client area of ihe window, not in the title har and other parts of it- Internally, in the device context's data structure, a dipping region is main- maintained Any drawing outside the clipping region is ignored. However, there is another way to acquire a device context, GetWindowDC, which sets the clipping region to the entire window. Other calls restnci the clipping region jn other ways. Having multiple calls thai do almost the same thing is another characteristic of Windows. A complete

treatment of the GDI is out of the question here For the interested reader, the references cited ahove provide additional information. Nevertheless, a few words ahouc the GDI are probably worthwhile given how important it is. GDI has various procedure calls to get and release device con- contexts, obtain information about device contexts, get and set device context attri- attributes (e.g.. the background color), manipulate GDI objects such as pens, brushes, and fonts, each of which has its own attributes. \ ma]|y, of course, there arc- a large number of GDI calls to actually- draw on [he screen. The drawing procedures lall into four categories' drawing lines and curves, drawing filled areas, managing bifmaps, and displaying text. We saw an example of drawing text above, so let us take a quick look at one ot the others The call Rectangle(hdc, xleft, ytop. xright, ybottom); draws a filled rectangle whose corners are (\lefi, \iop) and (\rigki, vlwuoni) For example. Rectangle(hdC, 2, 1, 6.4), will draw the rectangle shown in Fig 5-42 The line width and color and 1i[I color are taken trom the device context. Other GDI calls are similar in flavor BitMaps The GDI procedures are examples of vector graphics. They arc used to plate geometric figures and text on the screen They can bo scaled easily to larger ur smaller screens (provided the number of pixels on the screen is the same) They are aUc, relatively device independent. A collection of calls (o GDI procedures can be assembled in a file that can describe a complex drawing Such a file is

GRAPHICAL USER INTERFACES called a Windows metafile, and is widely used to transmit drawings from one Windows program to another Such files have extension .wmf. Many Windows programs allow the user to copy (part of) a drawrng and put in on the Windows clipboard. The user can then go to another program and paste the contents of the clipboard into another document One way of doing this is for the first program to represent the drawing as a Windows metafile and put it on the clipboard in .wmf format. Other ways also exist Not all the images that computers manipulate can be generated using vector graphics. Photographs and videos, for example, do not use vector graphics. Instead, these items are scanned in by overlaying a grid oit the image The aver- average red, green, and blue values of each grid square are then sampled and saved as the value of one pixel. Such a file is called a bitmap There are extensive facili- facilities in Windows for manipulating bitmaps. Another use for bitmaps is for text. One way to represent a particular charac- character m some font is as a small bitmap. Adding text to the screen then

becomes a matter of moving bitmaps. One general way to use bitmaps is through a procedure called biiblt. It is called as follows: bitbft(dsthdc, dx, dy, wid, ht, srehde, sx, sy, rasterop); In its simplest form, it copies a bitmap from a rectangle in one window lo a rec- rectangle in another window (or the same one) The first three parameters specify the destination window and position. Then come the width and height- Next come the source window and position. Note (hat each window has iis own coordi- coordinate system, with @, 0) in the upper left-hand comer of the window. The last parameter will be described below The effect of BitBlt(hdc2, 1, 2. 5, 7, hdd, 2, 2, SRCCOPY);

354 INPUT/OUTPUT CHAP. 5 is shown in Fig. 5-43. Notice carefully that the entire 5x7 area of the letter A has been copied, including the background color. ■BVH I! HI HI::: •••■*■■■•■ ■aaakaaaan ■ ■■■> aavaai Windowl Window 2 — . I 0 4 • • • * •• j ■■■ ■ ■■■Mlllllltltlll Mtl ■ ■■■••iiii«i«itiit mi ■ IIIMIIIIIItMMIMK Window r Window 2- .:: :: ■• -^ •• ■: in 9- Figure 5-43. Copying bitmaps using fiuBii. (a) Before, (b) After. BilBIi can do more than just copy bitmaps. The last parameter gives the pos- possibility of performing Boolean operations to combine the source bitmap and the destination bitmap. For example, the source can be ORed into the destination to merge with it. li can also be EXCLUSIVE ORed into it. which maintains the characteristics of both source and destination. A problem wiih bitmaps is thai they do not scale. A character that is in a box of 8 x 12 on a display of 640 x 480 will look reasonable. However, if this bitmap is copied to a printed page at 1200 dots/inch, which is 10200 bits x 13200 bits the character width (8 pixels) will be 8/1200 inch or 0.17 mm wide. In addition, copying between device* with different color properties or between monochrome and color does not work well. For this reason. Windows also supports a data structure called a DIB (Device Independent Bitmap). Files using this format use the extension bmp. These files have file and information headers and a color table before the pixels. This informaiion makes it easier to move bitmaps between dissimilar devices. Fonts In versions of Windows before 3.1. characters were represented as bitmaps and copied onto the screen or printer using Bitfiti. The problem with that, as we just saw. is that a bitmap that makes sense on the screen is too small for the printer. Also, a different bitmap is needed for each character in each size In other words, given ihe bitmap for A in 10-poinl type, there is no way to compute ll tor 12-point type. Because every character of every font mighi be needed for sizes ranging from 4 point to 120 point, a vasl

number of bitmaps were needed The whole system was just too cumbersome for text. The solution was the introduction of TrueType fonts, which are noi bitmaps but outlines of the characters. Each TrueType character is defined by a sequence

GRAPHICAL USER INTERFACES of points a iund i perim the pom ; relative lo (he @, 0) c Using this system, it is, easy 10 scale the characters up or down. All !hat has to be done is lo multiply eacli coordinate by the same stale facior. In this way. a True- TrueType character can be scaled up or down to any potnl size, even fractional point sizes. Once at the proper size, the points can be connected using the well-known follow-the-dots- algorithm taught in kindergarten (note that modern kindergartens use splines for smoother results). After ihe outline has been completed, the char- actei can he filled in An example of si,me characters scaled to three different point sizes is given in Fig. 5-44 Figure 5-44. S< Once the filled character is available m mathematical form, it can be raster- ized, that is. converted lo a bitmap at whatever resolution is desired. By first scal- scaling and then rasterizing. we can be sure that ihe characters displayed on the screen and those that appear on the printer will be as close as possible, differing only in quantization error. To improve the quality still more, it js possible to embed hints, tn each character telling how to do the rasterization. For example, both serifs on the top of the letter T should be identical, something that might not otherwise be the case due to roundoff error. 5.8 NETWORK TERMINALS Network terminals are used to connect a remote user to t'(irr*puter over a net- network, either a local area network or a wide area network. There an: two different philosophies of how network terminals should work. In one view, the terminal

356 INPUT/OUTPUT <-'HAP 1 should have a large amount of computing power and memory in order 10 run com- complex protocols id compress, the amount of data sent over ihe network (A protocol is a set of requests and responses that a sender and receiver agree upon in order to communicate over a network or other interface.) In the other view, the terminal should be extremely simple, basically displaying pixels and not doing much think- thinking in order to make it very cheap In the following two suctions we will discuss an example of each philosophy. First we will examine the sophisticated X Win- Window System. Then we will took at the minimal SLIM terminal 5.8.1 The X Window System The ultimate in intelligent terminals is a

terminal that contains a CPU as powerful as the main computer, along with megabytes ol memory, a kevboiirci. and a mouse One terminal of this type is the X terminal, which iuns the X Win- Window System (often just called X), developed at M.I.T. as part of project Athena. An X terminal is a computer that runs the X software and which interacts with programs running on a remote computer. The program inside the X terminal that collects input from the keyboard or mouse and accepts commands from a remote computer is called the X server It has to keep track of which window is currently selected (where the nwwe pointer is), so it knows which client to send any iiev. keyboard input to. It communicates board and mou.se input and accepts display commands from them It may seem strange to have the X server inside the terminal and the clients on the remote host, but the X server's job is to display bits, so it makes sense to be near the user From the program's point of view, it is u client telling the server to do things, like display text and geomeinc figures The server (m the terminal) just does what it is told, as dt, all servers. The arrangement ot client and server is shown in Fig. 5-45. It is also possible to run the X Window System on top ot UNIX or .mother operating system. In fact, many UNIX systems run X a.s their standard windowing system, even on standalone machines or tor accessing remote computers over the Internet What the X Window Sy.stem really defines is the protocol between the X client and the X server, as shown in Fig 5-15. It does not matter whether the client and server are on the same machine, separated by iOO meters over a local area network, or are thousands of kilometers apart and connected by the Internet, The protocol and operation of the system is identical m all cases- X isjust a windowing system. It is not a complete GUI To get a complete GUI. others layer of software <ire run on lop ot it. One layer is XNb. which is a set ot library procedure*, tor accessing the X functionality. These procedures form the basis of the X Wmdow System and are what we wdl examine below, but rhe> are too primitive for most user programs to access directly. For example, eacli mouse click is reported separately, so that determining that two clicks really form a double click has to be handled above Xhb.

NETWORK TERMINALS Remo Tansger enosi Application program Molif lnmns.CS xiib Xcwnt UNIX Hardware Figure S-45. Che vet\ m ihe M 1 T X Window p i, deletion* and f nds commands to the X server t achme as the X client, but m the To make programming with X easier, a toolkit consisting of the Intrinsics is supplied as part of X. This layer manages buttons, scroll bars, and

other GUI ele- elements, called widgets To make a true GUI interface, with a uniform look and leel. yet another layer is needed. The most popular one K called Motif. Mosi applications make use oi falls to Motif rather than Xlib Also worth noting is that window management is not part ot X itself. The decision to leave it out was fully intentional Instead, a separate X client proc called a window manager, controls the creatio dows on the screen. To manage windows, it se ling what to do. It often runs on the s can run anywhere. This modular design, consisting of several layers and multiple programs, makes X highly portable and flexible. It has been ported to most versions ot UNIX, including Solaris. BSD. A1X. Linux, and so on, making it possible for appli- application developer* to ha\e a standard user interface for multiple platforms,. It has also been porled to other operating systems. In contrast, m Windows, the win- windowing and GUI systems are mixed together in the GDI and located in the kernel, which makes them harder to maintain. For example, the Windows 98 GUI is- still fundamentally 16 bits, more than a decade after the Intel CPUs were 32 bits Now lei us take a brief look at X as viewed from the Xltb level When an X program starts, it opens a connection to one or more X servers— let us call them workstations even though they might be colocated on the same machine as the X

 program itself. X considers this connection to be reliable m ihc sense that lust ;md duplicate messages are handled by the networking software and it does not h.ive to worry about communication errors Usually. TCP/IP is used bctwee,, the client and server. Four kinds- of messages go over the connection I Drawing commands from the program to the workstation. 2. Replies by the workstation to program queries. 3. Keyboard, mouse, and other event announcements. 4. Error messages. Most drawing commands; are sen! from the progtam to the works-tat inn as one-way messages. No teply ts expected. The reason for this design is that when the client and server processes are on different machines, it may take a substantial period of time for the command to reach the server and be carried out Blocking the application program during this time would slow it down unnecessarily. On the other hand, when the program needs information from the workstation, i! sim- simply has to wait until the reply comes back. Like Windows, X is highly event driven. Events flow from the workstation to the program, usually in response to some human action such as keyboard strokes, mouse movements, or a window being uncovered. Each event message is 32 bytes, with the first

byte giving the event type and the next 31 bytes providing additional information. Several dozen kinds of events* exist, but a program is sent only those events that it has said it is willing to handle. For example, if a program does not want to hear about key releases, it is not sent any key release events. As in Windows, events are queued, and programs read events from the queue How- However, unlike Windows, the operating system never calls procedures within the application program on its own. It does not even know which procedure handles which event. A key concept in X is the resource. A resource is a data structure that holds certain information. Application programs create resources on workstations Resources can be shared among multiple processes on the workstation Resources tend to be short lived and do not survive workstation reboots. Typical resources include windows, fonts, colomiaps (color palettes), pixmaps (bitmaps), cursors, and graphic contexts. The latter are used to associate properties with windows and are similar in concept to device contexts in Windows. A rough, incomplete skeleton of an X program is shown in l-'ig 5-46 It begins by including some required headers and then declaring some variables. It then connects to the X server specified as the parameter to XOpenOisplav Then it aljocaies a window resource and stores a handle to it in win. In practice, some initialization would happen here After that it tells the window manager that the new window exists so the window manager can manage it.

NETWORK TfcRMINAI,S #mclude<X11/Xiib.h> #include<Xi1/XiJtil.h> main(mt argc, char *argv[]) Display disp; Window win; GCgc, XE> /* server identifier */ /* window identifier */ /* graphic context identifier' /* storage for one event */ = 1; disp = XOpenDisplay("display..name"); /* connect to the X server */ win = XCreateSimp|eWmdow(disp,); /* allocate memory for new window */ XSetStandardProperties(disp,...); /* announces window to window mgr * gc - XCreateGC(disp, win, 0, 0). /* create graphic context */ XSelectlnput(disp, win, ButtonPressMasK I KeyPressMask I ExposureMaskj; XMapRaised(disp, win); /* display window; send Expose event */ while (running) (XNextEvent(disp, Sevent): switch (event type) { case Expose: ...; case ButtonPress case Keypress ...; /* get i break, break; break, next event */ /* repaint window */ /* process mouse click */ /* process keyboard input */ XFreeGC(disp, gc). XDestroyWindowfdisp, win); XCIoseDisplay(disp); * release graphic context •/ * deallocate window's memory spac * tear down network connection */ -e5-46. A skeleton eaieGC LLieate> ii X Windo The

call lo XCveateGC cieates a graphic context in which properties of the window are stored. In a more complete program, they might be initialled here. The nexi statement, the call to XSelectlnput, tells the X server which events the program is prepared to handle. In this case it is interested in mouse clicks, keys- keystrokes, and windows being uncovered. In practice, a real program would be interested in oilier events as well Finally, the call to XMapRuised maps the new window onto the screen as, the uppermost window. At this, point [he wmduw becomes visible on the screen. The main loop consists of two statements jnd is logically much simpler than the corresponding loop in Windows. The first statement here gets an even! and the second one dispatches on the event type for processing. When some event

360 INPUT/OUTPUT CHAP. 5 indicates that the program has finished, running is set to 0 and the loop terminates Before exiting, the program releases the graphic context, window, and connection. II is worth mentioning that not everyone tikes a GUI Many programmers prefer a traditional command-line oriented interface of the type discussed in Sec. 5.6.2 above. X handles this via a chem program called xterm. This program emu- emulates an old VT102 intelligent terminal, complete with all the escape sequences. Thus editors such as vi and emacs and other software that uses termcap work in these windows without modification. 5.8.2 The SLIM Network Terminal Over the years, the main computing paradigm has oscillated between central- centralized and decentralized computing. The first computers, such as the ENlAC, were, m fact, personal computers, albeit large ones, because only one person could use one at once. Then came timesharing systems, in which many remote users at sim- simple terminals shared a big central computer. Next came the PC era, in which the users had their own personal computers agam While the decentralized PC model has advantages, it also has some severe disadvantages that are only beginning to be taken seriously Probably the biggest problem is that each PC has a large hard disk and complex software that must be maintained. For example, when a new release of the operating system comes out, d great deal of work has to be done to perform the upgrade on each machine eparately. At most corporations, the labor costs of doing this kind of software maintenance dwarf the actual hardware and software costs. For home users, the labor is technically free, but few people are capable of doing it correctly and fewer still enjoy doing it. With a centralized system, only one or a few machines have to be updated and those machines have a staff of experts

to do the work. A related issue is that users should make regular backups of their gigabyte file systems, but few of them do. When disaster strikes, a great deal oi moaning and wringing of hands tends to follow. With a centralized system, backups can be made every night by automated tape robots Another advantage ts that resource sharing ts easier with centralized systems A system with 64 remote users, each with 64 MB of RAM will have most of that RAM idle most of the time. With a centralized system with 4 GB of RAM, it never happens that some u.ser temporarily needs a lot of RAM but cannot gel it because it is on someone else's PC. The same argument holds for disk space and other resources. It is probably a fair conclusion to say thai most users want high-performance interactive computing, but do not really warn to administer a computer. This lias led researchers to reexamine timesharing using dumb terminals (now politely called thin clients) that meet modem terminal expectations. X was a step in this direction, but an X server is still a complex system with megabytes of software

SEC. 5 8 NETWORK TERMINALS 361 that must be upgraded from lime 10 time. The holy grail would be a high perfor- performance interactive computing system in which the user machines, had no software at all interestingly enough, this goal is achievable Below we will describe one such system, developed by researchers at Sun Microsystems and Stanford Univer- University and now commercially available from Sun (Schmidt et al.. 1999). The system is called SLIM, which stands for Stateless Low-level Interface Machine. The idea U based on traditional centralized timesharing as shown in Fig 5-47. The client machines are just dumb]2S0x 1024 bitmap displays with a keyboard and mouse but no user-installable software They are very much in the spirit ot the old intelligent character-oriented terminals that had a little firmware to interpret escape cudes. but no other software Like the old character-oriented terminals, the only user control is an on-off switch. Terminals of this type that have Little processing capacity are called thin clients. Computer Figure 5-47. The archiiec i of the SUM is 1 v, lem The simplest model—having the server ship bitmaps over the network to the SLIM clients 60 times a second—does not work. It requires about 2 Gbps of net- network bandwidth, which is far too much for current networks to handle. The next simplest model—storing the screen image m a frame buffer inside the terminal and having it refreshed 60 limes a second locally— is much more promising. In particular, if the central server maintains a copy of each

terminals frame buffer and sends only updates (i.e., changes) to jt as needed, the bandwidth requirements, become quite modest. This is how the SLlM thm clients work. Unlike the X protocol, which has hundreds of complex messages tor manag- managing windows, drawing geometric figures, and displaying text in many fonts, the SLIM protocol has only five display messages, listed in Fig 5-48 (there are also a small number of control messages noi listed). The first one, SET, just replaces a rectangle in the frame buffer with new pixels. Each pixel replaced requires 3 bytes in the message to specify its full B4-bit) color value. In theory, this mes- message is sufficient to do the job; the other ones are just for optimization.

INPUT/OUll'Ur _ _ JMeaning __ | llpdateaj-ectangle with new pixels _ __ _} Fill a rectangle with one pixel vatue_ _ __) Expand a bitmap to fill a rectangle | Copy a rectangle from one part of the frame buffer to another ' Convert a rectangle from television color (YUV) to RGB^ _ Mc.--.iiKV- used inthi. SUM protocol tioiTiihe seivcr Idihe lenninjK The hll message tills an entire rectangle on the screen with a single pixel value It us used for filling in uniform backgrounds The BITMAP message tills .in entire rectangle by repeating a pattern contained in a bitmap supplied in the mes- message. This command i*, useful for filling in patterned backgrounds that have some Tlic COPY message instructs the terminal to copy a rectangle within the frame buffci to another part of ihe frame buffer It is most useful for scrolling the screen and moving windows, for example. Finally, the CSCS message converts the YUV color system used in US. (NTSC) television sets to the RGB system used by computer monitors It is pn- inanly used when a raw iideo frame has been shipped to a terminal in the YUV rtthmieaUy simple but time consuming, so it is better to offload the work to the terminals. If the terminals will not be used for watching videos, this message and its functionality are nut needed. The whole idea of dumb thin clients sUnds or falls with the performance, which Schmidt et al. extensively measured. In the prototype. 100-Mbps switched Fast Ethernet wa.s used on both the server-to-switch segment and switch-io- tenninal segments. Potentially a gigabit network could be used between thL- server and the switch because that segment r.s local to the central computer room. The first measurement deals with character echoing to the scrceii. Each typed character ts» .sent to the server, which computes which pixels have to be updated to place the character on the screen in the right position, font, and color. Measure- Measurements show that it takes 0.5 msec

for the character to appear on the screen. In contrast, on a local workstation the echoing lime is 30 msec due to kernel buffer- ing The rest of the tests measured the performance with users running modern in- interactive application programs .such as Adobe Photoshop (a program tor retouch- retouching photographs). Adobe Framemaker (a desktop publishing program), and Net- Netscape (a Web browser) It Was observed that half the user commands required up- updating fewer than 10,000 pixels, which is 30.000 byte.s uncompressed At 100 Mbp.s. it takes 2.4 msec to pump 10,000 pixels out onto (he wire, it takes another 2.7 msec to place them m the frame butfer upon arrival, tor a total of 5.1 msec (but this varies a little depending on the circumstances). Since human reaction

 lime is about 100 msec, such updates appear to be instantaneous Even the larger updates were almost instantaneous- Furthermore, when compression is used, more than 85^ of the updates, arc under 30.000 bytes The experiments were repealed with a 10-Mbps network, a I-Mbps network, and a 128-Kbps network At 10 Mbps the system was virtually instantaneous and I Mbps it was still good. Al 128 Kbps it was too slow to use. Since I Mbps con- connections lo the home are rapidly becoming a reality using cable TV networks and ADSL (Asymmetric Digital Subscriber Loop), il appears [hat this technology may be applicable lo home users as well as business users. 5.9 POWER MANAGEMENT The first general-purpose electronic compuier. the ENIAC, had 18.000 vacuum tubes and consumed 140,000 waits of power. As a result, it ran up a non- tnvial electricity bill. After the invention ot the transistor, power usage dropped dramatically and the computer industry tout interest in power requirements. However, nowadays power management is back m the spotlight for several rea- reasons, and the operating sysiem is playing a role here. Let us start with desktop PCs. A desktop PC often has a 2<H-watt power sup- supply (which is typically 85% efficient, that is, loses 15% of the incoming energy to heat). If 100 million oi these machines arc turned on at once worldwide, together they use 20,000 megawatts of electricity This is, the total outpui (,f 20 average- sized nuclear power plants. If power requirements could be cut m halt, we could get rid of 10 nuclear power plants. From an environmental point of i'icw, getting rid of 10 nuclear powei plants (Or an equivalent number of fossil fuel plants) is a big win and well worth pursuing. The other place where power is a big issue is on battery-powered compuier.s. including notebooks, laptops, palmtops, and

Webpads. among others- The heart of the problem is that the batteries cannot hold enough charge to last very lung, a few hours al most. Furthermore, despite massive research efforts by battery com- companies, compuier companies, and consumer electronics companies, progress is glacial. To an industry used to a doubling of the performance every 18 mouths (Moore's law), having no progress at all seems like a violation of the laws of phy- physics, but that is the current situation. As a consequence, making computers use less energy so existing batteries last longer is high on everyone's agenda. The operating system plays a major role here, as we will see belnw. There are two general approaches to reducing energy consumption. The first one is, for the operating system to turn off parts of the computer (mostly I/O devices) when they are not tn use because a device that js off uses little ot no energy. The second one is for the application program to use less energy, possibly degrading [he quality of the user experience, in order lo stretch oul battery time. We will look at each of these approaches in turn, but first we will say a little bit about hardware design with respect io power usage.

5.9.1 Hardware Issues Batteries come m two general types, disposable and rechargeable. Disposable batteries (most commonly AAA, AA, and D cells) can be used to run handheld devices, but do not have enough energy lo power laptop computers with large bright screens. A rechargeable battery, m contrast, can store enough energy to power a laptop for a few hour;. Nickel cadmium batteries used to dominate here, but they gave way to nickel metal hydride batteries, which last longer and do not pollute the environment quite as badly when they are eventually discarded. Lithium ion batteries are even better, and may be recharged without first being hilly drained, but their capacities are also severely limited The general approach most computer vendors take to buttery conservation is to design the CPU, memory, and I/O devices to have multiple states: on, sleeping, hibernating, and off. To use the device, it must be on. When the device will not be needed for a short time, it can be put io sleep, which reduces energy consump- tion When it is not expected to be needed for a longer interval, it can be made io hibernate, which reduces energy consumption even more. The trade-off here is that gelling a device out of hibernation often takes more time and energy than get- getting it out of sleep stale. Finally, when a device is off, it does nothing and con- consumes no power. Not all devices have all these states, but when they do, it is up to the operating system to manage the state transition;, at the right moments Some computers have two or

even three power buttons One of these may put the whole computer in sleep state, from which it can be awakened quickly by typ- typing a character or moving the mouse. Another may put the computer into hiberna- hibernation, from which wakeup takes much longer tn both cases, these buttons, typi- typically do nothing except send a signal io the operating system, which does the rest in software. In some countries, electrical devices must, by law, have a mechani- mechanical power switch that breaks a circuit and removes power trom the device, for safety reasons. To comply with this law, another switch may be needed Power management brings up a number of quesiions that the operating system must deal with. They include the following Which devices can be controlled' Are they on/off, Or do they have intermediate states ? How much power is saved in the low-power states? Is energy expended to restart the device7 Must some context be saved when going to a low-power state? How long does it take to go back to full power? Of course, the answers to these questions vary from device to device, so the operating system must be able to deal with a range of possibilities. Various researchers have examined laptop computers to see where the power goes. Li et ai. A994) measured various workloads and came to the conclusions shown tn Fig. 5-49. Lorch and Smith A998) made measurements on other ma- machines and came to the conclusions shown m Fig. 5-49. Weiser et al. A994) also made measurements but did not publish the numerical values- They simply stated that the top three energy sink*, were the display, hard disk, and CPU, in that order. While these numbers do not agree closely, possibly because the different brands

 of computers measured indeed have different energy requirements, it & that the display, hard disk, and CPU are obvious targets for saving enerj , Device Display CPU ; Hard disk Modem [Sound Memory Other Figure 5-49. Pov, Lletal, A994} 68% 12% 20% 0.5% cr consumption ,if v Lorch and Smith A998) 39% 18% i 12% 6% 2% 1% 22% i 5.9.2 Operating System Issues The operating system plays a key role in energy management, it controls all the devices, so it must decide what to shut down and when to shut tt down. If it shuts down a device and that device is needed again quickly, there may be an annoying delay while tt is restarted. On the other hand, if it waits too long to shut down a device, energy is wasted for nothing. The trick is to find algorithms and heuristics that let the operating system make good decisions about what to shut down and when. The trouble is that "good" is highly subjective. One user may find it acceptable that after 30

seconds of not using the computer it takes 2 seconds for it to respond to a keys-keystroke. Another user may swear a blue streak under the same conditions. In the absence of audio input, the computer cannot tell these users apart. The Display Let us now look at the big spenders of the energy budget to see what can be done about each one. The biggest item in everyone's energy budget is the display. To get a bright sharp image, the screen must be backlit and that t the user can decide frequent blanking of the screen and using the battery up quickly bac (who probably really does not want it) Turning off the display is a sleep because it can be regenerated (from the video RAM) almost instantaneously when any key is struck or the pointing device is moved.

366 INPU'L/OinPUT One possible improvement was proposed by Fhnn and Saiyanarayanaii A999). They suggested having the display consist or some number ot zones that can be independently powered up or down. In Fig. 5-50. we depict 16 zones using dashed lines to separate them. When the cursor is in window 2, as shown in Fig- 5-5O(a), only the four zones in the lower nghthand comer have to be In up. The other 12 can be dark, saving 3/4 of the screen power. When [he user moves the cursor to window I. the zones for window 2 can be darkened and the zones behind window I can be turned on- However, because window 1 straddles 9 zones, more power is needed- If the window manager can sense of what is happening, it can automatically rnove w[ndow 1 to fit into four zones, with a kind of snap-to-zone action, as shown in Fig. 5-5O(b). To achieve [his reduction from 9/16 of full power to 4/16 of full power, the window manager has to understand power management or be capable of accepting instructions from some other piece of the system thai does. Even more sophisticated would be the ability to partially illuminate a window that was not completely full (e.g., a win- window containing short lines, of text could be kept dark on the nght hand side). ...] [Window 1 -1 --I-- 1 ' i ; Window 2 N, ! —-j——j- ! _ _ Window 2 The Hard Disk e of zones lor backlighting the display (a) When iv noved (b) When window I is selccled. >l moves !i illuminated. Another major villain is the hard disk. It takes substantial energy to keep it spinning at high speed, even if there are no accesses. Many computers, especially laptops, spin the disk down after a certain number of minutes of activity. When it is next needed, it is spun up again. Unfortunately, a stopped disk is hibernating rather than sleeping because it takes quite a few seconds to spin it up again, which causes noticeable delays for the user. In addition, restarting the disk consumes considerable extra energy. As a

consequence, every disk has a characteristic time, Td, that is its break-even point.

sf-(s g MOWER MASAGFMF'NT 367 often in the range 5 hi 15 sec. Suppose that the next disk acess is evicted to some time t in the fuime If 1 < Td. it takes less energy to keep the disk -spinning rather than spin it down and then spin it up so quickly 11 r > Tlt, the energy saved makes it worth spinning the d[sk down and up agam much later. It" a good predic- prediction could he made (eg., based on past access patterns), the operating system could make good shutdown predictions and save energy. In practice, most sys- systems are conservative and only spin down the disk after a tew minutes of mac- Anothcr way to s;iire disk energy is io have a substantial di.sk cuclie in RAM. II a needed block is m the cache, an idle disk does not have to be restarted to satisfy the read Similarly, if a write to the disk can be buffered in the cache, a stopped disk does not have to restarted |iist 10 handle the write The disk can remain off until the cache fills up or a read miss happens Another way to avoid unnecessary disk starts is for the operating system to keep running programs informed about the disk state by sending it messages or signals. Some programs have discretionary wiites that can be skipped or delayed. For example, a word processor may be set up to write the tile being edited to disk every few minutes. If the word processor knows that the disk is off at the moment it would normally write the file out, it can delay this write until the disk is next turned on or until a certain additional time has elapsed The CPU The CPU can alst, be managed to save energy A laptop CPU can be put to sleep in software, reducing power usage to almost zero. The only thing it can do in this state is wake up when an interrupt occurs Therefore, whenever the CPU goes idle, either waiting for I/O or because there is no work to do, it goes to sleep. On many computers, there is a relationship between CPU voltage, clock cycle, and power usage The CPU voltage can often be reduced m software, which saves energy but also reduces the clock cycle (approximately linearly) Since power consumed is proportional to the square of the voltage, cutting the voltage in half makes the CPU about half as fast but at 1/4 the power This property can be exploited for programs with well-defined deadlines, such as multimedia viewers that have to decompress and display a frame every 40 msec, but go idle if they do it faster. Suppose that a CPU uses x joules while run- running tull blast for 40 msec and v/4 joules running at half speed If a multimedia viewer can decompress and display a frame in 20 msec, the

operating system can run at full power for 20 msec and then shut down for 20 msec for a total energy usage of x/2 joules. Alternatively, it can run at half power and just make the deadline, but use only v/4 joules instead. A comparison of running at full speed and full power for some time interval and at half speed and one quarter power for twice as long is shown in Fig. 5-51. In both cases the same work is done, but in Fig. 5-51 (b) only half the energy is consumed doing it.

Figure S-Sl. (a) Running ai full clock speed (b) Cutting voltage by two cuts clock speed by two and power consumption by lour. In a similar vein, if a user is typing at 1 char/sec, but the work needed to proc- process the character takes 100 msec, it is better for the operating system to detect the long idle periods and slow the CPU down by a factor of 10. In short, running slowly is more energy efficient than running quickly. The Memory Two possible options exist for saving energy with the memory- First, the cache can be flushed and then switched off. It can always be reloaded from main memory with no loss of information. The reload can be done dynamically and quickly, so turning off the cache is entering a sleep state. A more drastic option is to write the contents of main memory to the disk, then switch off the main memory itself. This approach is hibernation, since virtu- virtually ail power can be cut to memory at the expense of a substantial reload time, especially if the disk is off too. When the memory is cut off, the CPU either has to be shut off as well or has to execute out of ROM. If the CPU is off, the inter- interrupt that wakes it up has to cause it lo jump to code in a ROM so the memory can be reloaded before being used. Despite all the overhead, switching off the memory for long periods of time (e.g., hours) may be worth it if restarting jn a few seconds is considered much more desirable than rebooting the operating sys- system from disk, which often takes a minute or more. Wireless Communication Increasingly many portable computers have a wireless connection to the out- outside world (e.g., the Internet). The radio transmitter and receiver required are often first-class power hogs. In particular, if the radio receiver is always on in order to listen for incoming email, the battery may drain fairly quickly. On the other hand, if the radio if, switched off after, say, 1 minute of being idle, incoming messages may be missed, which is clearly undesirable.

SEC 5 9 POWER MANAGEMENT 369 One efficient solution to this problem has been proposed by Kravets and Krishnan A998). The heart of their solution

exploits the fact that mobile comput- computers communicate with fixed base stations that have large memones and disks and no power constraints. What they propose is to have the mobile computer send a message to the base station when it is about to turn off Ihe radio. From that tune on, the base station buffers incoming messages on its disk. When the mobtje computer switches on the radio again, u tells the base station. At that point any accumulated messages tan be sent to it. Outgoing messages that are generated while the radio is off are buffered on the mobile computer. It the buffer threatens to fill up, the radio is turned on and the queue transmitted to the base station When should the radio be switched off One possibility is to let the user or the application program decide. Another is turn it off after some number of sec- seconds of idle time. When should it be switched on again? Again, the user or pro- program could decide, or it could be switched on periodically to check for inbound traffic and transmit any queued messages Of course, it also should be switched on when the output buffer is, close to full Various other heuristics are possible. Thermal Management A somewhat different, but still energy-related issue, is thermal management. Modem CPUs get extremely hot due to their high speed. Desktop machines nor- normally have an internal electric fan to blow the hot air out of the chassis,. Since reducing power consumption is usually not a driving is£ue with desktop machines, the fan is usually on all the time. With laptops, the situation is different. The operating system has to monitor the temperature continuously. When it gets close to the maximum allowable tem- temperature, the operating system has a choice. It can switch on the fan, which makes nojse and consumes power. Alternatively, it can reduce power consump- consumption by reducing the backlighting of the screen, slowing down the CPU, being more aggressive about spinning down the disk, and so on. Some input from the user may be valuable as a guide. For example, a user could specify in advance that the noise of the fan is objectionable, so the operating system would reduce power consumption instead. Battery Management In ye olde days, a battery just provided current until it was drained, at which time it stopped. Not any more. Laptops, use smart batteries now, which can com- communicate with the operating system. Upon request they can report on things like maximum voltage, current voltage, maximum charge, current charge, maximum drain rate, current drain rate, and more. Most laptop computers have programs that can be run to query and display all these parameters. Smart batteries can also

^70 INPL WtH IK I' OKI' -i be inslriiaed to change various operational paiamctcrs undei control ot the operating system. Some laptops have multiple batteries. When the operating system detects ihat one battery is about to s:o. il has to arrange tor a graceful cutover to the nexi one. without causing any glitches during ihe transition When the final battery is on its last legs, it is up to the operating system to warn the user and ihen cause an ord- Driver Interface The Windows system has an elaborate mechanism tor doing power manage- management called ACPI (Advanced Configuration and Power Interface). The oper- operating system can send any conformant Jrtvei coinmandt, asking il to report on the capabilities of its devices and their current slates This feature is especially important when combined with plug and play because just after it is booted, the operating system does not even know what devices are present, let alone their pro- properties with respect to energy consumption 01 power manageability. It can also sends commands to drivers instructing them to cut their power lev- levels (based on the capabilities that it learned earlier, of course). There is also some traffic the other way. In particular, when a dev]Ct such as a keyboard or a mouse detects activity after a period of idleness, this is a signal to the system to go back (O (near) normal operation 5.9.3 Degraded Operation So far we have looked at ways the operating system can reduce energy usage by various kinds' of deuces. But there is another approach as well: tell the pro- programs to use less energy, even if this means providing a poorer user experience (better a poorer experience than no experience when the battery dies and the lights go out). Typically, this tnh mutton is p issed on when [he battery charge is below some threshold. It is then up to the programs to decide between degrading perfor- performance to lengthen battery life or to maintain performance and risk running out of energy. One of (he questions that comes up here is how can a prograni degrade its per- performance to save energy? This question has been studied by Flinn and Satya- narayanan A999). They provided four examples of how degraded performance can save energy. We will now look at these. In this study, information is presented to the user in various forms. When no degradation is present, [he best possible information is presented When degrada- degradation i.s present, the fidelity (accuracy) of [he information presented to the user is worse than what it could have been. We will .see examples of this shortly. In order to measure energy usage, Flinn and Satyanarayanan devised a soft- software too] called PowerScope. What it does is provide a power usage profile ot ,i

SI c 5 4 lJ0Wh.R MANAGtMEM -V71 program. To usc it, a computei must be hooked up U> an external power supply Jhroujih a s<jTiware-coniroiK'd digiiaJ inultimeier Using ihe mullimetcr. software can read out the number of milliamperes turning in from the power supply and thus determine the instantaneous power being consumed by the computer. What PowerScope does is periodically sample (he program counter and the power usage and write these daia 10 a tile. After the program has- terminated the file is analyzed to give the c-nergy usage of each procedure. These measurements formal the basis of their observations Haidware energy saving measure* were also used and formed the- baseline against which the degraded performance was The first program measured was a video player- In undegraded mode, it plays 30 trames/sec in full resolution and in color. One tonn of degradation is to aban- abandon ihe color information and display the video in black and white. Another form of degradation is to reduce the frame rate, which leads, to flicker and gives the movie a jerky quality. Siill another fonn of degradation h to reduce the number of pmels m both directions, either by lowering the spatial resolution i)r making the displayed image smaller. Measures of this type saved about 30% of the energy. The second program was a speech recognizer. It sampled ihe microphone to construct a waveform. This waveform could either be analyzed on ihe laptop computer or sent over a radio link for analysis on a fixed computer. Doing this saves CPU energy but uses energy for the radio. Degradation was accomplished by using a smaller vocabulary and a simpler acoustic model The win here was about 35% The next example wa.s a map viewer (hat fetched (he map over the radio link. Degradation consisted of either cropping the map (o .smaller dimensions or telling the remote server to omil smaller roads, thus requiring fewer bits to be transmit- transmitted Again here a gain of about 35% was achieved. The fourth experiment was with transmission of JPEG images to a Web browser. The JPEG standard allows various algorithms, trading image quality against file size. Here the gain averaged only 9%. Still, all in all. the experiments showed that hy accepting some quality degradation, the user can run longer on a given baitery. 5.10 RESEARCH ON INPUT/OUTPUT There ts a fair amount of research on input/output, but most of it is focused on specific devices, rather than I/O m general Often the goal ts to improve perfor- performance in one way or another. Disk systems are a case in point. Older disk arm scheduling algorithms use a disk model that is not really applicable any more, so Worthington el aJ. A994) took a look at models that correspond to modem

disks. RAID is a hot topic, with various researchers look at different aspects of u. Alvarez et al. A997) looked a(

372 INPUT/OUTPUT CHAP *> enhanced fault tolerance, as did Blaum ei a) A994). Cao et al A994) examined the idea of having a parallel controller on a RAID. Wilkes et al. A9%) described an advanced RA|D system they built at HP Having multiple drives requires good parallel scheduling, so that is also a research topic (Chen and Towsley, 1996; and Kallahatla and Varman, 3999). Lumb el al. B000| argue for utilizing the idle time after the seek but before the sector needed rotates by the head to preload data. Even better than using the rotational latency to due useful work is to elim- eliminate the rotation in the first place using a solid-state mtcroelectromechanical storage device (Griffin et al., 2000; and Carley et al, 2000) or holographic Storage (Orlov, 2000). Another new technology worth watching is magneto-optic a I storage (MeDaniel, 2000). The SLIM terminal provides a modern version of the old timesharing system, with all the computing done centrally and providing users with terminals that jusi manage the display, mouse, and keyboard, and nothing else (Schmidt et al.. 1999). The main difference with old-time timesharing is that instead of connecting the terminal to the computer with a 9600-bps modem, a 10-Mbps Ethernet is used, which provides enough bandwidth for a full graphical interface at the user's end GUIs are fairly standard now, but there is still work continuing to go on in that area, for example, speech input (Malkewitz, 1998, Manaris and Harkreader, 1998; Slaughter et al., 1998; and Van Buskirk and LaLomia, 1995). Internal structure of the GUI is also a research topic (Taylor et al.. 1995). Given the large number of computer scientists with laptop computers and given the microscopic battery lifetime on most of them, it should come as no surprise that there is a lot of interest in using software techniques to manage and conserve battery power (EUis, 1999; Flinn and Satyanarayanan, 1999: Kravets and Krishnan, 1998; Lebeck eta].. 2000; Lorch and Smith, 1996; and Luet al., 1999). 5.11 SUMMARY Input/output is an often neglected, but important, topic. A. substantial fraction of any operating system is concerned with I/O I/O can be accomplished m one of three ways. First, there is programmed I/O, in which the main CPU inputs or out- outputs' each byte or word and sits in a tight loop waiting until it can get or send the next one. Second, there is interrupt-driven I/O, in which the CPU starts an I/O transfer for a character or word and goes off to do something else until an inter- interrupt arrives signaling completion of the I/O. Third, there k

DMA. in which a separate chip manages the complete transfer of a block of data, given an interrupt only when the entire block has been transferred. I/O can be structured in four levels: the interrupt service procedures, the de- device drivers, the device-independent I/O software, and the I/O libraries and spool- spoolers that run in user space. The device drivers handle the details of running th<.>

SRC 5 II SUMMARY 373 devices and providing uniform interfaces to the rest ot the operating system The device-independent I/O software does things like buffering and error reporting Disks come in a variety of types, including magnetic disks, RAIDs, and vari- various kinds of optical disks. Disk arm scheduling algorithms can often be used to improve disk performance, but the presence of virtual geometries complicates matters By pairing two disks, a slable storage medium with certain useful pro- properties can be constructed. Clocks are used for keeping track of the real time, limiting how long processes can run, handling watchdog timers, and doing accounting. Character-oriented terminals have a variety of issues concerning special char- characters that can be input and special escape sequences that can be output Input can be m raw mode or cooked mode, depending on how much control the program wants over the input. Escape sequences on output control cursor movement and allow for inserting and deleting text on the screen. Many personal computers use GUIs for their output. These are based on the WIMP paradigm: windows, icons, menus and a pointing device. GUI-based pro- programs are generally event driven, with keyboard, mouse, and other events being sent to the program for processing as soon as they happen Network terminals come in several varieties. One of the most popular con- consists of those running X, a sophisticated system that can be used to build various GUIs, An alternative to X Windows is a low-level interface that simply ships raw pixels across the network. Experiments with the SUM terminal show that this technique works surprisingly well. Finally, power management is a major issue for laptop computers because battery lifetimes are limited. Various techniques can be employed by the operat- operating system to reduce power consumption. Programs can also help out by sacrific- sacrificing some quality for longer battery lifetimes 1. Advances in chip technology have made it possible lo pur ing all me bus access logic, on an inexpensive chip. How Fig l-5*> 2. Given the speeds listed in Fig. 5-|, is u possible to scan document from a scanner onto an BIDE disk attached to an ISA bus ai full speed? Defend your answer 3.

Figure 5-3(b) ihows one way of having memory-mapped I/O even in the presence of separate buses for memory and [/O devices, namely, to first try the memory bus and if thai faik try ihe i/O bus. A clever computer science student has thought of an improvement on this idea: try both in parallel, io speed up the process of accessing I/O devices. What do you think of this idea?

374 rNPuT/m ii'ut <-H\i' *■ 4. A DMA controller h;«s lour channels The conirollei is i_j|wbk' ol reijuestiny a i_-foii uord cvets 100 n>eo A response lakes ci|uj!1> lr>ny How fast does ihe bus have 1,) hch.jvoidbeing.ibwiluneck' 5. Suppose thin a compuiei can read t,r *nte a mcmor> *ord in 10 nsec Also suppose that when an ,nterrupt occurs, all 12 CPU registers, plus ihe program counter .md PSW nrc pushed onto ihe stuck Whai k ihe maximum number ot interrupts per output to Ihe pnnier. Could II ha\L- cquuliv well been acknowledged right at the sluii nl ihe interrupt service procedure1 If so gt\e one reason lor doing ii at the erui as it] rhciwt If not. ^hyuot' 7. /\ computer has j three-stage pipeline as shown in Pi^ I~o(a) On each clock cycle, one new instruction is tciched Irom nwnKwj at the address pointed to by the VC and put inio the pipeliue and ihe PC advanced bach iiistriiction occupies exactly one When an inierrupi occurs, ihe current PC is pushed onto ihe suck, iind ihe PC i4* sei m the address ot the inierrupt handler Then the pipeline is shitied nghi one stage and the tirsi insiruetion nl ihe irlienupt handler is fetched into the pipeline Does ihi, 8. A typical printed page »f te*i L-iintains 50 lines ot 80 characters each Imagine that .i primer's output register is si, short h <-an he ignored. Docs n make scnso to run this printer using n>ierrupt-driven I/O if each characiej- printed requires an interrupt th<it lakes. W nscc a[I-tn to service' 9. What is "device independence" ' 10. In which ol the four I/O soli ware layers is each of the iullnwing done (<i) Computing the track, sector, and head for a disk read. (b) Writing commands to the device register. (c) Checking to see if the user is permitied to use the device. (d| Convening binary integers to ASCII for printing 11. Based on Ihe daia i>t Fig. 5-17. what is tlie iwnsfcr raie tor transfers between the disk and the controller for a floppy disk and a hard disk" Hov, dues this compare with a 56-Kbps modem and ltlO-Mbps Fasi Ethernet, respectively'1 12. A local area network is used as follows The user issues, j system call to w.nie data packets to (he network The operating system then copies the data io a kernel butter Then it copies ihe data to the network conirullcr board When all the bytes arc saiely inside the controller, they are sent over the network at a rate of 10 tnegahits/sct Tlic receiving network

controller siores each bit a microsecond after it is sent. When ilic last bit arrives, the deslination CPU is interrupted, ,md ihe kernel copies ihe newly arrived packet to a kernel buffer io inspect » Once u has figured oui which user tiit; packet is for, Ihe kernel copies the data io the user space. It we assume (hat eaih interrupt and its associated processing takes 1 msec, lhat packets are 1024 bytes (ignore the headers), and (hal copying a byte takes- 1 psec, whai is (he maximum rale

al which one process eai> pump dat.i lo anothoi ' Assume that the sendei is blocked until the work is timihed at the receiving side ;iiid an acknowledgement tomes back r-or Mmplicity, assume thai the t,tne 10 get thL-acknowledgement buck is so suiall H can be ignored. 13. Why .ire ouiput iiles fur ihe printer nonn.ill) spooled on disk before being printed'1 \4. How much cyhndei skew is needed i»r „ 720lJ-rpm di.sk with j liuck-tn-irack sfck lin>c of I msec'1 The disk hat. 200 seniors oi "> 12 b>!Cs e.ich on each track problem 16. RAID level 3 is able lo unreel smg|e-bn errors using only one panty ilnve Wh.it is the point ot RAID level 2'* After all, ■ t ulso Ciin only coired one ciror ^ind takes more pose thai ihe probabilily of one dnve crashing in 3 given hour is /j Whai is the proba- probability ot a^drive RAID failing in a given hour' 18. Why are optical storage devices inherentl) capable ot higher data density ihan mag- magnetic slorage devices" Note- This problem requires some knowledge of high-school physics and how magnetic fields are generated receives Ihem. wiih no miemal buffering, is inierlcjvmg conceivably useful'1 Discuss 20. A floppy disk is double interleaved, as in Fig 5-26(c) It has eight <.ecu>i\ ol 512 bytes per (rack, and a rotaiion rate of 300 rpm How long does it lake io read all the sectors of a track in order, assuming che arm is already correctly positioned, and 1/2 rotation is needed to get sector 0 under the head11 WhJi is the data rate" Now repeat the problem for a nomnterleaved disk with ihe same characteristics How much does the data rate degrade due to interleaving'.' 21. If a disk has double interleaving, does it also need eylindei skew in order lo jvoid missing data when making a track-to-track seek'1 Discuss your answer 22. A disk manufacturer has two 5 25-inch disks that each have 10.000 cylinders The newer one has double the linear recording density ol ihe older one Which disk pro- properties are belter on (be newer drive and which art- the same9 23. A eompuier manufacturer decides to redesign the partition table ot a Pentium hard disk io provide more ihan four partitions. What are some consequences of this change ' 24. Disk requests come in to the

disk driver for cylinders |0. 22, 20, 2, 40, 6t and 3&. in thai order A seek takes 6 msec per cylinder moved How much seek time is needed (it) First-come, first served, (b) Closest cylinder next. In all cases, the arm k inilially at cylinder 20

376 INFUT/OLiTPUl CHAP 5 25. A personal computer salesman siting a university in Soulh-West Amsterdam remarked during his sales piich that his company had devoted substantial effort to making their version of UNIX very fasi As an example, he noted that their disk drivci used the elevator algorithm and also Queued mulliple requests witriin a cylinder in sec lor order A student. Harry Hacker, was impressed and bought one He took n home and wrote ;i program id randomly read 10,000 blocks spread across the disk To his amazement, the performance that he measured was fdeniical to what would be expected from firsl-come, first-served Was ihe salesman lying1 26. In the discussion of stahle storage using nonvolatile RAM, the li.llowmg point was 27. The clock inierrupi handler on a certain computer requires 2 msec (including process switching overhead) per clock uck The clock runs at 60 Hz What fraction of the CPU is devoted to the clock" 28. Many versions of UNIX use an unsigned 32-bit mieger to keep track of the time as the number of seconds since the origin (if time When will ihese systems wrap around (year and month)? Do you expect this to actually happen9 29. Some computers need to have large numbers ol RS-232 lines-, ior example, servers r>r Internet providers, for this reason, plug-in cards w,th multiple RS-232 lines exist Suppose that such a card contains a processor [hat must sample each incoming line iit 8 times ihe baud rate to see if the incoming bit ts a 0 or a I. Also suppose that such a sample takes I lisec For 28,800-bps lines operaung at 3200 baud, huu many lines can the processor support? Note. The baud rate of a line is the number of signal changes per second. A 1200-baud line can support 28.800 bps if each .signaling inter- interval encodes 9 bits using various amplitudes, frequencies, and phases. As an aside, 56K modems do not use RS-232, so are not a suitable example of RS-232 timing. 30. Why arc RS232 terminals interrupt driven, bui memory-mapped terminals not inter- interrupt driven? 31. Consider ihe performance of a 56-Kbps modem. The driver outputs one character and then blocks. When the character has been printed, an interrupt occurs and a message is sent to the blocked driver, which outputs the next character and then blocks again If the time to pass a message, output a character, and block is 100 i^sec, what traction or the CPU ts eaten by the

modem handling' Assume [hat each character has one start bit and one stop bit, for I Obits in all 32. A bitmap terminal contains 1280 by 960 pixels To scroll a window, the CPU (or con- controller) must move all the lines of text upward by copying their bits from one part of the video RAM io another. If a particular window is 60 lines high by 80 character, wide E280 characiers, total), and a character's box is 8 pixels wide by 16 pixels high, how long does it lake to scroll the whole window at a copying rate ol 50 nsec per byte' If all lines are 80 characters long, whai is the equivalent baud rate of the termi nal? Putting a characier on the screen takes 5 psec. How many lines per second can be displayed?

CHAP. 5 PROBLEMS 377 currently queued for (hat display Why' 5 occupying chapiciei positions 1 through and including \~L Assuming the irur-oi K nut on line 5 when the command is given, wjtat ANSI escape sequence should [he edi- editor emit to deleie the word > 35. Many RS232 terminals have escape sequence^ for deleting the current lint? and mov- moving all the lines belo1^ it up one line How do yon think ihis feature is implemented inside the terminal0 36. On the original IBM PC's color displav, writing to the video RAM at :iny lime other than during ihc CRT beam's vertical retrace caused ugly spots to appear all over the screen. A screen image is 25 by SO characters, each of which fits m a box 8 pixels by 8 pixels Each row of 640 pixels is drawn on a single horizontal scan ol ihc bc;un. which takes 63 6 jisec. including the horizontal retrace. The screen is redrawn 60 times a second, each ol which requires 3 vertical retrace period to get the beam Kick to the lop. What fraction of the time is the video RAM available for writing in' 37. The designers of a computer system expected that the mouse could be moved at a bytes, what is the maximum data rate Ot the mouse assuming that each niickcy is reported separately *> 38. The primary additive Lolors are red, green, and hlue. whieh means that anv ctilor L.in be constructed from a linear superposition ol these colors Is u possible that someone could have a color photograph thai cannot be represented using full 24-bit color' 39. One way jo place a character on a bitmapped screen j> to use bitblt from a iunt Kilile Assume |hat a particular font uses characters that are 16x24 pixels m true RGB cokn (a) How much font table space does each character lake ' (b) If copying a byte ukes 100 nsec, including overhead what is the outpui r<ne to the screen in charaders/sec? 40. Assuming ihai it takes 10 nsec to cop]y a byte, how much time docs Jt take to com- completely rewrite the screen of an 80 character x 25

line text mode memory-mapped screen? What aboui a 1024 x 768 pixel graphics screen w«h 24-bn coW 41. In Fig. 5-41 there is a class to ReghterCtau. In the corresponding X Window code m Fig 5-46, ihere is no such call or anything like it Why not'1 42. In ihe texi we gave an example of how to draw a rectangle on ihe screen using the Windows GDI. Rectangiefhdc. xleft. ytop, xright, ybottom), Is there any real need for the first parameter {hth), and if so, what9 After all, the coor- coordinates of the rectangle are explicitly specified as parameters 43. A SLIM terminal is used to display a Web page containing an animated cartiwn of size 400 pixels x 160 pixels running at 10 frames/sec What fraction of a 100-Mbps Fasi Ethernet is consumed by displaying the carloon0

378 [NPUT/Ol TPUT fH 44. It has been observed that Ihe SLIM system works well with a 1-Mbps netwo of users watching a scheduled TV show and the same number of ut,ers brows World W,dc Web. 45. It a CPLJ\ maximum voltage. V, is cut to V/n, its powei consumpiion drop* In \ us original value and lit, cluck speed drops to]/n of us original value Suppos user is lypinj, at I char/sec, but the CPU time required to process each charawr msct. What is ihc (ipiimal value i,f " and what is the corresponding energ> sa energy ai all 46. A laptop compuitr is set up to take maximum advantage ol power saving f including shutting down (he display and [he hard disk after periods ol inacti user sometimes run>, liNIX programs in text mode, and ai other nmes uses the d S Sh d fid h b f f p y xt-only progran>s Why' a program that simulates stable siorage Use ivity A X Win- Winhen she o large fixed-length hlc, on you

FILE SYSTEMS All computer applications need to store and retrieve information. While a process is running, it can stoie a limited amount of information within its own address space. However, the storage capacity is restricted to the size of the virtual address space. For some applications this .size is adequate, but for others, such as airline reservations, banking, or corporate record keeping, it is far too small. A second problem with keeping information within a process' address space is that when the process terminates, the information is lost. For many applica- applications, (e.g., for databases), the information must be retained for weeks, months, or even forever. Having it vani.sh when the process using it lerminates is unaccept- unacceptable. Furthermore, it must not go away when a computer crash kills the process. A third problem is that it is frequently necessary for multiple

processes 10 access (parts of) the information at the same time. It we have an online telephone directory .stored inside the address space of a single process, only that process can access it. The way to solve this problem is to make the information itself independent of any one process. Thus we have three essential requirements for long-term information storage- 1. It must be possible to store a very large amount ot information. 2. The information must survive the termination of the process u*,mg it. 3. Multiple processes must be able to access the information concurrently The usual solution to all these problems is to store information on disks and other external media in units called files. Processes can then read them and write new

380 HLE SYSTEMS CHAP 0 affected b\ process creation and termination. A file should only disappear wnen Us owner explicitly removes it f-iles are managed by the opeiatmg system. How they are structured, named, ■iccessed, iKed, protected, and implemented are major topics in operating system design. As a whole, that part ot the operating system dealing with tiles, is km.wn as the file system and is the subject of this chapter. From the users' standpoint, the most important aspect of a file system is ho1* it appears to them, that is, what constitutes a file, how files are named and pro- protected, what operations are allowed on files, and so on The details of whether linked lists or bitmaps are used to keep track of free storage and how many sectors there are in a logical block are of less interest, although they are of great impor- importance to the designers of the file system. For this reason, we have structured the chapter as several sections. The first two are concerned with the user interface to files and directories, respectively Then comes a detailed discussion of how the file system is implemented. Finally, we give some examples ot real die systems. In the following pages we will look at files from the user's point ot view, mat is, how they are used and what properties they have 6.1.1 File Naming Files are an abstraction mechanism. They provide a way to store information on the disk and read it back later This must be done m such a way as to shield the user from the details of how and where ihe information is stored, and how ihe disks actually work. Probably the most important characteristic of any abstraction mechanism is the way the objects being managed are named, so we will start our examination of file systems with the subject of file naming. When a process creaies a file, it gives the file a name. When the process terminates, the file continues to exist and can be accessed by other processes using ha name The exact rules for file naming vary somewhat from system to system, bui all

current operating systems allow strings of one to eight letters as legal file names. Thus andrea, brute, and cathy are possible file names. Frequently digits and spe- special characters are also permitted, so names like 2, urgent', and Fig.2-14 are often valid as well. Many file systems support names as long as 255 characters Some file systems distinguish between upper and lower case letters, whereas olhers do not. L'NTX falls in ihe first category. M.S-DOS falls in the second. Thus a UNIX system can have all of the following as three distinct files: maria, Maria, and MARIA. In MS-DOS, all these names refer id the same file.

SEC FILES 381 An aside on file systems is probably in order here. Windows 95 and Windows 9S both use the MS-DOS file system, and thus inherit many nf its properties, such as how file names are constructed. In addition, Windows NT and Windows 20TO support the MS-DOS file system and thus also inherit its properties- However, the laner two systems also have a native file system (NTFS) thai has different proper- tics (such as file names in Unicode). In this chapter, when we refer to the Win- Windows file system, we mean the MS-DOS file system, which is the only file system supported by all versions of Windows. We will discuss the Windows- 2000 native file system in Chap. II. Many operating systems support two-pan file names, with the two parts separated by a period, as m prog.c. The pan following the period k called the file extension and usually indicates something about the file In MS-DOS, for exam- example, file names are I to 8 characters, plus an optional extension of 1 to 3 charac- character';. In CNIX, the size of the exiension, if any, is up to the user, and a file may even have two or more extensions, as in prog c.Z, where .Z is commonly used to indicate that the file (prog.c) has been compressed using the Ziv-Lempel com- compression algorithm. Some of the more common file extensions and their meanings are shown in Fig. 6-1. file 9if I Cornpuserve Graphical Interchange Fotmat image , We .hip ~\ Help f tie~ I filThiml ! World Wide Web Hyperjext Markup Language document 1 \ tile |pg I Strtl ptctwe encoded wth tt\e_JPEG standard i 1 file.mpS _ Music encoded ,n MPEG layer 3 audio formal fflempg ~~ 1 Movie encoded with the MPEG standard I I tile o ' Ob|ect tile (compiler output, not yet linked) j Portable Document Format file [jPostScnptJile Inpul for me TEX formatting jsrogra f General text file - -\ 'tile zip \ Compressed archive __ __ J Figure (i-1. Sume ivpicdl lik extensions. In some systems (e.g.. UNIX), file extensions are just conventions and are not enforced by the operating system. A file named file.txt might lie some kind of

text file, but that name is more to remind the owner than to convey any actual infor- information 10 the computer. On the other hand, a C compiler may actually insist that files it is to compile end in .t, and it may refuse to compile them if they do not.

382 FILE SYSTEMS CHAP 6 Conventions like this are especially useful when the same program can handle several different kinds of files. The C compiler, for example, can be gii-en a list of several tiles to compile and link together, some of them C files and some of them assembly language files. The extension then becomes essential for the com- compiler to tell which are C files, which are assembly files, and which are other files. In contrast. Windows is aware of the extensions and assigns meaning to them. Users (or processes) can register extensions with the operating system and specify for each one which program "owns- that extension. When a user double clicks on a file name, the program assigned to its file extension is launched with the file as parameter. F()r example, double clicking on fde.doc starts Microsoft Word w\thfile.doc as the initial file to edit. 6.1.2 File Structure Files can be structured in any of several ways. Three common possibilities are depicted in Fig. 6-2, The file in Fig. 6-2(a) is an unstructured sequence of bytes. In effect, the operating system does not know or care what is in ihe tile. All it sees are bytes. Any meaning must be imposed by user-level programs. Both UNIX and Windows use this approach. Figure 6-2. Three kinds of tiles (a) Byte sequence (b) Record sequence. Having the operating system regard files as nothing more than byte sequences provides the maximum flexibility. User programs can put anything they want in their files and name them any way that is convenient. The operating system does

not help, but it also does not get in the way For useis who want to do unusual things, the latter can be very important The first step up in structure is shown in Fig o-2(b). In this model, a file is a sequence of fixed-length records, each with some internal structure. Central to the idea ot a file being a sequence of recoids is the idea that the read operation returns one record and the write operation overwrites or appends one record. As a historical note, in decades gone by. when tlie K()-column punched card was king, many (mainframe) operating systems based their file systems on files consisting of 80-character records, in effect, card images. These systems also supported files of 132-eharacter records, which were intended for the line printer (which in those

days were big chain printers having 132 columns). Programs read input in units be spaces, oi course. No current general-purpose system works this way The third kind of file structure is shown in Fig. 6-2(c). In this organi/.ation, a tile consists of a tree of records, not necessarily all the same length, each contain- containing a key field in a fixed position in the record The tree is sorted on the key field, to allow rapid searching for a particular key The basic operation here is not to get the "next" record, although that is also possible, but to get the record with a specific key. For the zoo file or Fig. 6-2(l). one could ask the system to get the recoid whos,e key is pony, for example, without worrying about its exact position in the file Furthermore, new records can be added to the file, with the operating system, and not the user, deciding where to place them. This type of file is clearly quite different from the unstruc- unstructured byte streams used in UNIX and Windows but is widely used on the large mainframe computers still used in some commercial data processing 6.1.3 File Types Many operating systems support several types ot files UNIX and Windows, for example, have regular files and directories UNIX also has character and block special files. Regular files are the ones that contain user information All the files of Fig. 6-2 are regular files. Directories, are system files for maintaining the structure of the file system. We will study directories below. Character special files are related to input/output and used to model serial I/O devices such as termi- terminals, printers, and networks Block special files are used to model disks. In this chapter we will be primarily interested in regular files Regular files are generally either ASCII files Or binary files. ASCII files con- consist of lines of text. In .some systems each line is terminated by a carnage return character. In others, the line feed character is used. Some systems (e.g , MS- DOS) use both. Lines need not all be of the same length. The great advantage of ASCII files is that they can be displayed and printed as is. and they can be edited with any text editor. Furthermore, if large numbers

 of programs use ASCII files for input and output, it is eayy to connect the output of one program to tbe input of another, as in shell pipelines. (The interprocess plumbing is not any easier, but interpreting the information certainly is if a stan- standard convention, such as ASCII, ts used for expressing it) Other files are binary files, whtch just means that they are not ASCII files. Listing them on the printer gives an incomprehensible listing full of what is apparently random junk. Lt,uillv tht\

have some intern I structure known to programs that use them For example, in Fig. 6-3(a) we see a simple executable binary file taken from a version of UNIX- Although technically the file is just a sequence of bytes, the operating system will only execute a file if it has the proper format. It has five sections- header, text, data, relocation bits, and symbol table. The header starts with ,1 so-called magic number, identifying the tile as an executable file (to prevent the accidental execution of a file not in this format). Then come the size* of the various pieces of the file, the address at which execution starts, and some flag bits. Following the header are the text and data of the program itself. These are loaded into memory and relocated using the relocation bits. The symbol table is used for debugging. Our second example Of a binary file is an archive, also from UNIX. It consists of a collection of library procedures (modules) compiled but not linked Each one is prefaced by a header telling its name, creation date, owner, protection code, and size. Just a^ with ihe executable file, the module headers are full of binary numbers. Copying them to the printer would produce complete gibberish. Every operating system must recognize at least one file type its own execut- executable file, but some recognize more. The old TOPS-20 system (for the DECsystem 20) went so far as to examine the creation time of any file to be executed. Then it located the source file and saw if the source had been modified since the binary was made. If it had been, it automatically recompiled the source. In UNIX terms, the make program had been built into the shell The file extensions were manda- mandatory so (he operating system could tell which binary program was derived from which source. Having strongly typed files like this causes problems whenever the user does anything that ihe system designers did not expect Consider, as an example, a sys- system in whieh program output files have extension dut (data files,) If a user writes a program formatter that reads a .c file (C program), transforms it (e.g . by con- converting it to a standard indentation layout), and then writes the transformed file as output, the output file will be of type -dm |f the us,er tries, to offer this to the C compiler to compile it, the system will refu.se because ii hat. the wrong extension. Attempts to copy file.dat to fde.c will be rejected by the system as invalid (to pro- protect the user against mistakes). While this kind of "user friendliness" may help novice*,, it drives experienced users up the wall since they have to devote considerable effort to circumventing the operating system's idea of what is reasonable and what is not.

r Mag.c number Text size Data size BSS Size Symbol lable size Entry pent Flags Text : Data Relocation bits Symbol | table Object Header Object module Header Object moduie Figure 6-3. (a) An executable tile (b) An archive. 6.1.4 File Access Early operating systems, provided only one kind of file access: sequential access. In these systems, a process could read all the bytes or records in a file in order, starting at the beginning, nut could noi skip around and read them out of order. Sequential files could be rewound, however, so they could be read as often as needed. Sequential files were convenient when the storage medium was, mag- magnetic tape, rather than disk. When disks came into use for storing files, it became possible to read the bytes or records of a file out of order, or lo access records by key. rather than by position. Files whose bytes or records can be read m any order are called random access files. They are required by many applications.

386 FILESYVTFMS C'HAIJ ticular flight, the reservation program must be able to access the record for that Two methods are used lor specifying where to start reading. !ii the tirst one. even, read operation gives the position in the file to start reading at. In the second seek, [fie file can be read sei|UL-ntMlly troiii the now-current position. En some older mainliame oper.riine systems, files are classified as being either sequential or random access at the time They are ucated. This aliows the system 10 use different stoiagc techniques lor the two classes Modern operating syMems 6.1.5 File Attributes Every file lias a n.mie and its data In addition, all operating systems associate other information with each file, for example, the date and time ihe trie was crealed and the file's si/e We will call these extra items the file's attributes The list of attributes vanes considerably from system to system The table <>l Fig. 6-4 shows some of the possibilities, but other tmes al*o exist N(, existing system has all or these, but each one is present in some system The first four attributes lelate to the iile s protection and tell who mav access it and who ntay not All kinds of schemes are possible, some of winch we will study later. In son>e systems the user must present u password to access a tile, m which case the password must be one ot the attributes The flags are bits or short fields that control or enable some specific prdpertv. Hidden files, tor exaniple. do not appear in listings' of all the files. The ,irchi\e flag is a bit that keeps tiack of whether the file has been backed up The backup program clears it. and lh*j opeiatinji system sets it whenever a file is changed. In this way, the backup program can tell which ides need backing up The tem- temporary flag allows a

fi|e to be marked for automatic deletion when the process that created it terminates The record length, key position, and key length fields are only present in files whose records can be looked up using a key Ihev provide the inlormation re- required to find the keys The various times keep track of when the file was created, most recently accessed and most recently modified. These are useful for a variety of purposes For example, a source tile ihat has, been modified after the creation of the corresponding object file needs to be recompiled These fields provide the neces- necessary information. The currem size tells how big the file is at present Some old mainframe oper- operating systems require the maximum si/e to be specified when the file is crealed, m order to let the operating system reserve the maximum amount ot storage m ad-

Archive (lag I ASCII/binary t!ag_ j o lor has been backed up; 1 for needs 10 be backed up : 0 for ASCII [Me 1 for binary file j 0 for sequential access only, 1 for random access Oforn. for locked_ i cord lo do 6.1.6 enough File Operations les exist to slcire mlormatton and allov provide differeni operations tt to he retn /ed later. Difiei itrteva]. Below .rage and re ussion of the most common system calls relating 10 Hies. 1. Create- The file is created with no data. The purpose of the call is to announce that the file is coming and to set some of the attributes- 2. Delete. When the file ts no longer needed, it has lo be deleted to free up disk space. There is always a .system call tor this purpose. 3. Open. Before using a file, a process ntusi open tt. The purpose of the open call n. to allow the system to fetch the attributes and list of disk addresses into main memory for rapid access on later calls.

S FILE SYSTEMS CHAP 4. Close. When ail the accesses, arc finished, the attributes and disk addresses are no longer needed, so the file should be closed lei free up iniemal table space. Many systems encourage this by imposing a maximum number of open files on processes. A disk is written in blocks, and closing a file forces writing of the file's last block, even though that block may not be entirely full yet. 5 Read. Daia aie read from file. Usually, the bytes come from the current position. The caller musi specify how much data are needed and must also provide a buffer to put them in. 6. Write. Data are written to the file, again, usually at the current posi- position. If the curreni position is the end of ihe file, the file's size increase!.. If the current position is in the middle of the file, existing daia are overwritten and lost forever. 7. Append. This call is a restricted form of wnte It can only add data lo the end of the file.

Systems that provide a minimal set of system calls do not generally have append, but many systems provide multi- multiple ways of doing the same thing, and these systems sometimes have append. 8. Seek. For random access files, a method is needed to specify from where lo take the data. One common approach is a system call, Seek, thai repositions the file pointer to a specific place in the file. After this call has completed, data can be read from, or written to. that position 9 Get attributes. Processes often need to read file attributes to do [heir work. For example, the UNIX make program is commonly used to manage software development projects consisting of many source files. When make is called, it examines the modification times ot all the source and object files and arranges tor the minimum number of compilations required to bring everything up to date To do its job. it must look at the attributes, namely, the modification times. 10. Set attributes. Some of the attributes are user sellable and can be changed after the file has been created. This system call makes lhat possible. The protection mode information is an obvious example Most of the flags also fall in this category. 11. Rename. It frequently happens that a user needs to change the name of an existing file This system call makes that possible li is not always strictly necessary, because the file can usually be copied to a new file with the new name, and the old file then deleted.

SEC. 6 I FILES 3W 6.1.7 An Example Program Using File System Calls In this section we will examine a simple UNIX program that copies one file from its source file to a destination file It ■& listed m Fig. 6-5 The program has minimal functionary and even worse error reporting, but it gives a reasonable idea ot how some of the system calls related to files work The program, copy file, can be called, lor example, by the command line copyhle abc xyz to copy the file «/?<■ to _u.-. If xyz ill ready exists, it will be overwritten. Otherwise, it will be created The program must be called with exactly two arguments, both legal file names. The four ^include statements near the top of the program cause a large number of definitions and function prototypes to be included in the program. These are needed to make the program conformant to the relevant international standards, but will not concern w further. The next line is a function prototype for main, something required by ANSI C. but also not important for our purposes. The first #define statement is a macro definition that defines the string BUF.SIZE as a macro that expands into the number 4096. The program will read and wnte in chunks of 4096 bytes It is considered good programming practice to give names to constants like this und to use the names

instead of the constants Not only does this convention make programs easier to read, but it also makes them easier to maintain The second #define statement determines who can ac- access the output file. The main program is1 called mam, and it has two arguments, urge, and <i/gi\ These are supplied by the operating system when the program is called "[he first one tells how many strings were present on the command line that invoked the program, including the program name. It should be 3. The second one is an array ot pointers to the arguments- In the example call given above, the elements of this array would contain pointers to the following values. argv[01 = "copyfile" argvt2] = "xyz" It is via this array that the program accesses its arguments Five variables are declared. The first two, W-fd and out_jd, will hold the file descriptors, small integers returned when a file is opened The next two, rd-count and *t_count, are the byre counts returned hy the read and write system calls, lespectively The last one. buffer, it, the buffer used to hold the data read and supply the data to be written The first actual statement checks a,g< to see it tt is 1 If not, it exits with status code 1. Any status code other than 0 means thai an error has occurred. The

FILE SYSTEMS /+ File copy program. Error checking and reporting is minimal V #mclude <sys/types.h? /* include necessary header files */ #include <fcntl.h> #include <stdlib.h> Sinclude <umstd.h> int main(int argc, char *argv[]); /• ANSI prototype +/ #define BUF_SI2E 4096 /* use a buffer size ot4Q96 bytes •/ #define OUTPUT.MODE 0700 /* protection bits for output file */ int mainjint argc, char *argv[]) { int in—fd, out_fd, rd count, wt_count, charbuffer[BUF SIZE]; if (argc !- 3) exitA), /* syntax error if argc is not 3 */ /* Open the input file and create the output file */ in fd = open(argv[1], O-RDONLV); /* open the source file */ it (in_fd < 0) exitB), /* if it cannot be opened, exit */ out_fd = creat(argv[2j, OUTPUT.MODE); /+ create the destination file */ if (out_fd < 0) exitC): /+ if it cannot be created, exit */ /* Copy loop */ while (TRUE) (rd_count = read(in ,fd, buffer. BUF SIZE), /* read a block of data */ if (rd_count <= 0) break; /* ,t end of file or error, exit loop */ wt_count = write(out,.fd, buffer, rd_count); /* Write data */ if (wt_count <= 0) exitD): /* wt_ count <= 0 is an error */ /* Close the files +/ dose(in_tdj; c|ose(out__fd); if (rd count == 0) /* no error on last n exit@); else exitE); /* error on last reaa Figure 6-5. A ,impk pmgi

SEC. 6] PILES 391 status code is the only error reporting present in tins

program A production ver- version would normally print error messages as. well. Then we try to open the source file and create the destination file. If the source file is successfully opened, the system assigns a small integer lo wjd, to identify the file Subsequent calls must include this integer so the system knows which tile it wants. Similarly, it the destination is successfully created, out_fd is given a value to identify il. The second argument to creat sets the protection mode. If either the open or the create fails, the corresponding file descriptor is set to -1. and the program exits with an error code. Now cornes the copy loop. It starts h> trying io read in 4 KB of data to buffer. It does this by calling the library procedure read, which actually invokes the read system call The first parameter identifies the file, the second gives the buffer, and the third tells how many bytes to read The value assigned to rd_count gives the number <>t bytes actually read. Normally, this* will be 4096. except d7 fewer bytes are remaining in the file. When end {,f file is reached, it will be 0 If rd count is ever zero or negative, the copying cannot continue, so the break state-statement is' executed to terminate the (otherwise endless) loop. The call (o wnte outputs the buffer to the destination file The first parameter identifies the file, the second gives the buffer, and the third tells Dow many bytes to wnte, analogous to it-ad Note that the byte count is the number of bytes actu- actually read, not BUF SIZE. Th[S point is important because the lust read will not return 4096, unless the file |Ust happens to be a multiple ot 4 KB When the entire file has been processed, the firsi call beyond the end of tile will return 0 to rd_c<ntnt. which will make it exit the loop. At this point the two files- are closed and the program exits with a status indicating normal termination. Although the Windows system calls are different from those ot UNIX, the general structure of a command-hue Windows progiam to copy a file is mod-moderately similar to that of Fig. 6-5. We will examine the Windows 2000 calls m Chap. 11 6.1.8 Memory-Mapped Files Many programmers feel that accessing files as shown above is cumbersome and inconvenient, especially when compared to accessing ordinary memory. For this reason, some operating systems, starting with MU1.TICS, have provided a way to map files mto the address space of a running pn.cess. Conceptually, we can imagine the existence of two new system calls, map and unmap The former gives a file name and a virtual address, whicli causes the operating system to map the Tile into the address space at the virtual address. For example, suppose that a file, {, ot length 64 KB, is mapped into the virtual address starting at address 5I2K. Then any machine instruction that reads the contents of the byte at 512K gets byte 0

of the file, and so on. Similarly, a write to address 5|2K + 21 (XX) modifies byte 21000 of the file. When the process

392 FILE SYSTEMS CHAP 6 terminates, the modified file ,s let! on the disk, jusl as though it had been changed by a combination of seek and write system calls. What actually happens it, lhat [he system's internal tables are changed to make the file become the backing store for ihe memory region 512K to 576K. Thus a read from 5I2K causes j page fault, bringing m page 0 of the file. Similarly, a write to 5I2K + I 100 causes a page tault, bringing in ihe page containing that address, after which the write to memory can lake place It thai page is ever evicted by ihe page replacement algorithm, it is written back lo the appropriate lilace m the file When the process finishes, all mapped, modified pages art- writ- written back to their tries. File mapping works best in a system that supports segmentation, hi such a system, each file can be mapped onto its own segment s,o that byte A: in the file is also byte k in the segment. In Fig. 6-6(a) we see a process that has two segments, text and data Suppose that this process copies files, like the program ot Fig 6-1. First it maps the source file, say, ahc, onto a segment. Then it creates an empty segment and maps it onto the destination file, \y:. in our example. These opera- operations give the situation shown in Fig 6-6(b|. Figure 6-6. (j) A -L'gn>enicd process helorc nuppin? files into ik jddn.. space (b) The process after mapping an exutinji file flhi inin (jiie segment .)e cradling Si new segment foi tile xy-. At this point the process can copy the s twice segment into the destin; ment using an ordinary copy loop No read or write system calls art When it is all done, it can execute the unmap system call to remove the i the address space and Chen exit The output tile, .tv;. will now exist, as had been created in the conventional way. Although file mapping eliminates the need for I/O and ihus makes mmg easier, it introduces a few problems of Us own First, it is hard fo tern to know the exact length of ihe output file, jcv?, m our example It (tell the number of the highest page wntlen. but it ha.s no way o(kno\ many bytes m that page were written Suppose that the program only ust £mo after execution all the bytes are still 0 (th^ir initial Viilue) ^^aybe xi consisting of 10 zeros. Maybe it is a file consisting of 100 zeros. M,i> file consisting of 1000 zeros. Who knows1; The operating system canno it can do is create a file whose length is equal to the page size. : needed, files from though it program- r the sys-

SLC HLfrS 393 A second problem can (potentially) occur if a file is mapped in by oi,e process and opened for conventional reading by another It the first process modifies a page, [hat change will not be reflected tn the file on disk unit) the page is evicted. The system has to take grept cart- to make sure the tw(, processes do not see inconsistent versions of the file. A third problem with mapping is thai a file may be larger than a segment, or even larger than the entire virtual address space The only way out is to arrange the map system call to be able to map a portion of a file, rather than the entire file. Although this works, it is clearly less satisfactory than mapping the entire file. 6.2 DIRECTORIES To keep track of files, file systems normally have directories or folders, which, m many sysiems. are themselves files In this section we will discuss directories, their organization, their properties, and the operations that can he per- 6.2.1 Single-Level Directory Systems The files Soi rung .ill it is called the rool directory, but since u is the the name does not matter mueh On early personal computers, this system was common, m pan because there was only one user Imeiestmgly enough, the world's first Miperuon>ptiter. ihe CDC 6600. also had only a single director) for all files, even though it was used by many users at oncv. This decision was no di.iibt made to keep the softwaie design simple. An example of a system with one direetoiy is given hi Pig 6-7. Here the directory contains foui tiles. The file owners are shown in the figure- not the file namt>\ (because the owners are important to the point we lMe about to make) The advantages of this scheme are its simplicity and the ability to locate files quickly—there h only one place to look, after all and<~ The problem with having only one directory m a system with multiple users is that different users may accidentally use the same iiames for [hen files Foi example if user A creates a file called mailbox, and then later user B also creates a tile called mailbox, B\ file will overwrite A\ file. Consequently, this scheme is

394 FILE SYSTEMS re. hut could be i not used on multiuser system*. an> more, hut could be used on a small embedded system, for example, a system in a car that was designed to store user profiles for a small number of drivers. 6.2.2 Two-level Directory Systems To avoid conflict caused by different users choosing the same file name f their own files, the next step up is giving each user a private directory In th way. names chosen by one user do not interfere with names chosen by a differc way. names chosen by one user do not interfere with names chosen by a differen usci and there it, no piobleni caused by the same name occurring in two or mon directories This design leads to the system

of Fig 6-8. This design could hi used, foi L-xiiinple. on a multiuser compuiLT or on a simple network of persona computers that shared a common file server over a local area netwtn k Implicit in this design is that when ,i user tries to o|ien a file, the ■ knows which user it is in order to know whicli directory to search As a lost basic form, users c<a slight extension is to allo- indication (,t wh(,se file i login name or identification, something nut system. When this system is implemented in i access iilcs in their own directories Howtv to access other users' files by providing so opened Thus, tor example. open(V) might be the call to open a file called.* in the User's dnectoiy. and open{"nancy/x") might be the call to open a tile l in the directory ot another user, Nancy One situation in which users need to access files other than their ow execute system binary programs Having copies of all the utility programs m each directory cleaily is inefficient- At the very least, there is a need foi tern directory with the execuiable binary programs

5EC 6 2 D1RECTOR1TS 3^5 6.2,3 Hierarchical Directory Systems The iwo-lcvcl hieraichy eliminates name conflicts among users bill is not satisfactory for users with a large number of files liven on a siflgle-us,er personal computer, it is inconvenient. It is iltnte common for users to want to group their files togethei in logical ua\s \ professor for example, might have a collection of files that together form a book that he is writing for one course, a second col-collection of tiles containing student programs submitted for another course, a third giou[i of files containing the code of an advanced compiler-writing system he is building, a fourth group of tiles containing grant proposals, as well as other files tor electronic mail, minutes of meetings papers he is writing, games, and so t,n. Some way k needed to group these files together in flexible ways chosen by the user What is needed is a general hierarchy (i e.. a tree of directories). With this approach, each user can have as many directories as are needed so that files' can be grouped together in natural ways This approach is shown in Fig. 6-9 Here, the directories A, B, and C contained in the root directory each belong to a dif- different user, two of whom have created subdirectories tor projects they are work- working on. The ability for users to create an arbitrary number of subdirectories provides a powerful siructuring tool for users to organize then work For this reason, neaily all modem hie systems aie organized in this manner 6.2.4 Path Names When the tile system is organized as a directory tree, some way is needed for specifying file name... Two different methods are commonly used In the iir,t method, eacl>

tile is given an absolute path name consisting of the path from the root directory lo the file. As an example, the path /mr/ast/mwlbex means that the

396 HLh SYSTEMS CHAP 0 root directory contains a suMirecmO- "w. which in lurn contains a subdirectory ait which contains the Tile mailbox. Absolute path names always start at the mot directory and are unique In UNIX the components ot the path are separated by / In Windows the separator is \ in MULTICS it was >. Thus the same path name would be written aK follows in these three systems Windows \usr\ast\mailbox UNIX /usr/ast/mailbox MULTICS >usr>ast>mailbox No matter which character is used, if the first character ot the path name is the separator, then the path is absolute The othei kind ot name is the relative path name. This is used in conjunc- conjunction with the concept of the working directory (also called the current direc- directory) A user can designate one directory as the currenl working directory, in which case ail path names not beginning at the rool directory are laken relative tii the working directory For example, it the current working directory is /nsr/dst. then the file whose absolute path is /usr/tul/mmlbox can be referenced simplj as mailbox In other words, the UNIX command cp /usr/ast/mailbox /usr/ast/mailbox.bak and the command cp mailbox mailbox bak do exactly the same thing if the working directory is /us t Ant The relative form is often more convenient, but it does the same thing us the absolute form. Some programs need to access a specific tile without regard to what the work- working directory is, In that case. the\ should jlways use absolute path names For example, a spelling checker might need to read hisi/hb/dh tionars w do its work It should use the full, absolute path name in this case because it does nut know what the working directory will be when it is called The absolute path name will always work, no matter what the working directory K Of course, if the spelling checker needs a large number ot tiles trom /u\r/lib an alternative approach is tor it to issue a system call to change its working direc- directory to /u\r/lib, and (hen use just dictionary as the first parameter to open. By explicitly changing the working directory, it knows tor sure wheic it !s in the directory tree, so it can then use relative paths Each process has its own working dnectory. s» when a process changes its woiking directory and Liter exits, no othej processes are affected and no traces of the change are left behind iti the file system lit this way it is always perfectly sate for a process to change its working directory whenever that is convenient. On the other hand, it a libnin prnt vduic changes the working directory and di>o.s not change back

to whcie it was when k is timshed, the rest of the program may not v\ork since its assumption about where it K may now suddenly be invalid. For

StX 6 2 DIRECTORIES 397 this, reason, library procedures rarely change the working directory, and when they must, they always change it back again before returning. Moit operating systems that support a hierarchical directory system have two special entries in every directory. " " and "..", generally pronounced "dot" and "dotdot." Dot refers to the current directory, dotdot refers to its parent. To see how these are used, onsider the UNIX file tree of Fig. 6-10 A certain process has Atsr/ust as its working directory. It can use to go up the tree. For example, it can copy the file lu\r/iihldicuonary to its own directory using the command cp /lib/dictionary The firsl path instructs the system to go upward (to the usi directory), then to go down to the directory lib to find the file dictionary. un ■*— Root directory Figure 6-10. A UNIX d.rewori iree Tho second argument (doi) names the current directory When the ,/> com- command gets a directory njme (including dot) as its last argument, it copies all the files there. Of course, a more normal w.iy to do the copy would be to type cp /usr/lib/dictionary

, . , , . , . ,,,,v ■, spcond rime Heic ihe use ofdotsavcsihe user the mwbleot typing J((/(wi«m a-eumd time. Nevertheless, typing cp /usr/lib/dictionary dictionary also works fine, as does cp /usr/lib/dictionary /usr/ast/dictionary All of these do exactly the same tiling 6.2.5 Directory Operations The allowed system calls for managing direct or its exhibit iwore variation Iroin s>stem to system tl>an system calls for files. To give an impression ot what they are and how they work, we will give j sample (taken from UNIX). 1 Create A directory is created It is empty except)ur dot and dotdot. by the mkdif program). 2 Delete A directory is deleted. Onl\ an empty directory can he deleted. A directory conUinnig only doc and dotdot is considered empty as. these cannot usually be deleted. 3. Opendir. Direct Dries can be read For example, to list all the tiles in a directory, a listing program opens the directory to read out the names of all the files it contains. Betore a directory can he read, it must be opened, analogous to opening and reading a file 4. Closedir. When a directory has been read, it should be closed to tree up internal table space. 5. Readdir. This call returns the next entiy in an open directory Form- call. but that approach has the disadvantage at forcing the progr.itn- iner to know and deal with the internal structure of directories, In contrast, readdir always

returns one entry in a siandaid format, no matter which of the possible directory structures is beir>£ used (>. Rename. In many respects, directories are |iist like files antl can be renamed the same way files can be 7 Link Linking is a technique that allows a file to appe.ir in more than one directory. Phis system call specifies an existing file and a path name, and creates a link from the oxisling file to the name specified by the path. In this way, the same file m.iy appear in muhiplc duec- lones A link ot this kind, which increments the counter m the file's i-node (to keep track of the number of directory entries containing the file), is sometimes called a hard link,

DIRECTORIES 8. Unlink. A directory entry is removed. If the file being unlinked is only present in t,ne directory (the normal case), it is removed from the file system.]f n is present in multiple directories. onK ihe path name specified is removed The others remain In I'MX, the system call for deleting file- (discussed carlici) n. in I act, unlink 6.3 FILE SYSTEM IMPLEMENTATION Now it is tune to turn from (he user's vieu ot the tile s>sit-in to the iin- plemcnior s \iew Llsets are conLcmed with hoiv files dvc named, whjit opera- lions <ire allowed on them, what the directoi> tree looks like, and similar interface issues [mplementors are mtcres-ted m how files and directories arts stored, how disk space i.s managed, and how to make everything work ftticienth ,ind reliably. In the following sections we will examine a number of these areas to see what the issues and trade-ofh are. 6.3.1 File System Layout File systems are stored ()n disks. Most disks can be divided up into one or more partitions, wiih independent file systems on each partuion Sector 0 ot the disk ts called the MBR (Master Boot Record) and is used to boot the computer. The end of the MBR contains the partition table This table gives the sorting and ending addresses ot each partition. One of the partitions m the table is marked ,is active. When the computer is booted, the BIOS reads in and executes the MUR The first thing the MBR program does i., locate the actis'e partition, read in its first block, called the boot block, and execute it. The ptoguim m the boot block loads the operating system contained in that partition For uniformity, every partition starts with a boot block, even if it does not contain a bootable operating system Besides it might contain one m the future, so reserving j boot block is a good strongly from file system to tile system. Olten the file system will contain some of the items shown in Hiy. 6-11. The first one is the s.uperblock It contains all the key parameters about the tile system and u, read mto memory when the com- computer is booted or the

tile system ts fust touched, lypic.il information m the superblock includes a magic number to identify the file system type, the number of blocks in the file system, and other ke> administuitive mlomiation Next might come information about free blocks m the file system, tor exam- example in the form of a bitmap or a !i_st of pointers. This might be followed by the i-

FILE SYSTEMS Bool block Super block Free space mgmi 1-nodes Rooi dir F1es.nd4ractc.nes nodes, an array of daca structures, one per file, telling all about the file. After th.it might come the root directory, which contains the top ol the file system tree. Finally, the remainder (>f the disk typically contains all the other directories Jtid 6.3,2 Implementing Files Probably the most important issue m implementing file storage is keeping track of which disk blocks go with which tile. Various methods arc used in dif- different operating systems. In this section, we will examine ,i few of them Contiguous Allocation The simplest allocation scheme is to store each file as a contiguous run of disk blocks. Thus on a disk with 1-KB blocks, a 50-KB file would be ,iI located 50 consecutive blocks With 2-KB blocks, it would be allocated 25 consecutive blocks. We see an example of contiguous storage ;iUocj|i0n in hg. 6-I2U) Here the first 40 disk blocks are shown, starting with Mock 0 on the left Initially, the disk was empty. Then a file 4, or length four blocks was written to disk starting at the beginning (block 0). After thai a siK-block file. B. w;is written starting right atter the end of lile A. Note that each tile begin*, <n the start of a new block, so that if Tile A was really 3V: blocks, some space is wasted <it ihe end of the last block. In the figure, a total of seven files are shown, each one starting at the block follow- following the end ot ihe previous one Shading is usiid |ust to make it easier to tell the files apart Contiguous disk spuue allocation has iwo sigmlicant advantages First, it is simple to implement because keeping irack ot where a file's blocks aie in reduced

SEC 6 3 FILE SYSTEM IMPLEMENTATION File C File E ocks) Fbl (Filed (FlIeE) (File G] I I I I I I I I I II I I I I I I I I I I I I I I I I I I I to remembering two numbers: the disk address ot the first block and the number of blocks in the file. G iven the number of the first block, the number of any other block can be found by a simple addition. Second, the read performance is excellent because the entire file can be read from the disk in a single operation. Only one seek is. needed (to the first block). After thai, no more seeks or rotational delays are needed so data come in at the full bandwidth of the disk Thus contiguous

allocation is simple to implement and has high performance. Unfortunately, contiguous, allocation also has a significant drawback: in time, the disk becomes fragmented. To see how this comes about, examine Fig. 6- I2(b). Here two files. D and F have been removed. When a file is removed, its blocks are freed, leaving a run ot free blocks on the disk. The disk is not com- compacted on the Spot to squeeze out the hole since thai would involve copying all the blocks, following the hole, potentially millions of blocks. As a result, the disk ulti- ultimately consists of files and holes, as illuslraied in the figure. Initially, this fragmentation is not a problem since each new file can be writ- written at the end of disk, following the previous one. However, eventually the disk will fill up and it will become necessary to enher compact the disk, which is prohibitively expensive, or lo reuse the free space m the holes. Reusing the space requires maintaining a hst of holes, which is doable However, when a new file is to be creaied, it is necessary to know its final size in order to choose a hole ot ihe correct size to place it in. Imagine the consequences of such a design. The user stans a text editor or word processor in order to type a document. The first thing the program asks is

402 HLh SYSTEMS rHAP 6 how man) hvies the tui.il dotument will lx* The question must be answered or the program will not continue It the mimbn given ultimately proves too small, the program has]o terminate prematurely because the disk hole is full and there is no pUce to put the rest oi the tile. It the Usd tries lo a\oid this problem b> giving an unre'ihsticallv larce number as the final size, say. 100 MB. the editor may lie . r i -h I ■ h 1 ■ ii c ■ ill-1 the (i)e ■ nni t he ere itt'd Of course, the user would be five to start tilt piogiam again and say 50 MB this time, and so on until a suitable hole was located. Still this scheme is not likely to lead to happv users However, there is one situation in which umttguous allocation is feasible and. in fact, widely used on CD-ROMs Heie all the file si/es aie known m advance will study the most common CD-ROM file system later in this chapter As we menlioned in Chap 1, histor> olten repeats itself in computer suture as new generations of technology occur Contiguous allocation was aetualK u>,ed on magnetle disk file systems years ago due to its simplicity anJ Ingh perfor- performance (user I r tend] mess did nut count for itilicIi ihcn) Then the idea was dropped due to the nuisance of having to specify final file size at iile creation tune Bm with the advent of CD-ROMs. DVDs, and en her wnte-once optical media, sud- suddenly contiguous files are a good idea again It is, mus important to study old sys- systems anJ

ideas thai were conceptually clean and simple because they may be applicable to future systems m surprising wavs. Linked List Allocation The second method tor stoitng files is to keep each one as a l,nkcd lust of disk blocks. as shown m Fig. 6-13. The first word of each block is Used as a pointer lo the next one. The rest of the block is for data Unlike contiguous allocation, every disk block can be used m this method. No space is lost lo disk fragments ion (uxcepi lor internal fragmentation m the la>t block). Also, it is sufficient for the directory entry to merely stoie the disk address of the first block. The rest can be found starting there. On the other hand, although reading a file sequentially is straightforward ran- dom access is extremely slow To get to blotk ir* the operating svsteni has to styrt at the beginning and read the n - I blocks pi lor m tt. one at a time Clearly, doin^t so many reads will be painfully slow. Also, the amount of data storage itt a block ts no longer a jiOvver o(two because the pointer t^kes tip a few bytes- White not fatal, having a peculiar size ls power of l\vo. With the first lew bytes ol each block occupied t(> a pointer to the next block, reads of the full block size require acquiring and concatenating infor- information from two disk blocks, which generates extra overhead due to the copying.

F1LF. SYSTEM IMPLEMENTATION Figure 6-13. Stonuc a tTSc j^ a hnked hs[iti disk Mocks Linked List Allocation Using a Table in Memory Both disadvantages, of ihe linked list allocation can be eliminated by taking ihe pointer word from each disk block and putting it in a table in memory. Figure 6-14 shows whai the table looks like for the example ot Fig 6-13. In both figures, we have two riles. File A use.s disk blocks 4, 7, 2, 10, and 12, m that order, and tile Buses disk blocks 6. 3, I Land 14, in that order Usiitg the table ot Fig- 6-14, we can start with block 4 and follow the chain all the way to ihe end The same can be done starting with block 6. B(,th chains are terminated with ;i special marker (e.g., -1} that is not a valid block number Such a table in main memory is called a FAT (File Allocation Table) Using this organization, the entire block is available tor data. Furthermore, random access is much easier. Although the chain must still be followed to find a given offset within the file, the chain is entirely in memory, so it can be followed without making any disk references- Like the previous method, it is sufficient for Ihe directory entry to keep a single integer (the starting block number) and still be able to locate all the blocks, no matter how large the file- is The primary disadvantage ol this method is that the entire table must be in memory alt the time to make tt

work. With a 20-GB disk and a 1-KB block size, the table needs 20 million entries, one for each t,i ihe 21) million disk blocks Each entry has to be a minimum of 3 bytes bar speed in lookup, they should be 4 bytes. Thus the table will lake up 60 MB or 80 MB of main memory all the time, depending on whether the system is optimized for space or time Conceivably the table could be put m pageable memory, but it would still occupy a greal deal ot virtual memory and disk space as well as generating extra paging traffic.

FILE SYSTEMS upies n bytes and a maximum of*, files may be open at once, ihe l ry cupied by the array holding ihe i-nodcs for the open files is only kn bytes. Only this much space need be reserved in advance. This array is usually far smaller than the space occupied by the tile table ous sectio The easn m e proble with i-nodes . .-)GB. ^^ii^- jjujfi^-in v>mi i-11<,n_n,*, ii that if each one lias room lor a tixtu uuinua v>i disk addresses,, what happens when a file grows bevond thi<. limn? One solution is to reserve the last disk address not for a data block, but instead tor the address

FILE SYSTEM IMPLEMENTATION File Attributes Afldres Addres. Addres Addres Addres Addres Addres Addres old Old old sold sotd sold sold sold Address of bloc kblockO sk block 1 k block 2 sk block 3 sk block 4 sk block 5 sk block S sk block 7 k of oo mtets 1 *■ ^ Disk block containing additional Figure 6-15. An t'\jmp]c]-n<Kle of a block containing m(,re disk block addresses, as s>hown in Fig. 6-15. Even more advanced would be two or more such blocks coniaining disk addresses or even disk blocks pointing to other disk blocks full ot addresses We will cAme back to i-nodes when studying UNIX later 6.3.3 Implementing Directories Be/ore a file can be read, it must be opened When a file is opened, the operating system uses the path name supplied by the user to locate the direciory entry. The directory entry provides the information needed to find the disk blocks. Depending on the system, this information may be ihe disk address. i'f tlie entire tlie (contiguous allocation), the number of the first block (both linked list schemes), or the number of the i-node. In all cases, ihe mam function of the directory system in to map the ASCII name ot the tile onto the information nee.Ied to locate the data A closely related issue r_s where the attributes, should be stored. Ei/ery file they must be stored somewhere. One obvious possibility is to store them directly in ihe directory entry Many systems do precisely that- This option i^ shown in Fig 6-16|a) in this simple design, a direciory consists of a list of

ftxed-stze

406 FILE SYSTEMS CHAP <> ot the fill- attributes, and one or more disk addresses (up to some maximum) telling when- the disk blocks are. -□ -□ For systems, thai use i-nodes, another possibility toi storing the attributes is iii the i-nodes, rather than m tht directory entries In that case, the directory entry can be shorter- just a file name and an i-nocJe number Tins approach is illustrated in Fig. 6-I6(b). As we sliall see later, tlnri method has certain advantages over putting them in the directory enlry. The two approaches shown in Fig 6-16 correspond to MS-DOS/Windows and UNIX, respectively, as we will see later in thi.s So far IS-DOS tiles have a 1-8 character base name and an optional extension or 1-3 characters In UNIX Version 7. file names were |-|4 characters, including any extension- However, nearly all modern operating systems support longci. variable-length file- filenames How can these be implemented1' The simplest approach is to sei a limn on file name length, typically 255 char- characters, and then use one of the designs of Fig 6-16 with 255 characters reserved for cadi file name. This approach rs simple, but wastes a great deal of directory structure is desirabfe. One alternative is to give up the idea thai all directory entries are the saim.- id tli ;d h size. With th starting with the length of the entry, mat. usually including the owner, eft attributes. This rixed-tength header iong it may be, as shown in Fig. 6-1 this example we have three files, project-budget, personnel, and Joa Each iik name is terminated by a special character (usually 0), which is represented in (he fixed portu ollowed by data with , protection intot'inatio ved by the actual tile name, ho> '(a) in big-eitdian format (eg. SPARC). Jn

SEC ,, i FlLfc SYSlbM IMPLEMENTATION 407 %ure by a bo, with a cross in ,1 To allow each directory entry It. begin on a wi.rd buunclary, each file name is filled out to an integral number o\ words, shown by shaded boxes in the figure F e1 en F»l. p e b e c u t Fne a en File 2 a P o 1 e n File 3 en File 3 a f o y length nbutes 0 t d IS! 1 g V tength nbufes ' I 5 n I e ry leng ttnbujes o M - POI ertol File i a Pomfer to die 2's name - Ftle 2 a s Pomier lo I F,»3. nbules e 3 s name ttnbuies P e \n~~~ r 1 o —~— t ~n~ f 'A s e 0 I -— p 0 1 0 (a) ,b) figure 6-17. Tw() ways of handling lolij! lilc names m j (liiCL[,)r> ui) In line (b) In j heap 11 disadvyiita&G ot this n>ethtid ts that ^vhen ,i tile is reiiioved, a varicible~si"od gap is introduced mlo tlie directory into which the next hie to be entered may not Tit. This problem is the same une we saw with

contiguous disk filcs. only now compacting the directory is feasible because it is entirely in memory Another problem is that a single directory entry may span multiple pages, so a page fault may occur while reading a file name Another way id handle variablc-letiglh names is to nuiki? ihc directory entries themselves all fixed length and keep the file names together in a heap ;il the end of the directory, as shown in Fig. 6-17(b>). Tins method has the advantage that when an eniry is removed, thu next file entered will always tit there Ot course, the heap must be managed and page faults can still occur while processing Hie names- One minor win here is that there is no longer any real need tor file names to begm at word boundaries, so no filler characters arc needed after file names in Fig. 6-17(b] and they are in Fig. 6-]7(a). In all of the designs so tar. directories, are searched linearly trom beginning to end when a file name has to be looked up For exiremely long directories,, linear

408 HLE SYSTEMS CHAP 6 searching can be slow. One way to speed up the search is. lo use a hash table in each directory. Call the si?e of the table ,i. To enter a file name, the name is hashed onto a value between 0 and « - L for example, by dividing it by n and tak- taking the remainder. Alternately, the words comprising the file name can be added up and this quantity divided by n, or something similar hither way. the lable eniry corresponding to the hash code is inspected If n is unused, a pointer is placed there lo the rile entry- File eiitrtes roliow the hiisti table Ifihat slot is already in use. J linked list is constructed, headed at that table entry and threading through all entries with the same hash value. Looking up a file follows the same procedure The ttle name is hashed to select a hash table entry. All the entries on the chain headed at that slot are checked to see if the file name is present. It the name is no! on the chain, (he file is, not present in the directory Using a hash table has the advantage of much taster lookup, but tlie disadvan- disadvantage t>f a more complex administration, [t is only really a serious candidate in systems where it is expected that directories will routinely contain hundreds or thousands of files. A completely different way to speed up searching large directories is to cache the results of searches. Before starting a search, a check is first made to see if [lie file name is in Che cache It so. it can be located quickly, avoiding a long lookup Of course, caching only works if a relatively small number of files comprise the majority of the lookups. 6.3.4 Shared Files files. Ah a result, it is often convemeni for a shared tilt to appear simultaneously in different directories belonging to different users Figure 6-18 shows the file system of

Fig. fi-9 again, only with one ot ("s files now present in one of B\ directories as well. The connection between B\ directory and the shared file is called a link. The ttle system iiself is now a Directed Acyclic Graph, or 1>AG. rather than a tree Sharing files is convenient, bui it also introduces some pioblems To sttirt with, if directories really do contain disk addresses, then a i.opy of the disk addresses will have to be made in B's directory when the file is linked It either B or C subsequently appends to the file, the new blocks will be listed only in the directory of the user doing the append The changes will not be visible to the other user, thus defeating the purpose of sharing. This problem can be solved m two ways In the first solution, disk blocks arc not listed in directories, but m a little data structure associated with the file itself. The directories would then point just to (he little data structure. This is the approach used in UNIX (where the little data strocturc is the i-node).

FILE SYSTEM IMPLEMBNTATION ~| Root directory Figure 6 .. File sy.it tuning .1 shared f>le In the second solution, B links to one ot C's files by having the syste a new file, oftype LINK, and entering that file in B\\\ directory The new tains just the path name of the file to which it is linked. When B reads linked file, the operating system sees that the file being read trom is LINK, looks up the name of the tile, and reads thai file. This approach symbolic linking. Each of these methods has its drawbacks. In the first method, at the that B links to the shared file, the i-node records the file's owner as C C [ink does not change the ownership (see Fig 6-19), but n does incre count in the i-node. so the system knows hc>w many directory entri point to the tile. am the if type called real ing .1 link mtly C's directory B's directory C's

410 FILE SY STEMS CH \P (> It C subsequently tries to remove the file, the system is tami with a problem It it removes the file and dears the i-node, B will have a directory entr; pointing to an invalid i-node It the i-node is later reassigned lo another file. B\ link will point to the wrong tile The sjstem can see fiom ihe count in the i-mxle that tilt file is still in use, but there is no way tor it to find all the directory entries tor the file in older to erase them. Pointers to the direi lories cannot be stored in the i node because there can be an unlimited numbei of directories The onlv thing to do i^ remove t s directory entr\. but leave tht i~nodc intact witli count set lo I, as shown in Fig 6- I9(c) We now have a situation in which H accounting or has quotas. C will continue to be billed tor the die until B decides to remove if. it evci, at which time the

count goes to I) and the file is deleted With s>mbohc links this problem does not arise because only tht* hue ownei has a pomtet to the i-oode. Users who have linked to the Tile just have path names, not i-node pointeis When the owner lemoves the file, n is destroyed Subsequent attempts to use the file via a svmbotic link will fail when the system is unable to locate the file Removing a symbolic link does not affect the lile at all The problem with symbolic links is the e\trJ overhead required. The Tile con- containing the path must be read, then the path ntust be parsed and followed, com ponent by component, until the i-node is reached All of this activity may require a considerable number ol extra disk accesses furthermore, an extra i-node il> needed for each symbolic link, as is yn extra disk block to store the path, although if the path name is short, the system could store it in the i-node itself, as an optim- optimization. Symbolic links have the advantage thai they can be used to link to dies on machines anywhere in the world, by simpl> piovidmg the network address o1 the machine where the file resides in addition to its path on that machine. There is also another problem introduced by links, symbolic or otherwise When links are allowed, files can have two or more paths. Programs thai start at ,i locale a linked file multiple times. For example, a program that dumps all the files in a directory and its subdirectories onto a tape may make multiple copies ol a linked file Furthermore, if the tape k tht-n tend into another machine, unless the dump program is cle\er, the linked (ile will be copied twice onto :he disk instead ot being linked. 6.3.5 Disk Space Management concern to file system designers Two general strategies ar<? possible lor storing an a byte file it consecutive bytes of disk space are allocated, or the till.- is split up into a number or {not nt-cessardy) contiguous blocks The same trade-off is present in memory management systems between pure segmentation and paging

SFC <■< 1 PILE SYSTHM IMPLEMENTATION 411 As we have seen, storing a file as a contiguous sequence of bytes has the obvious problem that if a file grows, it will probably have to be moved on the disk. Thy s.ime problem holds for segment'; in memory, except that moving a seg- segment in memory is a relatively fast opeiation compared to moving a file from one d,sk position to another For this reason, nearly all file systems chop files' up into fixed-size blocks that need not be adjacent Block Size Once it has been decided to »tore files in fixed-size blocks, the question arises of ho* big the block should be. Given the way disks are organized, the sector, the track and the cylinder are obvious candidates for the unit of allocation (although these are all device

dependent, which is a minus) In a paging system, the page size is also a major contender. Having a large allocation unit, such as a cylinder, means that every Tile, even a I-byte file, lies up an entire cylinder Studies (Mullender and Tanenbaum. 1984) have shown thai the median file sL?e in UNIX environments is about 1 KB, so allocating a 32-KB block for each lile would waste 31/32 or 97 percent of the total disk space. On the other hand, using a small allocation unit means tliat each file will con- consist of many blocks. Reading each block normally requires a seek and a rotational delay, so reading a file consisting of many small blocks will be slow As an example, consider a disk with 131.072 bytes per track, a rotation time of 8.33 msec, and an average seek time of 10 msec. The time in milliseconds to read a block of i bytes is then the sum of the seek, rotational delay, and transfer times 10 + 4 !65 + (t/niO72)x8.33 The solid curve of Fig 6-20 shows the data rate for such a disk as a function o(block size To compute the space efficient:}, we need to make an assumption about the mean file size A recent measurement at the author's department, which has 1000 users and over 1 million UNIX disk files, gives a median <;ize of 1680 bytes, meaning that half the files arc smaller than 1680 bytes and hdlf are larger. As an aside, the median is a better metric Irian the mean because a verv sm&ll number of tiles can influence the mean enormously, but not the median. (The mean is in fact 10,845 bytes, due in part to a few @0 MB hardware manuals tfiat happen tt, be online) For simplicity, let us assume all tiles are 2 KB, which leads to the dashed curve in Fig 6-20 for the d,sk space efficiency. The two curves can be understood as follows The access time for j block is completely dominated by the seek tune and rotational delay, so given that it is going to cost 14 msec to access a block, the more data that are fetched, the better. Hence the data rate goes up with block si7e (until the transfers take so long thai

FILE SYSTEMS ize (bytes) I1 igurc 6-20, The ^I'ltd cjrvu jitf|-h3nd scjiIc) £tvcs Ihc dJId r*il^ ot £1 disk 1 he iid^hcil furvc (rt^ht-hiiml >cdlc) ^iv^s (}1£ diNk spjee efficiency All hlCi ""^ 2 KB ihe transfer time begins to dominate) Wuh small blocks ihat me powers of two and 2-KB file:,, no .space is wasted in a block However, with 2-KB files and 4 KB or larger blocks, some disk space is wasted In reality, few (lies are a multiple of the disk block si?e, so som^ space is always wasted in the last block of a file What the curves show, however, is thai performance and space utilisation ;ire inherently in conflict. Small blocks are bad for performance but good for disk space utilization. A.

compromise size is needed. For this data. 4 KB might be a good choice, but some operating systems made their chimes a long lime ago. when the disk parameters and file sizes were different For UNIX. 1 KB is com- commonly used. For MS-DOS the block size can be ,my power of two front ^ 1 ^ bvtes to 32 KB, but is determined by the disk size and for icasons unrelaled to these arguments (the maximum number of blocks on a disk partition is 2lft. which forces large blocks on large disks) In an experiment to see if Windows NT file usage was appreciably ditlerent from UNIX file usage. Vogels made measurements on files at Cornell University (Vogels, 1999). He observed that NT file usage is more complicated than an UN|X. He wrote: »rncit we tvpc (i /^^ t'ti{ir(iwtj i~s in t/tc no ten,id ic xi fill tor sivwyji? tin \ it) file will trigger 26 sysiem <alh. imludm^ 3 faded open uitewpis. I /i!e overwrite end 4 additional open and (/a \e \f quetn r * Neverthelej,-,, he observed a median size (weighed by usage) of files iust read at I KB, files just written as 2.3 KB and files rea.t and written as 4 2 KB Given thy fact that Cornel) has much more large-scale scientific computing than tht author'1- institution and the difference in measurement technique (static versus dynamic), ihe results arc reasonably consistent with a median file size of around 2 KB.

SEC 6 1 FTI F SYSTEM IMPLEMENTATION Keeping Track of Free Blocks Onc blocks ick of ti thods are widely used, as- shown in Fig 6-21 The first one con- consists or using a linked list of disk blocks, with each block holding as many free disk block numbers as will fit. With a I-KB block and a 32-bit disk block number, each block on the free list holds the numbers of 255 free blocks, (One slot is needed for the pointer to the next block) A 16-GB disk needs a free list of maximum 16.794 blocks- to hold all 224 disk block numbers. Often free blocks are used to hold the tree list. 42 136 210 97 41 63 21 48 262 310 516 hre edi r k blocks 16 230 162 612 342 214 160 664 216 320 180 482 Ib 86 234 897 422 140 £23 323 160 126 142 141 01O11O11O11OH0 000011101101011 100100011101111 Figure 6-2t. (j)Snirmg the The other free space man age me blocks requires a bitmap wi[h n hit- map, allocated blocks by 0s (or vice and thus requires 224 bits for the r ,urprising thai the bitmap requi ree li.t on ,\ linked Mm (b) A bnirup it technique is the bitmap. A disk wiih n Free blocks are represented by Is in the /ersa) A 16-GB disk has 224 1-KB blocks ap. which requires 2048 blocks It is not 1 bit per block, versus 32 bits in the linked list model Only il the disk is nearly full (blocks) will the linked list scheme require fewer blocks than the bitmap On the other hand, if

there are many blocks free, some of them can he borrowed to hold (lie free list without any loss (if disk capacity. When the free list method is used, only one block of pointers need be kept in mam memory. When a file is created, the needed block* are taken from the block

FILE SYM EMS of pointers- When it runs oui, a new block or pointers is read in from the disk Similarly, when a file is deleted, its blocks die freed and added to the block ot pointers in mam memory When this block (ills up. it is written to disk. Under certain circumstances, this method leads to unnecessary disk I/O Con- Consider the situation of Fig. 6-22(a). in which the block of pointers in memory has sider the ronly ■rflows and ha three-block file VJ*~r» (n i'uiui^i i> hi iii^ii'.-'.j .... . . _ _,. If a three-block file is freed, the pointer block written to disk, leading to the situation of Fig 6-22(b). If u uiree-[)ioLK me is nuw written, the full block of pointers has to be read in again, taking us hack to Fig. 6-22(a) If the three-block file just written was a temporary '■'- -'- en it is freed, another disk write is needed to write the full block of back tn the disk In short, when the hlock of pointers is almost empty, a t shorl-hved temporary files can cause a lot ot disk I/O. file, pointers Figure 6-22. (a) An jlmosl-lull block at pointers in Iree disk blocks in memory Jnd ihree blocks ot pointers on disk, (b) Result iif Irecing ,1 three block tile (O An jlternalive strategy tor handling the three free blocks The sh;i[led entries represent pointers to free disk blocks An alternative approach that avoids most of this disk 1/0 is to split the full block of pointers. Thus instead of going from Fig 6-22(j) to Fig. 6-22(b), we go from Fig. 6-22(a) to Fig 6-22<c) when three blocks are freed. Now the system can handle a series of temporary files without doing any disk I/O If the block in memory fills up. it is written to the di.sk and the half-full block from disk is read in. The idea here is to keep most of the pointer blocks on disk full (to minimize disk usage), but keep the one in memory about half full, so it can handle both file creation and file removal without disk I/O on the free hst With a bitmap, it is al*o possible to keepjua one block in memory, going to disk for another one only when it becomes full or empty. An additional benetu of this approach is that by doing all the allocation from a single hlock of the bitmap, (he disk blocks will be close together, thus minimizing disk arm motion. Since the bitmap is a fixed-size data structure, it the kernel is (partially) paged, the bit- bitmap can be put in virtual memory and have pages of it paged in as needed

SEC 6 } FILE SYSTEM IMPLEMENTATION 4'5 Disk Quotas To prevent people from hogging tot) much disk space, multiuser operating systems oiten provide a mechanism for enforcing disk quoias The idea is that the system administrator assigns each user a maximum allotment of files and blocks, and the operating system makes sure that the users do noi exceed their quotas. A typical mechanism is described below. When a user opens a file, the attributes and disk addresses are located and put into an open file table in main memoiy. Among the attributes is an emry telling who the owner is. AnV increases in the file s si/e will be charged to the owner s file. it the |ik- or every user with a currently open else This table is shown in Fig. 6- ■r ihe users whose files are currently jpon. When all the files are closed, the record is written bdek to the quota file opened by s Open tile table Attributes disk addresses User = 8 Soft block limit Hard Curren # Block Soft Hat Curre *Rle* Jloek limit i of blocks warnings left tile nmil file limit t # of files anMoge left Figur c kept tratk n Whe madt- i open tile i.ible a il easy u> find tK<_- o the owner a ne-w quota record is entered into time a block is added lo a til incremented, and a check is made againsi both the hard and soil limits. The soft limit may be exceeded, but the hard limit may not An attempt to append to a file when the hard block limit has been reached will resuh in an error. Analogous checks also exisi for the number of files. When a user attempts lo log in, ihe sysicm checks the quota file to see it ihe user has exceeded the sofi limit for either number of files or number of disk blocks. If either limit has been violated, a warning is displayed, and the count or

 warnings remaming is reduced by one]f the count ever gets to zero, the user has ignored the warning one timts too many, and is noi permitted to log m. Getting permission to log in again will require some discussion with the system adminis- administrator This method has the property that users may go above their soft limit* during a login session, provided they remove the excess before logging out The hard lunits may never be exceeded. 6.3.6 File System Reliability Destruction ot a file system is often a far greater disaster than destruction u(a computer If a computer is destroyed by fire, lightning surges, or a cup of Coffee pouied onto the keyboard, it ts annoying and will cost money, bill generally a replacement can be purchased with a minimum of fuss. Inexpensive personal compulers can even be replaced within an hour by just going to ihe dealer (except at universities, where issuing a purchase order takes three committees, five signa- signatures, and 90 days)

software, or rats gnawing on the floppy disks, restoring all the mformaiion will be difficult, time consumtng, and m many cases, impossible Foi the people whose programs, documents, customer files, tax records, databases, marketing plans, or other data are gone forever, the consequences can be catastrophic. While ihe file system cannot offer any protection against physical destruction ot the equipment and media, it can help protect the information. In this section we will look :it some of the issues involved in safeguarding the file system Floppy disks are generally perfect when they leave the factory, bur ihey tan develop had blocks during Use Hard disks Irequcntly h.ive had blocks right from the start: it is just too expensive to manufacture them completely free <>f all defect;,. As we saw in Chap 5, bad blocks are generally handled by ihe controller by replacing had sectors wiih spares provided for that purpose. Despite th)s tech- technique, there are other reliability issues winch we will consider below Backups Most people do not think making backups of their filei is worth the time and effort—until one fine day their disk abruptly dies, at which lime most o' rhum undergo a deathbed conversion Companies, however, (usually) well understand the value oi their data and generally do a backup at le,ist once a day usually to tape Modern tapes hold lens or sometimes even hundreds of gigabytes and cos! pennies per gigabyte. Nevertheless, Making backups is not uuife as trivial as it sounds, so we will examine some of the related issues below

SEC 6 1 TILF SY-STFM |MPI EMENTATION 417 Backups lo lape are generally made to handle one of two potential problems. 1. Recover from disaster. 2. Recover from stupidity. The fust one covers getting the computer running again after a disk crash, fire, flood, or other natural catastrophe. In practice, these things do not happen very often, winch is why many people do not bother with backups. These people also lend not to have fire insurance on their houses for the same reason The second reason is that users often accidentally remove files that they later need again This problem occurs so often that when a file is "removed" in Win- Windows, ii is not deleted at all, but just moved to a special directory, the recycle bin, so ii can be fished out and restored easily later. Backups take this principle further and allow files thai were removed days, even weeks ago. to be restored from old backup tapes Making a backup takes a long time and occupies a largu amount of space, so doing it efficiently and conveniently is important. These considerations Kiise the following issues. First, should the entire file system be backed up or onl\

part of if Al many installations, the executable (binary) programs arc kept nt a limited part of the tile system tree It is not necessaiy to back up these files it ihey can a|| be reinstalled from the manufacturers' CD-ROMs Also, most systems have a directory for temporary files There k usually no reason to back it up either. In UNIX, all the special fiiw, (I/O devices) are kept in a dircc-iory /dev Not only is backing up this directory not necessary, it is downright dangerous because fhe backup program would hang forever if it tried lo read each of these to completion- In short, it is usually desirable to back up only specific directories and everything in them rather than the entire file system. Second, it is wasteful io back up files thai have not changed since the last backup, which leads to the idea of incremental clumps. The simplest form of incremental dumping is to make a complete dump (backup) periodically, say weekly or monthly and to make 3 daily dump of only those tiles that have been modified since the last full dum|i Even better is io dump only those file*, that have changed since they weie last dumped. While this scheme minimizes dump- dumping time, it makes recovery more complicated because first the most recent full dump has to be restored, followed by all the incremental dumps in reverse order. T very, more sophisticated incremental Jumping schemes aie often >cd. Third, since immense a l th d b dered.

418 HLLStSL'EMS CHAP 6 directories are being added, deleted, and modified during ihc dumping process, (he resulting dump nia% be inconsistent. However, since making a dump may lake hours, it may be necessary to take the system offline fol much or the night to make the backup, something that is not always acceptable. For this reason, algo- algorithms have been devised for making rapid snapshots of the file system state b> copying critical datj structures, and then requiring future changes to tiles and directories to copy the blocks mvtead of updating ihem in place (Hutchison c-t al . 1990). In this, way. the file system is effectively frozen at the moment o(the- snapshot, so it can be backed up at leisure afterward. Fifth and last, making backups introduces many nontechnical problems into an organization The best online security system in the world may be useless if the system administrator keeps all the backup tapes in his office and leaves it open and unguarded whenevei he walks down the hall to gel output from the printer All a spy has to do is pop in for <i second, put one tiny tape in his pocket, and saunter off jauntily. Goodbye security Also, making a daily bac'kup has little use it the fire that burns down the computers also burns up all the backup tapes For this reason, backup tapes should be kept

off-gte. but that introduces mare seeurhy risks. For a thorough discussion of these and other practical adminisuation issues, see (Nemeth et al., 2000). Below we will discuss only the technical issues involved in makmg Tile system backups. Two strategies can be used for dumping a disk to tape, a physical dump or a logical dump. A physical dump starts at block 0 ol the disk, writes all the disk blocks onto the output tape in order, and stops when ii has copied the last one Such a program is so simple thai it can piobably be made 10O bug tree, some- something that can piobably not be said about any other Useful program Nevertheless, it is worth making seveial comments about physical dumping For one thing, there k no value in backing up unused disk blocks. II the dumping program can get access to the free block data structure, it can avoid dumping unused blocks. However, skipping unuseti blocks requires writing the number of each block in front of tlit- block (or the equnaietit). since it is no longer true that block k on Ihe tape was block k i)n the disk. A second concern ,s dumping bad blocks It all bud blocks are remapped h> the disk controller and hidden from the operating system as we described in Sec 5.4.4. physical dumping works' fine On the other hand. if fhey are visible to the operating system and mtimiaincd m one or more "bad block hies' oi bitmaps. It is absolutely essential that the physical dumping program get access to Ihts informa- information and avoid dumping them to prevent endless disk read enors during the dump- dumping process The main advantages ol physical dumping are simplicity and great speed (basically, it can run at the speed ot the disk)- The main disadvantages are ihc inability to skip selected directories, make incremental dumps, and restore indivi- individual files upon request, hor these reasons, most installations make logical dumps.

SEC 6 "i FILE SYSTEM IMPLEMENTATION 41 •» A logical dump Mart1; <it one of more specified directories and recursively dumps all files and directories found there that have changed since same given base dale (e.g., the last backup lor an incremental dump or system installation for a full dump) Thus in a logical dump, the dump tape get:, a series (if carefully identified directories Slid files, which makes it easy to restore a specific tile or directory upon request Since logical dumping is the most common form, let us examine a common algorithm m detail using the example or Fig. 6-24 to guide us. Most UNIX sys- systems use this algorithm In the figure we- see a ti|e tree with directories (squares) and files (circles) The shaded iiems have been modified

since the base date and thus nc-ed to be dumped The unshaded ones do not need to be dumped Root dtrectory yslem 10 be dumped The • shaded item?* hjve been : is labeled by ils i-nixie nu oilil'ied since the lasi s (even unmodified o reasons. First, to to a fresh file syste e piograms can be Figure 6-24. , circles dre file fcaeh direiiorv This algorithm also dumps all directon the path to a modified file or directory for t to restore the dumped files" and directories compuier. In this way. the dump and resto eniire file systems between computers The second reason for dumping unmodified directories abov s to make it possible to incrementally restore a single file (possibly to handle recovery from stupidity). Suppose thai a full file system dump is done Sund.iy evening and an incremental dump is done on Monday evening. On Tuesday the directory Atsr/jhs/proj/nrJ is removed, along wuh all the directories and file,-, under it. On Wednesday morning bright and early the user wants (o restore the file Aisr//hs/prof/nr3/pians/summar\ However, is, not possible to jusi restore the ones} thai lie on make it possible On a different sed to transport odified files

420 FILE SYSTEMS CHAP 6 file \umman because there is no place la put il. The directories nr] and plum must be restored first. To get iheir owners, modes, times, etc . correct, these directories muM be present on the dump tape even though they themselves were nol modified since the previous full dump The dump algorithm maintains a bitmap indexed by i-node number with several bits per i-node Bits will be set and cleared in this map as the algorithm proceeds The algorithm operates in four phases. Phase I begins at the starling directory (the root in this example) and examines all the entries in it. For each modified tile, its i-nudt> is marked in the bitmap. Each directory is also marked (whether or nol i! has been modified) and then recursively inspected. At the end of phase I, all modified files and all directories have been marked in the bitmap, as shown (by shading) in Fig. 6-25(a). Phase 2 conceptually recur- recursively walks the tree again, unmarkmg any directories lhat have no modified files or directories in them or under them. This phase leaves ihe bitmap as shown in Fig 6-25(b) Note that directories 10, II. 14, 27, 29, and 10 are now unmarked because they contain nothing under them lhat has been modified They will not be dumped. In contrast, directories 5 and 6 will be dumped even though they themselves have not been modified because they will be needed to restore today's changes to a fresh machine For efficiency, phases ! and 1 can be combined in one tree walk. , | 2 | 3 At this point it is known which

directories and files must be dumped. These art; the ones marked in Fig. 6-25(b). Phase 1 consents of scanning the i-nodes m numerical order and dumping all the directories that are marked for dumping These are shown m Fig. 6-25(c). Each directory is prefixed by the director's attributes (owner, times, etc.) so they can be restored. Finally, in phase 4. the files marked in Fig. 6-25(d) are also dumped, again prefixed by iheir attributes This completes the dump, Restoring a file system from the dump tapes is straightforward To start with, an empty file system is created on the disk. Then ihe most recent full dump is restored Since the directories appear first on the tape, they are all restored first.

SEC 6.3 HILE SYSTEM IMPLEMENTATION 421 giving a skeleton of the file system. Then the files themselves are restored This process is then repeated wtth the first incremental dump made after the full dump. then the next one, and so on. Although logical dumping is siraight forward, (here are a few tricky issues For one. since the free block list is not a file, it is not dumped and hence it must be reconstructed from scratch after all the dumps have been restored. Doing so is always possible since the set of free blocks is just the complement of the sei of blocks contained in all the files combined Another issue is links. If a file is linked to two or more directories, it is important that the file is restored only one time and that all the directories that are supposed to point to it do so Still another issue is the fact that UNIX files may contain holes. It is legal to open a file, write a few bytes, then seek to a distant file offset and write a few more hytes. The blocks in between are not part of the file and should not be dumped and not he restored Core files often have a large hole between the data segment and the stack If not handled properly, each restored core file will fill this area wilh zeros and thus be the same size as the virtual address space (e g., 2*2 bytes, or worse yet. 2M bytes) Finally, special fiies. named pipes, and the like should never be dumped, no matter m which directory they may occur (they need not be confined to/dev). For more information about file system backups, see (Chervenak et al., 1998; and Zwicky, 1991). File System Consistency Another area where reliability is an issue is file system consistency Man> file systems read blocks, modify them, and write them out later. If the system crashes before all the modified block-, have been written out. the file system can be left in an inconsistent state. This problem is especially critical it some of the blocks that have not been written out are i-node blocks, directory blocks, or blocks containing the free list. To

deal with the problem of inconsistent file systems, most computers have a utility program that checks file system consistency. For example, UNIX has fitk and Windows has scaiidi,\k. This utility can be run whenever the system is booted, especially aftei a crash. The description below tells how fsck works, Scandisk is somewhat different because it works on a different file system, but the general principle of using the file system's inherent redundancy to repair it is still valid All file .system checkers venfy'each file system (disk partition) indepen- independently of the other ones. Two kinds of consistency checks can be made: blocks and files, To check for block consistency, the program huilds two tables, each one containing a counter tor each block, initially set to 0. The counters in the first table keep track of how

422 Ml ESYSTEMS CHAP iiun> limes The prograr ich block is present m the ficc list (or the bitmap Starting ttom an i-imdc u is possible iii build a list ot all the block numbers Used in the corresponding file As block number is ie<id. its counter in the first table is incremented The program Bach occurrence of a blo^k m the free list iesulis in its counter in thu second table being incremented I| the tile system is consistent, each block will h;i\e ,i I tithet m ihe tnsl table or m the second table, js illustrated in F-m. 6-26(<i> Howcvei. as a resull of a crash, the tables might look like Fig 6-26(b), in which block 2 does not occur m eithei lable It will be repoiied as being a missing block. While missing blocks do no real harm, they do waste ->p.ice and thus reduce the capucilv ot the disk The solution to missing blocks, is straightforward' ihe file system checker jim dddsihem to the free list 0 oh 0 1 11111 11 0 1 1 1 0 0 0 10 i|o o|o 0 1 i|o i 0 1 2 3 4 5 6 7 8 910 I 0 ' 0 1 0 1 1 1 1 11 0 0 0 1 1 I 1 1 0 0 o|i o|s o|o o|o IP 0 0 0 1 1 1J1 0 1 0 1 1]1 i|o 0 ill ,00 0 0 0 0 i 0 0 0 o|i 1 0 0 o|i 1 Blocks-nUSS RTMil|2[.|l|r|0|o|i|ih|o|o|Bi F*9 blocks FHTRi|0|oHo|ili|0|o|0RTl F, Figure b-26- i"i\e ^\>lem ^uies l.il C^nsisltni (b. Missing b]t>ck \c)]!>uph-tJte block ill tree Ijst (d) DuplitJte daU hlntk Another situation that might occur is that of Fig 6-26<c) Here we see a block, number 4. that occurs t^ice m the free list. (Duplicates can ^iccur only if the tree list is really a list: with a bitmap i! is impossible) The solution here is ako simple: rebuild the tree list. The worst thing that can happen is thai the saitic dala bldck k present in tv.Li or more files, as bhown in Fig. 6-26(d) with block 5. If enher ol these files is removed, block 5 will be pui on the free list, leading to a situation in which the same block is both in use and free at the same time. If both files are removed, the block will be put onto the free list twite The appropriate action for the file

system checker to take is to allocate J free block, copy the contents of block 5 into it, and insert the copy into one of the files.

SEC 6 1 F|Lf. SYSTEM IMPLFMSNTATION 423 In ihii. way, the uiformamm content of the dies is unchanged (although almost assuredly one is garbled), but the file system structure is ,tt least made consistent. In addm rounted for. the checker also checks the director) s)Uem It- too, usc-s a table of counters, but these aie per tile, lather than per block It starts at the root directory tones Symbolic links do not count and do not cause the counter loi' the target tile to be incremented. When it is all tidiie it has a list, indexed by i-nodc- nunibci, telling how many directories contain each tile. It [hen compares these uumbcis with the link counis stored in tlie i-nodes themselves These counts start at I when a hie is created and are incremented each time a (hard) link is made to ihe tile. In a consistent rile system, both counts' will agree However, two kinds of errois can otcui the link count in ihe i-node can be too high or it can be too low. If the link count is higher than the number of directory entries, then e\cn il ^ 11 the Tiles are removed troin the directors, the count will still be nonzero and the i-uodc will not be removed. This error is not serious, but it wastes space on the disk with hies that are ,iot in any director) It should be iixed by ^enrns the link count in the i-node it> the correct value The other error is potentially catastrophic. If mo directory entries aic linked lo a file, but the i-node says that there is only one. when either directory entry is removed, the i-node count w ill go to zero. When an i-node count goes u, zero, the file system marks il as unused and releases all of its blocks This action will result in one of the directories, now pointing to an unused t-node. whose blocks may soon be assigned to other files Again, the solution is just to force [he link count in the i-node to the actual number of directory entries. These two operations, checking blocks any checking directories, arc often integrated for efficiency reasons (;c . onl) one pass over the i-nodes is icuuired). Othei checks are akso possible For example, directories have ;i definite Tormat, with i-node numbers and ASCII names. If an i-m.de number is l.irgei than the number of i-nodes on the disk, the director; has been damaged. Furthermore, each i-node has a mode, some of which ait legal but strange, such as 0007. which allows the ownei and his group no access at all, but allows outsiders to read, write, and execute the file. It might be useiul to at leasi report files that give outsiders more rights than the owner. Directories with more than, say. 1000 entries arc also suspicious Files located in user directones, but which are owned by the

.superuser and have the SETUID bif on, are potential secunty problems because such files acquire- the powers of the supemser when e\mifed by any user With a little effort, one can put together a fairly long list of techni- technically legal but still peculiar situations that might be worth reporting.

424 FILE SYSTEMS CHAP b The previous paragraphs have discussed the problem of protecting the user against crashes. Some file systems also worry about protecting the user against himself- It the user intends to type rm *.O to remove all the files ending with o (compiler generated object files), but accidentally types (note the space after the asterisk), rm will remove all the files in the current direc- loiy and then complain that it cannot find o. In MS-DOS and some other systems, when a file ts removed, all that happens is that a bit is set in the directory oi i- node marking the file as removed No disk blocks are returned to Ihe free list until they are actually needed. Thus, if the user discovers the error immediately, it is possible to run a special utility program that "unremoves" (i e.. restores) the removed files. In Windows, files that are removed are placed in the recycle bin. from which they can later be retneved if need be Of course, no storage is reclaimed until they are actually deleted from this directory. 6.3.7 File System Performance Access to disk is much slower than access to memory Reading a memory word might take 10 ns,ec. Reading from a hard disk might proceed at 10 MB/sec, which is forty times slower per 32-bit word, but to this must be added 5-10 msec to seek to the track and then wan for the desired sector to arrive under the read head. If only a single word is needed, the memory access is on the oider of a mil- million times as fast as disk access. As a result of this difference in access time, many file systems have been designed with various optimizations to improve per- performance In this section we will cover three of them Caching The most common technique used to reduce disk accesses is the block cache or buffer cache. (Cache is pronounced "cash" and is derived from the French ta- cher. meaning !O hide.) In this context, a cache is a collection of blocks that log- logically belong on the disk but are being kept in memory for performance reasons. Various algorithms can be used to manage the cache, but a common one is to check all read requests to see if the needed block is in the cache. If it is, the read request can be satisfied without a disk access It the block is not m the cache, it is first read into the cache, and then copied to wherever it is needed. Subsequent requests for the same block can be satisfied from the cache. Operation of the cache is illustrated in Fig. 6-27. Since there are many (often thousands of)

blocks in the cache, some way is needed to determine quickly if a

SEC 6 1 FILE SYSTEM iMPLF.MFNTATION 425 given block i> present. The usual way is io hash the device and disk address and look up the result in a hash table All the blocks with the same hash ualue are chained together on a linked list so the collision chain can be followed FiKure ft-27. The buftci ldlhc ,l<it.i ^trmiliic- When a block has to be loaded into a full cache, some block has to be removed (and rewritten to the disk if it has been modified since being brought in) This situation k very much like paging, and all the usual page replacement algo- algorithms described in Chap. 1 such as FJH). second chance, and I.RU. are applica- applicable. One pleasant difference between paging and caching is that cache references are relatively infrequent, so (hat it is feasible k, keep all the blocks in exact LRU order with linked lists- In Fig. 6-27, we see thai in addition to the collision chains starting at the h.ish table, there is also a bidirectional list running thiough all the blocks in the order ol usage, with the leas! recently used block on the front of this hst and the most recently used block at ihe end ot ihis list When a block is referenced, n can be removed from its position on the bidirectional list and put at the end. !n this way. exact LRU order can be maintained Unfortunately, there is a catch Now ihat we have a situation in which exact LRU is possible, it turns out that LRU is undesirable. The problem has to do with the crashes and file system consistency discussed in the previous section. If a cnt- ical block, such as an i-node block, is read into the cache and modilied. but not rewritten to the disk, a crash will leave ihe tile system in an inconsistent slate I! the i-nnde block is put at the end of the LRU chain, it may be quite a while before it reaches the front and is rewritten u, the diL,k Furthermore, some blocks, such as i-node blocks, are rarely referenced rwo times within a short interval These considerations lead to a modified LRU scheme, taking two factors uito account ■ 1. Is the block likety to be needed agi,in soon" 2. Is the block essential to the consistency ot the tile s\ stern7 For both questions, blocks can be divided into categories such as i-node blocks, indirect blocks, directory block*, full data blocks, and partially full data blocks

426 HLfciYSlHVlS ("HAP (■> Blocks that will piobablj noi be needed again soon go on the tu«". raihci than ihe [ear of the I.RU list, so then buffers will be icused quickly Blocks thai might be ■Lccdcd again soon, such us ,i partly hill block that is being wnllcn. go <>n the end of the l'sl, so thev will stay aiound

for ;i loiig lime The second question it mdcpcndcni of the first one. If the block is essential to the t'lt: system consistency (basically, eveiything except daia blocks), and it has been modified, n shonkl be written to disk immedidlely. regardless of which end .,1 the LRU list it is put on. B> writing cnlitjl blocks quickly, we gieatlv re-duce the 1'iobabihty thai a cush will wreck Ihc tile sWem While a user ina> be unhappy il oik: ot his tiles is lUincd in a crash, he is likely 10 be iai more unhappy ,1 ilic whole file system is lost Hie plight ol someone- who is using a peisoual computci to write a book Hven it thcte is a sriiOtl chance thai c\eiythmg ivill still be m the cache and nothing tin the di^k If the system trashes, the die system structure will not be corrupted, but a whole day's work will be losl 1 his situation need not happen vci\. otteii bcfoie we have a fairly unhappy Uicr Systems take two approaches to dealrnjj with ir The I'NlX w;n is to have a system tall, sync, which forces all the mnditicd blocks out onto the disk immedi- immediate I > When the system is started up. a program. Usually ca'led update, is started up in the background to sit in an endless loop issuing Sync calls, sleeping for 30 sec between calls As a result, no more that, 10 seconds of work is losl due to a crash The MS-DOS way is to wiite every modified block to disk as soon as it has been wntteii Caches in which all modified blocks are wnttcn back to the disk immediately are called write-through caches. Thev requiie more disk I/O than nonwrite-through caches. The difference beuveen ihese two approaches can be seen when a program writes ;i I-KB block lull, one character at a urnu UNIX will collect all the characters in the cache and write ihc block out once every 30 seconds, or whenever the block is removed from the cache MS-DOS will make a dishk access for every characlci wnlien. O| course, niost programs do internal buffering, so they normally \vnte not a character, but a hue or a largei unit on each write system call. A consequence of this difference in caching strategy is thai just removing a (t loppy (disk from a U N |X system wnh-jut doing a sync will almosl a I wa\ s result in lost data, and Irequemiy in a corrupted file system as well With MS-DOS, no problem arises. These differing strategies wcie chosen because UN|X was de- veloped in an environment m which all disks were hard disks and not removable, whereas MS-DOS started out in the floppy disk world. As hard disks became the norm, the liNlX approach, with iis better efficiency, became the norm, and is also used now on Windows for hard disks.

 Block Read Ahead A second

technique (or improving puiccivcd (ik syslem perfcu o gel bk> rate. In my Tiles are read sequentially When the file system is asked to produce block k in a Ilk-, it docs that, but when il is finished, it makes ,1 sneaky check in. cache Al Ihe ..cry le.isi. il will be onthe^av. tact, it hurts b> tying up disk bandwidth reading in useless blocks and removing potentially useful blocks from the cache (and possibly tying up more disk bandwidth writing them back to disk if they are dirt)). To see whether read ahead is worth doing, the file system can keep track of the access patterns to each open file For example, a bit associated with each tik c^fi keep truck ot wh^thei ihc file is m "sequential access- mode" or "random access mode." Initially, ihc file is given the benefit ol the- doubt and put m sequential access mode. However, when- whenever a seek is done, the bii is cleared. It sequential ieads -;tait happening again, the bit is set once again In this way. the die system can make a reasonable guess about whether it should read ahead or not li it gets it wrong once it a while, it is Reducing Disk Arm Motion Caching and read ahead are no| the only ways to increase tile system perlor- niance. Anothei important technique is to reduce the amouni of disk arm motion by putting blocks that are likely to be accessed in .sequence close to each other, preferably m the same cylinder. When an output file is written, the file system has to allocate the blocks one at a time, as they are needed If the fiee blocks are recorded in a bitmap* and the whole bitmap is m mam meiuorv. 't is casN enough to choose a free block as close as possible to the previous block With a free list. part of which is on disk, it is, much harder to allocate blocks close together However, even with a free list, some block clustering can be done The trick is to keep track of disk slorage not in blocks, but in groups of consecutive block*,. If sectors consist of 512 bytes, the system could use 1-KB blocks B sectors) but allocate disk storage in units ot 2 blocks D sectors) This is not the same as hav- having a 2-KB disk blocks, since the cache would still use 1-KB blocks and disk transfers would Still be | KB but reading a file sequentially on an otherwise idle .system would reduce the number of seeks by a factor of two. considerably im- improving performance. A variation on the same iheme is to take account of rota- rotational positioning. When allocating blocks, the system attempts to place consecu- consecutive blocks in a file in the same cylinder.

428 Another perfor RLE SYSTEMS ; bottleneck in CHAP 6 s thai use i-nodes or anything equivalent to inos i t rg it file requires iwo disk accesses: one tor the t-node and one tor the block. The usual i-node placemen! is shown in Fig 6-

28<a) Here all the i-nodes are near the beginning or the disk. cylinders, requiring long seeks. l-noaes are be about half the number of Disk is divided mlo at ilit: sun or ihe disk ib) Disk dnidcd mio n blocks jiid I nodes One easy performance improvement is to put ihc i-nodc- in ihe middle ot the disk, rather than at the start, thus reducing ihe average seek between the mode and the first block by a factor of two. Another idea, shown in Fig. (i-28(b). u to divide the disk into cylinder groups, each with iis own i-uodes, blocks, and free list (McKusick et a]., 1984). When creating a new file, any t-node can be chosen, but an attempt is made to find a block in the s.une cylinder group as the i-node II none is available, then a block 111 a nearby cylinder group is used 6.3.8 Log-Structured File Systems Changes in technology are putting pressure on current tile systems In partic- particular, CPUs keep getting faster, disks are becoming much bigger and cheaper (but not much faster), and memories are growing exponentially in size. The one parameter that is not improving by leaps and bounds is disk seek time The com- combination of these factors means lhat a performance bottleneck is arising in many file systems. Research done at Berkeley attumpied to alleviate this problem by designing a completely new kind of file system. LFS (the Log-structured File System). In this secnon we will briefly describe how LFS works. For a more complete treatment, see (Rosenblum and Ousierhout. 199\) The idea that drove the LFS design is ihai as CPUs get taster and RAM mem- memories get larger, disk caches are also increasing rapidly Consequently, it is now

SH- f, 3 Fll F. SYSTEM [MPLEMfcNTAl [ON 429 possible Hi satisfy a very substantial fraction ot all read request:, directly from ihe file system cache, svith no disk access needed. It follows trom this observation, thai in the future, most disk accesses will be writes, so the read-ahead mechanism used in some file s>siems lo fetch blocks before ihe> arc needed no longer gain^ much performance To make matters worse, in most lile systems, writes are done in \ery t,ma!l chunks Small writes are highly inefficient since a 50-u.scc disk write is often preceded by a 10-insce seek and a 4-msec rotational delay. With these pauime- lers. disk eifidcncy drops to a traction oi 1 percent To see where all the small writes come from, considei creating a new tile on a UNIX s)su-iTt. To write this file, the i-nodc ioi the directory, the directory block, the i-nodc lor the tile, and the file itselt must Jll be written. While these writes can be delayed, doing so exposes the file system to serious consistency problems it a. crash occurs before1 the writes are done. For this reason, the wiode writes

are generally done immediately From this reasoning, the LFS designers decided to re-implement the UN[X file system in such a way as to achieve the full bandwidth of the disk, even in the face of j workload consisting in largt^ part of snml| random writes. The basic idea is to structure the entire disk at, a log Periodically, and when theie is a special iieed for it. all the pending writes- bemg buffered in memory are collected into a single segment and written to the disk as a single contiguous segment at the end ol the log A single se^meiit may tlius contain i-nodes, directory blocks* aito data blocks, all mixed together. At the start of each segment is a segment summary, telling what can be tound in the segment. It the average segment can be made to be about I MB, almost the full bandwidth oi the disk can be utilized In this design, i-nodes still exist and have the same structure as in UNIX, but they are now scattered all over the log. instead of being at a fixed position on the disk. Nevertheless, when an i-node is located, locating the blocks is done m the Usual way. Of course, finding an i-node is now much hardei, since its address cannot simply be calculated from its ,-number. as in UNIX To make it possible to find i-nodes. an i-node map. indexed by i-number, is maintained. Entry ' in this map points to i-node ; on the disk. The map i.s kept on disk, but it is also cached, s-o the most heavily used parts will be in memory most of the time To summarize what we have said so tai. all writes are initially buffered in memory, and periodically all the buffered writes are written to the disk rn a single segment, at the end of the log Opening a file now consists of using the map to locate the i-node for the file. Once the i-uode has been located, the addresses of the blocks can be tound trom it AH of fhe blocks will themselves be in segments, somewhere in the log. If disks were infinitely large, the above description would be the entire story. which time no new segments can be written to the log Fortunately, many existing segments may have blocks ihai are no longer needed, for example, it" a file is

 h overwritten, its i-node will now point to the new blocks, but the old ones will still be occupying space in previously written segments. To deal with this problem. LFS has j cleaner thread that spends its tune scan- scanning \he log circularly to compact it II starts out by reading the summary of the first seamen! in the log to see which i-nodes and files, -ire there. It then checks the current" i-nc,(fc map (o see il (he i-uodtsS are still current and file blocks are still m use I] not. thai information is discarded Tlie i-nodes and blocks that are still in use go into memory to be

Written out m the next segment. The original segment is then marked as free, so the log can use it for new data. In this manner, the cleaner moves along the log. iemoung old segments from the back and putting any Ifve big cuculur buffer, with the writer thiead adding new segments to the front and the clcanei thread removing old ones from the hack The bookkeeping here ,s tfontnvial, since when J file block is written back 10 ii new segment, the i-node of the file (somewhere in the log) must be located, updated, and put into memory id be written out in the next segment. The i-node map must then be updated to point to the new copy. Nevertheless, it is possible to do the administration, and the performance results show that all this complexity is worthwhile. Measurements given m ihe papers cited above show rhat LFS outper- outperforms UNIX by an ordei of magnitude on small writes, while having a perfor- performance that is as good us or better than UNIX for reads and large wiites 6.4 EXAMPLE FILE SYSTEMS In (he following sections, we will discuss several example file systems, rang- ranging from quite simple to highly sophistics led Since modem UNIX file systems und Windows 2(XX)'s native file system are covered in the chapter on UNIX (Chap. 10) and the chapter on Windows 2000 (Chap. 11) we will not covei those systems here. We will, however, examine their predecessors below. 6.4.1 CD-ROM File Systems CD-ROM^ These systems are particularly simple because they were designed toi write-once media. Among other things, for example, they have no provision lor keeping track of free blocks beaust> on a CD-ROM files cannot be freed or added after the disk has been manufactured. Below we will take a look at the mam CD- ROM file system type and two extension!, to u The ISO 9660 File System The most common standard for CD-ROM file systems was, adopted as an International Standard m 1988 under ihe name ISO 9660. Virtually every CD- ROM currently on the market is compatible with this standard, sometimes wifh

SEC 6 4 431 ihe extensions to be discussed below One of the goals of this standard wus to make every CD-ROM readable on every computer, mdependenl of the byte order- ordering used and independent of the operating system used. As a consequence, wme limitation* were placed on the file system to make n possible for the weakest operating systems then in use (such as MS-DOS) to read it CD-ROMs do noi have concentric cylinders the w.iy magnetic disks do. Instead there is a single continuous spiral containing the bits in a linear sequence (although seeks across the spiral are possible) The bits

along the spiral are divided into logical blocks (also called logical sectors) of 2352 bytes Some of these aie loi preambles, error correction, and othei overhead The paylnad por- portion of each logical bkx k is 2048 bytes When used for music. CDs have leadins. leadouts. and mtertracK gaps, but these are not Used for data CD-ROMs. Often the position or a block along the spiral is quoted in minutes and seconds It can be converted to a linear block number using the conversion factor or | sec = 75 blocks ISO 9660 supports CD-ROM sets with as m.my as 2lfl - I CDs in the set The individual CD-ROMs may also be partitioned into logical volume*, (partitions). However, below we will concentrate on ISO 9660 for j single.- unpartitioned CD- ROM. Every CD-ROM begins with 16 blocks whose function is not detmed by the ISO 9660 standard. A CD-ROM manufacture! could use this aiea for piovldmg a bootstrap program to allow the computer to be booted from the CD-ROM, or tor some other purpose. Next comes one block containing the primary volume des- descriptor, which contains some general information about the CD-ROM Among this information are the system identifier C2 bytes), volume identifier C2 bytes), publisher identifier A28 bytes), and data prepaier identiiiei A28 bytes). The manutacturcr can fill in these fields in any desired way. except that only upper case letters, dtgiis, and a very sniall numbei ot punctuation maiks may be used to ensure cross-platform compatibility The primary volume descriptor also contains the names or three files, winch may contain the abstract, copyright notice, and bibliographic information, respec- respectively. In addition, certain key numbers are also present, including the logical block size (normally 2048, but 4096. 8192, and larger powers or two are allowed in certain cases), tile number ot blocks on the CD-ROM, and the creation and expiration dates of the CD-ROM. Finally, the primary volume descriptor also contains a directory entry tor rhe root directory, telling where to find it on the CD-ROM (i.e., which block it stalls at). From this directory, the rest of the file system can be located. In addition to the piimary volume descnptor. a CD-ROM may contain a sup- supplementary volume descriptor. l| contains siuiilai information to the primary, but thai will not concern us here. The root directory, and all other directories for that matter, consists ot a van- able number of entries, the last ot which contains a bit marking it as the final one

 le of which are in ASCII and others of which itr> coded mice, once m little tuple) and once in big-endian tornial The directory entries thi-mse consists ot It) to 12 fields, s. numerical fields m binary The

binar enduin founai (lined on Pentiums, foi (used on SPARCs, for example). Thus a 16-bit number uses 4 bytes and a 32-bit number uses 8 bytes The use of this redundant coding was necessary to avoid hurting anyone's feelings when the standard was developed It the standard had dictated little endian. then people trotn companies with btg-endian products would have felt like second-class citizens and would not have accepted the standard. The emotional content ot a CD-ROM can thus be quantified and measured exactly iti kilob>tes/houi of wasted space. I he format ,if an ISO 9660 directory entry is illustrated in Fig. 6-29 Since directory entries have variable lengths, the tirst field is a byte telling how long the eniiy is This byto is defined to have ihc high-ordci bit on the left to avoid any ambiguity — Extended attribute re ectory entry tenglh 6-29. The ISO %6() din Directory entries may optionally have an used tor a directory entry, the second byte t are. ing the extended attnbute^ Next runs of blocks, so a file's location is completely specified by the starting block and the size, which is contained m the next field The date and time that the CD-ROM was recorded ts stoied in the next field, with separate bytes for the year, month, daj, hour, rmnute. second, and time zon«. Years begin to couni at 1900. which means that CD-ROMs will suffer from a Y2156 problem because the year following 2155 wtll be 1900 This pioblem could have been delayed by defining the origin of time to be 1988 (the year tbe standard was adopted) Had that been done, the problem would have been post- postponed until 2244. Every S8 extra years helps entry in hMings (a feature copied from MS-DOS), one to distinguish an entry that is a file troni an entry that is a directory, one to enable the use ot the exiended attributes, and one to mark the last entry in a directory A few other biis are also

SEC. 6.4 EXAMPLE FIU SYSTEMS 433 present in this field but they will not concern us here. The next field deals with interleuv Lng pieces of files m a way that is not used in the simplest version of ISO 9660, so we will not consider it further The next field tells which CD-ROM the file is located on. It is permitted that a directory entry on one CD-ROM refers to a file located on another CD-ROM in the set In this way it is possible to budd a master directory on the first CD-ROM that lists all the Hies on all the CD-ROMs, m the complete set. The field marked L in Fig. 6-29 gives the size of the file name in bytes It is followed by the file name itself. A file name consists of a base name, a dot, an extension, a semicolon, and a binary version number (I or 2 bytes). The base name and exiension may u,se upper case letters, the digits 0-9,

and the underscore character AH other characters are forbidden to make sure that every computer can handle every file name. The base name can be up to eight characters: the extension can be up to three characters. These choices were dictated by ihe need to be MS-DOS compatible. A file name may be present in a directory multiple times. as long as each one has a different version number. The last two fields are not always present The Padding field is used to force every directory entry to be an even number of bytes, to align the numeric fields of subsequent entries on 2-byte boundaries if padding is needed, a 0 byte is used. Finally, we have the System use field. Its function and size are undefined, except that it must be an even number of bytes. Different systems use it m different ways. The Macintosh keeps Finder flags here, for example. Entries within a directory are listed in alphabeiicai order except for ihc first two entries. The first entry is for (he directory itself. The second one is for ap- apparent- In this respect, these entries are similar to the UNIX . and directory' entries. The files themselves need not be in directory order- There is no explicit limit to the number of entries in a directory. However. There is a lim(t to the depth of nesting. The maximum depth of directory nesting is eight. ISO 9660 defines what are called three levels. Level 1 is the most restrictive and specifies that file names are limited to 8 + 3 characters as we have described, and also requires all files to be contiguous as we have described. Furthermore, it specifies that directory names be limited to eighi characters with no extensions. Use of this level maximizes the chances that a CD-H-OM can be read on every computer. Level 2 relaxes the length restriction. It allows files and directories to have names of up to 31 characters, but still from the same set of characters. Level 3 uses ihe same name limits as level 2, but partially relaxes the assump- assumption that files have to be contiguous. With this level, a file may consist of several sections, each of which is a contiguous run of blocks. The same run may occur multiple times in a file and may also occur in two or more files. If large chunks of data are repeated in several flies, level 3 provides some space optimization by not requiring the data io be present multiple times.

434 FILE SYSTEMS CHAP, d Rock Ridge Extensions As we have seen, ISO 9660 is highly restrictive in several ways. Shortly alter it came out, people in the UNIX community began working on an extension to make it possible to represent UNIX tile systems on a CD-ROM These extensions weie named Rock Ridge, after a lown in the Gene Wilder movie Blazing Saddles, piobablv

because one of the committee members hked the film The extensions use the S\'\leni use field in order ti> make Rock Ridge CD- ROMs readable on any computer. All the other fields retain then normal ISO 9660 meaning Any system not aware of the Rock Ridge extensions just ignores them and sees a normal CD-ROM. 1. PX - POSIX attributes. 2. PN - Major and minor device number* ^ SL - Symbolic link 4. NM - Alternative name. 5 CI- - Child locution 6 PL - Parent location. 7 RE - Relocation. « TF-Timc stamps. The PX field contains the standard [1N|X nui-«mn permission bits for the owner, group, and otheis- It als" contains the otlier bits contained iti the mode word, such as the SETUID and SCTGID bns. and so (>u To allow raw devices to be represented on a CD-ROM, the PN held is present It contains Ihe major and minoi device numbers associated with the lilt;. In this way. the contents ot the /(lev directory tan be written to a CD-ROM and laler reconstructed correctly on the target system to a file on a different file system. Prohably the mcist important field is NM. It allows a second name to be asso- associated with the file This name is not subject to the characiei set or length restric- restrictions of ISO 9660. making it pos.sib.le to express arhitrary UNIX file names on j CD-ROM. The next three fields .ire used together lo get around the ISO 9660 limit of directories that may only be nested eight deep. Using them it is possible to specify that a directory is to be relocated, and to tell where ii goes in the hierar- hierarchy. It is effectively a w,Ly ti, work around the artificial depth limit.

Finally, the TF field contains the three tmieslamps included in each UNIX i-node. namely the time the file was created, the time it was, last modified, and the time it was last accessed. Togethc-r. these extensions make ii possible to copy a UNIX file system to a CD-ROM and then restore it correctly to a different system Joliet Extensions The UNIX community was nut the only group that wanied a way to extend ISO 9660. Microsoft also lound it too lestrictive (although il was Microsoft's own MS-DOS that caused ino.si of the restrictions m the lirsl place) Therefoie Microsoft invented some extensions that were catled Joliet They were designed to allow Windows file systems to be copied to CD-ROM and then restored, in pre- precisely tlic same way that Rock Ridge was designed for UN[X Virtually all pro- programs that run under Windows and use CD-ROMs support Joliet- including pro- programs that burn CD-recordables Usually, these programs offer a choice between the various ISO 9660 levels and Joliet. The major extensions provided by Johet are I Long tile names 2. Unicode character set. 3 Directory nesting deeper than eight levels. 4.

Directory names with extensions. enables the use of the Unicode character set tor file names. This extension is important for software intended for use in countries that do not use the Latin alphabet, such as Japan. Israel, and Greece Since Unicode characters are two bytes, the maximum file name in Juliet occupies 128 bytes Like Rock Ridge, the limitation on directoiy nesting is removed by Johet. Directories can be nested as deeply a-, needed. Finally, directory names can have extensions. It is not clear why this extension was included, since Windows direc- directories virtually never use extensions, but maybe some day they will 6.4.2 The CP/M File System The first persona! computers (then called microcomputers) came out in the early 1980s. A popular c-arly type used the 8-bit Intel 8080 CPU and had 4 KB ot RAM and a .single 8-inch floppy disk with a capacity ol 180 KB. Later versions used the slightly fancier (but Still 8-bit) Zdog Z80 CPU. had up to 64 KB ot RAM. and had a whopping 720-KB floppy disk as the mass stoiage device. Despite the slow speed and small amount of RAM, nearly all of these machines ran a surprisingly powerful disk-based operating system, called CP/M (Control

 Program for Microcomputers) (Golden and Pechura. 1986). This system dom- dominated its era as much as MS-DOS and later Windows dominated the IBM PC world- Two decades later, it has vanished without a trace (except I of a small group of diehard fan*;), which gives reason lo think that systems- that now dom- dominate the world may be essentially unknown when current babies become college students (Windows what7) II is worth taking a look ai CP/M for several reasons First, it was a histori- historically very important system and was the direct ancestor oi MS-DOS. Second, current and future operating system designers vsho think ihat a computer needs 32 MB just to boot the operating system could probably learn a lot about simplicity from a system that ran quite well in 16 KB of RAM Third, in the coming dec- decades, embedded systems are going lo be extremely widespread Due to cost, space, weight, and power constraints, the operating systems used, toi example, in watches, cameras', radios, and cellular telephones, are of necessity going to be lean and mean, not unlike CP/M. Of course, these systems do not have 8-inch floppy disks, hut they may well have electronic disks, using flash memory, and building a CP/M-hke file system on such a device is straightforward. The layout of CP/M m memory is shown m Fig. 6-30 At the tup of main memory (in RAM) is the BIOS, which contains a basic library ot I 7 I/O calls used by CP/M (in this section we will describe

CP/M 2 2, which was the standard ver- version when CP/M was at the height of its popularity) These calls lead and write the keyboard, screen, and floppy disk Figure 6-31). Memory layout (it CP'M. Just below the BIOS is the operating system proper. The size of the operating system in CP/M 2.2 is 3584 bytes. Amazing but true: a complete operating system in under 4 KB. Below the operating system comes the shell (command Ime pro- processor), which chews up another 2 KB. The rest of memory is for user programs, except for the bottom 256 bytes, which are reserved for the hardware interrupt

SEC 6 4 437 vectors, a tew variables, and a buffer for the current command line ko u^er pro- programs can get ai it The reason for splitting the BIOS tiom CP/M itself (even though both are tn RAM) is portability. CP/M interacts with the hardware only by making BIOS oilK To port Cp/M 10 a new machine, all that k necessary is to port the BIOS there Once that has been done. CP/M itself can he installed without modification. A CP/M syaem has only one directory, which contains fixed-size C2-byte) entries The directory M?e. although fixed for a given implementation, may be different in other implementations of CP/M All files in the system are listed in this directoiy After CP/M bo:iN, it reads in the director) and computes a bitmap con- taming the free disk Mocks by seeing which blocks are not in any file This bit- bitmap, which is only 2"! bytes tor a 180-KB disk, is kept in memory during execu- execution At system shutdown time it is discarded, that is, not written back to the disk. This approach eliminates the need for a disk consistency checker (like ftck) and saves i block on the disk (percentually equivalent to saving 90 MB on a modern 16-GB disk). When the user types a command, the shell tirsi copies the command to a buffer in the bottom 256 bytes of memon. Then it looks Up the program to be executed and reads it into memory at address 256 (above the interrupt vectors), and jumps to it. The program then begins running It discovers iU arguments by looking in the command line buffer The program is allowed to overwrite the shell it' it needs the memory. When the program finishes, it makes a system call to CP/M telling it to reload the shelf (if ,t was overwritten) and execute it. In a nutshell, that is pretty much the whole CP/M story In addition to loading programs. CP/M provides 38 system calls, mostly file services, fot user programs The most important of these are reading and writing files. Before a file can be read, it must be opened When CP/M gets an open sys- system call, it has to read in and search the one and only directory. The directory is not kept in memory ail the time to save precious RAM When CP/M finds the entry, it

immediately has the disk block numbers since they are siored right in the directory entry, as are all the attributes. The format of a directory entry is given in Fig 6-31. The fields in Fig. 6-31 have the following meanings. The User code field keeps track of which user owns the file. Although only one person can be logged

438 F[Lb SYSTEMS C"|iA.P t into a CP/M at any given moment, the syuem support'; multiple users who take turn, using the system. While parching tor a file name, only those entries belonging id the currently logged-in user are checked In eflect. each user has a virtual directory without the overhead of managing multiple directories. The next two fields give the name and extension of the file. The base name is jp to eight characters; an optional extension of up to three characters may be present Only upper case letters, digits, and a small number of special characters arc allowed in file names. This S + 3 naming using upper case only was latei taken over by MS-DOS. The Block count field tells how many byles this file entry contains, measured ui units of 128 bytes (because I/O is actual!} done in 128-byte physical sectors) The last 1-KB block may not be full, so the system lias no way to determine the exact size of a file. It is up to the user to put in some END-OF-FILE marker it desired The final 16 fields contain the disk block numbers themselves Each block is I KB. so that maximum file size is 16 KB Note that physical I/O is done in units ol 128-byte sectors and sizes are kept track of in sectors, but file blocks are allocated in units of I KB (8 sectors at a time) to avoid making the directory entry too large However, the Cp/M designer realized that some [lie-., even on a [80-KB floppy disj;. might exceed 16 KB, so an escape hatch was built around the 16-KB limit. A file (hat is between 16 KB and 32 KB uses not one dirtsctoiy entry, but two. The first entry holds the first 16 blocks; the second entry holds the next Kr blocks. Beyond 32 KB, a third directory entry k used, and so on. The Extent field keeps track of the order of the directory entries so the system knows which 16 KB comes first, which comes second, and so on After un open call, the addresses of all the disk blocks are known, leaking read straightforward. The write call is also Nrinple. li just requires allocating ,i free block from the bitmap in memory and then writing the block. Consecutive blocks on a file are not placed in con.secimve blocks on ihe disk because ihc 8080 cannot process an interrupt and start reading ihe next block on time. Instead, interleaving is used to allow several blocks id be read on a single rotation. CP/M is clearly not the last word in advanced file systems, but it is

simple, fast, and can be implemented by a comiieteni programmer m less than a week For many embedded applications, it may be all thai is needed 6.4.3 The MS-DOS File System To a hist approximation, MS-DOS is a bigger and better version (>t CP'M It runs only on Intel platforms, does not support multiprogramming, and runs only m the PC's real mode (which was originally the only mode) The shell has niort features and there are iLK)re system calls, bui trie basic function of the operating system is still loading programs, handling the keyboard and screen, and managing the file system. It is Ihe latter functionality that interests us here.

PX- f,4 439 flic MS-DOS file system was patterned closely on the CP/M hie system, including the use of 8 + 3 {upper case) character file names. The first version (MS-DOS 1 0) was evtMi limited to a single.directory, like CP/M However, starting with MS-DOS 2.0. the Tile system functionality was greatly expanded The biggest improvement was the inclusion of a hierarchical tile system in which direcione.s could be nested to an arbitrary depth. This meant thai the rool directory (which still had a fixed maximum si?e) could contain subdirectories, and these could con- lain luither subdirectories ad infiniiem Links in ihe style or UN|X were noi per- permitted, so the file syslem formed a tree starting at the root director) . II is common for different application programs to start out by creating a siib- direciory in the root directory and pul .ill their files there (or in subdirectories thereon. ,so that different applications do not conflrct Since directories are them- themselves JIM stored as files, there are no limits on the number of dueciories or files that may be created Unlike CP/M. however, there is n<> concept ol different users in MS-DOS Consequently, the logged in user has access to all files,. To read a file, an MS-DOS program must first make an open system call to get a handle for it. The open system call specifies a path, which may be either abso- absolute or relative to the current working directory The path is looked up component by component until the final directory is located and read into memory It is ihen searched for the file to be opened Although MS-DOS directories are variable sized, like CP/M. they use a fixed- s-ize 12-byte directory entry. The formal oi an MS-DOS direciory entry is shown in Kig. 6-32 It contains (he file name, attributes, creation dale ami time, starting block, and exact file size File names shorter than 8+1 characters are left justi- tied and padded with spaces on the rmbt. in each field separately. The Aiti'ibut'1^ archived, is hidden, or i>- a system file. Rcid-onK files cannoi be written Thi», is to protect thent from

accidental damage. The archived bit has no jciiial operotina system function (i.e. MS-DOS does not examine or set ii). Tht- intention is to allow user-level archive piogrUWs lo cleai it upon archiving ,i file and lo have other programs set it when modi lying a file In this vsa>, a b.ickup program can just examine this attribute bit on every file to sec which files to b.itk up The hid- hidden bit can be set to pievent a file from appearing in direUor> lutings Its inatn use is to avoid confusing novice users with tiles lhe\ might not understand. Finally, the system bit also hides files In addition, system files cannot acciden- accidentally be deleted using ihe del command, The mam components <>t MS-DOS have this bit set The directory entrv also cont.iiiis' th£ date aiKl time the tile was Created or IjsI modified. The time is accurate only lo +2 sec because il is stored ir) a 2-bytc hours E bits) The date counts in days using three subfields: day E bits), month D bits), and year-1980 G bits). With a 7-bit number for the year and time

440 FILE SYSTEMS Bytes 8 Figure 6-32. '1 he MS-DOS d: beginning in 1980, the highest expressible year is 2107. Thus MS-DOS has a built-in Y2I08 problem To avoid catastrophe, MS-DOS user* should begin with Y2108 compliance as early as possible. If MS-DOS hail used the combined date and time fields, as a 32-bit seconds counter, it could have represented every second exactly and delayed the catastrophe until 2116 Unlike CPfM. which does not Store the exact file size, MS-DOS does. Since a 32-bit number is used for the file size, in theory files can be as large as 4 GB However, other limits (described below) restrict the maximum file size to 2 GB or less. A surprising large part of the entry A0 byteS) is unused. Another way m which MS-DOS differs from CP/M is that it does not store a file's disk addresses in its directory entry, probably because the designers realized that large hard disks (by then common on minicomputers) would some day reach the MS-DOS world. Insiead, MS-DOS keeps track uf file block* via a file alloca- allocation table in main memory. The directory entry contains the number of the first file block. This number is used as an index mto a 64K entry FAT in main mem- memory. By following the chain, all the blocks can be found. The operation of the FAT is, illustrated in Fig. 6-|4 The FAT file system comes in three versions for MS-DOS. FAT-12, FAT-16. and FAT-32. depending on how many bits a disk address contains. Actually. FAT-32 is something of a misnomer since only the low-order 28 bits of the disk addresses are used. If should have been called FAT-28, but powers of two sound so much neaier. For all FATs, the disk

block can be set to some multiple of 512 byies (possi- (possibly different for each partition), with the set of allowed block sizes (called cluster sizes by Microsoft) being different for each variant. The first version of MS-DOS used FAT-12 with 512-byie blocks, giving a maximum partition Mze of 2|2 x 512 bytes (actually only 4086 x 512 bytes because 10 of (he disk addresses were used as special markers, such a« end of file, bad block, etc. With these parameters, the maximum disk partition size was about 2 MB and the size of the FAT table in memory was, 4096 entries of 2 bytes each. Using a 12-bit table entry would have been too slow. This system worked well for floppy disks, but when hard disks came ciiit, U became a problem. Microsoft solved the problem by allowing additional block

SEC FXAMPI.EFII F SYSTFMS 441 sizes of 1 KB. 2 KB. and 4 KB. This, change preserved the structuic and size of the FAT-12 table, bui allowed diak partitions or up to 16 MB. Since MS-DOS supported tour disk partitions per disk drive, the new h'AT-12 f,le system worked up lo 64-MB disks Beyond that, something had to give What happened was the introducing of FAT-16, with 16-bit disk pointers Addi- Additionally- block size-, of 8 KB, 16 KB. and 32 KB weie permitted O2,7f>8 is the largest power ot two that can be represented in 16 bits.) The FAT-16 table now occupied 128 KB of mam memory all the time, but with the larger memories by The largest disk partition that can be supported by FAT-16 is 2 GB F4K entries or 32 KB each) and the largest disk 8 GB. namel> four partitions of 2 GB each using the DV standard, a 2-GB file holds |ust over 9 minutes of video As a consequence of the fact thai a PC disk can support only four partitions, the largest video that can be stored on a disk is about 38 minutes, no matter how large the disk is. This hmii also means that the largest video that cm be edited on line is less than 19 minutes, since both input and output files are needed Starting with the second release of Windows 95. the FAT-32 file system, u ith its 28-bit disk addresses, was introduced and the version of MS-DOS underlying Windows 95 was adapted to support FAT-32. In this system, partitions could theoretically be 228 x 2|5 bytes, but they are actually limited to 2 TB B048 GB) because internally the system keeps (rack ot partition sizes in 512-byte sectors using a 32-bit number and 29 x 231 is 2 TB. The maximum partition size tor vari- various block sizes and all three FAT types is shown in Fig 6-3.3. Block size 0.5 KB t KB 2 KB 4KB FAT-12~T 2MB | 4 MB 8MB ' 16MB ' FAT-16 FAT-32 128MB 256 MB 1 TB Figure 6-33. Maximum parmion We for ditterenl block sues The emptj boxe^ represeni

lorbidden oombin.ilions In addition to supporting larger disks, the FAT-32 file system has wo other advantages over FAT-16. First, an 8-GB disk using FAT-32 can be a single parti- partition Using FAT-16 it has to be four partitions, which appears |() the Window-, user as the C D', E\ and F: logical disk drives. It is up to the user to decide which file to place on which drive and keep track of whaf is where.

442 FILE SYSTEMS CHAP 6 The other advantage of FAT-32 over t-AT-16 is that for a given si^e disk par- tition a smaller block size can be used. For example, for j 2-GB disk partition, FAT-'l6 must use 32-KB blocks, otherwise w,th only 64K available d.Oc addresses, it cannoi cover the whole partition. In contrast, FAT-32 can use, tor example, 4-KB blocks tor a 2-GB disk partition The advantage of the smaller biock size is that mosi hies, are much shorter than 12 KB. It the block si?e is 32 KB. a file of 10 bytes lies, up V- KB of disk space If the average file is. say 8 KB. [hen with a 32-KB block, Vt of the disk will be wasted, not a terribly efficient way io use the disk. With an 8-KB file and a 4-KB block, there is LLo disk was- wastage, but the price paid \t, more RAM eaten up by the FAT. Wuh a 4-KB block and a 2-GB disk partition, there are 5I2K blocks, so the FAT must have 512K entries in memory (occupying 2 MB of RAM!. MS-DOS uses the FAT to keep track ot free disk blocks- Any block that is not currently allocated is marked with a special code. When MS-DOS needs a new disk block, it searches the FAT for an entry containing this code Thus no bitmap or free list is required 6.4.4 The Windows 98 File System The original release ot Windows 95 used the MS-DOS file system, including the use of 8 + 3 character file names and the FAT-12 and FAT-16 file systems. Starting with the second release ot Windows 95. file names longer than 8 + 3 characters were permitted. In addition. I;AT-12 was introduced, primarily to allow larger disk partitions larger than I GB and disks laiger than 8 GB. which were then available. Both the long file names and FAT-32 were used in Windows 98 in the same form as in the second release of Windows 95 Below we will describe these features of the Windows 98 file system, which have been carried lorward intci Windows Me a« well Since long file names are more exciting lor users than the I'AT sliucturc. let us look at them first. One way (o introduce long file names would have been to just invent a new directoiy structure The problem with this approach is that it Microsoft had done this, people who were .still m the process of convening iioiri Windows 3 to Windows 95 or Windows 98 could not have accessed their fik's from both

systems A political decision was made within Microsoft that files created using Windows 98 must be accessible from Windows 3 as well (fw dual- boot machines). This constraint forced a design for handling long hie names that was backward compatible with the old MS-DOS 8 + 3 naming system. Since ,uth backward compatibility constraints" are not unusual m the computet industry, it is worth looking in detail at how Microsoft accomplished [his goal The effect of this decision to be backward compatible meant thui the Win- Windows 98 directory structure had to be compatible with the MS-DOS directory structure As we saw. this structure is |ust a list ot 32-byte entries; as shown in

SEC 6 4 EXAMPLE FILE SYSTEMS 443 h'ig. 6-32 This format came directly from t'P/M (which was written tor iiie 8080). which goes to show how long (obsolete) structures can live in the computer world. do with long names, but it is used in Windows 98, si s woith understanding Figure 6-34. The <?' changes ended WOS DCK d Upper 16 bits block ol the addition or rive new fields where the H) unused bytes used to be. The ATiieid is mostly there for some compatibility with Win- Windows NT in terms of displaying file names in the correct case (in MS-DOS, all file names are upper case). The Set field solves Ihe probletii that it is not possible to More the time of day m a 16-bit field. It provides additional bits so (hat the new Creation time field is accurate to 10 msec. Another new field U Last aice\s. which stores the date (but not time) of the las! access to the Hie. Finally, going to the FAT-32 file system means- that block numbers are now 32 bits, so an addi- additional I6-bi(field is needed to siore the upper 16 bits of (he starting block number. Now we come to the heart of the Windows 98 file system: how long file names are represented m a way that is backward compatible with MS-DOS The solution chosen was to assign two names to each file- a (potentially) long file name (in Unicode, for compatibility with Windows NT), and an 8 + 3 name lor compatibility with MS-DOS. Files can be accessed by cither name. When ,i file is created whose name does not obey the MS-DOS naming rules (8 + 3 length, no Unicode, limited character set, uO spaces, etc.), Windows 98 invents an MS-DOS name for it according to a certain algorithm. The basic idea is to take the lira six characcers of the name, convert them to upper case, if need be, and append -1 to form the base name, If this name already exists, then the suffix ~2 is used, and sn on. In addition, spaces and extra periods are deleted and certain special characters are convened lo underscores As an example, a file named The lime

has came the walrus said is assigned the MS-DOS name THETIM-I, If a subsequent file is created with the name The time has come the rabbit said, it is assigned the MS- DOS name THETlM-2. and so on. Every file has an MS-DOS file name stored using jhe directory formal of Fig. 6-14. If a file also has a long name, that name is stored in one or more direc- directory entries directly preceding the MS-DOS ("tie name. Each tong-naine entry holds up to 13 (Unicode) characters. The entries are stored in reverse order, with

444 FILE SYSTEMS CHAP the start ot the file name just ahead ot the MS-DOS entry and subsequent pio before it. The format of each long-name entry is given it! Fig 6-35. 'uz \ Sequence / tttnbutes An obvious question is. "How does Windows 98 know whether a directory entry contains an MS-DOS file name or a (piece of a) long file name1" The answer lies, in the Attribute* field For a lonH-narne entry, this field has ihe value OxOF, which represents an otherwis MS-DOS programs that read direeto impossible II JU- ■>..v... The pieces of the name are sequeneed using ihe ... last part of the long name (the first entry in the sequence to the sequence numbej Since only 6 bus are used fc theoretical maximum for file names is 63 x 13 or 819 charau.... - ...._, - limited to 260 characters for historical reason*. F_ach long-name entry contains a Chei ksitm field to avoid the following prob- >nd, Ihe ation of ^tributes Old , as invalid Little do they irst hyie ol the entry The rked hy adding 64 the sequence number, 'he In taut they First, a Windows' 98 program creates a file with a long nai,.~ computer is rebooted to run MS.DOS or Windows 3. Third, an old piogram there (hen removes the MS-DOS file name from the directoiy bui does not remove ihe long file name preceding il (because it does not know aboul n| Finally, some program creates a new file that reuses the newly-freed directory entry. At this ' " ilid sequence of long-name entries prei-eedmg an MS-DOS fill field allows Windows pro point point we have a valid sequence of long-name entries preLeedmg an MS-DOS file entry that has nothing to do with that file. The Checksum field allows Windows 98 to detect this situation by verifying that the MS-DOS tile name following a long name does, in fact, belong to it. Of course, With only I byte being used, there is chance in 256 that Windows 98 will not not To see an example of how long names wor 36. Here we have a file called The quick braw 42-characters, it certainly qualifies as a long f the file substitute , consider the example cvi Fig. 6- /ox jumps over the lazy clog. At ?. The MS-DOS name con- aracters, it certainly qualifies as a long file .stutted from it is THEQUl-1 and is stored in

the ._ J Some redundancy is built into the directory structure to help detect problems in the event ihat an old Windows 3 program ha.s made a mess of the directory The sequence number byte at the start of each entry is not strictly needed since the 0x40 bit marks the first one, but it provides additional redundancy, for example Also, the Low field of Fig. 6-36 (the lower halt of the starting cluster) \s 0 in all entries but the last one, again to avoid having old programs misinterpret it and

FXAMPLE FILE SYSTEMS w 2 , T w H E Q e U I - f 0 1 A A A A 0 0 0 n N K s t h e x I U 1 C Crutnn Last u k Upp I a m p b 0 0 0 0 Low z y s r 0 S,ze Figure 6-36. ample o] how J long is ■Jdred in W red in Win- Winrum [lie tile system. The NT byte in Fig. 6-36 m used in NT and ign dows °8 The A byte contains the attributes The implementation of the FAT-32 file system is conceptually si implementation ot the FAT-16 file system However, instead of a y 65,536 entries, there are as many entries as needed to cover the part ol the disk with data on it. It the first million blocks are used, the Uble conceptually has million entries. To avoid ha illion blocks ving all of them ilar to the rray h d , Windows 98 6.4.5 The UNIX V7 File System ophistica /e will di s We « ted scu 'ill multi ss the exam user file ■ V7 file s me modei iys VSl ■n tt-'ni em. v^r- Even early versions of UN|X had a fairly since it was derived from MULT1CS Below the one for the PDP-I t that made UNIX fam sions in Chap 10 The file system is in the form of a tree starting at the root directory, with the addition of links, forming a directed acyclic graph. File names are up to 14 char- characters and can contain any ASCII characters except / (because that is the separator between components in a path) and NUL (because that is used to pad out names shorter than 14 characters) NUL has the numerical value of 0. A UNIX directory entry contains one entry for each file in thai directory. Each entry is extremely simple because UN|X uses the i-node scheme illustrated tn Fig. 6-15 A directory entry contains only two fields- the file name A4 bytes) and the number of the i-node for (hat file B bytes), as shown in Fig. 6-37 The.,e parameters limit the number of files per file system to 64K Like the i-node of Fig. 6-15, the UNIX i-nodes contains some attributes. The attributes contain the filu size, three times (creation, last access, and last modifica- nonj, owner, group, protection information, and a count oi the number of direc- directory entries that point to the i-node. The latter field is needed due to links. When- Whenever a new link is made to an i-node, the count in the i-node is increased. When a

/cd. the t s decrctnc When reclaimed and the disk blocks arc put back Keeping track of disk blocks is done oicler to handle very laige hies The flrsi node itself, so for small files, all the neces fetched from disk to gets > 0. the i-node i what larger files, one of the addresses in the called a single indirect block. This block c this still is not enough, another address block, contains ihe address of a block that cor Each of those single indirect blocks points to this is not enough, a triple indirect block can is given m Fig 6-38. When a file is opened, the file system in locate its disk blocks. Let us consider how th up. We will use UNIX as an example, bul thf all hierarchical directory systems- First the ti In UNiX its i-node is located at a fixed pl using a generalization of l;ig 6-1." m II) disk addresses are stored in Ihe i- ;ary information is right in the i-node, y when the file is opened For sonc- node is" the address of a disk block contains additional disk -addresses If the i-node. called a double indirect ontains a list ot single indirect blocks o a few hundred data blocks [t e\en in also be used. The complete picture x-ates the root directory which can be anywhere on the i Then tt reads the root 1 take ihe file name <,uppl iath name /u\i/a\t/mbo\ is Igonthm is basically the same ft system locates the root direuoi; on the disk >ked this isk, but say block 1 in >t the path. nib' t 1 e and looks up the first compon fnd h i-nodu number of the file /w,w. Locating an i- e e tern locates the directory for /usr and looks up When it has found the entry tor as!, it hds the i- F m this i-node it can find the directory itself and de |or this file is then read into memory and kept there Until the file is closed. The lookup process is illustrated in FLg. 6-19. Relative path names are looked up the same way as absolute ones, only start- starting from the working directory instead of starting from the root directory Every directory has entries for . and .. which are put there when the directory is created. The entry . has the i-nodc number for the current directory, and the entry for . has the i-node number for the parent directory. Thus, a procedure looking up .Jdkk/prog.c simply looks up .. in the working directory, finds the i-node number node froi the disk. the ne\t component node for th t look up mbox. The i-ni

Root 1 4 7 14 9 6 8 directory t»n dev lib etc usr imp Mode limes 132 6 19 30 5! 26 45 ectory dick enk lim ast bai Aisr'ast Mode i,meS 406 ai 26 64" 92 60 01 17 eciory 9ran.S books mbov mimx src block 132 Figure 6-39. The

<is the director)- system is concerned. the> arejusl ordinary ASCil strings, just the same as any other names 6.5 RESEARCH ON FILE SYSTEMS File systems ha\e always attracted more research than other parts of the operating system and that is still the case Some of this research relates to tile system structure. Log-structured file systems and related topics are populai (Matthews et al.. 1997; and Wang et al., I999r The logical disk spills the hie system into two distinct layets: the file system and the disk system (De Jonge et al., 1993) Building a file system from stackable layers is also a research topic (Heidemann and Popek, 1994). Chap [They allow new features to be added to the file system without having to redesign it from scratch (Karpovich el al., 1994. and Khalidi and Nelson, 1993) Another research topic that has achieved ,i certain amount or popularity is making measurements of Hie system contents and use. People have measured what the file size distribution is. how long files hve. vvhethei all files are accessed equally, how reads compare to writes, and many other parameters (Douceur and Bolosky. 1999; Gill et a] . 1994:Roselli aml.Lorch. 2000; and Vogels, 1999) Othw researchers have looked at file system performance and how to improve it using prefetching, caching, less copying, and other techniques Usually, these researchers make measurements, tind out where the bottlenecks are. eliminate at least one of them, and then make the measurements, on the improved swem to validate their results (Cao ct al., 1995; Pai et al., 2000, and Patterson ut al.'. f9951 An area that many people rarely think about until disaster strikes is file system backup and recovery. Here. too. some new ideas have come up tor better ways to do things (Chen et al.. 1996; Devarakonda et al., 19%; and Hutcrnnsou et al , 1^99). Somewhat related to tins is the question of what to do when a user removes a file—remove it or hide it? The Elephant file system, for example, never forgets (Santry et a]., 1999a; and Santry et .,1 , 1999b) 6.6 SUMMARY When seen from the outride, a file system is a collection ol files and direc- directories, plus operations on them Files can be read and written, directories can be created and destroyed, and files can be moved from director) to directory Most modem file systems support a hierarchical directory system, in which direuones may have subdirectories atid these may have subsubdirectones ad infinitum.

SUMMARY When seen from thy mside, a file system looks quite different. The file sys- system designers have to be concerned with how storage is allocated, and how the system keeps- irack ot which block goes with which file.

Possibilities include con- contiguous files, linked hsis. file allocation tables, and i-nodes. Different systems have different directory structures. Attributes can go in the direcion&s or some- somewhere else (e.g., an i-node). Disk space can be managed using free lists of bit- bitmaps File system reliability is enhanced by making incremental dumps and by having ;i program that can repair sick file systems. File system performance is important and can be enhanced in several ways, including caching, read ahead, and carefully placing the blocks of a file close to each other. Log-structured file systems also improve performance by doing writes in large units. Samples of file systems include ISO 9660. CP/M. MS-DOS, Windows 98, and UNIX. These differ in many ways, including how they keep track of which blocks go with which file, directory structure, and management of free disk space. 1. Give 5 different path names for the file /etc/paswd Hint. Think aboui the directory 2, in Windows, when a user double clicks on a file hsted by Windows Explorer, a \uo- grain is run and given ttiai hie as a parameter List two diflcrem wiiys" |he operating system could know which program torun° number, not one chosen at random These files began with a header, followed by the cmable files, whereas other file types had a m0re-or-|ess random magk number .is ihe first word'1 4, In Fig. 6-4, one of the attributes i> the record length Why does the iiperatinji system ever care about this' 5. |s the open system tall in UNIX absolutely essential" What would the consequences be of not having W> 6. Sysiems that support sequential files always have an operation to rewind iiles. Do systems thai suppon random access files need this too^1 7, Some iiperaiing systems provide ,i system call rename io give a hie a new name h ihere any difference at all between Using this call to rename a file, and |iist copying ihe Icle lo a new file wnti the new name, followed hy deleting the o|d one>' X. In some systems il |^ possible to map part of a file into memoiv ^Vhat restrictions must such sysiems impose*1 How is ihis partial mapping implemented?

450 H|.tS\STFMS I-HAP b 9. \ simple operating s>stcm only aipport- j single dnectop, hut allow, that director, to hu\e arbitrary m,mj files with arbiunrily long file name. Can something appmM- rtiatrug a hierarchical |,|e syslcm be Mnwl.itcti.' How' 1A. |n I NiX and Windows random access i> d,mc by luvinj: a special system tall thai ,tx>\l-s ihe "current position" pointer ..ssocuued with a tile u> j given r>vtc m the Me 1'n.po^ .in dltortuilix e wa> to do iandt,m jlccss tvilhoilt havmu- this system tall 13, One way in use tonngii(iui> allot at ion <A iIil- Ji^k and not sutlci from holes is tn coin pact the disk c\eiy time .i

file is leimned Since .ill life., aic tontigiu'Us, top>mg j speed Writing ihc tile back regunc [he s.itiir v,(,rk Assuming d <.cek mne oi *> <>l'8 KB. how lung d,,cs M uke to tead a tile itiio mam irit-tni.iy then vitile il luck to lull of a 16-GB disk' 14, In hglit i>f the .msufi to the iireMous qufMion. does Lonipjtiiug the di^k c^fi nuke 15, some digital consumci dewce^ need io stmc ii.itj, lor simple ,is tiles N;ime j a Uni- idea 16, 1 low does MS-DOS implement undom a^i_ess to liLes ' 17, Cons,dei ihe i-node sh()wn in Fig. 6-[i If it u.nt.ims 10 ducit .iddiesse^. „} 4 b>tes J8. It has been suggested tliat cffiujenijv could lie nupioved Jtul disk sp^tv sav^d hv .nir ing Ihe data of a short tile wiihm thf i-node Foi tlic t-ii,ide (it Tiy ft-11, lu,\\ mau\ hytet of data Could be stored inside the i-nodt" (V. Two computer wiencc siudent-.. t*arol>n and Elinor, arc h.ivini! j disfUSMOti jboni ,- nodes i_aroljri) inyintams iluu [iiemdries have gotten so Un'ne riuo sis cheap thai when i-node table, rather than search the entfre tLiblc to L,ce H t' ^ aheadv flicrc f'liiun disagrets Who is rightJ 20. Name one advantage oi hard links ovei s>mbolit links <mt[one .idv^niasc ol" symbolic links ov^r h;trd Imks. 21, Free disk -.pa^c cm \k kepi tr.itk of using ,i free list <i,r a h,miap Disk addresses

fllAP h PROBLhM.S 451 " nuuied I0OH OlKIO dfXIO 0000 (lite first blink is jsetl bv the n.oi directory) The s>s ic-m jIwjvs stJrche~ (or free blocks startin" .11 ihe lowest numbered block, so jftcr vmMUBfliiM whid^ises 6 blocks, the hitm.ip looks like thts 1IH M 100000 OOW (.1) 1"tIu «k written. UMiig*! bleaks ih) Fillip is deleted (t 1 hie C" is written. u,m« K blocks (<li hU-'/3 is tickled 23. Wlui would happen il the humap ,,r irtfc ltsl mnijminji ilic information <ibi,ut Iree disk blocks was cornpletcfy losi due to j cr;uh^ Is there .M\ way to iccover fioin Ihts disaster, ni is it byc-hye djsk° Disl'Uss \onr tins^ei t(ir L Nl\ .Tnci the FA1 -16 file svs~ [uii! separately. 24. Oliver Owl's night |oli j[the univuesiij tom|H]Linu (.cntct is 10 Lhjngt' ihc i;ipe? Used for overnighi data hackups. Wfiile wjitm;: lor eJL'li tape it> eumpleie, lie works oil wrilmg his thesis thjt proves Sliakcpejn-'\ pl<i>s were wntldi by uxliiitcircitnal visi- visitors ^'s texr proccssof rntis oil Ihc svslein being backed up since iriti(is the only nnc 25. We discussed making incremental dumps in some detail m the te\l In Windows it is easy 10 tell when 10 dump a file because tvety file lus an arclnvc hn This bn is miss- mg in I rN^X. tiow tio fjNIX hiicKup pn>^f(iii]s kiiow ^hicli lilCs to dump1' would ihc tuur bitmaps ol Fig 6-25 he different' 27. It hus' been suggested that the first part of eucli 1_'N]X file be kepi in Ihe same disk block as its i-nodc. What good would tins

do' 28. Consider Fig 6-26 Is « possible lhai ior stjine particufar hlotk numbet the counter in both hsts have the value 2 > How should this problem be coneued^ lound in the cache). It it takes I mseL id satisfy a request Irom ihe cache, bui 40 nisei. M>. A (loppy disk has 40 cylinders A seek tyke-, 6 msec per cylinder moved. If no oilfy consecutive A e follow one another in the file) will be about 11 cylinders apart, on the average. It. howevet. the operating system makes an attempt to cluster related blocks, the mean interblock distance can be reduced 10 2 cylinders, (for example} How long does i(take to read a |0fl-bloek file in both cases, if ihe rotational latency is 100 msec and the transfer time is 25 msec per block" msec, a rotational rme of !5,(XM) rpm, and 262,144 bytes per (rack. What are (he daia rates for block sixes of | KB. 2 KB, and 4 KB. respectively"

452 FILE SYSTEMS CHAP 6 32. A tertam file system uses 2-KB disk blocks The median iile si^e is 1 KB |i all files were exactly 1 KB, what fraction of the disk space would be wasted'1 Do von think the wastage" for a real file system will be higher than this number or lov.er than it' Explain your answer 33. TP/M was designed io run with a small floppy disk 3s the mass storage deuce Sup- Suppose that it were Io be ported to a modem computer with a large hard disk Wh.it is the largest possible disk that could be supported without changing Iht- si/.c ot ihe directory format shown in Fig/ 6-31*1 The fields within the directory *tnd other system paiameters may be changed if necessary What changes would you make" 34. The MS-DOS FAT-16 table contains MK entries. Suppose that one of the bits had been needed for some other purpose and that ihe table contained exactly 32,768 entries instead. With no other changes, whai would the largest MS-DOS file h;t\e heen under 35. Files in MS-DOS have to compete for spate in the FAT-16 ublc in memory It -mil- hie uses A entries, that is k entries that arc not available to an> other lite wlut con- constraint does this place on the total length ol all hies combined" 36. A UNIX file system has 1-KB blocks and 4-b\ie disk addiesscs Whai is ihe max- maximum file size if i-nodes contain |0 direct entries, and one ■•ingle, double, and triple indirect entry each1 37. How many disk operations are needed to fetch the i-node jor the 1,1c /u\rhi\t/coursns/os/iuindout P. Assume lhat the i-node for [lie root directory is in memory, but nothing else along the path is in memory Also assume ihat >il! direc- directors fit m one disk block 38i In many UNIX systems, the i-nodes are kepi ai the st.trt ol the disk An jliernjttue design is to allocate an i-node when a file n created and put the 1-nodu .it the siari ol ihe

first block of the file Discuss the pros and i_iins i>i this" alternative 39. Writc a program that inverses the bylcs of a tile so ih,it the Usi byie is now tirsi and the first byte is now last It must work with .m ..irr>itj\tiily long tile, but ti> to make m reasonably efficient 40. Write a program thai starts at a given directors .mil dt\(cuds the Lile nee |r(,m ihai poinl recording (he sizes of all the files it finds WIkh h is all done, 11 should pi mi a hisiogram ot the file sizes using a bin width "peutu'd as ,1 parameter ie g . with 1024. file sizes ofOto 1023 go in one bin. 1024 to 2'147 ^ m thenoM btn.cn:) 41. Write a program that scans all directories tn a L'NIX tile systctn and tmds and loCdtcs all t-nodes with a hard link count of two or more For each such lile. [[lasts tosetrK-i all lile names that point to the Hie 42. Write a new version ol the UNIX h program This version takes .is an argument one or more directory names and for each directory lists all the file, in that directors. one line per file. Each field should be lormatted in a reasonable v.,iy gisen its type List only ihe first disk address, if any. 43. Write CPrM in C or C++ Look on the Web to find inlormalion aboul it.

MULTIMEDIA OPERATING SYSTEMS Digital movies, video clips, and music are becoming an increasingly common way to present information and entertainment using a computer Audio and video h)t.s Liin he stored on a disk and played back on demand. However, their charac- characteristics are very different from the traditional text tiles that current file systems \^ere designni for. As a consequence, new kinds of Tile systems are needed lo handle them Strongei yet, storing and playing back audio and video puts new demands on the scheduler and other parts ol the operating system as well. In the sections that follow, we will study many of these issues and their implications for operating systems that are designed to handle multimedia. Usually, digital movies go under the name multimedia, which literally means" more than one medium. Under this definition, this book is it multimedia work. After all. it contains two media: text and images (the figures). However, most people use ihe term "multimedia" to mean a document containing two or more continuous media, that is media thai must be played back over some time interval In thus book, we will use the term multimedia m this sense. Another term that is somewhat ambiguous is "video." In a technical s,ense. it is just the image portion ol .1 movie (as opposed to ihe sound portion). In fact, camcorders and televisions often have two connectors, one labeled "'video" and one labeled "audio." since the signals arc separaie. However, the term "digital video"

normally refers to the complete product, with both image and sound Below we will use the term "movie" 10 refer to the complete product. Note that a nio\ie in this sense need not be a two-hour long film produced by a Hollywood 453

454 MULTIMEDIA OPERATING SYSTEMS CHAP 7 studio at a cost exceeding thai of a Boeing 747 A 30-scc news clip downloaded (rom CNN's home page over the Intcmci is also & movie under our definition We u ill also call these "video clips" when we are referring to very short movies 7.1 INTRODUCTION TO MULTIMEDIA Before getting imu the technology of multimedia, a lew words about its current and tuture use-, are perhaps helpful to set ihe stage. On a single computer, multimedia often means playing a prerecorded movie from a DVD (Digital Ver- Versatile Disk) DVDs arc optical disks that use the same 120-mm polycarbonate (plastic) blanks that CD-ROMs use. but are recorded at a higher density, giving a capacity or between i OB ami 17 GB. depending on ihe format. Another use of multimedia js for downloading video dips over ihe Internet. Many Web pages have items lhat can be clicked on io download short movies. Ai 56 Kbpf,. downloading even a short video clip lakes a long tune, but as faster dis- distribution technologies lake over, such as cable TV and ADSL (Asymmetric Digi- Digital Subscriber Loop), the presence of video clips on the Internet will skyrocket. Another aiea m which multimedia must be supported i.s in the creation ot videos themselves. Multimedia editing systems exist and for best performance need to run on an operating system that supports multimedia as well as traditional work- Yet another arena where multimedia is becoming important is in computer games, flames' often run short video clips to depict some kind of action. The clips are usually short, but there are many of them and the correct one is selected dynamically, depending on some action the user has taken These are increas- increasingly sophisticated Finally, the holy grail of the multimedia world is video on demand, by which people mean the ability for consumers al home to select a movie using their telev- television remote control (or iiiouse) and have it displayed on their TV set (or com- computer monitor) on (he spot. To enable video on demand, a special infrastructure is needed. In Fig. 7-1 we see two possible video-on-demand infrastructures. Each one contains three essential components: one or more video servers, a distribution network, and a set-top box in each house toi decoding the signal The video them back on demand Sometimes mainframes' are Used as video servers, since connecting, say. 1000 large disks to a mainframe is

straightfoiward, whereas con- connecting I 000 disks of any kind to a personal computer is a serious problem Much of the material in the following sections is about \ ideo servers and their operating The distribution network between the ustr and the video server must be capa- capable o| transmitting data at high speed and in real time. The design of such net- networks is interesting and complex, but falls outside the scope of this book We

SEC. 7.1 INTRODUCTION TO MULTIMEDIA 455 ..* \ \ C«T*|TV Figure 7-1. Video on demand using different local distribution technologies, (a) ADSL, (b) Cable TV. will not say any more about them except io note that these networks always use fiber optics from the video server down to a junction box in each neighborhood where customers live. In ADSL systems, which are provided by telephone com- companies, the existing twisted-pair telephone line provides the last kilometer or so of transmission. In cable TV systems, which are provided by cable operators, exist- existing cable TV wiring is used for the local distribution. ADSL has the advantage of giving each user a dedicated channel, hence guaranteed bandwidth, but the bandwidth is low (a few megabits/sec) due to limitations of existing telephune wire. Cable TV uses high-bandwidth coaxial cable (at gigabits/sec), but many users have to share the same cable, giving contention for it and no guaranteed bandwidth to any individual user. The last piece of the system is the set-top box, where the ADSL or TV cable terminates. This device is. in fact, a normal computer, with certain special chips

456 MULTIMEDIA OPF.RATING SYSTEMS CHAP 7 for video decoding and decompression. As a minimum, it contains, a CPU. RAM, ROM and interface to ADSL or the cable. An alternative to a set-top box is to use the customer's existing PC and display the movie on tlie monitor. Interestingly enough, the reason set-top boxes are even considered given that most customers probably already have a computer, is that video-on-demdnd operators expect that people will want to waldi movies in their living rooms, whiih usually contain a TV bui rarely a computer. From a technical perspective, using a personal computer instead of a set-top box makes far more sense since it is more powerful, has a laige difk, and has a far higher resolution display. Either way, we will otter make a distinction between the video server and the client process at the user t-'nd that decodes and displays the movie. In terms of systeni design, however, ii does not matter much if the client process runs on a set-top box or on a PC

For a desktop video editing system, all the processes run on the same machine, bui we will continue to use the terminol- terminology of server and client to make it clear which process is doing what Getting back to multimedia itself, it has two key characteristics, thai must be well understood to deal with it successfully. 1. Multimedia use.s extremely high data rates. 2. Multimedia requires real-time playback The high data rates come from the nature of visual and acoustic information. The eye and the ear can process prodigious amounts of information per second, and have to be fed ai that rate to produce an acceptable viewing experience The data rales of a few digital multimedia sources and some common hardware devices are listed m Fig. 7-2 We will discuss some or these encoding formats later in this chapter. What should be noted is the high data rates multimedia requires, the need for compression, and the amount of storage that is required For example, an uncompressed 2-hour HDTV movie fills a 570-GB file. A video server that stores 1000 such movies needs 570 TB of disk space, a nontnvral amount by current standards. What is also of note is that without data compression, current hardware cannot keep up with the data rates produced We will examine video compression later in this chapter. The second demand that multimedia puts on a system is the need for real-time data delivery. The video portion of a digital movie consists of some number of frames per second. The NTSC system, used in North and South America and Japan, runs at 30 frames/sec B9.97 for the purist), whereas the PAL and SECAM systems, used in most of the rest of the world, runs at 25 frames/sec B5.00 for the purist). Frames must be delivered at precise intervals of ea. 33 3 msec or 40 msec, respectively, or the movie will look choppy. Officially NTSC stands for National Television Siandajds Committee, but the poor way color was hacked into the standard when color television was invented

INTRODUCTION TO MULTIMEDIA GBfhr \ Device iMbpsJ 0 03 | _Fasf Ethernet l_ 100_ ~0 06 " EIDE disk ~f 1331 0.62" ATM OC-3 network J 156 I jSCSl UHraWide disk 32o IEEE 1394 (FireWirel 4Q0 Gigabit Ethernet ! 1000_ " 286 1 SCSI Ulira-160 disk ~]_ 1280_ Hgure 7-2. Soinu daw cites lnr mullinicJia Ji,d liiyh-periorm<mcc I/O devices Nole liiat I Mbps is I06 bils/sec but I GB is 2iA hytes has led to the industry l«ke that it really stands tor Never Twice the Same Color. PAL stands for Phase Alternating Line. Technically it is the best ot the systems. SECAM is used in France (and was intended to protect French TV manufacturers from foreign competition) and

stands for SEquenttel Couleur Avec Memoire. SECAM is also used in Eastern Europe because when television was introduced there, the then-Communist governments wanted 10 keep everyone from watching German (PAL) television, so they chose an incompatible system ttseconds in delivery times wl|] be noticeable Variability in delivery rates is called jitter and must be strictly bounded tor good performance. Note that jitter is not the same as delay If the distribution network of Fig 7-1 uniformly delays all the biu by exactly 5.000 sec, the movie will start slightly later, but will look fine. On the other hand, if it randomly delays frames by between 100 and 200 msec, the movie will look like an old Charlie Chaplin film, no matter who is star- starring The real-time properties, required to play back multimedia acceptably are often described by quality of service parameters They include average band- bandwidth available, peak bandwidlh available, minimum and maximum delay (which together bound the jitter), and btl loss probability. For example, a network opera- operator could offer a service guaranteeing an average bandwidth of 4 Mbps- 999i of the transmission delays in the interval 105 to 110 msec, and a bu loss rate of 10"|C1, which would be fine for MPEG-2 movies. The operator could also offer a cheaper, lower-grade service, with an average handwidth of I Mbps (e.g . ADSL), m which case the quality would have to be compromised somehow, possibly by lowering the resolution, dropping the frame rate, or discarding the color informa- information and showing the movie in black and white The most common way to provide qualuy ot service guarantees is to reserve capacily in advance for each new customer The resources reserved include a por- portion of the CPU, memory buffers, disk transfer capacity, and network bandwidth- If a s along and wants to watch a movie, but the video server

458 MULTIMEDIA OPHRA1 ING SYSTEMS CHAP 7 or network calculates thai il does not have sufficient capacity for unothei custo- mer. it has to reject the new uisiomer 10 avoid degrading the service to current customers- As a consequence, multimedia servers need resource reservation scheme*, and an admission control algorithm to decide when they can handle more work. 7.2 MULTIMEDIA FILES without any structure that the operating system knows about or cares about With muliimedia. the situation is more complicated To start with, video and audio are completely different. They arc captured by distinct devices (CCD chip versus microphone), have a ddferent internal structure (video has 25-30 trames/sec; audio has 44JO0 samples/see), and they

arc played back by different devices (monitor versus loudspeakers) Furthermore, most Hollywood movies are now aimed at a worldwide audi- audience, mo'it of which docs not speak English. The latter point is dealt with m one of two ways. For some countries', an additional sound track is produced, with the voices dubbed mto the local language (but not the sound effects) In Japan. n\\ televisions have two sound channels to allou the viewer to listen to foreign films in either the original language or in Japanese. A bulion on ihe remote control is used for language selection. In still other countries, ihe original sound track is Used, with subtitles in the local language In addition, many TV movies now pro\ide closed-rapti(in subtitles in English as well, to allow English-speaking but hearing-impaired people to watch the movie. The net result is that a digital movie may actually consist of many files: one video file, multiple audio files, and multiple text tiles with subtitles m various languages. DVDs have the capability tor sioring up to 32 language and subtitle files. A simple set of multimedia files is shown in Pig 7-1. We will explain ihe meaning of fast forward and rast backward laier m this chapter As a consequence, the file' system needs to keep tiack <>(multiple 'subfile,"." per file. One possible scheme is to manage each subfile as a traditional file (e g.. using an i-node to keep track ot us blocks) and to have a new data structure that lists all the subfiles per multimedia file. Another way is to invent a kind of two- dimensional i-node, with each column listing the blocks of each subtile In gen- eral. the organization must be such that the viewer cdn dynamically choose which audio and subtitle tracks to use at the time the movie is viewed. In ail cases, some vvay to keep the subfiles synchronized is also needed so that when the selected audio track" is played back it remains in sync with the video 1(the audio and video get even slightly out oi sync, the viewer may hear an aclor s words before or after his lips move, which is easily detected and fairly annoying.

MULTIMEDIA FILES JQQQHUJJIIE3 English subtitles [HeimBac | Helm Awe | Niceda, j Suies |Mo.areyQu| Crust To better understand how multimedia tiles are orgam/ed, it is neeessan to understand how digital audio and video work m some detail. We will now give an introduction to these topics. 7.2.1 Audio Encoding An audio (sound) wave is a one-dimensional acoustic (pressure) wave. When an acoustic wave enters th(; ear, the eardrum vibrates, causing the liny bones of the inner ear to vibrate along with it, sending nerve pulses to the brain These pulses are perceived as sound by Ihe listener. In a

similar way, when an acoustic wave strikes a microphone, the microphone generates an electrical signal, representing the sound amplitude as a function of time The frequency range of the human ear runs from 20 Hz io 20,000 Hz. although some animals, notably dogs, can hear higher frequencies The ear heart, logarithmically, s0 the ratio of two sounds with amplitudes A and B n, convention- conventionally expressed in dB (decibels) according to the formula If we define the lower limit ot audibility (a pressure of about 0.0003 dync/cnr) for a I -kHz sine wave as 0 dB, an ordinary conversation is about 50 dB and the

460 MI'l-TIMEDIA OPERATING SYSTEMS CHAP 7 ncl pain threshold is about 120 dB, a dynamic lange or a factor of I million any contusion, A and B above are amplitudes It we were to use the powei level, which is proportional to the square of the amplitude, the coefficient of the loga- logarithm would be 10, nOi 20. Audio waves can be converted to digital form by an ADC (Analog Digital Converter). An ADC takes an electrical voltage as input and generates a binary number as output- In Fig. 7-4(a) we see an example of a sm^ wave. To represent this signal digitally, we can sample it every A7" seconds, as shown by the bar heights in Fig. 7-4(b), If a sound wave is n<it a pure sine wave, but a linear super- superposition of sine waves where the highest frequency component present is /, then it is sufficient to make samples at a frequency 2/ This result was proven mathenui- ically by H Nyquist in 1924 Sampling more often is of no value since the higher frequencies that such sampling could detect are not present- Digttal samples are never exact. The samples of Fig. 7-4(c| allow only nine values, from -1.00 to +1.00 in steps of 0.25. Consequently. 4 bits are needed to represent all of them. An 8-bit sample would allow 256 distinct values. A 16-bn sample would allow 65.536 distinct values. The error introduced hy the finite number or bits, per sample is called the quantization noise If it is too large, the ear detects it. nr or i ear detects it. Two well-kno mpa ples of sampled knon examples of s . Pulse code modulat sund are the telephone und audio p is Ple code modulation is used within the telephone system arid uses 7-bit (North America and Japan) or 8-bit (Europe) samples 8000 times per second This system gives a data rate of 56,000 bps or 64.000 bps With only 8000 samples/sec, frequencies above 4 kHz arc lost. Audio CDs are digital with a sampling rale of 44,100 samples/set, enough to capture frequencies up to 22,050 H7, which is good tor people, bad for dogs. The maniples are |6 bits each, and are linear over the range of amplitudes. Note that

16-bit samples allow only 65.536 distinct values, even though the dynamic ranpe

SEC 7.2 MULTIMEDIA FILES 461 of the ear is about I million when measured m steps of the smallest audible sound. Thus using only 16 bits per sample introduces some quantization noise (although the lull dynamic range ts not covered—CDs are uoi supposed to hurt). With 44.100 samples/sec of 16 bits each, an audio CD needs a bandwidth or 705.6 Kbps for monaural and 1,41 1 Mbps tor stereo (see Fig. 7-2) Audio compression is pos- possible based on psychoacoustic models of how human hearing works A compres- compression df I Ox is possible using the MPHG layer 3 (MP1) system Portable music players for this format have been common in recent years. Digitized sound can easily be processed by computers in software. Dozens or" programs exist for personal computers to allow users to retord. display, edit, mix, and store sound waves frt>m multiple sources. Virtually a‖ professional simiid recording and editing is digital nowadays 7.2.2 Video Encoding The human eye has the property that when an image js flashed on [he ietina. it is retained for some number of milliseconds before decaying Ii a sequence of images is flashed at 50 or more images/sec, the eye does not notice that it is look- looking at discrete images All video- and film-based motion picture systems exploit this principle to produce moving pictures To understand video systems, it is besf to start with simple, old-tashtoned black-and-white television. To represent the two-dimensional image in from of it as a one-dimensional voltage as a function of lime, the camera scans an electron beam rapidly across the image and slowly down it. recording the light intensity as it goes. At the end of the scan, called a frame, the beam retraces. This intensity as a funciion of time is broadcast, and receivers iepeat the scanning process to reconstruct the image The scanning pattern used by both the camera and the receiver is shown in Fig. 7-5. (As an asidt. CCD cameras integrate rather than scan, but some cameras and alt CRT monitors do scan,) The exact scanning parameters' vary from country to country NTSC has 525 scan lines, a horizontal to vertical aspect ratio ot 4:3, and 30 frames/sec. The European PAL and SECAM systems have 625 scan lines, the same aspect ratio of 4:3. and 25 frames/sec. In both systems, the top few and bottom few lines are not displayed (to approximate a rectangular image on the original round CRTs). Only 483 of the 525 NTSC scan lines (and 576 or the- 625 PAL/SECAM scan lines) are displayed. While 25 frames/sec is enough to capture smooth motion, at that

frame rate many people, especially older ones, will perceive the image to flicker (because the old image has faded off the retina before the new one appears) Rather than increase the frame rate, which would require using more scarce bandwidth, a dif- different approach is taken. Instead of displaying the scan lines in order fromtop to bottom, first all the odd scan lines are displayed, then the even ones are displayed. Each of these half frames is called a field. Experiments have shown that although

Ml I.TIMEDIA OPFRATINC1 SY.STXMS Figure 7-5. flic d for NTSC udeo and te people ncitice flicker at 2*5 fran\es/sec, they do not notice it ;il 50 fields/sec This technique is called Interlacing N on i merited television or video is said to be progressive Color video uses the same scanning pattern as monochrome (hlaclv and white!, except that instead of displaying the image with one moving beam, three beams moving in unison are used. One beam is used for each or thi: three addinve pri- primary colors: red, green, and blue (RGB). This technique works because any color can be constructed from a linear superposition oi" red, green, and blue With the appropriate intensities. However, for transmission on a single channel, the three color signals must be combined into a single composite signal three systems linearly combine the RGB signals mui ii luminance (brightness) signal, and two chrominance (color) signals, although they .ill use ditteriaU coet- ftcienis tor constructing thesis signals from the RGB signals. liiierestingW enough, the eye is much more sensitive to Hie luminance signal than to the chrom- chrominance signals, so the latter need not be transmitted as accurately. Consequently, the luminance signal can be broadcast at ihe same frequency as the old black- and-white signal, -so it can be received on hlack-and-whne television set* Th^ two chrominance Signals arc broadcast hi narrow bands at higher frequencies. Some television sds Viave knobs, or controls labeled brightness, hue, and satura- saturation lor brightness, tint and color) for controlling these three signals separately

Sr,C 7 2 MULTIMEDIA FILES 4oJ Understanding luminance and chrominance is necessary lor understanding Ik'W video compression works So tar we have looked at analog video. Now let us turn to digital video. The simplest representation of digital video is ,i sequence of frames, each consisting of a rectangular grid of picture elements, or pixels For color video. 8 bits per pixel are Used lor each of the RGB colors, giving 16 million colors, which is

unougb- To produce smooth motion, digital video, like analog video, musl display at least 25 trames/sec However, since good quality computer monitors often rescan the screen irom images stored in video RAM at 75 times per second or mt>r<;, interlacing is not needed Consequently, all computer monitors use progressive scanning Just repainting (i.e.. redrawing) the same trame three times in a row is enough to eliminate flickci. In othei words, smoothness of motion i* determined by the number of dif- different images per second, whereas flicker is determined by the number of titn^s the screen is painted per second These two parameters arc different. A still image painted a! 20 frames/sec will not show jerky motion but it will flicker because one frame will decay from the retina before the next one appears. A movie with 20 different frames pei second, each of which ts painted lour times in a row at 80 Hz. will not flicker, hm the motion will appeai jerky. The significance ot these two parameters becomes clear when we consider Hie bandwidth required for transmitting digital video over a network. Current com- computer monitors all use the 4 3 aspect ratio so they can Use inexpensive, mass- produced picture tubes designed for the consumer television maiket Common configurations are 640 x 480 (VGA). 800 x 600 (SVGA), and 1024 x 768 (XGA). An XGA display with 24 bits per pixel and 25 irames/sec needs 10 be fed at 472 Mbps. Doubling this rate to avoid flicker is not attractive A bettei solunon is to transmit 25 frames/see arid have the computer store each one and paint it twice Broadcast television does not use this strategy because television >;ets do not have memory, and in any event, analog signals cannot be stored in RAM without first converting them to digital form, which requires extra hardware. As a conse- consequence, interlacing is needed for broadcast television but not for digital video. 7.3 VIDEO COMPRESSION It should be obvious by now that manipulating multimedia material m uncompressed form is completely out of the question—it is much too big fhe Only hope is that massive compression is possible. Fortunately, a large body of research over the past lew decades has led to many compression techniques ;ind algorithms thai n»ake multimedia transmission feasible. In this section we will Study some methods for compressing multimedia data, especially images For more detail, see (Fluekiger, 1995; and Stcinmetz and Nahrstedt, 199.51.

 A]] compression systems require two algorithms, one for compressing the data at the source, and another for decompressing it at che destination. In the literature, these

algorithms are referred to as the encoding and decoding algorithms, respec- respectively. We will use this terminology here, too. These algorithms have certain asymmetries that are ,mportani to understand. First, for many applications, a multimedia document, say, a movie will only be encoded once (when it is stored on the multimedia server) but wtll be decoded thousands of times (when it is viewed by customers). This asymmetry means that tt is acceptable for the encoding algorithm to be slow and require expensive hardware provided that the decoding algorithm is fast and does not require expen- expensive hardware. On the other hand, for rea!-time multimedia, such as video con- conferencing, slow encoding is unacceptable- Encoding must happen on-the-fly. in real time. A second asymmetry is that the encode/decode process need not be invertible. That is, when compressing a file, transmitting it, and then decompressing it, the user expects to get the original back, accurate down to the last bit- With mul- multimedia, this1 requirement does not exist.]t is usually acceptable to have the video signal after encoding and then decoding be slightly different than the original When the decoded output is not exactly equal to the original input, the system is said to be lossy. All compression systems used for multimedia are lossy because they give much better compression. 7.3.1 The JPEG Standard The JPEG (Joint Photographic Experts Group) standard for compressing continuous-tone stili pictures (e.g., photographs) was developed by photographic experts working under the joint auspices of ITU. [SO. and 1EC. another standards, body. It is important for multimedia because, to a first approximation, the mul- multimedia standard for moving pictures. MPEG, is just tht; JPEG encoding of each frame separately, plus some extra features for interframe compression and motion compensation JPEG is defined in International Standard 10918. It has four modes and many options, but we will only be concerned with the way it is used for 24-bit RGB video and will leave out many of the detail. Step 1 of encoding an image with JPEG is block preparation. For the sake of specificity, let us assume that the JPEG input is a 640 x 480 RGB image with 24 bits/pixel, as shown in Fig. 7-6(a). Since using luminance and chrominance gives belter compression, the luminance and two chrominance signals are computed from the RGB values. For NTSC they are called Y, I, and Q, respectively. For PAL they are called Y, U, and V, respectively, and the formulas are different. Below we will use the NTSC names, but the compression algonthm is the same. Separate matrices are constructed for Y, I, and Q, each with element.', in the range 0 to 255. Next, square blocks of four pixels are averaged in the / and Q matrices to reduce

them to 320 x 240. This reduction is lossy, but the eye barely

VIDEO COMPRESSION r RGB —— 640 ll (a) 24-Bit pixel Figure 7-6. (j| RGB input ddt Block 4799 (bj) After block preparatu notices it since the e\e responds to luminance more than to chrominance Ne\ trtheless ii compresses the data b\ a (actor or two. Now 128 is subtracted from eaLh element ot all three matrices to put 0 in the middle of the range. Finally each matm i*. druded up mto 8x8 blocks. The Y matrix has 4800 blocks the other two hne 1200 blocks each, as shown m Fig. 7-6(b). Step 2 of JPEG is io apply a DCT (Discrete Cosine Transformation) to each of the 7200 blocks separately The output of each DCT is an 8 x 8 matrix of DCT coefficients. DCT element @, 0) is the average value of the block. The other ele- elements tell how much spectral power is present at each spatial frequency. In ^1 .. .. _ ni^T . I **... I ~_ 1 .__ ■ ....~._ i.i~...._.~ -."—* I J ■—..-.,, ments tell how much spectr theory, a DCT is lossless, but in practice using floating-point numbers and tran- transcendental functions always introduces some roundoff error that results in a little information loss. Normally, these elements decay rapidly with distance from the origin, @, 0). as suggested by Fig- 7-7(b). Once the DCT ,s complete. JPEG moves on to step 3, which is called quanti- quantization, m which the less important DCT coefficients are wiped out. This (lossy)

466 Ml I TIMF-DIA OPERATING SYSTEMS CHAP transformation is done by dividing each ol the coefficients in the 8x8 DCT matrix hy a weight taken from a lable It all the weight* are 1. the transformation does nothing However, if the weights increase sharply from the origin, higher spatial frequencies are dropped quickly. An example ot this step is given in Fig 7-8. Here we sec the initial DCT matrix the Quanti7iiuoii tab)e and the resuli obtained by dividing each DCT ele- element by the corresponding ijuantizanon table element The values in the quanti- quantization table are not pint oi the JPhG standard Each application must supply its own (|uanti/ation table, giving it the ability to control its own loss-compression tiacie-ott OuantJnation table 150 1 0 38 1 0 40 I o o 14 o 0 4 6 • o 0 2 ° n 0 ! 0 ° o o - -i 60 92 » o » 0 1—<P 0 o 0 -3" o| o [o | o [o 0 0 u 0 I 1 j I ! Z\ 4 M J 8 |64|64 8 « 32 | 32 64 64 32l_64_ 32 U#_ 32 64 3,164 «|« Figure 7-8. Computation uf ihc i d DCT (.peffioe Step 4 reduces the @, 0) value of each block (the one in the upper left-hand corner) by replacing it with the amount it differs from the corresponding element in the previous block. Since [hcse elements are the averages of their respective

blocks, they should change slowly, so taking the differential values should reduce most ()t them to small v a lues Ino dit fcrentials are computed from the other values. The @. 0) values are referred to as the DC components: the other ..allies are the AC components. Step 5 linearizes the M elements and applies run-length encoding to the list Scanning the block from left to right and then top to bottom will not concentrate the zeros together, so a zig zag scanning pattern is used, as shown in Fig. 7-9. Iri this example, the zig zag pattern ultimately produces 38 consecutive Os ai the end of the matrix. This String can be reduced to a single count saying there are IS Now we have a list ol numbers thai represent the image (in transform space) Step 6 Huffman encodes the numbers for storage or transmission JPEG may seem complicated, but thai is because it h complicated Still, since it often produces a 20:1 compression or better, it is widely used. Decoding a JPEG image requires running the algorithm backward. JPEG is roughly sym- symmetric- it takes about as long to decode an image as to encode it.

VIDEO COMPRESSION 92 3 1 0 0 0 "IT 75 2 0 0 0 0 "liT- 18 13 a 0 0 □ 0 3 1 0 0 0 0 ' 0 0 0 0 0 0 0 0 0 0 °_L° 0 ° 0 0 0 0 0 0 0 0 0 o 0 0 0 0 Figure 7-9. I he order in whii_h (he qujnli/ed vjlues arc ir 7.3.2 The MPEG Standard : [he MPEG (Motion Picture 1 algorithms used to comprew 2 1993- MPEG-t (International :r-quality output A52 x 240 for Finally, we come to the heart of (he ma Experts Group) standards. These arc the i videos and have been international standards s> Standard I tl72) was designed for video ree< NTSC) using a bit rate of I -2 Mbps- MPEG-2 (International Standard 138 18) v designed for compressing broadcast quality video into 4 to 6 Mbps. so it could fit in a NTSC or PAL broadcast channel. Both versions take advantages of the two kinds nf redundancies that exist in movie.s: spatial and temporal. Spatial redundancy can be utilized by simply cod- coding each trame separately with JPEG. Additional compression can be achieved by taking advantage of the tact that consecutive frames are often almost identical (temporal redundancy). The DV (Digital Video) system used by digital camcord- camcorders uses only a JPEG-]ike scheme because encoding has tu be done in real time and it is much faster to just encode each frame separately. The consequent-*;- of this decision can be seen in Fig. 7-2: although digital camcoiders have a lower data rale than uncompressed video, they art not nearly as good as full MPECi-2 (To keep the comparison honest, note lhat DV camcorders sample the luminance with 8 bits and each chrominance signal

with 2 bits, but there is still d factor of live compression using the JPEG-like encoding.) actors are moving around slowly, nearly all the pixels will be identical from, frame to frame. Here, just subtracting each fiame from the pre\ious one and running JPEG on the difference would do fine. However, for scenes when- the cdnieni is panning or zooming, this technique fails badly What impensate for this n differ e between MPEG and JPEG sly what MPEG does; in fact, this is the

468 MULTIMEDIA OPERATING SYSTEMS CHAP. 7 MPEG-2 output consists of three different kinds of frames that have to be pro- processed by the viewing program: 1 I (Intracoded) frames: Self-contained JPEG-encoded still pictures. 2. P {Predictive) frames: Block-by-block difference with the last frame. 3. B (Bidirectional) frames: Differences with the last and next frame. 1 frames are just still pictures coded using JPEG, also using full-resolution luminance and half-resolution chrominance along each axis. Ii is necessary to have 1-frames appear in the output stream periodically for three reasons. First, MPEG can be used for television broadcasting, with viewers tuning in at will, if all frames depended on their predecessors going back to the first frame, anybody who missed the first frame could never decode any subsequent frames. This would make it impossible for viewers to tune in after the movie had started. Second, if any frame were received in error, no further decoding would be possi- possible. Third, without l-frames, while doing a fast forward or rewind, the decoder would have to calculate every frame passed over so it would know the full value of the one it stopped on. With I-fraines, it is possible to skip forward or backward until an 1-frame is found and start viewing there. For these reasons, 1-frames are inserted into the output once or twice per second. P-frames, in contrast, code interframe differences. They are based on the idea of macroblocks, which cover 16x16 pixels in luminance space and 8x8 pixels in chrominance space. A macroblock is encoded by searching the previous frame for it or something only slightly different from it. An example of where P-frames would be useful is given in Fig. 7-10. Here we see three consecutive frames that have the same background, but differ in the position of one person. The macroblocks containing the background scene will match exactly, but the macroblocks containing the person will be offset m posi- position by some unknown amount and will have to be tracked down. ■ji. IT -1-4 i Figure 7-10. Three consecutive video frames The MPEG standard does not specify how to search, how far to search, or how good a match has to be to

count. This is up to each implementation. For example, an implementation might search for a macroblock at the current position in the previous frame, and all other positions offset ±Ax in the jt direction and ±Av

SRC 7 3 VIDEO COMPRESSION 469 in the \ direction For each position, the number of matches in the luminance matrix could be computed. The position wiih the highest score would be decided the winner provided it was above some predefined threshold. Otherwise, the macroblock would be said to be missing. Much more sophisticated algorithms are ll a maeroblock is lound, it is encoded by taking the difference with its value in the previous frame (for luminance and both chrominances). These difference matrices are then subject to the JPEG encoding. The value for the macroblock in ihe output stream rs then the motion vector (how far the macroblock moved from its previous position in uach direction), followed by the JPEG-encoded differences with the one in the previous frame. If the macrobiock is not located in the previ ous frame, the current value is encoded with JPEG, just as in an l-frame B-irames are similar to P-frames, except thai they allow the refetente macroblock to be in either a previous frame or in a succeeding frame, either in an I-frame or in a P-frame. This additional freedom allows unproved motion torn pcnsation. and is also useful when object-, pass in from of, or behind, other objects. For example, m a baseball game, when the third baseman throws the ball to first base, there may be some frame where the ball obscures the head of ihe moving second baseman m the background In the next frame, the head may be partially visible to the left of the ball, with the next approximation of the hiad being denved from the following frame when the ball is now past the head. B- frames allow a frame to be based on a future framt To do B-frame encoding, the encoder needs to hold three decoded trames in memory at once; the past one, the current one, and the future one. To simplify decoding, frames must be present in the MPEG stream in dependency order rather than in display order. Thus even with perfect timing, when a video is viewed over a network, buffering is required on the user's machine to reorder ihe frames for proper display. Due to this difference between dependency order and display order, trying to play a movit; backward will not work without considerable buffering and complex algorithms. 7.4 MULTIMEDIA PROCESS SCHEDULING Operating systems that support multimedia differ from traditional ones in three main ways: process scheduling, the tile system, and disk scheduling We will start with process scheduling here and continue with the other topics in siibse-

7,4.1 Scheduling Homogeneous Processes The simplest kind of video server is one that can support the display of a fixed number of movies, all using the same frame- rate, video resolution, data rale, and other parameters. Under these circumstances, a simple, but effective scheduling

MULTIMEDIA OPERATING SYSTFMS algorithm , (or thread) whose job it is lo read the movie irom the disk one frame ai a time and then transmit that frame to the user. Since all (he processes are equally important, have tlie same amount of work to do per frame, and block when the> have Im- ished processing the current frame, round-robin scheduling does (lie job just fine. The only addition needed to standard scheduling algorithms is a timing mechan- mechanism kj make sure each piocess runs at the correct frequency. One way to achieve the proper timing is lo have a master clock that ticks at. say. 30 times per second (for NTSC). A! every tick, all the processes are run sequentially, in the same order When a process has completed its work, it 1ssues a suspend s'ystem call that releases the CPU until th£ master clock I/ckl again When that happens, all the processes are run again in the same order. As long as the number of processes is small enough thai all the work can he done in one frame time, round-robin scheduling is sufficient. 7.4.2 General Real-Time Scheduling Unfortunately, this model is rarely applicable in reality The number of ust>r> changes as viewers come and go. frame sizes vary wildly due to the nature of video compression A-frames are much larger than P- or B-frames). and different movies may have different resolution^ As a consequence, different processes may have to run at different frequencies, with different amounts of work, and with different deadlines by which the work must be completed These considerations lead to a different model, multiple processes competing for the CPU, each with its own work and deadlines In the lollowing models, we will assume that the system knows the frequency at which each process must run. how much work it has to do, and what Us next deadline ,s. (Disk scheduling is also an issue, but we will consider that later.) The scheduling ot multiple compet- competing processes, some or all of which have deadlines that must be met is called real-time scheduling. As an example of the kind of environment a real-time multimedia scheduler works in. consider the three processes. A. B, and C shown in Fig. 7-] |. Process A rum every 30 msec (approximately NTSC speed). Each frame requires 1() msec of CPU time. In the absence of competition- it would ,un in the bursts ,41. A2.A3, etc., each one starting 30 msec after the previous one. Each CPU burst handles i>ne

frame and has a deadline it must complete before the next one is to sfait. Also shown in Fig 7-1 I are two other processes. B and C Process B runs 25 times/see (e.g., PAL) and process C runs 20 times/sec (e.g., a slowed down NTSC or PAL stream intended for a user with a low-bandwidth connection to the video server). The computation tune per frame is shown as 15 msec and 5 msec fot li and C. respectively, just to make the scheduling problem more general than hav- having all of them the same.

MULTIMEDIA PROCFSS .SCHEDULING T l/ 20 30 40 50 60 70 100 110 120 130 140 Figure 7-11. Thru The scheduling question now is how to schedule A, B, and C to make sure they meet their respective deadlines. Before even looking tor a scheduling algo- algorithm, we have to see if this set ot processes is schedulable at all. Recall from Sec. 2 5.4, that if process i has period P, msec and requires C, msec of CPU time per frame, the system is schedulable if and only if IfSI where m is the number of processes, in this case. ^ Note thai P,IC, is just the fraction of the CPU being used b> process i For the example of Fig. 7-1 I, A is eaimg 10/30 of the CPU, B is eating 15/40 of the CPU. and C is eating 5/50 of the CPU. Together these tractions add lo 0.808 of the CPU. so the system of proc- processes is sehedulable. So far we assumed [hat theie is one process per stream Actually, there inighl be two (or more processes) per stream, toi example, one tor audio and one for video. They may run at different rates and may consume differing amounts of CPU time per burst- Adding audio processes lo the mix does not change the gen- oral mode}, however, since all we are assuming is that ihere are i/i processes, each running a(a fixed frequency with a fixed amount ot work needed on each CPU burst. In some real-time systems, processes are preemptable and in others ihey aie not |n multimedia systems, processes are generally preemptable, meaning that a process thai is in danger of missing its deadline may interrupt the running processes before the running process has finished with its frame When it is done, the previous process can continue. This behav ior is jum multiprogramming, as we have seen before We will study preemptable real-lime scheduling algorithms because there is no objection to them in multimedia systems and they give better

 performance than nonpreemptable ones The only concern is that if a transmis- transmission buffer is being filled in little bursts, the buffer is completely full by the dead-

deadline so it tan be sent to the user in a single operation Otherwise jitter might be mtioduced. RL il time algOTithms tan be cither static or dynamic. Static algorithms assign tdch process a fixed priority in advance and then do prioritized preemptive scheduling using those priorities. Dynamic algorithms do not have fixed priori- priorities Below we will study an example ot each type. 7.4.3 Rate Monotonic Scheduling The classic static real-time scheduling algorithm for preemptable, periodic processes is RMS (Rate Monotonic Scheduling) (Liu and Layland. 1973) It can be used for processes that meet the following conditions: 1. Each periodic process must complete within its period. 2. No process is dependent on any other process. 3. Each process needs the same amount of CPU time on each burst. 4. Any nonperiodic processes have no deadlines. 5. Process preemption occurs instantaneously and with no overhead. The first four conditions are reasonable. The last one is not, of course, but it makes modeling the system much easier RMS works by assigning each process a fixed priority equal to the frequency of occurrence of its triggering event For example, a process that must run every 30 msec C3 times/sec) gets priority 33. a process that must run every 40 msec B5 times/sec) gets priority 25 and a process g ready process, preempting the running process if need be. Liu and I proved that RMS is optimal among the class of static scheduling algorithms. Figure 7-12 shows how rate monotonic scheduling works in the example of Fig. 7-11. Processes, A, B, and C have static priorities, 33. 25, and 20. respec- respectively, which means that whenever A needs to run. it runs, preempting any other process currently using the CPU. Process B can preempt C, but not A. Process C has to wait until the CPU is otherwise idle in order to run. In Fig. 7-12, initially all three processes are ready to run The highest prioniy one. A, is chosen, and allowed to run until it completes at 15 msec, as shown in the RMS line. After it finishes. B and C are run in that order. Together, these processes take 30 msec to run, so when C finishes, it m time for A to run again. This rotation goes on until the system goes idle at f = 70.

MULTIMEDIA PROCESS SCHEDULING A B C HM- __A1_ B1 m AT H B1 A2 A? 1 B2 tC? A3 A3 A4 B3 A4 B3 | A m 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 Time (msec) —- Figure 7-12. An example i>f RMS and tDF real-time scheduling At f = 80 8 becomes ready and runs. However, at t = 90, a highei priority process. A, becomes leady, so it preempts 8 and runs until it is finished, at I = 100. At that point the system can choose between finishing 8 or

starling C. so it chooses the highest priority process, 8. 7.4.4 Earliest Deadline First Scheduling Another popular real-time scheduling algorithm is Earliest Deadline First EDF is a dynamic algorithm that does not require processes to be periodic, as does the rate monotonic algorithm. Nor does it require fhe same run time per CPU burst, as does RMS. Whenever a process needs CPU time, it announces its pres- presence and its deadline The scheduler keeps a list of runnable processes, sorted on deadline. The algorithm runs the first process on the list, the one with the elosest deadline. Whenever a new process becomes ready, the system checks to see if il.s deadline occurs before that of the currently running process. Tf so, the new proc- process preempts the current one. An example of EDF is given in Fig 7-12. Initially all three processes are ready. They are run in the order of their deadlines. A must finish by / =30, 8 must finish by / = 40, and C must finish by t = SO, so A has the earliest deadline and thus goes first. Up until 7 = 90 the choices are the same as RMS At i = 90, A becomes ready again, and its deadline is t = 120, the same as fl's deadline. The scheduler could legitimately choose either one (o run. but .since in practice, preempting 8 has some nonzero cost associated with it. it is belter to let 8 con- continue to run. To dispel the idea that RMS and EDF always give the same results, let us now look at another example, shown in Fig 7-13. In this example ihe periods of A. 8. and C are the same as before, but now A needs 15 msec of CPU time per burst instead of only 10 msec. The schedu]ability test computes the CPU utilization as

 0.500 + 0.375 + 0.100 = 0 975. Only 2 5^ of the CPU is |cfl over, but in theor the CPU is not oversubscribed and il should be possible to find a legal schedule. & -j""! 20 30 40 50 60 70 figure 7-13. Another mple of real-t e scheduling with RMS Ji With RMS, the priorities or the three processes are still 3.3. 25, and 20 a.s only the period matters, not the run time This time, B\ does not finish until i = 30, at which time A is ready to roll again. By the time A is finished, at / = 45, B is, ready again, so having a higher priority than C, it runs and C misucs its deadline, RMS fails. Now look at how EDF handles this case. At i = 30. there is a contest between A2 and C\. Because Cl's deadline is 50 antic's deadline is GO, C is scheduled. This is different from RMS, whereas higher priority wins At t = 90 ,4 becomes ready for the fourth time. A's deadline is the same as thai of the current process A20), so the scheduler has a choice of preempting or not. As before, it is better not to preempt it it is not needed, so S3 is allowed to

complete. In the example of Fig. 7-13, the CPU is 100% occupied up to t = 150. How- However, eventually a gap wi|] occur because the CPU is only 97. Wr utilized Since all the starting and ending times are multiples ol 5 msec, the gap will be 5 msec In order to achieve the required 2.5% idle time, the 5 msec gap wril have to occui every 200 msei:. which is why it does not show up in Fig. 7-1 V An interesting question is why RMS failed. Basically, using static priorities only works if the CPU utilization is not too high Liu and Lay land A973) proved that for any system of periodic processes, if then RMS is guaranteed to work. For 3, 4, 5, |0, 20. and |00, the maximum per- permitted utilizations are 0.780, 0.757. 0.743. 0 718. 0.705, and 0.6%. As m -> »,

SRC 7 4 MULTIMEDIA PROCESS SCHEDULING 47>S the maximum utilization k asymptotic to In 2. In other words. Liu and Lay land proved that for three processes, RMS always works if the CPU utilization is at or below 0 780 In our first example, ,1 was 0 808 and RMS worked, but we were just luck> With different periods and run tunes, a utilization cif 0.808 might fail In ihe second example, the CPU utilization was so high @.975), there was no hope that RMS could work. In contrast. EDF always works for an> schedulable set ol processes. It can achieve 100%- CPU utilization. The price paid is a more complex algorithm. Thus mi an actual video server, if the CPU utilization is below the RMS limit, RMS can he used. Otherwise EDF should be chosen. 7.5 MULTIMEDIA FILE SYSTEM PARADIGMS Now that we have covered process scheduling in multimedia systems, let us continue our study by looking at multimedia file systems. These file systems Use a dittereni paradigm than traditional file systems. We will first review traditional tile I/O, then turn our attention to how multimedia file servers are organized. To access a tile, a process first issues an open system call. If this succeeds, the caller is given some kind ot token, called a file descriptor in UNIX or a handle in Win-Windows to be used in future calls.. At thai poini the process can issue a read system call, providing the token, buffer address, and hyte count as parameter The operating system then re-turns the requested data in the buffer. Additional read calls can then be made until the process is finished, at which time it calls close to close the lile and return its resources. This model does not work well for multimedia on account of ihe need lor real-time behavior. It works especially poorly for displaying multimedia files coming off a remote video server. One problem is that the user must make the read calls fairly precisely spaced in time A second problem is that the video server inu.st be able to

supply the data blocks wuhout delay, something thai is dif- ticult for it to do when the requests come in unplanned and no resources have been reserved in advance. To solve the^e problems, a complete!) different paradigm is used by mul- multimedia file servers: they aci |,ke VCRs (Video Cassette Recorders). To read a multimedia fiie, a user process issues a start system call, specifying the tile to be read and various other parameters, tor example, which audio and subtitle tracks to use. The video server then begins sending out frames at ihe required rate It is up to the user to handle them at the rate they come m If the user gets bored with the movie, the Stop system call terminates the stream File servers with ihis stream- streaming model are otten called push servers (because ihey push data at the user) and are contrasted with traditional pull servers where the user has to pull the data in one block at a lime by repeatedly calling read to get one block after another. The difference between these two models is illustrated in Fig. 7-14.

MULTIMEDIA OPERATING SYSTEMS Figure 7-14. (si) A pull server (bj A push server 7.5.1 VCR Control Functions Most video servers also implement standard VCR control functions, including pause, fast forward, and rewind. Pause is tairly straightforward. The user sends a message back to the video server thai iei|s it to stop All it has, to do ai that point is, remember which frame goes oui next When ihc user tells (he server to resume, it just continues from where it let] off. However, there is one complicalion here To achieve acceptable perfor- performance, the server may reserve resources such as disk bandwidth and memory buffers for each outgoing stream. Continuing to tie these up while a movie is paused wastes resources, especially if the user is planning a trip to the kitchen to locale, microwave, cook, and eat a frozen pizza (especially an extra large) The resources can easily be released upon pausing, of course, bui this introduces the danger that when the user tries lo resume, they cannot be reacquired. True rewind is actually easy, wiih no complications- All the server has to do forward and fasl backward (i.e.. playing while rewinding) are much trickier. It it were noi for compression, one way to go forward at |0x speed would be to just display every 10th frame. To go forward at 20x speed would require displaying every 20th frame, in fact, in the absence or compression, going forward or back- backward ai any speed is easy. To run at k times normal speed, just display every k-lh frame. To go backward at k limes normal speed, do the same thing m the other direction. This approach works equally well for both pu|] servers and push servers. Compression makes rapid

motion either way more Complicated. With ;i cam- camcorder DV tape, where each frame is compressed independently of a|) the others,

SEC 7 •> MULTIMEDIA F|LE SYSTEM PARADIGMS 477 it is possible to use this strategy, provided that the needed fr;ime can be found quickly Since each frame compresses by a different amount, depending on its- content, each frame is a different si?e, so skipping ahead k frames in the tile can- cannot be done by doing a numerical calculation Furthermore, audio compression is done independently of video compression, so for each video frame displayed m high-speed mode, ihe correct audio frame must also be located (unless sound is turned off when running faster than normal). Thus fast forwarding a DV file requires an index that allows frames to be located quickly, but it is at least doable in theory. With MPEG, this scheme does not work, even in theory, due to the use of I-, P-, and B-trames. Skipping ahead k frames (assuming that can be done at all), might land on a P-frame that is based on an I-frame that was just skipped over. Without the base frame, having the incremental changes from it (which is what a P-frame contains) is useless. MPEG requires the file to be played sequentially. Anolher way to attack the problem is to actually try to play the file sequen- sequentially at |0x speed. However, doing this requires pulling data off the disk at |Ox speed. At that point, the server could try to decompress the frames (something it normally does not do), figure out which frame is needed, and reconipress every 10th frame as an I-frame. However, domg this puts a huge load on the server. It also requires the server to understand the compression format, something it nor- normally does not have t<i know. The alternative of actually shipping all the data over the network to the usei and letfing the correct frames be selected out there requires running the network at lOx ^peed, possibly doable, but certainly not easy given the high speed at which it normally has, to operate. All in all, there is no easy way out The only feasible strategy requires ad- advance planning. What can be done js build a special file containing, say, every 10th frame, and compress this file using the normal MPEG algorithm. This file is what is shown in Fig. 7-3 as "fast forward." To switch to fast forward mode, what the server must do is, figure out where in the fast forward file the user currently is. For example, if the current trame is 48,2 |0 and the fast forward file runs at lOx, the server has to locate frame 4821 m the fast forward file and start playing there at normal speed. Of course, that frame might be a P- or B-frame. but the decoding process at the client can just skip frames until it sees an I-frame. Going

backward is done in an analogous way using a second .specially prepared file. When the user switches back to normal speed, the reverse trick has to be done. If the current frame m the fast forward file is 5734. the server just switches back to the regular file and continues at frame 57.340. Again, if this frame is not an I-frame, the decoding process on the client side has to ignore all trames until an I-frame is seen. While having these two extra files does the job, the approach has Mime disad- disadvantages. First, some extra disk space is required to store the additional files. Second, fast forwarding and rewinding can only be done at speeds corresponding

 to the special files. Third, extra complexity is needed to switch back and forth between the regular, fast forward, and fast backward files. 7.5.2 Near Video on Demand Having it users getting the same movie puts essentially the same load on the server as having them getting it different movies. However, with a small change in the model, great performance gains are possible. The problem with video on demand is that user.s can start streaming a movie at an arbitrary moment, si) il there are 100 users all starting to watch some new movie at about 8 P.M., chances are that no two will start at exactly the same instant so they cannot share a stream. The change that makes optimization possible is to tell all users that movies only start on the hour and every (for example) 5 minute*; thereafter. Thus if a user wants to see a movie at 8:02, he will have lo wait until 8:05 The gain here is that for a 2-hour movie, only 24 streams are needed, no matter how many customers there are. As shown in Fig 7-15, the first stream starts at 8:00 At 8:05. when the first stream is at frame 9000, slream 2 starts. At 8: |0. when the first stream ,s a(frame 18,000 and stream 2 is at frame 9000. stream 3 starts, and so on up to stream 24, which starts at 9:55. At 10:00, stream I terminates and starts ail over with frame 0. This scheme is called near video on demand because the video does not quite start on demand, but shortly thereafter. The key parameter here is how often a stream starts. If one starts everv 2 minutes, 60 streams wi‖ be needed for a two-hour movie, bill the maximum wait- waiting time to start watching will he 2 minutes The operator has to decide how long people are willing to watt because the longer they are willing to wait, the more efficient the system, and the more movies can be shown at once. An alternative strategy is to also have a no-wail option, in which case a new stream is started on the spot, but to charge more for instant startup. In a sense, video on demand is like using a taxi- you call is and it

comes. Near video on demand is like using a bus: it has a fixed schedule and you have to wail for the next one. But mass transit only makes sense if there is a mass. In mid- town Manhattan, a bus that runs every 5 minutes can count on picking up at least a few riders. A bus traveling on the back roads of Wyoming might be empty nearly a‖ the time. Similarly, starting the latest Steven Spielberg release might attract enough customers to warrant starting a new stream every 5 minutes, but for Gone with the Wind it might be better to simply offer it on a demand basis. Wjih near video on demand, users do not have VCR controls. No user can pause a movie to make a trip to the kitchen The best th.n can be done is upon returning from the kitchen, to drop back !o ;i stream that started later, thereby repeating a few minutes of material. Acrually, there is another model for near video on demand a.s well Instead of announcing in advance that some specific movie will start every 5 minutes, people can order movies whenever they want to. Every 5 minutes, the .system sees which

MULTIMEDIA FILE SYSTEM PARADIGMS I pTsOOol [270001 [36000] |45000| E40001 [630001 [72000] [siOOp] [JkjOo] rieOOOl [g7QQ0i [360O0i 1450001 f&40ob1 [63000] |7gQ00| |_0_| | 900Q | jiaoOQf }g70QO| [360001 [450001 1540001 [630001 r~o~n fioool fia^ool [g70oo| |36ooo| [45000] 1540001 ~o~| HooTI [iaooo] [27000! [36000I [0 I [90001 haoogj B70001 j_o_| [9oqo jiaooof CO 3 00 8 05 8 10 8-20 8 25 8 30 Figure 7-15. Near video on demand lus j nciv slream starting ai icpular inter vaK in this example ever> 5 minulcs (9000 frames) movies have been ordered and starts those. With this approach, a movie may start at 8:00, 8:10, 8:|5. and 825, but not at the intermediate times, depending on demand. As a result, streams with no viewers ^ire not transmitted, saving disk bandwidth, memory, and network capacity On the other hand, attacking the freezer is now a bit ot a gamble as there is no guarantee that there h another stream running 5 minutes behind the one the viewer was watching Of course, the operator can provide an option for the usei to display a list ot all concurrent streams, but most people think their TV remote controls have more than enough buttons already and are not likely to enthusiastically welcome a few more 7.5.3 Near Video on Demand with VCR Functions The ideal combination would be near video on demand (ior the efficiency) plus full VCR controls for every individual viewer (tor the user convenience). With slight modifications to the model, such a design is possible Below we will give a slightly Simplified description of one way to achieve this goal (Abram- Profeta and Shin, 1998). We start out with the standard near

video-on-demand scheme of Fig 7-15. However, we add the requirement that each client machine butter the previous A7" mm and also the upcoming AT m,n locally. Buffering the previous AT min is

480 MULTIMEDIA OPERATING SYSTEMS CHAP. 7 easy, juu save it after displaying it. Buffering the upcoming A7' nun is harder, but can be done if clients have the ability to read two streams ai once. One way to get the buffer set up can be illustrated using an example. It a user 'hich is at frame 0). In parallel, it reads and stores the 8:|0 stream, which js, rrently at the 5-min mark (i.e., frame 9000) At 8.20, frames 0 to 17,999 have been stored and the user is expecting to see frame 9000 next From lhat point on, :he 8:15 stream is1 dropped, the buffer is filled from the 8:10 stream (whtch is at 8,000), and the display is driven from the ,nidd|e of the buffer (frame 9000) As ich new frame is read, one frame is added k> the end of the buffer and one frame dropped from the beginning of the bufiW The current frame being displayed, tiled the play point, is always in the middle uf the buffer. The situation 75 inin into the movie is shown ,n Fig. 7-l6(aj. Here all frames between 70 nun and HO mm are in the buffer. It the data rate is 4 Mbps, a 10-min buffer requires 100 mil-million bytes of storage With current prices, the buffer can certainly be kept on disk and poss,b|y m RAM. If RAM is desired, but 300 million bytes is too much, a smaller buffer can be used. rp ing 3 mm (d) After starting (o refill the bufiei fcj Butler full Now suppose that the in,er decides to fast torward or fait reverse. As long a the play point stays within the range 70-80 min, the display can be fed from (h

SEC 7 *) MULTIMEDIA FILE SYSTEM PARADIGMS 481 buffer. However, if the play point moves outside that interval either way, we have a problem. The solution i-s to turn on a private (j e.. video-on-demand) stream to service the user Rapid motion in cither direction can be handled by the tech-techniques diseased earlier. Normally, at some point the user wi‖ settle down and decide to watch the movie at normal speed again At this point we can think about migrating the user over 10 one of the near video-on-demand streams so the private stream can be dropped. Suppose, for example, (hat the user decides lo go back to the 12 min mark, as shown m Fig 7-l6(b). This point is far outside the buffer, so the display cannot be fed trom it Furthermore, since the switch happened (instantaneously) at 75 mm, there are streams showing the movie at 5, 10, [5, and 20 min, but none The solution is to continue viewing on

the private stream, but to start filling the buffer from the stream currently 15 minutes into the movie After 3 minutes, the situation is as depicted in Fig 7-16(c). The play point is now |5 mm, the buffer contains minutes 15 to [8, and the near vjdeo-on-demand streams arc at 8, 13, 18, and 23 nun, among others. Al this point the private stream can be dropped and the display can be fed from the buffer. The buffer continue;, to be tilled from the .stream now at 18 min. After another minute, the play point is |6 mm, I he buffer contains" minutes 15 to 19, and the stream feeding the buffer k al 19 mm, as shown in Fig. 7-16(d) After an additional 6 minutes have gone by, the buffci ,s full and the play point is at 22 min. The play point is not in the middle of the buffer, although that can be arranged if necessary. 7.6 FILE PLACEMENT Multimedia files are very large, are often written only once but read many times, and tend to be accessed sequentially Their playback must also meet strict quality of service criteria. Together, these requirements suggest different file sys- system layouts than traditional operating systems use. We will discuss some of these issues below, first for a single disk, then for multiple disks. 7.6.1 Placing a File on a Single Disk The most important requirement is that data can be streamed to the network or output device at the requisite speed and without j,tter For this reason, having multiple seeks during a frame is highly undesirable. One way to eliminate intrafile .seeks on video servers is to use contiguous files. Normally, having files be contiguous, does not work well, but on a video server that is carefully preloaded in advance with movies that do noi change afterward it can work.

482 MULTIMEDIA OPERATING SYSTEMS CHAP One complication, however, is the presence of video, audio, and text, as Shown in Fig. 7-3. Even if the video, audio, and text are each stored as. separate contiguous files a seek will be needed to go from the video file to an audio file and from there to a text file, if need be This suggests a second possible storage arrangement, W]th the video, audio, and text interleaved as .shown in Fig. 7-17, but the entire file still contiguous. Here, the video for frame I is directly followed by (he various audio tracks for frame I and then the various text tracks for frame I- Depending on how many audio and text tracks there are, it may be simplest just to read m all Ihe pieces tor each frame in a single disk read operation and only transmit the needed pans to the user. A .TIT,. Video A A A \t/ V This organization requires extra disk I/O for leading in unwanted audio and text, and extra buffer space in memory to store them. However il eliminates all seeks (on a single-

User system) and does not require any overhead far keeping track of which frame is where on the disk Mnce the whole movie is m one contigu- contiguous file. Random access is impossible with this layout, but if it is not needed, its loss is not serious. Similarly, fast forward and tast backward are iimpossible without additional data structures and complexity The advantage of having an entire movie as a single contiguous tile js lost on a video server with* multiple concurrent output streams because alter reading a frame from one movie, the disk will have to read in frames from many other movies before coming back to the first one. Also, for a system in which movies are being written as we|] as. being read (e.g., a system used tor video production or editing), using huge contiguous files is difficult to do and not that useful 7.6.2 Two Alternative File Organization Strategies These observations lead to two other file placement organizations tor mul- multimedia files. The first of these, the s,ma|| block model, is illustrated m Fig 1- |8(a). In this organization, the disk block .size is chosen to be considerably smaller than the average frame size, even for P-frames and B-traines. For MPEG-2 at 4 Mbps with 30 frames/sec, the average frame ts 16 KB. so a Mock Size of | KB or 2 KB would work well. The idea here is to have a data structure, the frame index, per movie with one entry for each frame pointing to the aart of

FILE PLACEMENT 483 the frame Each rraine itself consists of all the video, audio, and text tracks lor thai frame as a contiguous run of disk blocks, as shown. In this way. reading frame * consists of indexing into the frame index to find the fc-th entry, and then reading in the entire frame in one disk operation Since different frames have dif- different s,zes. the frame size (in blocks) is needed in the frame index, but even w I-KB disk blocks, an 8-bit field can handle a frame up to 255 KB, which enoueh for art uncompressed NTSC frame, even with many audio tracks. ith The other way to store the movie is by using a large disk block (say 256 KB) and putting multiple frames m each block, as shown m Fig. 7- |g(b). An index is still needed, but now it is a block index rather than a frame index- The index it., in fact, basically the same as the i-node of Fig f>-15, possibly with [he addition Of in- information telling which frame is at the beginning of each block to make it possible to locate a given frame quickly. In genera], a block will not hold an integral num- number of frames, so something has to be done to deal with this. Two options exist. In Che first option, which is illustrated in Fig. 7-|8(b), whenever the next frame does not fit in the current block, the res! of the block w just left empty. This

wasted space is internal fragmentation, the same as in virtual memory sys-systems wrth fixed-size pages. On (he other hand, it is never necessary to do a seek in the middle of a frame The other option ,s to fill each block to the end, splitting frames over blocks. This option introduces the need for seeks in the middle of frames, which can hurt performance, but saves disk space by eliminating internal fragmentation.

484 MULTIMEDIA OPERATING SYSTEMS CHAP. 7 For comparison purposes, the use of small blocks in Fig. 7- [8(a) also wastes some disk space because a fraction of the last block in each frame is unused. With a 1-KB disk block and a 2-hour NTSC movie consisting of 216,000 frames, the wasted disk space will only be about 108 KB out of 3.6 GB. The wasted space is harder to calculate for Fig. 7-18(b), but it will have to be much more because from (ime to time (here will be [00 KB left at the end of a block with the next frame being an l-frame larger than that. On the other hand, the block index is much smaller than the frame index. With a 256-KB block and an average frame or 16 KB, about [6 frames fit in a block, so a 2|6,000-frame movie needs only 13,500 entries in the block index, versus 216,000 for the frame index. For performance reasons, in both cases the index should list a‖ the frames or blocks (i.e. no indirect blocks as UNIX), so lying up 13.500 8-byte entries in memory D bytes tor the disk address. I byte for (he frame size, and 3 bytes for the number of the starting frame) versus 216,000 5-byte entries (disk address and size only) saves almost 1 MB of RAM while the movie is playing. These considerations lead to the following trade-offs: 1. Frame index: Heavier RAM usage white movie is playing; little disk wastage. 2. Block index (no splitting frames over blocks)- Low RAM Usage; major disk Wastage. 3. Block index (splitting frames over blocks is allowed): Low RAM usage; no disk wastage: extra seek^ Thus the trade-offs involve RAM usage during playback, wasted disk space all the time, and performance loss during playback due to extra seeks. These problems can be attacked in various ways though RAM usage can be reduced by paging in parts, of the frame table just in time. Seeks during trame transmission can be masked by sufficient buffering, but this introduces the need tor extra memory and probably extra copying. A good design has to carefully analyze a‖ these factors and make a good choice for the application at hand. Yet another factor here is that disk storage management is more complicated in Fig. 7-|8(a) because storing a frame requires finding a consecutive run of blocks the right size. Ideally, this run of blocks should not

cross a disk track boundary, but with head skew, the loss ts not serious. Crossing a cylindei boun- boundary should be avoided, however. These requirements mean thai the disk's free storage has to be organized as a list of variable-sized holes, rather than a simple block list or bitmap, both of which can be used m Fig 7-18(b|. In all cases, there is much to be said tor putting all the blocks or trames ot a movie within a narrow range, say a few cylinders, where possible. Such a place- placement means lhat seeks go faster so that more time will be left over for other (nonreai-nme) activities or for supporting additional video streams- A constrained

SEC 7 6 FILE PLACEMENT 485 placement of this son can be achieved by dividing the disk into cylinder groups and for each group keeping separate lists or bitmaps of the free blocks. If holes are used, tor example, ihere could be one list for 1-KB holes, one for 2-KB holes, one tor holes of 1 KB to 4 KB, another for holes of size 5 KB to 8 KB, and so on.]n this way it is easy to find a hole of a given size in a given cylinder group. Another difference between these two approaches, is buffering. With the small-block approach, each read gets exactly one frame. Consequently, a simple double buffering strategy works fine: one buffer for playing back the curreni frame and one for fetching the next one. if fixed buffers are used, each buffer has to he large enough for the biggest possible l-frame On the other hand, if a dif- terent buffer is allocated trom a pool on every frame, and the frame size is known before the frame is" read in, a small buffer can be chosen for a P-frame or B-frame. With large blocks, a more complex strategy is required because each block contains multiple frames, possibly including fragments of frames on each end of the block (depending on which option was chosen earlier). If displaying or transmitting frames requires them to be contiguous, they must be copied, but copying is an expensive operation so it .should be avoided where possible. If con- contiguity is not required, then frames that span block boundaries can be .sent out over the network or to the display device in two chunks. Double buffering can also be used wjlh large blocks, but using two large blocks wastes memory, One way around wasting memory is to have a circular transmission buffer slightly larger than a disk block (per stream) that feeds the network or display. When the buffer's contents drop below «mt threshold, a new large block is read in from the disk, the contents copied to the transmission buffer, and the large block buffer returned to a common pool. The circular butter's ">ize must be chosen so that when n hits the threshold, there is room for another tu|l disk

block The disk read cannot go directly to t|ie transmission butter because it might have to wrap around Here copy,ng and memory usage are being traded off against one another. Yet another factor in comparing these iwo approaches is disk performance. Using large blocks runs the disk at full speed, often a major concern. Reading in ht(|e P-frames and B-tVames as separate units is noi efficient In addition, striping large blocks over multiple drives (discussed below) is possible, whereas striping individual frame*, over multiple drives is not. The small-block organization of Fig. 7-l8(a) is sometimes called constant time length because each pointer m the index represents the same number of mil- milliseconds of playing time. In contrast, the organization of Fig. 7-!8(b) is some- sometimes called constant data length because the data blocks, are the same size. Another difference between the two file organizations is that if the frame types are stored in the index of Fig. 7- |8(a). it may be possible to perform a last forward by just displaying the i-frames. However, depending on how often 1- trames appear in the stream, the rate may he perceived as too fast or loo slow. In any case, wjth the organization of Fig. 7-l8(b) fast forwarding is not possible this

 way Actually reading the file sequentially to pick out the desired frames requires massive disk I/O. . A second approach is to use a special file that when played at iiciimal speed gives the illusion of fast forwarding at [Ox speed. This file can be structured ihe same as oiher files, using either a frame index or a block index. When opening a file the system has to he able to find the fast toward file if needed If the user hits the fast forward button, the system must instantly find and open the last lor- ward file- and then jump to the corfec! place in the tile. What it knows is [he irame- number it is currently at. but it needs the ability to locate the corresponding trame in the fast forward file. If it is currently at frame, s;ty, 4816. and it knows the fast forward tile is at [Ox, then it must locate irame 482 in that file and start playing from there. If a frame index is used, locating a specific frame is easy just index into the frame index. It a block index is used, extra mlorliiation in each entry is needed to identity which frame is in wliieh bli)Ck and a binary search of the block index has to be performed. Fast backward work'' in an analogous way lo last forward 7.6.3 Placing Files for Near Video on Demand So far we have looked at placement strategies tor video on demand. For near video on demand, a different file placement strategy is more efficient. Remember that the same movie is going out as

multiple staggered streams Even it the mmrie is stored as a contiguous file a seek is needed for each stream. Chen and Thapar A997) have devised a tile placement strategy to eliminate nearly all i,t those seeks. Its use is illustrated in Fig. 7-19 tor a movie running at 30 frames/sec wiih a new stream starting every 5 mm, as in Fig. 7-15 With these parameters, 24 concurrent streams are needed for a 2-hour movie. Track 1 | 0 | 9000 11BOOO127000136000145000|54000|6300o|72000|81000| |F070OO| Track 2 | 1 | 9001 |tS001[27001J360Q1j45001|540O1|63O01[72001161001j [2070011 Track 3 | 2 \ 9002 [iB0O2 [27002136002145002154002 163002172002181 OOg] |207002| V

SEC 7 6 FILE PLACEMENT 487 In this placement, Irame sets of 24 frames are concatenated and written to the disk as a single record. They can also be read back on a single read. Consider the instan! that stream 24 is |ust start,ng. It will need frame 0. Frame 23. which started 5 inin earlier. w,I| need frame 9000 Stream 22 will need frame |8,000, and so on hack to stream 0 which will need frame 207.000. By pulling these frame-, consecutively on one disk track, the video server can satisfy all 24 streams in ieverse order with only one seek (to frame 0). Of course, the frames can be reversed on the disk if there is some reason to service the streams in ascending order. After the last stream has been serviced, the disk arm can move to track 2 to prepare servicing ihem a|l again. This scheme does not require the entire file to be contiguous, but still affords good performance to a number of streams at once. A simple buffering strategy is to use double buffering. While one buffer is being played out onto 24 streams, another buffer is being loaded in advance. When the current one finishes, the two buffers are swjpped and tne one just used for playback is now loaded in a single disk operation. An interesting question is how large to make the buffer Clearly, it has to hold 24 frames However, since frames arc variable in size, it is noi entirely trivial to pick the righi size buffer, Making the butter large enough for 24]- frames is overkill, but making it large enough for 24 average frames is living dangerously. in the movie is known in advance, so a buffer of precisely that .size can he chosen, However, it might just happen that in the biggest track, there are, say. 16 I-frames, whereas the next biggest track ha.s only nine I-franies A decision to choose a buffer large enough for the second biggest case might be wiser. Making this choice means truncating the biggest track, thus denying some streams one frame in the movie. To avoid a glitch, the previous frame can bt; redisplayed. No one will notice this. Taking

this approach further, if the third biggest track has only four I-frames, using a buffer capable of holding four l-fra,nes and 20 P-frames is worth it Intro- Introducing two repeated frames for some streams twice in the movie- is probably acceptable. Where does this end? Probably w,th a buffer size that is big enough for 99% of the lrames. Clearly, there is a trade-off here between memory used for buffers and quality of the movies shown. Note that the more simultaneous streams there are, the better the statistics arc and the more uniform the frame sets will be. 7.6.4 Placing Multiple Files on a Single Disk So far we have looked only at the placement of a single movie. On a video server, there will be many movies, of course. If they are strewn randomly around the disk, time will be wasted moving the disk head from movie to movie when multiple movies are being viewed simultaneously by different customers.

488 MULTIMEDIA OPERATING SYSTEMS CHAP 7 This situation can bis improved by observing that some movies are more popu- popular than others and taking popularity into account when placing movies on the disk. Although little can be said about the popularity of particular movies in gen- general (other than noting that having big-name stars seems to help), something can be said about the relative popularity of mov,es ,n general. For many kinds of popularity contests, such as movies being rented, books being checked out of a library. Web pages bt--ing referenced, even English words being used in a novel or the population of ihe largest cities, ;i reasonable approxi- approximation of the relative popularity follows a surprisingly predictable pattern. This pattern was discovered by a Harvard professor of linguistics, George Zipf A902-1950) and is now called Ziprs law. What it states is that if the movies, book;,, Web pages, or words are ranked on their popularity, the probability that the next customer will choose the item ranked k-th in the list is C/k, where C is a nor- normalization constant. Thus the fraction of hits for the top three movies are C/|, G2. and C73. respectively, where C is computed such that the sum of ail the terms is I. In other words, it there are N movies, then C/l + C/2 + C/3 + C/4+ ■ + C/N ■= I From this equation, C can be calculated. The values of C for populations with |0. 100, 1000, and |0,000 items are 0.341,0.193. 0 134. and 0 [02, respectively. For example, for 1000 movies, the probabilities for the- top five movies are 0.134. 0.067, 0.045, 0.034. and 0 027, respectively Zipf s law is illustrated in Fig. 7-20 Just tor fun. it has been applied to the populations of the 20 largest U.S. cities Zipt\ law predicts that

the second larg- largest city should have a population half of the largest city and the third largest city should be one third ot the largest city, and so on. While hardly perfect, it is a surprisingly good fit. For movies on a video server, Zipf s |aw states that the most popular movte is chosen twice as often as the second most popular movie, three times as oiten as the third most popular movie, and so on. Despite the fact th;it the distribution falls off fairly quickly at the beginning, it has a long tail. For example, movie 50 has a popularity of 0/50 and movie 5] has a popularity of £751, so movie 51 is 50/5 [as popular as movie 50, only about a 2% difference. As one goes out further on the tail, the percent difference between consecutive movies becomes less and less. One conclusion is that the server needs a lot of movies since there is substantial demand for movies outside the top 10 Knowing the relative popularities of the ditterent movies makes il possible to model the performance of a video server and to use that information foi placing files. Studies have shown that the best strategy in surprisingly simple and distn- bution independent It is called the organ-pipe algorithm (Grossman and Silver- man, 1973; and Wong, 1983). It consists of placing the most popular movie in the middle of the disk, with the second and third most popular movies on either side

F[LE PLACEMENT § I 0 250 0 200 0 150 0 100 0 050 0 _ \ \ - \ - \ i i i i i i i i i i i i T T T T T—T—t—i—T- Figure 7-20. TTic curve gives Zipfs law |< is I. Los Angeles is 2, Chicago is 3, etc) of it. Outside of these come numbers four and five, and so on, as shown in Fig. 7-21. This placement works best it each movie is a contiguous file of the type shown in Fig. 7-17, but can also be used to some extent it'each movie is ton- strained to a narrow range of cylinders. The name of the algorithm conies from the fact that a histogram of the probabilities looks like a slightly lopsided organ. □ D Dd Mov,e } Movie 10 I 8 Movie Movie I Movie 4 | 2 1 Movie Movie Movie Movie 9 Movie Cylinder — Figure 7-21. The organ-pipe dij-tribiilio What this algorithm does is try to keep the disk head in the middle of the disk. With 1000 movies and a Zipfs law distribution, the top five movies, represent a

 total probability of 0 307. which means thai the disk head will stay in the cylinders allocated io the top five movies, about 30% of tin.- time, a suiprismgly large amount if 1000 movies are available. 7.6.5 Placing Files on Multiple Disks To gei higher performance, video servtis often have many disks that can run in paiallel.

Sometimes RAIDs are used, but often noi because what RAIDs offer is higher reliability at the cosl ot performance Video servers generally want high performance and do not care so much about correcting transient errors Also RAID controllers can become a bottleneck if they have too many disks to handle ;it once A more common configuration is simply a large number ot disks sometime*, referred to as a disk farm. The disks do not roiate in a synchronized way and do not contain any parity bits, as RAIDS do. One possible configuration is to put movie A on disk 1, movie S on disk 2. and so on, as shown in Fig- 7-22(a). !n practice, with modem dir.ks, several movies can be placed on each disk AO Al A2 A3 A4 A5 A6 UtJ BO B1 B2 B3 B4 B5 B6 CO C1 C2 C3 C5 C6 DO Dl D2 D3 D4 D5 D6 A0 B0 B4 CO C4 DO Lei- A1 A5 B1 B5 C1 C5 Dl LsL « 02 B6 02 C6 D2 Ipej A3 B3 B7 C3 C7 D3 AO A4 B3 B7 C2 C6 D1 LdsJ GD A1 A5 BO B4 C3 C7 2O B1 CO C4 D3 2^ A3 A? B2 B6 C1 C5 DO ■^54- lUllimedia Hies over mulnplc disks (jr all filev (c) Slaggered sinp,np < Figure 7-22. Four u-Jy. ul organi/i No slriping (h) Same striping pjtd Random striping This organization i.s simple to implement and has straightforward failure char- characteristics: if one disk fails, all the movies on it become unavailable. Note that a company losing a disk full of movies is not nearly as bad as a company losing a

SEC 7.(i FILE PLACEMENT 491 duk full of data because the movies can easily be reloaded on a spate disk from a DVD. A disadvantage of this approach is that the load maj not be well balanced. It some disks, hold movies that are cuneptly much in demand and other disks hold less popular movies, ihe system w[|| not be fully utilized. Of course, once the usage frequencies of the movies are known, it may be possible to move some of them to balance the load by hand. A second possible organization is, to stripe each movie over multiple disks, four in the example of lig. 7-22(b). Let us assume for the moment that all frames are the same size (i.e.. uncompressed). A fixed number of bytes from movie A is written to disk I. then the same number of bytes is written to disk 2, and so on until (he last disk is reached (in this case with unit ^3). Then the siriping contin- continues at the first disk agam with A4 and so on until the entire file has been written. At that point movies S. C. and D are striped using the same pattern. A possible disadvantage of this striping pattern is that because a|l movies start on the first disk, the load across the disks ma) not be balanced. One wa> to Spread the load better i*i to stagger the starling disks, as shown in Fig. 7-22(c). Yet another way to attempt to balance the load

ib to use d random striping patiern for each file, as shown in Fig. 7-22(d). So far we have assumed that a|| frames are the- same Mze. With MPECi-2 movies, this assumption is false- I-frames are much larger than P-frames. There are two ways of dealing with this, complication, stripe by frame or stripe by block. When striping by frame, the first frame ot movie A goes on disk I as a contiguous unit, independent of how big it is. The next frame goes on disk 2. and so on. Movie S is striped in a similar way, either starting at the same disk, the next disk (if .staggered), or a random disk. Since frames are read ont at a lime, this form of striping does not speed up the reading of any given movie However, ii spreads the load over the disks much belter than in Fig. 7-22(a), which may behave badly if many people decide to watch movie A tonight and nobody wants movie C On the whole, spreading the load over all Ihe disks makes better use of the total disk bandwidth, and thus increases the number of customers that can be served The other way of striping is by block For each movie, fixed-size units are written on each of the disks in succession (or at random). Each block contains one or more frames or fragments thereof The system can now issue requests for multiple blocks at once tor the same movie. Each rc-quest asks to lead data into a different memory buffer, but in such a way that when a|l requests ha\e been com- completed, a contiguous chunk of the movte (containing many frames) is now assem- assembled in memory contiguously. These request can proceed m parallel. When the last request has been satisfied, the- requesting process tan be signaled that the work has been completed. It can then begin transmitting the data to the user. A number Of frames later, when the buffer is down to the last few frames, more requests arc issued to pieload another buffer. This approach u.ses large .(mounts of memory for buffering in c(rder to keep the disks busy. On a system with 1000 active users and l-MB buffers'(for example, using 256-KB blocks on each of four

 disks). 1 GB of RAM is needed for (he buffers. Such an amount is small potatoes on a |OOO-user server and should not be a problem. One final issue concerning striping is how many disks to stripe ever At one h dk F l h (B moies triped f h _-(iB movi extre, p and 1000 disks, a block of 2 MB could be written on each disk so that no movie uses the same disk twice. At the other exticme. the disks are partitioned into small groups (as in Fig. 7-22) and each movie ts restticted to a single partition. The former, called wide striping, does a good job of balancing the load over the disks Its main problem is that it every inoue uses every disk

and one disk goes down, no movie can be shown- The latter, called narrow striping, may sufter from hot spots (popular partitions), but loss of one disk only rums the movies in its partition. Striping of variable-sized frames is analyzed m detail mathemati- mathematically in (Shenoy and Vin, 1999) 7.7 CACHING Traditional LRU file caching does not work well with multimedia lilcs, because the access patterns for movies arc different from those of text tiles. The idea behind traditional LRU buffer caches is that alter a block is used, it should be kept in Ihe cache in case it is needed again quickly For example, when editing a file, the set of blocks on which the file is written tend to be used over and over until the edit session is finished In other woids, when there is idatively high pro- probability that a block will be reused within a short interval, it is worth keeping around to eliminate a future disk access. With multimedia, the usual access pattern is that a movie is viewed trom be- beginning to end sequentially. A block is unlikely to be used a second time unless ing techniques do not work However, caching can still help, but only if ust>d dif- differently. In the following sections we will look at caching tor multimedia 7.7.1 Block Caching Although just keeping a block around in the hope thai it may be reused quickly is pointless, the predictability of multimedia systems can be exploited to make caching useful again. Suppose that two uscis are watching the t,arne movie, with one of them having started 2 sec after the other. Aiter the first user has fetched and viewed an> given block, it is very likely that the second user will need the same block 2 sec later The system can easily keep track ot which together in time Thus whenever a block is read on behalt ot a movie that will be needed again shortly, it may make sense to cache it, depending on how long it has to he cached

SEC 7 7 CACHING 493 and how tight memory is. Instead ot keeping all d.sk blocks in the cache and du.- carding the least recently used one when the cache fills up. a different strategy shuuld be used. Every movie that has a second viewer within some time if of the first Viewer can be marked as cachable and ail its blocks cached until the second (and possibly third) viewer has used them Foi othei movies, no caching is done at all This idej can be taken a step further In some cascs it may be feasible to meise two streams. Suppose that two users are matching ihe same movie but with a |0~-sec delay between them. Holding the hlocks in the cache for 10 sec is possi- possible but wastes memory An alternative, but slightly sneaky, approach is to try to gei the two movies m sync This can be done by changing the frame rate for both movies.

This idea is illustrated in Fig. 7-23. 1 0 Ser2 0 1 0 8 n u 3 0 6 0 5 0 0 1 n u 2 0 0 2 0 { e 0 o 3 6 0 3 C 5 4 0 b 0 0 2 0 0 7 0 Merging ihu lw,i Mruairu. '"lo one. In Fig 7-23(a), both movies run at the standard NTSC rule of |800 frames/min. Since usei 2 started 10 sec later, he continues to he 10 sec beyond for the entire movie In Fig. 7-23(b), however, user \\ stream is slowed down when user 2 shows up. Inslead of running |800 frames/mm, for the next 3 mjn, it

494 MULTIMEDIA OPERATING SYSTEMS CHAP. 7 runs at 1750 frames/mm. After 3 minutes, il is at trame 5550. In addition, user 2*s stream is played at 1850 Irames/min for the first ? min. also putting il at frame 5550. From that point on. both play ai normal speed. During the catch-up period, user I's stream is running 2.8% slow and user 2's stream is running 2.8% fast. It is unlikely that the users will notice this How- However, it that is a concern, the <.atch-up period can be spread out over a longer inter- interval than 1 minutes An alternative way to slow down a user to merge with another stream is to give users the option ot having commercials in their movies, presumably for a lower viewing price than commercial-free movies. The usei can also choose the product categories. s;o the commercials will be less intrusive and m^re likely to be watched By manipulating the uumbei, length, and tuning of the commercials, the stream can be held back long enough to gut in sync with the desired stream (Knshnan, [999). 7.7.2 File Caching Caching can also be useful in multimedia systems in h different way. Due to the large size of most movies B GB). video servers often cannot store all their movies on disk, so they keep them on DVD or tape. When a movie is needed, it can always be copied to disk, but there is .t substantial startup time to locate the movie and copy it lo disk. Consequently, most video servers maintain a disk cache of the most heavily requested movies The popular movies are stored in their entirety on disk. Another way to use caching is to kee-p the first lew minutes of each movie on disk That way, when a movie is requested, playback can start immediately from the disk file. Meanwhile, the movie is copied from DVD or tape to disk By stor- storing enough of the movie on disk all the time, it is possible to have a very high pro- probability that the next piece of the movie has been feiched before it is needed. If all goes well, the entire movie- will be on disk well before it is needed. It will then go in the cache and stay on disk in case there are more requests later. If too. much time goes by without another request, the movie will be removed irom the cache to make room for a more popular one. 7.8 DISK SCHEDULING FOR MULTIMEDIA Mullimedia puts

different demands on the disk.s than traditional text-oriented applications such as compilers or word processors [,i particular, multimedia demands an extremely high data rale and iea|-time delivery of the data. Neither of these is trivial to provide. Furthermore, in the case of *i video sei~ver there is economic pressure to have a single server handle thousands of clients simultane- simultaneously. These requirements impact the entire system. Above we looked at the file system. Now let us look at disk scheduling for multimedia.

DISK SCHEDULING K)K MULTIMEDIA 7.8.1 Static Disk Scheduling ; and data-rate demands on all nake* it easier to handle than .1 Although multimedia puts enormous reaRir parts ot the system, il also has one property that traditional system; predictability. In a traditional operating system, requests ire made lor disk blocks m a fairly unpredictable way. The best the disk sub wem can do is, perform a one-block read ahead for each open file. Other than that, all it can do is wait for requests to come in and process them on demand. Multimedia is different. Each acti\e stream puts, a well-defined load on the system thai is highly predictable. For NTSC playback, every 33.3 msec, each client wants the next trarne m its file and the system has 33.3 msec to provide all the frames (the system needs to buffer at least one frame per stream so that the fetching of frame k + 1 can proceed m parallel with the playback of frame k). This predictable load can be used to schedule the disk using algorithms tailored to multimedia operation Below we will consider just one disk, but the idea can be applied to multiple disks as well- For this example we will assume that there are |0 users, each one viewing a different movie. Furthermore, we will assume that all movies have the same resolution, frame rate, and other properties. Depending on the rest of the system, the computer may have |0 processes, one per video stream, or one process with |0 threads, or even one process with one thread that handles the |0 streams in round-robin fashion. The details are not important. What is important, is that time is divided up into rounds, where a round is the frame time C3.3 msec for NTSC, 40 msec for PAL). At the start of each round one disk requesi is generated on behalf ot each user, as shown in Fig. 7-24. Stream lor odd frames - for even frames - Block requesled •foi T J 701 1 1 92 ! 1 2B1 I ! 130 Oph f I 336 w La] T T 410 16O n algonlhm 3 1 466 f \ 204 f 1 524 160 204 281 326 410 466 524 r in which disk requests are processed *■ e round, each mov,e ask, for one frame. After all the requests have come m at the start of the round, the disk

knows what it has to do during that round. It also knows that no other requests will coine in until these have been processed and the next round has begun. Consequently, it

tin ro de Stl H erag< ,hcd 5. ll ies und can . If, lines ■ad 11 r avini a se froi ics ; s at ! tw . which n other the sm ilso be i iver has n distan p are rare n once. N ro Or m e 496 MULTIMEDIA OPERATING SYSTEMS CHAP 7 can sort the requests in the optimal way, probably in cylinder order (although con- conceivably in sector order in some casesl and then process them in the optimal order In Fig 7-24. the requests are shown sorted in cylinder order. At first glance, one might think that optimizing the disk m tins way has no value because as long a the di -k meets the deadline it does not matter if it meets it with 1 msec to sp e n p H h n n B optimizing socks n p oc h q h knhnn nP nonh nngd khn hnn n n p h n h n n ne q h n o n h k h i k n d d B g i nh 1n ohnnj,n i h n h f m b h scheduling To keep the flow of data out to the clients moving smoothly, double buffering is needed in the server During round I. one set ot buffers is used, oiie buffer pci stream When the round is finished, the output process or processes are un- unblocked and told to transmit frame I- At the same time, new requests come in foi frame 2 of each movie (there might be a disk thread and an output thread k>r each movie) These requests must be satisfied using a second set of buffers, as the first ones are still busy. When round 3 starts, the first set of buffers are now free and can be reused to fetch frame "h We have assumed that there is one uiuiid per fr,,me This limitation is not strictly necessary. There could be two rounds pei frame to reduce the amount of buffer space required, at the eost of twice as many disk operations Similarly, two frames could be fetched from the disk per round (assuming pairs oi frames .ire stored contiguously on the disk). This design cuts the number ot disk operations in half, at the cost of doubling the amount of buffer space required Depending on the relative availability, performance, and cost of memory versus disk I/O. the optimum strategy can be calculated and used.

7.8.2 Dynamic Disk Scheduling In the example above, we made the assumption that all sireams hj\e the saine resolution, frame rate, and other property. Now let us drop this assumption. Different movies may now have different data rates so it is not possible tii have one round every 33.3 msec and fetch one frame tor each stream Requests come in to th(; disk more or less at random Each read request specifies which block is to be read and in addition ai whal tune the block is needed, that is, the deadline. For simplicity, we wt‖ assume that

SEC 7.8 DISK SCHEDULING FOR MULTIMEDIA 497 the actual service time tor each request is the same (even tnougn tnis is ceridinij not true). In th,s way we can subtract the fixed service time from each request to get the latest time (he request can be initiated and still meet the deadline. This makes the model simpler because what the disk scheduler cares about is the dead- deadline lor scheduling the request. When the system starts up, there are no disk requests pending When the tirst request comes in, it is serviced immediately While the first seek is taking place, other requests may conie m, so when the first request is finished, the disk driver may have a choice of which request to process nexi Some request is chosen and started When that request is finished, there is again a set of possible requests, those that were not chosen the first time and the new arrivals thai came in while the second request was being processed In general, whenever a disk request ;i choice. The question is: "What algoiithm does it Use to select the next request to service?" Two factors play a ro|e in selecting the next disk request: deadlines and cylinders. From a performance point ot view, keeping the requests sorted on cylinder and using the elevator algorithm minimises total seek time, hut may cause requests on outlying cylinders 10 miss their deadline. Hrom a real-time point of view, sorting the requests on deadline and processing theiii in deadline order, earliest deadline first, in immizes the cnance or missing dcadlmes, bt.it increases total seek time These factors can be combined using thu stan-EDF algorithm (Reddy and Wyllie, 1992). The basic idea of this algorithm is to collect requests whose dead- deadlines are relatively close together into batches and process these in cylinder order. As an example, consider the situation of Fig 7-25 at i =70A The disk driver knows it has 11 requests pending for various deadlines and various cylinders It batch, sort them on cylinder number, and use the elevator algorithm to service these in cylinder order. The order would then be | |(). 330, 440, 676, and (S80. As long as every request is completed before its deadline, the requests can be sate|> rearranged to minimize the total seek time required. When different streams have different data rates, a serious ts^ue arises when a new customer shows up: should the customer be admitied. If admission of the customer Will cause other streams to miss their deadlines frequently, the answer is probably no. There are two ways to calculale whether to admit the new customer or not One way is to assume that each customer needs n certain amount of resources on the average, for example, disk bandwidth, memory buffers, CPU time. etc. If there is enough of each left tor an average customer, the new one is admitted. The

other algorithm is more detailed. It takes a look at the specific movje the new cusiomer wants and looks up the (precompuied) data rate for that movie, which differs for black and white versus color, cartoons versus filmed, and even

MULTIMEDIA OPERATING SYSTEMS Figure 7-2S. The sian-bDF algor - iheduling. ne, and cylinder niimher^ ith lo love stories versus war film'; Love glories move sloy slow cross dissolves, all of which compress well whereas war films have many rapid cuts, and fast action, hence many 1-frsmes and large P-trames. If the server has enough capacity for the specific film the new customer wants, then admission is granted; otherwise it is denied. 7.9 RESEARCH ON MULTIMEDIA Multimedia is a hot topic these days, so there is a considerable amount of research about it- Much of this research is about ihe content, construction tools, and applications, all of which are beyond Ihe scope of this book. However, some of it involves Operating system structure, either writing a new multimedia operat- operating system (Brandwein et a|.. 1994). or adding multimedia support to an existing operating system (Mercer, 1994). A related area is the design of multimedia server. (Bemhardt and Biersack, 1996; Heybey et a| , 1996; l.ougher ei al.. 1994; and Wong and Lee, [997). Some papers on multimedia are not about complete new systems, but about algorithms useful in multimedia systems. A popular topic has been real-time CPU scheduling for multimedia (Baker-Harvey. 1999; Bolosky et al.. 1997; Dan et al, 1994; Goyal et al., 1996; Jones et al., 1997; Nieh and Lam. [997; and Wu and Shu, 1996). Another topic that has been examined is disk scheduling for mul- multimedia (Lee et a|., 1997; Rompogiannakis et a|., 1998; and Wang et al.. 1999). File placement and load management on video servers are also important (Gafsi and Biersack, 1999; Shenoy and Vm, 1999; Shenoy et a!.. 1999; and Venkata- s,ubramanian and Ramanathan, [997) as is, merging video streams to reduce bandwidrh requirements (Eager et al.. |999). In the text, we discussed how movie popularity affects placement on ihe video server. This topic is an ongoing area of research (Bisdikian and Patel. 1995; and

SEC 7.9 RESEARCH ON MULTIMEDIA 4W Gnwod/ et a|.. 1997). Finally, security and privacy m multimedia (e.g., in video- videoconferencing) are also subjects of rt-search interest (Adams and SassC 1999; and Honeymanet. a|, 1998) 7.10 SUMMARY Multimedia is an up-and-coming Use of computeis. Due to the large sizes of multimedia files and their stringent iea|-time playback

requirements, operating systems designed tor text are not optimal lor multimedia. Multimedia files con- consist ot multiple, parallel tracks, usually one video and at least one audio and some- sometimes subtitle tracks as well. The^e must all be synchronized during playback Audio is recorded by sampling the volume periodically, Usualh 44,100 times/sec (for CD quality sound). Compression can be applied to the audio signal, giving ii uniform compression rate of about lOx. Video compression uses both intraframe compression (JPEG) and jntertramc compression (MPEG). The latter represents P-frames as differences from the previous frame. B-frames can he based either on the previous frame or the next frame Multimedia needs real-time scheduling m order to meet its deadlines. Two algorithms are commonly used. The first is rale monotonic scheduling, which is a static preemptive algorithm thai assigns fixed priorities to processes ba.sed on their periods. The second is earliest deadline first, which is a dynamic algorithm that always chooses the process with the closest deadline. EDF is more complicated, but it can achieve 100?! utilization, something that RMS cannot achieve. Multimedia file systems usually use a push model rather than a pull model. Once a stream is started, the bits come off the disk without further user requests. This approach is radically different from conventional operating systems, but is needed to meet the real-time requirements Files can be stored contiguously or noi. In t|ie latter case, the unit can be vari- variable length (one block is one frame) or fixed length (one block is many frames). These approaches have different trade-offs. File placement on the disk affects performance When there aie multiple files, the organ-pipe algorithm is sometimes used Striping tiles across multiple difcks, either wide or narrow, is common. Block and file caching strategies aic also widely employed to improve perform a ncu. nic off an UliraWide SCSI disk'

500 MULTIMEDIA OPERATING SYSTEMS CHAP. 7 2. Can iincompressed black-and-white NTSC television be senl o\er fast Eiliernet so, how many channeU Jl once'1 3. HDTV has iw.cc the horizontal resolution ol regular TV A280 versus 640 pixels). Using information provided in the jexi, huw much more bandwidth docs tt require than siandardTV1 4. In Tig 7-3. (here are separate flics ior fasl Jorward .md last reverse. If a video server is intended to support slow monon as well, is anothei hie required for slow motion in the forward direction' What about in [he backward direction" 5. A Compact Disc holds 74 mm of music or 6^0 MB of data. Make- an cstnnaie tif the 6. A sound

sigiial is sampled using a signed 16-bit number (I sign hit. 15 magnitude hits) Whji is the maximum quantization noise m percent? Is this a bigger problem 7. A recording sludio is able to make a master digital recording using 20-bit sampling The final distnbunon In listeners will use Id bits. Suggest a way [o reduce the efleet nf quantization noise, and discuss advantages and disadvantages ofyuur schema 8. NTSC and PAL both use a 6-MHz broadcast channel, yei NTSC has 30 frames/sec whereas PAL has only 25 frames/set How is ihis possible" Dues this mean ihai ll both syslems were lo use the same color encoding scheme, NTSC would have inherently better quality than PAL1' Explain your answer 9. The DCT transformation uses an 8x8 block. vt?i the iilgsinthni used for motion com- compensation uses 16 X 16 Does |hts dilference tause problems, and if si>- how are |hey solved in MPEG' 10. In Fig 7-10 we saw how MPEG works with a stationary background and a moving actor- Suppose ihat an MPEG video is made front a scene in which the camera is mounted on a inpod and pans sluwing from left to right at a speed such that no |wo consecutive frames are ihe same Do all ihe frames have to be]-frame<. now ' Why or why iuh? 11. Suppose lhai each of the three processes m Fig 7-11 is accompanied bv a process [hai supports an audio stream running with the same period as Ms video process so audio buffers can be updated beiween Video frames All ihree of these audio pnicesses aie identical. How much CPU time is available for each burst of an audio process > for |() msec. The second one runs every 40 msec for li msec Will RMS always work for them'1 13. The CPU of a video server has a uiihzanon ol (>5ri How many movies can it show using RMS scheduling? the CPU busy indefinitely because iherc is milv "75-msec work per second for it to di> so Extend (lie figure buyond 150 msec and determine when the CPU first goes idle wiih RDF

CHAP 7 PROBLEMS 501 15. A DVD can hold enough data ioi a full-lengih movie and the lianster rale is adequate to display a television-quality prog.am Why not jusl use a "farm- of many DVD dnvcs as. the data source for a video server'1 tarn City are nor »illing to wail more ihjn 6 mmuies lor a movie to sum. How many parallel streams do they need for a ^*hour movie11 17. Consider a system a^ing ihe scheme of \bram-Proleta and Shin in which the video server operator wishes eustoniers to be ;)ble to search forward or backward for I min enliiely locally Assuming the video stream is MPEG-2 at 4 Mbps- how much bulter ,\ I-KB disk block. If the video resolution is 1280x720 and (he daia siream is 12 Mbps. how much disk space is waslcd on miemal

fragmentation in a 2-ht>ur movie using NTSC > given disk block and movie si £C, does ojie of them suffer moie internal fragmentation 20. Consider the too alternative!, shown in Pig 7-18. Does the sbilt toward HDTV favor either of these systems over the other? Discuss 21. The near video-on-demand scheme nf Chen and Thapar works besl when each frame set is the same size. Suppose ihat a movie is being shown m 24 simultaneous streams, and ihai one frame in |0 is an l-frdnie Also assume thai I-frames are 10 times larger buffer equal io 4 I-fra,ries and 20 P-frainCs w,ll not be big enough* Do yau think ihal types are randomly and independently disiribu^d over ihe streams 22. The end resuli ot Fig. 7-16 is ihai |he play pomi ,s not in the middle of the buffer .my more Devise a scheme to have al least 5 nun behind [he play point and 5 min ahead of it. Make any reasonable assumptions you have to, but state them explicitly 23. The design of Fig 7-17 requires that all language tracks be read on each frame Sup- Suppose ihai the designers of a video server have to support a large nuiiiber <:f languages, bui do not wani m devote so much RAM io buffers to hold each frame What older alternatives are available, and what are the advaniagcs and disadvantages of each one" 24. A small video server has eight movies. What does 7ipf s lav, prc-dict as the probabili- probabilities for |he moil popular movie, second mosr popular movie and so on down to ihe leasl popular ttiovic1 25. A I4-GB disk with 10(XI cylinders ,s used U> hold 1000 30-sec MPEG-2 v.deo clips running a[4 Mbps. Thi'y are siorcd according to |he organ-pipe algorithm Assuming Zinfs law, what fraction ot Ihe iime viiil ihe disk arm spend in Ihe middle 10 cylinders?

502 MULTIMEDIA OPERATING SYSTRMS CHAP 7 26. Assuming ih.il iho relative demand lor dims A, B. C, and D is described by Zipf s law, what is the txpeued relative utilisation oi the lour disks in Fit: 7-22 for (he tour striping methods, shown? 27. Two vrdeo-on-demand customers started watching the same PAL movie 6 sci. apart If the system speeds up <>nc stream and slows down the other to get ihem to merge, wha/ percent speed up/down is needed to nK*rge ihem m 3 mm.' 28. An MPEG-2 video server uses the round i-chem.- of F-Jg. 7-24 lor NTSC video. All Die videos coinu (ifl a smgie 10,800 ipin Ultr;iWide SCSI disk with an average seek imic ol 1 msec Ho* many sireami can be supporicd'1 29. Repeal the previous problem, bul now assume thai scan-EDF reduce-- [he average sei'k time by 20% How many streams can now bt> supported.' 30. Rtpciit the previous problem once more, but now assume that each Iramu is slupc'd can now be suppurled. 31. The iexi

describes usmg a natch ol tive daia rcctuests 1'^ schedule ihe situation described in Fig. 7-25U) It all rct|uesrs lake an equal amount of" lime, what is the maximum time per reqiics* allowable in]h]S example * 32. Many ol ihe bitmap (mages ituu are supplied lor yenerdimg coinpiilcr ' wallpuper" use lew colors and are easily compressed. A simple compression scheme i\ ihe following, choose a data value that does, noi appear in Ihe mpul file, und use it as :i Hag Read rhe tile, byie by byte, looking lor repeated h>it values Copy singlo values and byits repeated up to three times directly lo the output file When a repeated string ot 4 ot more bytes is found, write to ihe output file a string of three bytes consisting ol the slag byte, a byte indicating a count from 4 to 2^5- atid the actual value lound m the input file Write a compression program using this algorithm, and a decompression program that can resioie the original file txtr,i credit how c.iii you deal witli files (hat contain the flag byic in their data? jjt Compmer animation is accomplished hy GispJavit^g a sequence ol slightly dillercni images. Write a program to calculate I he byte by byte difference bclween two uncompressed bitmap images of the same dimensions The output im" be |hc sairic size as the inpul tiles, of course Use this difierence file a^ mpui io the Lompressmi) program of the previous problem, and compare the ellccttvencss ol ihis approach with compression of individual images

8 MULTIPLE PROCESSOR SYSTEMS Since its inception, ihe computer industry has heen driven by an endless quest for more and more computing power. The ENIAC could perform 300 operations per second, easily |000 times faster than any calculaior before it. yei people were not satisfied. We now have machines a million times faster than the ENIAC and still there is a demand for yet more horsepower Astronomers are trying to make sense of the universe, biologists are trying to understand the implications of the human genome, and aeronautical engineers are interested m building safer and more efficient aireraft, and afl want more CPU cycles. However much computing power there is, it is never enough. in the past, the solution was always (o make the clock run faster. Unfor- Unfortunately, we are beginning to hit some fundamental limits, on clock speed. According to Einstein's special theory of relativity, no electrical signal can pro- propagate faster than the speed of light, which is; about 30 cm/nsec in vacuum and about 20 cm/nsec in copper wire or optical fiber. This means* that in a compuler with a 10-GHz clock, the signals cannot travel more than 2 cm in total Fot a lOO-GHz computer the total path

lengih is at most 2 mm. A |-THz A000 GHz) computer will have to be smaller than 100 microns, just to let the signal get from one end to the other and back once with a single clock cycle. Making computers this small may be possible, but then we nil another funda- fundamental problem: heat dissipation. The faster runs the computer, the more heat it generates, and the smaller the computer, the harder it is to get rid of this heat.

504 MULTIPLE PROCESSOR SYSTEMS CHAP 8 Already Ori high-end Pentium systems, the CPU cooler is b.gger than the CPU itself All in all, going from I MH? to 1 GH? smiply required incrementally bettei engineering of ihe chip mairaiactunng process Going from i GH^ ti> 1 THz is, going to ,equire a radically different approach One Approach to greater speed is through massively parallel computers. These machines consist ot many CPUs, each of which rims at "normal"' spcuti (whatever that may mean in a given year), but which collectively ha\'e far more computing power than a single CPU. S>items with 1000 CPUs are now commer- commercially available. System-, with I million CPUs are likely to be built in the coming decade. While there arc other potential approaches to greater speed, such as bio- biological computers, in this chapter we will focus on systems with multiple conven- conventional CPUs Highly parallel computers are often used for heavy numbei crunching Prob- Problems such as predicting the weather, modeling airflow around an aircraft wing, simulating the world economy, or understanding drug-receptor interactions in the bram are all computationally intensive Their solutions require long runs on many CPUs at once The multiple processor systems discussed in this chapter are widely-used for these and similar problems in science and engineering, among other areas. Another relevant development is the incredibly rapid growth ot the Internet. It was, originally designed as a prototype for a fault-lolcrant military control sys- system, then became popular among academic computL'i scientists, anil has recently acquired many new uses One ol' these is linking up thousands O| computers a|l over the world to work together on large scientific problems. In a sense, a system consisting of |000 computers spread all over the world is no different than one consisting of 1000 computers in a single room, although the delay and other technical characteristics are different. We will also consider these systems in this chapter. Putting | million unrelated computers m u room is easy to do provided that you have enough money and a sufficiently Urge room. Spreading I million unre- unrelated computers around the world is even easier since it finesses the

second prob- problem. The trouble comes in when you want them to communicate wiih onu another lo work together on a single problem. As a consequence, a great deal of work has been done on the interconnection technology, .ind different interconnect technolo- technologies, have led to qualitatively different kinds ot systems and different software organizations. All communication between electronic <or optical) components ultimately c(imes down to sending messages—well-defined bit strings—between (hem The differences ypp ^ri the time scale, distance scale^ and logical orsanization involved At one extreme are the shared-memory multiprocessors, systems in which some- somewhere between two and about 1000 CPUs communicate via a shared memory In this model, every CPU has equal access to the entire physical memory, and can read and write individual words using LOAD and STORE instructions. Accessing a

505 memory word usually takes 10-50 nsec. While this model, illustrated in Fig. 8- l(a>. sounds simple, actually implementing it is far from simple and usually involves considerable message passing under the covers, as we will explain shortly. CMJ c c iaj (b) (c) Figure 8-1- (a) A s,hared-memory multiprocessor, fb) A message-passing mul- multicomputer, (c) A wide area distributed system. Next comes the system of Fig. 8-l(b> in which a number of CPU-memory pairs are connected by some kind of high-speed interconnect. This kind of system is called a message-passing multicomputer. Each memory is local to a single CPU and can be accessed only by that CPU. The machines communicate by sending multiword messages over the interconnect. With a good interconnect, a short message can be sent in 10-50 (isec, but still far longer than the memory access time of Fig. 8-l(a). There is no shared global memory in this design. Mul- ticomputers (i.e., message-passing systems) are much easier to build than (shared-memory) multiprocessors but they are harder to program. Thus each genre has its fans. The third model, illustrated in Fig. 8 ! <c), connects complete computer sys- systems over a wide area network, such as the Internet, to form a distributed sys- system. Each of these has its own memory, of course, and the systems communicate by message passing. The only real difference between Fig. 8-He) and Fig. 8-l(fc) is that in the former, complete computers are used and message times are often 10-50 msec. This long delay forces these loosely-coupled systems to be used in different ways than the tightly-coupled systems of Fig. 8-l(b). The three types of systems differ in their delays by something like three orders of magnitude. That is the

difference between a day and three years. This chapter has three major sections, corresponding to the three models of Fig. 8-1. In each one. we start out with a brief introduction to the relevant hard- hardware. Then we move on to the software, especially the operating system issues for that type of system. As we wUl see, in each case different issues are present.

506 MULTIPLE PROCESSOR SYSTEMS CH*P 8 8.1 MULTIPROCESSORS A shared-memory multiprocessor (or just multiprocessor henceforth) is a computer system in which two or more CPUs share full access to a common RAM. A program running on any of the CPUs sees a normal (usually paged) vir- virtual address space. The only unusual property this system has is that the CPU can write some value into a memory word and then read the word back and gel a dif- different value (becau.se another CPU has changed it)- When organized correctly, this property forms the basis of interprocessor communication: one CPU writes some data into memory and another one reads the daia out. For the mos! part, multiprocessor operating systems are just regular operating systems. They handle system calls, do memory management, provide a file sys- system, and manage I/O devices. Nevertheless, there are some areas in which they have unique features. These include process synchronization, resource manage- management, and scheduling. Below we will first take a brief look at multiprocessor hardware and then move on to these operating systems issues. 8.1.1 Multiprocessor Hardware Although all multiprocessors have the property that every CPU can address a‖ of memory, some multiprocessors have the additional property that every memory word can be read as fast as every other memory word. These machines are called UMA (Uniform Memory Access) multiprocessors. In contrasi, NUMA (Nonun- lfonn Memory Access) multiprocessors do not have this property. Why this dif- difference exists will become clear later. We will first examine UMA multiproces- multiprocessors and then move on lo NUMA multiprocessors. UMA Bus-Based SMP Architectures The simplest multiprocessors are based on a single bus, as illustrated in Fig. 8-2(a). Two or more CPUs and one or more memory modules a|] use the same bus for communication. When a CPU wants to read a memory word, it first checks to see if the bus is busy. If the bus is idle, the CPU puts the address of the word it wants on the bus, asserts a few control signals, and waits until the memory puts the desired word on the bus. If the bus is busy when a CPU wants to read or write memory, the CPU just waits until the bus becomes idle. Herein lies the problem with this design. With two

or three CPUs, contention for the bus will be manageable; with 32 or 64 it will he unbearable. The system will be totally limited by the bandwidth of the bus, and most of the CPUs will be idle most of the time. The solution to this problem is to add a cache to each CPU, as depicted in Fig. 8-2(b). The cache can he inside the CPU chip, next to the CPU chip, on the

MULTIPROCESSORS Itiprocessors (a) Wilhoui caching (b) Witli processor board, or some combination of all three. Since many reacts can now be satisfied out of the local cache, there will be much less bus traffic, and the system can support more CPUs. In genera!, caching is no| done on an individual word basis but on the basis of 32- or 64-byte blocks. When a word is referenced, ils enljre block is fetched into the cache of the CPU touching it Each cache block is marked as being enher read-only (m which case it can be present in multiple caches at the same time), or as read-write (in which case it may not be present in any other caches). If a CPU attempts to write a word that is in one or more remote caches, the bus hardware detects, the write and puts a signal on the bus informing all other caches of Ihe write. If other caches have a "clean" copy, that is, an exact copy of what is in memory, they can just discard their copies and let the writer fetch the cache block from memory before modifying it If some other cache has a '■dirty" (i.e., modified) copy, it must either write it back to memory before the write can proceed or transfer il directly to the writer over the bus. Many cache transfer protocols exist. Yet another possibility is, the design of Fig. 8-2(c). in which each CPU has not only a cache, but also a local, private memory which it accesses over a dedicated (private) bus. To use ihis configuration optimally, ihe compiler should place all the program text, sirings, constants and other read-only data, stack-,, and local variables in the private memories. The shared memory ^ then only used for writ- writable shared variables, in most cases, this careful placemeni will greatly reduce bus traffic, but it does require active cooperation from the compiler. UMA Multiprocessors Using Crossbar Switches Even with the best caching, the use of a single bus limits the size of a UMA multiprocessor to about 16 or 32 CPUs. To go beyond that, a differeni kind ot interconnection network is needed. The simplest circuit for connecting n CPUs to k memories is the crossbar switch, shown in Fig 8-3. Crossbar sw,tches have been used for decades within telephone switching exchanges to connect a group of incoming lines to a set of outgoing lines jn an arbitrary way.

508 MUL'I1PLE PROCESSOR SYSTfcMS CHAP S At each intersection ot a horizontal (incoming) and vertical (outgoing) line is a crosspoint A crosspoint is a small switch that can be electrically opened 01 closed, depending on wliethei ihe horizontal jnd vertical tines are 10 be connected or not' In Fig. }J-3(a) we see three crosspoints closed simultaneously, allowing connections between the (CPU, memory) pairs @01. 000), (|0l, 101), and A 10, 010) at the same time. Many other combinations are also possible |n fact, ihe be safely placed on a chess board n»D- nm- Closed crosspoir swilch Figure 8-3. One of the nicest properties of (he crossbar switch is that it is a nonlocking network, meaning that no CPU is ever denied the connection it needs because some crosspoint or line is, already occupied (assuming the memory module itseit is available). Furthermore, no advance planning is needed Even ii seven arbi- arbitrary connections are already set up, it is always possible to tonneci the remaining CPU to the remaining memory. One ot the worst properties of the crossbar switch is the fact that the numlier of crosspoints grows as n2. With 1000 CPUs and I()(M) memory modules we need a million crosspoints. Such a large crossbar switch is not teasible Ncverihdess, for medium-sized sysiems. a crossbar design is workable.

SFC 8 I MULTIPROCESSORS SUV UMA Multiprocessors Using Multistage Switching Networks A completely different multiprocessor design is based on the humble 2x2 switch shown in Fig. 8-4(a). This switch has two inputs and |wo outputs. Mes- Messages arriving on either input line can be switched to either output line. For our purposes, messages will contain up to four parts, as shown in Fig. 8-4(b). The Module field tells which memory to use. The Address specifies an address within a module The Opcode gives the operation, such as READ or WRITE Finally, the optional Value field may contain an operand, such as a 32-bit word to be written on a WRITE. The .switch inspect-, the Module field and uses it to determine if the message should be sent on X or on Y. i [Address I Opcode | Value , Our 2x2 switches can be arranged in many ways to buiid larger multistage switching networks (Adams et al., 1987. Bhuyan et a|.. [989, and Kumar and Keddy. 1987). One possibility is the no-frills, economy class omega network. illustrated m Fig. 8-5. Here we have connected eight CPUs to eight memories using 12 switches More generally, for n CPUs and n memories we would need login stages, with n/2 switches per stage, for a total of (w/2)Uigifi switches, which is a lot better than n2 crosspomts, especially for large values of n. The wmng pattern of the

omega network is often called the perfect shuffle. since the mixing of the signals at each stage resembles a deck of cards being cut m half and |hen mixed card-tor-c&rd. To see how the omega network works;, sup- pose that CPU Oil wants to read a word from memory module I ID. The CPU sends a READ message to switch |D containing | 10 m the Module field. The switch takes the first (i.e., leftmost) bit of 1 10 and uses it for routing. A 0 routes to the upper output and a I routes to the lower one. Since this bit is .i I, Ihe mes- message is routed via the lower output to 2D. All the second-stage switches, including 2D. use the second bit tor routing. This, too. ts a 1. so the message is now forwarded via (he lower output to .'D. Here the third bit is lesied and found to bt; a 0 Consequently, the message goes out on the upper output and arrives at memory 110, as desired. The path followed by this message is marked in Fig. 8-5 by the tetter u end of the module number are no longer needed. They can be put to good use by recording the incoming line number there, so the reply can find its way back For

MULTIPLE PROCESSOR SYSTEMS Figure 8-5. An omega ^wik'hing ntTui>rk path a, the incoming lines are 0 (upper input to ID). | (lower input ti» 2D). and I (lower input to 3D), respectively. The reply is routed back using 01 I. only read- reading it from right to left this time. At the same time all this is going on. CPtJ 001 wants to write a word to memory module 001. An analogous process happens here, with the message routed via the upper, upper, and lower outputs, respectively, marked by tlie letter b. When it arrives, us Module field reads 001, representing the path it took. Since these two requests do not use nv of the same switches, lines, or memory modules, they can proceed in parallel Now consider what would happen it CPU 000 simultaneously wanted to ac- access memory module 000. Its reque t would come into conflict with CPU 001's request at switch 3A. One ot them would have to wait. Unlike the crossbar switch, the omega network is a blocking network. Not every set of requests can be processed simultaneously Conflicts mn occur over the use of a wire or a switch, as well as between reque t to memory and replies from memory It is clearly desirable to spread the memory reference*, umroimh itross the modules. One common technique is to use the low order bits as the module num ber. Consider, for example, a byte onenttd iddress sp te tor a computer that mostly accesses 32-bit words. Tht 2 low order b,ts will usually be 00 but the. nexi 3 bits will be uniformly distributed B\ using these 3 bits a thL midtile number, consecutively addressed word will

be m Lonsccuti\e modules A mem ory system in which consecutive woids are in difkrent modules t*. said to ht interleaved. Interleaved memories maximize pirillelism because most niemor\ references are to consecutive addre ses ll is also possible to design svutchm- networks that are nonlocking and which offer multiple paths from taeh CPU to each memory module, to spread the traffic better

SEC 8 1 MUfTIPROCESSORS S'1 NUMA Multiprocessors Single-bus UMA multiprocessors are generally limited to no more than a few dozen CPUs and crossbar 01 switched multiprocessors need a lot of (expensive) hardware and are not that much bigger. To get to more than [00 CPUs, some- something has to give. Usually, what gives- is the idea that a!| memory modules have the samt access time. This concession leads to the idea ot NUMA multiproces- multiprocessors. ns mentioned abo%e Like their UMA cousins, they provide a single address space across all the CPUs, but unlike the L'MA machines, access to local memory modules is faster than access to remote ones. Thus a‖ UMA programs will run without change on NUMA machines, but the performance will be worse than on a UMA machine at the same clock speed NUMA machines have three key characteristics that all of them possess and which together distinguish them from other multipiocessors- 1 There is a single address space visible to all CPUs. 2 Access to remote memory is via LOAD and STORE instructions. 3. Access to remote memory is slower than access to local memory When the access time to remote memory is not hidden (bt-cause there is no cach- caching), the system is called NC-NUMA. When coherent cacht-s are present, the sys- system is called CC-NUMA (Cache-Coherent NUMA). The most popular approach for building large CC-NUMA multiprocessors currently is the directory-based multiprocessor. The idea is to maintain a data- database telling where each cache line is- and what its status is. When a cache line js referenced, the database is queried to find out where it is and whether it is clean or dirty (modified). Since this database must be queried on every instruction that references memory, it must be kept in extremely-fast special-purpose hardware that can respond in a fraction of a bus cycle. To make the idea of a directory-based multiprocessor somewhat more con- concrete, let us consider as a simple (hypothetical) example, a 256-node system, each node consisting of one CPU and 16MB of RAM connected to the CPU via a local bus. The total memory is 212 bytes, divided up into 22f" cache lines of 64 bytes each. The memory is statically allocated among the

nodes, with 0-I6M in nodcO. I6M-32M in node I. and so on. The nodes are connected by an interconnection network, as shown in Fig. 8-6(a). Each node also holds |he directory entries for the 2IS 64-byte cache lines comprising its 2:4 byte memory For the moment, we will assume that a line can be held in at most one cache. To see how the directory works, let us trace a LOAD instruction from CPU 20 that references a cached line. First the CPU issuing the instruction presents it to its MMU. which translates it to a physical address, say, 0x24000108. The MMU splits th,s address into the three parts shown m Fig. S-6(b). In decimal, the three

MULTIPLE PROCESSOR SYSTEMS 21B-1 4 3 2 1 D 0 0 1 0 0 dt node 36 82 Division \>f d Figure 8-6. (a) A 256-node d]reclory-hased 35-bil memory address mio fields (c) The direc parts are node 36, line 4, and offset 8. The MMU sees thai the memory word referenced is from node 36. not node 20, so it sends d request message through ihe interconnection network lo the line's home node, 36, asking whether its line 4 is cached, and if so. where. When (he request arrives at node 36 over the interconnection network, it is routed to the directory hardware. The hardware indexes into its table of 2IS entries, one for each of its cache lines and exiract-s entry 4. From Fig. 8-6(c) we see (hat the line is not cached, so the hardware fetches line 4 from the local RAM, sends it back (o node 20, and updates directory entry 4 to indicate ihat the line is now cached at node 20. Now Jet us consider a second request, this time asking about node 36's line 2. From Fig. 8-6(c) we see that this line is cached at node 82. At this point ihc hardware could update directory entry 2 to say that the line is, now at node 20 and then send a message to node 82 instructing it to pass the |me to node 20 and invalidate its cache. Nole that even a so-called "shared-memory multiprocessor" has a lot of message passing going on under the hood. As a quick aside, let us calculate how much memory is being taken up by (he directories. Each node has |6 MB of RAM and 2|S 9-bit entries to keep track of that RAM. Thus the directory overhead is about 9 x 2IS bits divided by |6 MB or

MULTIPROCESSORS 513 about 1.76 percent, which is generally acceptable (although it has to be high- highspeed memory, which increases its cost). Even with 32-byte cache lines the over- overhead would only be 4 percent. With |28-byte cache lines, it would be under I percent. An obvious limitation of this design is. that a line can be cached at only one node To allow lines. Co be

cached at multiple nodes, we would need some way of locating all of them, for example, to invalidate or update them on a write. Various options are possible lo allow caching at several nodes at the same time, but a dis- discussion of these is beyond the scope of this book. 8.1.2 Multiprocessor Operating System Types Let us now turn from multiprocessor hardware to multiprocessor software, in particular, multiprocessor operating systems. Various organizations are possible. Below we will study three of them. Each CPU Has Its Own Operating System The simplest possible way to organize a multiprocessor operating system is to statically divide memory into as many partitions as there are CPUs and give each CPU its own private memory and its own private copy of the operating system. In effect, the n CPUs then operate as n independent computers-- One obvious optimi- optimization is to allow all the CPUs to share the operating sysiem code and make private copies of only the data, as shown in Fig, 8-7- Has OS CPU 2 Has private OS Has private OS Has private OS Memory I/O Figure 8-7. Partitioning multiprocessor ineiwy among lour CPUs, but ^h;inng a single copy of the operating system code The boxes marked Daw jtc ihe operating system's privaie dala for eath CPU This scheme is still better than having n separate computers since it allows all the machines to share a set of disks and other I/O devices, and ii also allows the memory to be shared flexibly. For example, if one day an unusually large pro- program has to be run, one of the CPUs can be allocated an extra large portion of memory for the duration of that program In addition, processes can efficiently communicate with one another by having, say a producer be able to write data into memory and have a consumer fetch it from the place the producer wrote it.

 each CPU Still, from an operating systems' perspectiv operating system is as primitive as it gets. It is worth explicitly menrioning four aspects of this design ihai may not be obvious. First, when a process makes a system call, the system call is caught and handled on its own CPU using the data structures in thai operating system's tables. Second, since each operating system has its own tables, it also has ,ts own set of processes that it schedules by itself. There is no sharing of processes. If a user logsintoCPU I, all of his processes run on CPU 1. As a consequence, it can hap- happen thai CPU 1 is idle wh,le CPU 2 is loaded with work. Third, there is no sharing of page- It c n happen that CPU I has pages to spare while CPU 2 is paging Lontmuoush There is no wa\ for CPU 2 to borrow

some pages from CPU I sinu. the memorv al|o. tion is fixed. Fourth, and worst, if the operating sy tern maintain a buffer cache of recently used disk blocks, each operating system does thi independently of Ihe t,ther ones. Thus it can happen that a certain disk block is present and dirty ,n multiple buffer caches at the same time, leading |o inLon istent results. The only way to avoid this problem is to eliminate ihe butftr tithes Doing so iS nol hard, but it hurts performance considerably. Master-Slave Multiprocessors For ihese the early da; systems to sonic new uiumprute^ur **s i<im as possu in Fig. 8-8. Here, one copy of the operating system K CPU 1 and not on any of the others. A‖ system calls are redirected to CPU I lay also run user processes if there is CPU time left — "— ■--■: CPU I is the master and all the ihese reasons, this model is rarely used any more, although it was used in y days of multiprocessors, when the goal was io port existing operating to some new multiprocessor as fast as possible. A second model is shown and its table e present t CPU 1 and not on any of t processing there. CPU I may also run us over. This model is called master-slave s others are slaves. ZJZ The master-slave model solves most of the problems of the first model. There a single daia structure (e.g., one list or a set of prioritized lists) (hat keeps track

MULTIPROCESSORS The problem with this model is that with many CPUs, the master will become a bottleneck. After all. it must handle all system calls from all CPUs. If, say, 10% ot all time is spent handling system calls, then 10 CPUs will pretly much saturate the master, and with 20 CPUs it will be completely overloaded. Thus this model is simple and workable for small multiprocessors, but for large ones il fails. Symmetric Multiprocessors Our third model, the SMP (Symmetric Multiprocessor), eliminates ihis asymmetry. There is one copy of the operating system in memory, but any CPU can run it. When a system call is made, the CPU on which the system call was made traps to the kernel and processes the system call. The SMp model is illus- illustrated in Fig. 8-9. Runs users and shared OS Runs users and shared OS CPU 3 Runs users and snared OS Runs users and shared OS OS a Figure 8-9. The SMP mull .processor model This model balances processes' and memory dynamically, since there is only one set of operating sysiem tables. It also eliminates the master CPU bottleneck, since there is no master, but it introduces its own problems. In particular, if two or more CPUs are running operating system code at the same time, disaster wi|] result. Imagine two CPUs simultaneously picking the same process to run or claiming the same free

memory page. The simplest way around these problems is to associate a mutex (i.e , lock) with the operating system, making the whole sys- system one big critical region. When a CPU wants to run operating system code, it must first acquire the mutex. If the mutex is locked, it just waits. In this way, any CPU can run the operating system, but only one at a lime. This model works, but is almost as bad as the master-slave model. Again, suppose that 10% of all run time is spent inside the operating system. With 20 CPUs, there will be long queues of CPUs waiting to get in. Fortunately, it is easy

516 MULTIPLE PROCESSOR SYSTEMS CHAP 8 to improve. Many parfs of the operating system are independent of one another. For example, there is no problem with one CPU running the scheduler while mother CPU is handling a file system call and a third one is processing a page fault Tli's observation leads to splitting the operaiing system up inio independent critical regions, that do not interact with one another. Each critical region is pro- protected bv iU own mutex. so only one CPU at a time can execute it. In this way, f;ir more parallelism ciin be achieved. However, it may well happen that some tables, such as the process table, are used by multiple critical regions. For exam- example, (he process table is needed for scheduling, but also for the tork system call ind aho tor signal handlms. Each table ihat may be used by multiple critical regions needs its own muiex. In this way. each critical region can be executed by iml\ ont CPU at a time and each critical table can be accessed by only one CPU Most modern multiprocessors use this arrangement. The hard part about writ- writing the operating system for such a machine is not that the actual code is so dif- different trom a regular operating system It is not. The hard part is splitting it into critical regions; that can be executed concurrently by different CPUs without interfering with one another, not even in subtle, indirect ways in addition, every table used by two or more critical regions must be separately protected by a muiex and all code using the table must use the muiex correctly furthermore, great care must be taken to avo]d deadlocks. If two critical regions, both need cable A and table S, and one of them claims A first and the other claims S first, sooner or later a deadlock will occin and nobody will know why In theory, all Ihe tables could he assigned integer values and all the critical regions could be required to acquire tables jn increasing order. This strategy avoids deadlocks, but it requires the programmer to think very carefully which tables each critical region needs to make the requesis in the right order As the code evolve^ over time, a critical

region may need a new table it did not previously need.]f the programmci i,s new and does not understand the lull logic of the system, then the temptation will be to just grab the nmtex on (he table at the point it is' needed and release it when it is no longer needed. However rea- reasonable this may appear, it may lead to deadlocks, which the user will perceive as the system freezing. Getting it right is- not easy and keeping it right over a period ot years ,n the face of changing programmers is very difftculi 8.1.3 Multiprocessor Synchronization The CPUs in a mulnproces^or frequently need to synchronize We just saw the case in which kernel critical regions and tables have lo be protected by mutexes- Let us now take a close look at how this synchronization actually works in a multiprocessor- It is far from trivial, as we will soon see.

SEC 8 1 MULTIPROCESSORS 517 To Start with, proper synchronization primitives are really needed. If a proc- process on a uniprocessor makes a system call that requires accessing some critical kerne] table the kernel code can just disable interrupts before touching the table. It can then do its work knowing that it will be ahle to finish without any other process sneaking in and touching the table before it is finished. On a multiproces- multiprocessor, disabling interrupts affects only the CPU doing the disable. Other CPUs con- continue to run and can still touch the critical table. As a consequence, a proper mutex protocol must be used and respected by all CPUs to guarantee that mutual exclusion works. The heart of any practical mutex protocol is an instruction that allows a memory word to be inspected and set in one indivisible operation. We saw how TSL (Test and Set Lock) was used in Fig. 2-22 to implement critical regions. As we discussed earlier, what this instruction does is read out a memory word and store it in a register. Simultaneously, it writes a 1 (or some other nonzero value) into the memory word. Of course, it takes two separate bus cycles to perform the memory read and memory write. On a uniprocessor, as long as the instruction cannot be broken off halfway, TSL always works as expected. Now think about what could happen on a multiprocessor. In Fig. 8-10 we see the worst case timing, in which memory word 1000, being used as a lock is ini- initially 0. In step 1. CPU 1 reads out the word and gets a 0. In step 2, before CPU 1 has a chance to rewrite the word to 1, CPU 2 gets in and also reads the word out as a 0. |n step 3, CPU 1 writes a 1 into the word. In step 4. CPU 2 also writes a 1 into the word. Both CPUs got a 0 back from the TSL instruction, so both of them now have access to the critical region and the mutual exclusion fails. CPU1

Memory \, Figure 8-10, The TSi in four steps show a sequent To prevent this problem, the TSL instruction must first lock the bus, prevent- preventing other CPUs from accessing it, then do both memory accesses, then unlock the bus. Typically, locking the bus is done by requesting the bus using the usual bus request protocol, then asserting (i.e., setting to a logical 1) some .special bus },ne until bovh cycle* have been completed. As long as this special line is being asserted, no other CPU will be granted bus access. This instruction can only be

518 MULTIPLE PROCESSOR SYSTEMS CHAP 8 implemented on ;i bus that has the necessary line and (hardware) piotocol ioi using them. Modem buses have these facihnes. but on earlier ones that did not. it was not possible 10 implement TSL correctly This is why Peterson's protocol was invented, to synchronize entirely in software (Peterson, 1981) If TSL is correctly implemented and used, it guarantees that mutual exclusion can be made to work. However. th,s mutual exclusion method uses a spin lock because the requesting CPU jus! sits in a tight loop testing ihe lock as tast as it ciin Not only does it completely waste the time ot the requesting CPU (or CPUs), but it may also put l\ massive load on the bus or tneniory^ seriously slowing down all other CPUs trying to do their normal work At tirst glance, it might appear that the presence ol caching should eliminate the problem of bus contention, but it does not In theory, once Ihe requesting CPU has lead the]<Kk word, it should get a copy in its cadie. As long a*, no other CPU attempts io use the lock, the requesting CPU should be able to run out ot its cache When the CPU owning the lock writes a I to it to release it. the cache protocol automatically invalidates a‖ copies of it in remoie caches requiring the correct value to be fetched again The problem is that caches operate in blocks of 32 or 64 bytes. Usually, ihe words, surrounding the lock are needed by the CPU holding the lock Since the TSL instruction is a write (because it modifies the lock), it needs exclusive access to the cache block containing the lock. Therefore every TSL invalidates the block in the lock holder's cache and fetches a privaie, exclusive copy for the requesting CPU. As soon as the lock holder touches a word adjacent lo the lock, the cache block is moved to its machine. Consequently, the entire cache block containing the lock is constantly being shuttled between the lock owner and the lock reques- requester, generating even more bus traffic than individual reads on the lock word would have. If we could gel rid of a‖ the TSL-induced writes on the requesting side, we could

reduce cache thrashing appreciably. This goal can be accomplished by hav- having the requesting CPU firs,t do a pure read to sec it the lock is fret;. Only it the lock appears to be free doe*, ii do a TSL to actually acquire it. The result of this small change is that most of the polls are now reads instead ot writes. If ihe CPU holding the lock is only reading the variables in the <ame cache block, they can each have a copy of the cache block in shared re.id-only mode, eliminating a|] the cache block transfers. When the lock is finally treed, the owner does a write, which require*, exclusive access, thus invalidating all ihe other copies tn lenioic caches. On the next read by the requesting CPU, (he cache block will be re- reloaded. Note that if two or more CPUs arc contending for the same lock, it ran happen that both see that it is free simultaneously, and both do a TSL simultane- simultaneously to acquire it. Only one of these will succeed, so there is no race condition here because tlie real acquisition is done by the TSL instruction, and this instruc- instruction is atomic. Seeing that the lock is free and then trying to grab it immediately wifh a CX u TSL does not guarantee that you get it. Someone else might win.

SEC. 8.1 MULTIPROCESSORS 519 Another way to reduce bus traffic is to use the Ethernet binary exponential backoff algorithm (Anderson, 1990). Instead of continuously polling, as in Fig. 2-22, a delay loop can be inserted between polls. Initially the delay is one instruction. If the lock is still busy, the delay is doubled to two instructions, then four instructions and so on up to some maximum. A low maximum gives fast response when the lock is released, but wastes more bus cycles on cache thrash- thrashing. A high maximum reduces cache thrashing at the expense of not noticing that the lock is free so quickly. Binary exponential backoff can be used with or without the pure reads preceding the TSL instruction. An even better idea is to give each CPU wishing to acquire the mutex its own private lock variable to test, as illustrated in Fig. 8-11 (Mellor-Ctummey and Scott, 1991). The variable should reside in an otherwise unused cache block to avoid conflicts. The algorithm works by having a CPU that fails to acquire the lock allocate a lock variable and attach itself to the end of a list of CPUs waiting for the lock. When the current lock holder exits the critical region, it frees the private lock that the first CPU on the list is testing (in its own cache). This CPU then enters the critical region. When it is done, it frees the lock its successor is using, and so on. Although the protocol is somewhat complicated (to avoid hav- having two CPUs attach themselves to the end of the list simultaneously), it is effi-

efficient and starvation free. For all the details, readers should consult the paper. CPU 3- CPU 2 spins on this (private) lock \ CPU 3 spins on this (private) lock CPU 4 spins on this (private) lock Shared memory - CPU1 holds the real lock di When CPU 1 is finished with the real lock, it releases it and also releases the private lock CPU 2 is spinning on Figure 8-11. Use of multiple locks lo avoid cache thrashing. Spinning versus Switching So far we have assumed that a CPU needing a locked mutex just waits for it, either by polling continuously, polling intermittently, or attaching itself to a list of waiting CPUs. In some cases, there is no real alternative for the requesting CPU to just waiting. For example, suppose that some CPU is idle and needs to access the shared ready list to pick a process to run. If the ready list is locked, the CPU

520 MULTIPLE PROCESSOR SYSTEMS CHAP 8 cannot just decide to suspend what it is doing and run another process, because doing that would require access to the ready list. It must wan until ii can acquire ihe ready list. However, in other cases, there is a choice. For example, if some thread on a CPU needs to access the file system buffer cache and that is currently locked, the CPU can decide to switch to a differeni thread instead of waiting- The issue of wheiher to spin or whether to do a thread switch has been a matter of much research, some of which will be discussed below. Note ihat this issue does not occur on a uniprocessor because spinning does not make much sense when there is no other CPU to release the lock. If a thread rries to acquire a lock and fails, it is always blocked to give the lock owner a chance to run and release the lock. Assuming that spinn,ng and doing a thread switch are both feasible options, the trade-off is as follows. Spinning wastes CPU cycles directly Testing a lock repeatedly is noi productive work. Sw,tching, however, also wastes CPU cycles since the current thread's state must be saved, the lock on the ready list must be acquired, a thread must be selected, its state must be loaded, and n must be started. Furthermore, the CPU cache will contain all the wrong blocks, so many expensive cache misses will occur as the new thread starts running. TLB fault are also likely. Eventually, a switch back to the original thread must take plact with more cache misses following it. The cycles spent doing these two context switches plus a[] the cache misses are wasted. If it is known that mutexes are generally held for, say, 50 jisec and it takes I msec to switch from the current thread and I msec to switch back later, it is more efficient just to spin on the mutex. On the other hand, if the average mutex is

held for 10 msec, it is worth the trouble of making the two context swiiches1 The trou- trouble is that critical regions can vary considerably in their duration, so which approach is better? One design is to always spin. A second design is io always switch. But a ihird design is to make a separate decision each time a locked mutex is encoun- encountered. At the time the decision has to be made, it is not known whether it is better to spin or switch, but for any given system, it is possible io make a trace of all activity and analyze it later offline. Then it can be said m retrospect which deci- decision was, the best one and how much time was wasted in the best case. This hind- hindsight algorithm then becomes a benchmark against which feasible algorithms can be measured. This problem has been studied by researchers (Karlm et al., 1989: Karlin et a|... 1991; and Ousterhout, 1982). Most work uses a mode! in which a thread fail- Ing to acquire a mutex spins for some period of time. If this threshold is ex- exceeded, it switches. In some cases the threshold is fixed, typically the known overhead for switching to another thread and then switching back. In other cases it is dynamic, depending on the observed history of the mutex being waited on. The best results are achieved when the system keeps track of the last few observed spin times and assumes that this one will be similar to the previous ones

SEC 8 I MULTIPROCESSORS 521 R>r example, assuming a l-mscc context switch lime again, a thread would spin lor a maximum of 2 msec, but observe how long it actually spun. If it tails to acquire a lock and sees thai on the previous three runs il waned an average of 200 u.sec, it should spin for 2 msec hetore switching However, it if sees that it spun |or (he full 2 msec on each at the previous attempts, it should switch immediately and not spin at all. Moie details can be found m (Karlin et al., l99l) 8.1.4 Multiprocessor Scheduling On a uniprocessor, scheduling is one dimensional The only question that must be answered (repeatedly) is: "Which process should be run next'" On a multiprocessor, scheduling is two dimensional The scheduler has to decide which process to run and which CPU to run it on. This extra dimension greatly complicates scheduling on multiprocessors Another complicating factor is that in some systems, a|l the processes are unrelated whereas in others they come in groups. An example of the former situa- situation is a timesharing system in which independent liters start up independent processes The processes are unrelated and each one- can be scheduled without regard to the other ones An example of the latter situation occurs regularly in program development

environments. Large systems often consist ot some number ot header files con- containing macros, iype definitions, and variable declarations that are used by the actual code files. When a header tile is changed, all the code files that include it must be recompiled. The program make is commonly used |o manage develop- development. When make is invoked, it starts the compilation of only those code file*, that must be recompiled on account of changes to the header or code tilts. Object files that are still valid are not regenerated The original version of make did its work sequentially, but iiewer \ersions designed for mu|liprocessors can siart up all the compilations at once. If 10 com- compilations are needed, it does not make sense to schedule ') or them quickly and leave the last one until much later since the user will not perceive the work as completed until Ihe last cine finishes. In this case it makes sense to regard the processes as a group and to take that into account when scheduling them Timesharing Let Us, first address the case of scheduling independent processes; later we will consider how to schedule related processes The simplest scheduling algo- algorithm tor dealing with unrelated processes (or threads) is to have a single system- wide data structure for ready processes, possibly just a list, but more likely a set of iisls for processes at different priorities as depicted in Fig 8-|2(a) Here the 16 CPUs are all currently busy, and a prioritized set of 14 processes are waiting to run The first CPU to finish its current work (or have its process block) is CPU 4,

522 MULTIPLE PROCESSOR SYSTEMS CHAP R which then locks the scheduling queues and selects the highest priority process, A, as shown in Fig- 8-12(b). Next, CPU |2 goes idle and chooses process 8, ;is illus- illustrated m Fig. 8-[2(c) As long as the processes are completely unrelated, do,ng scheduling this way is .1 reasonable choice 0000 0 6 0 d 03 [HI Figure 8-12. Using j single dau struaurt: for scheduling a inutii Having a single scheduling data st CPUs, much as they would be in a uni matic load balancing because it can nev ers are overloaded. Two disadv cture used by all CPUs timeshares the rocessor system It also provides auto- autor happen that one CPU is idle while oth- othf this approach are the potential conten- contention for the scheduling data structure as the numbers of CPU;, grows and the u overhead in doing a context switch when a process blocks for I/O It is also possible that a context switch happens when a process' quantum expires. On a multiprocessor, that has. certain properties not present on a unipro- uniprocessor. Suppose that the process holds a spin lock, not unusual on multiproces- multiprocessors, as

discussed above. Other CPUs waiting on |he spin lock jusi waste their time spinning until that process is scheduled again and releases the look On a uniprocessor, spin locks are rarely used so it a process is suspended while it holds a mutex, and another process, starts and tries to acquire the muiex. it will be immediately blocked, so |itt|e lime is wasted To get around Ihis anomaly, some systems- use smart scheduling, in which a process acquiring a spin lock sets a process-wide flag to show that it currently has a spin lock (Zahorjan et a[., |99I). When it releases the lock, rt clears the flag The scheduler then does not stop a process holding a spin lock, but instead gives it a little more time to complete its critical region and release the lock. Another issue that plays a role in scheduling is the fact that while all CPUs are equal, some CPUs are more equal. In particular, when process A has run for a long time on CPU k, CPU it's cache will be full of A\ blocks. If A gets to run

 again won, it may perform belter if it is run on CPU k. because k\ cache may still contain some of A\ blocks. Having cache blocks preloaded will increase the cache hit rate and thus the process' speed In addition, the TLB may also contain the riyht pages, reducing TLB faults. Some multiprocessors take ihb effect into account and use what is called affinity scheduling (Vaswani and Zahorjan, 1991). The basic idea here is to make a serious effort to have a process run on the same CPU it ran on las! lime One \vay to create this affinity is to use a two-level scheduling algorithm When a process is created, it is assigned to a CPU, for example based on which one has the smallest load ai thai moment. This, assignment of processes to CPUs is the top level of the algorithm As a result, each CPU acqu]res its own collection ol processes. The actual scheduling of the processes is the bottom level of the algoruhm. It is done by each CPU separately, using pnoriiics or some other means. By trying to keep a process on the same CPU, cache affinity is maximized, However, if a CPU hat. no processes to run. il takes one from another CPU rather than go idle. Two-level scheduling has three benefits. First, it distrtbuics the load roughly evenly over ihe available CPUs. Second, advantage is taken of cache affinity where possible. Third, by giving each CPU its own ready list, contention for the ready lists is minimized because attempts to use anoiher CPU's ready list are rela- relatively infrequent. Space Sharing The other general approach to multiprocessor scheduling can be used when proce;>ses are related to one another in some way Earlier we mentioned the example of parallel make as one case. It also often occurs thai a single process

creates multiple threads that work together. For our purposes, a job consisting of multiple related processes or a process consisting of multiple kernel threads are essentially the same thing. We will refer to the schedulable entities as threads here, but the material holds tor processes as well Scheduling multiple threads ai the same lime across multiple CPUs is called spate sharing. The simplest space sharing algorithm works like this. Assume that an entire group of related threads is created a(once At the time it is created, the scheduler check;, to see if there are as many free CPUs as there are threads. If there are, each thread is given us own dedicated (i.e , nonmultiprogrammed) CPU and they all start. If ihere are not enough CPUs, none ot the threads are started until enough CPUs, are available. Each thread holds onto its CPU until it terminates, at which time the CPU is put back into the pool of available CPUs. If a thread blocks on I/O, it continues to hold the CPU, which is simply idle until the thread wake.s up When the next batch of threads appears, the same algorithm is applied. At any ,nstant of time, the set of CPUs ,s statically partitioned into some number of partitions, each one running the threads of one process. In Fig. 8-13.

524 MULTIPLE PROCESSOR SYSTEMS CHAP 8 we have partiiions of sizes 4. 6, 8. and 12 CPUs, with 2 CPUs unassigned, tor example. As time goes on, the number and size of tlic partitions will change as processes come and go Periodically, scheduling decisions have to be made In uniprocessor systems, shortest job first is a well-known algorithm for batch scheduling. The analogous algorithm for a multiprocessor is to choose the process needing the smallest number of CPU cycles, thai is tbe process whose CPU-count x run-time is the smallest of the candidates However, in practice, this information is rarely avail- available, so the algorithm is haid to carry onl In fact, studies have shown lhat, in praciice, healing first-come, first-served is hard to do (Kruegcr et al-, 1994) In this simple partitioning mode|. a process just ask*- tor some number of CPUs and either gets them all or has to wait until they are available. A different approach is for processes 10 actively manage the degree of parallelism One way to do manage the parallelism is to have a central server that keeps track of which processes are running and want 10 run anil what their minimum and maximum CPU requirements are (Tucker and Gupta. |98lJ) Periodically, each CPU polls the central server to ask how many CPU1, it may use It then adjusts the number of processes or thread-, up or down to match whs! is available. For example, a Web server can have I, 2, 5, 10, 20, or

any other number of threads running m parallel. If ii currently has 10 threads and there is suddenly more demand for CPUs and it if, told to drop to 5. when the next 5 threads finish their current work, they are told to exit inroad of being given new work This scheme allows the partition sizes to vary dynamically to maich ihc curreni workload better than the fixed system of Fig- 8-13 Gang Scheduling

SEC 8.1 MULTIPROCESSORS 525 that attempt to schedule in both time and space together, especially for processes that create multiple threads, which usually need to communicate with one another. To see the kind of problem that can occur when the threads of a process (or processes of a job) arc independently scheduled, consider a system with threads .4A and A , belonging to process A and threads So and 8 , belonging to process B. threads ^o and So are timeshared on CPU 0; threads A j and 8, are timeshared on CPU I threads Aa and A , need to communicate often. The communication pattern is that^o sends A i a message, with A i then sending back a reply toA0, followed hy another such sequence. Suppose that luck has it that A 0 and 8, start first, as shown in Fig. 8-14- CPU of" CPUiP \Requ B, eSr1 / ^/Rep =, i A «, piv2 Figure 8-14. Communication beiween two threads belonging to process A ihat In time slice 0, At) sends A ; a request, but A j does not get it until it runs in time slice 1 starting at 100 msec It sends the reply immediately, but A (, does not get the reply until it runs again at 200 msec. The net result is one request-ieply sequence every 200 msec. Not very good The solution io this problem is gang scheduling, which is an outgrowth of co-scheduling (Ousterhout, 1982). Gang scheduling has three part-,: 1 Groups of related threads are scheduled as a unit, a gang- 2, All members of a gang run simultaneously, on different timeshared CPUs. 3, All gang members start and end their time slices together. The trick that makes gang scheduling work is that a|] CPUs are .scheduled syn-synchronously. This means thai nine is divided into discrete quanta as we had in Fig. 8-14. At the start of each new quantum, all the CPUs are rescheduled, with a new thread being started on each one, Ai the start of the following quantum, anolher scheduling event happens. In beiween, no scheduling is done. If a thread blocks, its CPU stays idle until the end of the quantum. An example of how gang scheduling works is given in Fig. 8-15. Here we have a multiprocessor with six CPUs being used by five processes, A through E.

MULTIPLE PROCESSOR SYSTFMS CHAP H a total of 24 ready threads.

During time slot 0. threads AQ through Au are lukd and run. During time slot I, Threads B().Bt< Ba, CV C,. and C 3 aie lulccl and run During time slot 2, D'i, five threads and f0 gel to run. The irung six threads belonging to process E run m time slot 3. Then the cycle ts. with slot 4 being the same as slot 0 and so on Bo Do F, \ B, Do E, Ai B, E, 8i D, E3 A, B, Da E-i Aa B., O, E, cd D3 E4 A, Oq E, A4 C, D4 E5 D4 E5 A5 c3 Eo ER A5 C2 Figure 8-15. Gang sUieduhni: The idea of gang scheduling js to have all the- threads of a process run together, so that if one of them sends a request to another one. it will get the mes- message almost immediately and be able to reply almost immediately. In Fig. 8-15, since all the A threads are running together, during one quantum, they may send and receive a very large number of messages in one quantum, thus eliminating the problem of Fig. 8-|4. 8.2 MULTICOMPUTERS Multiprocessors are popular and attractive because they offer a simple com- municaiion modei: all CPUs s'hare a common memory. Processes can write mes- messages to memory that can then be read by other processes. Synchronization can be done using mutexes, semaphores, monitors, and other well-established tech- techniques The only fly in t|ie ointment U that large multiprocessors are difficult to build and thus expensive. To get around these problems, much research has been done on multicomput- ers. which are tightly-coupled CPUs that do not share memory. Each one has its own memory, as shown in Fig. 8-|(b) These systems, are also known by a variety of other names, including cluster computers, and COWS (Clusters of Worksta- Workstations). Multicomputer are easy to build because the basic component is jusi a stripped-down PC with the addition of a network interface card. Of course, the secret to getting high performance is to design the interconnection network and

SEC. K 2 M5 LT1COM|><-' I'EKS 527 the interface card cleverly This problem is completely analogous 10 building the shared memory m a mul microsecond time scale, .so it is simpler, cheaper. In the following s& hardware, especially Ihe iproc L-essor. However, the goal is to send messages on a d g memory on a nanosecond time s :r to accomplish. I first take a brief look at multicompute interconnection hardware. Then we will move onto the starting with [ow-leve! communication witwaie. then high-level com- communication software We will also look at a way shared memory can be achieved on systems that do not have it. Finally, we will examine scheduling and load 8.2.1 Multicomputer Hardware . The basic node of a multicomputer

consists ot a CPt-I, memory, a network interface, and sometimes a hard disk. The node may be packaged in a standard PC case, but ihe graphics adapter, moniior, keyboard, and mouse are nearly always absent. In some cases, the PC contains a 2-way or 4-way muliiprocessor board instead of a single CPU, but for simplicity, we will assume thai each node has, one CPU. Often hundreds or even thousands ot nodes are hooked together to form a multicomputer. Below we will say a little about how this hardware is organized. Interconnection Technology Each node has a network interface card with one or two cables (or fibers) coming out of it. These cables connect to either other nodes or to switches, in a small system, there may be one switch to which all the nodes are connected in ihe star topology of Fig. 8-16(a). Modern switched Ethernets use this topology As; an alternative to the single switch design, the nodes may form a ring, with two wires coming out the network interface card, one going into the node on the left and one going into the node on the right, as shown in Fig. 8-|6(b).]n this topology, no switches are needed and none are shown. The grid or mesh of Fig. 8-16<c) is a two-dimensional design that has been used in many commercial systems. It is highly regular and easy to scale up to large sizes. It has a diameter, which is the longest path between any two nodes, and which increases only as the square root of the number of nodes. A variant on the grid is the double torus of Fig. 8-16<d), which is a grid with the edges con-connected. Not only is it more fault tolerani than the grid, but the diameter is also less because the opposite corners can now communicate in only two hops. The cube of Fig. 8-f6(e) is a regular three-dimensional topology. We have illustrated a 2 x 2 x 2 cube, but in the general case it could be a k x k x it cube. In Fig. 8-l6(f) we have a lbur-dimensional cube constructed from tw<(three-dimensional cubes with the corresponding nodes connected. We could make a

o h>r i tuhe five-dimensmnal cube by clonmg the stiuctuie of Fig 8-!6(O and connecting the corresponding nodes 10 fonii a block of four cubes. J\> go to six dimensions, we could replicate the block of four cubes and interconnect the corresponding nodes, and so on. An rc-djmensional cube formed this way is called a hypercube. Many parallel computers use this topology because- the diameter grows linearly with the dimensionality. Put in other words, the diameter is the base 2 logarithm ot the number ot nodes, so, tor example, a 10-iiiinensi0nal hypercubc has 1024 nodes, but a diameter of only iO, giving excellent delay properties. Note that in contrast, 1024 nodes arranged as a 32 x 32 grid has a diameter of 62. more than six times worse than the hypercube.

The price paid tor the smaller diameter is that tlic fanout and thus the number ot links (and the cost) is much larger tor the hyper- hypercube. Two kinds of switching schemes arc used in multicomputer*- In the first one. each message is firsf broken up (either by the user software or the neiwork inter- interface) into a chunk of some maximum length called a packet The switching scheme, called store-and-forward packet switching, consists of the packet being injected into the first switch by the source node's network interface board, as shown in Fig. 8-17(a). The bits come in one ;it a time, and when the whole packet has arrived, it is copied to the next switch along ihe path, as s,hown in Fig. 8- 17(b). When ihe packet arrives at the switch attached to the destination node, as

shown in Fig- 8-l7(C), ihe packet and eventually to its RAM. MULTICOMPUTERS ipied to thai node\ netv 529 ark interface board Figure B-i7. Store-and-for d pjckei iiching. While store-and-forward packet switching is flexible and efficient, it does have the problem of increasing latency (delay) through the interconnection net- network. Suppose that the time to move a packet one hop in Fig. 8-17 is T nsec Since the packet must be copied four limes to get it from CPU I to CPU 2 (to A. to C, to D, and to the destination CPU), and no copy can begin until the previous one is finished, the latency through Ihe interconnection network is 47". One way out is to design a hybrid network, with some of the properties of circuit switching and some of the properties of packet switching. For example, each packet can be logically divided inio smaller units. As soon as the first unit arrives at a switch, it can be moved to the next switch, even before ihe tail of the packet has arrived. The other switching regime, circuit switching, consists of the first switch first establishing a path through all the switches to the destination switch Once that path has been set up, ihe bits are pumped al(the way from the source to the destination nonstop. There is no intermediate buffering at the intervening switches. Circuit switching requires a setup phase, which takes some time, but is faster once the setup has been completed. After the packet has been sent, the path must be torn down again. A variation on circuit swiiching, called wormhole routing, breaks each packet up into subpackets and allows the first subpacket to start flowing even before the full path has been built. Network Interfaces All ihe nodes in a multicomputer have a plug-in board containing the node's connection to the interconnection network that holds the multicomputer together. The way these boards are built and how they connect to the main CPU

and RAM

530 MULTIPLE PROCESSOR SYSTEMS CHAP 8 have subsiantial implications for the operating system. We will now briefly look at some of the issues here This material is based in part on (Bhoedjang. 2000) Other references are (Buzzard et al., 1996: Pakm et al., 1997: Steenkiye, 1994; and Von Eicken et al., 1992). In virtually a|l muti[computers, the interface board contains some RAM tor holding outgoing and incoming packets. Usually, an outgoing packet has to be copied to (he interface board's RAM before it can be transmitted to the first switch. The reason for this design is that many interconnect ion networks are syn- synchronous, s-o that once a packet transmission has started, the bits must continue flowing at a constant rate. If the packet is in the main RAM, this continuous flow out onto the network cannot be guaranteed due to other traffic on the memory bus. Using a dedicated RAM on the interface board eliminates this problem. This design is shown in Fig. 8-18. Figure S-18. Posmon of the network mterface boards m a multicomputer The same problem occurs with incoming packets. The bits arrive from the network at a constant and often extremely high rate.]f the network interface board cannot store them in real time as they arrive, data will be lost. Again here, trying to go over the system bus (e.g., the PCI bus) to the main RAM is too risky. Since the network board is typically plugged into t|ie PCl bus, this is the only con- connection it has to the main RAM, so competing for this bus with the disk and every other I/O device is inevitable. It is safer to store incoming packets in the interface board's private RAM and then copy them to the main RAM later. The interface board may have one or more DMA channels or even a complete CpU on board. The DMA channels can copy packets between the interface board and (he main RAM at high speed by requesting block transfers on the system bus, thus transferring several words without having to request the bus separately for each word. However, it is precisely this kind of block transfer, which ties up the

SPX $ 2 MULT [COMPUTERS 531 system bus for multiple bus cycles, thai makus the interface bo;ird RAM necessary in the first place. Some interface boards, have a full CPU on them, possibly m addition to one or more DMA channels,. This design means that the main CPU can offload some work to the network board such as handling reliable transmission (if the underly- underlying hardware can lose packets), multicasting (sending a packet to more

than one destination), and taking care of protection in a system with multiple processes. However, having two CPUs means that they must synchronize to avoid race con- conditions, which adds cxira overhead and means more work for the operating sys- 8.2.2 Low-Level Communication Software The eneinv of hi"h performance communication in multicomputer systems is excess cop ne or p k In h best case, there will be one copy from RAM tt> the mterfa e b a d h ode. one copy from the source interface board to the destin on n b d (if no storing and torwarding along the path occurs), and one py n h to the destination RAM, ,i total of three copies. However n n an en worse. In particular, if the interface board is mapped in k n | rt 1 Id space and not user virtual address space, a user process c n onl nd p k bv issuing a system call that traps to the kernel. The kern I nay ha pj he packets to their own memory both on output and on inp n p| a >id page faults while transmitting over the net- network. Al o he ng k n I probably does not know where to put incoming packets un I h h d h n o examine (hem. These five copy steps are illus- illustrated in Ftb 8 18 If copies (o and from RAM domina(e Iho performance, the extra copies to and from the kernel ma\ double the end to-tiid delay and cut the bandwidth in half. To avoid this pertormantt hit many multicomputer^ map (he interface board directly into user sp ce and allow tht ustr process to put (he packets on the board directly, without the kernel being involved. While this approach definitely helps performance, it introdui.es two problem First, whai if several processes are running on the node and need network access to send packets7 Which one gets the interface board in its address space? Having a system call 10 map the board in and out of a virtual address space is expensive, but if only one process gets ihe board, how do |he oiher ones send packets'? And what happens if |he board is mapped into process ,4's virtual address space and a packet arrives for process 8, especially iM and B have dif- different owners, neither of whom wanis to put in any effort to help iht other One solution is to map the interface board into all processes that need it but then a mechanism is needed to avoid race conditions. For example if 1 claim i buffer on the interface board and then due to a time slice, 8 run and daims the same buffer, disaster results. Some kind of synchronization mechanism is needed

 but these mechanisms, such as mutexes, only work when (he processes are assumed to be cooperating. In a timesharing environment with multiple users all in a hurry to get their work done, one user might just lock the mutex associated with the board and never

release it. The conclusion here is that mapping the inter- interface board into user space only really works well when there is just one user proc- process running on each node unless special precautions are taken (for example, dif- different processes gei different portions of the interface RAM mapped into their address spaces). The second problem is that the kernel may well need access to the intercon- interconnection network itself, for example, to access the file system on a remote node. Having the kernel share the interface board with any users is not a good idea, even on a timesharing basis. Suppose that while the board was mapped into user space, a kernel packet arrived'1 Or suppose that the user process sent a packet to a remote machine pretending to be the kernel1' The conclusion is that the simplest design is to have two network interface boards, one mapped into user space tor application traffic and one mapped into kernel space for use by the operating sys- system. Many multicomputer do precisely this. Node to Network Interface Communication Another issue is how to get packets onto the interface board The fastest way is to use the DMA chip on the board to just copy them in from RAM. The prob- problem with this approach is that DMA uses physical rather than virtual addresses and runs independently of the CPU. To start with, although a user process cer- certainly knows the virtual address of any packet it wants to send, it generally does not know the physical address. Making a system call to do the vinual-to-physical mapping is undesirable, since the point of putting the interface board in user space in the first place was to avoid having to make a system call for each packet to be sent. In addition, if the operating system decides to replace a page while the DMA chip is copying a packet from it, the wrong dala will be transmitted. Worse yet, if the operating system replaces a page while the DMA chip is copying an incoming packet to it, not only will the incoming packet be lost, but also a page of innocent memory will be ruined These problems can be avoided by having system calls to pin and unpin pages in memory, marking them as temporarily unpageable. However, having to make a system call to pin the page containing each outgoing packet and then having to make another call later to unpin it is expensive. If packets are small, say, 64 bytes or less, the overhead for pinning and unpinning every buffer is prohibitive For large packets, say. 1 KB or more, it may be tolerable. For sizes m between, it depends on the details of the hardware (Bhoedjang. 2000) In theory, the same problem occurs with DMA from a disk or other device, but since these are set up by the operating system to kernel buffers, it in easy for

SEC. 8.2 MULTICOMPUTERS 533 the system to avoid paging the buffers. The problem here is that the user is sewing up and managing the DMA, and the operating system does not know that remov- removing a page could be fatal, something it does know for I/O that it starts itself. The reason using kernel buffers is acceptable for disk I/O and not for multiprocessor communication is that an extra 20-nsec delay is tolerable for disk latency but not for process-to-process communication latency. The DMA problem can be avoided by having the user process first pin one page ai startup and asking for its physical address. Outgoing packets are first copied there and then to the network interface, but this extra copy is just as bad as copying to the kernel. For these reasons, using programmed I/O to and from the interface board is usually the safest course, since any page faults encountered are just ordinary CPU page faults and can be handled in the usual way by the operating system. When a page fault occurs, the copy loop stops instantly and remains stopped until the operating system has handled the page fault. A more sophisticated scheme is to use programmed I/O for small packets and DMA with pinning and unpinning for large ones. If the network interface boards have their own CPUs (e.g.. as do Myrinet boards), these on-board CPUs can be used to speed up communication. However, care has to be taken to avoid race conditions between the main CPU and the on- onboard CPU. One way to avoid races is illustrated in Fig. 8-19, where we focus on node 1 sending packets and node 2 receiving them, not necessarily from each other. The key synchronization data structure for senders is the send ring; for receivers it is the receive ring. All nodes have both since they all send and receive. Each ring has room for n packets. There is also a bitmap per ring with n bits, possibly separate (as shown) or possibly integrated into die rings, telling which ring slots are currently valid. Figure 8-19. Use of send and receive rings to coordinate the main CPU with the on-board CPU

534 MULTIPLE PROCESSOR SYSTEMS CHAP H When a sender has j new packet to send, it fust checks to see it there is an available slot in the send ring If nor. it must wart, to prevent overrun If (here is a slot, it copies the packet to the next available slot, and atter that is completed, sets the corresponding bit in the bitmap. When the on-board CPU has finished whatever it is doing, it checks the send ring If it contains any packets, it takes the one there longest and transmits it When ii [S done, it clears the corresponding hit in the bitmap. Since the main CPU is the only one that sets the bits and the on- onboard CPU

is the only one that clears them, there are no lace conditions. The receive ring works the olher way. with the on-board CPU setting a bit to indicate packet arrival and the main CPU clearing il to indicate thai it has copied ihe packet and freed the buller This scheme can also he used even without programmed I/O done by the main CPU In that case, ihe send ring entry docs not contain the packet itselt. but a pointer to the packet m the mam RAM. When the on-board CPU is ready to transmit the packet, it fetches the packet to the interface board, either using pio- grammed I/O itself or via DMA. In both cases, this approach works only if the page containing the packet is known to be pinned. 8.2.3 User-Level Communication Software Processes on different CPUs on a multicomputer communicate by sending messages to one another In the simplest form, this message passing rs exposed to the user processes. In other words, the operating system provides a way to send and receive messages, and library procedures make these underlying calls avail- available to user processes. In a more sophisticated torm, the actual message passing is hidden from users by making remote communication look like a procedure call We will study both of these methods below. Send and Receive At the barest minimum, the communication services provided can be reduced to two (library) calls, one for sending messages and one for receiving them. The calling for sending a message might be sendfdest, Smptr); and the call tor receiving a message might be receive(addr. &mptr); / oa process identified by dei\ h m age has been sent. The latter ves When one does, the mes- mes/ d the caller i,s unblocked. The h h he receiver is listening Many Tisare possible. g S mult (computers are static, with the numbt ICPJ t d h > h ndle addressing is to make addr a The torme and c cause sage .iddr vanai s the is cop param its o One i h 11 d d h 11 b h p h rn bt bl k b f h P d dd S P bock d n P n dd ng n d 1 d d d n n b h d n

SJ-C 8 2 MLLTirOMrL'lERS 535 two-part address consisting of a CPU number and a process or port number on the addressed CPU. In this way each CPU can manage its own addresses without potential conflicts. Blocking versus Nonblocking Calls The calls described aboie are blocking calls (sometimes called synchronous calls). When a process calls send, it specifies a destination and a buffer to send to thai destination. While the message is being sent, the sending process is blocked (i.e . suspended). The instruction following the call to send is not executed until the message has been completely sent, as shown in Fig. 8-20(a). Similarly, a call to receive does not return control until a message has actually been received and put in the

message butter pointed to by the parameter The process remains suspended m receive until d message arrives, even if it takes hours'. In some sys- systems, the receiver can specify from whom it wishes to receive, in which case it remains blocked until a message from that sender arrives. — *em81 inder blocked Message being sent Message ' kernel Duffer (b) Figure 8-20. (a) A blocking send call (b) A nonblocking send call An alternative to blocking calls are nonblocking calls (sometimes called asynchronous calls) If send is nonblocking, it returns control to the caller immediately, before the message is sent. The advantage of this scheme is that ihe sending process can continue computing m parallel with the message transmis- transmission, instead of having the CPU go idle (assuming no other process is runnable).

536 MLLHPLE PROCESSOR SYSTEMS CHAP 8 The choice between blocking and nonblocking primitives is normally made by ihe system designers (i.e.. either one primitive is available or the other), although in a few systems both are available and users can choose their fa\orite. However, the performance advantage ottered b\ nonblocting primitives is offset by a serious disadvantage: the sender cannot modify the message buffer until the message ha.s been sent. The consequences of the process overwriting the message during transmission are too horrible to contemplate. Worse yet, the sending process has no idea of when the transmission is done. so it never knows when it is safe to reuse the buffer It can hardly avoid touching it forever. There are three possible ways out. The first solution is to have the kernel copy the message to an internal kernel buffer and then allow the process to con- continue, as shown in Fig. 8-20(b). From the sender's point of view, this scheme is the same as a blocking call: as soon as it gets control back, it is free to reuse the buffer. Of course, the message will not yet have been sent, but the sender is not hindered by this fact. The disadvantage of this method is that every outgoing message has to be copied from user space to kernel space With many network interfaces, the message will have to be copied to a hardware transmission buffer later anyway, so the first copy is essentially wasted. The extra copy can reduce the performance of the system considerably The second solution is' to interrupt the sender when the message has been sent to inform it that the buffer is once again available. No copy is required here, which saves time, but user-level interrupts make programming tricky, difficult, and subject to race conditions, which makes them irreproduciblc and nearly impossible to debug. The third solution is to make the buffer copy on write, that is. to mark it

as read-only until the message has been sent. If the buffer is reused before the mes- message has been sent, a copy is made. The problem with this solution is that unless the buffer is isolated on its own page, writes to nearby variables will also force a copy, Also, extra adminisiration is needed because the act of sending a message now implicitly affects the read/wnte status of the page. Finally, sooner or later the page is likely to be written again, triggering a copy that may no longer be necessary Thus the choices on the sending side are 1. Blocking send (CPU idle during message transmission). 2. Nonblocking send with copy (CPU time wasted for the extra copy) 3, Nonblocking send with interrupt (makes programming difficult) 4, Copy on write (extra copy probably needed eventually) Under normal conditions, the first choice is the best one, especially if multiple threads are available, in which case while one thread is blocked trying to send, other threads can continue working. It also doe.s not require any kernel buffers to

SbC «2 MULTI COMPUTERS 537 be managed. Furthermore, as can be seen from comparing Fig 8-2O(■.! to hig 8- 20(b)_the message will usually be out the door taster if no copy is required For the record, we would like to point out thai some authors use a diffeient criterion To distinguish synchronous trom asynchronous primitives. In the alterna- alternative view, a call is synchronous only if the sender k blocked until the message has been received and an acknowledgement seni back (Andrews. 1991). In the world ot real-time communication, synchronous has yet another meaning, which can lead to confusion, unfortunately. Just as send can he blocking or nonlocking, so can receive A blocking call lust suspends the caller until a message has arrived If mulnple threads are avail- available, this is a simple approach Alternately, a nonblocking receive just tells the kernel where the buffer is and returns control almost immediately An interrupt can be used to signal that a message has arrived However, interrupts are difficult to program and are also quite slow, so it may be preferable for the receiver to poll for incoming messages using a procedure, poll, that tells whether any messages are waiting. If so, the caller can call get -message, which returns the first arrived message. In some systems, the compiler can insert poll call", in the code at appropriate places, although knowing how ofien to poll is tricky Vet another option is a scheme in which the arrival of a message causes a new thread to be created spontaneously in the receiving process' address space Such a thread is called a pop-up thread. It runs a procedure specified in advance and whose parameter

is a pointer !o the incoming message. After processing the mes- message, it simply exits and is automatically destroyed A variant on this idea is to run the receiver code directly in the interrupt handler, wilhout going to the trouble o(creating a pop-up thread. To make this scheme even faster, the message itself contains the address of the handler, so when a message arrives, the handler can be called in a few instructions. The big win here is that no copying at all is needed The handler takes the message from the interface board and processes it on the fly This scheme is called active mes- messages (Von Eicken et al, 1992). Since each message contains the address of the handler, active messages only work when senders and receivers trust each other completely.
8.2.4 Remote Procedure Call Although the message-passing model provides a convenient way to structure a multicomputer operating system, it suffers from one incurable flaw the basic paradigm around which all communication is built is input/output. The pro- procedures send and receive are fundamentally engaged in doing I/O and many peo- people believe that I/O is the wrong programming model. This problem has long been known, but little was done about it until a paper by Birrell and Nelson A984) introduced a completely different way of attacking the problem. Although the idea is refreshingly simple (once someone has thought

538 MULTIPLE PROCESSOR SYSTEMS CHAP 8 ill examine the of ii). the implications are often subtle. In this section cept, its implementation, its strengths, and its weaknesses. In a nutshell, what Birrell and Nelson suggested was allowing programs to call procedures located on other CPUs. When a process on machine 1 calls a pro- procedure on machine 2, the calling process on 1 is suspended, and execution of the called procedure takes place on 2. Information can be transported from the caller lo the callee in the parameters and can come back in the procedure result. No message passing or I/O at all is visible to the programmer. This technique is known as R3PC (Remote Procedure Call) and has become the basis of a large amount of multicomputer software. Traditionally the calling procedure is known as the client and the called procedure is known as the server, and we will use those names here too. The idea behind RPC is io make a remote procedure call look as much as pos- possible like a local one. In the simplest form, io call a remote procedure, the client program must be bound with a small library procedure called ihe client stub that represents the server procedure in the client's address space. Similarly, the server is bound with a procedure called the server stub. These

procedures hide the fact that ihe procedure call from the client to the server is not local The actual steps in making a RPC are shown in Fig. 8-21. Step 1 is the clieni calling the client stub. This call is a local procedure call, with the parameters pushed onto the stack in the normal way. Step 2 is the client stub packing the parameters into a message and making a system call to send the message. Pack- Packing the parameters is called marshaling. Step 3 is the kernel sending the message from the client machine to the server machine. Step 4 is the kernel passing the incoming packet to the server stub (which would normally have called recene earlier). Finally, step 5 is the server stub calling the server procedure. The reply traces the same path in the other direction. Ghent CPU Server CPU Operating system * 3 | Operating system)| Figure 8.21. Steps m making a remote procedure call. The stubs are shaded gray. The key item to note here is that the client procedure, written by the user, just makes a normal (i.e., local) procedure call to the client stub, which has the same

SEC 8 2 MULTICOMPUTERS 539 name as the server procedure. Suite the client procedure and client siub are in the same address space, the parameters are'passed in the usual way. Similarly, the server procedure is called by a procedure in its address space with the parameters it expects. To the server procedure, nothing is unusual. In this way, instead of doing I/O usmg send and receive, remote communication is done by taking a nor- normal procedure call. Implementation Issues Despite the conceptual elegance of RPC. there are a few snakes hiding under the grass. A big one is the use of pointer parameters. Normally, passing a pointer to a procedure is not a problem. The called procedure can use the pointer the same way the caller can because the two procedures reside in the same virtual address space. With RPC passing pointers is impossible because the client and server are in different address spaces. In some cases, tricks can be used to make it possible Co pass pointers Sup- Suppose thai the first parameter is a pointer to an integer, k. The client stub can marshal k and send it along to the Kerver. The server stub then creates a pointer to *: and passes it to the server procedure ju-.t as it expects. When the server pro- procedure returns control to the server stub, the latter sends it back to the client where the new k is copied over the old one, just in case ihe server changed it. In effect, the standard calling sequence of call-by-reference has been replaced by copy- restore. Unfortunately, this trick does not always work, for example, if the pointer points to a graph or other complex data structure. For this reason, some restric- restrictions must be placed on

parameters to procedures called remotely. A second problem is ihat in weakly-typed languages, like C, it is perfectly legal to write a procedure that computes the inner product of two vectors (arrays), wfthout specifying how large either one is. Each could be terminated by a special value known only to the calling and called procedure. Under these circumstances, it is essentially impossible for the client stub to marshal the parameters: it has; no way of determining how large they are, A third problem is that it is not always possible to deduce the types, of the parameters, not even from a formal specification or the code itself. An example is printj, which may have any number of parameters (at least one), and they can be an arbitrary mixture of integers, shorts, longs, characters, strings, floating-point numbers of various lengths, and other types. Trying to call print/as a remote pro- procedure would be practically impossible because C is so permissive. However, a rule saying that RPC can be used provided that you do not program in C (or C++) would not be popular. A fourth problem relates to the use of global variables. Normally, the calling and called procedure may communicate using global variables, in addition to sia parameters. If the called procedure is now moved to a remote machine, the code will fall because the global variables are no longer shared.

540 MULTIPLE PROCESSOR SYSTEMS CHAP 8 These problems are not meant to suggest thai RPC is hopeless In tact, it _is widely used, but some restrictions and tare are needed to make it work well in practice. 8.2.5 Distributed Shared Memory Alihough RPC has, its attractions, many programmers still prefer a model of shared memory and would like to use it, even on a multicomputer. Surprisingly enough, it is possible to preserve the illusion of shared memory reasonably well, even when it does not actually exist, using a technique called DSM (Distributed Shared Memory) (Li, 1986; and Li and Hudak, 1989). With DSM, each page is located in one of the memories of Fig. 8-1. Each machine has its own virtual memory and its own page tables. When a CPU does a LOAD or STORE on a page it does not have, a trap to the operating system occurs. The operating system then locates the page and asks the CPU currently holding it to unmap the page and send it over the interconnection network. When it arrives, the page is mapped in and the faulting instruction is restarted. In etfect. the operating system is just satisfying page faults from remote RAM instead of from local disk. To the user, the machine looks as if it has shared memory. The difference between actual shared memory and DSM is illustrated in Fig. 8-22. in Fig. 8-22(a), we see a

true multiprocessor with physical shared memory implemented by the hardware. In Fig. 8-22(b). we see DSM, imple- implemented by the operating system In Fig. 8-22(c), we see yet another (orni of shared memory, implemented by yet higher levels of software. We will come back to this third option later in the chapter, but for now we will concentrate on DSM. Let us now look in some detail at how DSM works, in a DSM system, the address space is divided up into pages, with the pages being spread over all the nodes m the system. When a CPU references an address that is not local, a trap occurs, and the DSM software fetches the page containing the address and restarts the faulting instruction, which now completes successfully. This concept is illus- illustrated in Fig. 8-23(a) for an address space with 16 pages and four nodes, each capable of holding four pages. In this example, if CPU 0 references instructions or data in pages 0, 2, 5, or 9. the references are done locally. References to other pages cause traps. For exam- example, a reference to an address in page 10 will cause a trap to the DSM software. Which then moves page 10 from node 1 to node 0, as shown in Fig. 8-23(b) Replication One improvement to the basic system that can improve performance consider- considerably is to replicate pages that are read only, for example, program text, read-only constants, or other read-only data structures. For example, if page 10 in Fig. 8-23

MULTICOMPUTERS Machine 1 Application Run-time system Operating Machine 2 Application syslem Operating Hardware Application system Operating syslem Hardware Application system Operating Hardware Application system Operating Hardware Run-time Operalmg Hardware (a] (b) (c) Figure 8-22. Various layers where shared memory con be implemented (a) The hardware (b) The operating system (c) User-level software is a section of program text. Us use by CPU 0 can result in a copy being sent to CPU 0, without the original in CPU I's memory being disturbed, as shown m Fig. 8-23(c). In this way. CPUs 0 and 1 can both reference page 10 as often as needed without causing traps to fetch missing memory. Another possibility is to replicate not only read-only pages, but also all pages. As long as reads are being done, there is effectively no difference between repli- replicating a read-only page and replicating a read-write page. However, if a repli- replicated page is suddenly modified, special action has io be taken to prevent having multiple, inconsisient copies in existence. How inconsistency ts prevented will be discussed m the following sections False Sharing DSM systems are similar

to mull [processors in certain key ways In boih s\s- teniK. when a nonlocal memory word is referenced, a ihunk of memory containing the word is fetched from its current location and put on the machine making thi reference (mam memory or cache, respectively). An important design is ue is how hig should the chunk be? In multiprocessors, the cache block size i u ually

MULTIPLE PROCRSSOR SYSTEMS CHAP 8 Globally shared virtual memory consisting of T6 pages 0SS 000 a 000 00 DID aaa SB cpuo 0130 C0 CPU 1 000 00 CPU 2 1 1 DID CPU 3 Figure 8-23. (a] Pages of the address space diMnbuied among four machines (b) Situation aftef CPU I reference--page 10 (c) Situation if page 10 is re.nl only and replication is mcd 32 or 64 bytes, lo avoid tymg up the bus with the transfer loo long. In DSM sys-systems, (he unit has to be a multiple of the page size (because the MMU works with pages), but it can be 1, 2.4. or more pages In effect, doing this sunulates a larger page size. There are advantages and disadvantages to a larger page size tor DSM The biggest advantage is that because the startup time for a network transfer is fairly substantial, it does not really take much longer to transfer 4096 bytes than it does to transfer 1024 bytes. By transferring daia in large units, when a large piece of

SEC 8 2 MLILTICOMPUTERS 543 address space has to be rnoved. the number ot transfers ma\ oiten be reduced This property is especially important because man\ program exhihit localit} of reference, meaning that if a program has iUereni.ed one word on a page it is likely lo reference other words on the same page in rhe immediate Juture On the other hand, the network will be tied up longer with a larger transfer, blocking other faults caused by other processes. Also, too large an effective page si?e introduces a new problem, called false sharing, illustrated in Fig. 8-24. Here we have a page containing two unreiaied shared variables, A and B. Processor I makes heavy use of A. reading and writing it Similarly, process 2 u«es B fre- frequently. Under these circumstances, the page containing both variables will con- constantly be traveling back and forth between the two machines \ Co va rB\ de using nable B _ AandB shared happen ,,,,, anab obs nrelaied es tha! on ihe s Figure S-24. RiKt sharing <i(.i page c<in!4ining iwn unrelated variables The problem here is thai although the variables are unrelated, they appear by accident on the same page, so when a process uses one of them. H also gets the other. The larger the effective page size, the more often false sharing will occur, and

conversely, the smaller the effective page size, the less often it will occur_ Nothing analogous to this phenomenon is present in ordinary virtual memory sys- systems. Clever compilers [hat understand the problem and place variables in fhe address space accordingly can help reduce false sharing and improve perfor- performance- However, saying this is easier than doing it. Furthermore, if the talse sharing consists of node I using one element of an array and node 2 using a dif- different element of the same array, there U little that even a clever compiler can do to eliminate the problem. Achieving Sequential Consistency If writable pages are not replicated, achieving consistency is not an issue. There is exactly one copy of each writable page, and it is moved back and forth dynamically a.s needed. Since it if. not always possible to see in advance which

pages are writable, in many DSM systems, when a process tries to read a remote page, a local copy is made and boih the local and remote copies are set up in their respective MMUs as read only. As long as all references are reads, everything is However, if any process attempts to write on a replicated page, a potential consistency problem anses because changing one copy and leaving the others alone is unacceptable. This siiualion is analogous to what happens in a multipro- multiprocessor when one CPU attempts to modify d word thai is present in multiple caches. The solution there is for the CPU about to do the write to first put a signal on lhe bus telling all other CPUs to discard their copy of the cache block. DSM systems typically work the same way Before a shared page can be written, a message is sent to all other CPUs holding a copy of the page telling them to unmap and discard the page. After all of them have replied that the unmap has finished, the original CPU can now do the write. It is also possible to tolerate multiple copies of writable pages under carefully restricted circumstances. One way is to allow a process to acquire a lock on a portion of the virtual address space, and then perform multiple read and write operations on the locked memory. At the time the lock is released, changes can be propagated to other copies. As long as only one CPU can lock a page at a given moment, this scheme preserves consistency Alternatively, when a potentially writable page is actually written for the iirst time, a clean copy is made and saved on the CPU doing the write. Locks on the page can then be acquired, the page updated, and the locks released Later, when a process on a remoic machine tries to acquire a lock on the page, the CPU that wrote it earlier compares the current state of the page to the clean copy and buildi a

message listing all the words that have changed. This list is then sent to rhe acquiring CPU so it can update its copy instead of invalidating it (Kelehcr et .il.. 1994). 8.2.6 Multicomputer Scheduling On a multiprocessor, all processes reside m the same memory. When a CPU finishes its current task it picks a process and runs it. In principle, all processes are potential candidates. On a multicomputer the situation is quite different. Each node has its own memory and its own set of processes CPU 1 cannot sud- suddenly decide to run a process located on node 4 without first doing a lair amount of work to go get it. This difference means that scheduling on multicomputer is easier but allocation of processes to nodes is niore important. Below we will study these issues. Multicomputer scheduling is somewhat similar to multiprocessor scheduling, but not a/I of the former's algorithms apply to the latter. The .simplest multipro- multiprocessor algorithm—maintaining a single central list of ready processes —does not work however, since each process can only run on the CPU it is currently located

SEC 8 2 MULTICOMPUTERS 545 on. However, when a irew process is created, a choice can be made where lo place it, tor example to balance the load. Given that each node has its own processes, any local scheduling algorithm can be used. However, it is also possible to use gang seheduhng, ihe same waj it is used on a multiprocessor, s^ce thai merely requires an initial agreement on which process to run in which time slot, and some way to coordinate the start ot the time slots. 8.2.7 Load Balancing There is relatively little to say about multicomputer scheduling because once a process has been assigned to a node, any local scheduling algorithm will do, unless gang scheduling k being used. However, precisely because there is so Ik- tie control once a process has been assigned to a node, the decision about which process should go on which node is important. This is in contiast to multiproces- multiprocessor systems, in which all processes live in the same memory and can be scheduled on any CPU at will. Consequently, it is worth looking ai how processes can be assigned to nodes in an effective way. The algorithms and heuristics for doing this assignment are known as processor allocation algorithms. A large number of processor (i.e., node) allocation algorithms have been pro- proposed over the years. They differ in what they assume is known and what the goal [S. Properties that might be known about a process include the CPU require- requirements, memory usage, and amount of communicat[On with every other process. Poss]ble goals include minimizing wasted CPU

cycles due io lack of local work, minimizing total communication bandwidth, and ensuring fairness to users and processes- Below we will examine a few algorithms to give an idea of what is possible. A Graph-Theoretic Deterministic Algorithm A widely-studied class of algorithms is for systems consisting of processes with known CPU and memory requirements, and a known matrix giving the aver- average amount of traffic between each pair of processes, if the number of processes is greater than the number of CPUs, k, several processes will have to be assigned to each CPU. The idea is to perform this assignment such as to minimize network traffic. The system can be represented as a weighted graph, with each vertex being a process and each arc representing the flow of messages between two processes. Mathematically, the problem then reduces io finding a way to partition (i e.. cut) the graph into k disjoint subgraphs, subject to certain constraints (e.g., toial CPU and memory requirements below some limits for each subgraph). For each solu- solution thai meets the constraints, arcs that are entirely within a single subgraph represent intramachine communication and can be ignored, Arcs that go from one

546 MUl.TlPLF PROCESSOR SYSTEMS ('HAP ! subgraph 10 another represent network traffic The goal is then to find the parti- partitioning that minimizes the network traffic while meeting all the constraints. As an example. Fig. 8-25 shows a system of nine processes. A through /_ W[th each arc labeled with Ibe mean communication load between those two processes (e.g , m Mbps) Figure In Fig. 8-25<a). we have partitioned the graph with processes A t, and 6 on node 1, processes B. F, ,ind H on node 2. ,ind processes- C />. and / on node 1 The tola! network traffic is the sum of the arcs intersected by the cuts (the dashed lines), or 30 units. In Fig. 8-25(b) we haie a different partitioning that has only 28 units of network traffic Assuming that it meets all the memory and CPU con- constraints this is a better choice because it requires less communication Intuitively. wh;it we are doing is looking tor clusters thai are tightly uoupled (high mtracluster traffic flow) but which interact little wlrh othei clusters (low intercjuster traffic fk>w) Some of the earliest papers' discussing the problem are (Chow and Abraham, 1982; Lo. 1984; and Stone and Bokhari. 1978). A Sender-Initiated Distributed Heuristic Algorithm Now let us look at some d[stnbuted algorithms. One algorithm says that when a process [S created, it rims on tbe node that created it unless that node is over- overloaded The metric tor overloaded might involve too many processes, too

big a lotal working set, or some other metric if it is overloaded, the node selects another node at random and asks it what its load is (using the same metric). It the probed node's load i.s below some threshold value, the new process is sent there (Eager et al_ 1986). If not, another machine is chosen tor probing Probing does not go on forever. If no suitable host is found within N probes, the algorithm ter- terminates and the process runs on the originating machine. The idea is lor heavily loaded nodes to try to get rid of excess work, as shown in Fig. 8-26(a).

MULT1C0MPUTERS figure 8-26. (a) Ad overload^ node looking for a lighilv le ofl processes to (bi An empty node Icmking for work to do Eager el a]. A986) rithm. Using this model, stable under a wide range of pa s, and probe limit i hld h tructed an analytical queueing model of this algo- algoas established that ihe algorithm behaves well and is f t luding different threshold values. transfer costs, and probe limits. Nevertheless, it should be observed that under conditions of heavy load, all machines will constantly send probes to other machines in a futile attempt to find one that is willing to accept more work. Few processes will be off loaded, but considerable overhead may be incurred m the attempt to do so. A Receiver-Initiated Distributed Heuristic Algorithm A complementary algorithm to the one given above, which is initiated by an overloaded sender, ts one initiated by an underloaded receiver, as shown m Fig. 8-26(b). With this algorithm, whenever a process finishes, the system checks to see if it has enough work. If not, it picks some machine at random and asks it for work. If that machine has nothing to offer, a second, and then a third machine is a^ked. If no work is found with ,V probes, the node temporarily stops asking, does any work it has queued up, and tries again when the next process finishes It no work is available, the machine goes idle After some fixed time interval, it begins probing again. An advantage of this, algorithm is that ji does not put extra load on the system at critical times- The sender-initiated algorithm makes large numbers of probes precisely when the system can least tolerate it—when it is heavily loaded With the receiver-initiated algorithm, when the system is heavily loaded, the chance of a machine having insufficient work is small. However, when this does happen, it will be easy to find work to take over. Of course, when there is little work to do.

MULTIPLE PROCESSOR SYSTEMS eates considerable probe traffic A Bidding Algorithm Another class of algorithms tries to turn the computer

system into a miniature economy, with buyers and sellers of services and prices set by supply and demand (Ferguson et al., 1988) The key players in the economy are the processes, which must buy CPU time to get their work done, and nodes, which auction their cycles off to ihe highest bidder. Each node advertises its approximate price by pulling it in a publicly readable file. This price is not guaranteed, but gives an incdcation ol what the service is worth (actually, it h the price that the last customer paid) Different nodes may have different prices, depending on their speed, memory size, presence of fast floal ing -point hardware, and other features. An indication of the service provided, such as expected response time, can also be published. When a process wants to start up a child process, n goes around and checks oul who is currently offering the service dial it needs. It then determines ihe set of nodes whose services it can afford From this set. it computes the best candidate, where "best" may mean cheapest, fastest, or best price/performance, depending on the application, [t then generates a bid and sends the bid to its first choice. The bid may be higher or lower than the advertised price Processors collect all the bids sent to them and make a choice, presumably by picking the highest one. The winners and losers are informed, and the winning process is executed. The published price ot [he server is then updated to reflect the new going rate. Although Ferguson et al, do not go into the details, such an economic model raises all kinds of interesting questions, among them the following. Where do processes get money to bid? Do they get regular salaries0 Doe^ everyone get [he same monthly salary, or do deans gel more than professors, who in turn get more than students? If new users are introduced into the system without a correspond- corresponding increase in resources, do prices get bid up (inflation)? Can nodes form cartels to gouge users? Are users' unions allowed? Is disk space also chargeable'1 How ibout printer output? Does printing pictures cost more than printing text due to the additional ink or toner used? The list goes on and on.

SE(8 1 DISTRIBUTED SYSTEMS 549 8.3 DISTRIBUTED SYSTEMS Having now completed our study of multiprocessors and multicomputer, it is lime to turn to the thud type of multiple processor system, the distributed sys- system. These systems die similar to multicomputer m that each node has its own private memory, with no shared physicd memory m the system. However, distri- T(, start with, the nodes of a multicomputer generally have a CPU, RAM, a network interface, and perhaps a hard disk lor paging In contrast, each

node in a distributed system is a complete computer with a full complement of petipheraU. Next, the nodes ot a multicomputei are normally in a single room so they can communicate by a dedicated high-speed network, whereas the nodes of a distri- buied system may he spread around the world Finally, all the nodes of a multi- multicomputer run the same operating system, share a single file system, and are under a common administration, whereas the nodes ot a distributed systems may run dif- different operating systems, each ha\e their own tile system, and be under ditfeient administrations A typical example of a multicomputer is 512 nodes in a single room at a company or university working on, nay. pharmaceutical modeling, whereas a typical distributed system consists of thousands of machines loosely cooperating over the Internet- Figure S-27 compares multiprocessors, multicom- multicomputer, and distributed systems on the points mentioned above Hem Multiprocessor , Multicomputer , Distributed System' Node configurator^ CPU_ CPU^ BAM, netjnterface Complete computer^ [Node penpherals_ A|l_shared Shared_exc maybe disk ' Full set per node ' j Location | Same_rack Same room __ [_ Possibly worldwide i inteinode communication Shared RAM Dedicated interconnect , Traditional network Operating systems : One shared Multiple same Possibly all different! File systems jjjne^sharad One^shared _ i Each node has own : Administration One organization One organization Many organizations Figure S-27. Gunpaiisoii „t ihrw I nidi of multiple CPl swum*. Multicomputer-, are clearly m the middle using these metrics. An interesting question is: "Are multicomputer more like muliipioccssors or more like distri- distributed systems?" Oddly enough, the answer depends strongly on your perspective From a technical perspective, multiprocessors have shared memory and the other !wo do not This difference leads to different programming models and different mind sets. However, from an applications perspective, multiprocessor!! and multi- computers are just big equipment racks in a machine room. Both arc used k>r solving computationally iniensive problems whereas a distributed system con- connecting computers all over (he Internet is. typically much more involved in com- communication than in computation and i^ used m a different way.

 both .1 strength and a weakness. It is d strength because ihe computers can be used lor a wide variety ot applications- but it is also a weakness, because pto- uramming ihese applications is difficult due io the lack ot any common uuderly- lypical Internet

application-, includL- access io remote computers (Using teliwl and riojtw), access to remote inclination (umii^ the World Wide Web and FTP, ihe File Transfer Protocol), persou-to-peison communication (using email and thai programs), and many emerging applications (e.g . e-commerce, tclcmedicme, and disiauce learning) The irouble With all these applications is that each one has tn reinvem ihe wheel For example, email. FTP. and Ihe World Wide Web all basically move files Irom point A to point B_ but each one has Us own way of doing it. complete with its own naming contentions, tranter protocols, replication techniques, and everything eKe Although many Web browsers Inde these differ- differences fioni the average user, the underlying mechanisms are completely different. Hiding them at the user interface level is like having a peison at a full-service travel agent Web site Order d trip from New York to San Francisco, and only later discover whether she has purchased a plane, train, or bus ticket. What distributed systems add to the underlying network i-, some common paradigm (model) that provides a uniform way of looking at the whole system. The intent ot the distributed system i.v to turn a loosely-connected bunch ot machines into a coherent system based on (Tne concept. Sometimes the paradigm is simple and sometimes it is more elaborate, but the idea is always to provide something that unities the system. A simple example of a unifying paradigm in a Jightl> different context is found in UNIX, where all I/O devices are made to look like files. Having key- keyboards, printers, and serial lines all be operated on the same w;iy. with the same primitives, makes it easier to deal with (hem than having them all be conceptually different- One way a distributed system can achieve somc measure of untfurmity in the tace of different underlying hardware and operating systems is to have a layer of software on top of the operating system The layer, called middleware, is illus- illustrated in Fig. 8-28. This layer provides certain data structures and operations that allow processes and users on far-flung machine^ to intemperate m a consistent In a sense, middleware i^ like Ihe operating system of a distributed system. That is why it is being discussed in a hook on operating systems On the other hand, it is not an operating system, so the discussion w il] not go into much detail For a comprehensive, book-length treatment of distributed systems, see Distri- Distributed System* (Tanenbaum and van Steen. 2002), In the remainder of this chap- chapter, wu will first look quickly at the hardware used in a distributed system (i.e.. the underlying computer network), then its communication software (the network pro-protocols). After that we will consider a variety of paradigms used m these

systems.

DISTRIBUTED SYSTEMS Application WlMOw, Pentium / Application Middleware «,,,, «,«,,»,,, So,™ SPARC Applicalion Middleware Mac OS Macintosh 1 Figure K-28. Positioning o+ mnidIcsfeme in j disiiibuied system 8.3.1 Network Hardware Distributed systems are budi on top ot computer networks, so it onef introduc- introduction to the subject I*, in order. Networks come m two major tanetics. LANs (Local Area Networks), which cover a building or a campus, and WANs (Wide Area Networks), lhat can be citywide, countrywide, or even worldwide. The most important kind ot LAN is Ethernet, so we will examme ih;it as an example LAN. As our example WAN, we will look at the internet, even though lechni- networks. However, tor our purposes, a is sufficient to think of tt .is one WAN Ethernet Classic Ethernet, which is described in IEEE Standard 802 ^5, consists of a coaxial cable to which a number of computers are <)ttached. The cable is called the Ethernet, in reference to the lumimfetaus ether, ihrnu^h which eleetrom.ig- netic radiation was once thought to propagate (When the nine tee nth-century British physicist James Clerk Maxwell discovered that electromagnetic radiation could be described by a wave equation, scientists assumed lhat space must be filled with some ethereal medium in which the r.idiation was propagating Only after the famous Michel son- Morley experiment m 1887. whi^h tailed" to detect the ether, did physicists, reulue that radialiou could propagsile in a vacuum.) literally drilling a hole halfway through the cable and screwing in u wire leading to the computer. This was. called a vampire tap. and is shown symbolically in Fig. 8-29(a). The taps were bard to get right, so before long, proper connector-;

MLLflPLE PROCESSOR SYSTEMS were used Nevertheless, electrically, all the computers wer< cables on their network interface cards weic soldered together "\ To send a packet on an Ethernet, a computer tirst listens to the cable to see it any other computer is currently transmuting If not, it jusi begins transmitting a packet, which consists of a short header followed by a ()- to 15U0-bytc pay load If the cable is in use. the computer simply waits until the current transmission fin- finishes, then it begins sending If two computers start transmitting simultaneously, a collision results, which both 01 them detect. Both respond by terminating their transmission^, waiting a random amount of time between 0 and T jjscc and (hen starting again. If another collision occurs,

all colliding computers randomize the wait into the interval 0 to 2T\\\tc. and then try again. On each further collision, the maximum wait interval is doubled, reducing the chance of more collisions. This algorithm i-. called bin- binary exponential backoff. We saw it earlier to reduce polling overhead on locks. An Ethernet has a maximum cable length and also a maximum number of computers that can be connected to it. To exceed cither of these limits. ,i large building or campus can be wired with multiple Ethernets, which are then con-connected by devices called bridges. A bridge allows traffic 10 pass from one Ether- Ethernet to another when the source is on one side and the destination is on the other To avoid the problem of collisions, modern Ethernets use switches, as shoivn in Fig. 8-29(b). Each switch has some number of ports, to which can be attached a computer, an Ethernet, or another switch. When a packet successfully avoids .ill collisions and makes it to the switch, n is buffered thtie and sent out on |he port where the destination machine lives. By giving each computer its own port, all collisions can be eliminated, at the cost of biggei switches Compromises, with just a few computers per port are also possible. The Internet The Internet evolved from the ARPANET, an experimental packet-swrtchud network tunded by the U.S Dept. of Defense Advanced Research Projects Agency, It went live in Dec. 1969 with three computers in California and one in

SEC 8 3 DISTRIBUTED SYSTEMS 553 Utah. |t was designed to a be a highly fault-tolerant network thai would continue to relay military traffic even in the event of direct nuclear hits on multiple parti ot the network by automatically rerouting traffic around the dead machines. The ARPANET grew rapidly [n the 1970s, eventually encompassing hundreds of computers. Then a packet radio network, a satellite network, and eventually thousands of Ethernets were attached to it, leading to the federation of networks we now know as the Internet. The Internet consists of two kinds of computers, hosts and routers. Hosts are PCs, laptops, palmtops, servers, mainframes, and other computers owned by indi viduals or companies that want to connect to the Internet. Routers are specialized switching computers that accept incoming packets on one of many incoming line and send them on their way along One of many outgoing lines. A router is similar to the switch of Fig. 8-29fb), but also differs from it in ways that will not concern us here. Routers are connected together in large networks, with each router hav- having wires or fibers to many other routers and hosts Large national or worldwide router networks are operated by

telephone companies and ISPs (Internet Service Providers) for their customers. Figure 8-30 shows a portion of the Internet At the top we have one of the backbones, normally operated by a backbone operator It consists of a number of routers connected by high-bandwidth fiber optics, with connections to backbones operated by other (competing) telephone companies Usually, no hosts connect directly to the backbone, other man maintenance and test machines run by the telephone company. Attached to the backbone routers by medium-speed fiber optic connections are regional networks and routers at ISPs, In turn, corporate Ethernets each have a router on them and these are connected to regional network routers. Routers at ISPs are connected to modem banks used by the ISP's customers. In this way, every host on the Internet has at least one path, and often many paihs. to every other host. All traffic on the Internet is sent in the form of packets. Each packet carries its destination address inside it. and this address is used for routing. When a packet comes into a router, the router extracts the destination address and looks (part of) it up in a table to find which outgoing Ime to send the packet on and thus to which router. This procedure is repeated until the packet reaches the destina- destination host_ The routing tables are highly dynamic and are updated continuously as routers and links go down and come back up and as traffic conditions change. 8.3.2 Network Services and Protocols AH computer networks provide certain services, to their users (hosts and processes), which they implement using certain rules about legal message ex- exchanges. Below we will give a brief introduction to these topics.

554 MULTIPLE PROCESSOR SYSTEMS CHAP. 8 d -<DDnoD- Figure 8-30. A portion cl ihe Internet. Network Services Computer networks provide services to the hosts and processes using them. Connection-oriented service is modeled after the telephone system. To talk to someone, you pick up the phone, dial the number, talk, and then hang up. Simi- Similarly, to use a connection-oriented network service, the service user first estab- establishes a connection, uses the connection, and then releases the connection. The essential aspect of a connection is that it acts like a tube: the sender pushes objects (bits) in at one end, and the receiver takes them out in the same order at the other end. In contrast, connectionless service is modeled after the postal system. Each message (letter) carries the full destination address, and each one is routed through the system independent of all the others. Normally, when two messages are sent to the same destination, the fitst one sent will be the first one to arrive.

However, it is possible that the first one sent can be delayed so that the second one arrives first. With a connection-oriented service this is impossible. Each service can be characterized by a quality of service. Some services are reliable in the sense that they never lose data. Usually, a reliable service is imple- implemented by having the receiver confirm the receipt of each message by sending back a special acknowledgement packet so the sender is sure that it arrived The acknowledgement process introduces overhead and delays, which are necessary to detect packet loss, but which do slow things down.

ShC K 1 DISTRIBUTFD SYSTEMS 55S A typical situation in which a reliable connection-oriented service is .ippropri- aie is file transfer. The owner of the file wants to be suie that all the bits arrive correctly and m the same order they were sent. Very few file transfer customers would prefer a service that occasionally scrambles or loses a few bits., even it it is much taster Reliable connectlon-onemed service has two minor variations message sequences and byte streams. In the rormer. the message boundaries are preserved When two 1-KB message), are sent, they arrive as two distinct 1-KB messages, never as one 2-KB message In the latter, the connection is simply a stream of bytes, with no message boundaries,. When 2K byles arrive at the receiver, there is no way to tell if they were seni as one 2-KB message, two I-KB messages, or 2048 l-byie messages It ihe pages ot a book arc sen! over a network to an image setter as separate messages, it rnighl be important to preserve the message boundaries On the other hand, wuh a lerminal logging into a remote timesharing system, a byte stream Irom the terminal to the computer is all that is needed. For some applications, the delays introduced by acknowledgements are unac- unacceptable. One such application is digitized voice traffic. It is preferable for tele- telephone users to hear a bit of noise on the line 01 a garbled word from time 10 time than to introduce a delay to wait for acknowledgements. Not all applications require connections. For example, to test the network, all that is needed is a way to send a single packet (hat has a high probability ot arrival, bui no guarantee. Unreliable (meaning not acknowledged) connectionless service is often called datagram service, m analogy with telegram service, which also does not provide an acknowledgement back to the sender. In other situations, the convenience of not having to establish a connection to send one short message is desired, but reliability is essential The acknowledged datagram service can be provided

for these applications- It is like sending n registered letter and requesting a return receipt. When the receipt come;; back, the sender is absolutely sure that the letter was delivered to the intended party and not lost along the way. Still another service is the request-re ply service. In this service the sender transmiis a single daiagram containing a request; the reply contains the answer For example, a query to the local library asking where Uighur is spoken falls into this category. Request-reply is commonly used to implement communication in the client-server model' the client issues a requesi and the server responds to it Figure 8-31 summarizes the types of services discussed above. Network Protocols All networks have highly-specialized rules for what messages may be sent and what responses may be relumed m response to these messages. For example, under certain circumstances (e.g.. file transfer), when a message is sent from a source to a destination, the destination is required to send an acknowledgement

MULTIPLE PROCESSOR SYSTEMS Service Reliable message stream Reliable byte si ream Unreliable connection Unreliable datagram Acknowledged datagram Request-re ply E Sequence Remote log Digitized vo Network las fleg,siered Databeseq ample f pages of a book ce I packets mail ery FiBUr back indicating correct receipt of the message. Under other circumstances (e.g., digital telephony), no such acknowledgement is expected. The set of rules by which particular computers communicate is called a protocol Many protocols exist, including router-router protocols, host-host protocols, and others. For a thorough treatmenl of computer networks and their protocols see Computer Net- Networks (Tanenbaum. 1996). All modern networks use what is called a protocol Stack to layer different protocols on top of one another At each layer, different issues are dealt with. For example, at the botlom level protocols define how tt, tell where m the bit stream a packet begins and ends. At a higher level, protocols deal with how to route pack- packets through complex networks from source to destination. And at a still higher level, they make sure mat all the packets m a multipacket message have arrived correctly and in the proper order. Since most distributed systems use the Internet as a base. the key protocols these systems use are the two major Internet protocols IP and TCP IP (Internet Protocol) is a datagram protocol in which a sender injects a datagram of up to 64 KB into the network and hopes that it arrives No guarantees ure glven. The data- datagram may be fragmented into smaller packets as it passes through the Internet These packets travel

independently, possibly along different routes. When all the pieces get to the destination, they are assembled in the correct order and delivered. Two versions of IP are currenily in use. v4 and v6. At itie moment. v4 still dominates, so we will describe thai here, but v6 is up and coming. Each v4 packet starts with a 40-byte header that contains a 32-bit source address and a 32-bit des- destination address among other fields. These are called IP addresses and form the basis of routing in the Internet They are conventionally written as four decimal numbers id the range 0-255 separated by dots, aR in 192.31 231,65 When a packet arrives at router, the router extracts the IP destination addres for routing the packet. Since IP datagrams are noi acknowledged. IP alone is not sufficient for reli- reliable communication in the Internet. To provide reliable communication, another s that

SEC 8.3 DISTRIBUTED SYSTEMS 557 protocol, TCP (Transmission Control Protocol), is usually layered on lop of IP. TCP uses IP to provide connection-oriented streams. To use TCP, a process first establishes a connection to a remote process. The process required is specified by the IP address of a machine and a port number on that machine, to which proc- processes interested in receiving incoming connections listen. Once that has been done, it just pumps bytes into the connection and they are guaranteed to come out the other end undamaged and in the correct order. The TCP implementation achieves this guarantee by using sequence numbers, checksums, and retransmis- retransmissions of incorrectly received packets All of this is transparent to the sending and receiving processes. They just see reliable interprocess Communication, jusi like a UNIX pipe . To see how all these protocols interact, consider the simplest case of a very small message that does not need to be fragmented at any level. The host is on an Ethernet connected to the Internei. What happens exactly? The user process gen-generates the message and makes a system call [o send it on a previously established TCP connection. The kernel protocol stack adds a TCP header and then an IP header to the front. Then it goes to the Ethernet driver, which adds an Ethernet header directing the packet to the router on the Ethernet. This, router then injects the packet into the Internet, as depicted in Fig. 8-32. Figure 8-32. Accumulation of packet headers To establish a connection with a remote host (or even to send it a datagram), n is necessary to know its IP address. Since managing lists of 32-bit IP addresses is inconvenient for people, a scheme called DNS (Domain Name System) was invented as a database that maps

ASCII names for hosts onto their IP addresses. Thus it is possible to use the DNS name star.cx.vu.nl instead of the corresponding IP address 130.37.24.6. DNS names are widely known because Internet email addresses are of the form user-name@DNS-host-name, This naming system al- allows the mail program on the sending host to look up the destination host's IP

S5S Mill TIP! r- PROCESSOR SVSTI \th <- HAP S address m ihe DNS daiabase establish d HT connection to ihc mail daemon pio^css iherc. ami send the message as a tile. The ,M,-r-n«»K' is seni alone id dieniitv whiUi mailbox tu pui tlrc message in 8.3.3 Document-Based Middleware Now thai we liave soine background on nulworks and protocols, we can Man looking at different middleware lasers that L,m tnertaj the basic network lo pro- produce ;i consistent paradigm for applications aiid i^ers We will stari wnh a sim- simple, but well-known example the World 'Aide %eb. The 'Acb was iinentcd by Inn Bcrneis-t.cu ai CbRN. the Emopean Nuclear Physics Research Cenler. in 19H9 and has spread like wLMliie all over the world since then i he orLCiii*il piir^dicni beliind the ^Vt^b wits v'^rv simple e\ eiy cc^mpuier can hold one or more documents, called Web pages Laeh Web page contains text, images, icons, sounds, movies, etc. as well as hyperlinks (pointers) to other Web pages When a user requests a Web page usinn a program called a Web browser. the page is displayed on ihe screen. Clicking o» a link causes ihe current page to be replaced on the screen by the page pointed to. Although many bells and whis- whistles have been grafted onto the Web recently, the underlying paradigm is Mill clearly present: the Web is a great big directed graph of documents that «m point to other documents, as shown in Fig. Ji-33. Figure 8-33. The Web is j big dircued ynipli 01 ilixumonts Each Web page has a unique address, called a URL (Uniform Resource Locator), of the farmprotocol://DNS-name/f]le~mime. The protocol is most com- commonly hup (HyperText Transfer Protocol), but Jip and others also exist. Then comes the DNS name of the host containing the file. Finally, there is a local file name telling which file is needed. The way the whole system hangs together is as follows. The Web u funda- fundamentally a client-server system, with the user being the client and the Web situ

Sbt s 1 DISTRIBUTED SYS [["MS 559 beiim the wrvei When the user provides the hrt>» ser with <i URL.eitha by typ- iiw It in or clicking on a hyperlink on the current page, the bro^ei takes ceitam step*. t» tetch ihe

recycled Web page M .»n example, suppose the URL pro- vided is lutp-//*»H«rmr>rtf/'Mtuihtm! The- browser then takes the following sicps lo jjt-t Ihe page 2 IJNS icpiics W[[h 199 222.69 15l. 3 The hrowsei makes a TCP connection to port W on 1 W.222 69 15 1. 4 It ihen studs j request asking f()i the tile til/Jaq html 5 The w-H iv.ai. in oru servei sends the file dl/faq html. 6 The TCP connection is released 7 The browser displays all the text in itl/faq html. 8- The browser fetches and displays all imiiges m dl/faq.iiiml To a fiw approx[mation. thai a, the baSIs ot the Web and how it Works Many other features have since been added to the basic Web, including style sheets, dynamic Web pages that are generated on-the-fly. Web pages that coniain small programs or scripts that execute on the client machine, jnd more, but they arc out- outside the scope of this discussion 8.3.4 File System-Based Middleware The basic idea behind the Web is tn make a distributed system look l[ke a giant collection of hypcrlinked documents. A second approach is to make a distri- distributed system look like a great big file system In this section we will look at some of the issues involved in designing a worldwide file system. Using a file system model for a distributed system means that there [h a single global file system, with users all over the worlci able to read and write files for which they have authorization. Communication i> achieved by having one proc- process write data into a file and having other ones read them back. Many ot the stan- standard file system issues arise here, but also some new ones related to distribution. Transfer Model The first issue is the choice between the upload/download model and the re- remote access model. In the former, shown in Fig 8-34(a), a process accesses a file by first copying it from the remote server where it lives If the tile is only lo be read, the tile is then read locally, for high performance. If the tile is to be writ- written, it is written locally When the process (s done with it, the updated file is put

 remoie access model, the file stays on ihe server and the client ^ends commands there to gel work done there, as shown in Fig. 8-34(bi. Figure «-34. (j) The upload/download model (b) The 1 The advantages of the upload/download mode] are us simplicity, and the fact thai transferring entire files at once is more efficient than transferring them in small pieces. The disadvantages are that there must be enough storage for tlie entire file locally, moving the entire tile is wasteful if only parts of it are needed, and consistency problem1; arise if there are multiple concurrent users. The Directory

Hierarchy Files are only part of the story The other pirt is the directory system All distributed file systems support directories torn i rung multiple tile The next design issue is whether all client have the sum. view of the directory hieiarchy, As an example of what we mean by this remark consider FIt, S 15 In Fig. g- 35(a) we show two file servers e th holding three directories nd ome tiles. In Fig. 8-35ft>) we have a system in which all clients (nd other machine > have the same view of the distributed hie system It the path /D'E/x i v lid m one machine, it is valid on al] of them In contrast, in Fig. 8-35(t| different machine can have different Mew of the file system. To repeat the preceding example the path 'D/Ef\ mi^ht well be valid on client 1 but not on client 2 In sy terns that manage multiple file servers by remote mounting. Fig. 8-35|t) n the norm It is flexible and straightforward to implement, but it has the di advantage ot not making the tntire s\ tern behave like a single old-fashioned timesharing svstem In a time hanne system the file system looks the same to any proce [i e the model r F,g S 3S|b)] Th(s pro- property makes a system easier to program and under land A closely related question i whether r not there is a global root directory, which all machines recognize as the root. One way to have a global root directory is to have the root contain one entry for each server and nothing else. Under these

DISTRIBUTED SYSTEMS Figure 8-35. (a) Two file servers The squares are directories are files (b) A sysiem in which all clients have the saine viev. Km. (c) A system i.n which different clLents may have different v circumstances, paths take the form /server/path, which has its own disadvantages, but at least is the same everywhere in the system. Naming Transparency The pnncipal problem with this form of naming is thai it is not fully trans- transparent. Two forms of transparency are relevant in this context and are worth dis- distinguishing. The firsi one, location transparency, means that the path name gives no hint as to where the file is located. A path like /serverl/dirl/dir2/x tells everyone that x is located on server 1. but it does not tell where thai server is located. The server is free to move anywhere it wants to in the network without the path name having 10 be changed. Thus this system has location transparency.

562 Mtii lii'l-f PW>(r^oK si si i \ks (n\h Houevei. suppose thjt 1tle j i- extremely laige and spme is tight 01, sciwm I Furthcimorc suppose that there is plenty ,>l room on send 2. The v,stan mtghl well like to move * to scivei 2

aulomancdllj UnfortunaWl>. when the first com- component ol all path name is the servci. the s> stem cannot move the file io the other seivcr automatically evoti it tin i and titr2 cms! on both «.erveis The problem is rh.it movmg the file automatically ihanires it-, natD name from Acn ,-' lhnr)hh>V\ cease to woik if ihe patli changes A sjsieni m which file- can lie moved without ihetr names changing is said to have location independence. A distributed svs- indc]>endcnt One based on remote mounting is not Either, since it is not possible to move a file from one file group (the unit of mourning) to another and still be able io use (he old paih name. Location independence is not cosy to achieve, hut ii is a desirable property to have in a distributed system To summarise what we ha\e said earlier, there are three common j.pproaches to file and directory naming in a distributed system. 1 Machine + path naming, such as /marhme/poth or uuuliine-patli 2. Mounting remote file systems onto the local file hierarchy 3. A single name space that looks the saine on all machines. The hrst two ate ea^y to implement, especially as a wav to connect existing svs- tems that were not designed for distributed use The latter is difficult and requires careful design, but mak^s life easier for programmers and users Semantics of Pile Sharing When two or more users snare the same file, il is neLessary to define the semantics of reading and writing precisely to avoid problems In single-processor systems the semantics normally state thtit when a read system call follows a wnle system call, the read returns the value just written, as *hown in F-ig. 8-36(a) Similarly, when two writes happen in quick succession, followed by a read, the value read is the value stored by the last wnte [n effect, the system enforces an ordering on all system calls and all processors see the saint- ordering. We will refer to this model as sequential consistency In a distributed system, sequential consistency can be achieved wisily as long as there is only one file server and clients do not cache files A|l reads and writes go directly to the file server, which processes them strictly sequentially In practice, however, the performance ot a distributed system in which all file requests must go to a single server is frequently poor. This problem is often solved by allowing clients to maintain local copies of Iieavily used files in their private caches. However, if client 1 modifies a cached file locally and shortly thereafter client 2 reads the file from the server, the second client will get an obsolete file, as illustrated in Fig. 8-36(b).

DIS1R1BI It D ^ Sit MS One way out of this difficulty is (o propagate all changes to cached tiles back to the server immediately. Although conceptually

simple, this approach is ineffi- inefficient. An alternative solution is to relax the semantics of file sharing Instead oi requiring a read to see the effects of all previous writes, one can have a new rule that says: "Changes to an open file are initially visible only to the process that made them. Only when the file is closed are the changes visible to other processes " The adoption of such a rule does not change what happens in Fig 8- 36(b), bui it does redefine the actual behavior (B getting the original value of the tile) as being the correct one. When client 1 closes the (lie, it sends a copy back to the server, so that subsequent reads get the new value, as required. Effectively, this is the upload/download model of Fig 8-34. Tins semantic.1 rule is widely implemented and is known as session semantics. Using session semantics raises the question ot what happens it two or more clients are simultaneously caching and modifying the same lile. One solution is to say that as each file is closed in turn, its value is sent back to the server, so the

564 MULTIPLE PROCESSOR SYSTEMS CHAP 8 final resull depends on who closes Ut A less pleasant, but slightly eas.er to implement, alternative is to say that the final result is one of the candidates, hut leave the choice of whkh one unspecified An alternative appmach to session semantics is to use the upload/download model, but to automatically lock a file that has been downloaded Attempts b\ other clients to download the file will he held up until the first client has ictumed it If there 11 a heavy demand for a file, the server could send messages to the client holding the file, asking it to hum up, but that ma\ or may not help. Ail in all, getting the semantics of shared filet right is a tnck> business with no elegant and efficient solutions. AFS Several file-system based middleware systems have been built and deployed. Below we will briefly discuss one (AFS) based on the upload/download model »t Fig. 8-34(a). In Chap. 10, we will discuss one (NFS) based on the remote access model of Fig. 8-34(b). AFS was designed and implemented at Carnegie-Mellon University (Howard etal. 1988; Morris et al. 1986, and Satyanarayanan ct a].. 1985). It was oiiginally called the Andrew File System m honor of the university's first benefactors. Andrew Carnegie and Andrew Mellon. The goal of the proiect. which started in the early 1980s, was to provide every student and faculty member at CMU with a powerful persona] workstation running UNIX, but with a shared file system. Here the file system was being used ai middleware to turn a collection of workstations into a coherent sysiem. Each AFS user has a private workstation running a slightly modified version of

UNIX. The modifications consist of adding a p[ece of code called venus to the keme] and running a file server called vice m user space (originally venus also ran in user space, but was later moved into the kernel for performance reasons). The positions of venus and vice are shown m Fig. S-37(a). User workstations are grouped into cells for administrative purposes. A cell might be a LAN or a col- collection of inierconnected LANs or even an entire academic department The name space V[sible to user programs looks like a traditional UNIX tree, with the addition of a directories /emu and /cache, as depicted in Fig. 8-37(b). The /cache directory contains cached remote files The /emu directory contains the names of the shared remote cells, below which are (heir respective file sys- systems, in effect, remote file systems are mounted in Amu. The other directories and files are strictly local and are not shared Symbolic links from local file names to shared files are permitted, as, indicated by sh m Fig. 8-37fb). The basic idea behind AFS is for each user to do as, much as possible locally and interact as little as possible with [he rest or" the system When a fjjc is opened, the venus code traps the open call and downloads the entire file (or if it is a huge file, a large chunk of it) to the local disk and inserts it into (he /cache directory.

Figure 8-37. (a) The The file descriptor returned by the open call reters to the file in /cache w that subsequent read and write calls use the cached file. The semantics offered by AFS are close to session semantics. When a file is opened, it rs fetched from the appropriate server and placed in /cache on the workstation's local disk. All reads and writes operate on the cached copy When the file is closed, it is uploaded back to the server. However, to prevent unsuspecting processes from using stale files in situa- situations where it matters, when venus downloads :i file into its cache, it tells vice whether or noi it cares about subsequent opens by processes on other worksta- workstations. If it does, vice records the location of the cached file. If another process, elsewhere in the system opens the file, vice sends a message to venus telling it to mark its cache entry as invalid and return the copy if it has been modified 8.3.5 Shared Object-Based Middleware Now lei us take a look at a third paradigm. Instead of say[iig that everything is a document or everything is a file, ne say that everything [S an object. An abject is a collection of variables that are bundled together with a set of access procedures, called methods. Processes are not permitted to access [he variables directly. Instead, they are required to invoke the methods CORBA Some programming languages, such as

C++ and Java, are object oriented, but these are language-level objects rather than run-time objects. One well-known system based on run-time objects is CORBA (Common Object Request Broker

566 Ml I TlPLH PROChSSUR S1SI1MS I MAI' s \rchit«liirO<VmosU 1007) CORBA .s a dier,™,-, s>s<em ,n >ih,J. Hie H processes on client machines cm invoke (.peiatioiK on ob|eUs located en (posM- ummnsi ;i variety of hardware platforms and optraimg spurns and pn,gr,i«irtwtl in a xaiiety of languages. To make u possible. |o, n client wi one platform 10 invoke a server orwi different platioim ORBs (Object Request Brokers) jic interposed between cheni and servei to dlli'w them to match up [he ORBs play ,ui impoiunt role m CORBA. e\en proMi'iiiy the sworn wLrli us nanio haeh CORBA object is defined hy an nmMfjce cklinition in a language calUd IDL (Interface Definition I,anguaa*-M v-hii-h telK what mc-thods the object exp,,its and what par.miclei types each out- expeus TJit- IDL speaficimon tan he compiled tnu, d client stub pn.ccduic and stored m j librae II .i cUcm jiioct^s knows in advance that it will need to access a certain ob|ect_ it is linked w itli the object's chent stub cede The IDL specilKarion can also he compiled niU. a skeleton procedure that is used on the server side If it is ndt knov^n in advame which CORBA objects a process needs to ui-e_ djnannc invocation js also pos-i- ble, but how that works is beyond the scope or our treatment When a CORBA object is created, a reference to it ts also created and returned to the creating process. Tim relerence ls how the process identities the object for subsequent imocations ol its methods The reference can be passed to other processes or stored in an object directory To invoke a method on an object, a elieni process must first acqune a refer- reference to the object. The reference can either come dtrecily from the creating proc- process, or more likely, by looking it up by name or by function m some kind of a directory. Once the object reference is available, the client process marshals |he parameters to the method calls into a convenient structure and then contacts [he client ORB. In turn, the client ORB send.s a message to (he server ORB. which actually invokes the method on the object. The whole mechanism is similar to RPC. The function of the ORBs is to hide all iho low-level distribution and com- communication details from the client and server code. In parncular_ the ORBs hide from the client the location of the server, whether the server is a biliary piogram or a script, what hardware and operating system the server iuiis on. whether the object It. currently active, and how the two

ORBs conimuincaie (e g . TCP/IF. RPC, shared memory, etc). In the first version ot CORBA, ihe protocol between the client ORB and the server ORB was not specified. As a result, every ORB vendoi used a different protocol and no two ot them could talk to each other In version 2.0. the protocol was specified. For communication over the Internet, the protocol ■* called HOP (Internet InterOrb Protocol). To make it possible to use objects that wcie not written ror CORBA with CORBA systems, every object can be equipped with an object adapter This is a wrapper that handles chores such as registering the object, generating object ref-

SIC M« DIM RLBI TLD SYSTEMS 5<>7 erences, and acdvaring the object ii it 1- invoked when it is nt)t acnve. The arrangement of all these CORBA parts is shown in Fig. 8-38. CORBA parts are shown in gray. A serious problem wnh CORBA is that every object is located on only one server, which means the performance will be terrible tor objects that are heavily used on client machines around the world In practice. CORBA only functions acceptably in small-scale systems, such as to connect processes on one computer, one LAN or within a single company. Globe As an example of a disinbuted object system thai was specifically designed 10 scale to a billion users and a trillion objects around the world, let us consider Globe (Van Steen et al., 1999a; Van Sieen ct al, 1999b). There art; iwo key ideas to scaling to very large systems. The First is having replicated objects, if there U a single copy of a popular object that millions of users around the world want to access, the object will die under the weight of the requests. Think about an object that maintains stock prices or sports scores Replicating this object allows [he load to be spread over the replicas. The second key idea is flexibility. In a worldwide system wnh a billion users, there is no way to get everyone to agree on one programming language, one repli- replication strategy, one security model, or one anything else. The system has to allow different users and different objects to behave differently, while at the same time providing a coherent overall model. Th[S is what Globe does. Globe is also unusual in that, like DSM. it is hased on (he model of distnbmed shared memory, but now applied to a worldwide context. In principle, using nor- normal page-based DSM on a worldwide system would work, only the performance

568 MULTIPLE PROCESSOR SYSTEMS CHAP 8 would be horrible, so Globe takes a different approach. Conceptually, the basic idea is thai the world

is full of objects, each one containing some (hidden) internal state plus methods for access[Hg the internal state in controlled ways. The secret to making shared memory scalable worldwide is prohibiting direct LOADS and STORES to an object's internal state and forcing all accesses to go through the methods. Because a Globe object can actively be shared by many processes at the same time, it is also called a distributed shared object. The positioning of sys- systems like Globe is shown m Fig. 8-22(c). Now let us see how scalability and flexibility are implemented. Every Globe object has a class object that contains the actual code for its methods. Every object also has one (or more) interfaces, each of which contains (method pointer, state pointer) pairs. Thus given an object interface, which is a table full of pointers present in memory at run time, a process can invoke the object's n-ih method by making a call to the procedure pointed to by (he 'r-th pair m the inter- interface table and passing it ihe corresponding stale pointer a.s a parameter The state pointer is needed so (hat if there are. say, two objects of class mailbox in memory, each one has its own interface, with shared method pointers but private state pointers, as shown in Fig. 8-39. In this example, the process has two open mail- mailboxes, each of which shares the code for the tour mailbox methods, but each <>f which has its own private state (the messages stored m the mailbox instance). One mailbox might be for business mail and the other for personal mail tor exam- example. Addi Figure JKW. The structure ot a Globe object The design of having interfaces be tables m i jects are not restricted to any particular langi ..o because a worldwide system will have many different people . ___,, . .__. _.nory ai run time means that objects are not restricted to any particular language. This decision was made any different

SEC. 8.3 DISTRIBUTED SYSTEMS 569 favorite languages. An object's methods may be written in C, C++, Java, or even assembly language if the object's owner so desires. The interfaces are there to shield the process from what is behind the method pointers. This mix-and-match design is more flexible than a single-language design present in some systems (e.g., only Java or only C++). To use a Globe object, a process must first bind to it by looking it up and find- finding at least one contact address (e.g., IP address and port). A security check is made at binding time, and if the process is authorized to bind to the object, the object's class object (i.e., its code) is loaded into the caller's address space, a copy of its state is instantiated and a pointer to its (standard) interface is returned. Using the interface pointer, the process can now invoke

methods on this instance of the object. Depending on the object, the state may be the default state or a copy of the current state taken from one of the other live copies. Imagine the simplest possible object. It has one integer as state and two methods: read and write that operate on the integer. If multiple processes in dif- different countries are simultaneously bound to the object, all of them have an inter- interface table pointing to the class object containing the two methods (which was loaded at bind time), as illustrated in Fig. 8-40. Each process (potentially) also has a copy of the integer comprising the state. Any read method is just invoked locally, but writes are more complicated. If the object wants to maintain sequen- sequential consistency, it must provide a mechanism for doing so. isr r figure 8-40. A distributed shared object can have its siate copied on multiple computers et once. One mechanism is to have a process called the sequencer whose job is to issue consecutive sequence numbers when requested to do so. To do a write, the write method might then first get a sequence number and then multicast a message

570 \HJ[TIP[X PROCFSSOK SYMhMS (HAL' 8 containing the sequence number, opcraiion rume, and paiaiiieter ti, all ihe oilier the> would be ass[gned different consecutive sequence numbers. All processes must apply incoming methods in sequence number order, not in message arrival order It a process gets sequence number 26 and the previous one was 24, it must wan for 25 before applying 26 If 25 does not show up within a curtain lime, ihe process njust take action to locate and get u Tins scnemC guaraniees trial a 11 write;, are done in ihe same order on all replicas of the object, ensuring sequential Using this technique works reasonably well, but not all objects need sequen- tial consistency C/onsider. for example, an oojject maintaining stocK prices lr trie market maker tor stock 1 issues an updated price tor il concurrently with another market maker issuing an update for stock 2, it is not essential that all copies of the object apply those two updates in the same order because they are independent It is probably sufficient that all processes apply the stream of updates from each market maker in the order they were sent, but this goal can be achmed by includ- including a sequence number generated by the sending process No object-wide sequencer is needed here. The above replication scheme, namely a replicated object with all copies being equal and any copy being allowed to issue updates after first getting a sequence number is only one of many replication protocols. Another one has one master copy of each ob|ect, plus some number of slave copies. AH updates are sent to the

object's master copy, which then applies the update and sends out tiie new state to all the slave copies. A third object replication strategy is having only one copy holding the object's state, with all the other copies being stateless proxies. When a read or write is done at a proxy (e.g.. a client machine), the request is forwarded to the copy holding the state and executed there? The strength of Globe is that each objeci can have its own replication policy Some objects can use active replication at the same time other objects arc using master-slave replication or any other strategy an object needs. Also_ each object can have its own policy concerning consistency, replica creation and removal, security, etc This is possible because all the policies are handled inside the object. Users of the object are not even aware of it, and neither .ire the system administrators This approach is in contrast to CORBA, which does not hide any of these policies inside objects, making it difficult to have 1000 different objects with 1000 different policies. A Globe object can be implemented as shown m Fig. 8-41 This figure illus- illustrates the subobjects from which a Globe object is composed. The control object accepts incoming method invocations and uses the other subobjects to get them done. The semantics subobject actually does ihe work required by the object's interface. It is the only part of the object's code that must be written by the object's programmer; all the rest can be taken from standard libraries, unless the

Six" [USTRtBLTED SYSTEMS 571 programmer wants a tic* strategy nut current^ avUiKible The itplicfition vuhob)C^\ joh is to nuiiagu replication This module can he replaced to switch fiorn dct>vo replication to muster-slave replication or any other replication Strategy without the rest of the object being affected. Similarly, the security subobjeel can be replaced 10 implement a ticw security policy (c g., to switch from ACLs to capabilities) ami the ci.mmiintcation subobjeel can t»e replaced to change network nroiocoK ie s from IP \4 to IP v6l without .itttcting the resi ot the ob|ect. Corn pule Replication—LC) (Security * \^_ subobject Operating system ^y subob|ect Messages in ana out go Ihjough Figure .if a Gl sider e objett. what happen To see how these subobjects interact, consider what happens, object's methods is invoked. The code pointed to by the interface is in the subobject, which then asks the replication subobject to do what it has to do object is actively replicated, a sequence number is first acquired. Then th cation subobject tells all replicas (including its own) to actually do the w nvoking ihei semantics object If the bject matslae and th re 8-41. Sim j s, when one of the control . If

(he y p, q q n the repli- replin subobject tells all replicas (including its own) to actually do the work by nvoking iheir semantics object. If the object is master-slave and the method invocation is on a slave, a message is sent to the master, and so on. At appropri- appropriate moments security checks are made by the security object (to see if the invoca- invocation is permitted, to see if outgoing data must be encrypted, etc.) A key element of Globe is the location service, which allo looked up anywhere in the world. The location service is built a . object registrations being kepi only in the node where the registration takes plac Pointers to this node are propagated up to]he top of the tree so it is always possi- possible to find the registration. Locality, partitioning of tree nodes, caching, and other techniques are used to make the scheme workable, even for mobile objects (Bal- lintijn et a!., 2000; Van Steen et al.. 1998a; and Van Steen et al., 1998b). s objects to be s built as a tree. With

572 MULTIPLE PROCESSOR SYSTEMS CHAP 8 8.3.6 Coordination-Based Middleware Our last paradigm for a distributed system is called coordination-based middleware We will start with the Linda system, an academic research project that started the whole Held, and then look at two commercial examples heavily inspired by it: publish/subscribe and Jim. Linda Linda is a novel system for communication and synchronization developed ai Yale University by David Geternter and his student Nick Carnero (Carriero and Gelernter. 1986; Carriero and Gelernter. 1989; and Gelernter. 1985). In Linda, independent processes communicate via an abstract tuple space. The tuple space is global to the entire system, and processes on any machine can imert tuples into the tuple space or remove tuples from the tuple space without regard to how or where they are stored. To the user, the tuple space looks like a big, global shared memory, as we have seen in various forms before [and in Fig. 8-22(C)] A tuple is like a structure in C or a record in Pascal. It consists of one or more fields, each of wh[Ch is a value of some type supported by the base language (Linda is implemented by adding a library to an existing language, such as C). For C-Linda, field types include integers, long integers, and floating-point numbers, as well as composite types such as arrays (including strings) and struc- structures (but not other tuples). Unlike objects, tuples are pure data, they do not have any associated methods. Figure 8-42 shows three tuples as examples ("abc", 2, 5) ("matrix-1-, 1,6,3.14) ("family", "is-sister", "Stephany", "Roberta") Figure 8-42. Three Linda luples Four operations are provided on tuples The first one, oul, puts a tuple into the tuple

space. For example. outf'abc", 2, 5); puts the tuple ("abc1, 2, 5) [nto the tuple space. The fields of out are normally constants, variables, or expressions, as in out("matrix-1", i, j, 3.14); which outputs a tuple with four fields, the second and third of which are deter- determined by the current values of the variables i and/ Tuples are retrieved from the tuple space by the in primitive. They are addressed by content rather than by name or address The fields of in can be expressions or formal parameters. Cons[der, for example. infabc", 2, ■>!);

SEC 8 3 DISTRIBUTED SYSTEMS 573 This operation "searches" the tuple space for a tuple consisting ot the string "abc", the integer 2, and a third field containing any integer (assuming that i is an integer) If found, the tuple [S removed from the tuple space and the variable i is assigned the value of [he third field. The matching and removal are atomic, so if two proceif.es execuie the same in Operation simultaneously, only one of them will succeed, unless two or more matching tuples are present. The tuple space may even contain multiple copies of the same tuple. The matching algorithm used by in K straightforward The fields, of the in primitive, called the template, are (concepiually) compared to the corresponding fields of every tuple in the tuple space. A match occurs if the following three con- conditions are all met; 1. The template and the tuple have the same number ot fields. 2. The types of the corresponding fields are equal. 3. Each constant or variable in the template matches its tuple field. Formal parameters, indicated by a question mark followed by a variable name or type, do not participate in the matching (except for type checking), although those containing a variable name are assigned after a successful match If no matching tuple is present, the calling process is suspended until another process inserts the needed tuple, at which time the caller is automatically revived and given the new tuple The fact that processes block and unblock automatically means that if one process is about to output a tuple and another is about to input it. it does not matter which goes first The only difference is that it the m is done before the out, there will be a slight delay until the tuple is available for removal The fact that processes block when a needed tuple is not present can be put to many uses. For example, it can be used to implement s-emaphores To create or do an up on semaphore S, a process can execute outC'semaphore Sh); To do a down, it does in("semaphore S1*): The state of semaphore S is determined by the number of ("semaphore S") tuples in the tuple space. It none exist, any attempt to get one will block until some other process supplies one. In addition to out and in, Linda also has

a primitive read, which is the same a.s in excepi that it does not remove the tuple from the tuple space. There is also a primitive eval, which causes its parameters to be evaluated in parallel and (he re- resulting tuple to be put in the tuple space. This mechanism can be used to perform an arbitrary computation. This is how parallel processes are created in Linda,

574 MULTIPLE PROCESSOR SYSTEMS CHAP 8 Publish/Subscribe Our next example of a coordination-based model was inspired by Linda and is called publish/subscribe (Oki et al.. 1993). It consists of a number of processes connected by a broadcast network. Each process can be a producer of informa- information, a consumer of information, or both. When an information producer has a new piece of information (e.g.. a new stock price), it broadcasts the information as a tuple on the network. This action is called publishing Each tuple contains a hierarchical subject line containing multiple Fields separated by periods. Processes that are interested in certain infor- information can subscribe to certain subjects, including the use of wildcards in the subject line. Subscription is done by telling a tuple daemon process on the same machine that monitors published tuples what subjects to look for. Publish/subscribe is implemented as illustrated in Fig. 8-43. When a process has a tuple to publish, it broadcasts it out onto the local LAN. The tuple daemon on each machine copies all broadcasted tuples into its RAM. It then inspects ihe subject line to see which processes are interested in it. forwarding a copy to each ore that is. Tuples can also be broadcast over a wide area network or ihe Internet by having one machine on each LAN act as an information router, collecting all published tuples and then forwarding them to other LANs for rebroadcasting. This forwarding can also be done intelligently, only forwarding a tuple to a remote LAN if that remote LAN has at least one subscriber who wants the tuple. Doing this requires having the information routers exchange information about subscribers. Producer LAN Consum' E Daemon Information router Figure 8-43. The publish/subscribe architecture. Various kinds of semantics can be implemented, including reliable delivery and guaranteed delivery, even in the face of crashes. In the latter case, it is neces- necessary to store old tuples in case they are needed later. One way to store them is to hook up a database system to the system and have it subscribe to all tuples. This can be done by wrapping the database system in an adapter, to allow an existing

DISTRIBUTED SYSTEMS 575 > work with the publish/subscribe model. As tuples come by. the Jini l-or over 50 years, computing has been CPU-centric, with a computer being a freestanding device consisting ot a CPU. some primary memory, and nearly always some mass storage such as a disk. Sun Microsystems' Jini (a variant spel- spelling ot genie) j,s an attempt to change that model to one tha(might be described as network-centric (Waldo. 1999). The Jini world consists of a large number ot self-contained Jini devices, each of which offers one or more services to the others. A Jini device can be plugged into a network and begin offering and using services instantly, with no complex installation procedure. Note that the devices are plugged into a network, not into a computer as is traditionally the case A Jim device could be a traditional com- computer, but it could also be a printer, palmtop computer, cell phone, TV set. stereo, or other dev[ce with a CPU, some memory, and a (possibly wireless) network connection. A Jini system is a loose federation of Jim devices that may come and go at will, with no ceniral administration. When a Jim device wants to jom the Jim federation, it broadcasts a packet on the local LAN or in the local wireless, cell asking if there is a lookup service present. The protocol used to find ,i lookup service is the discovery protocol and is one of the few hardwired protocols in Jim (Alternately, the new Jim device can wail until one of the lookup service's periodic announcements comes by. but we will not treat this mechanism here.) When the lookup service sees that a new device wants to register, it replies with a piece of code that can perform the registration Since Jim is an all Java system, the code sent is in JVM (the Java Virtual Machine language), which all Jini devices must be capable of running, u^iwlly interpretively. The new device now runs the code, which contacts the lookup service and registers with it for some fixed period of time. Just before the time period expires, the device can reregister if it wishes This mechanism means that a Jini device can jus! leave the system by shutting down and its previous ex[Stence will soon be forgotten, without the need for any central administration. The concept of registering for a fixed time interval is called acquiring a lease. Note thai since the code to register ihe device is downloaded into the devke. it can be changed as the system evolves without affecting the hardware or soft- software ot ihe device. |n fact, the device is not even aware of what the registration

576 MULTIPLE PROCESSOR SYSTEMS CHAP R protocol is. A part of the registration process that the deuce is aware ot corisi ts of it providing some

attributes and proxy code that other devices will later use to access it. A device or user looking for a particular service can ask the lookup service if it knows about one. The request may involve some of the attributes that device use when registering. If the request is successful, the proxy that the device pro-provided at registration time is sent back to the requester and is run to contact the device. Thus a device or user can talk to another device without knowing where u is or even what protocol it speaks. Jini clients and services (hardware or software devices) communicate and synchronize using JavaSpaces, which are modeled on the Linda tuple space but with some important differences. Each JavaSpace consists of some number of strongly typed entries. Entries are like Linda tuples, except that they are strongly typed, whereas Linda tuples are untyped. Each entry consists of some number of fields, each of which has a basic Java type. For example, an entry of type employee might consist of a string (for the name), an integer (for the department), a second integer (for the telephone extension), and a Boolean (tor works-full Just four methods are defined on a JavaSpace (although two of them have a variant form); I Write: put a new entry into the JavaSpace. 2. Read: copy an entry that matches a template out of the JavaSpace 3. Take: copy and remove an entry that matches a template. 4. Notify; notify the caller when a matching entry is written. The write method provides the entry and specifies its lease time, that is when » should be discarded. In contrast, Linda tuples stay until removed. A JavaSpace may contain the same entry multiple times, so it is not a mathematical set (jus,t as in Linda). The read and take methods provide a template for the entry being sought. Each field in the template can contain a specific value that must be matched, or can contain a "don'i care" wildcard that matches all values of the appropriate type. If a match is found, it is returned, and in the case of take, n is also removed from the JavaSpace. Each of these JavaSpace methods has two vanants, which differ in the case that no entry matches. One variant returns with a failure indica- indication immediately: the other one waits until a timeout (given as a parameter) has expired. The notify method registers interest in a particular template. If a matching entry is later entered, the caller's notify method is invoked. Unlike Linda's tuple space, JavaSpace supports atomic transactions. Using them, multiple methods can be grouped together. They will either all execute or

 none of thcnr will execute. During ihe iransaction. changes thai are made to the JavaSpace are not visible outside the

transaction Onh when the tr-insjuion com- commits, do the;* become visible to other callers JavaSpace can be used for synchronization between communicating processes For example, m a producer-consumer situation, the producer puts items in a JavaSpace as it produces them. The consumer removes them with take, blocking if none are available. JavaSpace guarantees that each of the methods is executed atoniiculjy, so there is no danger of one process trying to read an entry thai has onlj been half entered. S.4 RESEARCH ON MULTIPLE PROCESSOR SYSTEMS In this chapter we have looked at thrte kinds ot multiple processor systems: multiprocessors, multicomputer, and distributed systems. Let us also look briefly at the research in these three areas. Most of ihe research on multiprocessors relates to the hardware, in particular, how 10 build the shared memory and keep it coherent. However, there has also been some research on using \irtuul machine monitors on multiprocessors (Bugnion et a].. 1997) and on resource management on multiprocessors (Govil et a]., 1999). Thread scheduling is also an issue in terms of the scheduling algorithm (Arora et al . 1998; and Philbm et ,il.. 1996). and also in terms of contention for the run queue (Dandarnudi. |997). Multicomputer are much easier to build than multiprocessor* All thai is needed is a collection of PCs or workstations and a high-speed network For ihis reason, they are a popular research topic at universities. A lot ot the work relates to distributed shared memory in one form or another, sometimes page-bused but sometimes entirely in software (Carter et a].. 1995: Feeley et al . 1995. Johnson et al.. 1995; Iizkovitz and Schuster. 1999; Scales and Gharachorbo, 1997, and Stets et al.. 1997) Optimiz^g user-level communication is also a research topic (Von Eickenetal.. 1995) So is load balancing (Harchol-Baiter ,ind Downey, 1996) There are also many papers on distributed systems, for example on mid-middleware (Bernstein, 19°6), objects (Dogac et al , 1998). wireless systems (Lui e! al.. 1996), mobile agents (Chen et al.. 2000). programming environments iJo. 1999), distributed multimedia (Mourlas, 2000). theory (Buchx and Guelfi. 2000). and Web caching (Wolman et al., 1999). among others. Distributed file systems (Alexandrov el at.. 1998: Hartman and Oustcrhout. 1995: and Thekkath et al.. 1W) and mobile file systems (Segarra and Andn. 1999) are also also popular. S.S SUMMARY Computer systems can be made faster and more reliable by using multiple CPUs. Three organizations for multiCPU systems arc multiprocessors, multicom- puters. and distributed systems. Each of these has its own properties and issues.

578 MULTIPLF PROCESSOR SYSTEMS CHAP 8 A multiprocessor consols of two or more CPUs that jJiare a common RAM. The CPU-; can be interconnected by a bus, a crossbar switch, or u multistage switching network Vandu^ operating system configurations arc possible, includ- including giving each CPU its own operating system, having one master operating t-yv tern with the rest being slaves, or having a symmetric multiprocessor, in which [here [S one copy of Die operating system that any CPU can run In the latter case, lock-, are needed to provide synchronization When a lock is not available, a C PU can spin <>r do a context switch. Various, scheduling algorithms are possible, including timesharing, space sharing, and g.ing scheduling. Multicomputer*, also have two or mure CPUs, bur these CPUs each iuue- then own prtvdte memory. They do not shari: any common RAM. so all commumea- lion uses message pjssnig- In some cases, the network interlace boiiid has its own board CPU has to be tare fully organi7ed to avoid race conditions. User-level communication on multicomputer often uses remote procedure call, but distri- distributed shared memory can also be used. Load balancing of processes is Jn issue here, and the Cratious alEOnthms used tor n i ncludo sendci -initiated dig on trims receiver-initiated algorithms, and bidding algor ithnis. Distributed systems are loosely coupled systems each ot whose nodes is a complete computer with a complete set ot peripherals and iti, own operating sys- system. Often these systems are spread over a large geographical urea. Middleware is often put on top of the operating systcin to provide a uniform layer for applica- applications to interact With. The various kinds of middleware include documem-based, file-based, object-based, and coordination-based middleware Some examples ;<re the World W,de Web. AFS. CORBA, Globe, Linda, and Jmi PROBLEMS buted sysiems" (SET IC home uses several million idle personal lomputers in .ina radioielescope daia ro search for exiralerreMnal intelligence; II sO. how do i relate to the categones described in Fig. K I'1 2. Whai happens if two rPus m a multiprocessor attempi u> iiLL'css oxaeil> ihc s word of memory -it exactly the same insum'1 MIPS, about how many CPIK will it lake U, salur.iw a 400-MHi liu*'1 Assumt- th memory leierence jeqwres one bus cycle Now repeat ihis problem lor 4 sy^e which caching is used and the caches hive a W* hit rate. Finally, ivhai cache hit would be needed 10 allow 32 CPUs in share [he bus without overloading n?

CHAP 8 PROBLEMS 579 4. Suppose lhal the wire between switch 2A and

switch 3B in the omega network <if Fig 8-5 breakh Who is tun.If from whom ' 5. How is sijjnjl handling done in ihe model ot f-ig 8-717 6. Whin) a system cull i1- made in (he miidel ol h'ig 8-8, a prublcm has to be solved imnnMuK-ly jrier the inip lhai does no] oc-ur in rhe model of Fig 8-7 Wh,it is the naitirc i.f ihis problem .mil how miahi il be solved> 7. Ruwrue the emer-region code of Hg. 2-22 using the pure read lo reduce thrashing induced bv the TSL instruction 8. Are cniLCa] regions on code sections reall> necessary in Mil SMP operating system to containing the tnuiex will gel shuliled back and fonh between the CPL' holding ihe lock and the CPU requesting it rf boih of them keep touching the block To reduce bus traffic, the requesting CPU executes one TSL every 50 bus cycles, but ihe CPU holding q y bus runs ai 41@ MHz. what fraction of the bus hand width is eaten up by moving ihe cache block back and forth'1 10. In the text, n was suggested [hat a binarv exponential backoff algorithm he u-,ed between uses ol TSL ici poll a lock It was also suggested to have a maximum delay herween polls Would the algorithm work coircclly if there were no maximum dela\ ' 11. oupposc that the TSL instruction was not availiiblt' (or synchronizing it multiprocessor. Instead, another instruction. SWP was provided thai atornicallv swapped the contents ol a regisier with 4 word in memory Could thai be used to provide multiprocessor synchronization.' II so, how could it be Used ' If not, why does il not work7 12. In this problem you are to cotnpuie how much ol a bus load a spin lock puts on ihe hus. Imagine ihat each instruction executed by a CPL! takes 5 nsec After an instruc- instruction has completed, any bus cycles needed, tor example, for TSL arc carried tiui Each bus cycle takes an additional 10 nsec above and beyond the instruction execution time It' a process is attempting to enter a critical region using a TSL loop, what fraction of the bus bandwidth does n consume? Assume thai noimal caching is working so Ilia! letching an instruction inside the loop consumes no bus cycles 13. Fig. 8-12 wa^ said to depict a timesharing ens'ironment Why is only one process D) shown in part(b)' 14. Affinity scheduling reduces cache mi.sses Does il also reduce TLB misses" Whai about page faults.' 15. Fo, each ol the topologies of Pig. 8-16, whai is Hie diameter ol the interconnection network" Count all hops (host-router and rouier-rouicr) equally for this problem. 16. Consider the double torus topology of Fig. 8-l6(d) but expanded to size * x i. What is [he diameter ol (he neiwork'1 Hint. Consider odd k and even k differently.

 17. The bisection

bandwidth of an interconnection network is olten used js a measure of wotk mli) two equal-size units, [he capacity of the removed links is then added up If there are many way., to make the split, the one with the minimum bandwidth ■ ■. the hisCLljon b*indwidih Foi an interconnection network consisting of an s^oxS cube, what ,s the bisection bandwidth if each link is 1 Gbps' f8. Consider a multicomputer Ln which the network interface is in user mode, so only three copies are needed from source RAM to destination RAM Assume thai moving a '2-bil wciid to (it from the t>eiwori> interface hoard takes 20 nsec and that ihe nei- neiwork usell opcRiscs .it I Gbps What would ihe delay lot a f>4-byte packet being sent copying time? Now consider the case where two extra copies arc needed, to the ker- 19. Repeal the previous problem for bolh the three-copy case and the five-topy case bat this lime compute the bandwidth raiher than the delay 20. How musi the implementation of send and receive differ between a shared memory multiprocessor sVsteni and a multicomputer, and how does (his aftect performance1' 21. When transferring dala Irom RAM to ,i netwoik interlace, pinning a page tan be used, btu supjxise that system onIK io pin and unpin pages each take I jisec. Copying lakes 5 byte/nsec using DMA but 20 nsec per byte using programmed |/O How big does a packet have to be herore pinning the page and using DMA is worth it' 22. When a procedure is -looped up from one machine and placed on another to called b> RPC, some problems can occur In the ieM, we pointed otil tour ot these pointers, discussed is whai happens if the (remote) procedure exequies a system call Wh.it problems niighi that cause and whai mighi be d<jne to handle them? 23. In a DSM sysieni. when a page fault occurs, the needed page has tci be located List two possible ways to ftnd the pajie 24. Consider the processor allocation of Fig 8-25 Suppose that process H is mtiscd iroin node 2 to nude 3. Whai is the idtal weight of the extern.Li traffic now > 25. S!>me multicompUiers allow running processes io he migrated from one node to another Is it sufficient Lo stop a process, trecze us memory image, and just ship ihaL 26. Why is there a limit to cable length on an Ethernet neiwork" 27. [n Fig. 8-28, the third and fourth layers are labeled Middleware and Application on all are they different? ^ L S "^ ^"'^ ? ' °rmS ^

which service rype is most appropriate9 (a) Video on demand over ihe [inernei (h) Downloading .1 Web page. 29. DNS names, have <i hierarchical structure, sudi 3S fiMHi.edu 01 tales.general- h telnet com One way to ma,mani the DNS database would be as one centralized data- base, but that is

not done because il would get loo many requests/sec. Make a propo- proposal hnw the DNS database could be maintained in practice 30. In the discussion ol How, URLs are pressed by a browser, li was dialed lhai conneL- uoir- are made lo poi t SO Why'1 31. Cjn the URLs used m the Web exhibit location transparency? Fiplain your answer 32. When a browser ten-lies a Web p.ige, n first makes a TCP connection to get ihc text on the page (in ihe HTML language) Then li closes the connection and examines the page. If there are figures or icons, it then makes a separate TCP connection to fetch each one Suggest two alternative designs fro impTO\c performance here. 33. When session semantics are used, \K is always (rue thai changes to <i file art' immedi- immediately visible to the process making the change and never visible to processes on other machines. However, it is an open question as to whethet or not they should be immediately visible to other processes on the same machine Give an argument each 34. [n Ar-S, whole files are cached on ihe client machines Suppose lhai there k only so much disk space allocated tor cached fil^s and [he allocation is full When a new file is requested, what should bu done' Give an algorithm lor doing it 35. When multiple processes need access to daia, in what wa> ,s objecl-ba-.ed aweft better than shared memory ' 36. When a Linda in opetatum is done to locate j tuple, searching the enure iup|e spate linearly is very inefficient. Design a way to organize the tuple space that will speed up searches on all in operations 37. Copying bufters takts linic Write a C program to find out how much tune ii takes on a system lo which you have access. Use ihe thick ^r times functions to determine how long it lakes lo copj n large array Test with different array sizes to separaie copying time from overhead n,ne 38. Write C functions thai could he used as client and se^er stubs to make an RPC call ro the standard prmtf funcnon. and a main program lo test the functions. The client and server should communicate by means oi a data structure that could he transmuted over a network You may impose reasonable limits on the length of the formal string ^indihe number, types, and sizes of variables your client stub will accept 39. Wnic iwo programs to Simulate load balancing on a multicomputer The first program should set up m processes distributed across n machines according to an initialization tile t,ULn process stimuli! nave running lime cnosen ai random from a Oaussian

distribution. When a process exiis, the CPU must decide whether to give away processes or try 10 lind new processes The first program should use the sender-initiated algorithm 10 give away work if it has more lhan k processes iota! on us machine The second program should use the recewer-iniiiaied algorithm to fetch work when neuded. Make any other reasonable assumptions you need, but siaie them clcarl>

SECURITY Many companies possess valuable information that they guard closely. This information can be technical (e.g.. a new chip design or software), commercial (e.g., studies of the competition or marketing plans), financial (e.g., plans for a stock offering), legal (e.g.. documents about a potential merger or takeover), among many other possibilities. Frequently this information is protected by hav- having a uniformed guard at the building entrance who checks to see tbat all people entering the building are wearing a proper badge. In addition, many offices may be locked and some file cabinets may be locked as well to ensure that only author- authorized people have access to the information. As more and more of this information is stored in computer systems, the need to protect it is becoming increasingly important. Protecting this information against unauthorized usage is therefore a major concern of all operating systems. Unfortunately, it is also becoming increasingly difficult due to the widespread acceptance of system bloat as being a normal and acceptable phenomenon. In tbe following sections we will look ai a variety of issues concerned with security and protection, some of which have analogies to real-world protection of information on paper, bui some of which are unique to computer systems. In this chapter we wili examine computer security as it applies to operating systems.

9.1 THE SECURITY ENVIRONMENT Some people use ihe terms -'secant/' and "protection" interchangeably. Nevertheless, it is frequently useful to make a extinction beiween the general problems involved m making sure thai files are not iead or modified by unauthor- unauthorized persons, which mcludc technical, administrative, legal, and political issues un the one hand, and the specific operating sysiem mechanism*, used to provide seen- icy, on the other. To avoid confusion, we will use ihe term security to refer to lie o\crall problem, and [he ierm protection mechanisms to refer to the specific iijenitmg system mechanisms used to safeguard information in chc computer The boundaiy between iheni is not well defined, however. Firsi we will look ai

secu- security to see what ihe nature of the problem t<,. Later on in the chapter we will look at the protection mechanisms and models available lo help achieve security Security has many facets. Three of thu more important ones are the nature ot the threats, the nature of intruders, and accidental dald loss We will now look at these in turn. 9.1.1 Threats From a security perspective, computer systems have three general goals, wiih corresponding threats to them, as listed in Fig 9-1, The first one. data confiden- confidentiality, is concerned With havmg secret data remain secret More specifically, if the owner of some data has decided that these data are onl> to be made available to certain people and no others, the system should guarantee thai release of the data to unauthorized people does noi occur As a bare minimum, ihe owner should be able to specify who can see what, and ihe system should enforce these specifications. I Goal I Threat pala confidentiality""! Exposure of data [patajntegrity "Pfampenng with data] Sysiem_availabji^r^Deniai of service Figure 9-1. Sen The second goal, data integrity, means thai unauihonzed users should not be able (o modify any data without the owner's permission. Data modification in this context includes not only changing the data, but also removing data and adding false data as well. If a system cannot guarantee that data deposited in it remain unchanged until the owner decides to change them, it is not worth much as an information system. The third goal, system availability, means that nobody can disturb the system to make it unusable. Such denial of service attacks are increasingly common

SEC. 9 I THE SECURITY ENVIRONMENT 585 For example, if a computer is an Internet server, sending J flood ot requests to it may cripple It by eating up all ot its CPU time just examining anci discarding in- incoming requests. If it lakes, say, 100 [isec to process an incoming request to read a Web page, then anyone who manages to send 10.000 requests/sec can wipe it out Reasonable models and technology for dealing with attacks on corrfideniial- ity and integrity are available: foiling dental-of-services attacks is much hardei Another aspect of the security problem is privacy protecting individuals from misuse of information about them. This quic-k]y gets into many legal and moral issues Should the govemmem compile dossiers on everyone in order to catch X- cheaters. where X is "we]tare*' or "tax,'" depending on your polilics? Should the police be able to look up anything on anyone in order to stop organized crime" Po employers and insurance companies have rights9 What happens when these rights conflict with individual rights".' All of these issuer,

are txtremely important but are beyond the scope of this book. 9.1.2 Intruders Most people are pretty nice and obey the law. so why worry about security' Because there are unfortunately a few people around who are not so nice and want to cause trouble (possibly for their own commercial gam) In the security litera- literature, people who are nosing around places where they have no business being are called intruders or sometimes adversaries Intruders act in two different ways. Passive intruders just want to read files they are not authorized to read Active intruders are more malicious; they want to make unauthorized change to data_ When designing a system to be secure against intruders, it is important io keep in mind the kind of intruder one is trying to protect again.st. Some common categories are 1. Casual prying by nontechnical users Many people have personal computers on their desks that are connected to a shared file seiver, and human nature being what it is. some of (hem will read other people's electronic mail and other files if no barriers are placed in the way. Most UNIX systems, for example, have the default that ail newly created files are publicly readable. 2. Snooping by insiders. Students, system programmers, operators, and other technical personnel often consider it to be a personal challenge to break the security of the local computer system They often are highly skilled and are willing to devote a substantial amouni ot time to the effort. 3. Determined attempts to make money. Some bank programmers have attempted to steal from the bank they were working for. Schemes, have varied from changing the software to truncate rather than round interest, keeping the fraction of a cent for themselves, to siphoning

off accounts not used m years, to blackmail ("Pay me or I will des- destroy all the bank's records.")- 4 Commercial or military espionage. Espionage refers to a serious and well-funded attempt by a competitor or a foreign country to steal programs, trade secrets, palentable ideas, technology, circuit designs, business plans, and so forth Often ihu attempt will involve wiretap- wiretapping or even erecting antennas directed at the computer to pick up its electromagnetic radiation It should be clear that trying to keep a hostile toreign government from stealing military secrets js quite a different matter from trying to keep students from insert- inserting a funny message-of-the-day info the system. The amount of effort needed security and protection clearly depends on who the enemy is thought io be. Another category of security pest that has manifested itself in recent years is the virus, which will be discussed at length below Basically a virus is a piece of code that replicates itself and (usually) does

some damage. In a sense, the writer of a virus is also an intruder, often with high technical skills. The difference be- iween a conventional intruder and a virus is that the tormer refers to a person who is personally trying to break into a system to cause damage whereas the latter is a program written by such a person and then released into the world hoping it causes damage, intruders try to break into specific systems (e_g., one belonging to some bank or the Pentagon) to steal or destroy particular data, whereas a virus us- usually causes more general damage. In a sense, an intruder is like someone with a gun who tries to kill a specific person; a virus writer is more like a terrorist bomb-bomber who just wants to kill people in general, rather than some particular person. 9.1.3 Accidental Data Loss In addition to threats caused by malicious intruders, valuable data can be lost by accident. Some of the common causes of accidental data loss are 1. Acts of God: fires, floods, eanhquakes. wars, riots, or rats gnawing tapes or floppy d(sks. 2. Hardware or software errors; CPU malfunctions, unreadable disks or tapes, telecommunication errors, program bugs. 3. Human errors: incorrect data entry, wrong tape or disk mourned, wrong program run, lost disk or tape, or some other mistake Most of these can be dealt with by maintaining adequate backups, preferably far away from the original data. While protecting data against accidental loss may seem mundane compared to protecting against clever intruders, in practice, prob- probably more damage is caused by the former than the latter-

SEC 9 2 BASICS OF CRYPTOGRAPHY 587 9.2 BASICS OF CRYPTOGRAPHY A little knowledge of cryptography may be useful for understanding parts of tb[s chapter and some subsequent ones However, a serious discussion of cryptog- cryptography is beyond the scope ot this book. Many excellent books on computer secu- security discuss the topic at length The interested reader is referred to some of these (e_g.. Kaufman et al . 1995; and Pfleeger, 1997) Below we give a very quick dis- discussion ot cryptography for readers not tamiliar with ii at all. The purpose of cryptography is to take a message or file, called the plaintext. and encrypt it into the ciphertext in such a way that only authorized people know how to convert it back to the plaintext. For all others, the ciphertext is just an incomprehensible pile of bits. Strange as it may sound to beginners in the area, the encryption and decryption algorithms (functions) should always be public. Trying to keep them secret never works and gives the people trying to keep the secrets a false sense of security. In the trade, this tactic is called security by

obscurity and is employed only by security amateurs. Oddly enough, this category also includes many huge multinational corporations that really should know better. Instead, the secrecy depends on parameters to ihe algorithms called keys. It P is the plaintext file. Kh is the encryption key. C is the apherlext, and E is the encryption algorithm (i e., function), then C = EfP. KE). This is the definition of encryption. It says that the ciphertext is obtained by using the (known) encryption algorithm, E, with the plaintext, P. and the (secret) encryption key, Kr, as param- Similarly, P = D(C. Kn) where D is the decryption algorithm and KD is the decryption key. This says that to get the plaintext. P, back from the crphenext C. and the decryption key. KD, one runs the algorithm D with C and KVl as parame- parameters. The relation between the various pieces is shown in Fig. 9-2 J o P = D(C En ryp n Decrypuon Figure 9-2. RelJiionship between ihe plaintext and the ciphertexl

588 SRCURlfV t.HAP- y 9.2-1 Secret-Key Cryptography To make this clearer. consuter an encryption algorithm in which each letter is replaced by a diiierent letier. for example, all As are replaced by Q*. all Ss are replaced by W's. all C\ are replaced by £s. and so on like this. plaintext ABCDEFGH I JKLMNOPQRSTU VWXYZ eiphertc-xt QWEKTYU lOPASDhGHJ KLZXCVBNM This general system ls called a monoalphabetic substitution, with the key being the 26-iettL-r string corresponding ro the full alphabet The encryption key in this example is QWERTYUIOPASDFGHJKLZXCVBNM. For the key above, the plaintext A7TACK would be transformed into the ciphertext QZZQEA The de- decryption key lells how to ge, back from the apherte\i to the plaintext. In this example, the decryption key is KXVMCNOpHQRSZYUADLEiiWBUFT because an A in the ciphertext is a K in the plaintext, J B in the ciphertext is an X in the plaintext, etc. At tirst glance th[S might appear to be a sale system because although the cryplanalyst knows the general system (letter for letter substitution), he does noi know which ot the 26! = 4x 1026 possible keys is in use Nevertheless, given a surprisingly small amount of ciphertext. the ciphei can be broken easili. The basic attack takes advaniage of the statistical properties or natural languages In English, tor example, e is the most common letter, to]lowed by t.o. a, n, /. etc The most common two letter combinations, called digrams, are th. in, <'/: re. etc. Using this kind of information, breaking thu cipher is easy. Many cryptographic systems, like this one. have the property (hat given the encryption key it is easy lo find the decryption key. and vici; versa Such systems are called secret-key

cryptography or symmetric-key cryptography Although monoalphabetic substitution ciphers are worthless, other symmetric key algo- algorithms are kn,)wn and are relatively secure it the keys are long enough. Foi sen- ou^ security, probably 1024-bn keys should be used, giving a search space of 2mi ~ 2 x 10™ keys. Shorter keys may thwart amateurs, but not major govern- governments 9.2.2 Public-Key Cryptography Secret key systems are efficient because the amount of computation required to encrypt or decrypt a message is manageable, but have a big drawback, the sender and receiver must both be in possession ol the shared secret key They may even have to get together physically for one to give it to the other To get around this1 problem, public-key cryptography is used (Diffie and Hellman. 1976). This system has the property that distinct keys are used for encryption and decryption and that given a well-chCKen encryption key. it is virtually impossible

SPC. <i 2 BASICS OF CRYPTOGRAPHY 589 to discover the corresponding decryption key. Under ihese circumstances, trie encryption key can be made public and only the private decryption key kept Just 10 give a fee) for public-key cryptography, consider the following two questions: QueMion]: How much is 314159265358979 x 3141592653589797 Question 2: Whai is the square root ot 391257150641938709059482850824 1" Most sixth graders given a pencil, paper, and the promise of a really big ice cream sundae for the correct answer could answer question 1 in an hour or two. Most adults aiven a pencil, paper, and the promise ol a lifetime 50c/c tax cut could not solve question 2 at all without using a calculator, computer, or other external help. Although squaring and square rooting are inverse Dperations, they (lifter enor- enormously in their computational complexity. This kind of asymmetry forms the ba^is of public-key cryptography. Encryption makes use of the easy operation but decryption without the key requires you to perform the hard operation. A public key system called RSA exploits the fact that multiplying big numbers is much easier for a computer to do than factoring big numbers, espe- especially when all arithmetic is done using modulo arithmetic and all the numbers involved have hundreds of digits (Rivest et al., 1978). This system is widely used in the cryptographic world. Systems based on discrete logarithms are also used (El Gamal, 1985). The main problem with public-key cryptography is that it is a thousand times slower than symmetric cryptography. The way public-key cryptography works is that everyone picks a (public key. private key) pair and publishes the public key. The public- key is the

encryption key: the private key is the decryption key. Usually, the key generation is automated, possibly with a user-selected password fed mto the algorithm as a seed. To send a secret message to a user a correspondent encrypts 'he message with the receiver's public key. Since only the receiver has the private key, only the receiver can decrypt the message. 9.2.3 One-Way Functions There are various situations that we will see later in which it is desirable to have some function, / which has the property thai given f and its parametei t computing v -f{x) is easy to do, hut given only/(*), finding v is computational!} infeasihie. Such a function typically mangles the bits in complex ways, It might start out by initializing v to x. Then it could have a loop that iterates as many times as there are ! biis in x. with each iteration permuting the bits nf i m <in iteration-dependent way. adding in a different constant on each iteration, and gen erally mixing the bits up very thoroughly.

9.2,4 Digital Signatures Frequently it is necessary lo sign a document digitally For example, suppose a bank customer inserts the bank to buy some stock for him by sending the bank Jii cnuil message An hour after the order has been sent and executed, the stock crashes. The customer ntiv. denies ever ha\ing sent the email The bank can pro- produce the email, of course, bin the customer can claim ihe bank forged it m order to gel a commission. How does a judge know who is telling the truth'.1 Dignal signatures make it possible to sign email messages and other digital documents m such u way that they cannot be repudiated by (Vie sender later. One common way is to fir^t (un the document through a one-w-ay hashing algorithm that is very hard to invert The hashing functton typically produces a fixed-length result independent of the original document size The most popular hashing func- functions used are MD5 (Message Digest), which produces a 16-byte result (Rivest. 1992) and SHA (Secure Hash Algorithm!, which produces a 20-byie result (NIST. 1995) The next step assumes the use ot public-key cryptography as- described above. The document owner then applies his private key to the hash lo get D(hash) This value, called the signature block, is appended to the document and sent to the receiver, as shown m Fig. 9-3. The application of D to the hash is sometimes referred to as decrypting the hash, but it is not really a decryption because the hash ha.s not been encrypted. It is just a mathematical transformation on the hash. I Document Figure 9-3. (a) ('umputing a signature hk>ck (h> lfchal the receiver geis When the document and hash arrive, the receiver first computes the ha^h of the document using MD5 or SHA, as agreed upon m advance. The

receiver than applies the sender's public key to the signature block to gei E{D{ha^h)). In effect, it encrypts the decrypted hash, canceling it out and getting the hash back. If the computed hash does not match the hash from the signature block, the docu- document, the signature block, or both have been tampered with (or changed by accident). The value of this scheme is that it applies (slow) pubhe-key cryptogra- cryptography only to a relatively small piece of data, the hash. Note carefully that this method works only if for all x

BAS[CS OF CRYPTOGRAPHY 591 It is not guaranteed a priori that all encryption functions will have this property since all that we originally asked for was thai DiE(x)) = \ that is. E is the encryption function and /) is ihe decryption function. To get the signature properly in addition, the ordei ot application must not matter, that is, D and £ musi be commutative functions. Fortunately, the RSA algorithm has this property. To use this signature scheme, Ihe receiver must know the sender's public key. Some users publish their public key on their Web page. Others do not because they may be afraid of an intruder breaking in and secretly altering their key. For them, an altematfve mechamsm is needed to distribute public keys- One common method is tor message senders to attach a certificate to the message, which con- contains the user's name and public key and digitally signed by a trusted third party. Once the user has acquired the public key of the trusted third party, he can accept certificates from all senders who use this trusted third party to generate their Certi- Certificates- Above we have described how public-key cryptography can be used for digi- digital signatures. It is worth ment[oning thai schemes that do not involve public-key cryptography also exisf 9.3 USER AUTHENTICATION Now that we have some cryptographic background, let us start looking at security issues in operating systems When a user logs into a computer, the operating system normally wishes to determine who the user is, a process called user authentication. User authentication is one of those things we meant by "ontogeny recapitu- recapitulates phytogeny" in Sec. 1.2.5. Early mainframes, such as the EN1AC, did not have an operating system, let alone a login procedure. Later mainframe batch and timesharing systems generally d[d have a login procedure for authenticating jobs and users. Early minicomputers (e.g., PDP-1 and PDP-8) did not have a login procedure, but with the spread of UNIX on the PDP-11 mmicomputer, logging in was again needed. Early personal computers (e.g., Apple 11 and the original IBM PC) did not have a login procedure, but more sophisticated personal computer operating systems,

such as Windows 2000, again require a secure login. Using a personal compuier to access servers on a LAN (local area network) or one's account at an e-commerce Web site always requires logging in. Thus the subject of secure login has gone through several cycles, and is once again an important subject.

592 SECURITY CHAP 9 Having determined that authentication is often important, the next step is. to find a good way to achieve it. Most methods of authenticating users when they attempt 10 lug tn are based on one or threo general principles, namely [dentifymg 1. Something the user knows. 2. Something ibe user has 3 Something the user is. These principles lead to different authentication schemes with different complexi- complexities and security properties. In the following sections we will examine each of People who want to cause trouble on a particular system have to first log in to that sysiem, which means getting past whichever authentication procedure [S used. In the popular press, these people are called hackers. However, within the com- computer world, "hacker'" is a term of honor reserved for great programmers. While some of these are rogues, most are not. The press got this one wrong- In defer- deference to true hackers, we will use the term m the original sense and will call people who try to break into computer systems where they do not belong crackers. 9.3.1 Authentication Using Passwords The most widely used form of authentication is to requ[re the user to type a login name and a password. Password protection if- easy to understand and easy to implement. The simplest implementation just keeps a central list of (login name, password) pairs. The login name typed in is looked up in the list and the typed password is compared to the stored password. If they match, the login i allowed; if they do not match, the login is rejected. It goes almost withoui saying that while a password is being typed in, the computer should not display the typed characters, to keep them from prying eyes near the terminal. With Windows 2000, as each character is typed, an asterisk is displayed, With UNIX, nothing at all is displayed while the password is being typed. These schemes have different properties. The Windows 2000 scheme may make it easy for absent-minded users to see how many characters they have typed so far, but it a|so discloses the password length to "eavesdroppers" (for some rea- reason, English has a word for auditory snoopers but not for visual snoopers, other than perhaps Peeping Tom, which does not seem right in this context). From a security perspective, silence is golden. Another area in which not quite getting it right has serious security implica- implications is illustrated in Fig.

9-4. In Fig. 9-4(a), a successful login is shown, with system output in upper case and user input m lower case. In Fig. 9-4(b), a failed attempt by a cracker to log into System A is shown. In Fig. 9-4(c) a failed attempt by a cracker to log into System B is shown.

UStR AL"I H^ NT 1CATI0N LOGIN- ken LOGIN carol LOGIN carol PASSWORD FooBar INVALID LOGIN NAME PASSWORD Idunn. SUCCESSFUL LOGIN LOGIN- INVALID LOGIN LOGIN: (c) Figure 9-4. (a) A successful login (h) Login rejpcted after nan Login rejected Jttcr name and password arc Typed In Fig. 9-4(h), the system complains us soon as if sees an invalid login name. This is a mistake, as it allows the cracker to keep trying login names until she finds a valid one. In Fig. 9-4(c), the cracker is always asked for a password and gets no feedback about whether the login name itself is valid. All he learns is thai the login name plus password combination tried is wrong. How Crackers Break In Most crackers break in by just calling up ihe target computer and trying many (login name, password) combinations until they find one that works. Many people use their name in one form or another as their login name. For Ellen Ann Smith, ellen, smith, ellen_smith, ellen-smith, ellen.smith, esmith, eastnith, and eas are all reasonable candidates. Armed with one of those books entitled 4096 Names for Your New Baby, plus a telephone book full of last names, a cracker can easily compile a computerized list of potential login names appropriate to the country being attacked (ellen_smith might work fine in ihe U.S. or England, but probably not in Japan). Of course, guessing the login name is not enough. The password has to be guessed, too. How hard is that? Easier than you might think. The classic work on password security was done by Moms and Thompson A979) on UNIX sys- systems. They compiled a list of likely passwords: first and last names, street names, city names, words from a moderate-sized dictionary (also words spelled back- backward), license plate numbers, and short strings of random characters. They then compared their list to the system password file to see if there were any matches. Over 86% of all passwords turned up in their list. A similar result was obtained by Klein A990). Lest anyone think that better quality users pick better quality passwords, rest assured that they do not. A 1997 survey of passwords used in the financial district of London revealed that 82% could be guessed easily. Commonly used passwords were sexual terms, abusive expressions, people's names (often a family member or a sports star), vacation destinations, and common ohjects found around the office (Kabay,

1997). Thus a cracker can compile a list of potential login names and a list of potential passwords without much work.

594 SECURITY CHAP 9 Does it really matter it passwords, are easy to guess? Yes. In 1998. the Sun Jose Mercury News reported that a Berkeley resident. Peter Shipley, had set up several unused computers a» war dialers, which dial all 10.000 telephone numbers belonging to an exchange [e.g., D15) 770-xxxx]. usually tn random order to thwart telephone companies that frown upon such usage and try to detect it After making 2.6 million calls, he located 20,000 computers in the Bay Area, 200 of which had no security at all. He estimated that a determined cracker could break into about 75% of the other ones (Denning. 1999). The combination of a war dialer and password guessing can he deadly. An Australian cracker wrote a program that systematically dialed all the numbers at a telephone exchange and then attempted to break in using password guessing, noti- notifying him when it succeeded. Among the many systems he hroke into was a Citi- Citibank computer in Saudi Arabia, which allowed him to obtain credit card numbers and credit limits (in one case. $5 million) and transaction records (including at least one visit to a brothel). A cracker colleague of his also broke into the bank and collected 4000 cred.t card numbers (Denning, 1999). If such information were misused, Ihe bank would undouhtedly emphatically and vigorously deny th.it it could possibly be at fault, claiming that the customer must have disclosed the information. An alternative to vising a war dialer is to attack computers over the Internet Every computer on the Internet has a 32-bit IP address used to identify it People usually write these addresses in dotted decimal notation as w.»..v<-. where each of the four components of the IP address is an integer from 0 to 255 in decimal. A cracker can easily test if some computer has this IP address and is up and running by typing ping w.x.y.z If the computer is alive, it will respond and the ping program will tell how long the roundtrip time was in milliseconds (although some sites now disable ping to prevent this kind of attack). It is easy to write a program to ping large numbers of IP addresses systematically, analogous to what a war dialer does. If a live com- computer is found at w.x.y.z, the cracker can attempt to break in by typing telnet w x.y.z If the connection attempt is accepted (which it may not be. since not all system administrators welcome random logins over the Internet), the cracker can start trv ing login names and passwords from his lists At first, it is trial and error. How ever, ihe cracker may eventually be able to break in a few times and

capture the password file (located in /etc/passwd on UNIX systems and often publicly read- readable). Then he will begin to collect statistical information about login name usage frequencies to optimize future searches. Many telnet daemons break the underlying TCP connection after some number of unsuccessful login attempts in order to slow down crackers. Crackers

USER AUTHENTICATION 595 respond to that by starting up many threads in parallel, working on different target machines at once Their goal is to make as many tries per second as the outgoing bandwidth will allow. From their point of view, having to spray them over many machines being attacked simulianeously is not a serious disadvantage. Instead of pinging machines in IP-address older, a cracker may wish to target a specific company, university, or other organization, say. the University of Toobar atjoobar edit To find Dut what IP addresses, they use. all he has to do is lype dnsquery foobar edu and he will get a list ot some of their IP addresses. (Alternatively, the programs nstookup or dig can also he used) Since many organizations have 65,536 con- consecutive IP addresses (a common allocation unit in the past), once he knows the first 2 bytes of their IP addresses (which dnsquery supplies), it is straightforward to ping all 65,536 of them to see which ones respond and which ones accept telnet connections. From there on. it is back to guessing login names and passwords, a subject we have already discussed Needless to say. the entire process of starting with a domain name, finding the first 2 bytes of its IP addresses, pinging all ot them to see which ones are alive, checking to see if any accept telnet connections, and then trying statistically likely (login name, password) pairs is a process that lends itself very well to automation It will take many, many tries to break in, hut if there is one thing that computers are very good at. it is repeating the same sequence of commands over and over until the cows come home. A cracker with a high-speed cable or DSL connection can program the break in process to run all day long and just check back once in a while to see what has showed up. A telnet attack i.s clearly better than a war dialer attack since it goes much fas- faster (no dialing time) and is much cheaper (no long distance telephone charges), but it only works for machines that are on the Internet and accept telnei connec- connections. Nevertheless, many companies (and nearly all universities) do accept telnet connections so employees on a business trip or at a different branch office (or stu- students at home) can log in remotely. Not only are user passwords often weak, but sometimes the root password is too. In particular, some installations

never bother to change the default passwords that systems are shipped with Cliff Stoll, an astronomer at Berkeley, had observed irregularities on his system, and laid a trap for the cracker who had been trying to get in (Stoll, 1989). He observed the session shown in Fig 9-5 typed by a cracker who had already broken into one machine at the Lawrence Berkeley Laboratory (LBL) and was trying to get into another one. The uucp (UNIX to UNIX Copy Program) account is used for intermachine network traffic and has superuser power, so the cracker was now in a U.S. Dept. of Energy machine as superuser. Fortunately, LBL does not design nuclear weapons although its sister lab at Livermore, does. One hopes their security is better, but there is little reason

to believe that since amithei nuclear Weapons lah. Los AUmos. lost a hard disk full of classified information in 2000. LBL> telnet elxsi ELXSI AT LBL LOGIN root PASSWORD, root INCORRECT PASSWORD. TRY AGAIN LOGIN guest PASSWORD guest INCORRECT PASSWORD TRY AGAIN LOGIN uucp PASSWORD: uucp WELCOME TO THE ELXSI COMPUTER AT LBL Figure 9-5. How a tracker broke into a U S Pept or. tnergy computer at LBt Once a cracker has broken into a system and become superu.ser. it may be possible to install a packet sniffer, software that examines all incoming and oat- going network packets looking for certain patterns. An especially interesting pai- lern to look tor is people on the compromised machine logging into remote machines, especially as superuser there. This information can be squirreled away in a file for the cracker to pick up at his leisure later. In this way. a cracker who breaks into one machine with weak security can often leverage this into a way to break into other machines with stronger security. Increasingly many break ins are being done by technically naive users who are just running scripts they found on the Internet. These scripts either use brute force attacks of the type described above, or try to exploit known bugs in .specific programs. Real hackers refer to them as script kiddies. Usually, the script kiddie has no particular target and no particular intorma- tion he is, trying to steal. He is just looking for machines that are easy i(» hreak into. Some ol the scripts even pick a network to attack by chance, using a random network number (in the upper partof the IP address). They then probe all the machines on the network to see which ones respond. Once a database of valid IP addresses has been acquired, each machine is attacked in sequence. As a conse- consequence of this methodology, it can happen that a brand new

machine at a secure military installation can be attacked within hours of its being attached to the Inter- Internet, even though no one but the administrator even knows about it yet. UNIX Password Security Some (older) operating systems keep the password file on the disk in unen- unencrypted form, but protected by the usual system protection mechanisms. Having all the passwords in a disk file in unencrypted form is just looking for trouble because all too often many people have access to it. These may include system

SEC 9 > I-iSPK AUTHENTICATION 5V/ administrators, machine operators, maintenance personnel, programmers, man- management, and maybe even some secretaries A better solution, used in UNIX, works like this. The login program asks the user to type his name and password The password is immediately "encrypted ' by using it as a key to encrypl a fixed block ot data Effectively, a one-way fuc tion it, being run. with the password as input and a function of the password output This process is not really encryption, but it is easier to speak of i of ASCII lines, one per User, imul it finds the line containing the user's login name. If the (encrypted) password contained in tins line matches the encrypted password jusi computed, the login is permitted, otherwise it is refused. The users' passwords because they are not stored in unencrypted form anywhere in the system. However, this scheme can also be atcacked. as follows. A cracker first builds a dicttonary of likely passwords the way Morris and Thompson did Al leisure, these are encrypted using the known algorithm ll does not matter how Jong this process takes because it is done in advance of the bre.ik in. Now aimed with a list of (password, encrypted password) pairs, the cracker strikes. He reads the (pub- (publicly accessible) password file and strips out all the encrypted passwords These are compared to the encrypted passwords in his list For evtr> hit, the login name and unencrypted password are now kn<iwn A simple shell script can automate this process so it can be carried out in a fraction ot a second A typical run of the script will yield dozens of passwords. Recognizing the possibility of this attack, Morris and Thompson describee! a technique that renders the atiack almosi useless- Their idea is to associate an jt-bit random number, called the salt, with each password. The random number is changed whenever the password is changed The random number is stored in the password file in unencrypted form, so that everyone can read tt. Instead of just storing ihe encrypted password in the password file, the password and the random number are first concatenated and then encrypted together. This encrypted result is

stored in the password file, as shown in Fig. 9-6 for a password file with five users. Bobbie, Tony, Laura. Mark, and Deborah. Each user has one line in the file, with three entries separated hy commas" login name, salt, and encrypted password+salt. The notation e(Do$423ti) represents the result of concatenating Bobbie's password. Dog, with her randomly assigned salt. 4238. and running it through Ihe encryption function, e. It is tbe result of that encryption that is stored as the third field of Bobbie's entry. Now consider the implications for a cracker who wants to build up a list of likely passwords, encrypt them, and save the results in a sorted file.,/, so that any encrypted password can be looked up easily. If an intruder suspects that Dng might be a password, it is no longer sufficient just to encrypt Dog and put the result in/. He has to encrypt 2" strings, such a< DogOOOO, DogOOOl, Dog0002,

Bobbie, 4238, e(Do94238) _ _ i TOnyT9iejsF%%TaeFF29i8) | Laura, 6902, e(Shakespea^re6902) ' Mark 1694 e(XaB ® Bwczi 694) I Deborah, 1092. e(LordByron 1092) \ figure 9-6. The uv; of \LitUD dele.il precomp illation Ot encrypted pjsswords andsolorth and enter all of them in/ This (echnique increases (he size of tby 2". UNIX uses (his method with n = 12. For additional security, some modern versions ot UNIX make the password file itselt unreadable but provide a program to look up unifies upon request, adding just enough delay to greatly slow down any attacker The comhmation of sailing the password file and making il unreadable except indirectly (and slowly) can generally withstand most attacks on it Improving Password Security Although salting the password file protects against crackers who Try to preconipute a large list of encrypted passwords and thus bre.ik many passwords a! once, it does little to protect a user David whose password is ;ilso David A cracker can still just try guessing passwords one at a time. Educating users about the need ior strong passwoids is critical, bur tew insolations d« it One step further than user education is to have the computer help. Some computers have a program that generates random easy-to-pronounce nonsense words, such as fatally, garbungy, or bipittv that can be used as passwords (preferably with some upper case and special characters thrown in). The program that users call io install or change their password can also give a warning when a poor password is chosen. Among other Hems it might complain about are 1. Passwords should be a minimum oi seven characters 2. Passwords should contain both upper and lower case letters 3. Passwords should contain at least one digit or special character 4 Passwords should not be dictionary words,

people's names, etc. A lenient password program might just carp, a strict one could reject the password and demand a better one. The password program could also make a suggestion, as discussed above. Some operating systems require ysers to change (heir passwords regularly, to limit the damage done it a password leaks out. The trade-off here is that if users have to change their password loo often, they are going to run out of good ones

SRC l> "* L'SER AUTHENTICATION 599 and Man picking easy ones. If prevented In.m picking easy ones, they will torget them ami start writing them down on Micky notes attached to their monitors, which hecome*) a major security hole itseil One-Time Passwords The most extreme form of changing the passwords all the time is the one-time password When one-time passwords are used, the user gets a book containing a list of passwords Each login uses the next password m the list. If an intruder ever discovers a password, n will not do him any good, since next time a different password mus! be used it is suggested thai tbe user try to avoid losing the pass- password book". Actually, a book is not needed due to an elegum scheme devised by Leslie Lamport that allows a user to log in securely over an insecure network using one- onetime passwords (Lamport, 1981). Lamport's method can be used to allow a user running on a home PC to log in to a server over the internet, even though intruders may see and copy down all the traffic in bolh directions. Furthermore, no secrets have lo be stored m the file system of either the server or the user's PC The algorithm is ba>-ed on a one-way function, thai is. a function v = f(x) that has the property that given x it is easy to find y but given v it is computational int'easible lo find x. The input and output should be the same length, lor example, 128 bits. The user picks a secret password that he memorizes. He also picks an integer, n, which is how many one-time passwords the algorithm is able to generate. As an example, consider n = 4, although m practice a much larger value of n would be used. If the secret password is s, the first password is given by running the one-way function n times: Pr =/(/(/t/U)))) The second password is given by running the one-way function i, - I times: Pi = f(f(J 0)» The third password runs /twice and the founh password runs it once in general. P,_t =/(P|). The key faci to note here is lhat given any password in ihe sequence, it is easy to compute the previous one in the numerical sequence but impossible to compute the next one. For example, given P2 it is easy to find Pj but impossible to find P The server is initialized with Po, which is]usi f(P\). This value is stored in the password

file entry associated with The user's login name along with The integer 1, indicating that The nexi password required is P, When the user want to fog in for The first Time, he sends his login name to The server, which responds by sending the integer in the password file, 1. The user's machine responds with P,. which can be computed locally from s, which is typed in on the spot. The

600 SECURITY CHAP ' server then computes HP,) and compares this to the value stored in the password file (Po). It 'he values match, the login is permitted, the integer is incremented to 2, and P, overwrites Po in the password file. On the next login, the server sends 'he user a 2, and the user's machine com- p password file If mented to \ anij P /ulues match, the login is permitted, the integer is i eiwrites P i in the password file. The property 'ha! n that even though an intruder capture P,. he has compute PF+) fr< less When all n passwords hai new ^ecre! key. t. only P,_x which has already been used and i up. the server i> Challenge-Response Authentication A variation on the password idea is to have each new user provide a long list of questions and answers that are then stored on the server securely (e.g.. in encrypted form). The questions should be chosen so that the user doe1- not need to write them down. Possible questions are 1. Who is MarjoL-jn's sister? 2. On what street was your element 3. What did Mrs. Woroboff teach? At login, the server asks one of them at random and checks the answer. To make this scheme practical, though, many question-answer pairs would be nteded. Another variation is ch a I lenge-response. When this is used, the user picks an algorithm when signing up as a user, for example v'. When the user logs in. the strver ends the user an argument, say 7, in which ca.se the user types 49. The algorithm can be different in the morning and afternoon, on different days of trie If the user's terminal has real c a personal digital assistant, or a challenge-response can be used. ally brought to the mputing power, such as a personal coniputcr, cellular telephone, a more powerful form of In advance, the user selects a secret key, k, sysietn by hand A copy is also kept (securely) on the user's computer. Al login time, the server sends a i number, rto the user's computer, which then computes fir.k) and sends that back, where/is a publicly -known function. The server then does the compulation itself and checks if the result sent back agrees with the computation. The advantage of this scheme over a password ts that even if a wiretapper sees and records all the traffic in boih directions, he will leam nothing thai helps him next time. Of course, the function. /. has to be complicated enough that k cannot be deduced, even given a large set of

observations.

SEC 9.3 USER AUTHEN11CA1I0N 601 9.3.2 Authentication Using a Physical Object The second method tor authenticating users is to check for some physical object they have rather than something they know. Metal door keys have been used for centuries for this purpose Nowadays, the physical object used is often a plastic card that is insened into a reader associated with the terminal or computer Normally, the user must not only insert the card, but also type in a password, to prevent someone from using a lost or stolen card. Viewed this way, using a bank's ATM (Automated Teller Machine) starts out with the user logging in to the bank's computer via a remote terminal (the ATM machine) using a plastic card and a password (currently a 4-digit PIN code in most countries, but this is just to avoid the expense of putting a full keyboard on the ATM machine). informalion-bearing plastic cards come in two varieties magnetic stripe cards and chip cards. Magnetic stripe cards hold about 140 bytes of information written on a piece of magnetic tape glued to the back of the card. This information can be read out by the terminal and sent to the central computer. Often the information contains the user's password (e.g., PIN code) so ihe terminal can do an identity cheek even if ihe link to the main computer is down. Typically the password is encrypted by a key known only to the bank. These Cards cost about $0.10 to $0.50 each, depending on whether there is a hologram sticker on the front and the production volume. As a way to identify users in general, magnetic stripe cards are risky because ihe equipment to read and write them is cheap and widespread. Chip cards contain an integrated circuit (chip) on them. These cards can be further subdivided into two categories: stored value cards and smart cards. Stored value cards contain a small amount of memory (usually less than I KB) using EEPROM technology to allow ihe value to be remembered when the card is removed from the reader and thus the power turned off. There is no CPU on the card, so the value stored must be changed by an external CPU (in the reader) These cards are mass produced by the millions for about $i and are used. ft>r example, as prepaid telephone cards. When a call is made, the telephone just reason, these cards are generally issued by one company for use on only its machines (e.g., telephones or vending machines). They could be used for login authentication by storing a 1-KB password in them that the reader would send to the central computer, but this is rarely done However, nowadays, much security work is being focused on the smart cards which currently have something like a 4-MHz

8-bit CPU. 16 KB of ROM, 4 KB of EEPROM, 512 bytes of scratch RAM, and a 9600-bps communication channel to the reader. The cards are getting smarter in time, but are constrained in a variety of ways, including the depth of the chip (because it is embedded in the card), the width of the chip (so it does not break when the user flexes the card) and the cost (typically $5 to $50, depending on the CPU power, memory size, and presence or absence of a cryptographic coprocessor).

602 SECURITY CHAP 9 Smart cards can be used to hold money, as do stored value cards, but with much better security and universality. The cards can be loaded with money at an ATM machine or at home over the telephone using a special reader supplied hy ihe bank. Wben inserted into a merchant's reader, the user can authorize the card to deduct a certain amount of money from the card (by typing YES), causing ihe card to send a little encrypted message to the merchant. The merchant can later turn the message over to a bank to be credited for the amount paid. The bis advantage of smart cards over, say, credit or debit cards, is that they do not need an online connection to a bank If you do not believe this is an advan- t ge try ihe following experiment. Try to buy a single candy bar at a store and insist on paying with a credit card. If the merchant objects, say you have no cash with you and besides, you need ihe frequent flyer miles You will di cover that the merchant is not enthusiastic about the idea (because the associated costs dwarf ihe profit on the item). This makes smart cards useful for small store purchases, pay phones, parking meters, vending machines, and many other devices that nor- normally require coins. They are in widespread use in Europe and spreading else- elsewhere. Smart cards have many other potential uses (e.g., encoding the bearer's aller- allergies and other medical i not the place to tell secure login authenticat: tamperproof computer central computer t< renditions in a secure way for use m emergencies), but this :hat story. Our interest here is how they can be used for The basic concept is simple a smart card is a small, that can engage in a discussion (called a protocol) with a ithenticaie the user. For example, a user wishing to buy things at an e-commerce Web site could insert a smart card into a home reader attached to his PC, The e-commerce site would not only use the smart card to authenticate ihe user in a more secure way than a password, but could also deduct the purchase price from the smart card directly, eliminating a great deal of the overhead (and risk) associated with using a credit card for online purchases Various authentication schemes can be used with a smart

card. A simple challenge-response works like this. The server sends a 512-bit random number to ihe smart card, which then adds the user's 512-bit password stored in the card's EEPROM to it. The sum is then squared and the middle 512 bits are sent back to the server, which knows the user's password and can compute whether the result is correct or not. The sequence is shown m Fig. 9-7. If a wiretapper sees both messages, he will not be able to make much sense out of them, and recording them for future use is pointless because on the next login, a different 512-bit ran- random number will be sent. Of course, a much fancier algorithm than squanng can be used, and always i One disadvantage of any fixed cryptographic protocol is that over the course of time it could be broken, rendering the smart card useless. One way to avoid this fate is to use the ROM on the card not for a cryptographic protocol, hut for a Java interpreter. The real cryptographic protocol is then downloaded onto the card as a Java binary program and run interprelively. In this way, a soon as one

 Fig. t9-7. Use of a protocol is broken, a new one can be installed worldwide instantly. A disadvan- disadvantage of this approach is that it makes an already slow card even slower, but as technology improves, this method is very flexible. Another disadvantage of smart cards is that a lost of stolen one may be subject to a power analysis attack By observing the electric power consumed during repealed encryption operations, an expert with the right equipment may be able to deduce the key. Measuring the time to encrypt with various specially-chosen keys may also provide valuable information about the key. 9.3.3 Authentication Using Biometrics The third authentication method measures physical characteristics of the user that are hard to forge. These are called biometries (Pankanti et al., 2000). For example, a fingerprint or a voiceprint reader in the terminal could verify the user's identity. A typical biometrics system has two parts: enrollment and identification. During enrollment, ihe user's characteristics are measured and the results digi- digitized. Then significant features are extracted and stored in a record associated with the user. The record can be kept in a central database (e.g., for logging in to a remote computer), or stored on a smart card that the user carries around and inserts into a remote reader (e.g., at an ATM machine). The other part is identification. The user shows up and provides a login name. Then the system makes the measurement again. If the new values match the ones sampled at enrollment time, the login is accepted; otherwise it is rejected. The login name is needed because the measurements are not exact, so it is

difficult to index them and then search the index. Also, two people may have the same characteristics, so requiring ihe measured characteristics to match those of a specific user is stronger than just requiring it to match those of any user.

604 SECURITY The characterise chosen should have enough variability thai the sy disiinguish among many people withoui error. For example hair color i .._ ise loo many people share the same color Also, ihe char; not i good indicator toecau e loo many people --hare the ame color Mso, ihe charac teristic should noi varj much over time For example, a person voice mai be different due to a cold and & face may look different due to a beard or tnakt up not present at enrollment time Since later atnples Are never going lo match the enrollmeni values exaaly, the ystetn de ignen, have to decide how good the match has to be lo be accepted in particular the\ have K> decide whether tt is worse lo reject a iegitimaie user once m a while or lei an imposter get in once in a while. An e-commerce site might decide thai rejecting a loyal customer might be worse than accepting a small amount of fraud, whereas a nuclear weapons site might decide that refusing access to a genuine employee was better than letting random strangers m twice a year. Now let us take a brief look at some of the biometrics that are in actual use now. Finger length analysis is surprisingly practical When this is used, each ter- terminal has a device like the one of Fig. 9-8. The user inserts his hand into it, and the length of all his fingers is measured and checked against the database. Figure 9-8. A dev.ee for measuring finger lengih Finger length measurements are not perfect however. The system can be attacked with hand molds made out of plaster of Paris or some other material, pos- possibly with adjustable fingers to allow some experimentation. Another biomeiric that is gaining in popularity is retinal pattern analysis. Every person has a different pattern of retinal blood vessels, even identical twins.

SEC 9 3 USER AUTHENTICATION 605 They can be accurately photographed by a camera 1 meter from the subject, without the person even being aware of it. The amount of information in a retinal scan is much more lhan in a fingerprint, and it can be coded in about 256 bytes. Any technique thai relies on images is subject to spooting For example, a person could approach the ATM machine camera wearing dark glasses to which photographs of someone else's retinas were attached. Alter all. if the ATM's camera can take a

good reiinal photo at t meter, other people can do it loo. and at greater distances using (elephoto lenses For this reason, video cameras are gen- generally used instead of still cameras, and the pulsations normally present in the reti- retinal blptid vessels are looked tor A somewhat different technique is signature analysis. The user signs his name with a special pen connected to the terminal, and the computer compares it to a known specimen stored online or on a smart card- Even better is not to com- compare the signature, but compare the pen motions and pressure made while writing it. A good forger may be able to copy the signature, but will not have a clue as in the exact order in which the strokes were made or ;il what speed and what pres- A scheme that relies on minimal special hardware is voice hiomeincs (M;ir- kowitz, 2000). All that is needed is a microphone (or even a telephone): the rest ts software. In contrast to voice recognition systems, which try to determine what the speaker is saying, these systems try to determine who the speaker is Some systems just require the user to say a secret password, but these can be defeated by an eavesdropper who can tape record passwords and play them back later More advanced systems say something to the user and ask that it he repeated back, with different texts used for each login. Some companies are starting to use voice identification for applications such ds home shopping tiver the telephone because voice identification is less subject to fraud lhan using a PIN code lor identification. We could go on and on with more examples, hut two more will help make an important point. Cats and other animals mark off their territory by urinating around Us perimeter. Apparently cats can identify each other this way. Suppose that someone comes up with a tiny device capable of doing an instant urinalysii. thereby providing a foolproof identification. Each terminal could he equipped with one of these devices, along with a discreet sign reading: "For login, please deposit sample here," This might be an absolutely unbreakable system, but it would probably have a fairly serious user acceptance probltm The same could be <aid of a system consisting of a thumbtack and u small spectrograph. The user would be requested to press his thumb against the thumbtack, thus extracting a drop of blood for spectrographic analysis The point is that any authentication scheme must he psychologically acceptable to the user community Finger-length measurements probably will not cause any problem, but even something as nonintrusive as storing fingerprints on line may he unac- unacceptable to many people because they associate fingerprints with criminals

9.3.4 Countermeasures Computer installations that are really e b n eh na a frequently happens Ihe day after an intruder has broken m and done major di age, often take steps to make unauthorized entry much harder. For example, a company could have a policy that people working in the patent deparfmen a e only allowed to log in from 8 A.M. to 5 P.M M day h ah F day nd h n nl from a machine in the patent department connected to the company LAN. An attempt by a patent department employee to log in at the wrong time or from the wrong place would be treated as an attempt d b e k n Dial-up telephone fines can also be made secure as follows. Anyone can dial up and log in, but after a successful login, the system immediately breaks the con- connection and calls the user back at an agreed upon number This measure means than an intruder cannot just try breaking in from any phone line; only the user's (home) phone will do. In any event, with or without call back, the system should take at least 5 seconds to check any password typed in on a dial-up line, and should increase this time after several consecutive unsuccessful login attempts, in order to reduce the rate at which intruders can try. After three failed login attempts, the line should be disconnected for 10 minutes and security personnel notified. All logins should be recorded. When a user logs in, the system should report the time and terminal of the previous login, so he can delect possible break ins The next step up is laying baited traps to catch intruders. A simple scheme is to have one special login name with an easy password (e.g.. login name1 guest, password: guest). Whenever anyone logs in using this name, the system security specialists are immediately notified. All commands the intruder types are immediately displayed on the security manager's, screen so he can see exactly what the intruder is up to. Other traps can be easy-to-flnd bugs in the operating system and similar things, designed for the purpose of catching intruders in the act. Stoll A989) has written an entertaining account of the traps he set to track down a spy who broke into a university computer in search of military secrets. 9.4 ATTACKS FROM INSIDE THE SYSTEM Once a cracker has fogged into a computer, he can start doing damage. If the computer has good security, it may only be possible to harm the user whose account has been broken, but often this initial entry can be leveraged to break into more accounts later. In the following sections, we will look at some attacks that can be set up by someone already logged in, either a cracker who has gotten in llticlly or possibly a legitimate user with a grudge against someone.

SEC. 9.4 ATTACKS FROM INSIDE THE SYSTEM 607 9.4.1 Trojan Horses One hoary insider at lack is the Trojan horse, in which a seemingly innocent program contains code to perform an unexpected and undesirable function. This function might be modifying, deleting or encrypting Ihe user's files, copying them to a place where Ihe cracker can retrieve (hem later, or even sending them io the cracker or a temporary safe hiding place via email or FTP. To have Ihe Trojan horse run, the person planting il first has io get Ihe program carrying it executed. One way is to place Ihe program on ihe Interne! as a free, exciting new game, MP3 viewer, "special" porno viewer, or something else likely to altrac' attention, and encourage people to download it. When it runs, Ihe Trojan horse procedure ts called and can do anything (he user can do (e.g., delete files, open network con- connections, etc.)- Note (hat this ploy does not require the author of (he Trojan horse to break inio the victim's computer. There are other ways. Io trick the victim inio executing Ihe Trojan horse pro- program as well. For example, many UNlx users have an environment variable, SPATH, which controls which directories are searched for a command. It can be viewed by typing the following command to the shell: echo SPATH A potential selling for the user ast on a particular system mighl consist of Ihe fol- following directories:
:/usr/ast/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X i i :/usr/ucb:/usr/tnan\
:/usr/java/bin:/usr/java/lib:/usr/local/man:/usr/openwin/man Other users are likely Io have a different search path. When Ihe user types prog to the shell, Ihe shell first takes a look to see if there is a program named /imr/att/bin/prog. if there is. it is executed. If il is nol there, (he shell tries /usr/local/bin/prog, /usr/bin/prog, /bin/prog, and so on, trying all 10 directories in lurn before giving up Suppose thai just one of ihese directories was lefl unpro- unprotected so a cracker could put a program (here if this is Ihe first occurrence of Ihe program in Ihe list, it will be executed and Ihe Trojan horse will run. Most common programs are in /bin or /usr/bin, so putting a Trojan horse in /usr/bin/Xii/ls does not work for a common program because Ihe real one will be found first. However, suppose the Cracker inserts la into /usr/bin/XIh If a user mistypes la instead or Is (the directory listing program), now (he Trojan horse will run, do its dirty work, and then issue the correct message that la does not exist. By inserting Trojan horses into complicated directoties that hardly anyone ever looks at and giving them names that could represem common typing errors, there is a fair chance that someone will invoke one of them sooner or later. And thai someone mighl be the superuser

(even superusers make typing errors), in which

608 SECURITY CHAP 9 case the Trojan horse now has the opporiuniiy to replace /bin/Is wiih a version containing a Trojan horse, so U will be invoked all the time now. A malicious but legal user, Mai, could also lay a trap tor the superuser a,, tol- lows. He puts a version of Is containing a Trojan horse in his own directory ,ind Ihen does something suspicious that is sure lo attract Ihe superuser's attention, such as starting up 100 compute-bound processes ai once. Chances are the supcniser will check ihat out by typing cd /usr/mal Is-I CO see whai Mai has in his home directory Since some shells try the local direc- directory before working through $pATH, the superuser may have just invoked Mai's Trojan horse with superuser power. The Trojan horse could make /uxr/mal/btn/sh SETU1D root. All it takes is, two system calls: chown to change the owner of /usrfmaUbm/sh to root and chtnod, to set its SETUID bit. Now Mai can become superuser at will by just running that shell. If Mai finds himself frequently short of cash, he might use one of the follow- >ng Trojan horse scams to help his liquidity position. In the first one. the Trojan horse checks, to .see if the victim has an online banking program, such as Quicken, installed. If so, the Trojan horse directs the program to transfer some money from collection in cash later. In the second scam, the Trojan horse first turns off the modem's sound, then dials a 900 (pay) number, again, preferably in a far-away country, such as Mol- Moldova (part of the former Soviet Union). If the user was online when the Trojan horse was started, then the 900 phone number in Moldova needs to be a (very expensive) Internet provider, so the user will not notice and perhaps stay online for hours. Neither of these techniques is hypothetical; both have happened and are reported in (Denning, 1999). In the latter one, 800,000 minutes of connect time to Moldova were run up before the U.S. Federal Trade Commission managed to gei ihe plug pulled and filed suit against three people on Long Island They eventually agreed to return $2.74 million to 38,000 victims. 9.4.2 Login Spoofing Somewhat relaied to Trojan horses is login spoofing. Ii works as follows. Normally, when no one is logged in on a UNtx terminal or workstation on a LAN, a screen such as Fig. 9-9(a) is displayed. When a user sits down and types a logm name, the system asks for a password. If it is correct, the user is logged in and a shell is started. Now consider this scenario. Mai writes a program to display the screen of Fig. 9-9(b). It looks amazingly like the screen of Fig. 9-9(a), except that this is not the system login program running, but a phony one written by Mai. Mai now

ATTACKS FROM INSIDE THE SYSTEM walks away tn watch the fun from a safe distance. When ,i user sits down and lypes a Itigm name, the pnigram responds by asking for <i password and disabling echoing. After the login name and password have been collecied. they are writlen away to a file and the phony login program sends a signal to kill its shell. Tins action logs Mai oul and triggers the real Itigin program to start and display the pnimpt of Fig. 9-9fa). The user assumes that she made a lypmg emir and just logs in again. This time it works. Bui in ihe meantime. Mai has acquired another (login name, password) pair. By logging in at many terminals and starting the login spoofer on all of them, he can colleel many passwords The only real way to guard against this is to have the login sequence start wilh a key combination that user programs cannot catch Windows 2000 u-.es CTRL- ALT-DEL for this purpose, if a user sits down at a terminal and starts out by t> p- ing CTRL-ALT-DEL, the current user is logged out and the system login program is started. There is noway to bypass this mechanism. 9.4.3 Logic Bombs Another insider attack in these times nt high employee mobility is Ihe logic bomb. This device is a piece of code written by one of a company's (currently employed) programmers, and secretly inserted into the production operating s>s- lem. Af> long as the programmer feeds it its daily password, it does nothing However, if the programmer is suddenly fired and physically removed from the premises without warning, the next day @1 next week) the logic bomb does not get fed its daily password, so it goes off Many variants on this theme are also possible. In one famous case, the logic homb checked the payroll If the person-personnel number of the programmer did not appear in it fur two consecutive payroll periods, it went off (Spaf ford et al.. 1989). Going off might involve clearing the disk, erasing hies at random, carefully making hard-to-detect changes to key programs, or encrypting essential files. In the latter case, the company has a tough choice abnut whether to call the police (which may or may not result in a conviction many months later but certainly does not restore the missing files) nr to give m tn this blackmail and to rehire the ex-programmer as a "consultant" For an astronomical sum to fix the problem (and hope that he does not plant new logic bombs while doing so)

9.4.4 Trap Doors ■ is the trap door This problem i ,ysiem progiammer lo bypass somt mid add code to the login progran Another security hole caused by an tnsid created by code inserted into the system b> : normal check. For example, a programmer lo allow anyone to log in using the login name "zzzzz"

no matter what was in the password file. The normal code in the login program mighl look something like Fig. 9-lO(a). The trap door would be Ihe change lo Hg 9-I0(b) What the call to stremp does is check if the login name is "zzzzz" If so. (he login succeeds, no matter what password is typed If this trap door code were inserted by a progiam- progiammer working for a computer manufacturer and then shipped with its computers, the programmer could Jog into any computer made by his company, no matter who owned it or whal was in the password file The trap door simply bypasses the whole authentication process while (TRUE) (pnntfflogin; "); get-String(name); disab!e_echoing(); printf("password. "), get-Stnng(password); enable, echoingf); v = check_ validity! nam if (v) break; execute_she!!(name); while (TRUE) (printf("login: "), get Stnng(rame), disab!e_echomg(): pnntfC'password. "), geLstnng(password), enab!e_echomg(), = check_ validity (name, password); if (v 5trcmp(name, "zzzzz") == 0) break cecute shell(name) (a) (b) Figure 9-10. (a) Normat code (b) C ode with j irup door inserted One way for companies to prevent trap doors is to have code reviews as stan- standard practice. Wilh this technique, once a programmer has finished writing and testing a module, the module is checked into a code tiatahase. Periodically, all the programmers in a team get together and each one gets up in front of the group to explain what his code does, line by line. Not only does this greatly incieasc the chance lhat someone will catch a trap door, hui it raises the stakes tor the pro- programmer, since being caught red-handed is probably not a plus for his careei It the programmers protest too much when this is proposed, having two check each other's code is also a possibility. 9.4.5 Buffer Overflow orkers One rich source of att systems and most system (because progr like it and it due to the fact that virtually all opeiadng ; men in the C programming language : n he pled extremely efficiently).

Unfortunately, n<i C c lowing code sequence A ITACkS FROM INSIDE THE SYS TEM 611 impiler does anay bounds checking Consequently, the fol- while not legal, is also not checked char c[1024]- 1=12000. c[i] = 0, The result is that some byte of memory 10.976 bytes outside the array c is overwritten, possibly with disastrous consequences. No check is performed at run time to prevent this error This property of C leads to attacks of the following kind. In Fig. 9-11 (a), we set: the main program running, with its local variables on the stack. At some point it calls d procedure A, as shown in Fig 9-U(b) The standard calling sequence staris out by pushing the return address, (which

points to the instruction following the call) onto the stack It then transfers control io A, which decrements the stack pointer to allocate storage for Us local variables Virtual adores; ining (til AQer rh« pro Suppose that the job of A requires acquiring the full file path (possibly by con- concatenating the current directory path with a file name) and then opening it or doing something else with it. ,4 has a fixed-size buffer (i.e., array) B to hold a file name, as shown in Fig 9-lI(b). Using a fixed-size buffer to hold the file name is much easier to program than first determining the actual size and then dynami- dynamically allocating enough storage. If the buffer is 1024 bytes, that should handle a]] file names, right? Especially if the operating system limits file names (or better yet, full path*) to a maximum of no more than 255 characters

612 SECURITY ^HAP <) Unfortunately. thi\ redlining contain', a t;»taJ flaw. Suppose that the user oi the inograiti prnvidcs a tile name th.it i*. 1000 characters king. When (he]i!e name is used, it will fail t<i open, but the attacke, dues not care. When the pro- procedure copies the file name into the butter, ihe name overflows (he butter and mwu-ntes memory as showi, in the gray aiea ol" L-'ig 9-11(O Worse yet. it Ihe file name is king enough, it also overwrites tlic return address, so when A returns, the letiirn address is taken Irom (lie middle or (lie file name It this address is ran- random junk, the program will jump to a random address and probably crash wiihm a few instructions \ altd hniiirv program and ilic lavuut has been very, very carefully mude s<> that (lie word overlaying the letum address just happens to he the address or the slari ol the piogr.im. tor example, the addiess dl" #' What wilt happen c that when .4 returns, ihe piogram now in 8 will <slait executing. In etfect- the atlai-kei has inserted code into the piogram and gotten n executed long ciivnonine-ni strings, user input, oi anything else where the programmer has created a fixed-size butler to handle a user-supplied stung that wa\ expected to be short. By providing <i lona handcratted string containing a pmgiuin. it may he possible to get the pmgiam onio lhe stack and then get it executed. The C Libury tunction f;e-n, which re.ids a filing (or unknown size) into a hxed-suc buffer, but withoui checking for overflow, is notorious for being subject t<i this ktnd of attack Some compilers even detect the use of # <■/* and warn ah<nit it. Now comes the really bad part. Suppose that the program being attacked is SETUID root in UNIX (or has administraloi power in Windows 2000. which is effectively the saitie thing) Tht inserted code can now make A couple ot system calls to convert the .Mucker's shell file on the

disk into SETU1D mm. so tliat when it is executed it has snperusei powci AJlernalive-Iy, it can now niap in a specially prepared shared library that can do all kinds of damage. Or it can sim- simply issue an exec .system call |o overlay the current program with lhe shell, a eat- eating a shell with superior powers A substantial traction ot all security pitibleins are due to this flaw, which is difficult to fix because there are so many existing C programs around ihat do not cheek tor buffer oveiflow Detecting that d program hiis butter overflow problems is easy' nisi feed it l().0<)()-characier file names. l()O-digit salaues. or something equally unexpected to see if it dumps core. The nex| step is to analyze fhe core dump to see where (lie long stream is stored from Ihere. figuring out which characici o\erwntes the return address is nor so difficult If the source code is available, as it is for most UNIX programs, ihe attack is even easier because the hiyoui ot fhe stock is known in advance. The attack can be defended against by fixing (tie code to explicitly check Ihe length of all user-supplied strings be tore stuffing them into fixed-lenglh buffers. Unfortunately, tlie fact ihal some progiam i.s vulnerable to this, kind of attack generally shows up after a successful attack.

SEC. 9.4 ATTACKS FROM INSIDE THE SYSTf.M 613 9.4.6 Generic Security Attacks The usual way to test a system's security is to hire a group of experts, known as tiger teams or penetration teams, to see tf they can break in. Hebbard et al. A980) ined the same thing with graduate students. In the course of the years, these penetration team1; have discovered a number of areas tn which systems are likely to be weak. Below we have listed some of the more common attacks that are often successful While these were originally designed to attack ttmeshanng systems, they can often also be used to attack LAN servers and other shared machines. When designing a system, be sure tt can withstand attacks like these. 1. Request memory pages, disk space, or tapes and just read them Many systems do not erase them before allocating them, and they may be full of interesting information written by the previous owner. 2. Try illegal system calls, or legal system calls with illegal parameters, or even legal system calls with legal hut unreasonable parameters such as file names thousands of characters long. Many systems can easily be contused. 3. Start logging in and then hit DEL, RUBOlJT or BREAK hallway through the login sequence. In some systems, the password checking program will be killed and the login considered successful. 4 Try modifying complex operating system structures kept in us-er space (if any). Ln some systems (especially on

mainframes), to open a file, the program builds a large data structure containing the file name and many other parameters, and passes it to the system. As the file is read and written, the system sometimes updates the structure itself. Changing these fields can wreak havoc with the security, 5. Look for manuals that say "Do not do X " Try as many variations of X as possible. 6. Convince a system programmer to add a trap door by skipping cer- certain vital security checks for any user with your Login name 7. All else failing, the penetrator might find the system administrator's secretary and pose as a poor user who has forgotten his password and needs it quickly. An alternative approach is an out-and-out bribe of the secretary. The secretary probably has easy access to all kinds of wonderful information, and is usually poorly paid Do not underesti- underestimate problems caused by personnel These and other attacks are discussed by Linde A975), While this is an old paper, the attacks described in it often still work.

614 SECURITY CHAP 9 9.4.7 Famous Security Flaws Just as the transportation industry has the Titanic, the Hindertburg, and ihe Concorde disasters, operating system designers also have a few things they would rather forget about In this section we will look at some inieresting securily prob- problems that have occurred in three different operating systems: UNIX, TENEX, and OS/360. Famous Security Haws in UNIX The UNix utility lpr, which prints a file on the line printer, has art option to remove the file after it has been printed. In early versions of UNIX it was possible for anyone to use tpr to print and then have ihe system remove the password Hie Anoiher way to break into UNiX was to link a file called core in the working directory to the password file. The intruder ihen forced a core dump of a SETUIE) program, which ihe sysiein wrote on the core file, that is, on top of the password file. In this way, a user could replace the password file with one containing a few strings of his own choosing (e.g., command arguments). Yei another subtle flaw in UNIX involved the command mkdir foo Mkdir. which was a SETUID program owned by the root, first created the i-node for the directory foo with ihe system call mknod and then changed the owner of foo from its effective UiD (i.e., root) io its real UID (the user's UiD). When the system was slow, it was sometimes possible for the user to quickly remove the directory i-node and make a link to ihe password file under tlie name foo after the mknod but before the chown. When mkdir did the cfiown. i(made the user ihe owner of the password file. By putting the necessary commands in a shell scripi. they

could be tried over and over until the trick worked Famous Security Flaws in TENEX The TENEX operating system used to be very popular on the DEC-10 comput ers. It is no longer used, but it will iive on forever in the annals of computer secu rity due to the following design error. TENEX supported paging. To allow users to monitor the behavior of their programs, it was possible to instruct the system til call a user function on each page fault. TENEX also used passwords to protect files. To access a file, a program had to present ihe proper password to the operating sysiem at the time the file was opened. The operating system checked passwords one character at a rime, stop- stopping as soon as it saw that the password wa-. wrong. To break into TENEX an intruder would carefully position a password as shown in Fig- 9-|2(a), with the first character at the end of one page, and the rest at the start of the next page.

ATTACKS FROM INSIDE THE SYSTEM First page (in memory) Second page 1 in memory) ' — A A ~A~ ~~A~~ A~~ A ,_ Page (a) — B A A A (o) blern. — F A ~ a" A A A @ The next step was to make sure that the second page was not in memory, tor example, by referencing so many other page*, that ihe second page would surely be evicted to make room for ihetn. Now the program tried to open the vicutn's file, using the carefully aligned password. If the first character or the real pass- password was anything bui A, ihe sysiem would stop checking ai ihe firsl character and report hack with ILLEGAL PASSWORD. If. however, ihe real password did begin wiih A, ihe system continued reading and got a page fault, about which the intruder was informed If the password drd not begin with A. the intruder changed the password to that of Fig. 9-]2(b) and repeated ihe whole process to see if it began with B. It took at most 128 tries to go through the whole ASCII character set and thus deter-determine the first charactei Suppose lhat ihe first character was an F The memory layout of Fjg 9- I2(c) allowed the intruder to lest strings of the form FA, FB, and so on. Using this approach it took at most I28n tnes to guess an n-chamcter ASCII password, instead of 128". Famous Security Flaws in OS/360 Our last flaw concerns OS/360. The description lhat follows is slightly simpli-simplified but preserves the essence of the flaw. In this system it was possible to start up a tape read and then continue computing while the tape drive was transferring data to the user space. The trick here was to carefully stan up a tape read and then do a system call lhat required a user data structure, for example, a file to read and its password. The operating system first verified

that the password was indeed the correct one for ihe given file. After that it went back and read the file name again for the

m m ul»- ^mi i 3ctuat access (it could have savtd the name internally, hat it did not). Unfor- Unfortunately, just berore the system went to fetch the tile name the second time, the trie name was overwritten by the tape dnve The system then read the new file, tor which no password had been presented Getting the timing right took some practice, hut u was not that hard. 9.4.8 Design Principles for Security By now it should he clear that designing a secure operating system is not a trivial mailer People have been working on this problem ior decades without much sut.-t.ess As far hack as 1975, researchers have identified some general principles that should be used as a guide to designing secure systems (Saltzer and Schroeder, 1975). A brief summary of their ideas (hased on experience with MULT1CS) is given helow. These ideas are as valid now as when they were first stated. First the system design should be public. Assuming that the intruders do not know how the system works serves only to delude the designers. The intruders will find out sooner or later, and if the protection is compromised by this knowledge, the system is sunk. Second, the default should be no access Errors in which legitimate access is refused will be reported much faster than errors in which unauthorized access is allowed. When in doubt, say "No." Third, check for current authority. The system should not check for permis- permission, determine that access is permitted, and then squirrel away this information for subsequent use. Many systems check for permission when a file is opened, and not afterward. This means that a user who opens a file, and keeps it open for weeks, will continue to have access. even if the owner has long since changed the file protection or maybe even tried to delete the file Fourth, give each process the least privilege possible. If an editor has only ihe authority to access the file to he edited (specified when ihe editor is invoked), edi- editors with Trojan horses Will not be able to do much damage. This principle implies a fine-gramed protection scheme We will discuss such schemes latei in this chapter. Fifth, the protection mechanism should he Simple, uniform, and built into the lowest layers of the system. Trying to retrofit security to an existing insecure sys- system is nearly impossible. Security, like correctness, is not an add-on feature Sixth, the scheme chosen must be psychologically acceptable. If users feel that protecting their files is too much work, they just will not do it. Nevertheless, they will complain loudly if something goes wrong Replies of the form "It is your own

fault'" will generally not be well received To this list, we would like to add one other principle that has been gained by decades of hard-won experience: Keep ihe design simple

SEC 9.4 A lTACKS FROM INSIDE THE SYSTEM 617 If the system is elegani and simple, was designed by a single architect, and has a few guiding principles that determine the rcsl. tl has a chance of being secure. If ihe design is a mess, with no coherence and many fundamental concessions 10 ancient insecure systems m the name of backward compatibility, it is going to be a security nightmare You can design a system with many features (options, user-rriendliness, etc.) but a system with many features is a big system. And a big system is potentially an insecure system. The more code there is. the more secu- security holes and bugs there will be. From a security perspective, the simplest design is the best design 9.5 ATTACKS FROM OUTSIDE THE SYSTEM The threats discussed in the previous sections were largely caused from the inside, that is. perpetfated by users already logged in. However, for machines connected to the Internet or another network, there is a growing external threat A networked computer can be attacked from a distant computer over the network In nearly all cases, such an attack consists of some code being transmitted over the network to the target machine and executed there doing damage. As more and more computers join the Internet, the potential for damage keeps growing. In the following sections we will look at some ot ihc operating systems aspects of these external threats, primarily focusing on viruses, worms. Jnobilc code, and Jdva applets. It is hal-d to open a newspaper these days without reading about another com- computer virus or worm attacking the world's computers They .ire clearly a major security problem for individuals and companies alike. In the following sections we will examine how they work and what (.an be done about them. 1 was somewhat hesitant to write this section in so much detail, lest H give some people bad ideas, but existing books give tar more detail and even include real cede (eg., Ludwig. 1998). Also the Internet is full of information ahoul viruses so the genie is already out of the- hottk. In addition, it is hard for people to defend themselves against viruses if they do not know how they work Finally. there are a lol of misconceptions about viruses floating around that need correc- correction. Unlike, say, game programmers, successful virus writers tend nol lo seek pub- publicity after theii products have made iheir dchut. Based on the scanty evidence there is. it appears that most are high school or

college students or recent gradu- graduates who wrote the virus as a technical challenge, not realizing (or caringi that a virus attack can cost the collective victims as much as a hurricane oi eanhquakc. Let us call our antihero Virgil the virus wriier If Virgil is typical, his goals arc to produce a virus that spreads quickly, is difficult to detect, and is hard to get rid of once detected. What is a virus, anyway? To make a long story short, a virus is a program that can reproduce itself by attaching its code to another program, analogous to

618 SECURITY CHAP 9 how biological viruses reproduce. In addition, the virus call also do other rhings in addition to reproducing itself. Worms are like viruses but are self replicating. That difference will not concern us here, so we will use the term "virus" to cover both for the moment. We will look at worms in Sec. 9.5.5. 9.5.1 Virus Damage Scenarios Since a virus is just a program, it can do anything a program can do. For example, it can type a message, display an image on the screen, play music, or something else harmless. Unfortunately, it can al_so erase, modify, destioy, or steal files (by email ing them somewhere). Blackmail is also a possibility. Ima- Imagine a virus that encrypted a!l the files on the victim's hard disk, then displayed the following message. GREETINGS FROM GENERAL ENCRYPTION! TO PURCHASE A DECRYPTION KEY FOR YOUR HARD DISK, PLEASE SEND StOO IN SMALL, UNMARKED BILLS TO BOX 2154, PANAMA CITY, PANAMA THANK YOU WE APPRECIATE YOUR BUSINESS- Another thing a virus can do is render the computer unnsahle as long as the i irus is running This is called a denial of service attack The usual approach is consume resources wildly, such as the CPU. or filling up the disk with]unk Here Ls a one-line program that used to wipe out any UNIX system: mainf) {while A)fork();} This program creates processes until the process table is full, preventing any other processes from starring. Now imagine a virus that infected every program in the system wifh this code To guard against this prohlem. many modern UNIX sys- systems limit the number of children a process may have at once Even worse, a virus can permanently damage the computer's hardware. Many modern computers hold the BIOS in flash ROM, which can be rewritten und^i program control (to allow the manufacturer fo distribute bug fixes electronically) A virus can write random junk in the flash ROM so that the computer will no longer boot. If the flash ROM chip is in a socket, fixing the problem requires opening up the computer and replacing the chip If the flash ROM chip is sol- soldered to the parenfboard. probably the whole hoard has

to he thrown out and a new ont purchased. Definitely not a fun experience A virus can also be released with a specific target. A company could release a viru that checked if it was running at a competitor's factory and with no system administrator currently logged in. If the coast was clear, it would interfere with the production process, reducing product qualify, thus causing trouhle for the competitor. In all other cases if would do nothing, making it hard to detect. Another example of a targeted virus is one that could be written hy an ambi- ambitious corporate vice president and released onto the local LAN. The virus would

SEC. 9.5 ATTACKS FROM OUTSIDE THE SYSTEM 6l9 check if u was running on the president1!, machine, and if w. go find a spreadsheet and swap two random cells. Sooner or later the president would make a had decision based on (he spreadsheet output and perhaps get fired as .1 result, opening up a position for you-know-who. 9.5.2 How Viruses Work Enough for potential damage scenarios. Now lee us see how viruses work Virgil writes his virUh. probahly in assembly language, and then carefully inserts it into a program on his own machine using a tool called a dropper. That infected program is fhen distributed, perhaps by posting it to a bulletin board or a free software collection on the Internet. The program could be an exciting new g:ime. a pirafed version of some commercial software, or anything else likely to be con- considered desirable. People (hen hegin to download the intecied program. Once installed on the victim's machine, the virus lies dormant until the infected program is executed. Once started, li usually begins by infecting other programs' on the machine and then executing its payload, In many cases, the pay- load may do nothing until a certain dale has passed to make sun; lhat the virus is widespread before people begin noticing it. The date chosen might even send a political message (e.g.. if it triggers on the 10()th or 500th anniversary of some grave insult to the author's ethnic group) In the discussion below, we will examine seven kinds of viruses based on what is [nfecled. These are companion, executable program, memory, boot sec- sector, device driver, macro, and source code viruses. No doubt new types will appear in the future. Companion Viruses A companion virus docs not actually infect a program, but gets to run when the program is supposed to run. The concept is easiest to explain with an exam- example. In MS-DOS, when a user types prog MS-DOS first looks for a program named prog.com If it cannot find one. it looks for a program named prog.exe. In Windows, when ihe user clicks on

Slart jnd then Run, the same thing happens. Nowadays. mo«l programs are exe files; .<am Hies are very rare. Suppose thai Virgil knows lhat many people run prog.exe from an MS-DOS prompt or (rom Run on Windows. He can then simply release a virus called prog.com. which will get executed when anyone tries to run prog (unless he actu- actually types the full name, prog.exe). When prog.com has finished its work, it theii just executes prog.exe and the user is. none the wiser- A somewhat related attack uses the Windows desktop, which contains short- cub (symbolic links) to programs. A virus can ch;inge the target of a shortcut to

make it point to the virus- When the user double clicks on an icon, the virus rs executed. When it is done, the virus jusl runs the original targe! program. Executable Program Viruses One step up in complexity are viruses that infect executable programs. The simplest of these vtruscsjusl overwrites the executable program with rtselt These are called overwriting viruses. The infection logic of such a vims is given in Fig-9-13 #mclude <sys/types.h> /* standard POS1X headers •/ #include <;sys/stat.h> #mcliide ^dirent.h> #indude ^tcntl.h> #include <=urnstd.ri> stmcl slat shut; /• tor Istat call to see it Ue is sym link •/ search(char »dir_ name) { /* recursively search tor executables •/ D1R 'dirp; /* pointer lo an open directory siream •/ struct dirent *dp; /• pointer to a direclory enlry •/ dirp = opendir(dir name), /• open this directory •/ it (dirp == NULL) relurn; /• dir could not be opened1 torget it •/ while (TRUE) { dp = readdir(dirp); /* read nexl direclory enlry */ it (dp — NULL) { I* NULL means we are done •/ chdir ("..pl), /* go back lo parenl direclory */ break; /* exil loop •/ } it(dp->d name[0] == ■/) conlmue; /• skip the and .. directories */ lslat(dp->d_name, Ssbut), /* ts entry a symbolic link" •/ it (S_lSLNK(sbut.sl_ mode)) continue; /* skip symbolic links •/ it (chdir(dp->d_name) == 0) { /* it chdir succeeds, it musl be a dir */ searchf"); /* yes, enter and search it •/ }else{ /. no (file), mtectil */ if (access(dp->d_name,X OK) == 0) /• it execulable, mtect it */ irttect(dp->d_name), closedir(dirp); /• dir processed; close and return */ The mam program ot this virus would first copy its binary program into an irray by opening argv[0] and reading it in tor safe keeping. Thun it would tra-

 verse the entire file system starling at the root directory hy changing to the root directory and calling wan h with the root directory at, parameter. The recursive procedure

search processes a directory by opening it. then read- reading the entries one at a time using readdir until a NULL is returned, indicating that there are no more entries. If the entry is a directory, it is processed by changing to it and then calling search recursively; if it is an executahle Tile, it is infected by calling mfkxt with the name of the file to mlect as parameter. Files starling with •" " are skipped to avoid prohlems with the . and .. directories. Also, symbolic links die: skipped because the program assumes thai it can enter a directory using the chdtr system call and then get back to where it was by going to .. , something that holds for hard links but not symbolic links A fancier program could handle symbolic links, too - The actual infection procedure, inject (lu>t shown), merely has1 to open the file named in its parametet, copy the virus sa\cd in the array over the file, and then close the tite. This virus could be "improved" in various ways. First, a test could he inserted into infect to generate a random number and jusl return in most cases without doing anything. In, say, one call out of 128. infection would take place, thereby reducing the chances of early detection, before the virus has had a good chance to spread. Biological viruses have the same property: those that kill their victims quickly do not spread nearly as fast as those that produce a slow, lingering death, giving the victims plenty of chance to spread the virus An ;iltemative de-design would be to have a higher infection rate (say. 25%) but a cutotf on the num- number of files infected at once to reduce disk ac-tivity and thus be less conspicuous Second, mfect could check to see tf the file is already infected. Infecting the same file twice just wastes time. Third, measures could be taken io keep the time of last modification and file size the same as it was to help hide the infection. I-or programs larger than the virus, the size will remain unchanged, but for programs smaller than the virus, the program will now be bigger. Since most viruses are smaller than most programs, this is not a serious problem. Although this program is no! very long (the full program is under one page of C and the text segment compiles to under 2 KB), an assembly code version of it can be even shorter. Ludwig A998) gives an assembly code program forMS-pOS that infects all the files in its directory dnu is only 44 bytes when assemhled. Later in this chapter we will study antivirus programs, that ts programs that track down and remove viruses. Nevertheless, it is interesting to note that the logic of Fig. 9-13. which a virus could use to find all the executable tiles io infect them could also be used by an antivirus program to track down all the infected programs in order to remove the virus The technologies of infection and disin- disinfection go hand in hand, which is why

it is necessary to understand tn detail how viruses ivork in order to b« able to fight them effectively. From Virgil's point of view, the problem with an overwriting virus is that it is too easy to detect. After ali, when an infected program executes, it may spread

622 SECURITY CHAP 9 , supposed to do- and the user witl ,es attach themselves to the pro- function normally after- the vtrus some more, but it does not do notice this instantly. Consequently, most vinise gram and do their dirly work, but allow the pro* ward. Such viruses are called parasitic viruses. Parasitic viruses can attach themselves to the front, the back, or the middle of the executable program. If a virus attaches rtself to the front of a program, it has to tirst copy the program to RAM. write itself at the front of the tile, and then copy the program back from RAM following itselt as shown in Fig. 9-14(b). Unfortunately, the program will not run at its new virtual address, so either the virus has to relocate the program as it is moved, or slide it back to virtual address 0 after finishing its own execution. address I Executable Header Executable program Virus Header C Vtrus Executable program Header r "■■ «™ m™ Header g, hut all s position independent, using not hard for dn experienced pro To avoid either of the complex options required by these front loaders, must viruses are back loaders, attaching themselves to the end of the executable pro- program instead of the front, changing the starting address field in the header to point to the stan of the virus, as illustrated in Fig. 9-l4(c). The virus will now execute at a different virtual address depending which infected program is this means is that Virgil has to make sure his virus is position ind relative instead of absolute addresses. That grammer to do. Complex executable program formats, such as .cxe files on Windows anil nearly all modem UNIX binary formats, allow a program to have multiple text and data segments, with the loader assembling them in memory and doing relocation on the fly. In some systems (Windows, for example), all segments (sections) arc multiples of 512 bytes. If a segment is not full, the linker fills it out with (Is. A virus that understands this can try io hide itself m the holes. If it tits entirely, as in Fig. 9-t4(d), the file size remains the same as that of the unintected tile, clearly a plus, since a hidden virus is a happy virus Viruses that use thi« principle are called cavity viruses. Ot course, if the loader does not load the cavity areas into memory, the virus will need another way of getting staned.

Memory Resident Viruses So far we have assumed that when an infected

program is executed, the virus runs passes control to the real program, and exits, in contrast, a memory- resident virus stays in memory all the time, cither hiding at the very top of memory or perhaps down in the grass among the interrupt vectors, the last few hundred bytes ot which 4ie genera] h unused \ \er\ srri irt v irus can even modify the operating system's RAM bitmap to make the system Hunk the virus' memory is occupied, to avoid the embarrassment ot being overwritten. A typical memory-resident virus captures one of ihe trap or interrupt vectors hy copying the contents to a scratch variable and putting its own address there, thus directing that trap or interrupt to it. The best choice is ihe system call trap. In that way. the virus gets to run (in kernel mode) on every system cal] When it is done, iijust invokes the real system call by jumping io the saved trap address. Why would a virus want to run on ^very system call? To infect programs, naturally. The virus can just wait until an exec system call comes along, and then, knowing that the file at hand is an executable binary (and probably a useful one at that), infect it. This process does not require the massive disk activity of Fig. 9-13 so ([is far less conspicuous. Catching all system calls also gives the virus great potential for spying on data and performing all manner of mischief. Boot Sector Viruses As we discussed in Chap. 5. when most computers ale turned on. the BIOS reads the master boot record from the start of ihe boot disk inio RAM and exe- executes it. This program determines whrch partition rs active and reads in the first sector, the boot sector, from that partition and executes it. That program then eiiher loads the operating system or brings in a loader to load ihe operating sys- system. Unfortunately, many years ago one of Virgil's friends got the idea of creat- creating a virus that could overwrite the master hoot record or the boot sector, wiih devastating results, Such viruses, called boot sector viruses, are very common. Normally, a boot sector virus, [which include MBR (Master Boot Record) viruses!, first copies the true boot sector to a safe place on the disk so it can boot the operating system when ii is finished. The Microsoft di.sk formatting program. /disk, skips the first track, so that is a good hiding place on Windows machines. Another option is to use any free disk sector and then update the bad sector list t(j mark the hideout as defective. In fact, if ihe virus is large, it can also disguise the rest of itself as bad sectors, if the root directory is large enough and in a fixed place, as it is in Windows 98. the end of the root directory is also a possibility A really aggressive virus could even just allocate normal disk space for the true boot sector and itself and update the disk's bitmap or free list accordingly. Doing this requires an intimate

knowledge of the operating system's internal data structures, but Virgil had a good professor for his operating systems course and studied hard.

624 SECURITY CHAP. 9 When the computer is booted, the virus copies itself to RAM, either at the top or among the unused interrupt vectors. At this point the machine is jn kerne] mode, with the MMU off. no operating system, and nt, antivirus program running. Party lime for viruses. When it is ready, if hoots the operating system, usually staying memory resident One problem. howe\er. i-s how to get coniro] again later. The u^ual way is to exploit specific knowledge of how the operating system manages the interrupt veciors For example. Windows does not overwrite all the interrupt vectors in one blow. Instead, it loads device drivers one at a time, and each one capture^ the iniemipt vector it needs. This process can lake a minute. This design gives ihc virus the handle it needs, it starts out by capturing all the interrupt vectors as shown in Fig. 9-15(a) As drivers load, some ot the vec- vectors ale overwritten, but unless the clock driver is loaded first, there will be plenty of clock interrupts later that .start the virus Loss of the printer interrupt is shown in Fig. 9-15(h) As soon as the vims sees that one of its interrupi vectors has hetn overwritten. tt can overwrite that vector again, knowing that it is now safe (actu- (actually, some interrupt vectors are overwritten several times during booting, but the pattern is deterministic and Virgil knows it by heart) Recapture of the printer is shown in Fig. 9-]5(c). When everything is loadud, the virus restores all the inter-interrupt vectors and keeps only the system cal] trap vector for itself. After all. getting control on every system call is much mnre tun ihan getting control after every floppy disk operation, but during booting, » cannot take the nsk of losing control lorever. At this point we have a memory-re si dent virus in control of system calls. In fact, this is how most memory-resident viruses get started in life. Figure 9-IS. U\ (b) Aftei ihe v]rij.,rias noticed

SEC 9.^ A ITACKS FROM OUTSIDE THE SYSTEM 625 Device Driver Viruses Getiing into memory like this is a Intle like spelunking (exploring cavesl— you have to go through contortions and keep worrying about something falling down anil landing on your head. It would he much simpler ll the operating system would iust kindly load the virus officially With a little bii of work, that goal can he achieved. The trick is to infect a device driver, leading to a device driver virus In Windows and some UNIX systems, devrce drivers arc just executable programs that live on the disk and are loaded at

hoot time If one of them can be rnfected using a panrsitrc virus, the virus will always be officially loaded at hoot time. Even nicer, drivers run in kerne] mode and alter a driver is loaded, rt is called, grving the virus a ehmice to capture ihe system call trap vector. Macro Viruses Many programs, such as Word and Excel, allow users to write macros to group several comm;mds that tan later be executed with a single keystroke. Mac- Macros can also be attached to menu items, so that when one of them is selected, the macro is executed. In Microsoft Office, macros can contain entire programs in Visual Basic, which rs a complete programming language. The macros are lritei- preted rather than compiled, but that only affects execution speed, not what they can do. Since macros may be document specific. Office stores the macros for each document along with the document Now comes the problem. Vrrgil wrrtes a document in Word and creates a macro that he attaches to the OPEN FILE function. The macro contains a macro virus. He then emails the document to the victim, who naturally opens it (assum ing the email program has not already done this for him). Opening the document causes the OPEN F1LR macro to execute Since the macro can contain an afbi trary program, it can do anything, such ;rs rnfect other W/i/ documents ei (se files, and more. In all fairness to Microsoft. Word does give ;r warning when opening a file with macros, but most users do not understand what tbrs means and continue openrng anyway. Besrdes, Icgrtimate documents m ly also contam ni u. ros And there are other programs that do not evun give this warning, making rt even harder to detect a virus With the growth ot email, sending documents with viruses embedded in mac ros is an immense problem. Sirch viruses are much easier to v\ntc than coniealint. the true hoot sector somcwliere in the bad block Irst. hiding the virus among the interrupt vectors, and capturing the system call tnrp vector. This means thai increasrngly less skilled people can now write viruses, lowering the genera] qua] ity of the product and giving vrrus writers a bad name.

Source Code Viruses Parasitic and boot sector viruses are highly platform specific, document viruses are somewhat less" so {Word runs on Windows and the Macintosh, but not on UNIX). The most portable viruses of all are source code viruses. Imagine the virus of Fig. 9-13, but wrth the modification lhat instead of looking for binary exe- cutahle files, it looks for C programs, a change or only 1 line (the call to access). The infect procedure should be changed to insert the line #mc!ude <virus.h> at the top of each C source

program. One other insertion is needed, the line run virus(); to activate the virus. Deciding where to put this line requires' some ahility to parse C code, since rt must be at a place that syntactically allows procedure calls and j]so not at a place where the code would be dead (e.g.. following a reiurn state-statement). Putting it m the middle of a comment does not work erther, and putting it inside a loop might be ton much of a good thrng. Assumrng the call can be placed properly (for example, just before the end or main or before the relum statement if there is one), when the program is compiled, it now contains the virus, taken irom virus.h (although proj.h might attract less attention should somebody see it). When the program runs, the virus will be called. The virus can do anything it wants io, for example, look for other C programs to infect. If it finds one. it can include just the two lines given above, but this will only work on the local machine, where virus.h is assumed to be installed already. To have this work on a remote machine, the full source code of the viru.s must be included. This can be done by including the source L-ode of the virus as an initialised character string, preferably as a list of 32-bit hexadecimal integers to prevent anyone from figuring out what it does. This string will probably be fairly long, but with today's mul- timegaline code, it might easily slip by. To the uninitiated reader, all of these ways may look farrly complicated One can legitimately wonder if they could be made to work in practice. They can be Virgil is an excellent programmer and has a lot oi free time on his handv Check your local newspaper for proof. 9,5.3 How Viruses Spread There are several scenarios for distribution. Let us start with the classical one. Virgil writes his virus, inserts it into some program he has written (or stolen), and starts distributing the program, for example, by putting it on ,r shareware Web site. Eventually, somebody downloads the program and runs it. At this point there are several options. To start with, the virus probably infects more files on the hard disk, just in ca.se the victim decides to share some of these

SEC 9 5 ATTACKS FROM OUTSIDE THE SYSTEM 627 with a friend later, it can also try to infect the boot sector of the hard d'sk. Once the hoot sector is infected. U is easy to start a kernel-mode memory-resident virus on subsequent booh In addition, the virus can check to see if there are any floppy disks m the drives, and if so. infect their files and boot sectors- Floppy disks are a good target because they get moved from machine to machine much more often than hard disks. It a floppy disk boot sector is intecied and that disk is later used to boot a different machine, it can start infecting files and the hard disk boot

sector on that machine In the past, when floppy disks were the main transmission medium for programs, this mechanism was the main way viruses spread. Nowadays, other options are available to Virgil. The virus can be written to check if the infected machine is on a LAN, something ihat is very likely on a machine belonging to a company or university. The virus can Ihei, start infecting unprotected files on the servers connected tij [his LAN. This infection will not extend to protected files, but that can be dealt with hy making infected programs act strangely. A user who runs such a program will likely ask the system adminis- administrator for help The administrator will then try out the strange program himself io see what is going on. if the administrator does this while logged in as siiperuser, the virus can now infect the system hinaries. device drivers, operating system, and hoot sectors. All j] takes is one n,istake Itke this and all the machines on the LAN are compromised Often machines on a LAN have authoi ization to log onto remote machines over]he Internet or a private corporate, or even authorization to execute com- commands remotely without logging in. This ability provides more opportunity tor viruses to spread. Thus one innocent mistake can infect the entile company To prevent this scenario, all companies should have a general policy telling adminis- Another way to spread a vrrus rs to post an infected program]<> a UShNF.T newsgroup or bulletin board system to whrch programs are regularly posted. Wso possible is to create a Web page that requires a special hrowser plug-in to view, and then make sure the plug-ins are infected. A different attack is to infect a document and then enua\ it to many people or broadcast it to a mailing list or USENhT newsgroup, usually as an attachment. Even people who would never dream of running a program some stranger sent them might not realize that clicking oil the attachment to open it can release a virus on their machine To make matters worse, the urus can then look for the user's address book and then mail itself to everyone in the address hook, usually with a Subject line that looks legitimate oi interesting, like Subject: Change of plans Subject: Re- that last email Subject. The dog died last night Subject 1 am senously ill Subject-1 love you

628 SECURITY CHAP 9 When the email arrives, the receiver sees]hat the sender rs a triend or colleague, and thus does not suspect trouble Once the ernatl has heen opened, it is too late. The '"I LOVE YOU" virus thai spread around ihe world in June 201H worked this way and did a billion dollars worth of damage Somewhat related to the actual spreading or active viruses is

the spreading of virus technology. There are groups of vrrirs writers who actively communicaie over the Internet and help each other develop new technology, tools, and viruses. Most or these are probably hobhyists rather than career criminals, but the effects can be jusi as devastating. One other category of virus wrrtcrs is the mtlitary. which see^ viruses as a weapon of war potentially able to disable an enemy s computers. Another issue related to spreading viruses is avoiding detection. Jai]s have notoriously bad computing facilities, so Virgil would prefer avoiding them- if he posts the initial virus from his home machine he is- running a certain risk If the attack IS successful, the police might track him down by looking for the virus message with the youngest tirnestamp. since that is probably closest to the source of the attack. To minimize his exposure. Virgil might go to an Internet cafe in a distant city and log in there. He can either bring the virus on a floppy disk and read it in him- himself, or if the machines do not all have floppy disk drives, ask the nice young lady at the desk to please read in the file book. Joe so he can print it. Once it is cm his hard disk, he renames the file virus.exe and executes it, infecting the entire LAN with a virus that triggers two weeks later, just in case the police decide to ask the airlines for a list of all people who flew in that week. An alternative k to Forget the floppy disk and get the virus from a remote FTP site Or bring a lapiop and plug it in to an Ethernet or USB port that the Internet cafe has thoughtlully pro- provided for laptop-toting tourists who want to read iheir email every day. 9,5.4 Antivirus and Anti-Antivirus Techniques Viruses try to bide and users try to find them, which leads io a cat-and-mouse game. Let us now look at some of the issues here To avoid showing up in direc- directory listings, a companion virus, source code vims, or other tile that should not he there can mm on the HIDDEN bit in Windows Or use a file name beginning with the - character in UNIX. More sophisticated is to modify Windows1 expleier oi UNIX' h to retrain from listing files whose names begin with Virgil- Viruses can also hide in unusual and unsuspected places, such as ilie bad sector list on the disk the Windows registry (a d i) Th

SEC g 5 ATTACKS FROM OUTSEiJK THh SYS TEM 629 Virus Scanners Ckariy. (he average garden-variety user is no! going to find many viruses that do [heir besi to hide, so a market has developed for antivirus software. Below we will discuss how ihis software works. Aniivinis software companies have iabora- lones "i which deduced scientists work long hours tracking down and under- understanding new viruses. The first step is to have the virus infect a

program that does nothing, often tailed a goat file, to get a copy of the vim*, in Us purest form. The next step is to make an exact listing of the virus' code and enter it info the data- database of known viruses. Companies compete on the size of their databases. Inventing new viruses just to pump up your database is not considered sporting Once an antivirus program is installed on a customer's machine, the first thing it does is scan every executable file on the disk looking foi any of Ihe viruses in the datahase of known viruses. Most antivirus companies have a Web site from whicb customers' can download the descriptions of newly-discovered viruses into their databases- If the user has]0,000 tiles and the database has 10,000 viruses, some clever programming is needed to make it go fast, of course Since minor variants of known viruses pop up all the time, a fuzzy search is needed, so a 3-byte change to a virus does not let it escape detection Howe\er, fuzzy searches are not only slower than exact searches, but they may turn up false alarms, thai is, warnings about legitimate files that happen to contain some code vaguely similar to a virus reported in Pakistan 7 years ago What is the user sup- supposed to do with the message: WARNING' File xyz exe may contain the lahore-9x virus Delete? The more viruses in ihe database and ihe broadei ihe Criteria lor declaring a hit, the more false alarms there will be. If there are too many, the user will give up in disgust. But if the V]rus scanner insists on a very dose match, it may miss some modified viruses. Getting it right is a delicate heuristic balance Ideally, the lab should try to identify some core code in the virus that is not likely to change and use this as the virus signature to scan for. Just because the disk was declared i till ,, gar lit- aiiuucu il" auvvc Another way for the anlivirus program lo detect file infection is to record and store on the disk the lengths of all files If a file has gruwn since Ihe las! check, i(might be infected, as shown in Fig. 9-l6(a-b) However, a clever virus can avoid

 i deieclion by compressing Ihe program and padding out the file to Us original length. To make this scheme work, the virus musl contain both compression and decompression procedures, as shown in Fig. 9-I6(c). Executable program Header File is longer Virus Executable program Header Ong nal ze 13 Virus decompressor Comoressor Compressed executable progrem Header Figure 9-16. (a) A progra: ed program (d) An cocr (b) An infected program I Another way for the virus to try to escape detection is to make sure ils representa- representation on the disk does not look at all like its representation in the antivirus software's database. One way to achieve

this goal is to encrypt itselt with a dif- different key for each file infected. Before making a new copy, the virus generates a random 32-bit encryption key, for example by XORmg the current time with (he contents of, say. memory words 72,008 and 319.992. It then XORs its code with this key, word by word to produce the encrypted virus stored m the infected fife, as illustrated in Fig. 9-16(d). The key is stored in the tile. For secrecy purposes, putting (he key in (he file is not ideal, bu(the goal here is to foil the virus scanner, not prevent the dedicated scientists at the antivirus lab from reverse engineering the code. Of course, to run. the virus has to first decrypt itself, so ii needs a decrypting procedure in the file as well. This scheme is still not perfect because the compression, decompression, encryption, and decryption procedures are the same in all copies, so the antivirus program can just use them as the virus signature to scan fur Hiding the cumpies- sion. decompression, and encryption procedures is easy: they are just encrypted along wi(h the rest of the virus, as shown in Fig. 9-l6(e). The decryption cade cannot be encrypied. however. It has to actually execute on the hardware to dec rypt the rest of the virus so it rnunt be present tn plaintext form. Antivirus pro- programs know this, so they hunt for the decryption procedure.

ATTACKS FROM OUTSIDE THE SYSTFM 631 ir, Virgil enjoys having ihe last \vord. so he proceeds as follows. Sup- Suppose that the decryption procedure needs to perform fhe calculation The straightforward assembly code tor this, calculation for a generic two-address computer is .shown m Fig, 9-l7(a). The lirst address is ihe source; ihe second is the destination. so MOV A,Rt moves the variable A to the register R1. The code in Fig. 9-l7(b) does the same thing, only less efficiently due to the NOP (no opera- lion) instructions interspersed with the real code. MOV A.R1 ADDB.Rt ADDC.R1 SUB S4.R1 M0VR1.X MOV A,R1 NOP ADD B,R1 NOP ADDC R1 NOP SUB #4 NOP MOVR (b) hR1 MOV A.Rt ADD #0,R1 ADD B.Rt OR R1.R1 ADDCR1 SHL #0,R1 SUB S4,R1 JMP +1 MOVR1.X @ MOV ApR1 OR R1.R1 ADD B,R1 MOVR1.R5 ADDC.R1 SHLR1.0 SUB#4,R1 ADD R5 R5 MOV R1,X MOV R5,Y MOV A,R1 TSTR1 ADDC.R1 MOV R1,R5 ADD B,R1 CMP R2.R5 SUB #4 R1 JMP +1 MOV R1 ,X MOV R5,Y (e) Figure 9-17. Examples 111 a polymorphic- virus But we are not done yet. It is also possible to disguise the decryption ende. There are many ways u> represent NOP. For example, adding 0 to a register, ORing it with itself, shying it left 0 bits, and jumping 10 the nexl instruction all do nothing. Thus ihe program of Fig. 9-17(c) is functionally the

same as the one of Fig. 9-17(a). When copying itself, (he v,rus could use Fig. 9-l7(c) instead of Fig.9-]7(a) and still work later when executed. A virus lhai mutates on each copy is called a polymorphic virus Now suppose that R5 is noi needed during this piece of (he code. Then Fig. 9- 17(d) is also equivalent to Fig. 9- 17(a) Finally, in many cases it is possible to swap instructions without changing what (he program does, so we end up Vvith Fig. 9-l7(e) as another code fragment thai is logically equivalent to Fig. 9-l7(a). A piece of code that can mutate a sequence of machine instructions without changing its functionary is called a mutation engine, and sophisticated viruses contain them to mutate the decryptor from copy to copy. The mutaiion engine itself can be hidden by encrypting it along with the body of the virus. Asking the poor anlivirus software to realize lhai Fig. 9-]7(a) through Fig. 9- I7(e) are all functionally equivalent is asking a lot. especially if the mutation engme has many tricks up its sleeve The antivirus software can analyze the code to see what it does, and it can even iry to simulate the operation of ihe code, but remember i(may have thousands of viruses and thousands of files to analyze so it does not have mueh time per test or it will run horribly slowly.

632 SECURITY CHAP 9 As an aside. Ihe store into the variable Y was thrown in just to make it harder to detea the fact that the code relatod to R5 is dead code, that is, does not do any- thing. If other code fragments read and wrile K the code will look perfeclly legi- legitimate. A well-written mutation engine thai generates good polymorphic code c;m give anlivirus soilware writers nightmares The only bright side is that such an engine is hard to write, so Virgil's friends all use his etide. which means (here are not so many different ones in circulation—-yet. So tar we have talked about just trying to recognize viruses in injected exe- executable files In addilion, the antivirus scanner has to check the MBR, boot sec- lors. had sector list, (lash ROM, CMOS memory, clc , but whal if there is a memory-resident virus currently running" That will not he delected. Worse yei. suppose the running virus is monitoring all system calls. It can easily delect (hat the amivirus program is reading (he bool sector (to check for viruses). To Ihwart the anlivirus program, ihe virus does not make the system call Instead it just returns the true boot sector from its' hiding place in the bad block list (t also makes a mental note to reintect all the files when the virus scanner is linishcd. To prevent hemg spooled by a virus, the antivirus program c-ould make hard reads to the disk, bypassing the operating system However [his requires having built-in device drivers (or (DE. SCSI,

and other common disks, rruking the antivirus program less portable and suhject to failure on computers with unusual disks. Furthermore, since bypassing the operating syslem to read the boot sector is possible, bul bypassing it to read all the executable files is not. ihere is also some danger that the virus can produce fraudulent data about executable files as well Integrity Checkers A completely differenl approach to vmis detection is integrity checking. An anfivirus program that woiks (his way first scans the hard disk for viruses- Once u is convinced that the disk is clean, it computes a checksum for each executahle file and write the h t o(checksums for all the relc-vanl files m a directory to a file, checkium, m lhat directory. The next time it runs, il recomputes all the checksums and sees if thev match whal is in the file ,-hecksum. An infected file will show up immediatLh The trouble is Virgil is no! going to take this lying down. He can write a vims thai removes Ihe checksum file Worse yei, he can write a virus that computes (he checksum of (he infected file and replaces the old entry m the checksum file. To protect against this kind of behavior, the diitmrus program can try |(i hide ihe checksum ti]e, but thai ls not likely to work since Virgil can sludy ihe ;inlivirus program c-arefudy before writing (he virus. A better idea is to encrypt 11 to make tampering easier to detect. Ideally, (he encryption should involve use ot a .smart card with an externally stored key that programs cannot get at.

SEC 9 5 ATTACKS FROM OUTSIDE THE SYSTEM 633 Behavioral Checkers A third strategy used hy antivirus software is behavioral checking With this approach, the antivirus program lives in memory while the computer is running and catches all system calls itself. The idea is that it can then monitor all activity and try to catch anything that looks suspicious For example, no normal program should attempt to overwrite Ihe boot sector, so an attempt to do so is almost cer- certainly due to a virus Likewise, changing the flasli ROM i_s highly suspicious. But there are ako cases that are less clear cul For example, overwriting an executable file is a peculiar thing to do—unless you are a compiler. If the antivirus software detects such a write and issues a warning, hopefully the user knows whether overwriting an executable makes sense in the context of ihe current work Similarly, Word overwriting a .doc file with a new document full of macros is not necessarily the work of a virus. In Windows, programs can detach from their executable file and go memory resident using a special system call Again, this might be legitimate, but a warning might still be useful. Viruses do not have to passively lie around

waning tor an antivirus program to kill them, like cattle being led off to slaughter. They can fight back A panicu- larly interesting battle can occur if a memory-resident virus and a memory- resident antivirus meet up on the same computer Yeans ago there was a game called Core Wars in which two programmers faced off by each dropping a pro- program into an empty address space. The programs took turns probing memory, with the object of ihe game being to locate and wipe out your opponent hefore he wiped you out. The virus-antivirus confrontation looks a little like- that, only the battlefield is the machine of some poor user who does not really want U to happen there. Worse yet, the virus has an advantage hecaiise its writer can find out a lor about the antivirus program by just huying a copy of it Of course, once the virus is out there, the antivirus team can modify their program, forcing Virgil to go buy a new copy. Virus Avoidance Kvery good story needs a moral The moral of this one is Better safe than sorry. Avoiding viruses in the first place is a kit easier than trying to track" them down once they have infected a computer. Below are a few guidelines !<ir individual users, but a!so some things that the industry as a whole can do to reduce the proh- lem considerably. What can users do to avoid a virus infection? First, choose an operating sys- system that offers a high degree of security, with a strong kernel-user mode boundary and separate login passwords for each user and the system administrator. Under these conditions, a virus that somehow sneaks in cannot infect the system binaries'.

634 SECURITY CHAP 9 Second, install only shrink-wrapped software hought trom a reliable manufac- manufacturer. Even this is no guarantee since there have heen case* whets disgruntled employees have shpped viruses onto a commercial software product, but it helps a lot. Downloading software from Web sites and bulletin boards is nsky hehavior. Third, buy a good antivirus software package and use il as directed. Be sure to get regular updates from the manufacturer's Web site. Fourth, do not click on attachments to email and tell people not to send them 10 you. Email sent as plain ASCII text is always sate but attachments can start viruses when opened Fifth, make frequent backups of key files onto an external medium, such as floppy disk, CD-recordable, or tape. Keep several generations of each file on a series of backup media. That way. if you discover a virus, you may have a chance to restore files a.s they were before they were infected. Restoring yesterday's intecied file does not help, but restoring last Week's version might. The industry should also take the virus threat seriously and change some dangerous

practices. First, make simple operating systems. The more bells and whistles there are, the more security holes there are. That is a fact of lite. Second, forget active content. From a security point of view, it is a disaster. Viewing a document someone sends you should not require your running their program. JPEG files, for example, do not contain programs, and thus cannot con- contain viruses. All documents should work like that. Third, there should be a way to .selectively write protect specified disk cylinders to prevent viruses from infecting tht programs on them. This protection could be implemented by having a bitmap inside the controller listing the write protected cylinders. The map should only be alterable when the user has flipped a mechanical toggle switch on the computer's front panel. Fourth, flash ROM is a nice idea, but it should only be tnodifiahle when an external toggle switch has been flipped, something that will only happen when the user is consciously installing a BIOS update Of course, none of this will be taken seriously until a really big virus hits For example, one that hit the financial world and reset all bank accounts to 0. Of course, by then it would be too late. Recovery from a Virus Attack When a virus is detected, the computer should be halted immediately smoe j memory-resident virus may still be running. The computer should be rebooted from a CD-ROM or floppy disk that has always been write protected, and which contains the full operating system to bypass the hoot sector, hard disk eopy ot the operating system, and disk drivers, all of which may now he infected. Then an antivirus program should be run from its original CD-ROM, since the hard disk version may also he infected. The antivtrus program may detect some viruses and may even be able to elim- eliminate them, but there is no guarantee that it wjll get them all. Probably the safest

 course ot action at this point i.s to save all filo-, that cannot contain viruses (like ASCII and JPEG files). Those files that might contain viruses (like Word files) should be-converted to another format that cannot contain viruses, such as flat ASCII text (or at least the macros should be removed). AH the saved files should he saved on an external medium. Then the hard disk should be reformatted using a format program taken from a write-protected floppy disk or a CD-ROM to insure thai it itself is not infected It is especially imponant that'the MBR and boot sectors art; also fully er.ised. Then the operating system should be reinstalled trom the original CD-ROM. When dealing with MfUs infections, paranoia is your best friend. " 9.S.S The Internet Worm The first large-scale

internet computer security violation began in the evening ot Nov 2. 1988 when a Cornell graduate student. Rohen Tappan Morns, released a worm program into the Internet. This action brought down thousands ot com- computers at universities, corporations, and government laboratories ail over the world before it was tracked down and removed It also started a controversy that has nat yet died down. We will discuss the highlights of this event helow l-"or rnoie technical information see (Spafford. 1989) r-or tbe story viewed as a police thriller, see (Hafner and Markoif, 199! j. The story began sometime in 1988 when Morris discovered two bugs in Berkeley UNIX ihat made it possible to gain unauthorized access to machines all over the Internet. Working alone, he wrote a self replicating program, called a worm, that would exploit these errors and replicate itself in seconds on every machine it could gain access to. He worked on the program for months, carefully tuning it and having it try to hide its tracks 11 is not known whether the release on Nov. 2. 1988 was intended as a test, or was tile real thing. In any event, it did bring most ot the Sun and VAX systems on the Internet to their knees within a few hours of its release. Morris" motivation is unknown, but tt i.s possihle that he intended the whole idea as a higli-tech practical joke, hut which due to a programming error got completely out oi hand Technically, the worm consisted of two programs, the bootstrap and the worm proper. The bootstrap was 99 lines of C called // c It was compiled and exe- executed on the system under attack. Once running, it connected to the machine fioni which it came, uploaded the main woim and executed it After going to some trouble to hide its, existence, the worm then looked through its new hosts muting tables to see what machines that host was connected tit and attempted lo spread ihe bootstrap to those machines. Three methods were tried lo infeci new machines. Metliod 1 svas lo try to run a remoie shell using the /-th command. Some machines mis' other machines, and just run rsh without any further authentication. It this worked, the remote shell uploaded the worm program and continued infecting new machines from there.

Method 2 made use of a program present on all BSD systems called finger that allows a user anywhere on the Internet to type linger name@site to display information about a person at a particular installation This information usually includes the person's real name, login, home and work addresses and tele- telephone numbers, secretary's, name and telephone number, FAX number, and simi- similar information It is the electronic equivalent of the phone book.

Finger works as follows. At every BSD site a background process ca!!ed the finger daemon runs all the time fielding and answering queries from all over the Internet What the woiin did was call finger with a specially handcrafted 536- byte string at, paiameter This long stnng overflowed the daemon's buffer and overwrote its, stack, the way shown in Fig. 9-11 (c). The bug exploited here was the daemon's failure to check for overflow When the daemon returned from the procedure it was in at the time it got the request, it returned not to main, but to a procedure mside the 536-bytc string on the stack This procedure tried to execute sh. It it worked, the worm now had a shell running on the machine under attack- Method 3 depended on a bug in the mail system, \endmaii. which allowed the worm to mail a copy of the bootstrap and get it executed Once established, the worm tried to break user passwords,. Morris did not have to do much research on how to accomplish this. All he had to do was ask his father, a security expert at the National Security Agency, the U.S. government's code breaking agency, for a reprint of a classic paper on the subject that Morris, Sr. and Ken Thompson wrote a decade earlier at Bell Labs (Moms and Thomp- Thompson, 1979). Each broken password allowed the worm to log m on any machines the password's owner h;id accounts on. Every time the worm gained access to a new machine, it checked to sue if dny other copies of the worm were already active there If so, the new copy exiled, except one time in seven it kept going, possibly in an attempt to keep the worm propagating even if the system administrator there started up his own version at the worm to tool the real worm. The use of 1 in 7 created far tao many worms, and was the reason all the infected machines ground to a halt- they were infested with worms. If Moms had left this out and just edited whenever anothei worm was sighted, the worm would probably have gane undetected. Moms was caught when one of his hu-nds spoke with the New York Times computer reporter, John Markoff. and tried to convince Markoft thai the incident was an accident, the w(irm was harmless, and the author was sorry The friend inadvertently let slip that the perpetrator's login was rtm. Converting rrm into the owner's name was easy—all that Markoff had to do was to run fmgei The next day the story was the lead on page one. even upstaging the presidenti.il election three days later. Moms was tried and convicted in federal court. He was sentenced to a fine of $10,000, 3 years probation, and 400 hours of community service. His legal costs

$150,000. This sentence generated a great deal or contro- controversy. Many in the computer community felt that he was a bright graduate stu- student whose harmless prank had gotten out of control. Nothing m the worm sug- suggested that Morris was trying to steal or damage anything. Others felt he was a serious criminal and should have gone to jail. One permanent effect of this incident was the establishment of CERT (Com- (Computer Emergency Response Team), which provides a central place to report break-in attempts, and a group of experts to analyze security problems and design fixes While this action was certainly a step forward, it also has its downside. CERT collects information about system flaws that can be attacked and how to hx them. Of necessity, it circulates this information widely to thousands of system administrators on the Internet. Unfortunately, the bad guys (possibly posing as system administrators) may also be able to get bug reports and exploit the loo- loopholes, in the hours, (or even days) before they arc closed 9.5.6 Mobile Code Viruses and worms are programs (hat get onto a computer without the owner's knowledge and against the owner's wilt. Sometimes, however, people moie-or- Icss intentionally import and run foreign code on their machines. It usually hap- happens tike this. In the distant past (which, in the Internet world, means last year), most Web pagL\ were just static HTML files With a few associated images. Nowadays, increasingly many Web pages contain smalt programs called applets. When a Web page containing applets is downloaded, the applets are fetched and executed. For example, an applet might contain a form to be fitted out, plus interactive help in fitting it out. When the form is filled out. it could he sent somewhere over the Internet for processing Tax forms, customized product oider forms, and many other kinds of forms could benefit from this approach Another example in which programs arc shipped from one machine to another for execution on the destination machine are agents. These are programs that are launched by a user to perform some task anil then report back. For example, an agent could be asked to check out some travel Web sites to find the cheapest flight from Amsterdam to San Francisco. Upon arriving at each site, the agent would run there, get the information it needs, then move on to the uext Weh site. When it was all done, it could come back home and report what it had learned A third example ot mobile code is a PostScript file that is to be printed on a PostScript printer. A PostScript file is actually a program in the PostScript pro- programming language that is executed inside fhe printer. If normally tells ihe prinfer to draw certain curves and then fit! them m, but it can do anything else it wants to as welt Applets, agents, and PostScript file are jusf

fhree examples of mobile code, hut there are many others Given Ihe tone discussion about viruses and worms earlier, it should be clear that allowing foreign code to run on your machine m more than a wee bit risky.

638 SECURITY CHAP 9 Nevertheless some people do want to run these foreign programs, thus the ques- question arises: "Can mobile code be run safely"? The short answer l8: "Yes. but not easily." The fundamental problem is tbat when a process imports an applet or otber mobile code into its address space and runs it. that code is running a* part of ,1 valid user process and has all tbe power that user has. including the ability to read, write, erase or encrypt the user's disk tiles, email data to rar-away countries. Long ago, operattng systems developed the process concept to build walls between users. The jdea is that each process has its own protected address space and own UID, allowing it to touch files and other resources belonging to it. but not to other users. Pot providing protection against one part of tbe process (the applet) and the rest, the process concept does not help. Threads allow multiple threads of control wtthin a process, but do nothing to protect one thread against In tbeory, running each applet as a separate process helps a tittle, but is often infeasible. For example, a Web page may contain two or more applets that interact with each other and with the data on the Weh page. The Web browser may also need to interact with the applets, starting and stopping them, feeding (hem data, and so on. If each applet is put m its own process, tbe whole tbing will not work. Furthermore, putting an applet in its own address space does not make it any harder for the applet to steal or damage data. If anything, u is easier since nobody is watching m there. Various new methods ot dealing with applets (and mobile code in general) have been proposed and implemented. Below we will look at three of these methods, sandboxjng, interpretation, and code signing. Each one has Its own strengths and weaknesses. Sandboxlng The first method, called sandboxing, attempts to unifine each applet to a lim- limited range of virtual addresses enforced at run time (Wahbe eta!.. 1993) It works by dividing the virtual address space up into equal-size regions, whtcb we will call .sandboxes. Eacb sandbox must have the property that all ot us addresses share some string of htgh-order bits. For a 32-bit addiess space, W'e could divide it up into 256 sandboxes on 16-MB boundaries so a" addresses witbin a sattdbox had a common upper 8 bits. Equally well, we could have 512 sandboxes on &-MB houndanes. with each sandbox having a 9-bit address prefix The sandbox sue should be chosen to be large enougb to hold the largest applet without

wasting tot, much virtual address space. Physical memory is not an issue if demand paging is present, as it usually is Each applet i.s given two sandboxes, one for the code and one for tbe data, as illustrated in for the case of 16 sandboxes of 16 MB each Fig. 9-18(a). The basic idea behind a sandbox is to guaraniee that an applet cannot jump io code outside its code sandbox or reference data outside its data sandbox. The rca-

ATTACKS FROM OUTStDE THE SYSTEM 224 192 160 128 32 Figur check lei Mon Code Data Cod< 9-18. 7 I la) checking system y Applet 2 J- Applet 1 Mcmorv d uclion for vd vided in id»y CMPS1.S2 TRAPNE JMP(R1) t(S MB s.jn(tb(>xes (b) Oni son for having two sandboxes it. lo prevent an applet from modifying Us code dur- during execution to get around ibese restrictions. By preventing all stores into the code sandbox, we eliminate the danger of self-modifying code. As long as an applet is confined this way, it cannot damage the browser or olber applets, plant viruses in memory, or otherwise do any damage to memory. As soon as an applet is loaded, it is relocated to begin at Ihe start of its sand- sandbox. Then checks are made to see if code and data references are confined to the appropriate sandbox. In the discussion helow. we will just look at code references (i.e., JMP and CALL instructions), but the same story holds tor data references as well. Static JMP instructions that use direct addressing are easy to check: does Ihe target address land within the boundaries of the code sandbox'1 Similarly, relative JMPs are also easy to check. If the applet has code that tries to leave the code sandbox, it is rejected and not executed. Similarly, attempts to touch data outside the data sandbox cause the applet to be rejected. The hard part is dynamic JMPs Most machines have an instruction in which Ihe address to jump to is computed at run time, put in a register, and then jumped to indirectly, for example by JMP {R1) to jump to the address held in register K The validity of such instructions must be checked at run time. This is done by inserting code directly before the indirect jump to test Ihe target address. An example of such a test is shown in Fig. 9-18(b). Remember that all valid addresses have the same upper it bits, so this prefix can be stored in a scratch register, say S2. Such a register cannoi be used by the applei itself, which may require rewriling tl to avoid this regisier.

640 SECURITY CHAP 9 The code works as follows: First the target address under inspection is copied to a scratch register, S1. Then this register i.s shifted right precisely the correct number of bits to isolate the common prefix

in S1 Next the isolated prefix is compared to the correct prefix initially loaded into S2. If they do not match, a trap occurs and the applet is killed. Tins code sequence requires four instructions and two scratch registers. Patching the binary program during execution requires some work, but it is doahle. It would be simpler if the applet were presented in source form and then compiled locally using a trusted compiler that automatically checked the static addresses and inserted eode to verify the dynamic ones during execution. Either way. there is some run-time overhead associated with the dynamic checks. Wahbe et al. A993) have measured this as about 4%, which is generally accept- acceptable A second problem that must be solved is what happens when an applet tries to make a system call') The solution here is straightforward. The system call instruction is replaced by a call to a special module called a reference monitor on the same pass that the dynamic address checks are inserted (or, if the source code is available, by linking with a special library that calls the reference monitor instead of making system calls). Either way, the reference monitor examines each attempted call and decides if it Is safe to perform. If the call is deemed accept- acceptable, such as writing a temporary file in a designated scratch directory the tall is allowed to proceed. If the call is known to be dangerous or the reference monitor cannot tell, the applet is killed. If the reference monitor can tell whith applet called it- a single reference monitor somewhere in memory can handle the requests from all applets. The reference monitor normally learns about the per- lnterp relation The second way to mn untru.sted applets is to run them interpret!vely and not let them get actual control of the hardware This is, the approach used by Web browsers. Web page applets are commonly written in Java, which is ;i normal programming language, or in a high-level scripting language such as safe-TCL or Javascript. Java applets are first compiled to a virtual stack-oriented machine language called JVM (Java Virtual Machine) It is these JVM applets that are put on the Web page. When they are downloaded, they are inserted into a JVM interpreter inside the browser as illustrated in h'ig. 9-19 The advantage of running interpreted code over compiled code, is that every instruction is examined by the interpreter before heing executed This <?tves the interpreter the opportunity to check if the address is valid. In addition, system calls are also caught and interpreted. How these calls are handled is a matter of the security policy. For example, it an applet is trusted (e.g.. it came from the local disk), its system calls could be carried out without question. However, if an

ATTACKS FROM OUTSIDF THB SYSThM Virtual address space Figure 9-19. ApP'ets n be interpreted by <i Wct> hm^scr r the Inlernel), ii could be pui m \ 'hal r appiel is not trusted (e.g.. it came in o effectively a sandbox to resirie-t its behavi High-level scripting languages can also be interpreted. Here no machine addresses are used, so there is no danger of a scrip! trying lo access memory in an impermissible way The downside of interpret a (ion in general is thai it is very -slow compared to running native compiled code Code Yel another way to deal with appiel security is lo know where they came from and only accept applets from trusied sources. With this anproach. a user can maintain a list of irusled applet vendors and only run applets from [hose vendors. Applets from all other sources are rejected as too dicey. In this, approach, no actual security mechanisms, are present at run time. Applets from trustworthy vendors, are run as is and code from other vendors is not run at all or in ,i restricted way (sandboxed or interpreted with little or no access to user tiles and other sys- system resource^) To make this scheme work, as a minimum, there has to be a way lor a user to determine that an applet was written by a trustworthy vendor and not modified hy anyone after being produced. This is done u*;ing a digital signature, which allows the vendor to sign the applet in such a way that future modifications can be detected. Code signing is hased on public-key cryptography. An applet vendor, typi- typically a software company, generates a (public key. private key) pair, making the former public and zealously guarding ihe latter. To sign an applei, the vendor first computes a hash function of the applet to get a 128-bit or 160-bit number, depend- depending on whether MD5 or SHA is used- It then signs the hash value by encrypting it with its private key (actually, decrypting it using the notation of Fig 9-3), This signature accompanies ihe appiel wherever 11 goes.

642 SECURITY CHAP ' When the user gets the apptet, the browser computes (he hash function itself. It then decrypts the accompanying signature using the vendor's public ke> and compares what the vendor claims the hash function is with what the browser itself computed. If they agree, the applet is accepted as genuine. Otherwise it is rejected as a forgery. The mathematics involved makes it exceedingly difficult for anyone to tamper with the applet in such a way as its hash function will match the hash function that is obtained by decrypting the genuine signature. It is equally difficult to generate a. new false signature that matches without having ihe private key. The process of signing and verifying is illustrated m Fig. 9-20. Signature genera [3 I SipnatunTl H2 = decrypt!

Signature) Figure 9-20. Hou cmle signing works 9.5.7 Java Security The Java programming ianguage and ;tccotnpanying run-time sysiem were designed to allow a program io be wntien and compiled once and then shipped over the Iniernet in binary form and run on any machine supporting Java. Secu- riiy was a part of the Java design from the beginning In this section we will describe how ii works Java is a iype^afe language, meaning ihat ihe compiler will reject any attempt to use a variable in a way noi compatible with its t>pe. In conirast. consider the following C code: naughty_.func() { char *p; p = rand(); it generates a random number and stores it in the pointer p. Then it siores a 0 byie at ihe address coniained in p. overwriting whatever was there, code or data. In

SEC 9 "> ATTACKS FROM OUTSinF THE SYSTEM 643 Java, constructions that mix types like this are forbidden by the grammar. In addi- addition, Java has no pointer variables, casts, user-controlled storage allocation (such as malloc and free) and all array references are checked at run i,me Java programs are compiled to an intermediate binary code called JVM (Java Virtual Machine) byte code. JVM has aboui 100 instructions, mosl of which push cibiects of a specific type onto (he stack, pop [hem from the slack, or com- combine two items on the stack arithmetically. These JVM programs are typically interpreted, although in some cases they can he compiled into machine language for faster execution. In the Java model, applets sent over the Internet for remote execution are JVM programs. When an applet arrives, it is run through a JVM byte ct,dc verifier that checks if the applet obeys certain rules A properly compiled appiet will automatically obey them, but there is nothing to prevent a malicious user from writing a JVM applet in JVM assembly language. The checks include 1 Does the applet attempt to forge pointers' 2. Does it violate access restrictions on private class members? 3. Does it try to use a variable of one type as another type? 4. Does it generate stack overflows or underflows? 5. Docs it illegally convert variables ol one type to another? If the applet passes all the tests, it can be salcly run without tear that it will access memory other than its own. However, applets can still make system calls by calling Java methods (pro- (procedures) provided for that purpose. The way Java deals with that has evulved Over time. In the first version of Java, JDK (Java Development Kit) 1.0. applets were divided into two classes: trusted and untrusted. Applets fetched from the iocai disk were trusted and allowed to make any system calls they wanted. In contrast, applets

fetched over the Internet were untrusted. They were run in a sandhox, as shown in Fig. 9-19. and allowed to do practically nothing. After some experience with this model, Sun decided that it was too restrictive In JDK I I, code signing was employed. When an applet arrived over the Inter- Internet, a check was made lo see if ii was signed by a person or organization the user trusted (as defined by the user's list o! trusted signers). If so. the applet was allowed to do whatever it wanted. I! not. it was run in a sandbox and severely res- tricied. After more experience, this proved unsatisfactory as well, so the security model was changed again JDK 1.2 provides a configurable tine-gram security policy (hat applies to all applets, both local and remote. The security model is complicated enough that an entire book can be written describing it (Gong. 1999), so we will just briefly summarize some of the highlighis.

644 SECURITY CHAP 9 Each applet is. characterized by two things: where ii came from and who signed it. Where it came from is Us URL: who signed it is which private key was used for ihe signature. Each user can Create a security polk\ i_on isttng of a list or rules A rule may list a URL. a signer, an object, and an action that the applet may perform on the object if the applet's URL and signer match ihe rule. Con- Conceptually, the information provided is shown in the tahle of Fig- 9-21, although the actual formaiiing is different and is related to ihe Java class hierarchy. | URL ^Signer Object^ ____^^5."?n I www taxprep com i TaxPrep j /usr/susan/1Q40.xis ' Read I I - I "T/uSr/Imp/- [Read, Write _j [wwwmicrosofl.com] Microsoft | /usr/susan/Office/- | Read, Write, peietel Figure 9-21. Some eumpiev oi priiictnon th,it tan lit specified wiih JDK I 1 One kind of action permits file access. The action can specify a specific file iir directory, the set of all files in a given directory, or ihe set ot all files and direc- directories recursively contained in a given directory The three lines of Fig 9-21 correspond to these three cases. In the first line, the user, Susan, has set up her permissions file so thai appleis ongmaiing ai her tax preparer's machine, u'liit taxprepxom and signed by the company, have read access to her (ax daia locaied in the file 1040 xh. This is the only file they can re jd and no other applets can read this file. In addition, all appleis from all sources, whether signed or not. can read and write files vn/usr/tmp Furthermore, Susan also irusts Microsoft enough to aliow applets ouginating at its site and signed by Microsoft to read, write, and delete all the files helow the Office directory in the directory tree, for example. ii> fix bugs and insiall new ver sions of the software To verify the signatures, Susan musi either have the neces sary public

keys on her disk or must acquire them dynamically, i'or example in ihe form of a certificate signed by a company she trusts and whose public key she already has Files are not the only resources that can be projected Network access can also be protected. The objects here are specific ports on specific computers A computer is specified by an IP address or DNS name, ports on thai machine are specified hy a range of nutnhers. The possible actions include asking to ci>nneei to ihe remote computer and accepting connections originated by ihe reunite com- puier. In this way. an applet can be given network access, bui resincted to talking only to computers explicitly named in ihe permissions list. Applets may dynami- dynamically load additional code (classes) as needed, hut user-supplied class loaders can precisely control on which machines such classes may originaic Numerous other security features are also present

SEC 9.6 PROTECTION MECHANISMS 645 9.6 PROTECTION MECHANISMS In (he previous sections we have looked at many potential problems, some of (hem technical, some of them not. In (he following sections we will concentrate on some of Ihe deiailed technical ways that are used in operating systems to pro- protect files and other things. All of ihese techniques make a clear distinction he- tween policy (whose data arc to be protected from whom) and mechanism (how the system enforces the policy). The separation ot policy and mechanism is dis- discussed in (Sandhu,]993). Our emphasis will be on mechanisms, not policies in some systems, protection is enforced by a program called a reference monitor. Every time an access to a potentially protected resource is attempted, the system first asks the reference monitor to check its legality. The reference monitor then looks at iis policy tables and makes a decision. Below we will describe the environment in which a reference monitor operates. 9.6.1 Protection Domains A computer system contains many "objects" that need to be protected These objects can be hardware (e.g.. CPUs, memory segments, disk drives, or printers), or they can be software (e.g., processes, fiies. databases, or semaphores) Each object has a unique name by which il is referenced, and a finite set of operations that processes are allowed to carry out on it The read and write opera- operations are appropriate to a file; up and down make sense on a semaphore It is obvious that a way is needed to prohibit processes from accessing objects that they are not authorized to access. Furthermore, this mechanism must also make it possible to restrict processes to a subset of the legal operations when that is needed. For example, process A may be entitled to read, but not write, file F In

order to discuss different protection mechanisms, it k useful to introduce the concept of a domain A domain is a set ot (object, rights) pairs Each pair specifies an object and some subset of the operations that can be performed on il A right in this context means- permission to perform one of the operations Often a domain corresponds to a single user, telling what the user can do and not do. hut a domain can also be more general than just one user. Figure 9-22 shows three domains, showing the objects in each doin;tin and tiie rights fRead, Write. eXecute] available on each object Note that Primer/ is m two domains at the same time. Although not shown in this example, it is possible for the same object to be in multiple domams, with different rights in each one. At every instant of lime, each process runs in some protection domain. In other words, there is some collection of objects it can access, and for each object il has some set of rights. Processes can also switch from domain to domain during execution. The rules for domain switching are highly system dependent. To make the idea of a protection domain more concrete, let us look at UNIX. In UNIX, the domain of a process is defined by its UiD and GlD, Given any

Figure 9-22. Three proleilion dum-iin". <U1D. G1D) combination, it is possible to make a complete lisl of all objects (files, including I/O devices represented by special files, etc I thai can be accessed, and whether they can be accessed for reading, writing, or executing. Two processes with the same (UID. GID) combination wtll have access to exactly Ihe same set ot objects. Processes with different (UiD, GID) values will have access to a dif- different set of files, although there may be considerable overlap in most cases. Furthermore, each process in UNtX has two halves: ihe user part and the ker- kernel p;irt When the process does a system call, it switches from Ihe user part to the kernel part. The kernel part has access lo a different set of objects from the user parl For example, the kernel can access all the pages in physical memory. Ihe entire disk, and all the olhcr protected resources. Thus, a system call causes- a domain switch. When a process does an exec vn a file with Ihe SETUlD or SETG1D bit on. il acquires a new effective UID or GID. With a different (UlD, GiD) combination, it has a different set of tiles and operations available Running a program with SETUlD or SETG1D is also a domain switch, suite the rights available change An important question is how Ihe system keeps hack of which ob[ect belongs to which domain. Conceptually, at least, ane can envision a large matrix, with the rows being domains and Ihe columns being objects. Hach box lists ihe rights, if any. thaf Ihe domain

contains tor ihe object. The matrix foi Fig. 9-22 is shown in Fig. 9-23. Given this matrix and the current domain number, the system can tell if an access lo a given object jn a particular w;iy from a specified domain is allowed. Pnnten Plotted Read Reed Write Head Read Write Execute Read Write Read Execute Write Write Write

SbC 9.6 PROTECTION MFCHANtSMS 647 Domain swrtchmg itself tan be easily included m the matrix model by reatiz- ing that a domain is itself an object, with the operation enter Figure 9-24 shows the malax of Fig. 9-23 again. onl> now with the three domains as objects them- themselves Piocesses in domain t can switch to domain 2, but once there, they cannot go back. This situation models executing ,t SETUID program in UNIX. No other domain switches .ire permitted m this example. Ob|ect I Ptoner2 Domain 1 Oo<na[n2 Dnma«3 Read Write Read Write Bead Write Read Write »,. ««. 9.6.2 Access Control Lists In practice, actually storing the matrix oi Fig. 9-24 is rarely done because it is large and sparse. Most domains have no access ai all to most objects, so storing a very large, mostly empty, matrix is a waste of disk space. Two mcihods that are practical, however, are storing the matrix by rows or by columns, and then storing only the nonempty elements The two approaches are surprisingly different. In this section we will look at storing it by column, in the next one we will study storing it by row. The first technique consists ol assuciaJing with each object an (ordered) list containing all the domains that may access the object, and how. This list is called the Access Control List or ACL and is illustrated in Fig. 9-25 Here we see three processes, each belonging lo a diiiereni domain A, B. and C. and three tiles Fl, F2. and FS. For simplicily, we will assume lhat each domain corresponds to exactly one user, in this ease, users A. 8. and C. Often in the securuy literature, the users arc called suhjects or principals, to contrasi them with the things owned, the objects, such as files. Each file has an ACL associated with tt. File // has iwo entries in its AC'L (separated by a scniicoloo) The first entry says that any process owned hy usei A may re;id and write the fitt The second entry says tha' any process owned by user B may read the file All other accesses by these users and all accesses by oiher users arc forbidden. Note that the rights are granted by user, not by process.

Figi write file Fl. It docs not matter if there is one such process or 100 of them, it is the owner, not the process ID, that matters File F2 has three entries in its

ACL: A. B. and C can all read the file, and in addition B can also write t|. No other accesses are allowed File F3 is apparently an executable program, since B and C can both read and execute it. B can also write 11. This example illustrates the most basic form of protection with ACLs, More sophisticated systems are often used in practice. To start with, we have only shown three rights so far: read, write, and execute. There may be additional rights as well. Some of these may be generic, that is. apply to all objects, and some may be object specific Examples of generic rights are destroy obiecl and copy object. These could hold for any object, no matter what type it is. Object-specific rights might include append message for a mailbox object and sort alphabetical for a directory object. So far. our ACL entries have been for individual users. Many systems sup- support the concept of a group of users. Groups have names and can be included in ACLs. Two variations on the semantics of groups are possible.]n some systems, each process has a oser ID (UlD) and group ID (GlD). In such systems, an ACL entry contains entries of the form UiDt, G'Dt rightst, IJID2, GID2: nghts2,.. Under these conditions, when a request is made to access, an object, a check is made using the caller's UlD and GlD. if they are present in the ACL the rights listed are available. If the (UlD, GID) combination is not in the list, the access is not permitted. Using groups this way effectively introduces the concept of a role. Consider an installation in which Tana is system administrator, and thus in the group wsadm However, suppose that the company also has some clubs for employees

SEC 96 PROTECTION MECHANISMS 649 and Tana is a member of the pigeon fanciers dub Club members belting to the group pig/an and have access to the company's computers for managing thetr pigeon database. A portion of the ACL might be as shown in Fig. 9-26. Access control list lana, sysadm RW | Pigeon data ^ bill, pigfan. RW, lana, pigfan RW; Figure 9-26. Tvm 4i.ces!> control fisls]f T;ina fries lo access one of these flics, (he result depends on which group she is currently logged in us When she logs in. the system may ask her to choose wh>ch of her groups she is currently using, or there might even be different login names and/or passwords to keep them separate The pomt of this scheme is to prevent Tana from accessing the password file when she currently has her pigeon fancier's hai on. She can only do that when logged in as the system administrator. In some cases, a user may have access to certain files independent of which group she is currently logged in as. That case can be handled by mtroducing wildcards, which mean everyone. For example, the

entry tana, •: RW for the password file would give Tana access no matter which group she w;is currently inas. Yet another possibihty >s that >f a user belongs to any ot the groups that have certain access rights, the access is permitted. In lbs case, a user belonging to multiple groups does not have to specify which group to use at login time All ot them count all of the time. A disadvantage of th>s approach k that n provides les1- encapsulation- Tana can edit the password Ble during a p>geon club meet nig The use of groups and wildcards introduces the possib>lity of selectively blocking a specific user from accessing a file. For example, lhe entry virgil, •: (none); *, •: RW gives the entire world except for Virgil read and write access to the file This works' because the entries are scanned in order, and the first one that applies is taken; subsequent entries are not even examined. A match is found lor Virgil un the first entry and the access rights, in this casc. (none) are found and applied The search is terminated at that point The fact that the rest of the wt>rld has access is never even seen. The other way of dealing with groups is not tu have ACL entries consist ol (U1D, G1D) pairs, but to have each entry be a U1D or a G1D. For example, an entry for the file pigeon, darn could be debbie: RW; phi]: RW; pigfan; RW

650 SECURITY C'HAP "] members of the pigfan group have read and meaning that Debbie and Phil, and a write access to the file. It sometimes occurs that a user or a group ha>> certain permissions with respect to a file that the file owner later w,shes to revoke. With access control lists, it is relatively straightforward to revoke a previously granted access. All that has to be done is edit the ACL to make the change. However, if the ACL is; checked only when a lile is opened, most likely the change will only take effect on future calls to open. Any file that is already open will continue to have the rights it had when H was opened, even it the user is no longer authorized to access the file at all. 9.6.3 Capabilities The other way of slicing up the matrix of Fig ()-24 is by rows. When this method is used. associated wiih each process i-s ;i list of objects that may be accessed, along with an indication of which operations are permitted on each, m other words, its domain. This list is called a capability list or C-list and the indi- individual items on it are called capabilities (Dennis and Van Horn. 1966, Fabry.]974). A set of three processes and their capability lists is shown in Fig 9-27 Ownet ~x? e | F21 1 F2 R 1 FIR F2RW F3RWX 0 rF^rT~| | F3 RX | \ C^ISt Figure 9-27. When,:: Igl.t Each capability grants the owner «i.Um , ,e,, 27. the process owned by

user A can read files Fl capability consists of a |i|e (or more generally, .1. tor the various rights In a UNIX-like system, the the i-node number Capability lists are thenise] ■\,.^»» x~*i*~— i_.i:*.. i . *i .. i- ind F2, foi object. In Fig. 9- _, .-, .. .ample Usually, a object) identifier and a bitmap from other capability lists, thus facilitating shiinug of Cubd It is fairly obvious that capability lists must be protected Three methods ot protecting them are known The first w architecture, a hardware design in which each bit that tells whether ihe word com Jentifier wiiiild probably be >bjects and may be pointed to jrn user tampering require;, a tagged ;mory word has an extra (or tag) iipability or not. The tag bit is not used

SEC 9 6 PROTECTION MECHANISMS 651 by arithmetic comparison, or similar ordinary instructions, and it can be modified only by programs running in kernel mode (i.e.. the operating system). Tagged- architecture machines have been built and tan be made to work well (Feudal, 1972)- The IBM AS/400 is a popular example. The second way is tr> keep the C-list inside the operating system. Capabilities are then referred to by their position in the capability hst. A process might say: -Read I KB from the tile pointed to by capability 2." This form of addressing is similar to using file descriptors in UNIX. Hydra (Wulf et al., 1974) worked this The third way is to keep the C-list m user space, but manage the capabilities cryptographically so that users cannot tamper with them. This approach is partic- particularly suited to distributed systems and works as follows- When a client process vends a message to a remote server, for example, a file server, to create an object for it. the server creates the object and generates a long random number, the chock field, to go with it. A slot in the server's file table is reserved for the object and the check field is stored there along with the addresses of the disk blocks, etc. In UN|X terms, the check field is Stored on the server in the i-node. It is not sent back to the user and never put on the network. The server then generates ;ind returns a capability to the user of the form shown in Fig 9-28 l(Obj sets. Rig hts, Chec Figure 9-28. A ,rryplo£r,ipiili.<ill>-prole.? led c.ipdhilily The capability returned to the user contains the server's identifier, the object number (the index into (he server's tables, essentially, the i-node number), and the rights, stored as a bitmap. For a newly created object, all the rights bits are turned on. The last field consists of the concatenation of the object, rights, and chrck field run through a cryptographically-secure one-way function. (, of the kind we discussed earlier. When the user wishes to access the object, it sends the

capability to the server as part of the request. The server then extracts the ohject number to index into its tables to find the object. It then computes /'(Object, Rig/iri, Check) taking ihe first two parameters from the capability itself and the third one from its own tables. If the result agrees with the fourth field in the capability, the request is honored; otherwise, it is rejected. If a user tries to access someone else's ohject. he will not be able to fabricate the fourth field correctly since he does nof know the check field, and the request will be rejected. A user can ask the server to produce a weaker cupahility. for example. Un- Unread-only access- First the server verifies that the capability is valid. If so, if computes f (Object.New^ngbts.Check) and generates a new capability putting

652 SECURITY CHAP 9 this value in the fourth Held. Note that the original Che(k value is used hecause other outstanding capabilities depend on it. This new capability is sent back to the requesting process. The user can now give tins to a friend by)u« sending it in a message if the friend turns on rights hits that should he oft, the server will detect this when the capability is used since the J value will not correspond to the false rights field. Since the friend does not know the true check field, he cannot fabricate a capahility that corresponds to the false rights bits. This scheme was developed for the Amoeba system and used extensively there (Tanenbaum et a]., 1990). In addition to the specific object-dependent rights, such as read and execute, capahiliiies (both kernel and cryptograph] c-a] I y-pro tec ted) usually have generic rights which are applicable to all objects. Examples of generic rights are 1. Copy capability: create a new capahility for the same object. 2. Copy object: create a duplicate object with a new capability. 3. Remove capability, delete an entry from the C-hst; object unaffected. 4. Destroy object: permanently remove an object and a capabilii>. A last remark worth making ahout capahility systems is that revoking access to an object is quite difficult in the kernel-managed version. It is hard for the sys- system to find all the outstanding capabilities for any object to take them back, since they may be stored in C-lists all over the disk One approach is to have each capability point to an indirect object, rather than to the object itself By having (he indirect object point to the real object, the system can always break that con- connection, thus invalidating the capabilities (When a capability to the indirect object is later presented to the system, the user will discover that the indirect object is now pointing to a null object.) In the Amoeba scheme, revocation is easy. All that needs to be clone is, change

the check field stored w!th the object In one blow, all existing capabili- capabilities are invalidated. However, neither scheme allows selective revocation, that is. taking back, say, John's permission, but nohody else's This defect is generally recognized to be a problem with all capability systeml;. Another general problem is making sure the owner of a valid capability does not give a copy to 1000 of his best friends Having the kernel manage capabili- capabilities, as in Hydra, solves this problem, but this solution does not work weil in a dis- distributed system such as Amoeba. On the other hand, capabilities solve the problem of sandboxmg mobile cede very elegantly. When a foreign program is started, it is given a capability list con- containing only those capabilities that the machine owner wants to give it, such as the ability to write on the screen and the abtltt> to read and write files in one scratch directory just created tor it. If the mobile code is put into its own process with only these limited capahiliiies. it will not be able to access any other system re-

SEC. 9.6 PROTECTION MECHANISMS 653 sources and thus be effectively confined to a sandbox without the need to modify its code or run it interpret,vely. Running code with as few access rights as possi- possible is known as the principle of least privilege and ts a powerful guideline for producing secure systems. Briefly summarized, ACLs and capabilities have somewhat complementary properties. Capabilities are very efficient because if a process says "Open the file pointed to by capability 3," no checking is needed. With ACLs, a (potentially long) search of the ACL may be needed. If groups are not supported, then grant- granting everyone read access to a file requires enumerating all users in the ACL. Capabilities also allow a process to be encapsulated easily, whereas ACLs do not. On the other hand. ACLs allow selective revocation of rights, which capabilities do not. Finally, if an object is, removed and the capabilities are not or the capabil- capabilities are removed and an object is not, problems arise. ACLs do not suffer from this problem. 9.7 TRUSTED SYSTEMS Much of this chapter has been devoted to the fact that virtually all modern computer systems leak like a sieve. Estimates of the worldwide damage caused by viruses and similar problems exceed $1 trillion per year in wasted effort to repair problems, reconstruct damaged data, etc., not to mention lost businehs opportunities. A naive person might logically ask two questions concerning this state of affairs: 1. Is it possible to build a secure computer system'1 2. If so, why is it not done? The answer to the first

one is basically yes. How to build a secure system has been known for decades. MULTICS, designed m the 1960s, for example, had security as one of its main goals and achieved that fairly well. Why secure systems are not being built is more complicated, but it comes down to two fundamental reasons. First, current systems are not secure but users are unwilling to throw them out. If Microsoft were to announce that in addition to Windows it had a new product, SecureOS. that was guaranteed to be immune to viruses, but did not run Windows applications, it is far from certain that every per- person and company would drop Windows like a hot potato and buy the new system immediately. The second issue is more subtle. The only way to build a secure system is to keep it simple Features are the enemy of security. System designers believe (rightly or wrongly) that what users want is more features. More features mean more complexity, more code, more bugs, and more security errors.

654 SECURITY HUP 9 Here are two simple examples. The fust email systems sent messages as ASCJJ text. They were completely secure. There is nothing an incoming ASCII message can do to damage a computer system Then people got the idea to expand email to include other types of document, for example, Word files, which can contain programs in macros. Reading such a document means running some- somebody ehe's program on your computer No matter how much sandboxing is used, running a foreign program on your computer is inherently more dangerous than looking ai ASCII text. Did users demand the ability to change email from passive documenfs to active programs? Probably not, but systems designers (nought it would be a nitty idea, without worrying too much about the security implications The second example is the same thing for Web piiges. When ihe Web con- consisted of passive HTML pages, it did not pose a major security problem (allhough illegal HTML could cause a buffer overflow atiack). Now thai many Web pages contain programs (applets) lhat the user has to run 10 view the content, one secu- security leak after another pops up. As soon js one is fixed, jnother one takes its dynamic content" Not thai the auihor remembers, but us introduction bought with it a ialt of security problems. It looks like the Vice-President-In-ClwrgcOt- Saying-No was asleep at the wheel. Actually, there are some organizations that think good security is more impor- important than nifty new teatures. (he military being the prune example. In ihe follow- following sections we will look some of ihe issues involved, but they can be summarized ill one sentence. To build a secure

system, have a security model at the core of the operating system that is simple enough that the designers can actually under- understand it, and resisi all pressure u> deviate from it m order to add new features. 9.7.1 Trusted Computing Base In the security world, people often talk about trusted systems rather than secure systems. These are systems that have formally stated securiiy require- requirements and meet ihe.se requirements. At the hetui of every trusted system is a minimal TCB (Trusted Computing Base) consisting of the hardware and solt- ware necessary for en tore mg all ihe security rules. If the trusted computing base is working to specification, the system security cannot be compromised, no matter what else is wrong. The TCB typically consists of most ot the hardware (except I/O devices that do not affect security), a portion of the operating system kernel, and most or all of the user programs that have siiperuser power (e.g., SETUlD root programs in UNIX). Operating system functions that must be part t>! the TCB include process creation, process sw]tching, memory map management, and part of file and I/O management. In a secure design, often the TCB will he quite separate from the rest of the operating system in order (o minimize its size and verify its correctness.

StC 9.7 TRUSTED SYSTEMS 655 An important part or the TCB is the reference monitor, as shown in Fig. 9-29. The reference monitor accepts all system calls involving security, such as opening files, and decides whether ihey should be processed or not. The reference monitor thus allows all the security decisions to be put in one place, with no possibility of bypassing it. Most operating systems are not designed this way. which is part of the reason they are so insecure. User process All system caHs go through tha reference monitor lor security eftecking '] Reference monitor I T Operating system kernel User space Kernel Figure 9-29. A reference monitor. 9.7.2 Formal Models of Secure Systems Protection matrices, such as that of Fig. 9-23, are not static. They frequently change as new objects are created, old objects are destroyed, and owners decide to increase or restrict the set of users for their ohjects. A considerable amount of attention has been paid lo modeling protection systems in which the protection matrix is constantly changing. In the remainder of this section, we will touch briefly upon some of this work. Decades ago, Harrison et al. A976) identified six primitive operations on the protection matrix that can be used as a hase to model any protection system. These primitive operations are create object, delete object, create domain, delete domain, insert right, and remove right. The two latter primitives insert and remove

rights from specific matrix elements, such as granting domain 1 permis- permission lo read File6. These six primitives can be combined into protection commands. It is these protection commands that user programs can execute to change the matrix They may not execute the primitives directly For example, the system might have a command to create a new file, which would test to see if the file already existed, and if not. create a new object and give the owner all rights to it. There might

i lhe system, in eflect. inserting the "read" light m the new file ntry ny instant, lhe matrix determines what a process in any domain can do, t 11 is authorized lo do The matrix is what is enforced hy the system; ation has 10 do with management poiic> As an example of this distinc- us consider tiie simple system ,>t Fig 9-30 in which domains correspond In Fig 9-Wai we see the intended protection policy. Henry can read e mailbox?. Robert can read and wriie \e<r?t. and all three users can read Objects Compile! Mailbox 7 Head Execute Read Execute Read Read Read Objects Compile! Mailbox 7 Secret Read Execute Read Read Execute Read Write Read Read Write mands to have the matrix changed to Fig. 9-30(b) He has now gained access to madbnx7, something he is not authorized to have If he tries to read it. the operat- operating system will carry out his request because it does not know that the state of Fig. 9-30(b) is unauthon7ed. It should now be clear that the set of all possible matrices can he partitioned into two disjoint sets: the set of all authorized states and the set of all unauthorized states A question around which much theoretical research has revolved is this: "Given an initial authorized state and a set of commands, can i! be proven that the system can never reach an unauthorized state?'1 In effect, we are asking if the available mechanism (the protection commands) is adequate to enforce some protection policy. Given this policy, some initial state of the matrix, and the set of commands for modifying the matrix, what we would like is a way to prove that the system is secure. Such a proof turns out quite difficult to acquire; many general purpose systems are not theoretically secure. Harrison et al. A976) proved that in lhe ca.se of an arbitrary configuration for an arbitrary protection system, security is theoretically undecidable. However, for a specific system, it may be possible to prove whether the system can ever move from an authorized state to an unauthorized state. For more information, see LandwehrA98l).

SEC 9 7 TRUSTED SYSTFVIS 637 9.7.3 Multilevel Security Most operating

sysiems allow individual users to deteimine who may read and write their files and other objects This policy is called discretionary access control. In many environments this model works fine, but there are other porute patent departments, and hospitals In ihe latter environments, the organi7a- tion has stated rules about who can see what, and these may not be modified b> individual soldiers, lawyers, or doctors, at least not without getting special permis- permission from ihe boss These environments need mandatory access controls to ensure that ihe stated security policies arc enforced by the system, in addition to the standard discretionary access controls What these mandatory access lontrols do is regulate the flow of information, to make sure that it does not leak oul in a way ii is not supposed to. The Bell-La Padula Model The most widely used multilevel security model is the Bell-La Padula model so we will start there (Bell and La Padula, 1973). This model was designed lor handling military security, but it is also applicable to other organizations In the military world, documents (objects) can have a security Ie\el. such as unclassi- unclassified, confidential, secret, and top secret. People are also assigned these levels, depending on which documents they are allowed to see A gencial might be allowed to see all documents, whereas a lieutenant might be restricted to docu- documents cleared as confidential and lower A process running on behalf of a user acquires the user's security level. Since there are multiple security levels, this scheme is called a multilevel security system The Bell-La Padula model has rules about how information can flow 1. The simple security property: A process running at security level k Can read only objects at its level or lower. For example, a general can read a lieutenant's documents but a lieutenani cannot read a general's documents. 2, The * property: A process mnning at .security level k can write only objects at its level or higher- For example, a lieutenant can append a message to a general's mailbox telling everything he knows, but a general cannot append a message lo a lieutenant's mailbox telling everything he knows because the general may have seen top secret documents that may no(be disclosed to a lieutenant Roughly summarized, processes can read down and write up, but ni>t the reverse. If the system rigorously enforces these two properties, it can be shown that no information can leak out from a higher security level to a lower one. The *

 property was s,i named because m Ihe original report, the authors could not think of a good name for it and used * as a temporary placeholder until they could dev- devise a better name They never did and the

report was printed with this *. In this model, processes read and write objects, but do not communicate with each other directly The Bell-La P.idula model is illustrated graphically in Fig. 4-31. Figure 9-31. The Bell-La Padula multilevel ■.ecunty model In this figure a (solid) arrow from an object to a process indicates that ihe process i reading ihe object, that is, information is flowing from the object to the proce Similarly, a (dashed) arrow from a process to an object indicates that the process is writing into the object, that is, information Ls flowing from the process to the object Thus all information flows in the direction of the arrows For example, process B can read from object / but not from object 3. The simple security property says that all solid (read) arrows go sideways or up. The * property says that all dashed (write) arrows also go sideways or up. Since information flows only horizontally or upward, any information that starts out at level k can never appear at a lower level. In other words, there is never a path that moves information downward, thus guaranteeing the security of the model. The Blba Model To summarize the Bell-La Padula model in military terms, a lieuteiiam can ask a private lo reveal all he knows and then copy this information into a general's file without violating security Now let us pul the same model in civilian terms. Imagine a company in which janitors have security level I, programmers have

i^CLI C|^. IIUl ^ULII (LIILI.V 111** Iiii"£| 1»_? *'« *.•!*- ^*«*« ■ — c> ■ data, »e need precisely the reverse properties (Biba, 1977) !. The simple integrity principle: A process running at security level k can write only objects at its level or lower (no write up). 2 The integrity * property: A process running at security level k can read only objects at its level ot higher (no read down). Together, these properties ensure thai the programmer can update lhe janitor's files with information acquired from the president, but not \ice versa. Of couise. some organizalions want both lhe Bell-La Paduld properties and the Biba proper- properties but these are in direel conflict so they are hard to achieve simultaneously. 9.7.4 Orange Book Security Given all this background, it should come as no surprise that the U S. Dept. ot Defense has put some effort into the area of secure systems. In particular, in 1985, it published a document formally known as Dept. of Defense standard DoD 5200.28. but usually called the Orange Book on account of its cover, which divides operating systems into seven categories based on their security properties. While the standard has since been replaced by another (and far more complex one), it is Mill a useful guide to some security properties. Also, one occasionally still sees vendor

literature claiming conformance to some Orange Book security level. A table of the Orange Book requirements is given in Fig. 9-32 Below we will look at the security categories and point out some of the highlights. Level D conformance is easy to achieve, it has no security requirements at all. It collects all ihe systems that have failed to pass even the minimum security tests, MS-DOS and Windows 95/98/Me are levei D. Level C is intended for environments with cooperating users. Cl requires a protected mode operating system, authenticated user login, and lhe ability t'oi users to specify which files can be made available to other users, and how (disci e- tionary access control). Minimal security testing and documentation are also required. C2 adds the requirement that discretionary access Control is down to lhe level of ihe individual user. It also requires that ob|ects (e.g., dies, virtual memory pages) given to users must be initialized to all zeros, and a minimal amount of auditing is needed. The UNIX rwx scheme meets Cl but does not meet C2. For this, a more elaborate scheme, such as ACLs or equivalent, are needed.

i Criterion I Security policy Discretionary access control Object reuse Labels Label integrity Exportation of tabeied information Labeling human readable output j Mandatory access control I Subject sensitivity labels | Device labels Accountability Identification and authentication Audit Ttusted path Assurance System architecture System integnty Security testing Design specification and verification Covert channel analysis Trusted facility management Configuration management Trusted recovery Trusted distribution C1 X X C2 X X X B1 -» X X X X X x X B2 -> X _, -t _t X X X X X B3 X _, -, A1 -» _, _t -» _, _, _j -t -» 1 X -> X Documentation Security features user's guide Trusted facility manual Test documentation Design documentation Figure 9-31. Orange Book security cr new requirements here The symbol - nexl lower category also apply here a The symbol X mea leans thai the require The B and A levels require all controlled users and objects to be assigned a secuniy label, such as unclassified, secret, or top secret. The system must be capable of enforcing the Bell-La Padula information flow model. B2 adds to this requirement that the system has been designed top-down in a modular way The design must be presented in such a way [hat it can be verified, The possible covert channels (see the nexi section) must be analyzed.

SEC 9.7 TRUSTED SYSTEMS 661 B3 contains all of B2's features plus there must be ACLs with users and groups, a formal TCB must be presented,

adequate security auditing must be present, and secure crash recovery must be included. A1 requires a formal model of the protection system and a proof that the model is correct, It also requires a demonstration that the implementation con- conforms to the model. Covert channels must be formally analyzed. 9.7.5 Covert Channels All these ideas about formal models and provably secure systems sound great, but do they actually work? In a word: No. Even in a system which has a proper security model underlying it and which has been proven to be secure and is correctly implemented, security leaks can still occur. In this section we discuss how information can stil! leak out even when it has been rigorously proven that such leakage is mathematically impossible. These ideas are due to Lampson A973). Lampson's model was originally formulated in terms of a single timesharing system, but the same ideas can be adapted to LANs and oiher multiuser environ- environments. In the purest form, it involves three processes on some protected machine. The first process is the client, which wants some work performed by the second one, the server. The client and the server do not entirely trust each other. For example, ihe server's job is to help clients with filling out their tax forms. The clients are worried that the server will secretly record their financial data, for example, maintaining a secret list of who earns how much, and then selling the list. The server is worried that the clients will try to steal the valuable tax pro- program. The third process is the collaborator, which is conspiring with ihe server to indeed, steal the client's confidential data. The collaborator and server are typi- typically owned by the same person, These three processes are shown in Fig. 9-33. The object of this exercise is to design a system in which it is impossible for the server process to leak to ihe collaborator process the information that it has legiti- mstely received from ihe client process. Lampson called this the confinement problem, From the system designer's point of view, the goal is to encapsulate or con- confine the server in such a way that it cannot pass information to the collaborator. Using a protection matrix scheme we can easily guarantee that the server cannot communicate with the collaborator by writing a file to which the collaborator has read access. We can probahly also ensure that the server cannot communicate wiih the collaborator using the system's interprocess communication mechanism, Unfortunately, more subtle communication channels may be available. For example, the server can try to communicate a hinary bit stream as follows. To send a ! bit, it computes as hard as it can for a fixed interval of time. To send a 0 bit, it goes to sleep for the same lengih of time.

Client Server Collaborator Encapsulated server □ Figure □ Kernel (a) 9-33. M) 1 □ hL-Uieni □OP The collaborator can fry ti> detecl the bit stream by carefully monitoring its response time, in general, it will get better response when (he server is sending ;, 0 than when the server is sending a I. This communication channel is known as :i covert channel, and is illustrated in Fig 9-3?(b>. Ot course, the covert channel is a noisy channel, containing a lot Ot extrane- extraneous information, bul information can be reliably sent over a noisy channel by using an error-correcting code (e.g., a Hamming code, or even something more sophisticated). The use of an error-correctmg code reduces the already low bandwidth ot the coverl channel even more, but it sliH may be enough to leak sub- substantial information. It is fairly obvious that no protection model based on n matrix of objects and domains is going to prevent this kind of leakage. Modulating the CPU usage is not the only covert channel The paging rate can also be modulated (many page faults for a 1, no page faults for a 0) In fact, almost any way of degrading system performance in a clocked way is a candidate If the system provides a way of locking files, then the server c.m lock some file Ki indicate a I. and unlock ii to indicate a 0. On some systems, it may be possible for a process to detect the status ot a lock even on a tile that it cannot access This covert channel is illustrated in Fig 9-34. with ihe file locked or unlocked for some fixed time interval known lo both the server and collaboiator In this exam pie. the secret bit stream I lOlOlOO is being transmitted Locking and unlocking a prearranged f]]e. S is not an especially noisy chan- channel, but it does require fairly accurate liming unless (he bit rate is very low. The reliability and performance can be increased even more using an acknowledge- ment protocol, This protocol uses two more files, Fl and F2. locked by the server and collaborator, respectively to keep the (wo processes synchronized. After ihe server locks or unlocks S. it flips the lock siatus ot Fl to indicate thai a bil has been sent. As soon as the collaborator has re.td out the bit. it flips F2"s lock status to tetl the server il is ready for another bit and waits until Fl is flipped again tr. indicate that another bit is present in 5. Since timing is no longer involved, (his protocol is fully reliable, even in a busy system and can proceed as fast as the two

fKUSIhD SYS11MS o oo oo oo o Figure 9-34. A rovtrt channel using file locking processes can gel scheduled. To get higher bandwidth, why not use two Tiles per bit time, or make if a byte-wide channel with eight signaling files, SO through 57. Acquiring and releasing dedicated resources (tape drives, plotters,

eic.) can also be used for signaling. The server acquires the resource to send a 1 and releases it to send a 0. In UNIX, the server could create a file to indicate a 1 and remove it to indicate a 0; the collaborator could use the access system call to see if the file exists. This call works even though the collaborator has no permission to use the file, Unfortunately, many other covert channels exist. Lampson also mentioned a way of leaking information lo the (human! owner of the server process. Presumably the server process will be entitled to tell its owner how much work it did on behalf ot the client, so the client can be billed. If the actual computing bill is. say, $100 and the client's income is $53,000 dollars, the server could report the bill as $100.53 to its owner Just finding all the covert channels, let alone blocking them, ,s extremely dif- difficult. In practice, there is little that can be done. Introducing a process that causes page faults at random, or otherwise spends its time degrading system per- performance in order to reduce the bandwidth of (he covert channels is no! an attrac- attractive proposition. So far we have assumed that the client and server are .separate processes. Another case is where there is only one process, the client, which is running a pro- program containing a Trojan horse. The Trojan horse might riave been written by (he collaborator with the purpose of getting the user lo run it in order to leak out data that the protection system prevents the collaborator from getting at directly. A slightly different kind of covert channel can be used to pass secret informa- information between processes, even though a human or automated censor gets to inspect all messages between the processes and veto the suspicious ones. For example, consider a company that manually checks all outgoing email sent by company em- employees to make sure they are not leaking secrets to accomplices or competitors

664 SECURITY CHAP. « outside the company. Is there a way for an employee to smuggle substantial volumes of confidential information right out under the censor's nose? It turns out there is. As a case in point, consider Fig. 9-35<a). This photograph, taken by the author in Kenya, contains three zebras contemplating an acacia tree. Fig. 9-35(b) appears lo be the same three zebras and acacia tree, but it has an extra added attraction. It contains the complete, unabridged texl of five of Shakespeare's plays embedded in it: Hamlei. King Lear. Macbeth. The Merchant of Venice, and Julius Caesar. Together, these plays total over 700 KB of text. (a) (b) Figure 9-35. la) Three zebra* and a irec. (b) Three zebras a iree. and the com- complete text of five ploys hy

William Shakespeare. How does this coven channel work? The original color image is 1024 x 768 pixels. Each pixel consists of three 8-bit numbers, one each for the red. green, and blue intensity of thai pixel. The pixel's color rs formed by the linear superpo- superposition of the three colors. The encoding method uses the low-order bit of each RGB color value as a covert channel. Thus each pixel has room for 3 bits of secret information, one in the red value, one in the green value, and one in the blue value. With an image of this size, up to 1024 x 768 x 3 bits or 294.912 bytes of secret information can be stored in it. The full text of the five plays and a short notice adds up to 734,891 bytes. This was first compressed to about 274 KB using a standard compression algo- algorithm. The compressed output was then encrypted and inserted into the low-order bits of each color value. As can be seen (or actually, cannot be seen), the existence of the information is completely invisible. It is equally invisible in the large, full-color version of the photo. The eye cannot easily distinguish 7-bit color from 8-bit color. Once the image file has gotten past the censor, the receiver just strips off all ihe low-order bits, applies the decryption and decompression algorithms, and recovers the original 734,891 bytes. Hiding the existence of information like this is called steganography (from the Greek words

SFX 9 7 TRUSTED SYSTEMS 665 fur "covered writing"). Steganography is not popular with governments [hat try to restrict communication among their Citizens but it is popular with people who believe slronaly in free speech Viewing the two images in black and while w,th low resolution does not do justice to how powerful the technique is. To get a better feel for how steganogra- ph> works, the author has prepared a demonstration, including the full-color image of Fig. Q-35(b) with the five plays embedded in it. The demonstration can be found at »»»»'.cf.vw nl/~aW■ Click on the covered writing link under the head- heading STEGANOGRAPHY DEMO Then follow the instructions on that page to download the image and the steganography tools needed 10 extract the plays. Another use of steganography is to insert hidden watermarks into images used on Web pages to deteU their theft and reuse on other Web pages. It your Web page contains an image with the secret message. Copyright 2000, General Images Corporation, you might have a tough time convincing a judge that you produced the image yourself. Music, movies, and other kinds of material can also be water- watermarked in this way. Of course, the fact that watermarks are used like this encourages some people to look for

ways to remove them. A scheme that stores information in the low- order bits of each pixel can be defeated by rotating the image I degree clockwise then converting it to a lossy system such as JPEG, then rotating it back by t degree. Finally, the image can be reconverted to the original encoding system (e.g., gif, bmp, tif). The lossy JPEG conversion will me*s up the low-order bits and the rotations involve massive floating-point calculations, which introduce roundotf errrors, also adding noise to the low-order bits. The people putting in the watermarks know this (or should know this), so they put in their copyright infor- information redundantly and use schemes besides just using the low-order bits of the pixels. In tum, this stimulates the attackers to look for better removal techniques, and so it goes. 9.8 RESEARCH ON SECURITY Computer security is a very hot topic, with a great deal of research taking place, bui most of it is not directly operating system related. Instead, it deals with network security (e.g., email, Web, and e-commerce security), cryptography, Java, or just managing a computer installation securely. However, there is also some research closer to our subject For example, user authentication is still important. Monrose and Rubin A997) have studied it using keystroke dynamics, Pentland and Choudhury B000) argue for face recognition, and Mark B000) developed a way to model it, among others. Some other operating systems related security work is as follows. Bershad et a]. A995a) have argued that proteciion is a software issue, not a hardware (i.e., MMU) issue. Mazieres et al. ([999) have looked ai secure distributed file sys-

 terns. Myeis and Uskov A997) studied secure information How models. Chase et al A994) examined security in systems with a large address space occupied by multiple processes Smart card security has been investigated by Clark and Hoff- Hoffman A994) Goldberg et a] A998) have constructed virus phylogemes. 9.9 SUMMARY Operating systems can be threatened in many ways, ranging from Haider attacks to viruses coming in from the outside Many attacks begin with a cracker trying to break into a specific system, often by just guessing passwords. These attacks often use dictionaries of common passwords and they are surprisingly suc- successful. Password security can be made stronger using ^alt. one-time passwords, and challenge-response schemes. Smart cards and biometric indicators can also be used. Retina] scans are already practical. Many different attacks on operating systems are known, including Trojan horse, login spoofing, logic bomb, irap

door, and buffer overflow attacks. Generic attacks include asking for memory and snooping on it, making illegal system calls to see what happens, and even trying to trick insiders into revealing information they should not reveal. Viruses are an increasingly serious problem for many users. Viruses come in many forms, including memory resident viruses, boot sector infectors, and macro viruses. Using a virus scanner to look for virus signatures is useful, but really good viruses can encrypi most of their code and modity the rest with each copy made, making detection very difficult. Some antivirus software works not by looking for specific virus signatures, but by looking for certain suspicious behavior. Avoiding viruses in the first place by safe computing practices is betler than trying to deal with the aftermath of an attack, [n short, do not load and exe- execute programs whose origin is unknown and whose truyworthyness k question- questionable. Mobile code is another issue that has to be dealt with these days. Putting it in a sandbox, interpreting it, and only running code .signed by trusted vendors are possible approaches. Systems can be protected using a matrix of protection domains (e.g.. users) vertically and objects horizontally. The matrix can be sliced into rows, leading lo capability-based systems or into columns, leading to ACL-based systems. Secure systems can be designed, but that has to be a goal from the beginning. Probably she most important design rule is to have a minimal trusted computing base that canno! be bypassed whenever any resource is accessed. Multilevel secu-security can be based on the Bell-La Padula model, designed for keeping secrets, or the Biba model, designed to maintaining system integrity. The Orange Book describes requirements trusted systems must meet. Finally, even if a system is provably secure, attention must be paid to covert channels, which can easily sub- subvert the system by creating communication channels not included in the model.

SHC 9.0 SUMMARY PROBLEMS 667 1. Consider a secret-key cipher thai has a 26 x 26 matrix with ihe columns headed by ' ABC Z and the rows arc also ABC 7 Plaintext is encrypted two characters at a time The first character is the column, the second is the row The cell formed by the intersection of the row and column contains two eiphertext characters- What con- 2. Break the following monoalphabetic cipher The plaintext, consisting of letters only, is .i well-known cxceipt from a poem by Lewis Carroll. ktd kind fzm eubd kid p/yiom m/tx ku kzyg ur b/hj kfthem ur iiifudm zhx m|inm ih\ mdzythc p/q ur c/ss7cdm ?hx gtheni /hx pfa kfd mdz tm sutyihc fuk ?hx pfdkfdi ntcm tzld pthem

sok pztk 7 stk kfd uamkdim eitdx sdruid pd t?ld uoi efzk rui inubd ur om zid uok ur sidzkf zhx ;yy ur ntn zid rvk hu foiia mzix kfd e/.indhkdi kfda kf/.hgdx fib boet rui kfzk 3. Consider the following way to encrypt a file. The encryption algorithm uses two n- byte arrays. A and B The first n bytes are lead from the file into A Then A |0| is copied to B\i\, A[\] is copied io B\j], A[2\ is copied to B\k], etc. After all n bytes are copied [o the B urray. that array is written to the outpui file and n more bvtes arc read into A, This procedure continues until the entire file has been encrypted Note thai here encryption is not being done by replacing characters with other ones, but by changing their order. How many keys have to be tried to exhaustively search the key Space11 Give an advantage nf this scheme over a nionoalphabetic subsiitution cipher' 4. Secret-key cryptography ,s more efficient than public-key ciyptography, but requires the sender and receiver to agree on a key in advance Suppose that the sender and receiver have never mei, but there exists a. trusted third party ihat shares a secret kev with the sender and also shares a (different! secrei key wilht the receiver Hov. can thu sender and receiver establish a new shared secrei key under these cncunistanccs'* 5. Give a simple example of a mathematical function that to a first approximation will do as a one-way function, 6. Not having the computer echo the password is safer than having it echo an asterisk lor each character typed since the latter discloses the password length to anyone nearby who can see the screen Assuming that passwords consist of upper and lower tasfc letters and digits only, and thai passwords must he a minimum ot live characters and a maximum ol eight characters, how much safer is not displaying anvthing'1 puter center that has just put its ancient mainframe sysiem out to pasture and switched over to a large LAN server running UNIX You get the JAb. Fifteen minutes a tier discovered the algorithm we use liir encrypting passwords and posted it on the Inter- Internet " Wruii should you do'1 8. The Morris-Thompson protection scheme with the rt-bit random numheis (salt) was designed to make il difficult for an iniruder to discover a large number of passwords

by encrypiing common slrings in advance. Does ihe scheme also o against a student user who is trying to guess the superuser password o Assume ihe password file is available for reading. 9. II. 1. Name fhree ITS Is A ne characte a login authentic^ there any ack show computer twork U mactiin :ristics that a itor. feasible way (o use n in Fig science 9-lP Expl : department sers on any machine e4 who good biometric ii iidicator mu* ,t have for it the

MMU hardware to prevent the kind am why or why n has a large collection of UNI can issue a comrr land of the fi IX machines arm to be useful of overflow . on its local and have the command executed on machine4, without having the user log in on (he remote machine. This feature is implemented by having the user's kernel send the command and his UlD io the remote machine. Is this scheme secure if the kernels are all trustworthy? What if some of the machines are students' personal computers, with no protection" 12. What property do the implement a! ion of passwords m UNIX have in common with Lamport's scheme for logging in over an insecure network? 13. Lamport's one-time password scheme uses the passwords in reverse order. Would it not be simpler io use/f j) Ihe first lime, fifls)) the second time, and so on0 14. As, Internet cafes become more widespread, people are going to want ways of going to one anywhere in (he world and conducting business from them. Describe a way to produce signed documents from one using a smart card (assume that all the computers are equipped with smart card readers). Is your scheme secure.' 15. Can the Trojan horse attack work in a system pro(ec(ed by capabilities'1 16. Name a C compiler feature that could eliminate a large number of security holes Why is i(not more widely implemented" 17. When a file is removed, its blocks are generally put back oti the free list, but they are not erased. Do you think it would be a good idea to have ihe operating system erase each block before releasing it? Consider both security and performance factors in your answer, and explain the effeci of each. 18. How could TENEX be modified to avoid the password problem described tn the text? 19. How can a parasitic virus (a) ensure that it will be executed before its host program, and (b) pass control back to its host after doing whatever it does9 20. Some operating systems require that disk partitions must start at the r>eginmng of a lrack. How does this make life easier for a boot sector virus? 21. Change the program of Fig. 9-13 so it finds all the C programs instead of all the exe- executable files. 22. The virus in Fig.9-|6(d) is encrypted. How can the dedicated scientists, at the antivirus lab tell wnich pan of the file is the key so they can decrypt the virus and reverse engineer it? What can Virgil do to make their job a lot harder"

23. The virus of Fig 9-16<c) has both a compressor and a decompressor. The decompres- decompressor is needed 10 expand and run the compressed executable program What is the compressor for9 24. Name one disadvantage of a polymorphic encrypting virus (mm the point of v,ew of 25. O|tcn one sees

the following instructions for recovering from a virus, attack 1 Boot the iniected system. 2 Back up all files to an external medium 1 Kurt fiiisk to format the disk. 4 Reinstall the operating system from the original CD-ROM 5 Reload the tiles Irorp the external medium. 26. Arc companion viruses (viruses that do not modify any existing tiles) possible in UNIX'1 If m>, how' If not, why not? 27. What is the difference between a virus and a worm? How do they each reproduce ' 28. Self-extracting archives, which Contain one or more compressed files packaged with an extraction program, are frequently used 10 deliver programs or program updates Discus.s the security implications of this technique 29. On some machines, the SHR instruction used in Fig 9-18(b) fills the unused bits with zeros: on others the sign bit is extended to the right For the correctness of Fig. 9- 18(b), does it matter which kind or shift instruction is used1' If so, which is better1* 30. Represent the ownerships and permissions shown in this UNIX directory listing ,is j protection matrix. Note- asw is a member of two groups: tt<>er\ and tlevel, gum is <t member of only user\ Treat each of the two users and two groups as a domain so the? -rw-r—r— 2 gmw users 908 May 2616:45 PPP-Notes -rwx r-x r-x 1 asw devel 432 May 1312:35 progl -rw-rw 1 asw users 50094 May 3017:51 project.t -rw-r 1 asw devel 13124 May 31 14:30 splash.gif 31. Express the permissions shown in the directory listing of the presious problem us access control lists 32. Modify the ACL for one file to grant or deny an access that cannoi be expressed using (he UNIX nw system Explain this modification. 33. To verify that an applet has been signed by a trusted vendoi, the applet vendor may include a certificate signed by trusted third party that contains its public key How- However, to read the certificate, the user needs the trusted third party's public key This could be provided by a trusieit fourth party, but then the user needs that public ke>. H appears that there is no way to bootstrap the verification sysiem yet existing browsers use it How could ,t work? 34. In a lull access conirol matrix, the rows are for domain objects. What happens if some object is needed in two don

67O SECURITY CHAP V 3* Two different proteciion mechanisms that we have discussed are capabilities and access control lisis. For each of (he following protection problems, tell wtveh of these mechanisms can be used- fa) Ken warns his files readable by everyone except his office mate (b) Mitch and Sieve want to share some secret files (c) i.mda wantf slime of her files to be public. 36. In the Amoeba scheme for protecting capabilities, a user can

asfc iht server to produce a new capability with lewer rights, which can then be given to a friend What happens if the friend asks the server to remove even more rights s° the friend can give it to yet 37. In Fig 9-3i. there is no arrow from process B io «b|ect /. Would such an arrow be allowed" [f noi. what rule would it violate'1 38. In Fig 9-31, there is no arrow from object 2 io process A Would such an arrow he allowed? If not. what rule would it violate'1 39. It process to process messages were allowed m Fig 9-31. what rules would apply tu them1 For process B in particular, to which processes could it send messages and which not" 40. Consider the steganographic system of Fig 9-35. Each pixel can be represented in a color space by a pomi in the 3-dimen.sional system with axes for the R. G. and B values Using this spaLe, explain what happens to the color resolution when *,teganog- raphy is employed as it is in this figure. 41» ^aiura'-language text in ASCI| can oe compressed fay at least ^0% using various compression algorithms. Using this knowledge, what is the steganographic currying capacity for ASCII texi (in bytcs| of a 1600 x 1200 image stored using the low-order bits of each pixel? How much is the image size increased by ihe use oi' this technique (assuming no encryption or no expansion due to encrypiioti)' What is the efficient y ot the scheme, that is pay load/(bytes transmitted) > 43. Suppose that a tightly-^nil group oi' political dissidents living in ,i repressive country are using steganography to send out messages io the woild about conditions ni their country. The government is aware of this and i, fighting them by sending out bogus images containing false steganographic messages How can the dissidents try to help people tell ihe real messages from the fal.se ones' 43, Write a pair of shell senpts to send and receive a text message by a covnn channel on a UNIX system. (H/iir. use the execution time of processes as your covert Mgn.il The sleep command is guaranteed io run for a minimum time, set b\ its argument. <ind the ps command can be used to see jll running processes) 44. Write a pair of programs, in C or as shell scripts, to send and receive a mcssjge by a covert channel On a UNIX system. Hint: A permission bit can be seen even when a file is otherwise inaccessible, and ihe sleep command or system call can is guaranteed io delay for a fixed time, set by its argument Measure the data rJte on an idle system. Then create an artificTally heavy load by starting up numerous different background processes and measure the data rate again.

10 CASE STUDY 1: UNIX AND LINUX In the previous chapters. tractions, algorithms, and ic concrete systems to see how thi will begin wilh UNIX

becau: any oiher operating system, workstations and servers, but computers to supercomputers, and despite its age, is still modi are illustrated by UN|X. Quite Our discussion of UNIX wii Then we will provide examined many operating system principles, :hniques in general. Now it is time to look at iples are applied in the real world, ider variety of compute is ihe dominant operating system on high-end is also used on systems ranging from notebook was carefully designed with a clear goal in mind, -n and elegant. Many important design principles few of these have been copied by other systems, sidrt wiih its history and evolution of the system, w of the system, to give an idea of how it is used. This overview will be ot special value to readers familiar only with Windows, since the latter hides virtually all the details of the system from its users. Although graphical interfaces may be easy for beginners, they provide little flexi- flexibility and no insight into how the system works. Next we come to the heart of this chapter, an examination of processes, memory management, I/O, the file system, and security in UNIX. For each topic we will first discuss the fundamental concepts, then the system calls, and finally the implements!ion. One problem that we will encounter is thai there are many versions and clones of UNIX, including A|X. BSD, IBSD, HP-UX, Linux, MlNIX, OSF/1, SCO UNIX, System V, Solaris. XENIX, and various oihers, and each of these has gone through

672 CASE STUDY I UNIX AND LINUX CHAP. 10 many versions. Fortunately, the fundamental principles and system calls are pretty much the same for all of them (by design). Furthermore, (he general imple- implementation strategies, algorithms, and data structures are similar, but there are some differences. In this chapter we will draw upon several examples when dis- cussing implementation, primarily 4.4BSD (which forms the basis for FreeBSD), System V Release 4, and Linux. Additional information about various implemen- implementations can be found in (Beck el al.,]998; Goodheart and Cox. 1994; Maxwell, l999;McKusicket a!.. 1996; Pate, 1996; and Vahalia, 1996). 10.1 HISTORY OF UNIX UNIX has a long and interesting history, so we will begin our study there. What started out as the pet project of one young researcher has become a mu|- timillion dollar industry involving universities, multinational corporations, governments, and international standardization hodies In the following pages we will iel! how this story has unfolded. 10.1.1 UNICS Back in the 1940s and 1950s, all computers were personal computers, at least in the sense that the ihen-normal way to use a

computer was to sign up tor an hour of time and take over the entire machine for that period. Of course, these machines were physically immense, but only one person (the programmer) could use them at any given time. When baich systems took over, in the 1960s, the pro- programmer submitted a job on punched cards by bringing it to the machine room, When enough jobs had been assembled, the operator read them all in a.s a single batch. It usually took an hour or more after submitting a job until the output was returned. Under these circumstances, debugging was a time-consuming process, hecause a single misplaced comma might result in wasting several hours of the programmer's time, To get around what almost everyone viewed as an unsatisfactory and unpro- unproductive arrangement, timesharing was invented at Dartmouth College and M.I.T. The Dartmouth system ran only BASIC and enjoyed a short-term commercial suc- success before vanishing. The M.I.T. system, CTSS, was general purpose and was an enormous success among the scientific community. Within a short time, research- researchers at M.I.T. joined forces wTth Bell Labs and General Electric (then a computer vendor) and began designing a second generation system, MULTICS (MULTi- plexed Information and Computing Service), as we discussed in chap. 1 Although Bell Labs was one of the founding partners in the MULTICS project, it later pulled out, which left one of the Bell Labs researchers. Ken Thompson, looking around for something interesting to do. He eventually decided to wnte a stripped down MULTICS by himself (in assembler this time) on a discarded PDP-7

SEC 10.1 HISTORY OF UNIX 673 minicomputer. Despite the tiny size of the PDP-7, Thompson's system actually worked and could support Thompson's development effort. Consequently, one of the other researchers at Bell Labs, Brian Kernighan. somewhat jokingly called it UNICS (UNIplexed Information and Computing Service). Despite puns about •"EUNUCHS" being a castrated MULTICS, the name stuck, although the spelling was later changed to UNIX.

10.1.2 PDP-11 UNIX Thompson's work so impressed his colleagues at Bell Labs, that he was soon joined by Dennis Ritchie, and later by his entire department. Two major develop- developments occurred around this time. First, UNIX was moved from the obsolete PDP- 7 to the much more modern PDP-11/20 and then later to the PDP-11/45 and PDP-ll/70. The latter two machines dominated the minicomputer world for muchofthe 1970s. The PDP-] 1/45 and PDP-11/70 were powerful machines with large physical memories for their era B56 KB and 2 MB, respectively). Also, they had memory

.prelection hardware, making jl possible to support multiple users at the same time. However, ihey were both 16-bit machines that limited individual processes to 64 KB of instruction space and 64 KB of data space, even though the machine may have had far more physical memory. The second development concerned the language in which UNIX was written. By now it was becoming painfully obvious that having to rewrite the entire system for each new machine was no fun at ail, so Thompson decided to rewrite UNIX in a high-level language of his own design, called B. B was a simplified form of BCPL (which itself was a simplified form of CPL, which, like PL/I, never worked). Due to weaknesses in B, primarily lack of structures, this attempt was not successful, Ritchie then designed a successor to B, (naturally) called C, and wrote an excellent compiler for it- Together. Thompson and Ritchie rewrote UNIX in C. C was the right language at the right time, and has dominated system programming ever since. In 1974, Ritchie and Thompson published a landmark paper about UNIX (Ritchie and Thompson, 1974). For the work described in this paper they were later given the prestigious ACM Turing Award (Ritchie. 1984; Thompson, 1984). The publication of this paper stimulated many universities lr> ask Bell Labs' for a copy of UNIX. Since Bell Labs' parent company, AT&T, was a regulated mono- monopoly at the time and was not permitted to be in the computer business, it had no objection to licensing UNIX to universities for a modest fee. In one of those coincidences that often shape history, the PDP-11 was the computer of choice at nearly all university computer science departments, and the operating systems that came wich the PDP-11 were widely regarded as being dreadful by professors and students alike. UNIX quickly filled the void, not m the leaa because it was supplied with the complete source code, so people could, and did, tinker with it endlessly. Numerous scientific meetings were organized around

 UNIX with distinguished speakers getting up in front ot the room to tell about some obscure kerne] bug they had found and fixed. An Australian professor, John Lions, wrote a commentary on the UNIX source code of the type normally re- reserved for the works of Chaucer or Shakespeare (reprinted as Lions, 1996), The book described Version 6, so named because it was described in the sixth edition of the UNIX Programmer's Manual. The source code was 8200 lines of C and 900 lines of assembly code. As a result of all this activity, new ideas and improve- improvements to [he system spread rapidly- Within a tew yean,.

Version 6 was replaced by Version 7, the tirst portable version of UN|X (it ran on the PDP-11 and the Interdaia 8/32), by now 18,800 lines of C and 2100 hues of assembler. A whole generation of students was brought up on Version 7. which contributed to its spread atter they graduated and went to work m industry By the mid 1980s. UN|X was in widespread use on min-minicomputers and engineering workslations from a variety ot vendors, A number of companies even licensed the source code to make their own version of UNIX. One of these was a small startup called Microsoft, which sold Version 7 under (he name XENIX for a number of years until its interest turned ckewherc. 10.1.3 Portable UNIX Now that UNIX was written in C, moving it to a new machine, known as port- porting it, was much easier than in the early days. A port requires firsi writing a C compiler for the new machine. Then it requires writing device dnvers for the new machines I/O devices, such as terminals, printers, and disks. Although the driver code is in C, u cannot be moved to another machine, compiled, and run there because no two disks work the same w.iy. Finally, a small amount of machine- dependent code, such as-the interrupt handlers and memory management routine, must be rewritten, usually in assembly language. The first port beyond the PDP-11 wjs to the Interdata 8/12 mimcompuier This exercise revealed ,i large number of assumptions ihat UNIX implicitly made about the machine it was running on, such as the unspoken supposition thai integers heid [6 bits, pointer-, also held 16 bits (implying a maximum program size of 64 KB), and ihat the machine had exactly three registers a\ailable for holding important variables. None of (htsc wore true on the Interdata, so consid- considerable work was needed to clean UNIX up , Another problem was that although Ritchie's compiler was fast and produced good object code, it produced only PDP-1 I object code Rather than write d new compiler specifically for the Interdata, Sieve Johnson of Bell Labs designed and implemented the portable C compiler, which could be retargeted to produce code for any reasonable machine wjth a only a moderate amount of effort For years, nearly all C compilers tor machines other than the PDP-] 1 were based on Johnson's compiler, which greatly aided the spread of UNIX to new computers

 The port to the Interdata initially weni slowly because all the development work had 10 be done on the only working UNIX machine, a PDP-I I, which hap- happened to be on the fifth floor ai Bell Labs. The Interdata was on the first floor. Generating a new version meant compiling

ii on the fifth floor and then physically carrying a magnetic tape down to the first floor to see if it worked. After several months, a greai deal of interest arose m (he possibility of connecting these two machines togeiher electronically- UNIX networking traces its roots to this link. After the Interdata port. UNIX wa.s ported to the VaX and other computers. Aher AT&T was broken up m 1984 by the U.S. government, the company was legally free to set up a computer subsidiary, and did. Shortly thereafter. AT&T released its' first commercial UNIX product. System III. It was not well received- so it was replaced by an improved version. System V, a year later. Whatever happened to System IV is one of the great unsolved mysteries of com- computer science. The original System V has since been replaced by System V. releases 2, 3. and 4, each one bigger and more complicated than its predecessor. In the procew,, the original idea behind UN|X, of having a simple, elegant system ha.s gradually diminished. Although Ritchie and Thompson's group later pro- produced an 8th, 9th, and]Oth edition of UNIX, these were never widely circulated, as AT&T put all its marketing muscle behind System V. However, some of the ideas from the 8th. 9th, and]Oth editions were eventually incorporaied into Sys- System V. AT&T eventually decided that it wanted to be a telephone company, not a computer company, after all, and sold)ts UNIX business to Novell in 1993. Novell then sold it to (he Santa Cruz Operation in 1995 By then it was almost irrelevant who owned it since all the major computer companies already had 10.1.4 Berkeley UNIX One of the many universities that acquired UN|X Version 6 early on was the University of California at Berkeley. Because the complele source code was available, Berkeley was able to modify the system substantially Ajded by grant*. from ARPA, the U.S. Dept of Defense's Advanced Research Project* Agency. Berkeley produced and released an improved version tor the PDP-II called 1BSD (First Berkeley Software Distribution) This tape was followed quickly by 2BSD also for the PDP-11. More important were 3BSD and especially its suceessoi. 4BSD. for the VAX. Although AT&T had a VAX version of UNIX, called 32V. it w;,s essentially Ver- Version 7. [n contrast, 4BSD (including 4.|BSD. 42BSD. 4.1BSD, and 44BSD) con- contained a large number of improvements. Foremost among these was the use of virtual memory and paging, allowing programs to be larger than physical memory by paging parts of them in and out as needed Another change allowed h\l- names to oe longer than 14 characters- The implementation of the file system was also changed, making it considerably faster. Signal handling was made more reliable.

 Networking was introduced, causing the network protocol that was used, TCP/IP, to become a de facto standard in the UN[X world, and later in the Internet, which is dominated by UNIX-based servers. Berkeley also added a substantial number of utility programs to UNIX, includ- including ., new editor (yi), a new shell (ah), Pascal and Lisp compilers, and many more. All these improvements caused Sun Microsystems, DEC, and other com- computer vendors io base their versions of UNIX on Berkeley UNIX, rather than on AT&T's "official" version, System V. As a consequence, Berketey UNIX became well established in the academic, research, and defense worlds. For more iniormaiion about Berkeley UNIX, see (McKusick ei al., 1996). 10.1.5 Standard UNIX By ihe laie 1980s, two different, and somewhat incompatible, versions of UNIX were in widespread use: 4.3BSD and System V Release 3. In addition, vir- virtually every vendor added its own nonstandard enhancements. This split in the L'N]X world, together with ihe fact ihat there were no standards for binary pro- program formats, greatly inhibited the commercial success of UNIX because it was impossible for software vendors to write and package UNIX programs with the expectation that they would run on any UNIX system (as was routinely done with MS-DOS). Various attempts at standardizing UNIX initially failed. AT&T, for example, issued the SVID (System V Interface Definition), which defined all the system calls, file formats, and so on This document was an attempt to keep al! the System V vendors in line, but it had no effect on ihe enemy (BSD) camp, which just ignored it. The first senous attempt to reconcile the two flavors of UN]X was initiated under the auspices of the IEEE Standards Board, a highly respected and, most important, neutral body Hundreds of people from industry, academia, and government took part in this work. The collective name for this project was POSIX. The first three letters refer to Portable Operating Sysiem. The IX was added to make the name UNIXish. After a great deal ot argument and counterargument, rebuttal and counterre- buttal, the POSIX committee produced a standard known as 1003.1. It defines a set of library procedures that every conformant UNIX system must supply Most of these procedures invoke a system call, but a few can be implemented' outside ihe kernel Typical procedures are open, read, and fork. The idea of POSIX k that a software vendor who writes a program that uses only the procedures defined by 1003.1 knows that this program will run on every conformant UNIX system. While n is trne that most standards bodies tend to produce a horrible compromise with a few of everyone's pet features in it,

1003.1 is remarkably good considering the large number of parties involved and their respective vested interests. Rather than take the union of all features in System V and BSD as the starting point (the norm for most standards bodies), the IEEE committee took the

sec 101 history of Unix 677 intersection Very roughly, if a feature was present m both System V anil BSD. il was included in the standard, otherwise it was not. As a consequence of this algo- algorithm, 1003.1 bears a strong resemblance to the direct ancestor of both System V and BSD, namely Version 7 The two areas in which it most strongly deviates from Version 7 are signals (which is largely taken from BSD) and terminal han- handling, which is new The 10O1* 1 document is written m such a way that both operating system implemented and software wrnen can understand it. another no\elty in the standards world, although work is already underway to remedy this Although the 1003 1 standard addresses only the system calls, related docu- documents standardize threads, the utility programs, networking, and many other features of UNIX, In addition, the C language has also teen standardized by ANSI and ISO. Unfortunately, a funny thing happened on the way back from the standards meeting. Now that the System V versus BSD split had been dealt with, another one appeared. A group of vendors led by IBM. DEC. Hewlett-Packard, and many others did not like the idea that AT&T had control of the rest of UNIX, so ihey set up a consortium known as OSF (Open Software Foundation) to produce a sys- system lhai met all the IEEE and other standards, but also contained a large number of additional features, such as a windowing system (Xl |), a graphical user inter- interface (MOTIF), distributed computing (DcE). distributed management (DME). and much more. AT&T's reaction was to set up its own consortium. Ul (UNIX International) to do precisely the same thing. Ul's version ot L'NIX was based on System V The net result is that the world then had fwo powerful industry groups each otter- ottering their own version ot UNIX, so the users were no closer to a standard than they were in the beginning. However, the marketplace decided that System V was a better bet than the OSF system, so the latter gradually vanished Some companies have their own variants ot UNIX, such as Sun's Solaris (based on System V) 10.1.6 MINIX One property that all these systems have is that they are large and compli- complicated, in a sense, the antithesis of the original idea behind UNIX Even if the source code were freely available, which it is not in most case.s, it is out of (he question that a single person could understand it all any

more. This situation led to the author of this book writing a new UN|X-hke system that was small enough lo understand, was available with all the source code, and could be used for edu- educational purposes That system consisted of 11,800 lines of C and 800 lines of assembly code. It was released in 1987, and was functionally almost equivalent to Version 7 UNIX, the mainstay ot most computer science departments during the PDP-11 era. MINIX was one of the first LNIX-like systems based on a microkernel design. The idea behind a microkernel is to provide minimal functionality in the kernel to

678 CASE STUDY I UNIX AND LINUX CHA1J 10 make it reliable and efficient Consequently, memory management and the tile system were pushed out into user processes. The kernel handled message passing between the process and l.tllc else. The kernel was 1600 lines of C and 800 lines ot assembler. For technical reasons relating to the 8088 architecture, the I/O device drivers B900 additional lines ot C) were also in the kernel The file sys- system E 100 lines of C) and memory manager B200 lines- of C) ran as two separate Microkernels have ilie advantage over monolithic system1! lhat lhcy are easy io understand and maintain due 10 their highly modular structure. Also, moving code from the kernel to user mode makes them highly reliable because the crash ot a user-mode process does less damage than the crash ot a kernel-mode com- component. Their inain disadvantage is a slightly lower performance due to the extra switches between user mode and kernel mode. However, performance is noi everything, all modern UNIX systems run X Windows in user mode and simply accept the performance hit to gei the greater modularity (in contrast to Windows, where the entire GUI is, in the kernel). Other well-known microkernel designs of this era were Mach (Accetta et ai., 1986) and Chorus (Rozier et al..]988). A dis- discussion of microkernel performance issues is given in (Brickerct al., 199]). Within a few months ot its appearance, MfNIX became a bit of a cult item, with its own newsgroup. comp.(n,.min"x, and over 40,000 users. Many users con- tubuted commands and other user programs, soMlNtX became a collective under- undertaking done by large numbers of users over the Internet. It was a prototype of other collaborative efforts that came later In !W. Versmn 2.0 of MINiX. was released and ihe base system, now including networking, had grown to 62.200 lines, of code. A book about operating systems principles illustrated using the 500-page MIN|X source code given in an appendix and on an accompanying CD- ROM is (Tanenbaum and Woodhull, 1997) MlNlX is also available for free on the

World Wide Web at URL wwH.cs.vu.nV~asf/mimx.html. 10.1.7 Linux During the early years of M!N[X development and discussion on the Internet. many people requested (or in many cases, demanded) more and better features, to which the author often said "No" (to keep the system small enough for students to understand completely in a one-semester university course). This continuous "No" irked many users. At this time, FreeBSD was not available, so that was not an option. After a number of years went by like this, a Finnish student, Linus Torvalds, decided to write another UNIX clone, named Linux, which would be a full-blown production system with many features MINIX was (intentionally) lack- lacking. The first version of Linux, 0.01, was released in 1991. [t was cross- developed on a M!N!X machine and borrowed some ideas from MtNIX ranging from the structure of the source tree to the layout of the file system. However, it was a monoliihic rather than a microkernel design, with the entire operating sys-

SEC |0 1 HISTORY OF UNIX *»79 tern m the kernel- The code sire totaled 9-300 lines ot C and 950 lures ot assem- assembler, roughly similar io M1N1X version m size and also roughly comparable in functionality. Linux rapidly grew in size and evolved into a full production UNIX clone as virtual memory, a more sophisticated file system, and many other features were added. Although it originally ran only on the 3S6 (and even had embedded 386 assembly code in the middle of C procedures), it was quickly ported to other plat- tbrms and now runs on a wide variety ot machines, just as UNIX does One difference with UNIX docs stand out however Linux makes use ot many special features ot the gic compiler and would need a lot (:f work before n would compile with an ANSI standard C contpilei with no exira features The next major release ol Linux was version 1-0, issued in 1994. 1; was about 165,000 lines of code and included a new file system, memory-mapped files, and BSD-compatible networking with sockets and TCP/IP, it also included many new device drivers. Several minor revisions followed in lhe next two years By this tune, Linux was sufficiently compatible with UNIX that a vasi amount of UNIX software was ported to Linux, making it far more useful than it would have otherwise been In addition, a large number of people were attracted to Linux and began working on the coiic and extending it in nianv ways under Tor- valds' general supervision. The next major release. 2 0. was made in 1996- It consisted oi about 470,000 lines of C and 8000 lines of assembly code It included support tor 64-bit archi- architectures, symmetric

multiprogramming, new networking protocols, and numerous other features. A large iraction of the total code mass was taken up by an exten- extensive collection of device drivers. Additional releases followed frequently A large array of standard UNIX software has been ported to Linux, including over 1000 utility programs, X Windows and a great deal of networking soft wait;. Two different GUIs (GNOME and KDF.) have also been written for Linux. In short, it has grown to a full-blown UNIX clone with all the belk and whistles a UNIX lover might want. One unusual feature <if Linux is its business model- it is tree software It can be downloaded from various sites on the Internet, for example- nwH.kenwI.org Linux comes with a license devised by Richard Stallman, founder of the Free Software Foundation. Despite the fact that Linux is tree, lhis license, the GPL (GNU Public License), is longer than Microsoft's Windows 2000 license and Specifies what you can and cannot do with lhe code. Users may use. copy, modify, and redistribute the source and binary code freely The mam restriction is that all works derived from the Linux kernel may not be sold or redistributed in binary form only; the source code must either be shipped with the product or he made available on request. Although Torvalds still controls the kernel fairly cfoscly a large amount of user-level software has been written by numerous other programmers, many of them originally migrated over from the M1N|X, BSD, and GNU (Free Software

680 CASE STUDY T UMX AND LINUX CHAP 10 l-ouiidalion) online communities However, as Linux evolves, a steadily smaller traction of the Linux community want to hack source code (witness hundreds of books telling how to install and use Linux and <ml> a handful discussing lhe code or how it works) Also, many Linux users now forego the free distribution on the internet to buy one of many CD-ROM distributions available trom numerous cornpctiim commercial companies A Web site listing over SO companies that sell dilteicnt Linux packages is www.Itnux ors> As more and more software com- companies start selling their own versions o| Linux and more and more hardware companies otter to premstall it on the computers they ship, the line beiween com- commercial software and free software ts beginning to blur a little. As a footnote to the Linux story, it is interesting lo note that just a$ the Linux bandwagon was gaining steam, it got a big boost from an unexpected source— AT&T In)992. Berkeley, by now running out ot funding, decided to terminate BSD development with one final release. 4 4BSD, (which later formed the basis of FreeBSD). Since this version contained essentially no

AT&T code, Berkeley issued the software under an open source license (not GPL) that let everybody do whatever they wanted with it except one thing—sue the University of California The AT&T subsidiary controlling UNIX promptly reacted by—you guessed it— suing the University of California It simultaneously sued a company, BSD1, set up by the BSD developers to package the sysicm and sell support, much as Red Hai and other companies now do for Linux Since virtually no AT&T code was involved, the lawsuit was based on copyright and trademark infringement, includ- including items such as BSDIs 1-800-ITS-UN1X telephone number. Although the case wa.s eventually settled out ot court, this legal action kept FreeBSD off the market just long enough for Linux to get well established Had the lawsuit not happened, starting around]993 thtro would have been a serious competition between two free, open source UNIX systems: the reigning champion. BSD, a mature and stable system with a large academic following d.itmg back to 1977 versus the vigorous young challenger, Linux.just two years old but With a growing following among individual users Who knows how this battle ot the free UNICES would have turned ouf? Given this history, strict POS|X contormance. and overlap between the user communities, it should not come as a surprise that many of Linux' features, sys- system calls, programs, libraries, algorithms, and internal data structures are very similar lo those of UNIX. For example, over 80L7< of the ca. 150 Linux system calls aie exact copies of the corresponding sysl^m calls in POSIX. BSD. Or System V. Thus to a first approximation, much of the description of UNIX given in thTs chapter also applies to Linux In places where there are substantial algorithmic differences beiween UN|X and Linux (e.g.. the scheduling algorithm), we will point that out and cover both. Where they are essentially file same, for the sake of brevity we willjust refer to UNIX Be warned that Linux is evolving rapidly, and with thousands of people- hacking away on it. some of this matenal (based on ver-version 2.2) is sure to become obsolete before long.

SEC 10 2 OVERVIEW OF UNIX "al 10.2 OVERVIEW OF UNIX In this section we will provide a general introduction to UNIX and how it is used, for the benefit ot readers not already familiar with it Although different version of UN|X differ m subtle ways, the material presented here applies !o all ot them. The focus here is how UN|X appears at the terminal. Subsequent sec- sections Will focus on system calls and how it woiks inside. 10.2.1 UNIX Goals UNIX is an interactive system designed to handle multiple processes and mul-

multiple users at the same time. It was deigned by programmers. for progtatniners. cated and are engaged m (often quite complex) software development projects. In many cases, a large number of programmers are actively cooperating to produce a single system, so UN]X has extensive facilities to allow people to work togethei and share information in controlled ways The model of a group of experienced programmers working together closely to produce advanced software is obviously very different from the personal computer model ot a single beginner working alone with a word processor, and this difference is reflected throughout UN|X from start to finish. What is it that good programmers want in a system' To start with, most like their systems to be simple, elegant, and consistent. For example, at the lowesi level, a file should just be a collection of byies Having different classes of files frames do)ju«;t gets m the way. Similaily. it the command Is A* means list all the files beginning with "A" then the command whose name consists of an "A" and t,n asterisk This characteristic is sometimes called the principle oj IcaM surprise. Another thing that experienced programmers generally want is power and flexibility. This means thai a system should have a small number ot basic ele- One of the basic guidelines behind UNIX is that every program should do |ust one thing and do it well. Thus compilers do not produce listings, because other pro- programs can do that belter Finally, moM programmers have a strong dislike foi useless redundancy Why type copy when tp is enough" 'Io extract all (he lines containing the string "ard" from the file/, the UNIX programmer types grep ard f

682 CASK STUDY I" UNIX AND LINUX CHAP 10 The opposite approach is to have the programmer first select the gtep program (with no arguments), and then have #rey> announce itself bj saying: "Hi, I'm grep,] look tor patterns in files. Please enter your pattern " After getting the pattern. gre/y prompts for a file name. Then it asks if there are any more file names. Finally it summarizes what it is going to do and ask it that is correct While this kind ot usei interface may or may not be suitable for rank novices, it irritates skilled prog rammers no end. What [hey want is a servant, not a nanny 10.2.2 Interfaces to UNIX A UNIX system cm be regarded as a kind ot pyramid, as illustrated in Fig 10-1 At the bottom is the hardware, consisting of the CPU. memory, disks, terminals, and i,ther devices Running on the bare hardware is the UN|X operating system. Its function is to control the hardware and provide a system call interface to all the programs These system calls allow user programs to create and manage

processes, tiles, and other resources call interlace 1 brary erf ace 1 eltace Standard library (open, close, read, write, fork etc) (pros UNIX operating syslem jss management, memory management Hardware (CPU memory, disks, terminals elc) Figure 10-1. the ta MX ^ Programs make system culls' by putting the arguments in registers (or some- sometimes, on the stack), and issuing tmp instructions to switch from user mode to kei- nel mode to start up UNIX Since there is no way to write a irap instruction in C. a library is provided, with one procedure per system call. These procedures are written in assembly language, but can be called trom C Each one tirst puts its arguments in the proper place, then executes- the trap instruction. Thus to execute the read system call, a C program can call the read library procedure. As an aside, it is the library interface, and not the system call interface, that is specified by POSIX. In oiher words, PQSIX tells which library procedures a contonnam system must supply, what their parameters are, what ihey must do, and whd! results they must return. It does not even mention the actual system calls.

OVERVIEW Of UNIX 683 In addition to the operating system and system call library, ail versions of UNIX supply a large numbiM oT standard programs, some of which are specified by the POSIX 1003 2 standard, and some of which differ between UNIX versions. These include the command proccssot (shell) compilers, editors, text processing programs, and file manipulation utilities It is these programs thai a. user at a ier- Thus we can speal\ ot three differeiit interlaces to {-*y*\\. trie true system can interface, the library interface, and the interface formed by the set of standard util- ily programs. While the latter is what the casual user thinks of as "UNIX." in fact, it has almost nothing to do with the operating system itself and can easily be replaced Some- versions of UNIX, for example, have replaced this keyboard-oriented user interface with a mouse-oriented graphical user interface, without changing the operating system itself at all It is precisely this flexibility that makes UNIX so popular and has allowed it to survive numerous changes in the underlying technol- technology so well. 10.2.3 The UNIX Shell Many UNIX systems have a graphical user interface of the Wind made popular by (he Macintosh and later Windows. However, real programmers still prefer a command line interface, called the shell. It is much faster to use, more powerful, easily extensible, and does not give the user RSI from having to use a mouse all the time- Below we wilt briefly describe the Bourne shell (.5/1). Since then, many new shells have been written (fci/i, bu.^h, etc.). Although UNIX fully supports a graphical

environment (X Windows), even in this world many programmers sim- simply make multiple console windows and act as it they have half a dozen ASCII terminals each running the shell. When the shell starts up. it initializes itself, then types a prompt character, often a percent or dollar sign, on ihe screen and waits, for the user to type a com- command line. When the user types a command line, the shell extracts the first word from it. assumes it is the name of a program to be run, searches for this program, and if it finds it. runs the program. The shell then suspends itself until the program ter- terminates, a' which time it tries to read the next command. What is important here is simply the observation that the shell is an ordinary' user program. All H needs i>> the ability to read from and write to the terminal, and the power to execute other programs. Commands may take arguments, which are passed to the called program a.s character strings, For example, the command line cp sre dest invokes the cp program with two arguments, sre and dest. This program interprets

 the firsi one to be the name of an existing file if makes a copy of this file .Hid calls the copy (kit. Not all arguments are file names In head-20 file the firsi argument. -20, tells head to print thi-' first 20 lines of file, instead of the default number of lines', 10 Arguments that control the operation of a command or spc-c^y an optional value are tailed fiags. and by convention are indicated with d dash The dash is required to avoid ambiguity, because the command head 20 file ,s perfectly legal, and tells head to first print the initial 10 lines of a file called 20, and then print the initial 10 line;, of a second file called file. Most UNIX com- commands accept multiple flags and arguments. To make it easy to specify multiple file names, the shell accepts magic char- characters, sometimes called wild cards An asterisk, tor example, matches all possi- possible strings, so tells Is to list all the files whose name ends in x [t files named c c, \.c, and r.t all exist, the above command is equivalent to typing Is x.c y.c z.c Another wild card is the question mark, which matches any one character. A list of characters inside square brackets selects an^ of them, so Is [ape}* lists all tiles beginning wuh "a", "p". or 'V A program like the shell does not hu\e io open th^ terminal in order to read from it or write to it. Instead, when it (or anv other program) starts up. it automai- ically has access to a file called standard input (lor reading), a file called stan- standard output (for writing normal output), and a file called standard error (for writing error messages)- Normally, all

three default to the tetmmal. so that reads from standard input come from the keyboard and writes to standard output or stan- standard error go to ihe screen. Many UN]X programs read from standard input and write to standard output as the default. For example, invokes the sort program, which reads lines from the terminal (until the usei types a CTRL-D, to indicate end of file), sorts them alphabetically, and writes the result io the screen It is also possible to redirect standard input and standard output, as that is often useful. The syntax for redirecting standard input uses a less than sign (<)

SEC. IU.2 OVERVIEW OF UNIX «<>5 followed by the input file name. Similarly, standard output is redirected using a gicdtcr tiian sign (>) It is permuted lo redirect boili in the same command For example, the command sort <in >out causes sort to take its input frt'm the file m and write its output to the tile out Since standard error hjs not been redirected, any error messages go to the screen. A program thai reads its input from standard input, does some processing on it. and writes us output 10 standard output is called a filter Consider the following command line consisting of three separate commands: sort <in >temp, head -30 <temp, rm temp ll first runs sort, taking the input from in and writing the output to temp When that has been completed, the shell runs head, telling it to prim the first 30 lines of lern/) and print them on standard output, which defaults Co the terminal. Finally, the temporary file is removed. It frequently occurs that the first program in a command line produces output that is used as lhe input on the next program. In the above example, we used the file temp lo hold this output. However, UNIX provides a simpler construction to do the same thing. In sort <tn I head -30 the vertical bai, called the pipe symbol, says to take the output from mrt and u^e it as the input to head, eliminating the need for creating, using, and removing the temporary file A collection of commands connected by pipe symbols, called a pipeline, may contain arbitrarily many commands A four-component pipeline is shown by the following example grep ter *.t I sort I head -20 I tail -5 >foo Here all the lTnes containing the siring "ter" m all ihe files ending in .t are written to standard output, where they are sorted. The first 20 ot these are selected out by head which passes then to tad, which writes the last live (i.e., lines 16 to 2() in the sorted list) vofoo. This is an example of how UNIX provides basic building blocks (numerous filters), each of which does one job, along with a mechanism for them to be put together in almost limitless ways. UNIX is a general-purpose multiprogiammitig system. A single user can run several programs at

once, each as j separate process. The shell syntax for running a process in the background is to tolled its command with an ampersand. Thus we -I <a >b & runs the word count program, m, to couni the number of lines (-/ flagj (tt i(s input, a. writing the result to b, but does it in the background. As soon as the command has been typed, the shell types the prompt and is ready to accept and

686 CASE STUDY [■ UN|X AND LINUX CHAP. 10 handle the next command. Pipelines can also be put iri the background, tor exam- example, by sort <x I head & Multiple pipelines can run in ihc background simultaneously. It is possible to put a list ot shell commands in a file and then start <i shell with this file as standard input. The (second) shell just processes ihem in order, the same as it would with commands typed on the keyboard Files, containing shell commands are called shell scripts. She)) scripts may assign values to shell vari- variables and lhen read them later. They may also have parameters, and Use if, for, while, antl case constructs. Thus a shell script is really a program written in she]! language. The Berkeley C shull is an aliernative .sheil thai has heen designed to make shell scripts (and the command language m general) look like C programs in many respects Since the shell is just another user program, various other people have written and distributed a variety of other shells. 10.2.4 UNIX Utility Programs The user interface to UNIX consists not only of the shell, but also of a large number of standard utility programs- Roughly speaking, ihese programs can be divided into Six categories, as follows: I • File and directory manipulation commands. 2. Filters. 3. Program development took such as editors and compilers. 4. Text processing, 5- System administration. 6, Miscellaneous. The POS1X 1003.2 standard specifies the syntax and semantics of just under 100 of these, primarily in the first ihree categories. The idea of standardizing them is to make n possible for anyone lo write shell .scripts that use these programs and work on all UNIX systems. In addition to these standard utilities, there are many application programs a., well, of course, such as Web browsers, image viewers, etc. Let us consider some examples of these programs, starting with tile and diiec- tory manipulation. cpa b copies file (i [o b, leaving the original file intact. In contrast, mva b

OVBRVlbW Of UNIX 687 really making a copy m the usual sense. Seveial files can be concatenated using cat which reads each of its input files and copies them all to standard output, one after anothei. Files cm be removed by

the rm command The chmod command allows the owner to change the rights bits to modify access permissions. Direc- in a d& rectory I** cdn lie used it has <i viisi number or *i*i^s to control now mucn detail about each file is shown B-g-- size, owner, group, creation date), to deter- determine the- sort order (e g . alphabetical, by time of last modification, reversed), lo specify the layout on ihc screen, and much more. We have already seen several filters: grep extracts hues containing a yiven pattern from standard input or one or more input files: s,orr sorts us input and Writes it on standard output; head extracts the initial lines of its input; toil extracts the final lines of its input Olher filters defined by 1003.2 are cur and paste. which allow columns of text to be cut and pasted into files; vd which converts its (usually binary) input to ASCII text, in octal, decimal, or hexadecimal; tr, which does character translation (e.g., lower case to upper case), and pr which formats output for the printer, including options to include running heads, page numbers, and so on. Compilers and programming tools include <<■. which calls the C compiler and ur, which collects library procedures into archive files. Another important tool is make, which is used to maintain large programs whose source code consists of multiple files. Typically, some of these are header files, which contain type, variable, macro, and other declarations Source files otten include these using a special include directive This way, two or more source files can share the same declarations However, if a header file is modi- Tied, it is necessary to find all the source files that depend on it, and recompile them The function of make is to keep track ot which file depends on which header, and similar things, and arrange for all the necessary compilations to occur automatically. Nearly all UNIX programs, except the smallest ones, arc set up to be compiled with make A selection of the POSlX utility programs is listed m Fig. 10-2, along with a short description of each one. All UN|X systems have these programs, and many 10.2.5 Kernel Structure In Fig. 10-1 we saw the overall structure of a UN|X system Now let us zo<ir in and took more closely at the kernel before examining the various parts. Show ing the kernel .struciure is slightly tricky since there are many different versions c UNIX, but although the diagram of Fig. 10-3 describes 4.4BSD. it also applies ti many other versions with perhaps small changes here and there,

CASIZSlLDYl UNIX ANDLINJiX Concatenate multiple dies to standard output Change tje protection mode Copy one or m Cut c s of text fr Search a file for_som_e pattern Extract the Iirst lines of a hie] List directory ; Compile

files to build a binary Make a directory Octa I paste Paste columns ol text into a f u0r Format a file for printing Remove a directory J Sort a lne ol l.nes alphabetic; Extract the last lines of_a file Translate between character System calls Term.na Raw tty Cha dev han,ln8 Cooked tty Lme disciplines cef Sockets Network protocols Routing Network device drivers sys^rrs Buffer D device Map ping mV in Page [aulls mor' Page drivers errupls and traps Signal handling P,o«.,, OrrnnOion scheduling Process dispatching Figure 10-3. Stnic The bottom layer of the kernel cotisi ine Douom layer 01 trie Kernel consists or trie uevtce onvers plus process dispatching. All UNIX drivers are classified as either character device drivers or block device drivers, with the main difference that seeks are allowed on block devices and not on character devices. Technically, network devices are character

SEC 10 2 OVERVIEW OF UNIX 689 devices, bui they are handled so different!* that it is probably clearer to separate them, as has been done m the figure Process dispatching occurs whet, ,,n inter- interrupt happens. The low-level code here stop*, the running process, saves its state m the kernel process table, and starts the appropriate driver Process dispatching al- also happens when the kernel is finished and u is time to start up a user process again Dispatching code is m assembler and ^ quite distinct from scheduling. Above the bottom level, the code is different in each of the lour "columns" ot Fig. it)-3. At the left, we have lhe chdractei devices. There are two ways they are used Some programs, such as visual editors like \i and emais. want every key stroke as it is hit. Raw terminal (tty) I/O makes tins possible Other software, such as the shell (jM. is line oriented and allows users U> edit the current line before hitting ENTER to send it to the program This software uses cooked m,idc and line disciplines. Networking software ts often modular, with different devices and protocols supported. The layer above the network drivers handles a kind of routing func- function, making sure the right packet goes to the right device or protocol handler Most UNIX systems contain the full functionality of an Internet router within the kernel, although the performance is less than (hat of a hardware loutcr. hut this code predated modern hardware routers Above the router code is the actual pro- protocol stack, always including IP and TCP, but sometimes additional protocols as well. Overlaying all ihe network is the socket interface, which allows programs to create sockets for particular networks and protocols- getting back .1 file descriptor for each socket to use later. On lop of the disk drivers are the

file system's, buffer cache and lhe p.,ge cache. In early UNIX systems, the buffer ^ache was a fixed chunk of memory, with the rest of memory for user pages. In many modern LN|X systems, there is no longer a fixed boundary, and any page of memory can be grabbed for either function, depending on what is needed more On top of the buffer cache come the file systems. Most UNIX systems support multiple file systems, including the Berkeley fast rile system, log-structured file system, and various System V file sysiems All (,f these file systems share [he same buffer cache. On top of the file systems come tile naming, directory management, hard Jink and symbolic link management, and other file system pro- properties that are the same for all file systems. On top of the page cache is the virtual memory system All lhe paging logic is here, such as the page replacement algorithm On top of it is the code for map- mapping files onto virtual memory and the high-level page fault management code This is the code that figures out what to do when a page fault occurs, it first checks if the memory reference is valid, and if so. where ihe needed page k located and how it can be obtained. The last column deals with process management Above the dispatcher is the process scheduler, which chooses which process to run next. If threads are man- managed in the kernel, thread management is also here, although threads are managed

more closely at the basic concepts UNIX supports, namely, processes. File system, and inpul/ouipul These notions are important because alls—the interface to the operating system iiself—manipulate them 690 CASFSTl'DYl UNIX .\NDLl\UX CHAP 10^ in usei space on some UNIX systems. Above the scheduler comes (he tode for processing signals and sending them to the correct destination, as well as ihe proc- process creation and termination code The tt,p layer is the interface into the sysiem On the Iett i-s the system call utertace All svsteni calls coint* here iind are directed to one cit tnc lower modules, depending on the nature of the call On right part ot the top layer is the entrance for traps and interrupts, including signals, page faults, piocessoi excep- exceptions ol all k.nds. and I/O interrupts. 10.3 PROCESSES IN UNIX in the previous sections, we started out by looking at UNIX as viewed fioni the keyboard, that is. what the user sees at the terminal We gave examples of shell commands and utility programs that are frequently used We ended with a brief overview ot the system structure. Now it is time to dig deeply into the kei nel and look more closelv at the basic concents UNIX ■ memory, th the system For example, system calls exist to create piocesscs. allocate memory, open ill and

do I/O. Unfortunately, with so many versions of UNIX in existence, there are .some differences between them In this chapter, we will emphasize the features com- common (o all ot them rather than focus on any one specific version. Thus in certain sections (especially implementation sections), the discussion may not apply equally to every version fO.3.1 Fundamental Concepts The only active entities in a UN|X system are the processes. UNIX processes are very similar (o the classical sequential processes that we studied in Chap 2. Each process runs a single program and initially has a single thread of control. In other words, it has one program counter, which keeps track of the next instruction to be executed. Most versions of UN|X allow a process to create additional threads once it starts executing. UNIX is a multiprogramming sjstem, so multiple, independent processes may be running at (he same time. Each user may have several active processes ai once, so on a large system, there may be hundreds or even thousands of processes running. In fact, on most single-User workstations, even when the user is absent, dozens of background processes, called daemons, are running These are started automatically when the system is booted. ("Daemon" is a variant spelling of ■■demon," which is a self-employed evil spirit.)

SFC KM PROCESSES IN UN|X 691 A typical daemon is the cron daemon It wakes up once a minute To check if there is any work" foi it !o do. 11 so, u doe« the work. Then it goes back lo sleep uniil it is time for the next check This daemon is needed because n is possible m UNIX to schedule activities minutes, hours, days, or even months m the future. For example, suppose a user has a dentist appointment at 3 o'clock next Tuesday. He can make an entry in the cron daemon's database telling the daemon to beep at him at, say, 2:30. When the appointed day and time arrives, the cron daemon sees that it has work to do. and starts up the beeping program as a new process The cron daemon is also used to start up periodic activities, such as miiKin^ daily disk backups at 4 A.M.. or reminding forgetful users every year on October 31 lo stock up on tnck-ot-treal goodies for Halloween. Other daemons handle incoming and outgoing electronic mail, manage the lino printer queue, check it there arc enough free pages in memory, and so forth. Daemons are straightfor- straightforward io implement in UNIX because each one is a separate process, independent ot all other processes. Processes are created in UNIX in an especially simple manner. The fork sys- system call creates an exact copy of the original process. The forking process is called the parent process. The new

process is called the child process. The parent and child each have their own, private memory images,. If the parent sub- subsequently changes any of its variables, the changes are not visible to the child, and vice versa. Open files are shared between parent and child. That is, if a certain file wa.s open in the parent before the fork, it will continue to be open in both the parent and the child afterward Changes made to the file by either one will be visible to the other. This behavior is only reasonable, because these changes are also visible to any unrelated process lhat opens the file as well. The fact that the memory images, variables, registers, and everything else are identical in the parent and child lead.s to a small difficulty: How do the processes know which one should run the parent code and which one should run the child code? The secret is that the fork system call returns a 0 to the child and a nonzero value, the child's PID (Process IDentifier) to the parent Both process nor- normally check the return value, and act accordingly, as shown in Fig. 10-4. Processes are named by their PlDs. When a process is created, the parent is given the child's PID. as mentioned above. If the child wants to know its own PID, there is a system call, getpid. that provides it. PIDs are used in a variety of ways. For example, when a child terminates, the parent is given the PID of the child that just finished. This can be important because a parent may have many children. Since children may also have children, an original process ean build up an entire tree of children, grandchildren, and further descendants. Processes in UNIX can communicate wiih each other using a form of message passing. It is possible to create a channel oetween two processes into which one process ean write a stream of bytes for the other to read, These channels are call-

b STUDY |. UNIX ANDLINUX CHAP. 10 pid = fork(); /* if the fork succeeds, pid > 0 in the parent "I i < 0) { handle. error(! 5 if (pid > 0) { hand'ie_ error] I, /* fork failed (e g , memory or some table is lull) */ /* parent code goes here. /*/ (else { /* child code goes here /*/ } Figure 10-4, Prix..;^ i_re,itiO|l n) L'NIX ed pipes Synchronization is possible because when a process tries to read from an empty pipe il is blocked until data are available Shell pipelines .ire implemented with pipes. When the shell sees a line like sort <f I head it creaies two processes, \ort and head, and sets up a pipe between them in such a way that sort's standard output is connected io head's standard input In lhis way. all the data that son writes go directly to head, instead of going to a file It the pipe tills up, (he system stops running sort until head has removed some data from Ihe pipe Processes can also communicate in another wav software

interrupts. A proc- process can send what is called a signal tu another process. Processes can tell the sys- system what they want io happen when a signal arrives. The choices are to ignore it. to catch it, or io let the signal kill the process (the default tor most signals) It a process elects to catch signals sent to i(. h must specify a signal handling pro- procedure. When a signal arrives, control will abruptly switch to the handler. When the handler is finished and returns, control goes back to where it came from, analogous to hardware I/O interrupts. A process can only send signals io members of its, process group, which consists of its parent (;md further ances- ancestors), siblings, and children (and further descendants). A protess may also send a signal to all members of its process group with a single system tall Signals are also used for Other purposes For example, if a process, is doing floating-point arithmetic, and inadvertently divides by 0. it gets a a SlGFPE (floating-point exception) signal. The signals that arc required by POS1X are listed in Fig.]0-5. Many UNIX systems have additional signals a.s well, but pro- programs using them may not be portable to other versions of UNIX 10.3.2 Process Management System Calls in UNIX Let us now look at the UNIX system tails dealing with process management The main ones are listed in Fig. 10-6 Fork is a good place to start the discussion. Fork is the only way to create a new process in UNIX systems It creates an exact

PROCHSSBS IN UNIX f ^Ignal [_ SIGABRT Sent to abort a process and force Cause re dump SlGALRM i The alarm clock has gone off ___ _ SlGFPE A floating-point errOr has occurred [e.g , division fay 0) ; ;mg has been hung up The phone line the process was __. ., .,. .. The user has hit the DEL key to interrupt the proce The user has hit the key requesting a core dump ot be caught or ignored) a pipe which has no readers irocess has relerenced an invalid rr Used to request that a process terminate gracefully SIGUSR1 [Available tor application-defined purposes SiGUSRg ' Available lor appiication-delined purposes Figure 10-5. The signals required by POS1X duplicate of the original process, including all the file descriptor,, register and everything else. After the fork, the original process and the copy (the parent and child) go their separate ways. All the variables have identical values at the time of the fork, but since the entire parent core image is copied to create the child, subsequent changes in one of them do not affect the oiher one. The fork call returns a value, which is zero in the child, and equal to the child's PlD m the parent. Using (he returned PlD, the two processes can sec which is the parent and

which is the child. In most cases, after a fork, the child will need to execute different code from the parent. Consider the case of the shell. It reads a command from the (erminal forks off a child process, waits for the child to execute the command, and then reads the next command when the child terminates. To wait for ihe child to tin ish. the parent executes a waitpid system call, which just watts until the child ier minates (any child if more than one exists). Waitpid has three parameters The first one allows the caller to wait for a specific child.]t it is -I. any old child (i.e.. the first child to terminate) will do The second parameter is the address of a variable that will be set to the child's exit status (normal or abnormal termination and exit value). The third one determines whether the caller blocks or returns if no child is already terminated- In the case of the shell, the child process must execute the command typed by the user. It does this by using Ihe Bxec system call, which causes its entire core image to be replaced by the file named in its first parameter. A highly simplified shell illustrating the use of fork, waitpid. and exec is shown m Fig. 10-7. In the most general case, exec has three parameters: the name of the file to be executed, a pointer to the argument array, and a pointer to the environment array.

CASE STUDY]■ UNIX AND UMj a child process idenl pid = waitpidjpid, Sstatloc, opts) j Wait for a child tc s = sigactionfsig, &act. Soldac;) s = sigreturn(Scontext) ~ Is = s ig proem as k{ how, &set, Sold) I s = sigpendmg(set) _ I s = sigsuspend(sigmask) i_s_=_kill(P(d1sig)__ residual = alarm (seconds] Replace a process' coreimage Terminate process execution and return status neor changejhe signal mask e set of blocked signals j Replace the signal mask and suspend the pro Send a signal to a process I Set ihe alarm clock | Suspend the caller until the next signal aimg t(, T|ic r< code i while (TRUE) { /* repeat forever /*/ type_prompt{); /* display prompt on the screen */ read_command(command. params); /* read input line from keyboard */ pid = fork(); if (pid < 0) { printff Unable to forkO); continue; /* fork off a chtld process */ /* error condition */ /* repeat the loop *{ if {pid 1= 0) { waitpid(-1, Sstatus, 0);) else (8xecve(command, params, 0); /* child does the work */) I* parent waits for child */ Figure 10*7. A highly simplified ^hcll These will be described shortly, Various library procedures, including cm/, execv, execie, and execve, are provided to allow the parameters to be omitted or specified in various ways,. All ot" these procedures invoke the same underlying system call. Although the system call is exec, there is no library

procedure with this name; one of the others must be used.

PRCK'fcSShS IN UNIX 695 land typed to the shell such cpfilei file2 usl-,1 to copy file! to /,/c2 After the shell has forked, the child locates and exe- executes the file <p and passes it information ahout the files to be copied The mam piograin of <p (and man\ other programs 1 contains the tunctton dccldiation main(argc, argv, envp) where argt is a count ot the number o\ items on the command line, including the program name. For the example above, urgi is 3 The second parameter, urg\'~ is a pointer to Jin array. Element / ot that array is a pointer to the /-th string on the command line. In our example. tf/>fv[O] would point (o the string "cp" Similarly, arg\>\ 11 would point to ihe 5-charaeter siring -filer" and argv\2] would point lo the 5-character siring "tile2". The third parameter of main, envp, is a pointer (o the environment, an array ot strings containing assignments of the form name ~ value used to pass intormation such as the terminal type and home directory name to a program In Fig 10-7. no environment is passed to the child, so the third parameter of execve is ,i zero in this case. It gxgc seems complicated, do not despair, it is the most complex system call All the rest are much simpler. As an example of a simple one. consider exit, which processes should use when they are finished executing. It has one parame- parameter, the exit status @ to 255), which is returned to the parent in the variable \ia\us of the waitpid system call. The low-order byte of w,tu\ contains the termination status, with 0 being normal termination and the other values being various error conditions. The high-order byte contains the child's exit status @ io 2551. as specified in the child's call to extt. For example, if a parent process execute^ ihe statement n =wartpid(-1, Sstatus, 0); it will be suspended until some child process terminates If the child exits with, say, 4 as the parameter to exit, the parent will be awakened with n set lo the child's PlD and utarw. set to 0x0400 (Ox as a prefix means hexadecimal in C). The low-order byte of <,taiu\ relates to signals; the next one is, the value the child returned in its' call to exit. If a process exits and its parent has not yei waited for it, the process enters a kind of suspended animation called the zombie state When the paient finally waits for it. the process terminates. Several sysiem calls relate to signals, which are used in a variety of ways. For example, if a user accidently tells a text editor to display the entire contents of a very long file, and then realizes the error, sonic way is needed to interrupt the editor. The usual choice is for the user to hit some special key (e.g., DEL or

696 CASE STUDY I UNIX AND L|NLX CHAP 10 CTRI-C), which sends a signal to the editor The editor catches the signal and stops the pnnt-out. To announce its willingness 10 catch this, (or any other) signal, the process can use the sigaction system call. The first parameter is the signal to be caught (see Fig 10-5) The second is a pointer to a structure giving a pointer 10 the signal handling procedure, as well us some other bits, and flags. The third one points to a structure where the system returns mformaiion dboul signal handling currently in effect, in case it must be restored later. The signal handler may run for as long as it wants to. In practice, though, sig- signal handlers are usually fairly short When the signal handling procedure is done, it returns to the point from which it was interrupted. The sigaction system call can also be used to cause a signal to be ignored. <>r to restore the default action, which is killing the process Hitting the DEL key is not the only way t(, send a signal. The k,ll system call allows a process to signal another related process The choice of the name "kill" for this system call is not an especially good i^ne. since most processes send sig- signals to other ones with the intention that they bo caught. For many real-time applications, a process needs to be imerriipted alter a specific time interval to do something, such as to retransmit a potentially lost packet over an unreliable communication line. To handle this situation, the alarm system call has been provided. The parameter specifies an interval, in seconds, after which a S1GALRM signal is sent to the process A process may have only one alarm outstanding at any instant. If an alarm call is made with a parameter of 10 seconds, and then 3 seconds later another alarm call is made with a parameter of 20 seconds, only one signal will be generated, 20 seconds after the second call. The first signal is canceled by the second call to alarm.]f the parameter to alarm is zero, any pending alarm signal is- canceled If an alarm signal is not caught, the default action is taken and the signaled process is killed. Technically, alarm sig- signals may be ignored, but that is a pointless thing to do. It sometimes occurs that a process has nothing to do until a signal arrives. For example, consider a computer-aided instruction program that is testing reading speed and comprehension. It displays some text on the screen and then calls alarm to signal it after 30 seconds. While the student is reading the text, the pro- program has nothing to do It could sit in a tight loop doing nothing, but that would waste CPU time that a background process or other user might need A better solution is to use the pause system call, which tells UNIX to suspend the process until the next signal arrives. Thread Management System Calls The first versions of UNfX did not have threads. That feature

was added many years later. Initially there were many threads packages in use, but the pro-

SEC. |O j PROCESSES]N UNIX A97 hfer.ition of thread* package* made writing portable code difficult. Eventually, the system calls used Iq manage threads were standardized as part ot POSIX (PI 003 1c) The POSIX specification did not take a position on whethei threads should be implemented m the kernel or in user space The advantage of having user-space threads is that they can be implemented without having to change th^ kernel ano thread switching is very efficient The disadvantage of user-space threads is that it one thread blocks (e g , on I/O. a semaphore, or a page fault), all the threads in the process block because the kernel thinks there is only one thread and does not schedule the process until the blocking thread is released Thus the calls defined in PI003.1c were carefully chosen to be implementable either way. As long as user programs adhere carefully to the P1003 1c semantics, both implementation1, should work correctly. The most commonly-used thread calls are listed in Fig. 10-S When kernel threads are used, these calls are true system calls, when user threads are used, these calls are implemented entirely in a user-space run- runtime library. (For the truly alert reader, note that we have a typographical problem now If the kernel manages threads, then calls such as "pthread_create," are system calls and following our convention should be set in Helvetica, like this: pthread_create. However, if they are simply user-space library calls, our conven- convention for all procedure names is to use Times Italics, like this: jnhreud_ireate Without prejudice, we will simply use Helvetica, also in the next chapter, in which it is never clear which Win32 API calls are really system calls, ll could be worse: in the Algol 68 Report (here was a period [hat changed the grammar of the language slightly when printed in the wrong font.) Thread call pthread , create pthread .exit pth read, join pthread _mutex_mit pth read _ mutex _ d est roy pthread _mutex_. lock pthread _mutex .unlock pthread_cond_mit pthread _cond .destroy pthread cond_wait pthread_cond_signai 1 1 Create a n Description ew thread in the caller's address space Terminate theca|l,ng ihread Wait for a Create a r Destroy a Lockamu Unlock a Create a c hread to terminate ew mutex mutex tex nutex ondition variable Destroy a condition variable Wart on a Release o condition variable le thread waiting on a condition variable Figure 10-8. The principal POSiX Ihread calls

698 CVSE STUDY t- UNIX AN'P LINUX C HAP K) Lei us briefly examine the thread c.iUs shown in Fig]O-8 The first uill. pth read .Create, creates a new thread. It is1 called by err = pthread _creaie(&tid, attr, function, arg), This call creates a new thread in |he current process' running the code funnum the location pointed to by the first parameter-;. The atit parameter cin be used to. successful completion, one more thread is running in ihe caller s address space than was before the call A thread thai has done ils job and wants to terminate calk pthread_exit A thread can wail for another thread to exit by calling pthread_ioin. If the thread wailed for has already exiled, the pthread pin finishes immediately. Otherwise it blocks Threads can synchronize using locks called mutexes Typically a muiex guards some resource, such as a buffer shared by two threads. To make sure thai only one thread at a time accesses the shared resource, threads aie expected to tock the mutex before touching the resource and unlock ii when they are done. As long as all ihreads obey this protocol, race conditions can be avoided. Mutexes are like binary semaphores, that is, semaphores that can take on only ihe values of 0 and I. The name "mutex" comes from the fact thai mutexes are used to ensure mutual exclusion on some resource Mutexes can be created and destroyed by the calls pthread mutex _mit and pthread mutex__destroy, respectively. A mutex fan be in one of two states: locked or unlocked. When a thread needs to set a lock on an unlocked muiex (using pthread_mutex_lock). the lock is set and the thread continues However, when a thread tries to lock a mutex ihat is already locked, it blocks When the locking thread is finished with ihe shared resource, ii is expected to unlock the corresponding mute* by calling pthread_mutex_unlock. Mutexes are intended for short-term locking such as protecting a shand \an able. They are not intended for long-term synchronization, such as waiting for a iapc drive to become free. For long-term synchronizalion. condition variables are provided. These are created and destroyed by L-all.s to pthread cond init and ptfiread_cond_destroy, respectively. A condiiion variable is used by having one thread wait on it, dnd another thread signal it. For example, having discovered that the tape drive it needs is busy, a thread would do pthread_cond_wait on a condition variable that all the threads have agreed to associate with the tape drive. When Ihe thread using the tape drive is finally done with ii (possibly even hours later), it uses pthread_cond_signal to release exactly one lhread wailing on that condition vari- variable (if any). If no thread ,s waiting, the signal * lost. In other words, condition variables do noi couni like semaphores. A few othor operations arc also defined on threads, muiexes,

and condition variables

hut run' nap A pioce there is , the use s to ke: ss in UNIX also an im :r program rnel mode poitant part u However, « diid begins th more power and also SFC 10 1 PROCESSES IN UNIX 699 10.3.3 Implementation of Processes in UNIX ;berg what you see is the part above the water, jndemeath. Hvery prqcess has a user part that of its threads makes a system tall, it in kernel context, with a different resources. It is still the same thread, i kernel mode stack and kernel mode program counter. These are important because a system call can block part way through, tor example, waiting for a disk operation to complete- The program counter and registers are then saved so the thread can be restarted in kemel mode later. The kerne! maintains two key data structures related to processes', the process table and the user structure. The process table is resident all the time and con- contains information needed for all processes, even those that are not currently present in memory. The user structure is swapped or paged out when its associ- aied process is not in memory, in order not to waste memory on information that is not needed. The Information in the process table falls into the following broad categories 1. Scheduling parameters. Process priority, amount of CPU time con- consumed recently, amount of time spent sleeping recently. Together, these are used to determine which process to run next. 2. Memory image. Pointers io the text, data, and stack segments, or. if paging is used, to their page tables. If the text segment is shared, the text pointer points to the shared text table. When the process is not in memory, information about how to find its parts on disk is here too- 3. Signals. Masks showing which signals are being ignored, which are being caught, which are being temporarily blocked, and which are in the process of being delivered. 4. Miscellaneous. Current process state, event being waited for, if any, time until alarm clock goes off, P1D, PID of the parent process, and user and group identification. The user structure contains information that i not needed when the process is not physically in memory and runnable. For example although it is possible for a process to be sent a signal while it is swapped out it is not possible for it to read a file. For this reason, information about signals mu t be in the process table, so they are in memory all the time, even when the proces i not present in memory. On the other hand, information ahout file descriptors can be kept m the user struc- structure and brought in only wben the process is in memory and runnable.

700 CASLSTirDVI UNIX AND LINUX CHM- 10 The information contained in the user structure includes the following items" 1 Machine registers. When a trup to the kernel occurs, the machine registers (including the floating-point ones, if Used) are saved here. 2. System call slate information about the current sytjem call, includ- including the parameiers. and results 3. File descriptor table. When a system call involving a file descriptor is invoked, the file descriptor is used as an index into this table to locate the in-core data structure (i-node) corresponding to this file 4 Accounting Pointer to a table thai keep;, track ot the user and sys- system CPU time used by the process Some >>stems also maintain lim- limits' here on the amount of CPU time a process may use. the maximum size of its stack, the number of page frames it may consume, and other items, 5 Kernel stack. A fixed stack for use by the kernel part ot the process. Bearing the use ot these tables in mind, it is now easy to explain how processes are created in UNIX. When a fork system call is executed, [he calling process traps to the kernel and looks for a free slot in the process table ftif use by the child. If it finds one, it copies all the information from the parent's process (able entry to the child's entry. It then allocates memory for the child's data .ind stack segments, and makes exact copies ot the parent's data and stack segments there. The user structure (which is often kept iidjacetu to the stack segment), is copied along with the stack. The text segment may either be copied or shared since it is read only. At thi.s point, the child is ready to run When a user types a command, say. Is on the terminal, the shell creates a nuw process by forking oft a clone ot itself. The new shell then calls exec to overlay its memory with the contents of (he executable file h The steps involved arc shown in Fig. 10-9. The mechanism for creating a new process is actually fairly straightforward. A new process table slot and user area are created for the child process and tilled in largely from the parent. The child is given a PID. its memory map is set up. and ii is given shared access to its parent's files. Then its registers are set up and it is ready to run In principle, a complete copy of the address space should be made, since the semantics of fork say that no memory i_s shared between paront and child. How- However, copying memory i.s expensive, so all modern UNIX systems cheat. They give the child its own page tables, but have them point to the parent's pages, only marked read only Whenever the child tries to write on a page, n gets a prelection fault. The kernel sees this and then allocates a new copy of ihe page to the child and marks it read/write. In this way, only pages that are actually written have to .

PROCESSES IN UNIX Allocate PLD for child Set up chfld to share pc Copy page tables tor d£ Set up sharing of open Copy parent's registers Verify (he e: Read and verify the header Copy arguments environ to ke Free ihe old address space Allocate new address space Copy arguments- environ TO st< Reset signals Figure 10-0. luc steps in executing the command l\ lypeJ w (he shell copied. Thts mechanism is called copy-on-write It has the additional btncfu not requiring two copies of the program in memory, thus saving RAM After the child process starts running, the code running there (a cop) of the ell) does an exec, syslem call giving the command name as a parameter. The rnel now finds and verities the executable tile, copies the arguments and Now the new address space musl be created ,ind filled m. It the system sup- supports mapped files, as System V. BSD, and most other UNIX systems do. the new page tables are set up to utdicale that no pages are in memory, except perhaps one stack page, hut that the address space is backed by the executable tile on disk. When the new process starts running, it will immediately gc-t a page fault, which will cause the firsi page of code to be paged in from the executable file In this way. nothing has to be loaded in advance, so programs can start quickly and fault injust those pages they need and no more Finally, the arguments and environ- environment strings are copied to the new stack, the signals are reset, and the registers are initialized to all zeros. At this point, the new command can start tunning Threads in UNIX The implementation ot threads depends on whether they are supported in the kernel or not. I! they are not, such as in 4BSD, the implementation is entirely in a user-space library. If they are, as in System V and Solaris, the kernel has some

 work to do. We discussed threads in a general way m Chap. 2 Here we will jua make a few remarks about kernel thready in UNIX. The main issue in introducing threads is maintaining (he correct traditional UNIX semantics. Fir.st consider fork. Suppose that a process with mulnple (ker- (kernel) threads- does a fork system call. Should all the other threads be created in the new prwes ' Kir the moment,]et us answer thai question with yes. Suppose thai one ot the other threads was Mocked reading from the keyboard- Should the corresponding thread in the new process also be blocked reading from the key- keyboard? If so whn.h one gets the next line typed11 If not. what should thai thread be doing in the new process'.1 The same problem holds for many other (hings threads can do. In a single-threaded process, the problem does not arise because the one and only

thread cannot be blocked when calling fork. Now consider the case that the other threads are not created in the child process. Suppose lhat one of the not-created threads holds a mutex that the one-and-only thread in ihe new process tries lo acquire after doing the fork. The mutex will never be released .ind the one (hread wilt hang forever. Numerous other problems exist too. There is no simple solution. File I/O is anoiher problem area. Suppose that one thread is blocked reading from a file and another thread closes the file or doe.s an Iseek to change the current file pointer. What happens next'1 Who knows? Signal handling is another thorny issue. Should signals be directed at a specific thread or at the process in general1' A SIGFPE (floating-point exception) should probably be caught by the thread that caused i(. What if it does not catch it? Should just that thread be killed, or al] threads" Now consider the S1G1NT signal, generated by the user at ihe keyboard Which thread should catch thai'? Should all threads share a common set of signal masks'? All solutions to these and Other problems usually cause something to break somewhere. Getting the sem.m- Hl of threads right (not lo mention the code) is a nomnvial business. Threads in Linux Linux supports kernel threads in an interesting way that is worth looking at- The implementation is based on ideas from 4 4BSD, but kernel threads were not enabled in that distribution because Berkeley ran out of money before the C library could be rewritten to solve the problems discussed above. The heart of ihe Linux implementation of threads is a new system call, clone, that js not present in any oiher version of UNIX. It ts called as follows- pid = cloneftunction, stack.ptr, sharing_flags, arg). The call creates a new thread, either in the current process or in a new process, depending on sharing^jlags. If the new thread is in the currenl process, it shares the address space with existing threads and every .subsequent write to any byte in the address space by any thread is immediately visible to all the other threads m

 the process. On the other hand, if the address space is not shared, then !he new thread gets an exact cOp> of (he address spate, but subsequent writes by the new thread are nof visible to (he old ones. These semantics are the same as fork. In both cases, the new thread begins executing at function, which is called with arq as its only parameter. Also tn boih cases, the new thread gets its own private stack, with the stack poinier initialized to stack jptr. The sharing -flags parameter is a bitmap that allows a much liner gram of sharing than traditional UNIX systems. Five bits are defined, as listed in Fig. 10- 10 Fach bit controls some aspect of sharing,

and each ot the bits can be set independently ot the other ones The CLONE VM bit determines whether the virtual memory (i.e., address space) is shared wiih the oJd threads or copied If it is set, the new thread ju.st moves in with the existing ones, so the clone call effec- effectively creates a new thread in an exiting process. If the bit is cleared, the new Ihread gets its own address space. Having its own address space means that the effect of its STORE instructions are not \isihle to the existing threads. This behavior is similar to fork, except as noted below. Creating a new address spate is effectively the definition of a new process Flag Y_ Meaning when set ~T Meaning when cleared ! -QNE- VM | Create a new thread _^ 1~C~reate a new process 1 -ONE _ FS I Share umask, root, ana working dirs ' Dojiot share them CLONE_ FILES ["Share the hie descriptors ,_C_opy the file descriptors CLONE. SiGHAND "Tshare the signal handler table Copy the table LcLONE^Pjp j New thread gets old P|D New thread gets own PJD j Figure 10-10. Bits in the \haimf> /fry>s hiimap The CLONE. FS bit controls sharing of the root and working directories and of the umask flag. Even if the new ihread has its own address space, if thu. bit is •set. the old and new threads share working directories This means that a (.all to chdir by one thread changes the working directory of ihe other thread, even though the other thread may have its own address space. In UNIX, a call to chdir by a thread always changes the working directory for other threads in its process, but never for threads in another process. Thus this bit enables a kind of sharing not possible in UNIX- The CLONE _FILES bit is analogous to the CLONE_FS bit. If sot, the new thread shares its file descriptors with the old ones, so calls to Iseek by one thread are visible to the other ones, again as normally holds, for threads with.n the same process but not for threads in different processes Similarly. CLONE .SIGHAND enables or disables the sharing ot the signal handler table between the old and new thread*,. It the table is shared, even arm,ng threads in different address spaces, then changing a handler in one thread affects the handlers in the others. Finally. CLONE_PID controls whether the new thread gets its own PID or shares

 ("HAP 10 its parent's P1D This feature is needed during system booting. User processes are not permitied to enable it. This fine-grained sharing is possible because Linux maintains separate data structures for the various tiems listed at the start of Sec !(>3.3 (scheduling param- parameters, memory image, etc) The process table and user structure just point to these data structures, so it is easy to make a new

process table entry for each cloned thread and have it either point to the old thread's scheduling, memory, and other data structure-, or to copies of them The fact that such fine-grained sharing is possible does not mean that it is useful however, especially since UNIX does not otter [his functionality. A Linux program thai takes advantage of it is then no longer portable toUN|X. Scheduling in UNIX Let us now examine the UNIX scheduling algorithm Because UNIX has always been a multiprogramming system, its scheduling algorithm was designed from the beginning to provide good response to interactive processes It is a two- level algorithm The low-level algorithm picks the process to run next trom the set of processes in memory and ready to run. The high-level algorithm moves processes between memory and disk so that all processes get a chance to be in Each version of UNIX has a slightly different low-level scheduling algorithm, but most of them are close to the generic one we will describe now. The low- level algorithm uses multiple queues. Each queue is associated with a range of nonoverlapping priority values. Processes executing m user mode (the top ot the iceberg) have positive value*. Processes executing m kernel mode (doing system calls) navc negutzve values Negative values have the highest priority and l-irgo positive values have the lowest, as illustrated in Fig. 10-11 Only processes that are in memory and ready to run are located on the queues, since the choice must be made from this set. When the (low-level) scheduler runs, it searches the tjiieuet, starting at the highest priority (i.e.. most negative value) until it finds a queue that is occupied. The first process on thai queue is then chosen and started It is allowed to run for a maximum of one quantum, typically 100 msec. or until it blocks. If a process uses up its quantum, it is put back on the end of its queue, and the scheduling algorithm is run again Thus processes within the same priority range share the CPU using a round-robin algorithm Once a second, each process' priority is recalculated according to a formula involving three components- priority = CTU_umKe + nice + base Based on iis new priority, each process is attached to the appropriaie queue of Fig. 10-11, usually by dividing the priority by a constant to get the queue number. Let us now briefly examine each of the three components of the priority formula.

PROCESSES IN UNIX Wailing to Wailing for Wa.ting tor (Waiting for ie Wailing tor LJserp Userp Userp r disk I/O disk buffer rmmal input mmal oulpul hild to exist ontyO orityl onty 2 User priority 3 lenied by 1. a hlgher Figure 10-11. The UN|X scheduler is based on a multilevel queue Mructure CPU_uwge,

represents the average number of clock licks per second that the process has had during the past lew seconds Every time the clock licks, the CPU usage counter in the running process' process table entry is in< This counter will ultimately be added to the process' priority g numerical value and thus putting tt on a lower-pnonty queue. However, UNIX does not punish a process forever for using the CPU. so CPU-usage decays with time. Different versions of UNIX do ihe decay slightly differently. One way that has been used is to add the current value of CPU-u\age to ihe number of ticks acquired in the past AT and divide the result by 2. This algorithm weights the most recent A7" by v'j. the one before that by W. and so This weighting algorithm is very fast because it jusl has one ad" shift, but other weighting schemes have also been used. Every process has a nice value associated with it The default value :s 0. b the allowed range is generally -20 to +20 A process can set nice to a value in tl range 0 to 20 by the nice system call. A user computing n to a billion placc- the background mighi put this call in his program to be nice to the other u Only the system administrator may ask for better t values from -20 (o -1). Deducing the reason for this rule is left j the reader. When a process (raps to (he kernel to make a system call, it is' entirely possi- possible thai the process has to block before completing the system call and returning to user mode. For example, it may have just done a waitpid system call and have lal servi and one (meaning icrcise for

 to wait for one ot its children lo exit It tnay also have to wait for terminal input or for disk I/O 10 compleie, in mention only a few of thu many possibilities. When it blocks, n is removed from the queue structure, since it is unable to run However, when the event it was waiting tor occurs, it is put onto a queue w i(h a negative value. The etioice of queue is determined by the event it was waiting for In Fig. 10-1], disk I/O is shown as having the highest priority, so a process that has just read or written a block from tlie disk will piobabiy get the CPU within 100 msec. The relative priority otdisk I/O. terminal I/O. etc. is haidwued inio the operating system, and can only be modified by changing some constants in the source code and recompiling the system. These (negative! va]ues are represented by bme m the formula given above and are spaced far enough apart that processes beirt£ restarted for different reasons are dearly separated into dif- different qutues The idea behind this scheme is to get processes out of the kernel fast It a process is trying \o read a disk file, making u wait a second

between read calls will slow it down enormously, ll is far better to let it run immediately after each request is completed, so it can make the next one quickly Similarly, if a process was blocked waiting for terminal input, ii is clearly an interactive process, and as such should be given a high priority as soon as it is ready m order to ensure that interactive processes get good service.]n this light. CPtj bound processes (ie. those on the positive queues) basically get any service thai is letl over when all the I/O bound and mteraciive processes are blocked Scheduling in Linux Scheduling is one ot ihe tew areas in which Linux uses a ditfcreut algorithm from UNIX. We have jusl examined the IJN|X scheduling algorithm, so we will now look at (he Linux aigoriihm. To start with, Linux threads aie kenie! threads, so scheduling is based on threads, not processes. Linux distinguishes three classes of threads for scheduling purposes: 1. Real-time FIFO. 3 Timesharing. Real-time FIFO threads are the highest priority and are not pieemptable except by a newly.readied real-time FIFO thread Real-time round-robin threads are the same as real-time FIFO threads except that they are preemptable by the clock. It multiple real-time round-robin threads are ready, each one is run for iis quantum, after which i(goes to ihe end of the hst of real-time round-robin threads Neithei of these classes is actually real time in any sense. Deadlines cannot be specified and guarantees are not given. These classes are simply higher priority than ihreads in the standard timesharing class The reason Linux calls them reaf time

SEC 10 1 PROCESSES |N UNIX 707 is lhat Linux is conformant to the PI003 4 standard ("real-time" extensions to UNIX) which uses those names, Each Thread has a scheduling priority The default value is 20. but that can be altered using the nice(value) system call to a value of 20-value. Since value must be m the range -20 to +19, priorities always full in the range: I < prior- itv < 40 The intention is thai the qualiiy of service is roughiy proportional to the priority, with higher priority threads getting faster response time and a larger frac- fraction of the CPU time th.in lower priority threads. In addition to a priority, each thread has a quantum associated with it The quantum is lhe numher of clock ticks the thread may continue to run for. The clock runs at 100 Hi. by default, sn each nek u 10 msec, which is called a jiffy. The scheduler uses the priority and quantum as follows It first computes the goodness of each ready thread by applying the following rules. if (class == real-time) goodness = 1000 + priority; if (class = timesharing && quantum > 0) goodness = quantum +

priority; if (class == timesharing && quantum — 0) goodness = 0; Both real-time classes count for the firs! rule. All lhat marking a thread as teal time docs i.s make sure it gets a higher goodness than all timesharing threads. The algorithm has one litllc extra feature: if the process that ran last still has some quantum left, it gets a bonus point, so that it wins any ties. The idea here is that all things being equal, it is more efficient to run the previous process since its pages and cache blocks are likely to be loaded. Given this background, the scheduling algorithm t.s very simple: when a scheduling decision is made, lhe thread with the highest goodness is selected. As the selected thread runs, at every clock lick, its quantum is decremented by] The CPU is taken away from a thread if any of these conditions occur; 1. Its quantum hitsO 2. The lhread blocks on I/O, a semaphore, or something else. 3 A previously blocked thread with a higher goodness becomes ready. Since the quanta keep counting down, sooner or later every ready ihread will grind its quantum into the ground and they will all be 0 However. I/O bound threads thai are currently blocked may have some quantum left. At this pomi the scheduler resets the quantum of all threads, ready and blocked, using the rule: quantum = (quantum/2) + priority where the new quantum is in jiffies. A thread thai is highly compute bound will usually exhausl its quantum quickly and have it 0 when quanta are reset, giving it a quantum equal to its priority. An I/O-bound thread may have considerdble quantum left and thus get a larger quantum nexi time. If nice is not used, the priority will be 20, so the quantum becomes 20 jiffies or 200 msec. On the other

 hand, tor a highly I/O bound thread, it may still have a quantum of 20 left when quania are ,eset. so it Us pnonty .s 20, its new quantum becomes 20/2 + 20 = 30 jiffies If another reset happens before it has spent i tick, next time it get*- a quan- quantum or' 30/2 + 20 = 35 jiffies. The asymptotic value in jiffies is twice the priority. As a consequence of this algorithm, 1/O-bound threads get larger quanta and thus higher goodness than compute-bound threads. This gives l/O-bound threads preference in scheduling Another property of this algorithm is that when compute-bound threads are competing for the CPU, ones with higher priority get a larger fraction of it. To see this, consider two Lonipute-bound threads. A, with priority 20 and B, with pnontj 5 A goes fin,t and 20 ticks later has used up its quantum Then B gets to rim tor 5 quanta. At this point the quanta are reset. A gets 20 and B gets 5. This goes on forever, so A is getting 80^ of the CPU and

B is getting 20% of the CPU. 10.3,4 Booting UNIX The exact details of how UNIX is booted vary from system to system Below we will look briefly at how 4 4BSD is booted, bul the ideas are somewhat similai for all versions. When the computer starts, the iirst sector of the boot disk (the master boot record) is read into memory and executed This sector contains ;i small E|2-byte) program that loads a standalone program called boot from the boot device, usually an IDE or SCSI disk. The boot program first copies itself to a fixed high memory address to free up low memory for the operating system. Once moved, boor reads the root directory of the boot device To do this, ,t must understand the file system and directory format, which it dues Then it reads in the operating system kernel and jumps to it At this point, boot has finished its job and the kernel is running The kernel start-up code ,s written in assembly language .ind is highly machine dependent. Typical work includes setting up the kernel stack, identifying the CPU type, calculating the amount of RaM present, disabling interrupts, ena- enabling the MMU, and finally calling the C-language main procedure to start the main part of the operating system. The C code also ha.s considerable initialization to do. but this is more logical than physical. It .starts out by allocating a message buffer to help debug booi problems. As initialization proceeds, messages are written her*; about what is happening, so they can be fished out after a boot failure b> a special diagnostic program. Think of this as the operating system's cockpit flight recorder (the black box investigators look tor after a plane crash) Next the kernel daia structures, are allocated Most are tixed size, bui a tew, such as the buffer cache and certain page table structures, depend on the amount of RAM available. At this point the system begins autoconfiguratjon Using configuration files telling what kinds of I/O devices might be present, it begins probing the devices to

SEC |0 3 PROCESSES IN UNIX 709 see whith ones actually dpi present If a probed deuce responds to the probe, it i.s added 10 a table or attached devices. It it tail-! to respond, it is assumed to be absent and ignored henceforth Once ihe device lisl has been deiermined, the device drivers must be located. This is one .iiea m winch UNIX systems differ somewhat In particular. 4 4BSD cannot load device dinars dynamically, so any I/O device whose driver was not oi i/NIX. such a<, Linux, can load drivers dynamically (as can all versions of MS- DOS and Windows, incidentally) The argument for <>nd against dynamically loading drivers are interesting and worth stating briefly. The main argument tor dynamic loading is that a single binary can be

shipped to customers with divergent configurations- and have it automatically load the drivers it needs, possibly even over a network. The mam argument against dynamic loading is security If you are running a secure site, such as a bank's database or a corporate Web server, you probably want to make it impossible for anyone to mscit random code into the kernel. The system administrator may keep the operating system sources and object files on a secure machine, do ali system builds there, and ship the kernel binary to other machines over a local area network It drivers cannot be loaded dynamically, this scenario prevents machine operators and others who know the superuser password from injecting malicious or buggy code into the kernel Furthermore, at large sites, the hardware configuration is known exactly at the time the system is compiled and linked Changes are sufficiently r.ire that haung to relink the system when a new hardware device is added is not an issue. Once all the hardware has been configured, the next thing to do is to carefully handcrafi process 0, set up its stack, and run it. Process 0 continues initialization, doing things like programming the real-time clock, mounting the root tile system, and creating inn (process I) and the page daemon (process 2). hut checks its flags to see if it is supposed to come up single user or mul- multiuser. In the former case, it lorks o,t a process that execs the shell and waits for this process to exit. In the latter case, it forks off a process that executes the sys- system initialization shell script, /en/rc. which can do tile system consistency checks, mount additional file systems, start daemon processes, and so on Then it reads, /etc/ttvi, which lists the terminals and some of their properties. For each enabled terminal, it forks off a copy of itself, which does some housekeeping and then execs a program called geit\ Gem sets the line speed and other properties for each line (some of which may be modems, tor example), and then types login. on the terminal's screen and tries to read the user's name trom the keyboard. When someone sits down at the terminal and provides a login name, getry ter- terminates by executing /bin/login, the login program. Login then asks for a

 p.isswotd encrypts it, and verities it agamsi ihc encrypted password stored in the password file /eti/pawwd. If" it is conect. login replaces itself with the user's shell. Nvh,ch then waits l\:i the hist command II it « mtorrcct. login just ask* tor another user name This mechanism is illustrated in Hg 10-12 tor a system with In the figure, the #<7(v proces On terminal I. a user lids typed a logi on terminal 2 causing lhe shell to

type the prompt ic/c) The nsei then typed cp f 1 f2 which hdS caused the shell to fork off a child process and Iiave that process e the cp program. The shell is blocked, waiting tor the child in terminate, at ivl time the shell will type diiothei prompt and read from the keyboard. If the use terminal 2 had typed cc instead of </', the main piogram of the C compiler wi have been started, which in turn would have loiked oft more processes m run various compiler passes 10.4 MEMORY MANAGEMENT IN UNIX The UN[X memory model is straighifuiv.atd. to nuke programs pwiable and io make a possible to implement UNIX on machines wuh widely diffeunu itiemory management units, ranging trom essentially nothing (e.g.. the original IBM PC) to sophisticated paging hardware. This is an area ot the design that has

bjrely clunked m decades It has worked wd We will now examine the model and how lt i- MEMORY MANAGEMENT |N L'NLX > it has not n iplemented 10.4.1 Fundamental Concepts hsety UNIX process has an address space consisting (it ihree segments data, and stack An example process' address space is depicted in Fig 10- as piocess A. The text segment contains the machine instructions that Torn program s exe-eut.ible code. It is produced by the compiler and assemble tutnslating the C, C++. or othor program into machine code The text sejjme normally read-only Sell7 modifying programs went out of style in ab(>ut because they weie mo difficult to understand and debug. Thus the text -.eg neither grows nor shrink-* nt« changes in any other way text, Ma) PhysiCi •iddresi The data segment < data For histori the diita segmen value when the pg F<ir example, in C i storage for the program's variables, strings, t has two parts, lhe initialized data and the uninitialized ns. the latter is kn,>\cn as the BSS. The initialized pan of is variable and i-oinpilcr constants that need an initial s started. r . . _ .. .s possible to declare a character siring and initialize it at the same time. When the program starts up. it expects that the siring has its initial value To implement tins construction, the compiler assigns lhe string a location in the address space, and ensures that when the program is started up. ilns location contains the proper string From lhe operating system's point ot view, initialized daia are not all that different from program text—both contain tin patterns pro- produced by the eompiler thai musi be loaded into memory when the program stans.

 The existence of uninitialized data is actually just an optimization When a global variable is not

explicitly initialized, the semantics of the C language say thai its initial value is 0 In practice, most global variables are not initialled, and arc thus 0. This rauld be implemented by simply having a section of the execut- executable binary file exactly equal to the number of bytei of data, and initializing all of them, including lhe ones that have defaulted m 0 file contains all the explicitly initialized variables following the program tent. The uninitialized variables are all gathered together after lhe initialized ones, so all the compiler has to do is put a word in the header telling how many bytes to allocate To make this point more explicit, consider Pig. 10-I3U) again. Here the pro- program text is 8 KB and lhe initialized data is also 8 KB The uninitialised data (BSS) is 4 KB. The executable file is only 16 KB (text + initialised data), plus a short header that tells ihc system to aliocaie another 4 KB after lhe initialized data and zero it before starting the program. This trick avoids storing 4 KB ol zeros in the executable file Lnhke the texi segment, which cannot change, the data segment can change. Programs modify their variables all the time Furthermore, many programs need to allocate space dynamically, during execution UNIX handles tins by permitting the data segment to grow and shrink as memory is allocated and deallocated. A system call, brk, is available to allow a program to set the size of its data segment. Thus to allocate more memory, a program cm increase the size of its data seg- segment. The C library procedure malloc: commonly used to allocate memory, makes heavy use of this system call. The third segment is the stack segment- On most machines, it starts jt or ueji the top of the virtual addiess space and grows down toward 0. It the stack grows below the bottom of the stack segment, a hardware fault normally occurs, and the operating system lowers the bottom of the stack segment by one page Piograms do not explicitly manage the size of the stack segmcni. When a program starts up. its stack is not etnpt> Instead, n contains all the environment (shell) variables as well as the command line typed to the shell lo invoke it. In this way a program can discover its arguments For example, when the command cp sre dest is typed, the cp program is run with the string "cp sre dest" on the stack, so it can find out the names of the source and destination files The string js represented as an array of pointers to the symbols in the string, to make parsing easier When two users are running the same program, such as the editor, it would be possible, but inefficient, to keep two copies of the editor's program texi in memory at once Instead, most UNIX systems support shared text segments. In Fig IO-l3(a) and Fig.]0-!3(c) we see two processes, A and fi, that have the same

SEC. 10.4 MEMORY MANAGEMENT IN UNIX 713 text segment. In Fig. 10-l3(b) we see a possible layout of physical memory, in which both processes share the same piece of text. The mapping is done by the virtual memory hardware- Data and stack segments are never shared except after a fork, and then only those pages that are not modified. If either one needs to grow and there is no room adjacent to it to grow into, there is no problem since adjacent vinual pages do not have to map onto adjacent physical pages. On some computers, the hardware supports separate address spaces for instructions and data. When this feature is available, UNIX can use it. For exam- example, on a computer with 32-bit addresses, if this feature is available, there would be 232 bits of address space for instructions and an additional 232 bits of address space for the data and stack segments to share. A jump to 0 goes to address 0 of text space, whereas a move from 0 uses address 0 in data space. This feature dou- doubles the address space available. Many versions of UNIX support memory-mapped files. This feature makes it possible to map a file onto a portion of a process' address space so the file can be read and written as if it were a byte array in memory. Mapping a file in makes random access to it much easier than using I/O system calls such as read and write. Shared libraries are accessed by mapping them in using this mechanism. In Fig. 10-14 we see a file that is mapped into two processes at the same lime, at dif-different virtual addresses. >** ui tlllin- • ftmi>W fill An additional advantage of mapping a file in is that two or more processes can map in the same file at the same time. Writes to the file by any one of them are them instantly visible to the others. In fact, by mapping in a scratch file (which will be discarded after all the processes exit), this mechanism provides a

714 CASE STUDY | UNIX AND UWX CHAP ID high-bandwidth way for mulliple processes to share memory In |he most extreme Lasc. two or more processes could map in a file that covers the entire address space, giving a torni ot sharing that is partway between separate processes, and threads. Here the address space is shared (like threads), but each process main- maintains us own open files and signals, tor example, winch is not like threads. In practice, making two address spaces exactly correspond is nevct done, however- 10.4.2 Memory Management System Calls in UNIX POSIX does not specify any system calls tor menior> management. This topic was considered too machine dependent lor standardization Instead, [tie problem was swept under the rug by saying that programs needing dynamic memory management can usc the matloc

librdiy procedure (defined by the ANSI C standard) How mailoc is implemented is thus moved outside ihe scope of the POS|X standard In some circles, this approach is known as passittg the buck In practice, most UNIX systems have system calls for managing memory The most common ones a^ listed in Fig 10-15 Brk specifies the size of ihe data seg- segment by giving the address of the first byte beyond it If the new value is greater than the old one, the data segment becomes larger, otherwise it shrinks I System call i Description . \s = brk(addr) Change data segment size I a = mmap(addr, len, prot, flags, fd, offset) fjwlap a file in 1 Figure 10-15. Some syiteiii tails relating l<> memory inanagenlcni Tin- rt-[inn tude i C -t if an enor lid*, occurred, a and laidi Jit memory addresses. Ilh is ,i Length ji/~()l foniru|*i pro'cclion* fi^S^ Jirc mEiCcllJiicims hiis. itl i' n n\c (icsLfip The mmap and munmap system calls control memory-mapped f,|es. The first parameter to mmap, addr, determines the address a(which the file (or portion thereof) is mapped. It must be a multiple of the page size. II this parameter is 0, the system determines the address iuelf and returns it in <i. The second parameter, len, tells how many bytes to map. It. too, must be a multiple of the page size. The thtrtl parameter, pri/t. determines the protection for the mapped file It c;m be marked readable, writable, executable, or some combination of these. The fourth pardmeter,_/7<i£5, controls whether the file is private or sharable, and whether addr is a requirement or merely a hint. The tilth parametoi. /(/, is the file descriptor for the file to be mapped. Only open files can be mapped, so to map a file in, it must first be opened. Finally, offset tells where in the file to begin the mapping. It is not necessary to start the mapping at byte 0, any page boundary will do.

ShC. 10 4 MEMORY MANAGEMENTS UNIX 715 The olher call, unmap, removes a a mapped file. If only a portion .if the file is unmapped, the rest remain!, mapped. 10.4.3 Implementation of Memory Management in UNIX Prior to 3BSD, mosi UNIX systems' were based on swapping, which worked as follows When more processes existed than could be kept in memory, some of them were swapped out lo disk. A swapped out process was always swapped out in its entirety (except possibly for shared text) A process was thus either in memory or on disk. Swapping Movement between memory and disk was handled by the upper level of the two-level scheduler, known as the swapper. Swapping from memory to disk was initiated when the kernel ran out of free memory on account of one of the follow- following events: 1. A fork

system call needed memory tor a child process. 2. A brk system call needed to expand a data segment. 3. A stack became larger and ran out of the space allocated to it In addition, when it was time to bring in a process that had been on disk too long, it was frequently necessary to remove another process to make room for M To choose a victim to evict, the swapper first looked a(the processes that were blocked waiting for something (e.g., terminal input). Better to remove a process that could not run than one that could. If one or more were lound, the one whose priority plus residence time was the highest was chosen. Thus a process that had consumed a large amount of CPU time recently was a good candidate, as was one that had been in memory lor a long time, even if it was mostly doing I/O. If no blocked process was available, then a ready process was chosen based on the same criteria. Every few seconds, the swapper examined the list of processes currently swapped out to see if any ot them were ready to run. If any were, the one that had been on disk longest was selected. Next, the swapper checked to see if this was going to be an easy swap or a hard one An easy swap was one for which enough free memory currently existed, so that no process had to be removed to make room for the new one. A hard swap required removing one or more processes An easy swap was implemented by just bringing in the process. A hard one was implemented by first freeing up enough memory by swapping out one or mure processes, then bringing in the desired process This algorithm was then repeated until one of two conditions was met: A) no processes on disk were ready to run, or B) memory was so full of processes that

716 CASE STUDY 1 ■ UNIX AND LINUX CHAP 10 had just been brought ,n that there was no room left for any more To prevent thrashing, no process was ever swapped out until ii had been in memory for 2 sec. Free storage ,n memory and on the swap device was kept track of by linked lists of holes. When storage was needed on either one, the appropriate hole l,st was read using the first fit algorithm, *hich returned the first sufficiently large hole it could find. The hole was then reduced m size, leaving only the residual part not needed, list. Paging in UNIX All versions of UNIX for the PDP-11 and Interdata machines, at, well as the initial VAX implementation, were based on swapping, as just described. Starting with 1BSD, however, Berkeley added paging m order to handle the ever-larger programs that were being written. Virtually all UNIX systems now have demand paging, tracing their origins to 3BSD. Below we Will describe the 4BSD design, but the System v one is

closely based on 4BSD and is almost the same The basic idea behind paging in 4BSD is simple: a process need not be entirely in memory in order to run. All tbat is actually required is the user structure and the page tables. If these are swapped [n. the process is deemed "in memorj" and can be scheduled to run. The pages of the text, data, and stack segments are brought in dynamically, one at a time, as they are referenced If the user structure and page table are not in memory, the process cannot be run until the swapper brings them m. Berkeley UNIX does not use the working set model or any otber form of prepaging because doing so requites knowing which pages are tn use and which are not. Because ihe VAX did not have page reference bus, this information was not easily available (although it can be obuined in software at the cost of substan-substantial additional overhead). Paging is implemented partly by the kernel and partly by a new process called ihe page daemon. The page daemon j.s process 2 (process 0 is the swapper, and process 1 is init, as shown in Fig. | (M2). Like all daemons, the page daemon is started up periodically so it can look around to see if there is any work for it to do If it discovers that the number of pages on the list of free memory pages is too low, it initiates action to free up more pages. Main memory in 4BSD is organized as shown m Fig. 10-16 It consists of three parts- The first two parts, the kernel and core map, are pinned in memory (i.e., never paged out). The rest of memory is divided into page frames, each of which can contain a text, data, or stack page, a page table page, or be on the free The core map contains intonation about the contents of the page frames Core map entry 0 describes page frame 0. core map entry I describes page frame I, and so forth. With 1-KB page frames and |6-byte core map entries, less than 2 percent of the memory is taken up by the core map. The first two items in the

MEMORY MANAGEMENT IN UNIX Index Index of Diskb Disk de Block Index ,r Text/ Offset w f next entry previous entry ock number vice number hash code to p roc lable aata/stack thin segmenl Figure 10-16. The core map in 4BSD core map entry shown in Fig. 10-16 are used only when the corresponding page frame is on the free list. Then they are used [o hold a doubly linked list stringing together all ihe free page frames. The next three entries are used when the page frame contains information. Each page in memory also has a location on some disk where it is put when it is paged out These entries are used to find the disk location where the page is stored. The next three entries give the process table entry for the page's process, which

segment it is in, and where it is located in thai segment. The last one shows some of the flags needed by the paging algorithm. When a process is started, it may get a page fault because one or more of its pages are not present in memory. If a page fault occurs, the operating system takes the first page frame on the free list, removes it from the list, and reads the needed page into it If the free list is empty, the process is suspended until the page daemon has freed a page frame The Page Replacement Algorithm The page replacement algorithm is executed by the page daemon. Every 250 msec it is awakened to see if the number of free page frames is at least equal to a system parameter called loisfree (typically set to J/4 of memory). If insufficient page frames are free, the page daemon starts transferring pages from memory to disk until lotsfree page frames are available. If the page daemon discovers that more than lotsfree page frames are on the free list, it knows that it has nothing to

718 CASE STUDY I-UNIX AND LINUX CHAP. 10 do so it just goes back to sleep. If the machine has plenty ot memory and tew active processes the page daemon sleep nearly all the time. The page daemon uses a modified version of the dock algoi ithm. It is a glo- global algorithm, meaning that when removing a page, it does not take into account whose page is being removed. Thus trie number of pages each process has assigned to it varies in time The basic clock algorithm works by scanning the page frames circularly (as though they lay around the circumference of a clock). On the first pass, when the hand points to a page frame, the usage bit is cleared On the second pass, any page frame thai has not been accessed since the first pass will have its usage bit still cleared, and will be put on the free list (after writing it to disk. ,t it is dirty). A page frame 011 the free l,st retains its contents, which can be recovered if that page is needed before it is overwritten. On a machine like the VAX that had no usage bits, when the clock hand pointed to a page frame on the first pass, the software usage bit was cleared and the page marked as invalid ,n the page table When the page was next accessed, a page fault occurred, which allowed the operating system to set the software usage bit. The effect was the same as having a hardware usage bit, but the implementa- implementation was much more complicated and slowei. Thus the software paid the price for poor hardware design. Originally, Berkeley UNIX used the hjsic clock algorithm, but it w t discovered that with large memories, the passes took too long. The algorithm w is then modified to the two-handed clock algorithm, symbolized at the left of Fig. 10-16. With this algorithm, the page daemon maintain*; two

pointers into the core map. When it runs, |t first clears the usage bit at the tront end, and then checks the usage bit at (he back hand, after which it advances both hands. If the two hands are kept close together, then only very heavily used pages have much ot a chance of being accessed between the time the tront hand passes by and the time the back one does. If ihe two hands are 359 degrees apart (meaning the back hand is just ahead of the front hand), we are back to the original clock algorithm Each time the page daemon runs, the hands rotate less than a full revolution, Hil amount depending on how far they have to go to get the number of pages on the free list up to lotsfree If the system notices that the paging rate is too high and the number of free pages is always way below lotsfree. the swapper is started to remove one or more pmcesses from memory so that they will no longer compete for page frames. The 4BSD swap out algorithm !S as follows. First the swapper looks to see if any processes have been idle tor 20 sec or more. If any exist, the one that has been idle the longest is swapped out. If none exist, the four largest processes are exam- examined and the one that has been in memory the longest is swapped out. If need be. the algorithm is repeated until enough memory has been recovered Every few seconds the swapper checks to see if any ready processes on the disk should be brought in. Each process on the disk is assigned a value that is a

 function of how long it has been swapped out, us si/c. the value it set using nice (it any), and how long it was sleeping before being swapped out The function is weighted to usually bring m the process (hat has been out the longesi. unless it is eMremely large. The theory is that bringing in large processes is expensive, so they should not be moved ioo often. Swap-in only occurs if (here are enough free pages, so that when the inevitable page faults Start occurring, there will be page frame-, for them Only the user structure and the page tables are aciually brought in by the swapper. The text, data, and stack pages are paged in as they are used. Each segment of each active process has a place on disk where it resides when it is paged or swapped oui. Data and stack segments go lo a scratch device, but program text is paged in from the executable binary file itsell No scratch copy is used for program text. Paging m System V is fundamentally similar lo Ihat in 4BSD, which is noi entirely surprising since the Berkeley version had been stable and running tor years before paging was added to m V N erthel there are two interest- interesting differences,. First, System V uses the origin n h nd lotk gorithni. instead of the two-handed one. Furthermore, in e of j. g n n 'd page on

the free hsi on the second pass, a page is on p he nused for n consecutive passes- While this decision does of \ t q k as the Berkeley algo- algorithm, it greatly increases the chan eh p g feed will not be needed again quickly. Second, instead of a single variable lotsfree. System V has two variables, mm and max. Whenever the number of free page frames tails below nun, the page daemon is started to free up more pages. The daemon continues to run until there are max free page frames. This approach eliminates a potential instability in 4BSD. Consider a situation in which the number of free page frames is one less than lottfree. so the page daemon runs to free one page and bring it up to lotsjree Then another page fault occurs, using up one page frame, and reducing the number of available frames below lotifree again, so the daemon has to run again. By setting max substantially above mat. whenever the page daemon runs, if builds up a sufficient inventory so it does not have to run again for a substantial time Memory Management in Linux Each Linux process on a 32-bit machine gets 3 GB of virtual address .space for itself, with the remaining 1 GB reserved for its page tables and other kernel data. The kernel's | GB is not visible when running in user mode, but becomes accessible when the process traps into the kernel. The addre.ss space i.s created when the process is created and is overwritten on an exec system call. The virtual address space is divided inlo homogeneous, contiguous, page- aligned areas or regions. That is to say. each area consists of a run of consecutive pages with the same protection and paging properties. The text segment and

 mapped files are examples ot areas (see Fig. 10-14) There can be holes in the virtual address space between the areas. Any memory reference to a hole results in a fatal page fault. The page size is fixed, tor example. 4 KB for the Pentium and 8 KB for the Alpha. Each area is described in the kernel b> a vm_area-.struct entry. All the address so all the pages can be found. When the list gets too long (more than 32 entries), a tree is created to speed up searching The \>m_,area.^trua entry lists the area's properties. These include the protection mode (e.g, read only or read/write), whether it is pinned in memory (not pageable), and which direction it grows ,n (up for data segments, down for stacks). The \m.area._$iruci also records whether the area is private t<i the process or shared with one or more other processes After a fo^, Linux makes a copy of the area list for the child process, but sets up the parent and child to point to the same page tables. The areas are marked as read/write, but the pages are

marked as read ooly If eithei process trie1- to write on a page, a protection fault occurs and the kernel sees that the area is logically writable but the page is not, so it gives the process a copy of the page and marks it read/write. This mechanism is how topy-on-write is implemented The vm-area struct also records whether the area has backing storage on disk assigned, and if so, where. Text segments use the executable binary as back- backing storage and memory-mapped files use the disk file as backing storage. Other areas, such as the stack, do not have backing storage assigned until they have to be paged out. Linux uses a three-level paging scheme. Although thr$ scheme was put into the system for the Alpha, it is also used (in degenerate torm) for all architectures. Each virtual address is broken up into four fields, as shown in Fig. |O-I7 The directory field is used as, an index into the global directory, of which there is a private one for each process. The value found is a pointer to one of the page mid- middle tables, which is again indexed by a field from the virtual address The selected entry points to the final page table, which is indexed by the page field of the virtual address. The entry found here points to the page needed. On the Pen- Pentium, which uses two-level paging, each page middle directory has only one entry, so the global directory entry effectively chooses the page table to use Physical memory is used for various purposes. The kernel itself is fully hardwired; no pan of it is ever paged out. The resi of memory is- available for user pages, the buffer cache used by the file system, the paging cache, and other purposes. The buffer cache holds file blocks that have recently been read or have been read in advance in expectation of being used in the near future. It is dynamic in Size and competes for the same pool of pages as the user pages. The paging cache is noi really a separate cache, but simply the sei of user pages that are no longer needed and are waiting around to be paged out. If a page in the paging cache is reused before it is evicted from memory, it can be reclaimed quickly.

SEC. 10.4 MEMORY MANAGEMENT IN UNIX 721 Page Global Directory 1 Page middle directory Middle N i Page table =ege Offset Word selected Virtual address Figure 10-17. Linux uses three-level page tables. In addition, Linux supports dynamically loaded modules, generally device drivers. These can be of arbitrary size and each one must be allocated a contigu- contiguous piece of kernel memory. As a consequence of these requirements, Linux manages physical memory in such a way that it can acquire an arbitrary-sized piece of memory at will. The algorithm it uses is known as the buddy algorithm

and is described below. The basic idea for managing a chunk of memory is as follows. Initially memory consists of a single contiguous, piece, 64 pages in the simple example of Fig. 10-18(a). When a request for memory comes in, it is first rounded up to a power of two, say 8 pages. The full memory chunk is then divided in half, as shown in (b). Since each of these pieces is still too large, the lower piece is divided in half again (c) and again (d). Now we have a chunk of the correct size, so it is allocated to the caller, as shown shaded in <d). II 1 • i <• W Ml irf I lu.l Figure 10-18. Operation of the buddy algorithm. Now suppose that a second request comes in for 8 pages. This can be satis- satisfied directly now (e). At this point a third request comes in for 4 pages. The smallest available chunk is split (f) and half of ii is claimed (g). Next, the second of the 8-page chunks is released (h). Finally, the other 8-page chunk is released.

722 CASE STUDY I UNIX AND LINUX CHAP 10 Since the two adjacent 8-page chunks that we,,- just treed are buddies, that is, ori- originated from the same |6-page chunk, they arc merged to gel the 16-page chunk hack (i) Linux manages memory using the huddy algorithm, with ihe additional feature ol having an array in which the first clement is the head of a list of blocks ot we I unit, the second element is the head of a list of blocks of si?e 2 units, the next element points to the 4-unit blocks, etc In this way. any power-of-2 block cm he found quickly. This algorithm leads to considerable internal fragmentation because if you w;ini a 65-page chunk, you have to ask for and get a 128-page chunk. To alleviate this problem. Linux has a second memory allocation that takes chunks using the buddy algorithm but then carves slabs (smaller units! from them and manages the smaller units separately. A ihiid memory allocator is .iko avail- ible when the requested memory need only be contiguous m virtual space, hut not tern V. Linux is a demand-paged system with no prepagmg and no working set con- concept ulthc ugh there l a s\stcm tall in which i ust.r can give a hint that a certain page may be needed soon, in the hopes tt will be there when needed) Text seg- segments and mapped files are paged to their respective files on disk. Everything else is paged to either the paging partition, if present, or one of up to 8 fixed- lenglh paging files. Paging files can be added and removed dynamically and each one has a priority. Paging to a separate partition, accessed as a raw device, is moie efficient than paging to a file for several reasons First, the mapping between file blocks and disk blocks is not needed (saves disk I/O reading indirect blocks). Second, the

physical writes can be of any size, not just the tile block size. Third, a page is, always written contiguously to disk; with a paging tile, it may or may not be Pages are not allocated on the paging device or partition until they are needed Each device and file starts wi(h a bitmap telling which pages are free. When a page without backing store has to he tossed out of memory, the highest priority paging partition or file (hat still ha.s space is chosen and a page allocated on it Normally, the paging partition, if present, has higher priority than any paging file The disk location is written into the page table. Page replacement works as follows. Linux tries to keep some pages free _so they can be claimed as needed. Ot course, this pool must be continually replen- replenished, so the real paging algorithm is how this happens. At boot time, hii! starts up a page daemon, kswapd, that runs once a second. It checks to see if there are enough free pages available If so, if goes back to sleep for another second, although it can be awakened early if more pages are suddenly needed. The page daemon's code consists of a loop that it executes up to six times, with increasingly high urgency. Why six" Probably the author of the code thought four was not enough and eight was too many. Linux is sometimes like that.

SEC 1A4 MEMORY MANAGr'MFNT IN UNIX 723 The body of the loop makes calls to ihree procedures, each of which tries to reclaim different kinds of pages. The urgency value is passed as a parameter tel- telling (he procedure how much cttort to expend 10 reclaim sonic pages. LF.su.illy. this means how many pages to inspect be lore giving up. The effect of this algo- algorithm is to ttrst take the easy pages From each category before going after the hard ones. When enough pages have been reclaimed, the page daemon goes back to sleep. The first piocedure tries to reclaim pages from the paging cache and the tile system buffer cache that have not been referenced recently, using a clock like algorithm. The second procedure looks for shared pages that none of the users seems to be using much. The third procedure, which tries to reclaim ordinary usei pages, is the mosi interesting, si) let us take a quick look at it. First, a loop is made over all the processes to see which one has the most pages currently in memory. Once lhat process is located, all of its vim .area struct* are scanned and all the pages are inspected in virtual address order starting from where we left off last time If a page is invalid, absent trom memory, shared, locked in memoiy. or being used fur DMA. it is skipped ll the page has its reference bit on, the bit is turned off and the page is spared. 11 the reference bit is turned off, out goes the page, so this part of the algorithm is simi- similar to clock

(except thai the pages are not scanned in FIFO order) It the page is chosen for eviction and it is clean, it is abandoned immediately. It it is dirty and has a backing store page on disk, it is scheduled for a disk write. Finally, if ,(is dirty and has no backing store page, it goes into the paging cache, from which it might conceivably get a reprieve later if it is reused before it ,s actually paged out. The idea of scanning the pages in virtual address order is based on the hope that pages near each other m virtual address space will tend to be used or not used as a group, so they should be written to disk as a group and later brought in together as a group. One other aspect of the memory management system that we have not yel mentioned is a second daemon, bdflush. It wakes up periodically (and in some cases is explicitly awakuned) to check if too large a fraction of the pages are dirty. If so, it starts writing them back to disk 10.5 INPUT/OUTPUT IN UNIX The I/O system in UNIX is fairly straightforward Basically, all I/O devices are made to look like files and are accessed as such with the same read and write system calls that are used to access all ordinary files. In some cases, device parameters must be set, and this is done using a special system call. We will study these issues "in the following sections.

724 CASE STUDY i- UNIX AND LINUX CHAP 10 10.5.1 Fundamental Concepts Like all computers, those running UNIX have I/O devices such iw disks, primers, and networks connected to them. Some way is needed to allow programs io access these devices. Although various solutions are possible, the UNIX one is to integrate the devices into the file system as what are called special files. Each I/O device is assigned a path name, usually in Mev. For example, a disk might he /dev/hdl. a printer might be/dev/lp, and the network might be /dev/net These special files can be accessed the same way as any other files. No spe- special commands or system calls are needed. The usual read and write system calls will do just fine. For example, the command cp file /dev/lp copies iht file to printer, causing it to be printed (assuming that the user has per- permission to access /dev/lp. Programs can open, read, and write special files the same way as ihey do regular files. In fact, cp in the above example is not even aware that it is printing. In this way, no special mechanism h needed for doing I/O. Special files are divided into two categories, block and character. A block special file is one consisting of a sequence of numbered blocks. The key properly of the block special file is that each block can be individually addressed and accessed. In other words, a program can Open a

block special file and read, say, block 124 without first having to read blocks 0 to |23 Block special files are typ- typically used for disks. Character-special files are normaliy used for devices that input or output a character stream. Keyboards, printers, networks, mice, plotters, and most other I/O devices that accept or produce data for people use character special files, it is not possible (or even meaningful) to seek to block 124 on a mouse. Associated with each special file is a device driver that handles the corresponding device. Each driver has whal is called a major device iiumber that serves to identify it. If a driver supports multiple devices, say, two disks of the same type, each disk has a minor device number that identifies it Together, the major and minor device numbers uniquely specify every I/O device In a few cases, a single driver handles two closely related devices. For example, the driver corresponding to/dev/tty controls both the keyboard and the screen, which is often thought of as a single device, the terminal. Although most character special files cannot be randomly accessed, they orteii need to be controlled in ways that block special files do not Consider, tor exam- example, input typed on the keyboard and displayed on the screen. When a user makes a typing error and wants to erase the last character typed, he presses some key Some people prefer to use backspace, and others prefer DEL. Similarly, to erase the enure line just typed, many conventions abound. Traditionally @ was used, but with the spread of email (which uses @ within email address), many systems

SFC HH INPUT/OUTPUT IN UNIX 725 have adopted CTRL-U or some ofher character. Likewise, to interrupt the run- running program, some special key must be hit. Here too. different people have dif- different preferences,. Rather than making a choice and forcing everyone to use it, UNIX allows all these special functions and many others to be customized by the user. A special system call is generally provided for setting these options Th,s system call also handles tab expansion, enabling and disabling of character echoing, conversion between carnage return and line feed, and similar items. The system call is not permitted on regular files or block special files Networking Another example of I/O it, networking, as pioneered by Berkeley UNIX, and summarised below. The key concept in the Berkeley design is the socket Soc- Sockets are analogous, to mailboxes and telephone wall sockets in that they allow users to interface to the network, just as mailboxes allow people to interface to the postal system and telephone wall sockets allow them to plug in telephones and connect to the telephone system. The sockets' position is shown m Fig. 10-19

Sending procei Receiving proces Figure 10-19, The use lelworkmg Sockets can be created and destroyed dynamically Creating a socket returns a file descriptor, which is needed for establishing a connection, reading data, writ- writing data, and releasing the connection. Each socket supports a particular type cif networking, specified when ihe socket is created. The most common types are |. Reliable connection-onented byte strearn. 2. Reliable connect ion-oriented packet stream. 3. Unreliable packet transmission. The first socket type allows two processes on different machines to establish the

 equivalent df a pipe between them. Byte., arc pun,ped in at one end and they come cut in ihe same order at ihe olhef. The system guarantees that all by to that are sent arrive and in the same oider they were sent. The second type is similar to the first «ne. c*cept that it preserves packet boundaries. If the sender makes five separate call* to write, each tor 5 [2 bytes, and the receiver asks for 2560 bytes, with a iype 1 socket, all 2560 bytes will be returned at once With a type 2 -socket only 512 bytes will be returned l-'our inore calls, are needed to get the rest. The third lype of socket is used to give the user access to the raw network. This tvpc is especially useful for real-time appli- applications, and foi those situations in which the user wants to implement a special- specialized error handling scheme. Packets may be lost oi reordered by the network Theie are no guarantees, as in the first two cases. The advantage ot this mode is higher performance, which sometimes outweighs reliability (e.g.. tor multimedia delivery, m which being fast counts for more than being right) When a socket is created, one of the parameters specifies the protocol to be used for it. For reliable byte streams, the most popular protocol is TCP (Transmission Control Protocol), For unreliable packef-oriented transmission, UDP (User Datagram Protocol) is the usual choice Both arc these are layered on top of IP (Internet Protocol). Ail of these protocols originated with the U S. Dept. of Defense's ARPANET, and now form the basis of the Internet There is no common protocol for reliable packet streams. Before a socket can be used for networking, it must have an iiddrets bound to it. This address can be in one of several naming domains. The most common domam is the Internet naming domain, which uses 32-bit integers foi naming end- pomts in Version 4 and 128-bit integers in Version 6 (Version 5 was an experi- experimental system that never made it to the major leagues). Once sockets have been created on both the source and destination computers, a connection can be established

between them (for connection-oriented communi- communication) One party makes a Itsten system call on a local socket, which creaies a buffer and blocks until data arrive The other one makes a connect system call, giving as parameters the file descriptor for a local socket ,ind the address ot a remote socket. If the remote party accepts the call, the system then establishes a connection between tht sockets. Once a connection has been established, it tuna tons .inalogoiisly to a pipe A process can read and write from it using the file descriptor for its local socket. When the connection is no longer needed, it can be closed m the usual wjy. \ia the close system call, 10.5.2 Input/Output System Calls in UNIX Each I/O device in a UNIX system generally has a special tile associated with it. Most I/O can be done by just using the proper file, eliminating the need lor special system calls. Nevertheless, sometimes there is a need for something that

SEC 10 5 INPUT/OUTPUT IN UNIX 727 is device specific. Prior to POS1X most UNIX systems had a system call ioctl that performed a large number of device-specific actions on special files Over the course ot the years, it had gotten to be quite a mer>s. POSlX cleaned it up by spirt- ting its functions into separate function calls primarily for the terminal. Whether each ont; is a separate system call or they share a single system call or something elsL- is implementation dependent. The first tour listed in Fig [0-20 are used to set and get the terminal speed. Diflerent calls are provided for input and output because some modems operate at split speed. For example, old videotex systems allowed people to access public- databases with short requests from the home to the .server at 75 bits/sec with replies coming back at 1200 bits/sec This standard was adopted at a time when 1200 bitVsec both ways was too expensive tor home use This asymmetry still persists, with some telephone companies ottering inbound service at 1.5 Mbps and outbound service at 384 Kbps, often under the name of ADSL (\symmetric Digi- Digital Subscriber Line). Description ' Set the output speed j Set the input speed - clgetospeedf&termios. speed)) Get the output speedl > = cfgtetispeedf&termios, speed)" I Get the input speed ; = tcsetattr(fd, opt, Stermics) Set the attributes H i = tcgetattr(fd, &termios) _ { Get the attributes J^l Figure 10-20. The mam POSIX talk tor managing ihe terminal The last two calls m (he list are for setting and reading back all the special characters used for erasing characters and lines, interrupting processes, and so on. In addition, they enable and disable echoing, handle flow control, and other related

functions. Additional I/O function calls also exist, but they are somewhat specialized so we will not discuss them further. In addition, ioctl still exists on most UN|X systems. 10.5.3 Implementation of Input/Output in UNIX I/O ill UNIX is implemented by a collection of device drivers, one per dev,ee type. The function of the drivers is to isolate the rest of the system from the idiosyncracies of the hardware. By providing standard interfaces between the drivers and the rest of the operating system, most of the I/O .system can be put into the machine-1ndependeni part of the kernel. When the user accesses a special file, the file system determines the major and minor device numbers belonging to it and whether it is a block special file or

728 CASE STLIDY I LNiX AND LfNUX CHAP 10 a character special file. The major device number is used to index into one of two internal structure arrays, bdevsw for block special files or cdevm for character special files. The structure thus located contains pointers to the procedures to call to open the dev,ce, read the device, write the device, and soon The minor device numbei is passed as a parameter. Adding a new device type to UNIX means adding a new eniry to one ot these tables and supplying the corresponding pro- procedures to handle the various operations on the device. The most important fields in the cdevw an.iy foi a typical system might look as shown m Fig 10-21 Each row refers to a single I/O device (i.e., a single driver). The columns represent ihe functions' that all character driveis must sup- support. Several other functions also exist When an operation is performed on a character special file, the system indexes into the <(/enw array to select the pio|>er row (structure), ihen calls the function in the corresponding structure member (column) to have the work performed. Thus each row contains pointers to func- functions contained in one di iver. [Device i Close Other Memory Keyboard j k_open j k dose Tty tty_open tty close _Pnnter) lp_open Ip close Figure 10-21. Some ot the fields of J typical <*mm tahle Each driver is split into two parts The lop half runs m the context ot" the caller and interfaces to the rest of UNIX The bottom half runs in kernel context and interacts With the device. Drivers are allowed to make calls to kernel pio- cedures for memory allocation, timer management. DMA control, and other things. The set of kernel functions that may be called is defined in a document called the Driver-Kernel Interface. Writing device drivers for UNIX is covered in detail m (Egan and Teixe,ra, 1992). The I/O system is split jnto two major components' (he handling ot block spe- special files and the handling of character special files

We will now look at each of these component^ m turn The goal of that part of the system that does I/O on block special files (e g.. disks) is to minimize the number of actual transfers that mus! be done To accom- accomplish this goal, UNIX systems have a buffer cache between the disk drivers and the file system, as illustrated in Fig [0-22. The buffer cache is a table ,n the ker- kernel for holding thousands of the most recently used blocks. When a block is needed from a disk tor any purpose (i-node. directory, or data), a check is, tire! made to see it it is in the buffer cache. If so. ti ,s taken from there and a disk access is avoided. The buffer cache greatly improves system performance.

INPUT/OUTPUT IN UNIX Reading/writing files -t 1- Cooked Raw interface iteriace to lo/dev/tiy /bev/tly Bult.ic.ch. Disk drivers Figure 10-22. The UNIX I/O sysie If the block U not in the buffer cache, ,t !s read from the disk into the buffer cache and from there, copied to where it is needed. Since the buffer cache has room for only a fixed number of blocks, some algorithm is needed to manage it. Usually, [he blocks in the cache are linked together in a linked list. Whenever a block is accessed, it is moved to the head of the list. When a block must be removed from ihe cache 10 make room for a new block, the one at the rear of the chain is selected, since it is the least recently used block. The buffer cache works for writes as well as for reads. When a program writes a block, it goes to the cache, not to the disk, Only when ihe cache fills up and the buffer must be reclaimed is the block forced out onto the disk. To avoid having blocks stay too long in the cache before being written to the disk, all the dirty blocks are written to the disk every 30 seconds. For decades. UNIX device drivers have been statically linked into the kernel so they were all present in memory when the system was booted every time. Given the environment m which UNIX grew up, mostly departmental minicomput- minicomputers and then high-end workstations, with their small and unchanging sets of I/O devices, th,s scheme worked well. Basically, a computer center built a kernel

 containing drivers for the I/O devices and that was it If next year it bought a new disk, it relinked the kernel. No big deal. With the arrival of I inux on the PC platform, suddenly all that changed. The number of I/O devices available on the PC is orders of magnitude larger than on any minicomputer. In addition, although ail Linux users have (or can easily get) the full source code, probably the vast majority would have considerable difficulty adding a driver, updating cdevsw or

bdevs,\%. relinking ihe kernel, and then instal- installing it as the bootable system (not to mention dealing with the aftermath of build- building a kerne! that does not boot). Lmux solved this problem with ihe concept of loadable modules. These are chunks of code that can be loaded into the kernel while the system is running. Most commonly these are character or block device drivers, but they can also be entire file systems, network protocols, performance monitoring took, or anything be set up. Fourth, the appropriate driver switch lable has to be updated to handle the new major device type- Finally, ihe driver is allowed to run to perform any device-specific initialization it may need. Once all ihese steps are completed, the dnver is fully installed, the same as any statically installed driver. Some modern UNIX systems also support loadable modules now, too 10.5.4 Streams Since character special files deal with character streams and do not move blocks of information between memory and disk, they do not use the buffer cache Instead, in the first versions of UNIX, each character device driver did all the work it needed for its device. However, in the course of lime it became clear that many drivers were duplicating code present m other drivers such as code for buffenng, flow control, and network protocols. Two different solutions were developed for structuring and modularizing the character drivers. We will now briefly examine each of them in turn. The BSD solutions builds on data structures present in classical UNIX systems called C-lists. shown as small boxes in Fig. 10-22. Each one is a block of up to 64 characters, plus a count and a pointer to the next block. As characters arrive from terminals and other character devices, they are buffered in a chain of these blocks. When a user process reads from /dev/ttv (e.g.. standard input), the characters are not passed directly from the C-list to ihe process. Instead, they pass through a piece of kernel code called a line discipline. The line discipline act-, like a filter, accepting the raw character stream from the terminal driver, processing it, and

 producing what is called a cooked character stream. In the cooked stream, local line editing has been done (i.e. erased characters and lines have been removed), carriage returns have been mapped onto line feeds, and other special processing has been completed. The cooked stream is passed to the process. However, if the process wants to interact on every character, it can pu[the line in raw mode, in which case the line discipline will be bypassed. Output works in n similar way, expanding tabs [o spaces, converting line feeds to carnage returns + line feeds, adding

filler characters following carriage returns on slow mechanical terminals, and so on. Like input, output can go through the line discipline (cooked mode) or bypass it (raw mode). Raw mode is especially useful when sending binary data to other computers over a serial line and for GUIs. Here, no conversions are desired. The System V solution, streams, devised by Dennis Riichie. is more general and is shown in Fig. [0-23. (System V also has a buffer cache for block special files, but -since that is essentially the same as in BSD it is not shown here.) The basic idea of a stream is to be able to connect a user process to a driver and be able to insert processing modules into the stream dynamically, during execution To some extent, a stream is the kernel analog of pipelines in user space. Computer Q Fipire 10-23. An example of st

732 CASE STUDY I UNIX AND LINUX CHAP 10 A stream always has a stream head on the top and connects 10 a driver cm the bottom. As many modules as needed can be inserted into the stream. Processing can be bidirectional, so each module may need a section i'oi reading (from vhe driver) and one for writing (to the driver). When a user process writes to a stream, code in the stream bead interprets the system call and packs, the data into stream buffers that are passed from module to module downward, with each module performing whatever transformations it is supposed to Each module maintains a read queue and a write queue, so buffers are processed in the correct order, Modules have well-defined interfaces, defined by the streams infrastruc- infrastructure, so unrelated modules can be plugged together The example of Fig. 10-23 shows how streams are used when using the Inter- Internet TCP protocol over iwo different kinds of local area networks. Elhernet and Token ring. In particular, it illustrates another important feature or streams— multiplexing. A multiplexing module can take one stream and split it into multi- multiple streams or can take multiple streams and merge them into one stream The [P module in this configuration does both. 10.6 THE UNIX FILE SYSTEM The most visible part of any operating system, including UNIX, is the tile sys- system. In the following sections we will examine the basic ideas behind the UNIX file system, the system calls, and how the file system is implemented. Some of these ideas derive from MuLTiCS, and many of them have been copied by MS- DOS, Windows, and other systems, but others are unique to UNIX The UNIX design is especially interesting because it clearly illustrates the principle of Small is Beautiful. With minimal mechanism and a very limited number of s> stem calls, UNIX nevertheless

provides a powerful and elegant file system 10.6.1 Fundamental Concepts A UNIX file is a sequence of 0 or more bytes containing arbitrary information No distinction is made between ASCII files, binary files, or any other kinds of files. The meaning of the bits in a file is entirely up to the file's owner- The sys- system does not care. File names were originally restricted to 14 arbitrary characiers, but Berkeley UNIX increased the limit lo 255 characters, and this ha.s been adopted by System V and most other version;, as well. All ihe ASCII characters except NUL are allowed in file names, so a file name consisting of three carnage returns is a legal file name (but not an especially convenient one). By convention, many programs expect tile names to consist of a base name and an extension, separated by a dot (which counts as a character). Thus prog.i is typically a C program, prog.ftO is typically a FORTRAN 90 program, and prog.o

SEC. 10 6 THE UNIX FILE SYSTEM 733 is usually an object file (compiler output). These conventions are not enforced by the operating nb m p"l nd other programs expect them. Exten- Extension s may h r n leng h nd II ay have multiple extensions, as in proa.jdvaZ wl h p b bly np d Java program. Files c bfc, pig he de orjes for convenience. Directories are stored as til d o | g n be reated like files. Directories can con- contain subdir | d g h hi! file system. The root directory is called / and Hy 1 bd rectones. The / character is also used to separate di n n h he n n /usr/asi/x denotes the file x located in the director h h |f 1 / d rectory. Some of the major directories near ihe lop of the tree are shown in Fig. 10-24. Directory Contents Binary (executable) programs Special files tot I/O devices ^neoussysternjil (User directories Figure 10-24. Sor There are two ways to specify file names in UNIX, both to the shell and when opening a file from within a program. The first way is using an absolute path. which means telling how to get to the file starting at the root directory. An exam- example of an absolute path is Msr/asi/books/mos2/ckap-10. This tells the system to look in the root directory for a directory called usr. then look there for another directory, asl. In turn, this directory contains a directory books, which contains the directory mos2 which contains the file chap-10. Absolute path names are often long and inconvenient. For this reason. UNIX allows users and processes to designate the directory in which they are currently working as the working directory. Paih names can also be specified relative to the working directory. A path name specified relative to the working direciory is a relative path. For example, if /usr/ast/b(>oks/mos2 is the working

directory, then the shell command cp chap-10 backup 1 has exactly the same effect as the longer command cp /usr/ast/books/mos2/chap-10 /usr/ast/books/mos2/baekup1 It frequently occurs that a user needs to refer to a file that belongs to another user, or at leas! is located elsewhere in the file tree. For example, if two users are sharing a file, it will be located rn a directory belonging to one of them, so the other wiil have to use an absolute paih name to refer to it (or change ihe working

734 CASF; STUDY 'Ugh. . UNIX AND LINUX CHAP |0 directory). If this is long enough, it may become irritating to have to keep typing it. UNIX provides a solution to this problem by allowing users to make a new directory entry that points to an existing file. Such an entry '<» failed a link As an example, consider the situation of Fig. lO-25(a). Fred and Lisa are working (ogether on a project, and each one needs frequent access to the other's files. If Fred has Msr/fred as his working directory, he can refer to the file x in Lisa"s directory as /usr/tixa/x. Alternatively, Fred can create a new entry in his directory as, shown in Fig. 10-25(b), after which he can use x to mean /mrfiisafx. Figure tn-25. (a) Before linking (b) After linking In the example just discussed, we suggested that before linking, the only way for Fred to refer to Lisa's file x was using its absolute path Actually, this is not really true. When a directory is created, two entries, . and ., are automatic ally made in it. The former refers to the working directory itself. The latter refers to the directory's parent, that is. the directory in which it itself is listed. Thus from /usr/fred, another path to Lisa's file x is ../lna/x. In addition to regular files, UNIX also supports character special riles and block special files. Character special files are used to model serial I/O devices such as keyboards and printers. Opening and reading from /dev/ity reads from the keyboard; opening and writing xo/dev/lp writes to the printer. Block special files, often with names like /dev/hdl, can be used to read and write raw disk partitions without regard to the file system. Thus a seek to byte k followed by a read will begin reading from the k-th byte on the corresponding partition, completely ignor- ignoring the i-node and file structure. Raw block devices are u.sed for paging and swapping, by programs that lay down file systems (e.g.. mkfs). and by programs that fix sick file systems (e.g.,fsck), for example. Many computers have two or more disks. On mainframes at banks, for exam pie, it is frequently necessary to have 100 or more disks on a single machine, in order to hold the huge databases required. Even personal computers normally have

SEC 10.6 THE UNIX FILE SYSTEM 735 dl least two disk*—a hard disk and a diskette drive When there are multiple disk driven, the question arises of how to handle ihem, One solution is to put a self-contained tile system on each one and just keep [hem separate. Consider, tor example, the situation depicied in Fig. 10-261 a). Here we have a hard di«k. which we wilt call C, and a diskette, which we will call A. Each has its own root direciory and files. With this solution, the user has to specify both the device and the file when anything other than the default is iiccdcd. For example, to copy the file x to the directory d, (assuming C: is the default), one would type cp A./x /a/d/x This is the approach taken by systems like MS-DOS. Windows 98, and VMS. Hard disk Diskette Hard disk Figure 10-26. <)Separa The UNIX solution is to allow one disk to be mounted in another disk"s file tree. In our example, we could mount the diskette on the directory /b, yielding the file system of Fig. IO-26(b). The user now sees a single file tree, and no longer has io be aware or which file resides on whiuh device. The above copy command now becomes cp /b/x /a/d/x exactly the same as it would have been if everything had been on the hard disk in the first place. Another interesting property of the UNIX file system is locking. In some applications, two or more processes may be using the same file at the same time, which may lead to race conditions. One solution is to program the application with critical regions. However, if the processes belong to independent users who do not even know each other, this kind of coordination is generally inconvenient. Consider, for example, a database consisting of many files in one or more dir- directories that are accessed hy unrelated users. It is certainly possible to associate a

736 CASE STUDY I UNIX AND LINUX CHAP. 10 semaphore with each directory or file and achieve mutual exclusion by having processes do a down operation on the appropriate semaphore before accessing the data. The disadvantage, however, is thai a whole directory or file is then made inaccessible, even though only one record may be needed For this reason, POS1X provides a flexible and fine-grained mechanism for processes to lock as little as a single byte and as much as an entire file m one indi- indivisible operation. The locking mechanism requires the caller to specify the file to be locked, the starting byte, and the number of bytes. It ihe operation succeeds, the system makes a table entry noting that the bytes in question (e.g., a database record) are locked. Two kmds of locks are provided, shared locks and exclusive locks If a por- portion of a file already contains a shared lock, a

second attempt to place a shared lock on it is permitted, but an attempt to put an exclusive lock on it will fail. It' a portion of a file contains an exclusive lock, all attempts to lock any part of that portion will fail until the lock has been released. In order to successfully place a lock, every byte in the region to be locked muit be available When placing a lock, a process must specify whether it wants to block or not in the event that the lock cannot be placed If it chooses to block, when the exist- existing lock has been removed, the process is unblocked and the lock is placed It the process chooses not to block when it cannot place a lock, ihe system call returns immediately, with the status code telling whether the lock succeeded or not. Locked regions may overlap. In Fig. |0-27(a) we see that process A has placed a shared lock on bytes 4 through 7 of some file. Later, process B places a shared lock on bytes 6 through 9, as shown in Fig. |0-27(b) Finally. C locks bytes 2 through 11- As long as all these locks are shared, they can co-exist Now consider what happens if a process tries lo acquire an exclusive lock to byte <> of the file of Fig. 10-27(c), with a request to block if the lock tails Since iwo previ- previous locks cover this block, the caller will block and will remain blocked until both B and C release their locks. 10.6.2 File System Calls in UNIX Many system calls relate to files and the file system. First we will look at the system calls that operate on individual fiies Later we will examine those that involve directories or the file system as a whole. To create a new file, the creat call can be used. (When Ken Thompson was once asked what he would do dif- differently if he had the chance to reinvent UNIX, he replied that he would spell creat as create this time.) The parameters provide the name of the file and the protec- protection mode. Thus fd = creat("abc", mode); creates a file called abc with the protection bits taken from mode. These bits determine which users may access the file and how. They will be described later.

 I C's shared lock so opwis i[tor wnttng. To ccesstul creat reiurns as Us :riptor. fd in the example is truncated to length 0 and The Creat call not only creates a new file, but at allow subsequent system calls to access the file, a su result a small nonnegatjve integer called a file des. above. It a creat is done on an existing file, that file its contents are discarded. Now let us continue looking at the prtncipal tile system calls, which dre listed in Fig. 10-28. To read or write an existing file, the file murf first be upened using Open This call specifies the file name to be opened and how it is to be opened: for reading, writing, or both. Various options can be specified as well

Like creat, the call to open returns a file descriptor that can be used for reading or writ- writing. Afterward, the file can be closed by close, which makes the file descriptor available for reuse on a subsequent creat or open. Both ihe creat and open calls always return the lowest numbered file descriptor not currently in use When a program starts executing in the standard way. file descnpiors 0. I, and 2 are already opened for standard input, standard output, and standard error, respectively. In this way. a filter, such as the sori program, can just read us input from file descriptor 0 and write its output to file descriptor 1, without having to know what files they are. This mechanism works because the shell arranges for ihese values lo refer to the correct (redirected! files before the program is started. The most heavily used calls are undoubtedly read and write. Each one has thret: parameters- a file descriptor (telling which open file to read or write), a

738 CASE STUDY 1 UNIX AND LINUX CHAI' 10 | "system call Description | ■ fdTcreat(narne, mode) _ _ One way lo create a new file j jTd^opennile.how ..)__ _ |__Open aj.le jorj^ading^writingor both ' s = cbse(ld) _ Close an open Me _ _ | ' n = tead(fd, buffer, nbyies) 'Read data from ajile into a buffer ' n=wrte(td, butter nbytes) j Wrtte data tTom_ab\i«et_inte_a tile i | position - lseek[(d. offset, whence) i Movejhejile pointer _ 1 s = stat(name Sbuf) Get a die's status information I | s = istat(fd, &buf) ' Get a file's statusjnformation ' s_= pi_pe(&fd[pl)_ _ __ I Create a pipe_ __ j i s. = 1crrtl(t<i. cmd) _^ File locking and other operations errui hjs iiu-uned.]d is a file descriptor. Jnd \w.tn<»t it a file offset flit p,ir;iiii tiers should be sL-lfcKplanatory butter .iddrtss (teflnig where to put the data t.r get the data from), and d aium (tel- (telling how man> bytes fo transfer) That it, all there ^s It i\ .i very simple dcsLjzn A typical call \<, n = readffd, buffer, nbytes), Although most prograiis lead and write tiles sequentially, some piograms need to be able to access any pan of a file at random Associated *ith each tile is a pointer that indicates the current position in the lilc When reading (writing) sequentially, it nonnally points to the nexi bue ic' be read (written) It the pointer is at. sa>, 4096. before 1024 bytes are read, it will automatically be moved to position pointer, so that subsequent calls to read oi write can begin .mywhere in the file, or even beyond (he etid of it It ts called Iseek to avoid conflicting w,th seek, a now-obsolete call that wa^ foritici ly u^etl on 16-bit coniputers for seeking. Lseek has ihree paramcteis. the first one is the file descriptor foi the lilc; the second one is a file positton. the third one tells whefher the file position is iclative to

the beginning of the tile, the current portion, or the did ot the file The value returned by lseek is (he absolute position in the file after the file pointer was changed. Slightly ironically, Iseek is the only Hie system call that can never cause an. actual disk seek because all it does is update the current file position, winch is a number in memory. For each file, UNIX keeps track of the file niode (regular, direeiory, special file), size, time of last modification, and ofher information. Programs can ask ki see this information via the Stat system call. The first parameter is the (lie name The second one is a pointer to a structure where the informal ion requested is to be

SEC 10.6 THF. UNIX FILL SYSTEM 9 pat The fields ,n the structure are shown in Fig 10-29 The fstat call is the same as stat except that it operates on an open file (whose name may noi be known) rathei Hun on a path name. Device ttiejile is on _ 1-node number (which file on the device) File mode (includes protection inlormaiion) Number of links to the file Identity ot the file's owner Group the file belongs to File size (in bytes) Creation time Time of last access Time of last modification j Figure 10-29. The fields reldmcd hy ihe stat syj.,em call The pipe system call is used to create shell pipelines. It creates a kind of pseudofile, which buffers the data between the pipeline components, and reiums file descriptors tor both reading and writing (be buffer. In a pipeline such as sort <in I head -30 file descriptor I (standard output) in the process running sort would be set (by the shell) to write io the pipe and file descriptor 0 (standard input) in the process run- running head would be set to read from the pipe. In this way. sort just reads from file descriptor 0 (set to the file in) and writes to file descriptor 1 (the pipe) without even being aware ihat these have been redirected, it they have not been redirected, sort wilt automatically read from (he keyboard and write to the screen (the default devices). Similarly, when head reads from file descriptor 0. it is read- reading the data sort put into the pipe buffer without even knowing that a pipe is in use. This is a clear example where a simple concept (redirection) with a simple implementation (file descriptors 0 and 1) leads to a powerful tool (connecting pro- programs in arbitrary ways without having (o modify them at all) The last system call in Fig. 10-28 is, fcntl- If is used to lock and unlock files and perform a few other file-specific operations Now let us look at some system calls that relate more to directories or the file system as a whole, rather than just to one specific Hie. Some common ones are listed in Fig. 10-30. Directories are created and destroyed using mkdir and rmdir,

respectively. A directory can only be removed if it is empty. As we saw in Fig. 10-25, linking to a tile creates a new directory enlry that points to an existing file. The link system call creates the link. The parameters specify the original and new names, respectively. Directory entries are removed

CASE STUDY 1. UNIX AND LINl'X I &_= rrikdirjpath, mode_)_ s = rmdir(path) _ I s - link(oldpath, newpath) Create a link 10 anjjxtstvng. fite_ Jl l 5ctory_ | J I 5 - chdir(path) _ I Change the working Qirectory ir - opendir(path) Open a directory tor reading _ \ rs_^ciosedif(d,r)__ , Close a dtredory ___ | dirent = readdir(dir) Read one directory entry , rewindOirfdir) \ Rewind a directory so_it can be reread] ; The r« m and due . duap- has the can be i rector} Figure 10-30. Some iystcm i_alls relating to di if ,\t] error has occurred, cfir idemitres d direci<t jntr; The parameters should be self explanatory. with unlink When the lasi link to a tile it, removed, (he file is am deleted. For a file ihai has never been linked, the first unlink causes , peiir The working directory is changed by the endir system call Doing effect of changing ihe interpretation of relative path names.. The last four calk of Fig. 10-30 are for reading directories They opened, closed, and read, analogous to ordinary files Eat.h call to readdir exactly one directory entry in a fixed format There is no w,iy for users, to ' a directory (in order to maintain the integrity o) the file system). Files added to a directory using creat or link and removed using unlink. There is way to seek to a specific file m a directory, but rewinddir allows an open di lo be read again from the beginning. 10.6.3 Implementation of the UNIX File System In this section we will describe ihe implementation of the traditional UNIX file system Afterward, we will discuss the Berkeley improvements Other ide systems are also in u.se All UNIX systems can handle multiple disk partitions, each with a different file system on it. A classical UNIX, disk partition contains a file system with the layout illus- illustrated m Fig, 10-31. Block 0 is not used by LNIX and often contains code io hopt the computer. Block I is the super-block. It contains critical information about the layout of the file system, including the number of i-nodes, the number of disk blocks, and the start of the list of free disk blocks (typically a few hundred entries-) Destruction of the KUperblock will render the file system unreadable Following the superblock are the i-nodes (short for index-nodes, but never called that, although some lazy people drop the hyphen and call them inodes).

THE UNIX PILE SYSTEM Figim- 10-31. IVk layout i NlX s They are

numbered from I up 10 some maximum. Each i-node is 64 bytes long and describe^, exactly one file. An i-node contains accounting information (including all the information returned by Stat. which simply takes, it from the i- node), as well as enough information to locate all the d,sk blocks that hold the file's data Following the i-nodes are the daia blocks All the files and directories are stored here. If a file or directory consists of more than one block, the blocks need not be contiguous on the disk. In fact, the blocks of a large file sire likely to be spread all over the disk. |t is this scatter that the Berkeley improvements were designed to reduce. A directory in the traditional (i.e., V7) file system consists of an unsorted col- collection of 16-byte entries Each entry contains a file name (up to 14 arbitrary characters), and the number of the file's i-node. as shown in Fig. 6-37. To open a file in the working directory, the system just reads the directory, comparing the name to be looked up to each entry until it either finds the name or concludes that it is not present. It the file is present, the system extracts the i-node number, and uses this as an index into the i-node table (On disk) lo locate the corresponding i-node and bring it into memory. The i-node is put in the i-node table, a kernel data struciure that holds all the i-nodes for currently open files and directories The forrmit of the i- node entries vanes somewhat from UNIX version to UNIX version. As a bare minimum, all the fields returned by the stat system call must be present to make stat work (see Fig. 10-29). In Fig. 10-32 we show the formal used by all AT&T versions trom Version 7 through System V Looking up an absolute paih name such as M\r/nsl/fi.le is slightly more com- complicated. First, the system locates the root directory, which generally uses j-node 2, especially when i-node 1 is reserved for bad block handling. Then it looks up the string "usr" m the root directory, to get the i-node number of the /u\r direc- directory. This ,-node is then fetched, and the disk blocks are extracted from it. so the /usr directory can be read and searched for the- string "ast". Once this entry is found, the i-node number foi the Amr/ust directory can be taken from it. Armed with the i-node number of the /uir/<ist directory, this i-node can be read and the directory blocks located Finally, "file" is looked up jud its i-node number tound. Thus the use of a relative path name ,s not only more convenient for the user, but it also saves a substantial amount of work tor the system.

{-ASF STUDY 1: UNIX AND LINUX Generation number (increm e the tile was last accessed e the file was last modified ie the i-node was last changed

(except the oiher ti Figure 1U~32. Slniclure t in Syst i V Let us now see how the system reads a tile. Remwnber that a typical call to the library procedure for invoking the read system call looks like this: n = read(fd, buffer, nbytes), When ihe kerne! gets control, all it has to start with are these three parameters, and the information in its internal tables relating to the user One of the items m the internal tables is the tile descriptor array. It is indexed by a file descriptor and contains One eniry for eiich open file (up to the maximum number, usually about 20). The idea is to start with this file descriptor and end up with the corresponding i-node. Let us consider one possible design: just put a pointer to the i-node in the file descriptor table. Although simple, unfortunately, this method does not work. The problem is as follows. Associated with every file descriptor is a file position that tells at which byte the next read (or write) will start. Where should it go11 One possibility is to put it m the i-node table However, this approach fails if two or more unrelated processes happen (o open the same file a(the same tune because each one has its own file position. A second possibility is to put the file position in the file descriptor table. In that way, every process that opens a file gets its own private fi|c position Unfor- Unfortunately, this scheme fails too. but the reasoning is more subtle and has to do with the nature of file sharing in UNIX. Consider a shell scripi. j, consisting of two commands, pi and p2. to be run in order. If the shell script is called by the com- command line s >x is expected that pi wil also, starling at the plan write its output to x. and then p2 will write ; where pi stopped.

ShC 10 6 THE UNIX FILE; SYSTEM 743 When the shell fo,ks off pi, x is initially empty, so pi just starts writing at tile position 0. However, when pi finishes, some mechanism is needed to make sure thai the initial file position that p2 sees is not 0 (which it would be it the file posi- position were kept in the file descriptor table), but the value pi ended with The way this is achieved is shown m Fig. 10-33. The tr,ck is to introduce a new table, the- open file description table between the file descriptor table and the i-node table, and put the Hie position (and read/write bit) there In this figure, the parent is the shell and the child is first pi and laier p2 When the shell forks off pi, its user structure (including the file descriptor table) is an cxacr copy of the shell's, so both of them point to the same open file description table entry. When pi finishes, the shell's file descriptor is still pointing to the open file description containing pl\ file position. When the shell nou forks off p2, the- new child automatically inherits the file position, without either it or the shell even

having to know what Ihat position it.. Figure 10-33, The relation between the tile descriptor tabli. the , description tdbte. and ihe i-node table However, if an unrelated process opens the file, it gets its description entry, wjih its own file position, which is precisely ■ own open file hat is needed.

744 CASE STUDY I UNIX AND UNl'X CHAP 10 Thus ihe whole point ot the open file description table is to allow a parent -md child to -ihare a file position, bul to provide unrelated processes with thciT own values. Getting back to the problem ot doing ihe read, we have now shown how the file position and i-node arc located The i-node contains the d,sk addressed of the first |0 blocks of the file If the file position tails m the first 10 blocks, the block is read and the data are copied (o the user For files longer than 10 blocks, a field m the i-node contains fhe disk address of a single Indirect block, as shown in Fig 10-33. This block contains the disk addresses of more disk blocks. For example it a block is 1 K.B and a disk address is 4 bytes, the single indirect block can hold 256 disk addresses. Thus this scheme works for files of up to 266 KB in loial Beyond thai, a double indirect block ,s used it contains the addresses of 256 single i|id,rtct blocks, each ot which holds the addresses of 256 data blocks. This mechanism is sufficient to handle files up to 10 + 2|6 blocks, F7.1 \9,104 byte) If even this is not enough, the i-node has space tor a triple Indirect block Jts pointers point to many double mdireci blocks. The Berkeley Fast File System The descnpljon above explains how the classical UNIX tile system works. Lei us now take a look ai the improvement Berkeley made to it. First, directories w'ere reorganized Instead of limiting file names to 14 characters, the limit was set to 255 characters Of course, changing the struciure of all ihe directories meant that programs that naively read directories (which was and js still legal! and which expected a sequence of 16-byte entries, no longer worked To provide portability across the two kinds of directories, Berkeley pro\ided the s>stem calls Opendir. Closedif, readdir, and rewinddir to allow programs to read directories without having U) know their internal structure Long file names and these system calls for reading directories were later added to all other versions of UNIX and to POSIX The BSD directory structure allows file names up to 2i5 character- and is1 illustrated in Fig. |0-34. Each directory consists of some integral number of disk blocks so thai directories can be written atomically to the disk Within a direc- directory, entnes for files and directories are in unsoned ordfr. with each entry directly following the one before it Entries may uot span disk blocks, so

often ihc-re are Each directory entry in Fig. 10-34 consists ol four fixed-length fields and one variable-length field The first field is the i-node number, 19 for the file iotossal. 42 for the file voluminous, and 88 for the dnectory bigdir Next comes a field tel- telling how big the entry is (in bytes), possibly including some padding after the name. This field is needed to find the next entry for 'he case that the file name is

THE UNIX FILE SYSTEM 1, FllO voluminous «8; »«:W»K»'"^ _ ' : V////////////// padded by an unknown length. That is the meaning of the arrow m Fig, 10-34. Then comes ihe [ype field: file, directory- etc. The last fixed field k the length of the actual file name ,n bytes. 8. 10, and 6 in this example. Finally, comes the file name itself, terminated by a 0 byte and padded out to a ? 2-b|t boundary Addi- Additional padding may follow that In Fig. 10-34(b) we see the same direciory after the entry for voluminous has been removed. All that ,s done is increase the Size of the total entry field for colossal, turning the former field for voluminous into padding for the first entry. This padding can be used for a subsequent etitr>. of course. Since directories arc searched linearly, it can take a long time to find an entry at ihe end of a large directory. To improve search performance, name caching was added to BSD. Before looking up a name, the system firsi checks the caihe. If a cache hit occurs, the linear search call be bypassed. The second major change Berkeley made to the file system was ihe division of the disk up into cylinder groups, each with its own siiperblock. i-nodes, and data blocks. The idea behind this change is to keep the ,-node and data blocks of a file close together, to avoid long seeks Whenever possible, blocks arc allocated in the cylinder group containing the i-node The third change was ihe introduction of two block siz^ instead of just one. For storing large files, it is more efficient to have <t small number of large blocks raiher than many small ones On the other hand, many UNIX files are small, so having only large blocks would be wasteful of d,sk space. Having (wo sizes allows efficient transfers for large files and space efficiency for small ones. The pnee paid is considerable extra complexity in Ihe code

746 CASfc study i unix and linux chap. 10 The Linux File System The initial Linux file system was the MlNlX file system. However, due to the fact that it limited tile names to 14 character (in order to be compatible with UNIX Version 7) and its maximum file size was 64 MB. there was interest in better file systems almost from the beginning The firs! improvement was the Ext tile

system, which allowed file names of 255 characters and files of 2 GB, but it was slower than the MIN1X file system, so the search continued for a while. Eventually, the Ext2 file system was invented with long file names, long files, and better performance, and that has become the main tile system However. Linux still supports over a dozen file systems using the NFS file system (described in the nexi section) When Linux is linked, a choice is offered of which file systems should be built into the kernel Other ones can be dynamically loaded as modules during execution, if need be Ext2 is very similar to the Berkeley Fast File system with some small differ- differences. The Ext2 disk layoul is shown in Fig 10-35 Instead ot using cylinder groups, which mean alinost nothing with today's Virtual disk geometries, it divides the disk mto groups of blocks without regard to where the cylinder boun- boundaries fall. Each block group starts with a superblock. which tells hov. many blocks and i-nodes there are. gives the block size, eti. Next comes the group descriptor, which contains, information about the location ot the bitmaps, the number of free blocks and i-nodes m the group and the number of directories in the group. This information is important since Ext2 attempts to spread directories evenly over the disk Bool Block group 0 Block group 1 Slock group 2 B,«S,oW3 Block group 4 Figure 10-35. Layout of the t mux Fxt2 lut system Two bitmaps keep track of the fTtx blocks and tree modes, respectively. Each map is one block long Witb a |-KB block, this design limits a block group to 8192 blocks and 8192 ,-nodes. The former is a real restriction but the latter is not in practice. Next ccune the i-nodes themselves. Each i-node is 128 bytes, twice the size of the standard UNiX i-nodes of Fig. [0-32. The extra space is used as follows. Instead of 10 direct block addresses and 3 indirect block addresses, Linux allows |2 direct and 3 indirect addresses Furthermore, the addresses have been extended from 3 bytes to 4 bytes, lo handle disk partitions larger than 224 blocks (|6 GB), which was, already a problem in UNIX. In addition, fields are

SEC 106 THE UNIX FILE SYSTEM 747 reserved for pointers to access control lists for a fintr degree of protection, but these- are slill on the drawing board. The rest of the i-node is reserved for future use. History shows thai unused bits do not remain that way for long. Operation of ihe file system is similar to the Berkeley fast file system. One difference with Berkeley, however, is, the use of standard I KB throughoul The Berkeley fast file systems uses JS-KB blocks and then fragments them into 1-KB pieces if need be. Ext2

just does ii the easy way. Like (he Berkeley system, when a file grows. Ext2 tries to pot the block in the same block group as, ihe rest of the file, preferably right after the previous last block. Also. when a new file k added to a directory. Ext2 tries to put it into the same block group as its directory New directories are scattered uniformly across the disk directories are scattered uniformly across the disk Another Linux file sysiem is the /pirn- (process) file system, an idea originally devised in the 8th edition of UNIX from Bell Labs and later copied m4 4BSD Jnd System V. However. Linux extends, the ided in several ways The basic concept is that for every process in the system, a directory is created in /prec The mime of the directory is the proce;,s PID expressed as a decimal number for example. /pwc/619 is the directory Corresponding to the process with PID 619 In mis directory are files that appear to contain information about the process, Mich a-, its command line, environment strings, and signal masks In fact, these files do not e\is(on the disk. When they are read, the system retrieves the information from the actual process as needed and ieturns it in a standard format Many of the Linux extensions relate to other files and directories located in /proc. They contain a w[de variety of information about the CPU, disk partitions, devices, interrupt vectors, kernel counters, hie systems- loaded modules, and much more. Unprivileged user programs imy read much of [his information to learn about system behavior in a safe way Some of these files may be written to in order to change system parameters. 10.6.4 NFS: The Network File System Networking has played a major ro|e m UNIX right from the beginning (the first UNIX network was built to move new kernels trom the PDP-11/70 to tht In- terdaia 8/32 during the port to the laier) In this section we Will examine Sun Microsystem's NFS (Network File System), which is used on all modern UNIX systems (and some non-UNIX systems) to join ihe file systems on separate com- computers into one logical whole. Three aspects of NFS are of interest, the architec- architecture, the protocol, and the implementation We will now examine thest- in turn. NFS Arctaiteclure The basic idea behind NFS is to allow an arbitrary collection of clients and servers to t,hare a common file system. In many cases, all the clients and servers are on the same LAN, but this is not required. It is also possible to run NFS over

 a wide area network if the server is far from [he client. For simplicity we will speak of clients and servers as though they were on distinct machines, but m fact, NFS allows every

machine to be both a client and a server at the same time Each NFS server exports one or more of its directories for access by remote clients When a directory is made available, so are all of its subdirecciories. so in fact, entire directory trees are normally exported as a unit. The list of directories a server exports is maintained in a file, often /etc/exports, so ihese directories can be exported automatically whenever the server is booted. Clients access exported directories by mounting them. When a client mounts a (remote) directory, it becomes part of its directory hierarchy, as shown in Fig. 10-36. Of I nounled file In this example, client 1 has mounted the bin directory of server 1 on its own bin directory, so it can now refer to the shell as /bin/sh and get the shell on server 1. Diskless workstations often have only a skeleton file system (in RAM) and gtt all their files from remote servers like this,. Similarly, client 1 has mounted server 2's directory /projects on its directory M\r/astMark so it can now access file a as lusr/asi/work/projl/a. Finally, client 2 has also mounted the projects directory and can also acce.ss file a. only as /mnt/projl/a. As seen here, ihe same file can have different names on different clients due to its being mounted in a different place in the reactive trees. The mount point is entirely local to the clients; the server does not know where it is mounted on any of its clients.

SEC ,0 6 THE UNIX RLE SYSTEM 74V NFS Protocols Since one of the goals of NFS ,s to support a heterogeneous system, with clients and servers possibly running different operating systems on different hardware, it is essential that the interface between the clients and servers be well defined. Only then is it possible for anyone to be able to write a new client imple-implementation and expect it to work correctly with existing servers, and vice versa. NFS accomplishes this goal by defining two client-server protocols, A proto- protocol is a set ot requests sent by clients to servers, along with the corresponding replies sent by the servers back to (he clients. The first NFS protocol handles mounting A client can send a path name to a server and request permission to mount that directory somewhere m its directory hierarchy The place where it is to be mounted is not contained in the message, as the server does not care where it is to be mounted. If the path name is legal and the directory specified has been exported, the server returns a file handle lo the client. The file handle coniain.s fields uniquely identifying the file system iype, the disk, the i-node number of the directory, and security information. Subsequent calls to read and write files in the mounted directory

or any of its subdirectories use the file handle. When UNIX boots, it rnns the /ett/rc shell script before gomg multiuser. Commands to mounf remote file systems can be placed m this script, thus automatically mounting the necessary remote file systems before allowing any logins. Alternatively, most versions of UNIX also support automounting. This feature allows a set of remote directories to be associated with a local directory. None of these remote directories are mounted (or their servers even contacted) when ihe client is booted. Instead, the first time a remote file is opened, the operating system sends a message to each of the servers. The fira one to reply wins, and its directory is mounted, Automounting has two principal advantages over static mounting via the /etc/re tile. First, if one of the NFS servers named in /etcfrc happens to be down, it is impossible to bring the client up. at least not without some difficulty, delay, and quite a few error messages. If the user does not even need that server at the moment, all that work is wasted Second, by allowing the client to try a set of servers in parallel, a degree of fault tolerance can be achieved (because onl> one of them needs to be up), and the performance can be improved (by choosing the first one to reply—presumably the least heavily loaded). On the oiher hand, it is tacitly assumed that all the tile systems specified as alternatives for the automount are identical. Since NFS provides no support for file or directory replication, it is up to the user to arrange for all the file systems to be the same. Consequently, automounting is most often used for read-on]} file systems containing system binaries and other files that rarely change. The second NFS protocol is for directory and file access. Clients can send messages to servers to manipulate directories and read and write files. Also, they

 can also access file attributes, such a-s file mode, size, and time of last modifica- modification. Most UNIX system calls are supported by NFS. with Ihe perhaps surprising exception of open and close. The omission of open and close is noi an accident, il is fully intentional. It is not necessary to open a file before reading it, nor to close it when done. Instead, Ip read a file, a client sends the server a lookup message containing the file name, with a request co look it up and return a tile handle, which is a structure that iden- identifies the file (i.e., contains a file system identifier and i-node number, among other data). Unlike an open call, this lookup operation does not copy any infor- information into internal system tables The read call contains the file handle of rhe file to read,

the offset in the file to begin reading, and the number of bytes desired. Each such message is self-contained. The advantage of this scheme is that the server does not have lo remember anything about Open connections in between calls to it. Thus if a server crashes and then recovers, no information about open files is lost, because there is none. A server like this that does not maintain state information about open file;, is said to be stateless. Unfortunately, the NFS method makes it difficult to achieve the exact UNIX file semantics. For example, in UNIX a file can be opened and locked so that other processes cannot access it. When the file is closed, the locks are released. In a stateless server such as NFS, locks cannot be associated with open files, because the server does not know which files are open. NFS therefore needs a separate, additional mechanism to handle locking. NFS uses the standard UNIX protection mechanism, with the rwx bits for the owner, group, and others (mentioned in Chap. I and discussed in detail below). Originally, each request message simply contained the user and group IDs of the caller, which the NFS server used to validate the access In effect, it trusted the clients not to cheat. Several years' experience abundantly demonstrated thai such an assumption was—how shall we put if1—naive. Currently, public key cryptog- cryptography can be used to establish a .secure key far validating the client and server cm each request and reply. When this option is enabled, a malicious client cannot impersonate another client because it does not know that client's secret key NFS Implementation Although the implementation of the client and server code is independent of the NFS protocols, most UNIX systems use a three-layer implementation similar to (ha! of Fig. [0-37. The top layer is the system call layer This handles calls like open, read, and Close. After parsing the call and checking the parameters, it invokes Ihe second layer, the Virtual File System (VFS) layer. The task of the VFS layer is to maintain d table With one entry for each open file, analogous fo the table of t-nodes for open tiles in UNIX In ordinary UNIX. an i-node is indicated uniquely by a (device, i-node number) pair. Instead, the VFS layer has an entry, cailed a v-node (virtual i-node), for every open file. V-

 Figure 10-37. The NFS layer jlrutlme nodes are used to tell whether the file is local or remote For remote file*, enough information is provided to be able to access them For local tilts, the file system and i-node are recorded because modem UNIX systems can support multiple file systems (e.g., V7, Berkeley FFS, ext2fs. /proc, FAT. etc:).

Although Vr-S was invented to support NFS, most modern UNIX systems now support it as an integral part of ihe operating sysiem, even if NFS is not used To see how v-node* are used, let us trace a sequence of mount, open, and read system calls. To mount a remote file system, the system admimstratoi (Or /etc/rr) calls the mount program specifying the remote directory, the local direc- directory on which it is to be mounted, and other information The mount program parses the name of the remote directory to bt mounted and discovers the name of the NFS server on which the remote directory is located. It then contacts that machine asking for a file handle for the remote directory. If the directory exists and is available for remote mounting, the server returns a file handle for the direc- directory. Finally, it makes a mount system call, passing the handle to the kernel. The kernel then constructs a v-node for the remote directory and asks the NFS client code in Fig, 10-37 to create an r-node (remote i-node) in its internal tables to hold the file handle. The v-node points to the r-node. Each v-node jn the VFS layer will ultimately contain either a pointer to an r-node in the NFS client code, or a pointer to an i-node in one of the local file systems (shown as dashed lines in Fig. 10-37). Thus from the v-node it is possible to see if a file or directory is local

752 CASE STUDY I. UNIX AND LINUX CHAP 10 or remote. If it is local, the correct tile system and i-node can be located. If it is remote, the remote host and file handle can be located. When a remote file is opened on the cheni, at some point during the parsing of the path name, the kernel hits the directory on which the remote file system is mounted. It sees that this directory is remote and m the directory's v-nodc finds the pointer to the r-node. It then asks the NFS client code to open the file. The NFS client code looks up the remaining portion of the path name on the remote servei associated with thu mounted directory and gets back a file handle for it. It makes an r-node for the remote file in its tables and reports hack to the VFS layer, which puts in its tables a v-node tor the file thai points to the r-node. Again here we see that every open file or directory has a \-ruide that points to eilher an r-node or an i-node The caller is given a file descriptor for the remote tile. This file descnptoi is mapped onto the v-node by tables in the VFS layer. Note that no table entries are made on the server side. Although the server is prepared to provide file handles upon request, it does not keep track of which tiles happen to have file handles out- outstanding and which do not When a file handle is sent to it f<,r file access, it checks the handle, and it it is valid, uses it. Validation can

include verifying an authentication key contained in the RPC headers, if security is enabled. When the file descriptor is used in a subsequent system call, for example, read, the VFS layer locates the corresponding v-node, and trom that determines whether it is local or remote and also which i-node or r-nodt describes it. It then sends a message to the server containing the handle, the tile offset (which is main- maintained on the client side, not the server side), and the byte count. For efficiency reasons, transfers between client and server are done in large chunks, nOimally 8192 bytes, even if fewer bytes are requested. When the request message arrives at the server, it is passed to the VFS layer there, which determines which local file system holds the iequesied file. The VFS layer then makes a call to that local file system to read and return the bytes. These data are then passed back to the client After the client's VFS layer has got- gotten the 8-KB chunk i(asked for, it automatically issues a request tor the next chunk, so it will have it should ii be needed shortly. This feature, known a_s read ahead, improves performance considerably. For writes an analogous path is followed trom client to sei ver Also, transfers are done in 8-KB chunks here too. It a write system call supplies fewer than 8 KB bytes of data, the data are just accumulated locally Only when the entire 8 KB chunk is full is it sent to the .server. However, when a file is closed, all of its data are sent to the server immediately. Another technique used to improve performance rs caching, as in ordinary UNIX. Servers cache data to avoid disk accesses, but this is invisible to the client1!. Clients maintain two caches, one for file attributes (l-nodes) and one tor file data. When either an i-node or a file block ,s needed, a check is made io see if ii can be satisfied out of the cache, [f so, network traffic can be avoided.

SEC 10 6 THE UNIX FILE SYSTEM 753 While client caching helps performance enormously, it also introduces some nasty problems. Suppose that two clients are both caching the same tile block and thai one of them modifies it. When the other one reads the bloek, it gets the old (stale) value. The cache is not coherent. Given the potential severity of this problem, (he NFS implementation does several things to mitigate it. For one, associated with each cache block is a timer. When the timer expires, the entry is discarded Normally, the timer is 3 sec for data blocks and 30 sec for directory blocks. Doing this reduces the risk some- somewhat. In addition, whenever a cached file is opened, a message is sent to the server lo find out when (he file was lasi modified. If the last modification occurred after the local copy was' cached,

the cache copy is discarded and the new copy fetched from the server. Finally, once every 30 sec a cache timer expires, and all the dirty (i.e., modified) blocks in the cache are sent to the server. While not perfect, these patches make the system highly usable in most practical Cir- 10.7 SECURITY IN UNIX Despite the name, UNIX has been a multiuser system almost from the begin- beginning. This history means that security and control of information was built in very early on.]n the following sections, we will look at some of the security aspects of UNIX. 10.7.1 Fundamental Concepts The user community for a UNIX system consists of some number of registered users, each of whom has a unique UID (User ID). A UID is an integer between 0 and 65,535- Files (but also processes, and other resources) are marked with the UID of their owner. By default, the owner of a file is the person who created the file, although there is a way to change ownership- Users can be organized into groups, which are also numbered with |6-bit integers called GIDs Group IDs). Assigning users to groups is done manually, by the system administrator, and consists of making entries in a system database telling which user is in which group. Originally, a user could be in only one group, but some versions of UNIX now allow a user to be m more than one group at the same time. For simplicity, we will not discuss this feature further. The basic security mechanism in UNIX is simple. Each process carries the UID and G1D of its owner. When a file is created, it gets the UID and GID of the creating process. The file also gets a set of permissions determined by the creat- creating process- These permissions Specify what access the owner, the other members of the owner's group, and the rest of the users have to the file. For each of these three categories, potential accesses are read, wrtte, and execute, designated by the

 only letters r. w, and x, respectively The ability to execute a tile makes sense oy that file is an executable binary program, of course. An attempt to execute a file that has execute permission but which is not executable (i e . does not start with a valid header) will fail with an error. Since ihere are three categories of users and 3 bits per category. 9 bits arc sufficient to represent the access rights Some examples of these 9-bn numbers and iheir meanings are given in Fig. 10-38 Binary 111000000 111111000 110100000 110100100 111101101 oooooooo 0000001t1 Symbolic rwxrvux rvu-r rvu-i r— rwxr-xr-X rwx Allowed file accesses Owner can read, write, and execute Owner and group can read,

write, and execute Owner can read and write; group can read Owner can read and write, all others can read Owner can do everything, rest can read and execute Nobody has any access Only outsiders have access (strange, but legal) Figut 18. Sor mi* 11 The first two entries m Fig. 10-38 are cl owner's group full access, respectively. The ne read the file but not to change it, and prevent fourth entry is common for a data file the owne the fifth entry is the usual one tor a publicly ar, allowing the owner and the t one allows the owner's group to outsiders from any access. The wants to make public. Similarly, ailable program. The sixth entry all access to all users. This mode is sometimes, used for dummy files u>,ed tor mutual exclusion because an attempt to create such a file will tail if one already exists. Thus if multiple processes simultaneously attempt to create such, a file as a lock, only one of them will succeed. The last example is strange indeed, since it gives the rest of the world more access than the owner. However, its existence follows from the protection rules. Fortunately, there is a way tor the owner to subsequently change the protection mode, even without having any access to the file itself. The user with UID () is special and is called the superuser (or root). The superuser has the power to read and write all files' in ihe system, no matter who owns them and no matter how they are protected Processes with UID 0 also have the ability to mate a small number of protected system calk denied to ordinary users Normally, only the system administrator knows the superuser's password, although many undergraduates consider ii a great sport to try to look for security flaws in the system so they can log in as the superuser without knowing the pass- password. Management tends to frown on such activ&ty Directories are files and have the same protection modes that ordinary tiles do except that (he x bits refer to search permission instead of execute permission. Thus a directory with mode rwxr-xr-x allows its owner to read, modify, and

SEC. 107 SECURITY IN UNIX 755 search [he directory, bu! allows others only to read and search it, but not add or remove files from it. Special files corresponding to the I/O devices have the same protection bits as regular fiJes,. This mechanism can be used to limit access to I/O devices. For example, ihe printer special file, /dev/lp, could be owned hy root or by a special user, daemon, and have mode nv-- to keep everyone else from directly accessing the printer. After all. if everyone could just print at will, chaos would result. Of course, having /dev/lp owned by. say, daemon w,[h protection mode rw means that nobody else can use the printer. While this would save

many innocent trees from an early death, sometimes users do have a legitimate need to print something. In fact, there is a more general piobtem of allowing con- controlled access to alt I/O devices and other system resources. This problem was solved by adding a new protection bit. the SETUID bit to the 9 protection bits discussed above. When a program with the SETUID bit on is executed, the effective UID for that process becomes the UID of the executable file's owner instead of the UID of the user who invoked it. When a process attempts to open a file, it is the etfeciive UID that is checked, not Ihe underlying real UID. By making the program that accesses the printer be owned by daemon but with the SETUID bit on. any user could execute it, and have the power of dae- daemon (e.g., access to /dev/lp) but only to run that program (which might queue print jobs for printing in an orderly fashion). Many sensitive uNix programs are owned by the root bui with the SETUID bit on. For example, the program that allows users to change their passwords. passwd, needs to write in the password file. Making the password file publicly writable would not be a good idea. Instead, there is a program that is owned by the root and which has the SETUID bit on. Although the program has complete access- to the password file, ,t w,li only change the caller's password and not per- permit any other acL-es.s to the password file. In addition to the SETUID bit there is also a SETG1D bit that works analo- analogously, temporarily giving the user the effective G1D of the program. In practice, this bit is rarely used, however. 10.7.2 Security System Calls in UNIX There are only a small number of system calls relating to security The most important ones are listed in Fig. 10-39 The most heavily used security system call is chmod. It k used to change the protection mode For example, s = chmod("/Lisr/ast/newgame\ 0755); sets nenxame to rviwr-vr-t t,o thai e\eryone can run it (note that 0755 is an ctal constant, which is convenient since the protection bits come m groups of 3 bits). Only the owner of a file and the superuser can change it.s protection bus.

CASE STUDY I. UNIX AND LINUX i = cnmouijjaui,iii««/ __ , Change a files protection_modi ~~ s(path. mode) ^ '-—' s-chmod(path, mode) jj^=geiuid()__ __ gid = getegid;) _____ I s^chown(path, owner, group) Check access using the real UlD and GID 1 Gel the real UlD Get the effective UlD Gel the real GID _ Gel the effective GJD Change owner and group _ __ j I s = setuid<U|d) Set the U IP _ , | '~~' SetttreGlO _^ 1 Figure 10-39. Some system calls relating io security The rcium code a is -I if an error has occurred, uid and gtd are the

UlD and GlD, respectively The parameters should be self explanatory The access call test*, to see if a particular access would be allowed using the real UID and GlD. This system call is needed to avoid security breaches in pro- programs that are SETU1D and owned by the root. Such a program can do anything, and it is sometimes needed for the program to figure out if the user is allowed to perform a certain access. The program cannot just try it. because the access will always succeed. With the access call the program can find oui if the access is allowed by the real UlD and real GlD. The next four system calls return the real and effective UlDs and GIDs. The last three are only allowed for the superuser- They change j file's owner, and a process' UlD and GID. 10.7.3 Implementation of Security in UNIX When a user logs in. the login program, login (which is SETUID root) asks for a login name and a password. It hashes the password and then looks in the password file, /etc/passwd^ to see if the hash matches the one there (networked systems work slightly differently). The reason tor using hashes is to prevent the password from being Stored in unencrypted form anywhere in the system If the password is correct, the login program looks, in /eic/p<nswd to see the name of the user's preferred shell, possibly sh, but possibly some other shell such as ah or ksh. The login program then uses setuid and setgid to give itself the user's UlD and GID (remember, it started out as SETuiD root). Then it Opens the keyboard for standard input (tile descriptor 0). the screen for standard output (file descriptor 1), and the screen for standard error (file descriptor 2) Finally, it executes the preferred shell, thus terminating itself, Ai this point the preferred shell is running with the correct UlD and GID and standard input, output, and error all set to their default devices. All processes that

SEC 10 7 SECURITY IN UNIX 757 it forks off (i.e., commands iyped by ihe user), automatically inherit the shell's U1D and G]D, so they also w[ll have the correct owner and group. All files ihey create also get these value*,. When any process attempts to open a tile, the sysiem first cheeky the protec- protection bils in the file's i-node against the caller's effective UID and effective GID to see if the access' is permitted. If so. the tile is opened and a file descriptor returned. If not. the file is not opened and -1 is returned. No checks are made on subsequent read or write calls. As a consequence, if the protection mode changes after a tile is already ope", the new mode will no! affect processes that already have the file open. Security in Linux is essentially the same as in UNiX. All the UNIX security features are implemented and there is little else

in this area. 10.8 SUMMARY UNIX began life as a minicomputer timesharing system, but is now used on machines ranging from notebook computers to supercomputers. Three interfaces to it exist: the shell, the C library, and the system calls themselves. The shell allows users to type commands for execution These may be simple commands, pipelines, or more complex structures. Input and output may be redirected. The C library contains the system calls and also many enhanced calls, such as printf for writing formatted output to files. The aciual system call interface is lean and mean, roughly |00 calls, each of which doing what is needed and no more The key concepts in UNIX include the process, the memory model, I/O, and the file system. Processes may fork oft subprocesses, leading to a tree of processes. Process management in UNIX uses iwo key data structures, the process table and the user structure. The former is always in memory, but the latter can be swapped or paged out. Process creation is done by duplicating the process table entry, and then the memory image. Scheduling is done using a priority-based algorithm that favors interactive processes The memory model consists of three segments per process: text, data, and stack. Memory management used to be done by swapping, but is now done by paging in mosi UNIX systems. The core map keeps track of the state of each page, and the page daemon uses a clock algorithm to keep enough tree pages around. I/O devices are accessed using special files, each of which has a major device number and a minor device number. Block device I/O uses a buffer cache to reduce the number of disk accesses. An LRU algorithm is used to manage the cache. Character I/O can be done in raw or cooked mode. Line disciplines or streams are used to add features lo character I/O The file system is hierarchical with files and directories. All disks are mourned into a single directory tree starting at a unique root. Individual files can

 It) be linked into a directory frnm elsewhere in the file system. To use a file, it must be first opened, which yields a file descriptor for use in reading and writing the file. Jmernally, the file system uses three main tables: the file descriptor table, the open file description table, and the i-node table The i-node table is the most important of these, containing all the administrative information about a file and the location of itj, blocks. Protection is based on controlling read, write, and execute access for the owner, group, and others. For directories- the execute bit is interpreted lo mean search permission. 1. When the kernel catches system call,

how does, 1! know which system call it is sup- supposed to carry ouf 2. A directory contains the following files. aardvark bonefish capybara dingo emu feret grunion hyena ibex jellyfish koala llama marmot nuthatch osinch porpoise quacker rabbit seahorse tuna vicuna weasel yak zebu Which files will be Jisted by ihe command Is [abc]*e*-> 3. What does the following UNIX shell pipeline do? grep nd xyz I we -I 4. Write a UNIX pipeline ihai prints ih^ e,ghth line ()l file z on standard oi 5. Why does UNIX distinguish between standard output and standard e default to the terminal" 6. A user at a terminal types the following commands- alblcS dtelfS After Ihe shell has processed ihetn, how mjnv new processes arc runni 7. When the UNIX shell starts up a process, n puts copies <>l its enwroi ory is If this process should later fork, will the child automatically get thesu

CHAP HI PROBLEMS 759 8 About how long does it take io fork off a child process under ihe following conditions texi size = 100 KB, dam s,7e = 20 KB, stack size = 10 KB, process table size = I KB, user structure - S KB The kernel trap and return lakes I msec, and the machine can copy one 32-hu word every 50 nsec Text segments arc shaied 9. As megabyte programs became more common, the nrne spent execnttng tht fork sys- system call grew proportionally Worse yet. nearly all of this time was wasted, since mow piograms call exec shortly after forking To improve performance, Berkeley invemed a new sysl^m call, vfork. in which ihe child shares the parent's address space, instead ot getting its own copy of it. Describe a 5ituation in which a poorly-behaved child can do something that makes the semantics of vfork fundamentally different from those of fork 10. It ;t UMX process has a CPU.usage value ut 20, how many M intervals, does il lake to decay to 0 if it does not gel scheduled fora while? Use the simple decay algorithm given in ihe text. 11■ Docs it make sense to take away a process memory when it enters zombie state Why or why noP 12. To what hardware concept is a signal c-losely related" Give two examples ,)t how sig- signals are used 13. Why do you Ihmk the designers of UNIX made u impossible for a process to send a signal to a another process that is not in us process group' 14. A system call is usually implemented using a software interrupi (trap) instruction Could an ordinary procedure call be used as well on ihe Pentium hardware? II" so. under whai cnnditions and how' If not. why nof 15. [n general, (to you think daemons have higher priority <>r lower priority than interac- interactive processes? Why? 16. When a new process is forked off, il musl be assigned a unique integer as its P1D Is it

sufficient to have a counter in the kernel that is incremented on each process cre.ition. With the counter used as ihe new PID' Discuss your answer 17. in every process' eniry in the process table, ihe PlD of the process' parent is stored. Why? 18. What combination of the sharing .flags bits used by ihe Linux clone command corresponds to a conventional Unix fork call0 To creating .1 conventional Unix thread? 19. The Linux scheduler computes the goodness for real-time processes by adding the priority to 1000. Could a differeni constant have been chosen and still ha\e the algo- algorithm make the same choices as ihe regular one' 20. When booting UNIX (or mosl other operaimg systems for that matter), the booisirap loader in sector () of the disk first loads a boot program which then loads ihe Operating system Why is the extra step needed11 Surely it would be simpler to have the bootstrap loader in sector Ojusi load the nperatmg system directly.

760 CASF,STUDY I- UNIX AND LINUX CII^P. 10 21. A certain editor ha*. HH KB of program texi, JO KB of initialized dan, and 51) KB of BSS The initial stack is 10 KB. Suppose lhat three copies of this editor are started simultaneously. How much physical memory is needed (a) if shared text is used, and (b| it u is not? 22. In 4B.SD each core map entry contains, an index for the next entry on the free list, which is used when the current entry is on the free list This field is 16 bits. Pages are 1 KB Do ihese sizes have any implications for the total amount of memory BSD can support9 Explain your answer. 23. In BSD. the data and stack segments are paged anii swapped to a scratch copy kept on instead Why9 24. Describe a way to u^e mmap and Signals to construct an interprocess communication mechanism 25. A file is mapped in using the following mmap sysiem tall: mmapF5536. 32768, READ, FLAGS, frj, 0| Pages are 8 KB Which byte in ihe file is accessed by reading a byte at memory address 72,000" 26. After the system call tit the previous problem has been executed the call munmapF5536, 8192) is carried out. Does it succeed9 If so. which bytes of lhc file remain mapped9 If not. why does it fail" 27. Can a page fault ever lead io the faulting process being terminated? It so. give an example if not. why nof 28. Is it possible that with the buddy sysiem of memory management it ever occurs that two adjacent blocks of free memory of the same size co-exist without being merged impossible. 29. It is stated in ihe text that a paging partition will perform belter than a paging filt Why is this so? 3u. Give two examples of the advantages of relative path names over absolute ones 31. The following kicking calls are made by a

collection of processes For e4ch call, tell what happens, if a process fails to get a lock, it blocks (a) A wants a shared lock on bytes 0 through]() (b) B wants an exclusive lock on bytes 20 through 30 (O C wants a shared lock on bytes 8 through 40 (d) A wants a shared lock on hytes 21 through 35. (e) B wants an exclusive lock on byte 8 32. Consider the locked file of Fig, !0-27{c). Suppose that a process tries to lock bytes 10 and 11 and blocks- Then, before C releases its lock, yet another process tries to lock

CHAP, 10 PROBLEMS 761 bytes 10 and I I, <ffld also blocks What kind ot problems aiv introduced inn, the semantics by this, situation Propose and defend two solutions 33. Suppose that an Iseek system call seeks to a negative offsei m a file Given two possi- hle ways ot dealing with it 34. If a UNlx file has protection mode 751) (octal), what can the owner, the owner's group, and everyone else do to the file1 35. Some tape drives have numbered blocks and the ability to overwrite a particular block in place without disturbing the blocks m front or behind « Could such a device hold a mounted UNIX tile system11 36. [n Tig 10-25, both f-red and Lisa have access to the file x in [heir respective direc- directories afier linking, (s this access completely symmetrical in the sense anything one ot them can do with if the other one can too? 37. As we have seen, absolute path names are looked up siartuig at the root directory and cient way to implemenf both kinds of searches 38. When the file /usr/ast/work/f ,s opened, several disk accesses are needed to read i- node and directory blocks Calculate the number of disk accesses required under the assumption that the i-node tor ihe root directory is always in memory, .ind all direc- directories are one block long. 39. A UNIX t-node has 10 disk addresses lor data blocks, as well as the addresses of sin- single, double, and triple indirect blocks [f each of these holds 256 disk addresses, what is the size of the largest file that can be handled, assuming that a disk block is I KB" 4A. When an mode is read in from the disk during the process <>f opening a file, it is put into an i-node table in memory This table has some helds thai are not present on ihc disk. One of (hem is a counter that keeps track ol the number ol times the i-node has been opened Why is this field needed'1 41. Why does LRU work for managing the butter cache, whereas it rarelj works Iw keep-keeping track of pages in a virtual memory system" 42. UNIX has a system call, sync, that flushes the buffer cache back to disk At boot time, a pnigiam called update is started Once every 10 sec it calls sync, then goes to sleep for 3() sec. Why does this program exist7 43. After a system crash and reboot, a

recovery program is usually run. Suppose that this program discovers thai the link Count in a disk i-noctc is 2, but only one directory entiy references the i-node. Can it fix the problem and it so. how1 44. Make an educated guess as to which UNIX system call is the fastest 45. Is it possible to unlink a file that has nevc-r been linked'1 What happens^ 46. Based on the informaiion presented m this chapter, if a Linux ext2 file system were to be put on a 1,44 Mbyte floppy disk whal is the maximum amount of user file data that could be stored on the disk'* Assume that disk blocks are I KB

762 CASE STUDY I UNIX AND LINUX CHAP 10 se if they get to be supertiser, why di>es 47. in view of all ihe trouble thai students this concept exist m the first place'1 48. A professor shares tiles with his students bv placing them in a publically accessible directory on the CS department's UNIX system. One day he realizes that a file placed there the previous day was left world-writable. He changes the permission^, and veri- verifies Ihe file is identical to his master copy The next day he finds the file has been changed How could this have happened and how could il have been prevenicd1' 49. Write a minimal shell that allows simple commands to be started It should also allow [hem to he parted in Ihe background 50. Using assembly language and BIOS calls, write a program that boots itselt from a [loppy disk on a Pentium-class computer The program should use BIOS calls to ix-ad the keyboard and echo the characters lyped, just to demonstrate that it is running. 5|. Writ serial ports Use the POSlX terr nanagem onfigure the ports.

11 CASE STUDY 2: WINDOWS 2000 Windows 2000 is a modem operating system that runs on high-end desktop PCs and servers. In this chapter we will examine various aspects of it, starling with a bnef history, then moving on to its architecture. After this we w,lt took at processes, memory management, I/O, [he file system, and finally, security. One thmg we will not look at is networking as that could easily fill a chapter (or an entire book) all by itself. 11.1 HISTORY OF WINDOWS 2000 Microsoft operating systems for desktop and laptop PCs can be divided into three families: MS-DOS, Consumer Windows (Windows 95/98/Me), and Windows NT. Below we will briefly sketch each of these families. 11.1.1 MS-DOS In |98|, IBM, at the time the biggest and most powerful computer company hi the world, produced the 8088-based IBM PC. The PC uime equipped w,th a [6-bjt real-mode, single-user, command-line oriented operating system called MS-DOS 10. The

operating system was provided by Microsoft, a tiny startup, mostly known at that time for its BASIC interpreter used on 8080 and Z-80 sys- systems. This operating system consisted of 8 KB of memory resident code and was 7 3

764 CASE STUDY 2: WINDOWS 2000 CHAP 11 closely modeled on CP/M. a tiny operating system for the 8-bit 8080 and Z80 CPUs. Two years later, a much more powerful 24-KB operating system, MS-DOS 2.0, was released. It contained a command line processor (Shell), with a numbei ot features borrowed trom UNIX. When Intel eame out with the 286 chip, IBM built a new computer around it. ihe PC/AT, released in 1986 AT stood for Advanced Technology, because the 286 ran at a then-impressive 8 MHz and could address—with great difficulty—all of 16 MB of RAM. In practice, most systems had at mcist 1 MB or 2 MB. due to the great expense of so much memory. The PC/AT came equipped with Microsoft's MS-DOS 3 0, by now 36 KB Over the years. MS-DOS continued to acquire nev, features, but it was, still a command-line oriented system 11.1.2 Windows 95/98/Me Inspired by the user interface of the Apple l,,sa, the forerunner to the Apple Macintosh, Microsoft decided to give MS-DOS a graphical user interface (shell) lhat u called Windows. Windows 1.0, released in [985. was something ot a dud. Windows 2.0, designed for the PC-AT and released in 1987, was not much better. Finally. Windows 3.0 for the 386 (released m 1990), and especially its successors 3 1 and 3,l|, caught on and were huge commercial successes. None of these early versions of Windows were true operating systems, but more like graphical user interfaces on top of MS-DOS, which was still jn control of the machine and the file system. All programs ran in the same address space and a bug in any one of them could bring the whole system to a grinding halt. The release of Windows 95 in August 1995 still did not completely eliminate MS-DOS, although it transferred nearly all the features from the MS-DOS part to the Windows part. Together, Windows 95 and the new MS-DOS 7 0 contained most of the features of a fuli-blown operating system, including virtual memory, process management, anil multiprogramming. However, Windows 95 was not a full 32-bit program. It contained large chunks of old 16-bit assembly code (as well as some 32-bit code) and still used the MS-DOS file system, with nearly all its limitations. The only major change to the file system was the addition of long Hie names in place of the 8 + .3 character file names allowed m MS-DOS. Even with ihe release of Windows 98 in June |99X. MS-DOS was still there (now called version 7. |) and running 16-bit code. Although yet more

functional- functionality migrated from the MS-DOS part (o the Windows part, and a disk layout suitable for larger disks was now standard, under the hood, Windows 98 was not very dif- different from Windows 95. The main difference was the user interface, which integrated the desktop and the Internet more closely It was precisely this integra- integration (hat attracted the attention of the U.S Dept of Justice, which then sued Microsoft claiming that it was an illegal monopoly, an accusation Microsoft vig- vigorously denied. In April 2000. a U.S. Federal court agreed with the government

ShC 111 HISTORY OF WINDOWS 21X10 765 In addition to containing a large lump of old 16-bit assembly code m the ker- kernel, Windows 98 bad two other serious problems. First, although it was a mul- multiprogramming system, the kernel itself was no(reentrant. If a process was busy manipulating some kernel data structure and then suddenly its quantum ran out and another process started running, the new process might find the data structure in an inconsistent state To prevent this type of problem, after entering the kernel, most processes first acquired a giant mutex covering the whole system before doing anything. While this approach eliminated potential inconsistencies, it also eliminated much of the value of multiprogramming since processes were fre- frequently forced to wait for unrelated processes to leave the kernel before they could enter it. Second, each Windows 9X pion s had a 4-GB virtual address space- Of this, 2 GB was completely private to the process. However, the next I GB was shared (writably) among all other processes in the system. The bottom I MB %vas also shared among all processes to allow all of them to access the MS-DOS interrupt vectors This sharing facility was heavily used by most Windows 98 applications. As a consequence, a bug in one program could wipe out key data structures used by unrelated processes, leading to them all crashing. Worse yet, the last | GB was shared (writably) with the kernel and contained some critical kernel data structures. Any rogue program that overwrote these data structures with garbage could bring down the system. The obvious solution of not putting kernel data structures in user space was not possible because this feature was essential to making old MS-DOS programs work under Windows 98. In the millennium year, 2000. Microsoft brought out a minor revision to Win- Windows 98 called Windows Me (Windows Millennium Edition)- Although it fixed a few bugs and added a few features, under the covers it is essentially Win- Windows 98. The new features included better ways to catalog and share images, music, and movies, more support for

home networking and multiuser games, and more Internet-related features, such as support for instant messaging and broad- broadband connections (cable modems and ADSL). One mteresting new feature w.is the ability to restore the computer to its previous settings after a reconfiguration. If a user reconfigures the system (e.g.. changing the screen from 640x480 to 1024 X 768) and it no longer works, this feature makes it possible to revert to the last known working configuration. 11.13 Windows NT By the late 1980s, Microsoft realized that budding a modern 32-bit dperating system on top of the leaky 16-bit MS-DOS probably was not the best way to go. It recruited David Cutler, one of the key designers of DEC's VMS operating system, to work for Microsoft and gave him the job of leading a team to produce a brand- new 32-bit Windows compatible operating system from the ground up. This new

766 CASE STUDY 2- WINDOWS 2000 CHAP 11 system, laler called Windows NT (Windows New Technology), was intended for mission-critical business applications as well as tor home users. Al the time, mainframes still ruled the (business) world, so designing an operating system on the assumption that companies, would use persona! computers tor anything impor- important was a visionary goal, but one (hat history has shown to be a very good one. Features such as security and high reliability, clearly iacking on the MS-DOS-based versions of Windows, were high on the agenda tor (Windows) NT. Cutler's background with VMS clearly shows in various places, with (here being more than a passing similarity between tde design of NT and that of VMS The project succeeded and the first version, called Windows NT 3.1, was released in 1993. This initial release number was chosen to match the number of Microsoft's then-popular 16-bit Windows 3-1 system Microsoft expected ihat NT would rapidly replace Windows 3.1 because it was technically a far superior system. Much to its surprise, nearly a|l users preferred to stick with the old |6-bit sys- lem they knew, raiher man upgrade to an unknown 32-bit system they did not know, however better it rnighl be Furthermore. NT required far more memory than Windows 3-1 and there were no 32-bit programs tor it to run. so why bother" The failure of NT 3.1 to catch on in the marketplace was the reason Microsoft decided to build a 32-bit-ish version ofWmdows 3.1. namely Windows 95. The continued user resistance to NT then caused Microsoft lo produce Windows 98 and finally Windows Me; each one claimed to be the very last release of the MS- DOS-based systems. Despite the fact that nearly all consumers and most businesses ignored NT 1 I

tor desktop systems, K did acquire a small following in the server market. A few new 3.x releases with small changes occurred in 1994 and |995 These slowly began to acquire more following among desktop users ds well. The first major upgrade to NT came with NT 4.0 in 1996. This system had ihe power, security, and reliability of the new operaiing system, but also sported the same user interface as the by-then very popular Windows 95. This compati- compatibility made it much easier for users to migrate from Windows 95 (o NT, and many of them did so. Some of the differences between Windows 95/98 and Windows NT are summarized in Fig. | |-l. Prom the beginning. NT was designed lo be portable, sci it was written almost entirely in C, with only a tiny bit ot assembly code for low-level functions .such as interrupt handling. The initial release consisted of 1 I million lines of C for the operating system, libraries, and the environment subsystems (discussed below). When NT 4.0 came out, the uode base had grown to 16 million lines ot code, still mostly C. but with a small amount of C++ m the user interface part. By this time the system was highly portable, with versions running on (he Pentium. Alpha, MIPS, and PowerPC, among other CPUs. Some of these have been dropped Since then. The story of how NT was developed is given in the book Showitopper (Zachary. 1994). The book also tells a lot about the key people involved.

HISTORY OF WINDOWS 20<H 11.1.4 Windows 2000 The release of NT following NT 4.0 was originally going to be called NT 5 0. However, in |999, Microsoft changed the name to Windows 200A. mostly in an attempt to have a neutral name that both Windows 98 users and NT useis could see as a logical nexl Step for them. To the extent that this approach succeeds. Microsoft will have a single main operating system built on reliable 32-bit tech- technology but using the popular Windows 98 user interface. Since Windows 2000 really is NT 5.0. it inherits many properties trom NT 4.0, It is a tme 32-bit (soon to be 64-bit| multiprogramming system with individu- individually protected processes Each process has a private 32-bit (soon 64-brt) demand- paged virtual address space. The operating system runs in kernel mode, whereas user processes run in user mode, providing complete protection (witli none of the protection (laws of Windows 98) Processes can have one or m<ire threads, which are visible to, and scheduled by, the operating system. It has Depi. of Defense C2 security for all files, directories, processes, and other shareable objects (at least, ii the floppy disk is removed and the network is unplugged).

Finally, it also has full support for running on symmetric multiprocessors with up to 32 CPUs. The fact that Windows 2000 really is NT 5.0 ,s visible in many places. For example, the system directory is called \winni and the operating system binary (in \winnf^ystem32) is called ntoskml.exe. Right clicking on this file to examine its

768 CASE STUDY 2. WINDOWS 2000 CHAP |1 properties show-, (hat iis vision number K 5.xkjc yyy.zz:, where the -> stands to NT 5. \\x is the release number, yyy is the build (compilation) number, and zzz the mitioi variant. Also, many ot the files in \winn, and its subdirectories have nt in their names, such as ntvdm, NT's virtual MS-DOS emulator. Windows 2000 is more than just a better NT 4 0 with the Windows 98 user interface. To start with, tt contains a number of other features previously found only in Windows 98. These include complete support for p!ug-and-p|ay devices, the USB bus. IEEE 1394 (FireWire), IrDA (the infrared link between portable computers and printers), and power management, among others. In addition, a number of new features not present m any other Microsoft Operating system have been added, including active directory directory service, security using Ktsrberos, support for smart cards, system monitoring tools, better integration of laptop com- computers with desktop computers, a system management infrastructure, and job objects. Also, the main file system, NTFS, has been extended to support encrypted files, quouts, linked files, mounted volumes, and content indexing, for example. Another novel NTFS feature is the single instance store, which i*, a kind of copy-on-write link m which two users can share a linked fjle until one of them writes on it. at which nine a copy is made automatically One oilier major improvement is internationalization. NT 4.0 came in separate versions, for different languagc-S with the text strings embedded in [he code Installing an English software package on a Dutch computer often caused parts ot the operating system to stop using Dutch and start using English because certain files containing code and text strings were overwritten. This problem has been eliminated. Windows 2000 has a single binary that runs' everywhere- in the world. An installation, or even an individual user, can choose the language to use at run time because all the menu items, dialog strings, error leports, and other text strings have been removed from the operating system and put m separate direc- directories, one per installed language. Like all previous versions of NT. Windows 2000 uses Unicode throughout (he sysiem to support languages not using the Latin alphabet, such as Russian, Greek, Hebrew, and

Japanese. One thing that Windows 2000 does not have is MS-DOS It is simply not there in any form (nor was ii ihere in NT). There is a command line interface, but this is a new 32-bit program thai includes the old MS-DOS functionality and consider- considerably new functionality as well Despite many portability features with regard io the code, hardware, language, etc., in one respeel Windows 2000 is less portable than NT 4 0. it runs on only two platforms, the Pentium and the Intel IA-64. Originally Nt supported additional platforms, including the PowerPC, MIPS, and Alpha, but over the years. Microsoft dropped one after another for commercial reason*. Like previous vers,ons of NT. Windows 2000 comes in several product levels, this time: Professional, Server, Advanced server, and Datacenter server T|ie differences between all these versions are minor however, with the same execui- able binary used for all versions. When the system is installed, the product type is

SEC II I HISTORY OF WINDOWS 2000 ' recorded m an internal database |the registry). At boot time, the operating syst checks the registry to see which version ,t K. The differences are shown Fig 11-2 Max RAM j CPUsTMax clients ^Ciuster^ize 4 GB 1 2 I 10 4 GB ; 4 Unlimited I 8 GB 1 8 Unlimited ~_ 64 GB i 32 ! Unlimited [Figure 1-2. Thcdiffen mdows 2000 As tan be seen from ihe figure, the differences include the maximum memory supported, the maximum number of CPUs (for a multiprocessor configuration), and the maximum number of clients that can be served. The cluster size relates to the ability of Windows 2000 to make two or four machines look like a single server to the outside world, a useful feature foi Web servers, for example Finally, the default parameters are luned differently on Professional, to favor interactive programs over batch work, although these can easily he changed if desired One last difference is that some extra software is provided on the servers and some extra tools are provided on Datacenter server foi managing large jobs The reason for having multiple versions ,s simply marketing- this allows Microsoft to charge big companies more than they charge individuals foi what is essentially the same product. This idea it, not new. howevci, and hardly unique- to Microsoft. For years, airlines having been charging business passengers much more, not only for Business Class, but also for Cattle Class if they want the luxury of buying the ticket a day before the flight instead of a month before ihc- flight Technically, the way the version differences are maintained is that m a ftw places in the code, two variables- are read from the registry, Product I ype and Pro- ductSuite. Depending on their values, slightly

different code is executed. Chang- Changing these variables is m violation of the license In addition, the system traps any attempt to change them and records the attempt at tampering in an indelible way so it can be detected later. In addition to the basic operating system. Microsoft has also developed several tool kits for advanced users. These include the Support TXioU, the Software Development Kit, the Driver Development Kit. and the Resource Kit. These include a large number of utilities and tools for tweaking and monitoiing the system. The support tools are on the Windows 2000 CD-ROM in the diieo tOTy\supporf\tool.s. The standard installation procedure does not install them but there is a file ietup.exe in that directory that does. The SDK and DDK are avail- available to developers at msdn.mkrosoft.com. The Resource Kit u, a Microsoft pro- product in a box. There are also various third-party tools available tor snooping on the Windows 2000 internals, including a nice set ava,lable for free at the Web site

770 CASE STUDY 2 WINDOWS 2000 CHAP. II i\ Hn.syxmternats.com. Some of [hese even provide more information than ine corresponding Microsoft toois. Windows 2000 is an immensely complex system, now consisting of over 29 million lines of C code. If printed 50 lines per page and 1000 pages per hound book, ihe full code would occupy 580 volumes. This opus would occupy 23 run- running meters or shelf space (for the paperback version). If ananged m bookcases I m wide with 6 shelves per bookcase, the set would occupy a wall 4 m wide. Just for fun, a comparison ot a few operating system souTte code sizes is given in Fig 11 -3. However, this table should be taken wiih a grain (or better yet, a metric ton) of salt because what constitutes the operating system is different for different systems. For example, the entire window system and GUI is pan of the kernel in Windows, but not in any UNIX version. It is simply a user process there. Counting X Windows adds another 1.5 million lines of code to a]! the UNIX ver- versions, and that does not even count the GUI code (Molif, GNOME, etc.), which is also not part of ihe operating system in the UNIX world. Additionally, some sys- systems include code for multiple architectures (e g., five tor 4 4 BSD and nine tor Linux), wiih each architecture adding 10.000 to 50,000 lines of code. The reason Free BSD 1.0 has only 235,000 lines of code whereas 4BSD Lite, from which ii is derived, has 743,000 lines is that support for all the obsolete architectures (e.g., the VAX) was dropped ,n Free BSD Also, the number of file systems, devices drivers, and libraries supplied varies greatly from system to system. In

addition, Windows contains large amounts of test code that UNIX does not contain as well as some utilities and support for numerous languages besides English. Finally, ihe measurements were made by different people, which introduces considerable variance (e.g , did makefiles, headers, configuration files and documentation count and how much was there"). This is not like comparing apples with oranges; it is like comparing apples with telephones. However, all the counts within a single family came from the same source, so intrafamily counts are somewhat meaningful. Despite all these disclaimers, two conclusions are fairly clear 1. System bloat seems to be as inevitable as death and taxes. 2. Windows is much bigger than UNIX Whether small is beautiful or big is beautiful is a matter of heated controversy The argument for ihe former is that small size and a lean-and-me an mentality pro- produces a manageable, reliable system that users can understand. The argument for ihe latter is lhat many users want lots of features. In any event, it should also be clear that any students planning to write a full-blown, state-of-the-art operating system from scratch have their work cut out for them. Although Windows 2000 is already ihe world heavyweight champion in terms of pure mass, it is still growing, with bugs being fixed and new features being added. The way Microsoft manages its developmeni is worth noting. Hundreds

HISTORY OF WINDOWS 2000 each box is llic \crsion, ihe si^nind la]hc where K = 1000 and M = 1.000,000 Comparisons within a toliiinn h,,\c real meaning, companions across colunius <Kj mil, j^ diseased in shL- texl of programmers work on various aspecis of Windows 2000 all day. Whenever a piece of code is finished, the programmer submits it electronically to the build team. At 6 P.M. every day, thi: door is closed and the system ,s rebuilt (i e., recompiled and linked). Each build gets a unique sequence number, which can he seen by examining the version number of ninvkml exe (the fir* public release of Windows 2000 was build 2|95). The new operating system is electronically distributed to thousands ot machines around the Microsoft campus in Redmond, WA, where it is subjected to intense stress tests all nighi. Early the next morning, the results ot all the tests jre sent to the relevant groups. so they can see if their new code works Each team then decides which code they want io work on that day. During the day. the pio- grammers work on their code and at 6 P M the build-and-tesi-cycle begins anew 11.2 PROGRAMMING WINDOWS 2000 It ii now lime to start our technical study ol Windows 2000. Howevei. before getting into the details of the internal

structure, we will first take a look at the pro- programming interlace and the registry, a small m-memory data base.

772 CASE STUDY 2: WINDOWS 2000 11.2.1 The Win32 Application Programming Interface Like all other operating systems. Windows 2000 has a set of .sysiem calls it can perform However, Microsoft has never made the hst of Windows system calls pubhe. and it also changes them from release fo release. Instead, what Microsoft has done is define a set of function calls called the Win32 API (Win32 Application Programming Interface) that are publicly known and fully docu- documented These are library procedures that either make system calls to get ihe work done, or, in some cases, do the work right ,n user -space. The existing Win32 API calk do not change with new releases <,f Windows, although new API calls arc added frequently. Binary programs for the Intel x86 that jdhere exactly to the Win32 API inter- interface will run unmodified on all versions of Windows since Windows 95. As shown in Fig. 11-4, an extra library is needed for Windows -3.x to maich a subset of the 32-bit API calls to ihe 16-bit operating system, but for the other systems no adaptation is needed. It should be noted that Windows 2000 adds substantial new functionality to Win32, so it has additional API calls not included on older ver- versions of Win32 and which will not work on older versions of Windows,. The Win32 API philosophy is completely different from the UNIX philoso- philosophy. In the latter, the system calls are all publicly known and form a minimal interface: removing even one of them would reduce the functionality of the operating .system The Win2 philosophy is to provide a very comprehendie interface, often with three or four ways of doing the same thing, and including many functions (i.e., procedures) that clearly should not be (and are not) system calls, such a:, an API ca|J to copy an entire file Many Win32 API calls create kernel objects of one kind or another, including files, processes, threads, pipes, and so on. Every call creating an object returns a

SEC 112 PROGRAMMING WINDOWS 2000 773 result called a handle to the caller. This handle can subsequently be used to per- perform operations on the object. Handles are specific to the process that created the object referred to by the handle. They cannot be passed directly to another proc- process and used there (just as UNIX file descriptors cannot be passed to other processes and used there). However, under certain circumstances, it is possible to duplicate a handle and pass it to other processes m a protected way, allowing

them controlled access to objects belonging to other processes. Every object also has a security descriptor associated w,th it. telling in detail who may and may not per- perform what kinds of operations on the object. Not all system-created data structures are objects and not all objects are ker- kernel ob|ects. The only ones that are true kernel objects are those that need to be named, protected, or shared in some way Every kernel object has a systcm- defined type, has well-defined operations on it, and occupies storage in kernel memory. Although users can perform the operations (by making Win32 calls), they cannot get at the data directly. The operating system itself can also create and use objects and does so heavily. Most of these objects are created to allow one component of the system to store some information for a substantial period of time or to pass some data structure to another component. For example, when a device driver is loaded, an object is created for it holding |U properties and pointers to the functions it con- contains. Within the operating system, the driver is then referred to by using its object. Windows 2000 is sometimes said to be object-oriented because the only way to manipulate objects \s by invoking operations on their handles by making Win32 API calls. On the other hand, it lacks some of the most basic properties of object-orienied systems such as inheritance and polymorphism. The Wjn32 API calls cover every conceivable area an operating system could deal with, and quite a lew it arguably should not deal wiih. Naturally, there are calls for creating and managing processes and threads. There are also many calls that relate io interprocess (actually, inrerthread) communication, such a« creating, destroying, and using mutexes, semaphores, events, and other IPC objects Although much of the memory management system is invisible to program-programmers (fundamentally, it is just demand paging), one important feature is visible: namely the ability of a process to map a file onto a region of its virtual memory This allows the process the ability to read and write parts of the file as though they were memory words. An importani area tor many programs is file I/O In the Win32 view, a file is just a linear sequence of bytes. Win32 provides over 60 call^ for creating and destroying files and directories, opening and closing files, reading and wniing them, requesting and setting file attributes, and much more. Another area for which Win32 provides calls is security. Every process has an ID telling who it is and every object can have an access control list telling in great detail precisely which users may access it and which operations ihey may

774 CASb STUDY 2 WINDOWS 2000 CHAP II perform on it. This, approach provides lor a fine-grained security in which specific individuals can be allowed «r denied specific access to every object. Processes, threads, synchronization, memorv management, file I/O, and secu- security system calls are nothing new. Other operating systems have them too, although generally not hundreds ot them, as Win32 does. But what really distin- distinguishes Win32 are the thousands, upon thousands of calk for the graphical inter- interface There are calls for creating, destroying, managing and using windows, menus, tool bars, status bars, scroll bars, dialog boxes, icons, and many more items that appear on the screen. There are calls for drawing geometric figures, fil- filling them in, managing the color palettes they use. dealing with fonts, and placing icons on the .screen. Finally, there are calls fo, dealing with the keyboard, mouse and other input devices as well as audio, printing, and other output devices. In short, [he Win32 API ^especially the GUI part> is immense and we could not even begin to describe- it in any detail m this chapter, so we will not try. Interested readers should consult one of the many books on Win32 (e.g., Petzold. 1999; Simon, |9y7; and Rector and Newcomer. 1997,. Although the Win32 API is available <>n Windows 98 (a.s well as on the con- consumer electronics operating system, Windows CE), not every version of Windows implements every call and sometimes there are minor differences as well. For eiample, Windows 98 does noi have any security, so those API calls that relate to security just return error codes on Windows 98. Also. Windows 2000 file names use the Unicode character set. which is not available on Windows 98 and Win- Windows 98 file names are noi case sensitive, whereas Windows 2000 file names are case sensitive (although some kinds of searches on file names are not case sensi- sensitive). There are also differences in parameters to some API function calls. On Windows 2000, for example, all the screen coordinates given in the graphics func- functions are true 32-bit numbers; on Windows 98, only the low-order 16 bits arc used because much of the graphics subsystem is still 16-bit code. The existence of the Win32 API on several different Operating systems makes it easier to port pro- programs between them, but since these minor variations exist, some care must be taken to achieve portability. 11.2.2 The Registry Windows needs to keep track of a great deal of information about hardware, software, and users. In Window 3.x. this information was stored in hundreds ot ,im (initialization) files spread all over the disk. Starting with Windows 95. nearly al! the information needed for booting and configuring the system and tailoring it to the current user was

gathered in a big central database called the registry. In this section we will give an overview of the Wjndows 2000 registry. To start with, it is worth noting that although many parts of Windows 2000 are complicated and messy, the registry is one of the worst, and the cryptic nomenclature does not make it any better. Fortunately, entire books have been

SEC 11 2 PROGRAMMING WINDOWS 2000 775 written describing it (Bom. [998; Hipsuti, 2000; and Ivens. 1998). That .aid, the idea behind the registry is very simple It consists, of a collection of directories, each of which contains either subdireciorie.s or entries. In this respect it is a kind of file system for very small file1; It ha*, directories and entries (the files) The confusion starts with the fact that Microsoft calls a directory a key. which it definitely is not. Furthermore, all the top-level directories start with the string HKF.Y. which means handle to key. Subdirectories tend lo have somewhat better chosen names, although not always. Al the bottom of the hierarchy are the entries, called values, which contain the information. Each value has three parts* a name, a type, and the data. The name is just a Unicode string, often default if the directory contains only one value. The type is one of 1 I standard types The most common ones are Unicode string, a list ot Unicode strings, a 32-hit integer, an arbitrary length binary number, and a symbolic link to a directory or entry elsewhere in the registry. Symbolic names are completely analogous to symbolic links in file systems or shortcut!, on the Windows desktop; they allow one entry to point to another entry or directory. A symbolic link can also be used as a key. meaning that someihing that appears to be a directory is just a pointer to a different directory. At the top level, the Windows 2000 registry has six keys, called root keys, as listed in Fig. 11-5. Some interesting subkeys (subdirectories) are also shown here. To see this lis't on your system, use one of the registry editors, either regedtt or regedl32, which unfortunately display differeni information and Uf,c different formats. They can also change registry values. Amateurs should not change keys or values on any system they plan to boot again. Jus,t looking is safe, though. You have been warned. The first key (i.e. directory), HKEY_LOCAL_MACHINE, is probably the most important as it contains all the information about the local system. It has five ?,uh- keys (i.e., subdirectories). The HARDWARE subkL-y contains many subkeys telling all about the hardware and which driver controls, which piece of hardware. This information js constructed on the fly by the plug-and-p|ay manager as the system boots. Unlike the other subkeys, it

is not stored on disk. The SAM (Security Account Manager) subkey contains the user names, groups, passwords, and other account and security information needed for logging in. The SECURITY subkey contains general security policy information, such as minimum length for passwords, how many failed login attempts are tolerated before the police are called, etc. The SOFTWARE subkey is where software manufacturers store preferences etc. For example, if a user has Adobe Acrobat, Photoshop and Premiere installed, there will be a subkey Adobe here, and below that further subkeys for Acrobat, Photoshop, Premiere, and any other Adobe products. The entries in these sub-subdirectories can store anything the Adobe programmers want to put there, generally system-wide properties such as the version and build number, how to uninstall the package, drivers to use. and so forth. The registry saves them the trouble of hav-

CASE STLDY 2- WINDOWS 200A I Key __1ZT Description HKE~Y~ LOCAL .MACHINE Properties of the hardware and software I HARDWARE ' Hardware description and mapping of hardware to dnvers I SAM | Security and account information tor users SECURITY i System-wide security policies SOFTWARE Generic information about installed application programs ! SYSTEM I Information for booting the system ____ Thkey.USERS Information about the users, one subkey per user USER-AST-ID I User AST's profile AppEvents Which sound to make when (incoming email/fax, error, etc) Consols Command prompt settings (colors, fonts, history, etc) Control Panel | Desktop appearance, screensaver. mouse sensitivity, etc Environment Environment variables I Keyboard Layout Which keyboard 102-key US, AZERTY, Dvorak, etc Printers Information about installed printers Software I User preferences for Microsoft and third party software I HKEY_PERFORMANCE_DATA j Hundreds of counters monitoring system performance |~HKEY CLASSES, ROOT : LinktoHKE~Y"_LOCAL_MACHINE\SOFTWARE\CLASSES j HKEY_CURRENT_CONFIG Link to the current hardware profile __ __ __ |~HKEY_CURRENT_USER ~ Link to the current user profile __ Figure 11-5. The root ke>s registry keys and selected subkeys The capiiddzj- tion ha> no meaning but toJJows the Microsoft practice heit ing to invent their own method for storing this information User-specific infor- information also goes in trie registry, but under HKEy.USERS. The SYSTEM subkey holds mostly information about booting the system, for example, the list of drivers that must

be loaded It also holds the list ot seivii.es (daemons) that must be started after booting up and the configuration information for all of them. The next top-level key is HKEY.USERS. which contains the profiles of all the users. All the user-specific preferences in a number of areas are stored here When a user changes a preference using the control panel, for example, the desk- desktop color .scheme, the new settings are recorded here. In fact, many of the pio grams on the control panel do little more than collect user information and change the registry accordingly. Some of the subkeys under HKEY USERS are shown in Fig. 11-5 and should need little additional comment. Some of the subkeys, such as Software, contain surprisingly large numbers of subkeys. even if no software packages are installed. The next top-level key. HKEY .PERFORMANCE _DATA contains neither data read tn from the disk nor data collected by the plug-and-play manager Instead, it offers a window into the operating system. The system itself contains hundreds of counters for monitoring system performance. These counters are accessible via

 this registry key. When a subkey is queried, a specified procedure is run to collect and return the information (possibly by reading one or more counters and combin- inn them in wmp wavl This kev is not visible using regedil or regedttt Instead pfrnon, perfmon, and p ing them in some way) Th , . one has to use the performance tools, sui are many such tools, some on the Wmdo' kits, and some from third partie s 2000 CD-ROM, some in the n ist. Each one i The nexi three top-level keys do not actually ,—„.. ., link to some place elsewhere in the registry. The HKEY CLASSES ROOT key „ the most interesting It points to the directory that handles COM (Component Object Model) objects and also the associations between file extensions and pro- programs. When a user double clicks on a file ending in. say. .doc. the program m/f Word) The complete database of recognized extensions and which program each one is owned by is under this key The HKEY_CURRENT_ CONFIG key links to the current hardware configura- configuration. A user can construct multiple hardware configurations, for example by disa- disabling various devices to see if they were the cause of strange system behavior This key points to the current configuration. Similarly, HKEY_CURRENT USER points to the current user so that user's preferences can be found quickly. None of the last three keys really adds anything, since the information was available elsewhere anyway (although less conveniently accessible) Thus despite the fact that regedit and

regedlS2 list five top-level keys, there are really only three top-level directories and one of those is not shown among the five displayed The registry is fully available to the Win32 programmer. There are calls to create and delete keys, look up values within keys, and more Some of the more useful ones are listed in Fig. 11-6. the key of the handle j Figure 11-6- Some ol the Win32 AP] t or using the re When the system is turned off. most of the registry information (but not all. as discussed above) is stored on the disk in files called hives. Most of them are in \wmnAsystem32\conftg Because their integrity is so critical to correct system functioning, when they are updated, backups are made automatically and writes are done using atomic transactions to prevent corruption in the event of system crash during the write. Loss or the registry requires reinsialling all software.

778 CASE STUDY 2- WINDOWS 2000 CHAP 11 11.3 SYSTEM STRUCTURE In (he previous sections we examined Windows 2000 as seen by the program- programmer Now we are going to look under [he hood to see how the system is organ- organized internally, what the various components do. and how they interact with each other and with u^er programs. Although there are many books on how to use Windows 2000. there are many fewer on how it works. Far and away the best place to look for additional information on this topic is Inside Windows 2000, 3rd ed. by Solomon and Russinovich B000). Some of the material in this chapter is based on information m that book and information from the authors. Microsoft was also a key source 11.3.1 Operating System Structure Windows 2000 consists of two major pieces' the operating system itself, which runs in kernel mode, and the environment subsystems, which run in user mode. The kernel is a traditional kernel in the sense that it handles process management, memory management, file systems, and so on. The environment subsystems are somewhat unusual in that they are separate processes that help user programs carry out certain system functions. In the following sections we will examine each of these parts in turu. One of NT's many improvements over Windows 3.x was its modular struc- structure. It consisted of a moderately small kernel that ran in kernel mode, plus some server processes that ran in user mode. User processes tnteracted with (he server processes using the client-server model: a client sent a request message to a server, and the server did the work and returned the result to the client via a second message. This modular structure made it easier to port it to several com- computers besides the Intel line, including the DEC Alpha, IBM

PowerPC, and SOI MIPS. It also protected the kernel from bugs in the server code. However, for performance reasons, starting with NT 4.0. prelly much a|l of the operating sys- system (e.g., system call handling and all of the screen graphics) was put back into kernel mode. This design was carried over to Windows 2000. Nevertheless, there is still some structure in Windows 2000. It is divided into several layers, each one using the services of the ones heneath it. The structure is illustrated in Fig. |l-7. One of the layers is divided horizontally into many modules. Each module has some particular function and a well-defined interface to the other modules. The lowest two software layers, the HAL and the kernel, arc written in C and in assembly language and are partly machine dependent The upper ones are writ- written eniirely in C and are almost entirely machine independent. The drivers are written in C, or in a few cases C++. Below we will first examine (he various components of the system starting at the bottom and working our way up.

SEC. 11.3 SYSTEM STRUCTURE 779 OS/2 program Service process I POSIX program |] Win32 propram | | POSJX subsystem \-*\ Win32 aubsyslem \~-\ OS/2 subsystem ± System Interface (NT DLL DLL) System mtvIcm I/Omgr Cfcjeol Cache Pop Fmmt Ccrttg LPC mgr Win32 GDI Kernel Hartwsre Abstraction layer (HAL) Hardware Figure 11-7. The structure of Windows 2000 (slightly simplified). The shaded area is the executive. The boxes indicated by D are device drivers. The service processes are system daemons. The Hardware Abstraction Layer One of (he goals of Windows 2000 (and Windows NT before it) was to make the operating system portable across platforms. Ideally, when a new machine comes along, it should be possible to just recompile the operating system with the new machine's compiler and have it run the first time. Unfortunately, life is not like that. While the upper layers of the operating system can be made completely portable (because they mostly deal with internal data structures), the lower layers deal with device registers, interrupts, DMA, and other hardware features thai differ appreciably from machine to machine. Even though most of the low-level code is written in C, it cannot just be scooped up from a Pentium, plopped down on, say, an Alpha, recompiled, and rebooted due to the many small hardware differences between the Pentium and the Alpha that have nothing to do with the different instruction sets and which cannot be hidden by the compiler. Fully aware of this problem, Microsoft made a serious attempt to hide many of the machine dependencies in a thin layer at the bottom called the HAL (Hardware Abstraction Layer). (The

name HAL was no doubt inspired by the computer HAL in the late Stanley Kubrick's movie 2001: A Space Odyssey. Rumor has it that Kubrick chose the name "HAL" by taking the name of the then-dominant computer company—IBM— and subtracting 1 from each letter.) The job of the HAL is to present the rest of the operating system with abstract hardware devices, in particular, devoid of the warts and idiosyncracies with which

7gO CASE STUDY 2- WINDOWS 2000 CHAP. II real hardware is so richly endowed. These devices arc presented in the form of machine-independent services (procedure calls and macros) that the rest of the operating system and the drivers can use. By us,ng the HAL service;, (which are identical on all Windows 2000 systems, no matter what the hardware is) and not addressing the hardware directly, drivers and ihe kernel require fewer changes when being ported to new hardware. Porting the HAL itself is straightforward because all the machine-dependent code is concentrated in one place and the goals of the port are well defined, namely, implement all of the HAL services. The services chosen for inclusion in the HAL are those that relate to the chip set on the parentboard and which vary from machine to machine within reason-reasonably predictable limits. In other words, it is designed lo hide the differences between one vendor's parentboard and another one's, but not the differences beiween a Pentium and an Alpha. The HAL services include access to (he device registers, bus-independent device addressing, interrupt handling and resetting. DMA transfers, control of the umers and real-time clock, low-level spin locks and multiprocessor synchronnation, interfacing with the BIOS and its CMOS confi- configuration memory. The HAL does not provide abstractions or services tor specific I/O devices such as keyboards, mice, or disks or tor the memory management As an example of what the hardware abstraction layer does, consider the issue of memory-mapped I/O versus I/O ports Some machines have one and some have the other. How should a driver be programmed to use memory-mapped I/O or not? Rather than forcing a choice, which would make the driver not portable to a machine that did it ihe other way, the hardware abstraction layer offers three procedures for driver writers to use for reading the device registers and another three for writing them: uc = READ_PORT_UCHAR(port); WRlTE_PORT_UCHAR(port, uc); us = READ_PORT_USHOBT(port); WRITE_PORT_USHORT(port, us), Ul = R£AD_PORT_ULONG(port), WRlTE_PORT_LONG(port, ul); These procedures read and write unsigned

K-. |6-, and 32-bit integers, respec- respectively, to the specified port. It i-s up to the hardware abstraction layer to decide whether memory-mapped I/O is needed here. In this way, a driver can be moved without modification between machines that differ in Ihe way the device registers are implemented. Drivers often need to access specific I/O devices for various purposes. At the hardware level, a device has one or more addresses on a certain bus. Since modem computers often have multiple buses (ISA, PCI, SCSI, USB, 1394, etc.). it can happen that two or more devices have the same bus address, so some way is needed to distinguish them. The HAL provides a service for identifying devices by mapping bus-relative device addresses onto system-wide logical addresses In this way, drivers are not required to keep track of which device is on which bus.

SEC. 11 3 SYSTEM STRUCTURE '81 These logical addresses are analogous to the handles the operating system gives user programs to reter to files and other system resources. This mechanism also shields higher layers from properties of alternative hus structures and addressing Interrupts have a similar proMem-—they are also bus dependent. Here, too, Iht; HAL provides <<er\ices to name interrupts in a system-wide way and also pro- provides services to allow drivers to attach interrupt service routines to interrupts in a portable way. without having to know anything about which interrupt vector is for which bus. Interrupt request level management is also handled in the HAL. Another HAL service is setting up and managing DMA transfers in a device-independent way. Both the system-wide DMA engine and DMA engines on specific I/O cards can be handled Devices are referred to by their logical ad-addresses The HAL also implements software scatter/gather (writing or reading from noncontiguous blocks of physical memory). The HAL also m.inages clocks and timers in a portable way. Time is kept track of in units of 100 nsec starting at I January 1601, which is far more precise than MS-DOS's keeping track of time in units of 2 sec since 1 January 1980 and provides support for the many computer-related activities in the 17th. 18th, and 19th centuries. The time service;, decouple the drhers from the actual frequencies at which the clocks run. Kernel components sometimes need to synchronize at a very low level, espe- especially to prevent race conditions in multiprocessor systems. The HAL provides some primitives to manage this synchronization, such as spin locks, in which one CPU simply wans for a resource held by another CPU to be released, particularly in situations where the resource is

typically only held for a few machine instruc- instructions. Finally, after the system has been booted, the HAL talks to the BIOS and inspects the CMOS configuration memory, if any. to find out which buses and I/O devices the system contains and how they have been configured- This informa- information is then put into the registry so other system components can look it up without having to understand how the BIOS or configuration memory work. A summary of some of ihe things the HAL does is given in Fig. 1) -8. Since the HAL is highly-machine dependent, it must match the system it is installed on perfectly, so a variety of HALs are provided on the Windows 2000 CD-ROM. At system installation time, the appropriate one is selected and copied to the system directory \winnt\system32 on the hard disk as, hat.dtl. All subse- subsequent boots use this version of the HAL, Removing this file will make the system unhoo table. Although the HAL is reasonably efficient, for multimetha applications, it may not be fast enough. For this reason, Microsoft also produced a software package called DirectX, which augments the HAL with additional procedures and allows user processes much more direct access to the hardware. DirectX is somewhat specialized, so we will not discuss it further in this chapter.

782 CASE STUDY 2- WINDOWS 2000 Device Device Hardware abstraction layer Figure 11-8. Some nt ihc hardware functions lhe HAL manages The Kernel Layer Above the hardware abstraction layer is a layer thai contains what Microsoft calls the kernel, as, well as the device drivers. Some early documentation refers to the kernel as the "microkernel,"" which it really never was because the memory manager, file system, and other major components resided in kernel space and ran in kernel mode from day one. The kernel is certainly not a microkernel now since virtually the entire operating system was put in kernel space starting with NT 4.0 In the chapter on UNIX, we used the term "kernel"' to mean everything run- running in kernel mode. In this chapter we will (very grudgingly) reserve the term "kernel" for the part labeled as such in Fig. 11-7 and call the totality of the code running in kernel mode the ■'operating system." Part of kernel (and much of the HAL) is permanently resident in main memory (i.e., is not paged). By adjusting its priority, it can control whether it can tolerate being preempted by I/O interrupts or not. Although a substantial fraction of the kernel is machine specific, most of it is nevertheless written in C, except where top performance overshadows all other concerns. The purpose of the kernel is to make the rest of the operating

system com- completely independent of the hardware, and thus highly portable. It picks up where the HAL leaves off. It accesses the hardware via the HAL and builds upon the extremely low-leve| HAL services to construct higher-level abstractions. For example, the HAL has calls to associate interrupt service procedures with inter- interrupts, and set their priorities, but does little else in this area, The kernel in con- contrast, provides a complete mechanism for doing context switches. It properly saves all the CPU registers, changes the page tables, flushes the CPU cache and so on, so that when it is done, the previously running thread has been saved in tables in memory. It then sets up the new thread's memory map and loads iis registers so the new thread can start running.

I 3 SYSTEM STRUCTURE 783 The code for thread scheduling is also in the kernel When it is time to see if a new thread can run. for example, after a quantum runs out or after an I/O inter- interrupt completes, the kernel chooses the thread and does the context switch neces- necessary to run it. From the point of view of the rest of the operating system, thread switching is automatically h mdkd by louer ldjus without any work on their part jnd in a portable way. The scheduling algorithm itself will be discussed later in this chapter when we come to processes ind thrt-jds In addition [o providing a higher-level abstraction of the hardware and han- handling thread swilches. the kernel also has another key function- providing low- level support for two classes ot objects untrol objects and dispatcher objects. These objects are not the objects that user processes get handles to, but are inter- internal objects upon which |he executive builds the user objects Control objects are those objects thai control the system, including primitive process objects, interrupt objects, and two somewhat strange objects called DPC and APC. A DPC (Deferred Procedure Call) object is used to split off the run- runtime-critical part of an interrupt service procedure from ihe time critical part. Generally, an interrupt service procedure saves a few volatile hardware registers associated with the interrupting I/O device so they do not get overwritten and reenables the hardware, but saves1 the bulk ot the processing for later For example, after a key is struck, the keyboard interrupt service procedure reads the key code from & register and reenables the keyboard interrupt, but does not need to process the key immediately, especially if something more important (i.e.. higher priority) is currently going on As long as the key is processed within about 100 msec, the user will be none the wiser. DPCs are also used for tuner

expirations and other activities whose actual processing need not be instantane- instantaneous. The DPC queue is ihe mechanism for remembering that there is more work to do later. Another kernel control object is the APC (Asynchronous Procedure Call). APCs are like DPCs except that they execute in the context of a specific process. When processing a key press, it does not matter whose context the DPC runs in because all that is going to happen is that the key code will be inspected and prob- probably put in a kernel buffer. However, if an interrupt requires copying a buffer from kernel space to a buffer in some user process' address space (e.g., as it may on completion of a read frnm the modem), then the copying procedure needs to run in the receiver's context. The receiver's context is needed so the page table will contain both the kernel buffer and the user buffer (all processes contain the entire kernel in their address spaces, as we will see later). For this reason, the ker- kernel distinguishes between DPCs and APCs The other kind of kernel objects are dispatcher objects. These include sema- semaphores, mutexes, events. waitab|e timers, and other objects that threads can wait on The reason that these have to be handled (in part) in the kernel is thai they are intimately intertwined with thread scheduling, which is a kernel task. As a little aside, mutexes are called "mutants"' in the code because they were required to

784 CASE STUDY 2 WINDOWS 20A0 CHAP. II implement ihe OS/2 semantics of not automatically unlocking themselves when a thread holding one exiled, something the Windows 2000 designers cons.dered bizarre. (The OS/2 semantics are relevant because NT was originally conceived of as a replacement for OS/2, the operating system shipped on IBM's PC/2.) The Executive Above the kernel and device drivers is ihe uppei portion of the operating sys- system, called the executive, shown as the shaded area m Fig. 11-7. The executive is written m C. is architectuie independent, and can be ported to new niachines with relatively htlle effort. It consists of 10 components, each of which is just a collec- collection of procedures that work together lo accomplish some goal. There are no hard boundaries between the pieces and different authors describing the executive might even group the procedures differently inio components It should be noted that components on the same level can (and do) calf each other extensively. The object manager manages all objects known to [he operating system. These include processes, threads, files, directories, semaphores, I/O devices, timers, and many others The object manager allocates a block of virtual memory from kernel address

space when an object is created and returns it to the free list when the object is deallocated. Its job is to keep track of all the objects To avoid any confusion, mosl of the executive components labeled •'manager" in Fig. 11-7 are not processes or threads, but merely collections of procedures that other threads can execute when in kernel mode. A few of them, such as the power manager and plug-and-pi ay manager, really are independent threads though. The object manager also manages a name space m which newly created ob- objects may be placed so they can be referred to later All other components of the executive use objects heavily lo do their work. Objects are so central to the func- functioning of Windows 2000 that they will be discussed m detail in the next section. The I/O manager provides a framework for managing i/O devices and pro vides generic I/O services. It provides the rest of the system with device- independent I/O, calling the appropriate driver t<i perform physical I/O It is also home to all the device drivers (indicated by D in Fig. 11-7). The file systems are technically device drivers, under control of the I/O manager. Two different ones are present for the FAT and NTFS file systems, each one independent of the oth- others and controlling different disk partitions. A|l the FAT file systems are managed by a single driver. We will study I/O further in Sec |1.6 and one of the file systems. NTFS, m Sec. 11.7 The process manager handles processes and threads, including their creation and termination. It deals with the mechanisms used to manage them, rather than policies about how they are used. It builds upon the kernel process and thread objects and adds extra functionality to them. It is the key to multiprogramming in Windows 2000 We will study process and thread management in Sec. 11.4.

SEC. 11 3 SYSTEM STRUCTURE 785 The memory manager implement* Windows 2000's demand-paged v,rtual memory architecture. It manages the mapping of virtual pages onto physical page frames. It thereby enforce-, the protection rules that restrict each process to only access those pages belonging to its address space and not to other processes' address spaces (except under special circumstances). It also handles certain sys- system calls that relate to virtual memory. We will study memory management in Sec. 11.5 The security manager enforces Windows 2000\s elaborate security mechan- mechanism, which meets the U.S Dept. of Defense's Orange Book C2 requirements. The Orange Book specifies a large number of rules that a conforming system must meet, starting with authenticated login through how access control is handled, to the fact that virtual pages must be zeroed out before being reused. We will

study the security manager m Sec. 11.8. The cache manager keeps the most recently used disk blocks in memory to speed up access to them in the (likely) event that they are needed again. Its job is to figure out which blocks are probably going to be needed again and which ones are not It is possible to configure Windows 2000 with multiple file systems, in which case the cache manager works for all of them, so each one does not have to do its own cache management When a block is needed, the cache manager is asked to supply it. If it does not have the block, the cache manager calls upon the appropriate file system to get it. Since files can be mapped into processes' address spaces, the cache manager must interact with the virtual memory manager to provide the necessary consistency. The amount of space devoted to caching is dynamic and can increase or decrease as demands on it change. We will study the cache manager in Sec. 11 9 The plug-and-play manager is sent all notifications of newly attached devices. For some devices, a check is made at boot time and not thereafter. Orher devices, for example, USB devices, can be attached at any time and their attach- attachment triggers a message to the plug-and-piay manager, which then iocates and loads the appropriate driver. The power manager rides herd on power usage This consists of turning off the monitor and disks after they have been idle for a while. On laptops, the power manager monitors battery usage and takes action when the battery is about to run dry. Such action typically tells programs to save their files and prepare for a graceful shutdown. And pronto. The configuration manager is in charge of the registry.]t adds new entries and looks up keys when asked to. The total procedure call manager provides for a highly-efficient imerpro- cess communication used between processes and their subsystems. Since this path is needed to carry out some system calls, efficiency is critical here, which is why the standard interprocess communication mechanisms are not used. The Win32 GDI executive module handles certain system calls {but not ail of them) It was originally in user space but was moved to kernel space in NT 4.0 to

 improve performance. The GDI (Graphics Device' Interface) handles image management for the monitor and printers. Il provides system calls to allow user programs to write on the monitor and printers in a device-independent way It aho contains the window manager and display driver Prior lo NT 4.0. it, too. was in user space but (he performance was disappointing, so Microsoft moved ,t into the kernel to speed it up. It is worth mentioning that Fig. 11-7 is not ai all to «,cale For example,

the Win32 and graphics tlevn-e interface module is larger ihan the rest of the executive combined. At ihe top ot the executive is a thin layer called system services Its function is to provide an interface to the executive. It accepts ihc-true Windows 2000 sys- system calls and cails other pam of the executive lo have them executed. At boot time. Windows 2000 is loaded into memory as a collection ot files. The main part of the operating system, consisting of the kerne) and executive, is located in the file nwskrnl.exe. The HAL is a shared library located in a separate file, hal.dll. The Win32 and graphics device interface are together in a third file, wm32k.!>ys. Finally, many device drivers are also loaded. Most of these have extension .sys. Actually, ihings are not quite that simple. The ntoskinl.exe file comes tn uniprocessor and multiprocessor versions. Also, there are versions for the Xeon processor, which can have more than 4 GB of physical memory and the Pentium, which cannot. Finally, versions can consist of a free build (sold in stores and preinstalled by computer manufacturers) or a checked build (for debugging pur- purposes). Together there could be eight combinations, although two pairs were combined leaving only six. One of these is copied to ntoskrnl.exe when the sys- system is installed. The checked builds are worth a few words. When a new I/O device is installed on a PC, there i-, invariably a manufacturer-supplied driver that has to be installed to make it work. Suppose that an IEEE 1394 card is installed on a com- computer and appears to work fine. Two weeks later the system suddenly crashes. Who does the owner blame? Microsoft. The bug may indeed be Microsoft's, but some hugs are actually due lo fiakey drivers, over which Microsoft has no control and which are installed in kernel memory and have full accuss lo all kernel tables as wei| as the entire hardware In an attempt to reduce the number of irate customers on the phone, Microsoft tries to help driver writers debug their code by putting statements ol the form ASSERT(some condition) throughout the code. These statements make sanity checks on all parameters to internal kernel procedures (which may be freely called by drivers) and make many other checks as well. The free builds have ASSERT defined as a macro (hat does nothing, removing all the checks. The checked builds have it defined as #define ASSERT(a) if (! (a)) error(...)

SEC 113 SYSTEM STRUCTURE 787 causing all the checks to appear in the ntoskrnl.exe executable code and be earned out at run lime. While this slows down the system enormously, it helps driver writers debug their drivers before they ship them to customers. The checked builds also have numerous other

debugging features turned on. The Device Drivers The last part of Fig. 11-7 consists of the device drivers. Each device driver can control one or more I/O devices, but & device driver can also do things not related to a specific device, such as encrypting a data stream or even just provid- providing access to kernel data structures. Device drivers are not part of the ntoskml.exe binary. The advantage of this approach is that once a driver has been installed on a system, it is added to a list in the registry and is loaded dynamically when the system boots. In this way. ntoskrnl.exe is the same for everyone, but every system is configured precisely for those devices it contains. There are device drivers for macroscopically visible I/O devices such as disks and printers, but also for many internal devices and chips that practically no one has ever heard of. In addition, the file systems are also present as device drivers., as mentioned above. The largest device driver, the one for Win32. GDI. and video, is shown on the far right of Fig. 11-7. It handles many system calls and most of the graphics. Since customers can install new device drivers, they have the power to affect the kernel and corrupt the system. For this reason, drivers must be written with great care. 11.3.2 Implementation of Objects Objects are probably the single most important concept in Windows 2000 They provide a uniform and consistent interface to all system resources and data structures such as processes, threads, semaphores, etc. This uniformity has vari- various facets. First. a‖ objects are named and accessed the same way, using object handles. Second, because all accesses to objects go through the object manager, it is possible to put all the security checks in one place and ensure that no process can make an end run around them Third, sharing of objects among processes can be handled in a uniform way Fourth, since all object opens and closes go through the object manager, it is easy to keep track of which objects are still in use and which can be safely deleted. Fifth, this uniform model for objec! management makes it easy to manage resource quotas in a straightforward way. A key to understanding objects is to realize that an (executive) object is just some number of consecutive words in memory (i.e.. in kcruel virtual address space). An object is a data structure in RAM. no more and no less. A file on disk is not an object, although an object (i.e., a data structure in kernel virtual address space) is created for a file when it is opened. A consequence of the fact that objects are just kernel data structures is that when the system i$ rebooted (or

 crashes) all objects are

lost. In fact, when the system boots, there are no objects present at all {except for the idle and system processes, whose objects are hardwired into the ntoskrnl.exe file). All other objects are created on the fly as the system boots up and various initialization (and later user) programs run. Objects have a structure, as shown in Fig. 11-9. Each object contains a header with certain information common to all objects of all types. The fields in this header include the object's name, the object directory in which it lives in object space, security information (so a check can be made when an object is opened), and a list of processes with open handles to the object (if a certain debugging flag is enabled). Object header •yjn Object data hand) Otiect-apecilicdata T, name .-- tv Opan method Close method DaW* method Query name method Pane method Security method Figure 11-9. The structure of an objeci. Each object header also contains a quota charge field, which is the charge levied against a process for opening the object. If a file object costs 1 point and a process belongs lo a job that has 10 file points worth of quota, the processes in that job can only open 10 files in total. In this way resource limits can be enforced for each object type separately. Objects occupy valuable real estate—pieces of kernel virtual address space— so when an object is no longer needed it should be removed and its address space reclaimed. The mechanism for reclamation is to have a reference counter in each object's header. It counts the number of open handles held by processes. This counter is incremented every time the object is opened and decremented every time it is closed. When it hits 0, no more users hold handles to the object. When an object is acquired or released by an executive component, a second counter is incremented or decremented, even though no actual handle is issued. When both counters hit 0, no user process is using ihe object and no executive process is using the object, so the object can be removed and its memory freed.

SEC. 113 SYSTEM STRUCTURE 789 The object manager needs to maintain dynamic data structures (tts objects), but it is not the only pan of the executive with this need. Other pieces also need to allocate and release chunks of kernel memory dynamically. To meet these needs, lhe executive maintain!, two page pools in kernel address space: for objects and for other dynamic data structures. Such pools operate as heaps, similar to the C language calls malliK and/ree for managing dynamic data. One pool is paged and the other is nonpaged (pinned in memory). Objects lhat are needed often are kept in the nonpaged pool; objects that are rarely accessed, such as registry keys and

some security information, are kept in the paged pool. When memory ts tight, the latter can be paged out and faulted back on demand. In fact, substantial por-portions ot the operating sysiem code and data structures are also pageable, to reduce memory consumption. Objects that may be needed when the system is running critical code (and when paging is not permitted) must go in the nonpaged pool. When a small amount of storage is needed, a page can be taken from either pool and then broken up into units as small as 8 bytes. Objects are typed, which means each one has certain properties common to all objects of its type. The type is indicated by a pointer in the header to a type object, as shown in Fig. 11-9. The type object information includes items such as the type name, whether a thread can wait on the object (yes for mutexes. no for open files), and whether new objects of this type go on the paged or nonpaged pool. Each object points to its type object. The last thing a type object has is also the most important: pointers to the code foT certain standard operations such as open, close, and delete. Whenever one of these operations is invoked on an object, the pointer to the type object is followed and the relevant code located and executed- This mechanism gives the system the opportunity to initialize new objects, and recover storage when they are deleted Executive components can create new types dynamically. There is no defini- definitive list of object types, but some of the more common ones are listed in Fig. 11- 10. Let us briefly go over the object types in Fig. 11-10. Process and thread are obvious. There is one object for every process and every thread, which holds the main properties needed to manage the process or thread. The next three objects, semaphore, mutex, and event, all deal with interprocess synchronization. Sema- Semaphores and mutexes work as expected, but with various extra bells and whistles (e.g., maximum values and timeouts). Events can be in one of two states: sig- signaled ot nonsignaled. If a thread waits on an event that is in signaled state, the thread is released immediately. If the event is in nonsignaled state, it blocks until some other thread signals the event, which releases all blocked threads. An event can also be set up, so after a signal has been successfully waited for, it automati- automatically reverts to nonsignaled state, rather than staying in signaled state. Port, timer, and queue objects also relate to communication and synchroniza- synchronization. Ports are channels between processes for exchanging messages. Timers provide a way to block for a specific time interval. Queues are used to notify threads that a previously started asynchronous I/O operation has completed.

CASE STUDY 2: WINDOWS" 2000 fopenlile (Access token Profile Description Thread within a process Counting semaphore used tor interproc6ss_synchronization_ i ironization objectwith persistent state (signaled/not) J Mechanism for mterprocass mj I/O I Ob)ed allowng a thread to sleep tot a flxed time interval Object used tor completion notification on asynchrQi Object associated with an open file I Secui iData struduie used for profiling CPU usage ' j Structure used for mapping files onto virtual address space Key Object direcfory Registry key Directory for grouping ofyects within the object manager j Symbolic link J Pointer to another object by name Devica I/O device object |_Devica dnver j Each loaded device driver has its own object Figure 11-1A. Sor e objeU lypes i Open file objects are created when a file is opened. Files thai are not opened do not have objects managed by the object manager Access tokens are security objects; they identify a user and tell what special privileges the user has. if any. Profiles are structures used for storing periodic samples of the program counter of a running thread to see where the program is spending its; time Sections are objects used by the memory system for handling memory- mapped files. They record which file (or part thereof) is mapped onto which memory addresses. Keys are registry keys and are used to relate names to values. Object directories are entirely local to the object manager- They provide a way to collect related objects together in exactly the same way directories work in the file system. Symbolic links are also similar to their file system counterparts: they allow a name in one part of the object name space to refer to an objeci in a dif- different part of the objeci name space. Each known device has a device object that contains information about it and is used to refer io the device within the system. Finally, each device driver that has been loaded haj. an object in the object space Users can create new objects or open existing objects by making Win32 calls such as CreateSemaphore or OpenSemaphore. These are calls to library pro- procedures that ultimately result in the appropriate system calls being made. The result of any successful call that creates or opens an object is a 64-bit handle table

SEr 11' SYSTEM STRLiCTl'Rt 791 entry thai is stored in the process' private liandlc table in kernel memory Die 32-bit index of the handle's position in the table is returned to the user to use on -subsequent calls The 64-bit handle table entry in the kernel contains two 32-bit words One word conlains a 29-bit pointer to the object's header The low-order 3 bits are used dh flags (e.g..

whether the handle is inherited by child processes) These bits arc masked off before the pointer is followed The oiher word contains a 32-bit rights mask It is needed because permissions cheeking is done only at the time the ob|ect is created or opened It a process has only read permission to an object, all the other rights bus in the mask will be Os. giving the operating system the ability to reject any operation on the object other than reading it The handle tables tor two processes ynd iheir reiationships to some objects are illustrated in Fig. 11-11 In ihis example, process A has access to threads I and 2 and access to mutexes I and 2. Process B has access to thread 3 and mutexes 2 and 3. The corresponding entries in the handle tables hold the rights to each of ihese objects. For example, process A might have the nghts to lock and unlock its mulexes, but not the right to destroy them. Note that inutex 2 is shared by both processes allowing threads in them to synchronize The other mutexes are not shared, which might mean that the threads within process A use mutex I tor their internal synchronization and the threads within process B use mutex 3 for their internal synchronization. The Object Name Spate As objects are created and deleted during execution, the ob|ect manager needs a way to keep track of them. To do this job. it maintains a name space, in which all objects in the system are located. The name space can be used by a process to locate and open a handle for some other process' object, provided it has been granted permission to do so. The object name space is one of three name spaces maintained by Windows 2000. The other ones are the file system name space and the registry name space. All three are hierarchical name spaces with multiple lev- levels of directories for organizing entries. The directory objects listed m Fig 1 [-10 provide the means to implement thin hierarchical name space for objects. Since executive objects are volatile (i.e , vanish when the computer is .shut down, unlike file system and registry entries), when the system boots up, lhere are no objects in memory and the object name space is empty During booting, vari- various parts of the executive create directories and then fill them with objects. For example, as the plug-and-play manager discovers devices out there, it creates a device objeel for each one and enters this object into the name space. When the system is fully booted, all I/O devices, disk partitions, and other interesting discoveries are in the object name space. Not all objects get entered by the Columbus method—-|ust go look and see what you find. Some executive components look in the registry to see what to do, A key example here is device drivers. During bootup. the system looks in the

792 CASE STUDY 2: WINDOWS 2000 Figure 11-11. The relationship between handle tables, objects, and type objects registry to see which device drivers are needed. As they are loaded one by one, an object is created for each one and its name is inserted into the object space. Within the system, the driver is referred to by a pointer to its object. Although the object name space is crucial to the entire operation of the sys- system, few people know that it even exists because it is not visible to users without special viewing tools. One such viewing tool if, wtnobj, available for free at www.sysinternals.com. When run, this tool depicts an object name space that typ- typically contains the object directories listed in Fig. 11-12 as well as a few others. The somewhat strangely named directory V? contains the names of a|I the MS-DOS-style device names, such as A: for the floppy disk and C: for the first hard disk. These names are actually symbolic links to the directory XDevice where the device objects live. The name \?7 was chosen to make it alphabetically first to speed up lookup of all path names beginning with a drive letter. The contents of the other object directories should be self explanatory.
11.3.3 Environment Subsystems Going back to Fig. 11-7, we see that Windows, 2000 consists of components in kernel mode and components in user mode. We have now completed our exami- examination of the kernel mode components; now it is time to look at the user mode

.SYSTEM STRUCTURt Contents irting place for looking up MS-DOS devicesjike C^ All discovered I/O devices _ Objects corresponding to each loaded device driver I ObjectTypes The type objects shown in Fig VJ.~H ._| ■ Windows ' Objects for sending messages to all the windows , I BaseNamedObjs I User-created objects such as semaphores, mutexes, etc _! I Arcname Partition names discovered by the boot loader I I NLS [National language support objects i ■ FileSystem_ _J_File system driver objects and file system recognizer objedsi | Security Objects belonging to the security system I I Known DLLs [Key shared Nbranes that are opened early and held open Figure 11-1 components of which there are three kinds. DLLs, environment subsystems, and service processes. These components work together to provide each user process with an interface that is distinct from the Windows 2000 system call interface. Windows 2000 supports three different documented APIs' Win32. POSSX, and OS/2. Each of these interfaces has a published list t>f library calls that pro- programmers can use. The job of the DLLs (Dynamic Link Libraries) and environ- environment subsystems is (o

implement the functionality of tbe published interface, thereby hiding the true sysiem call interface from application programs. In partic- particular, the Win32 interface is the official interface for Windows 2000, Windows NT, Windows 95/98/Me. and to a limited extent. Windows CE. By using the DLLs and Win32 environment subsystem, a program can be written to the Win32 specification and run unmodified on ali these versions of Windows, even though the system calls are not the same on the various systems. To see how these interfaces are implemented, let us look ai Win32. A Winl2 program normally contains many calls 10 Win32 API functions, for example. CreateWindow, DrawMenuBar, and OpenSemaphore. There are thousands of such calls, and most programs use a substantial number of them. One possible implementation would be to statically link every Win32 program with all the library procedures that it uses. If this were done, each binary program would con- contain one copy of each procedure it used in its executable binary The trouble with this approach is thai it wastes memory if the user has multi- multiple programs open at once and they use many of the same library procedures. For example. Word, Excel, and PowerPoint all use exactly the same procedures for opening dialog boxes, drawing windows, displaying menus, managing the clip- clipboard, etc., so if a user had all of them open and active at once, there would be three (identical) copies of eacb of the libraries in memory.

794 CASE STUDY 2 WINDOWS, 2000 CHAP To avoid this, problem, all versions of Windows support shared libraries, called DLLs (Dynamic Link Libraries). Each DLL collects together a .set of closely related library procedures and their data structures into a single file, usu- usually (but not always) with extension .dll. When an application is linked, the linker sees thai some of the library procedures belong to DLLs and records this informa- information in the executable \ header. Calls to procedures in DLLs are made indirectly through a transfer vector in the caller's address space. Initially this vector h filled with 0.s, since the addresses of the procedures to be called are not yet known. When the application process is started, the DLLs that are needed are located (on disk or in memory) and mapped into ihe process' virtual address space. The transfer vector is then filled in with the correct addresses so that the procedures can then be called via the transfer vector with only a negligible loss of efficiency. The win here is that even though multiple application programs have the same DLL mapped in, only one copy of the DLL

text is needed in physical memory (but each process gets iis own copy of the private static data in the DLL) Win- Windows 2000 uses DLLs extremely heavily for all aspects of the system- Now we have enough background to see how the Win32 and other process interfaces are implemented. Each user process generally links with a number of DLLs that together implement the Win32 interface. To make an API cali. one of the procedures in a DLL is called, shown as step 1 in Fig. 11-13. What happen* next depends on the Win32 API call. Different ones are implemented in different ways. Envlrwnent (csrss.exe) i Gdi32 dll *-f~\ - — -v^Call ^i ^- User User32.dll i-Q \- ~ CaU 1 Kernel32.dll »-Q^ - - -V^aJT^/ I Win32 N UuDsysJ |3b 2a Svstem iniertace (rrtdil dll) | I- V Operating system Figure 11-13. Vanous routes taken io implement Win32 API runcium calls In some cases, the DLL calls another DLL {ntdll.dll) that actually traps to the operating system. This path is shown as steps 2a and 3a in Fig. 11-13, It is also possible Ihat the DLL does all the work itself without making a system call at all.

SYSTEM STRUCTURE 795 For other Win32 API calls a different route is taken, namely, first a message is sent to the Win32 subsystem process {csm.exe), which then does some work and then makes a system call (steps 2b. 3b, and 4b). Here, too. in some cases the environment subsystem does all the work in user space and just returns immedi- immediately. The message passing between the application process and the Win32 sub- subsystem process has been carefully optimized for performance using a special local procedure call mechanism implemented by the executive and shown as LPC in Fig. H-7. In the first version of Windows NT. virtually all the Win32 API calls took route 2b, 3b. 4b, putting a large chunk of the operating system in user space (e.g., the graphics), However, starting with NT 4.0, much of the code was put into ker- kernel mode (in the Win32/GD1 driver in Fig. 11-7) for performance reasons In Windows 2000, only a small number of Win32 API calls, for example process and thread creation, take the long route The other ones take the direct route, bypass- bypassing the Win32 environment subsystem. As an aside, the three most important DLLs are shown in Fig. 11-13, but they are not the only ones. There are over 800 separate DLLs in the \winnAs\siem32 directory totalling]30 MB. To avoid any confusion, the number of DLL filet is over 800; the number of API calls contained in them exceeds 13,000. (The 29 million lines of code had to compile into something, after all.) A few of the more important DLLs are listed in Fig. 11 -14. The number of exported functions (i

e.. those visible outside the file) in each one is given, bui these tend to change (mean- (meaning increase) over time. The number of exported functions in the first public release of nldlldU m Windows 2000 is 1179 These are the real system calls. The 1209 calls exported by nloskrnl.exe are the functions available to device drivers and other code linked with the kernel. The list of exported functions, in any exe or .dll file can be viewed using the depends program in (he platform SDK Kit. Contents Low-level hardware management, e g , port I/O ntoskrni.exe j Kernel) wm32k.sys I Kernel , Windows 2000 operating system (kernel + executive) I Many system calls including most of the graphics i 1179 Dispatcher Itom user mode to kernel mode \ kernel32.dll i^dll_ user32.dll 823 ni subsystem process Most of the core (nongraphics) system calls j Font, text, color, brush, pen, bitmap, palette, drawing, etc calls clipboard, etc calls Window, icon, menu, cursor, dUw, ..^wg,,.,,,,,, ^.k. Security, cryptography, registry, management calls J Figure 11-14. Some key Windows 2000 files, ihe mode they run m. the number of exported function calls, and the main contents of each file The calls in *in32lcsys are no! formally exported since wm32k.sys is noi called directly.

CASE STUDY 2- WINDOWS 2000 ,., would have made it usable, at no extra cost to Mic .... To allow UNIX users to migrate to Windows 2000. Microsoft has a product called Intern that provides a better degree ot UNIX compatibility than the POSIX subsystem. The OS/2 subsystem is similarly limited in rtfnctmnality and d<>es not support any graphical applications. In practice, it. too. is completely useless. Thus the original idea of having multiple operating system interfaces implemented by dif- different processes in user space is essentially gone What is left is a full Win32 impiemenlation m kernel mode and little else. 11.4 PROCESSES AND THREADS IN WINDOWS 2000 anaging the CPU and grouping will examine these, discussing how these concepts are imple- impleWindows 2000 has a number of concepts for resources together. In the following sections we some of the relevant Win32 API calls, and show mented. 11.4.1 Fundamental Concepts Windows 2000 supports traditional processes, which can communicate and vnchromze with one another, just as they can in UNIX. Each process contains at least one thread, which in turn contains at least one fiber (lightweight thread), Furthermore, processes can be collected into jobs for certain resource manage- management purposes. Together, jobs, processes- threads, and fibers provide a very gen- general set of tools for managing parallelism and resources, both on

uniprocessors (single-CPU machines) and on multiprocessors (multiCPU machines) A brief summary of these four concepts is given in Fig. 11-!5 Let us examine these concepts from the largest to the smallest. A j«b tn Win- Windows 2000 is a collection of one or more processes that are to be managed as a unit. In particular, there are quotas and resource limits associated with each job, stored in the corresponding job object. The quotas include items such as the max- maximum number of processes (prevents any process from generating an unbounded number of children), the total CPU time available to each process individually and

PROCR.SSES AND THREADS IN WINDOWS 201X1 Name ._ Description T Collection of proces; | Container for holdini lhat share quotas. Entity scheduled by |he kernel Lightweight thread managed entirely in user space 1- 15. Basic scd for CPU ,ind re to all the processes combined, and ihe maximum memory usage, again, per proc- process and total. Jobs can also impose security restrictions on the processes in the job. inch as not being able to acquire administrator (superuserl power, even with the proper password. Processes are more interesting than jobs and also more important. As in UNIX, processes are coniainers for resources. Every process has a 4-GB address space, with (he user occupying the bottom 2 GB (optionally 3 GB on Advanced Server and Dataeenter server) and the operating system occupying the rest Thus the operating system is present in every process' address, although protected from lampering by the memory management unit hardware A process has a process ID. one or more threads, a list of handles (managed in kernel mode), and .in acces token holding its security information. Processes arc created using a Win32 call thai takes as its input the name of an executable file, which defines the initial contents of the address space and creates the first thread Every process starts out with one thread, but new ones can be created dynami- dynamically. Threads, forms the basts of CPU scheduling as the operating system always selects a thread to run, not a process. Consequently, every thread has a Mate (ready, running, blocked, etc). whereas processes do not have states Threads can be created dynamically by a Win32 call that specifics the address within the enclosing process' address space it is to start running at. Every thread has a thread ID. which is taken from the same space as the process IPs. so an ID cun never be in use for both a process and a thread at the same time Process and thread IDs are multiples of tour so they can be used as byte indices into kernel tables, the same as other objecis. A thread normally

runs in user mode, but when it makes a system call it switches to kernel mode and continues to run as the same thread with the same properties and limits it had in user mode. Each thread has two stacks, one for use when it is in user mode and one for use when it is in kernel mode In addition to a state, an ID, and two stacks, every thread has a context (in which to save its regis- registers when it is not running), a private area for its own local variables, and possibly its own access token. If it has its own access token, this one overrides the process access token in order to let client threads pass their access rights to server ihreads who are doing work tor them. When a thread is finished executing, it can exit. When the last thread still active in a process exits, the process terminates,

798 CASE STUDY > WINDOWS 2000 CHAP 11]t is important to realize (hat threads are a scheduling concept, not -i resource hip concept. ile to access ail the objects that yivn^ All it has to do is grab the handle and make the appropriate Win32 call. There is no restriction on a thread that it cannot access an object because a dif- different thread created or opened it. The system does not even keep track of which thread created which object. Once an object handle has been put in a process' handle table, any thread in the process can use it. In addition to the normal threads that run within user processes, Windows 2000 has a number of daemon threads that run only in kernel space and are not associated with any user process (they are associated with the special system or idle processes). Some perform administrative tasks, such as writing dirty pages to the disk, while others form a pool that can be assigned to a component of the exe- executive or a driver that needs to get some work done asynchrunously in the back- background. We wi|l study some of these threads later when we come to memory management. Switching threads in Windows 2000 is relatively expensive because doing a thread switch requires entering and later leaving kernel mode. To provide very lightweight pseudoparallelism, Windows 2000 provides fibers, which are like threads, bui are scheduled in user space by the program that created them tor its run-time system). Each thread can have multiple fibers, the same way a process can have multiple threads, except that when a fiber logically blocks, it puts itself on the queue of blocked fibers and selects another fiber to run in the context of its thread. The operating system is not aware of this transition because the thread keeps running, even though it may be first running one fiber, then anoiher. In tact, the operating system knows nothing at all about fibers, so there are no executive

objects relating to fibers, as there are for jobs, processes, and threads. There are also no true system calls for managing fibers. However, there are Win32 API calls. These are among the Win32 API calls that do not make system calls, which we mentioned during the discussion of Fig. 11-|3. The relationship between jobs, processes, and threads is illustrated in pig. 11-16 Although we will not discuss it in much detail, Windows 2000 is capable of rnnning on a symmetric multiprocessor system. This requirement means that the operating system code must be fully reentrant, that is, every procedure must be writien in such a way that iwo or more CPUs may be changing its variables at once, without causing problems- In many cases this means that code sections have to be protected by spin locks or mutexes to keep additional CPUs at bay until the first one is done (i.e., serialize access to critical regions). The number of CPUs the system can handle is governed by the licensing restrictions and listed in Fig. 11-2. There is no technical reason why Windows Professional cannot run on a 32-node multiprocessor—it is the same binary as Datacenter Server, after all The upper limit of 32 CPUs is a hard limit because word-length bitmaps are used to keep irack of CPU usage in various ways. For example, one word-length

SEC I I 4 PROCESSES AND THREADS IN WINDOWS 2000 799 Add ' " Thread "" - £ f a DD- Process -~-B n n - Emm- A«„„«>k.n —~m DD Figure 11.F. The relationship between jobs, processes,. Md thresh. Several fibers can also be mulliplesedon one iliread (not shown) bitmap keeps track of which of the (up to) 32 CPUs are currently idle, and another bitmap i.s used per process to list ihe CPUs this process is permitted to run on. The 64-bit version of Windows 2000 should be able to effortlessly support Up to 64 CPUs; beyond Ihat requires actually changing the code substantially (to use multiple words for the bitmaps). 11.4.2 Job, Process, Thread and Fiber Management API Calls Nev ses are created using the Win32 API function CreateProcess This function has 10 parameters, each of which has many options. This design is clearly a lot more complicated than the UNIX scheme. Hi wliich fork has no param-parameters, and exec has just three; pointers to the name of the file to execute, the (parsed) command line parameter array, and (he environment strings. Roughly speaking, the |0 parameters to CreateProcess are as follows: 1. A pointer to the name of the executable file 2. The command]me itself (unparsed). 3. A pointer to a security descriptor for the process. 4. A pointer to a Security descriptor for the initial thread. 5. A bit telling whether the new

process inherits the creator's handles. 6. Miscellaneous flags (e.g., error mode, priority, debugging, consoles). 7. A pointer to the environment strings.

800 CASE STUDY 2 WINDOWS 200A CHAP II 8 A pointer to the name of the new process" current working directory- 9. A pointer to a structure describing the initial window on the screen 10. A pointer to a structure that returns 18 values to the caller Windows 2000 docs not enforce any kind of parenl-child or Other hierarchy. All processes are created equal (no processes drc created more equai) However, since 1 or [he 18 parameters returned to the creating process is a handle to the new process (allowing considerable control over (he new process), there is an implicit hierarchy in terms of who has a handle to whom Although these handles cannot just be passed directly (o other processes, there is u way for a process to make a duplicate handle suitable for another process and then give n [he handle, so the implicit process hierarchy may not last long. Each process in Windows 2000 is created with a single thread, but a process can create more threads later on. Thread creation is simpler than process creation: CreateThread has only six parameters instead of JO, 1 The optional security descriptor 2- The initial stack size. 3 The starting address. 4. A user-defined parameter. 5. The initial state of the thread (ready or blocked) n The thread's ID. The kernel does the thread creation, so it is clearly aware of threads (i.e., they .ire not implemented purely in user space as if. the case in some tither systems) Interprocess Communication Threads can communicate in a wide variety of ways, including pipes, named pipes, mailslols. socke[s. remote procedure calls, and shared files. Pipes have two modes: byte and message, selected at creation time Byte-mode pipes work the same way as in UNIX. Message-mode pipes are somewhat similar but preserve message boundaries, so that four writes of 128 bytes wil| be read as four 128-bjte messages, and not as one 512-byte message, as might happen with byte-mode pipes. Named pipes also exist and have the same two modes as regulai pipes. Named pipes can also be used over a network, regular pipes cannot Mailslols are a feature of Windows 2000 not present in LNIx. They are simi- similar to pipes m some ways, but not a‖. For one thing, they are one-way, whereas pipes are two-way They can also be used over a network but do not provide guaranteed delivery. Finally, they allow the sending process to broadcast a mes- message to many receivers, instead of lo tusi one receiver.

SEC.] I 4 PROCESSES AND THREADS IN WINDOWS 2000 801 Sockets

are like pipes, except that they normally connect processes on dif- different machines. For example, one process writes to a socket and another one on a remote machine reads from it Sockets can also be used to connect processes on the same machine, but since they entail more overhead than pipes, they are gen erally only used in a networking context Remote procedure calls are a way for process A to have process B call a pro- procedure in B's address space on A's behalf and return the result to A. Various res trictions on the parameters exist. For example, it makes no sense to pass a pointer to a differeni process. Finally, processes can share memory by mapping onto the same file at the same time. Al! writes done by one process then appear in the address spaces of the other processes. Uumg this mechanism, the shared buffer used in produLei consumer problems can easily be implemented. Just as Windows 2000 provides numerous interprocess communication mechanisms, it also provides numerous synchronization mechanisms, including semaphores, mutexes, critical regions, and events. All of these mechanisms work on threads, not processes, so that when a thread blocks on a semaphore, other threads in thai process (if any) are not affected and can continue to run, A semaphore is created using the Create Semaphore API function, which can initialize it to a given va|ue and define a maximum value as well Semaphores are kernel objects and thus have security descriptors and handles The handle for a semaphore can be duplicated using Duplicate Han die and passed to another process so that multiple processes can synchronize on the same semaphore Calls for up and down are present, although they have the somewhat peculiar names of ReleaseSemaphore (up) and WaitForSingleObiecf (down). It is also possible to give WaitForSingleObject a timeout, so the calling thread can be released eveniu- ally, even if the semaphore remains at 0 (although timers reintroduce races). Mutexes are also kernel objects used for synchronization, but simpler than semaphores because they do not have counters. They are essentially locks, with API functions for locking (WaitForSingleObject) and unlocking (ReleaseMutex). Like semaphore handles, mutex handles can be duplicated and passed between processes so that threads in different processes can access the same mutex The third synchronization mechanism is based on critical sections, (which we have called critical regions elsewhere in this book) which are similar to mutexes, except local to the address space of the creating thread. Because critical sections are not kernel objects, they do not have handles or security descriptors and cannot be passed between processes. Locking and unlocking is done with EnterCritical- Secfion and LeaveCriticalSecfion.

respectively. Because these AP| functions are performed initially in user space and only make kernel calls when blocking is needed, they are faster than mutexes. The last synchronization mechanism uses kernel objects called events, of which there are two kinds: manual-reset events and auto-reset events Any event can be in one of two states: set and cleared. A thread can wait for an event

gO2 CASE STUDY 2: WINDOWS 20O0 CHAP]l to occur with WaitForS ingle Object. If another thread signals an event with SetEvent, what happens depends on the type of event. With a manual-reset event, all waiting threads are released and the even! stays sei until manually cleared with ResetEvent. With an auto-reset event, if one or inore threads are waiting, exactly one thread is released and the event is cleared. An alternative operation is Pulse- Event, which is like SetEvent except ihat if nobody is waiting, the pulse is lost and the event is cleared. In contrast, a SetEvent thai occurs with no waiting threads is remembered by leaving the event in set state so a subsequent ihread waiting on it is released immediately. Hvents. mutexes, and semaphores can all be named and stored in the file sys- system, like named pipes. Two or more processes can synchronize by opening the same event, mutex, or semaphore, rather than having one ot them create the object and then make duplicate handles for the others, although the latter approach is certainly an option as well- The number of Win32 API calls dealing with processes, threads, and fibers is nearly 100, a subsiantial number of which deal with 1PC in one torm or another A summary of the ones discussed above as well as some oiher important ones are given ,n Fig. 11-17. Most of the calls in pig- 11-17 were either discussed above or should be suit- explanatory. Again note lhat not all of these are system calls. As we mentioned earlier, Windows 200A knows, nothing about fibers They are entirely imple- implemented in user space. As a consequence, the CreateFiber call does us work entirely in user space without making any system calls (unless it has to allocate some memory). Many other Win32 calls have this property as well, including EnterCriticaiSection and LeaveCriticalSection as we noted above. 11.4.3 Implementation of Processes and Threads Processes and threads are more important and more elaborate than jobs and fibers, so we will concentrate on them here A process is created when another process makes the Win32 CreateProcess call. This call invokes a (user-mode) procedure in kerneB2.dll thai creates the process in several steps using multiple sysiem calls and other work. 1. The executable file given as a

parameter is examined and opened If it is a valid POSIX. OS/2, 16-bit Windows. or MS-DOS file, a special environment is set up for it. If it is a valid 32-bit Win32 .eve h|e. the registry is checked to see if it is special in some way (e.g., to be run under supervision of a debugger) All of this is done in user mode inside kerneU2.dll. 2. A sysiem call, NtCreateProcess, is made to create the empty process object and enter it into the object manager's name space. Both the Kernel object and the executive object are created. In addition, the

PROCfcSSES AND THREADS IN WINDOWS 2000 t to signaled then to nonsignated iutex to allow another thread to acquir s semaphore count Dy 1 Acquite the lock on a critical section he lock process manager creates a process control block for Che object and initializes it with the process ID, quotas, access token, and various other fields. A section object is also created to keep track of the process" address space. When kerneU2.dll gets control back, it makes another system call. NtCreaieThread, to create the initial thread. The thread's user and kerne) stacks are also created. The stack size is given in the header of the executable file. Kernel32.dll now sends a message to the Win32 environment subsys- subsystem telling it about the new process and passing it the process and thread handles. The process and threads are entered into the subsys- subsystems tables so it has a complete list of a|l processes and threads. The subsystem then displays a cursor containing a pointer with an hour- hourglass to tell the user that something is going on but that the cursor can be used in the meanwhile. When the process makes its first GUI

804 CASE STUDY 2 WINDOWS 20O0 CHAP II call, usually !o create a window. !be cursor is removed (it times out after 2 seconds if no call is forthcoming). 5 At this point, the thread is able to run. It staris out by running a run- runtime system procedure to complete the initialization. 6 The run-time procedure sets the thread's priority, tells loaded DLLs that a new thread is present, and does other housekeeping chores. Finally, it begins running the code of the process' main program. Thread creation also consists of a number of steps, but we wi|! not go into them in much detail. It starts when the running process executes CreateThread, which calls a procedure inside kernel32.dll. This procedure allocates a user stack within the calling process and then makes the NtCreateThread call to create an executive thread object, initialize it, and also create and initialize a thread control block. Again, the Win32

subsystem is notified and enters the new thread in its tables. Then the thread starts running and completes its own initialization When a process or thread is created, a handle is returned for it. This handle can be used to start, stop, kill, and inspect the process or thread. It is possible for the owner of a handle to pass the handle to another process in a controlled and secure way. This technique is used to allow debuggers to have control over the processes they are debugging. Scheduling Windows 2000 does not have a central scheduling thread. Instead, when a thread cannot run any more, the thread enters kernel mode and runs the scheduler itself to see which thread to switch to. The following conditions cause ihe currently running thread to execute Che scheduler code 1. The thread blocks on a semaphore, mutex. event. I/O, etc 2. It signals an object (e.g., does an up on a semaphore) 3. The running thread's quantum expires In case 1, the thread is already running in kernel mode to carry out the operation on the dispatcher or I/O object. It cannot possibly continue, so it must save its own context, run the scheduler code to pick its successor, and load that thread's context to start it. In case 2, the running thread is in the kernel, too. However, after signaling some object, it can definitely continue because signaling an object never blocks. Still, [be thread is required to run the scheduler to see if the result of its action has released a higher priority thread that is now tree to run. If so, a thread switch occurs because Windows 2000 is fu|ly preemptive (i.e , thread switches can occur at any moment, not just at the end of the current thread's quantum).

SEC 114 PROCESSES AND THREADS IN WINDOWS 2000 805 In case 3, a trap to kernel mode occurs, at which time Che thread executes Ihe scheduler code lo see who runs next. Depending on what other threads are wait- waiting, the same thread may he selected, in which case it gets a new quantum and continues running. Otherwise a thread switch happens. The scheduler is also called under two other conditions: 1. An I/O operation completes. 2. A timed wait expires In the firsi case, a thread may have been waiting on this I/O and is now released to run. A check has to be made to see if it should preemp! Ihe running thread since there is no guaranteed minimum run time. The scheduler is not run in the inter- interrupt handler itself (since that may keep interrupt;, turned oft too long). Instead a DPC is queued for slightly later, after the interrupt handler is done. In the second case, a thread has done a down on a semaphore or blocked on some other object, but with a timeout that has now expired. Again it is necessary for the interrupt handler to queue a DPC to avoid

having it rnn during the clock interrupt handler. If a thread has been made ready by this timeout, the scheduler will be rnn and if nothing more important is available, the DPC will run next. Now we come to the actual scheduling algorithm. The Win32 API provides two hooks for processes to influence thread scheduling- These hooks largely determine the algorithm. First, there is a call Set Priority Class that sets the priority class of all the threads in the caller's process. The allowed values are. realtime, high, above normal, normal, below normal, and idle. Second, there is a call SetTh read Priority that sets the relative priority of some thread (possibly, but not necessarily, the calling thread) compared to the other threads in its process. The allowed values are- time critical, highest, above nor- normal, normal, below normal, lowest, and idle. With six process classes and seven thread classes, a thread can have any one of 42 combinations This is Ihe inpul to the scheduling algorithm. The scheduler works as follows. The system has 32 priorities, numbered from 0 to 31- The 42 combinations are mapped onto the 32 priority classes according to the table of Fig. 11-18. The number in the table determines the thread's base priority. In addition, every thread has a current priority, which may be higher (but not lower) than the base priority and which we win discuss shortly. To use these priorities for scheduling, the system maintains an array with 32 entries, corresponding to priorities 0 through 31 derived from (he lable of Fig. \l- 18. Each array entry points to the head of a list or ready threads at the corresponding priority The basic scheduling algorithm consists of searching the array from priority 31 down to priority 0. As soon as a nonempty slot is found, the thread at the head of the queue is selected and run for one quantum. If (he quantum expires, the thread goes to the end of the queue at its priority level and ihe thread ai the front is chosen next. In other words, when there are multiple

CASE STUDY 2: WINDOWS 20O0 Win32 process class priorities I Above Below : High I Normal ; Normal Normal I ■18. Mjppmgol Win32 prionii >v.s2000pnorme threads ready at the highest priority level, they run round robin for one quantum each. If no thread is ready, the idle thread is run. it should be noted Chat .scheduling is done by picking a thread without regard to which process thai thread belongs. Thus the scheduler does not first pick .1 process and Chen pick a thread in thai process. It only looks at ihe threads- it does noi even know which thread belongs to which process. On a nmItiprocessor, each CPU schedules itself using the priority array. A spin lock

m used to mate $ure that only one CPU at a time is inspecting the array. The array of queue headers is shown m Fig 1 i-|9. The figure shows that there are aciually four categories of priorities: realtime, user, zero, and idle, which is effectively -], These deserve some comment. Priorities 16-31 are called real time, but they are not. There are no guarantees given and no deadlines are met They are simply higher priority Chan 0-15. However, priorities 16 through 31 are reserved for the system itself and for threads explicitly assigned those priorities by the system administrator. Ordinary users may not run there for a good reason. If a user thread were 10 run aL a higher priority than, say, the keyboard or mouse thread and get into a loop, the keyboard or mouse thread would never nin, effec lively hanging the system. U^er threads run at priorities 1 —15. By setting the process and thread priori- priorities, a user can determine which threads get preference. The zero thread runs, 111 the background and eats up whatever CPU time nobody else wants. Its job is id zero pages for [he memory manager We will discuss its role later. If there is absolutely nothing 10 do, nof c-ven zero pages, the idle thread runs it is not reallj a full-blown ihread though. Over the course of tiine, some patches were made to Che basic scheduling algorithm to improve system performance Under certain specific conditions. Che current priority of a user thread can be raised above the base priority (by the operating system), but never above priority |5. Since the array of Fig. 11-19 is

PROCESSES AND THREADS IN WINDOWS 2000 Figure]]-19. Windows 2000 supports 32 pl ^s for threads. based on the current priority, changing this priority affects scheduling. No adjust- adjustments are ever made to threads running at priority 15 or higher. Let us now see when a thread's priority is raised. First, when an I/O operation completes and releases a waiting thread, the priority is boosted to give it a chance to run again quickly and start more I/O. The idea here is to keep the I/O devices busy. The amount of boost depends on the I/O device, typically | for a disk, 2 for a serial line, 6 for the keyboard, and 8 for the sound card. Second, if a thread was waiting on a semaphore, mutex, or other event, when it is released, it gets boosted by 2 units if it is in the foreground process (the proc- process controlling the window to which keyboard input is sent) and 1 unit otherwise. This fix tends to raise interactive processes above the big crowd ai level 8 Finally, if a GUI thread wakes up because window input is now available, it gets a boost for the same reason. These boosts are not forever. They take effect immediately, but if a

thread uses all of its next quantum, it loses one point and moves down one queue in the priority array. If it uses up another full quantum, it moves, down another level, and so on until it hits its base level, where it remains until it (s boosted again. Clearly, If a thread wants good service, it should play a lot of music. There is one other case in which the system fiddles with the priorities. Ima- Imagine that two threads are working together on a producer-consumer type problem. The producer's work is harder, so it gets a high priority, say 12, compared (o the consumer's 4. At a certain point, the producer has filled up a shared buffer and blocks on a semaphore, as illustrated in Fig. I] -20(a).

CASE STUDY 2- WINDOWS 2000 nd nke to do an up ar gels scheduled Figure 11-20. An evample of pnorny inversion Before the consumer gets a chance to run again, an unrelated thread at priority 8 becomes ready and siarts running, as shown in Fig. 1 l-20(b). As long as this thread wants to run, it will be able to, since it outguns the consumer and the pro- producer, while higher, is blocked. Under these circumstances, the producer will never get to run again until the prioriiy 8 thread gives up Windows 2000 solves this problem through what might be charitably called a big hack. The system keeps track of how long it has been since a ready thread ran last. If it exceeds a certain threshold, it is moved to priority 15 for two quanta. This may give it the opportunity to unblock the producer. After the two quanta are up, the boost is abruptly removed rather than decaying gradually Probably a better solution would be to penalize threads that use up their quantum over and over by lowering their prioriiy. After all, the problem was not caused by ihe starved thread, but by the greedy thread This problem is well known under the name priority inversion. An analogous problem happens if a prionly 16 thread grabs a mutex and does not get a chance to run for a long time, starving more important system threads that are waiting for the mutex. This problem can be prevented withm the operat- operating system by having a thread lhat needs a mutex for a short time just disable scheduling while il is busy. On a multiprocessor, a spin lock should be used Before leaving the subject of scheduling, it is worth saying a couple of words about the quantum. On Windows 2000 Professional (he default is 20 msec; on uniprocessor servers it is 120 msec; on multiprocessors various other values are used, depending on the clock frequency. The shorter quantum favors interactive users whereas the longer quantum reduces context switches and thus provides better efficiency. This is the meaning of the last column m Fig 11-2. These defaults can be increased manually by 2x, 4x. or 6x

if desired. As an aside, the sue of the quantum was chosen a decade ago and not changed since although ma- machines are now more than an order of magnitude faster. The numbers probably

SEC] i 4 PROCKSSES AND THREADS IN WINDOWS 2(MX) 809 could be reduced by a factor of 5 to 10 witli no harm and possibly better response time for interactive threads in a heavily loaded system One last patch to the scheduling algorithm says that when a new window becomes the foreground window, all of its threads get a longer quantum by an amount taken from Ihe registry. This change gives them more CPU time, which usually translates to better service for the window that just moved to the fore- foreground 11.4.4 MS-DOS Emulation One of the design goals of Windows 2000 wa.s inherited from NT: try to run many reasonable MS-DOS programs' as possible. This goal is quite different 98 l l ld MSDOS rogam (to which we add: many reasonable MSDOS programs as possible. This Windows 98"s stated goal' run all old MS-DOS progr matter how ill-behaved they may be) The way Windows 20O0 deals wiih ancient pr to run them in a lully started, a normal Win32 The way Windows 20O0 deals wiih ancient programs is to run them in a lully protected environment When an MS-DOS program is started, a normal Win32 process is started and loaded with an MS-DOS emulation program, ntvdm (NT Vir- Virtual DOS Machine) that will monitor the MS-DOS program and carry out its sys- system calls Since MS-DOS only recognized memory up to 1 MB on the 8088 and) calls. Since MS-DOS y g only up to 16 MB with bank switching a ntvdm high in the process' virtual addres address il. This situation is shown in Fig y y mory up to 1 MB on the 8088 and other tricks on ihc 286. il is safe lo put spaix where the program has no way to 1 2 (Trampoline - S / \ it 1 -CZ Opera ~-v. tvdm t 1 3 |2| WmS^DOS program ing system Figure 11-21. How .ild MS DOS pi <n imdci K indows 201X1 When the MS-DOS program is just executing normal instructions, il can run on the bare hardware since the Pentium includes all the 8088 and 286 in.structions as subsets. The interesting part is what happens when Ihc MS-DOS program wants to do I/O or interact with the sysiem A well-behaved piograin just makes a system call. In expectation of this, ntvdm instructs Windows 2000 to reflect all MS-DOS system calls back to it. In effect, the system call just bounces off the operating

emulator, as shown in step;, 1 and 2 in Fig. 11-21 Sometimes this technique is referred to as using a trampoline. Once it gets control, the emulator figures out what (he prngram was trying to do and issues its own Win32 calls to get the work done (step 3 and 4 in Fig. 11 - 21) As long as the program is well behaved and just makes legal MS-DOS system calls, this technique works fine. The trouble is that some old MS-DOS programs bypassed the operating system and wrote directly to the video RAM, read directly from the keyboard, and so on. things that are impossible in a protected environ- environment. To the extent that the illegal behavior causes a trap, there is some hope that the emulator can figure out what the program was trying to do and emulate it. If it does ut)t know what the program wants, the program is just killed because 100 percent emulation was not a Windows 2000 requirement. 11.4.5 Booting Windows 2000 Before Windows 2000 can start up. it must be booied The boot process creates the initial processes that bring up the system. In this section we will briefly discuss how the boot process works for Windows 2000. The hardware boot process consists of reading in the first sector of the first disk (the master boot record) and jumping to it, as we described in Sec 5.4.2. This short assembly language program reads the partition table to see which partition contains the bootable operating system. When it tinds the operating system partition, it reads in the first sector of that partition, called the boo! sector, and jumps to it. The program in the boot sector reads its partition's root directory, searching for a file called nlldr (another piece of archaeological evidence that Windows 2000 is really NT). If il finds that file, it reads the file into memory and executes it Ntfdr loads Windows 2000- As an aside, there are several versions of the hoot sector, depending on whether ihe partition is formatted as FAT-ln. FAT-32. or NTFS. When Windows 2000 is installed, the correct version ot the master boot record and bool sector are written to disk. Nlldr now reads a file called Boot mi, which is the only configuration mior- mation not in the registry It liMs all the versions ot full (ill and ntoskrnI. <• ye avail- available for booting in this partition. The file also provides many parameters, such as how many CPUs and how much RAM to use. whether to give user processes 2 GB or 3 GB. and what rate to set the real-time clock to Nlldr then selects and loads hal.dll and ntoskrnl.exe files as well as bootvuidll, the default video driver for writing on the display during the boot process Nlldr next reads the registry to find out which drivers are needed to complete the boot (e.g. the keyboard and mouse drivers, but also dozens more for controlling various chips on the parent- board). Finally, it reads in a‖ these

drivers and passes control to ntoskml.exe. Once started, the operating system does some general initialization and then calls the executive components to do their own initialisation. For example, the object manager prepares its name space to allow other components call it to insert

SEC 11 4 PROCESSES AND THREADS IN WINDOWS 2000 811 their objects into the name space. Many component also do specific things related to their function, such as the memory manager setting up the initial page tables and the plug-and-piay manager finding out wbich I/O devices are present and loading their drivers All in all. dozens of steps are involved, during which time the progress bar displayed on the screen is growing in length as steps are completed. The lasl siep is creating the first true user process, Ihe session manager, stms.exe Once this process is up and running, booting is completed. The session manager is a native Windows 2CH10 process. It makes true system calls and does not use (he Win32 environment subsystem, which is nol even run- running yet. In tact, one of its first duties is to start it (.ctrxs.exe). It also reads (he registry1 hives from disk and leam.s what else it is supposed to do Typically its work includes euteiing many objects in [he object manager's name space, creating an> extra paging tiles needed, and opening important DLLs to have them around all the time. After it has done most of ihis work, it creates the login daemon, Al this point, the operating system is up and running. Now it is iime to get ihe service processes (user space daemons) going and allow users to log in. Winlogon.exe first creates the authentication manager {ha\s.exe), and then ihe parent process of all the services (servtces.eve) The latter looks in the registry to find out which user space daemon processes are needed and what files they are in. It then starts creating them. They are shown in Fig [1-7 The fact that the disk is generally being heavily used after the first user has logged in (but has done noth- nothing) is not the user's fault. The culprit is services.exe creating a|l the services In addition, it also loads any remaining device drivers that have not yet been loaded. The hierarchy of initial processes and some typical services are shown in Fig 11- 22. Wmlogan.exi' is also responsible for all user logins The actual login dialog is handled by a separate program in msgina.dll io make it possible for third parties to replace the standard login with faeeprmt identification or something else other than name and password After a successful login, winlogun-t-xe gets the user's profile from the registry and from n determines which shell to run. Many people do not realize it. but (he standard Windows desktop is just explorer exe

w ith some options set. If desired, a user can select any other program as the shell, including the command prompt oi even Word, by editing the registry. However, editing the registry is not for the faint of heart; a misiake here can make the system unusable 11.5 MEMORY MANAGEMENT Windows 2000 has an extremely sophisticated virtual memory system It has a number of Win32 functions for using it and part of the executive plus six dedi- dedicated kernel threads for managing it. In the following sections we will look at the fundamental concepts, the W>n32 API calls, and finally the implementation.

CASE STUDY 2 WINDOWS 2000 j Not really a process, but home to the idle thread Pnnler server File server Telnet daemon Incoming email handler Incoming fax handler DNS resolver Event logger Plug-and-play manager Cieates smss.exe & paging First real proc, much initialization, create: Win32 subsystem process Login daemon Authentication manager Looks in registry and starts services Allows remote |obs to use llie printer Serves requests for local files Allows remote logins Accepts and stores inbound email Accepts and prints inbound faxes Internet domain name system server Logs various system events Monitors hardware to see what is out thei registry, opens DLLs 8 wirlogon 11,5.1 Fundamental Concepts In Windows 2000, every tual addressee are 32 bits long The lower 2 GB minus about: the upper 2 GB map onto ken space is demand paged, with s The virtual address space 23 in slightly simplified form, address space is normally um catch programming emirs In them on Windows 2000 will or. worse yet, writing to an L DOS programs are being run i Starting at 64 KB conies i almost 2 GB. The |a,st piece < and timers that are shared am allows processes lo access the; The upper 2 GB contains the paged and nonpaged pouli> among all user processes, exct page tables. The upper 2 Gl iser process has its own virtual address space Vir- , so each process has 4 GB of virtual addiess space. 156 MB are available tor the process' code and data, iet meiiiorv in a protected wav The virtual address fixed page si/e D KB on the Pentium) layout for thiee user processes is shown m hg. I i- The bottom and top 64 KB of i'aeh process' virtual mapped This choice was made intentionally to help alid pointers are often 0 or -I. so ailcmpls to use use an immediate trap instead of reading garbage orrcct memory location However, when old MS- emulation mode. Ihev can be mapped in he user's private code and dau. This extends up to of the bottom 2 OB contains some system counters ong all users read oulv Making them \i<nbL' here >n without the overhead of a system call the operating

system, including the code. data, and , (Used for objects, etc.). The upper 2 GB is shared ;pt for the page tables, which are each process' own 3 ot memory is not writable and mostly not even

MEMORY MANAGEMENT Stacks, datable HAL + OS NonpagwJpod ' Pagedpotf _ Process B's and data C;s_paae_tabjes Stacks, data, ate Nonpaged pool HAL + OS e shared a ng all readable tor user-mode processes The reason for putting it here k chat when a thread makes a system call, it traps into kernel mode and just keeps on running in the same thread. By making the whole operating system and all of its data struc- structures (as well as the whole user process) V]Sible within a thread's address space when it enters kernel mode, there is no need to change the memory map or flush the cache upon kernel entry. AH that has to be done is switch (o [he thread's ker- kernel stack. The trade-off here is less private address space per process in return for faster system calls. Large database servers already tee| cramped, which is why the 3-GB user space option is available on Advanced Server and Datacenter S Ser Each virtual page can be in one of three st; A free page is not currently in use and a referen a process is starter], all of its pages are in free data are mapped into its address space. Once l\ the page is said to be committed. A refe tes' free, reserved, or committed, :e to it causes a page fault. When ;iate until the program and initial de or data is mapped onto a page, utted page is mapped ising the virtual memory hardware and succeeds if the page [f the page is not in main memory, a page fault occurs and ihe operating system finds and brings in the page from disk A virtual page can also be in reserved slate, meaning it is not available for being mapped unt[l the reservation is explicitly removed For example, when a

814 C'ASfcSTl'DY? WINDOWS 2(KH CHAP. II new thread is created, I MB oi suck space is reserved in ihc process' virtual address space, but onlj one page is committed. This technique means thai Hk- stack can eventually grow to I MB without lear that some other thread will allo cate the needed contiguous piece ot virtual address space oui trom undei it hi addition to the free, reseived. and committed attributes, pares also have othei attributes, such as bein^ readable, writable, and executable. na*'t'h A sinihle strategy would be to assign j page in one ot the paging files to back up each committed page at the time ihe page was committed. This would should it he necessary to «\ ict it from memory. The downside of this strategy !-■ that the paging file might have

to he as Urge as the union ot dll processes' virtual ihis approach would waste disk space. To avoid wasting disk space. Windows 2000 committed pages that have m) natural home on the disk (e.g.. stack pages) are not assigned a disk page until the moment that they have to be paged out This design makes the system more com- complex beeau.se the paging tiles maps may have to be fetched during ;i page fauli. and fetching them may cau&e one or mcire additional page f.iults iiiside ihe page- fault handler. On (he other hand, no disk space need be allocated for pages that are never paged out. Trade-offs like this (system complexirv versus belter performance o< moro features) tend to get resolved in tavoi of the latter because the value ot bettci pei- formance or more features is clear but the downside of complexity (a bigger maintenance headache and m(,re crashes pei year) is hard to quantity. Free and reserved pages never have shadow pages on disk and references to them always cause page faults. The shadow pages on the disk are arranged into one or more paging files. There may be up to 16 paging files, possibly spread over 16 separate disks, for higher I/O bandwidth. Each one has an initial size and a maximum size it can grow to later if needed. These files can be created at ihe maximum size at system installation time in order to reduce the chances that they aru highly fragmented, but new ones can be created using the control panel later on The operating sys- system keeps track ot which virtual page maps unto which part of which paging file For (execute only) program text, the executable binary file (i e.. exe or dll filei contains ihe shadow page.s; for data pages, the paging tiles are used. Windows 2000. like many versions of UNIX, allow; files to be mapped directly onto regions of the vuiual address spaces d e . runs ot consecutive pages' Once a tile has been mapped onto the addiess space, ii can be read or written using ordinary memory references. Memory-mapped files are implemented in ihe same way as other committed pages, only the shadow pages are in the user's file instead ot in the paging tile. As a result, while a file is mapped in, the version in memory may not be identical to ihe disk veisiort (due to recent wnlet, to the vir-

 ally flushed. SfcC. 11.5 tua| address space) However, when the file is unmapped or i Che disk version is brought up to dale. Windows 2A00 explicitly allows iwo or more processes to map onlo ihe same part iii ihe same file .it the same time, possibly ai different virtual addresses, as shown in Fig 11-24 By reading and writing memory words, the processes can now communicate with each oiher and pass dau Kn-k and forth

a(very high bandwidth, since no copying is required Different processes may have different access permissions- Since all the processes using a mapped tile share the same pages, changes made by one or them are immediately visible to all the others, even if the disk file has not yet been updated. Care is also taken that if another process opens ihe file for normal reading, it see.s the current pages in RAM. not stale pages from the disk. Backing Prog- Figure 11-24. Mapped regions with their iJwdow pages on disk The IihJII f\\c is tnappetl inio iwo address spates ai ihe same lime It is worth noting that there is a problem if two programs share a DLL file and oiie of them changes the fi|e"s static data. If no special action is taken, the other one will see Che changed data, which is probably not what is desired. The prob- problem is solved by mapping all pages in as read only by secretly noting thai some are really writable. When a write happens to a page that is mapped read only but is really writable, a private copy of Che page is made and mapped in. Now it can be written safely without affecting other users or the original copy on disk. This technique is called copy-on-write. Also it is. worth noting Chat if program text is mapped into (wo address spaces at different addresses, a certain problem arises wiCh addressing. What happens if the first instruction is JMP 300? If process one maps Che program in at address

816 CASE STUDY 2. WINDOW'S 2000 CHAP, il 65 536 the code can easily he patched to read JMP 65836. But what happens it a second'process now maps' it in at 131.072' The JMP 65836 will go to jddrcss 65,836 instead of 131,372 and the program will fail The solution is to u->e only relative offsets, not absolute virtual addresses in code that is to be shared. For- Fortunately, most machines have instroctions using relative offsets as well as instruc- instructions using absolute addresses Compilers can use the relative offset instructions, bui they have to know in advance whether to use them or the absolute ones The relume ones are not used all the time because the resulting code is usually less efficient. Usually, a Lompiler flag tells them which io use. The technique of making it possible to place a piece of code ai any virtual address without reloca- relocation is called position independent code. Yc;irs ago, when 16-bit (or 20-bit) virtual address spaces were standard, but machines" had megabytes of physical memory, all kinds of tncks were thought of to allow programs to use more physical memory than fit in the address space Often these tricks went under the name of bank switching, in which a program could substitute some block of memory above the |6-bitor 20-bit limit tor a

block of its own memory. When 32-bit machines were introduced, people thought they would have enough address space forever. They were wrong The problem is back Large programs often need more than the 2 GB or 3 GB of user address space Windows 2000 allocates to them, so bank switching is back, now called address windowing extensions. This facility allows programs to map into shul- fle chunks of memory in and out of the user portion of the address space (and especially above the dreaded 4-GB boundary) Since it is only used on servers with more than 2 GB of physical memory, we will defer the discussion until the next edition of this book (by which time even entry-level desktop machines will be feeling the 32-bit pinch). 11,5.2 Memory Management System Calls The Win32 API contains a number of functions that allow a process to manage iU virtual memory explicitly The most important of these functions are listed in Fig 11-25. All of them operate on a region consisting dlhei of a single page or a sequence of two or more pages that are consecutive in the virtual address space. The first four API functions are used to allocate, free, protect and query regions ot virtual address space. Allocated regions always begin on 64-KB boun- boundaries to minimize porting problems to future architectures with pages larger than current ones (up to 64 KB), The actual amount of address space allocated can be less than 64 KB. but must be a multiple of the page size. The next two give a process the ability to hardwire pages in memory so they will not be paged out and (o undo this property. A real-time program might need this ability, tor example. A limit is enforced by the operating system to prevent processes from getting too greedy. Actually, the pages can be removed from memory, but only if the entire

MEMORY MANAGEMENT [win32APIfunction description rvirtualAlloc) Reserve or commit a region _ __ _ Release or decommit a region _ __ Change the read/write/execulg protection ona region__ Inquire about the status ot a region Make a region memory resident (i.e , disable paging tor it) I Make a region pageable in the usual way Create a file mapping ob]ect and (optionally) assign it a name ..._, ,,,, , Map (part of) a file into the address space UnmapViewOfFiie i Remove a mapped file from the address space OpenFileMapping ; Open a previousiy_created file mapping object process is swapped out. When it is brought back, all the locked pages are reloaded before any thread can start running again. Although not shown in Fig. 11 -25, Windows 2000 also has API functions to allow a process to access the virtual memory of a different process over which it has been given control (i.e., for

which it has a handle). The last four API functions listed are for managing me mory-mapped files To map a file, a file mapping object (see Fig H-10) must first be created, with CreateFileMapping. This lunction returns a handle to the file mapping object and optionally enters a name for it into the file system so anoiher process can use u The next two functions map and unmap files, respectively. The last one can be used by a process to map in a file currentl> also mapped in by a different process. In this way, two or more processes can share regions of their address spaces This technique allows them to write in limited regions of each other's memory 11.5.3 Implementation of Memory Management Windows 2000 supports a single linear 4-GB demand-paged address space per process. Segmentation is not supported in any form. Theoretically, page sizes can be any power of two up to 64 KB. On the Pentium they are fixed at 4 KB; on the Itanium they can be 8 KB or |6 KB In addition, the operating s7>-tem itself can use 4-MB pages to reduce page table space consumed. Unlike the scheduler, which selects individual threads lo run and doe.s not care much about processes, the memory manager deals entirely with processes and does not care much about threads After all, processes, not threads, own thi.- address space and that is what the memory manager deals with When a region of virtual address space is allocated, as four of them have been for process A in

818 CASE CHAI' Ha \\~24 the rnemor> manager creates a \AD (Virtual Address Descriptor) fen Jt. listing Ihe range of addresses mapped, [he backing store file and offset when' it is mapped, and the protection code. When Ihe first page is touched, the directoiy of page tables is, created and a pomier to it inserted is in the VAD In fact, an "address space- is completely defined by the list of its VADs. This scheme supports sparse address spaces because unused areas between the mapped regions Page Fault Handling Windows 2000 docs not use any form oi prepagmg. When a process starts, none of its pages- are in memory. All of them arc brought in dynamically as page faults occurs On each page fault, d trap to the kernel (in the sense of Fig. 1 [-7) i>ccurs The kernel builds a machine-independent descriptor telling what hap- happened and passes this to ihe memory manager part ot the executive. The memory manager then checks it for validity If the faulted page tails within a commitlcd or reserved region. \\ looks up the address in the list of VADs. finds (Or creates) the page table, and looks up the relevant entry The page table entries are different tor different architectures For ihe Pen- Pentium, the entry for a

mapped page is shown in Fig.] 1-26 Unmapped pages also have entries, but their formal is somewhat different. For example, tor an unmapped page that must be zeroed before it may be used, (hat fact is noted in the page table. G Page is global to ail processes L Large D-MB) page D Page is dirty A- Page has besn accessed C: Caching enabled/disabled Figure 11-26. A passi uble ui Wl Write through <nn caching) U Page is accessible in user mode W Writing to the page permitted V Vaiid page table entry d niJpped page on the Pcnlium The most important bits u, the page table entry tor purposes of the paging a rithm are the A and D bits. They are ted by ihe hardware and keep trad whether the page has been referenced or wntten o,i. respectively, since the time they were cleared. Page faults come in five categories 1 The page referenced is nc 2 A protection violation occ 3 A shared page has1 been v litted

SEC. It 5 MEMORY MANAGEMENT 819 4 The stack needs to grow 5. The page referenced ,s committed but not currently mapped in. The first and second cases are fatal error', from winch inert.- is no recovery toi the faulting process. The third case has the same symptoms as the second one [an attempt to write to a read-only page), bui the treatment is different The solution This is how copy-on-wrtie works ill a shared page is rturked writable in all processes using tt. it is noi copy-on-wnte and no fault otcuis when writing to it) The fourth case requires allocating a new page frame and mapping it in. Haw- ever, the security rules require that the page contain only Os. to prevent the proc-process fuim snooping on (lie previous owner of the page Thus a page of Os must be found, or if one is not .ivailable. another page frame must be allocated and zeroed on the spot Finally, the fifth case is a normal page fault The page is located and mapped in. The actual mechanics ol getting and mapping pages is fairly standard, so we will not discuss this issue. The only noteworthy feature is that Windows 2000 does not read in isolated pages from the disk. Instead, it reads m runs of uonsecu- live pages, usually about 1-8 pages, in an attempt to minimize the number ot disk transfers. The run size is larger for code pages than tor dara pages The Page Replacement Algorithm P<ige replacement works like this The system makes a serious attempt to maintain a substantial number of free pages m memory sti that when a page fault occurs, a free page can be claimed on Ihe spot, without the need lo first write some other page to disk. As a consequence of this strategy, most page faults can he satisfied with at most one disk operation (reading in the page), rather than sometimes two (writing back a dirty page and then reading in the needed page). Of course, the

pages on the free list have to come from somewhere, so the real page replacement algorithm Ls how pages get taken away from processes and put on the liee list (actually, there are four free lists, but for the moment it is simplest to think of there being just one; we will come to the details latei). Let us now take a look at how Windows 2000 frees pages To Man with, the entire paging system makes heavy use of the working set concept, bach process Uiot each thread) has a working set This set consists of the mapped-m pages that are in memory and can be thus" referenced without a page fault- The size and composi- composition of the working set fluctuates as the process' threads ntn. ot couise. Each process' working set is described h\, twt, paramefers the minimum size anu the maximum si^e. These are not hard bounds so a process may riave fewer pages in memory than its minimum or (under certain eircumstances) more than its maximum Every process starts with the same minimum and maximum, but these

CASE STUDY 2 WINDOWS 20iX ■r time. The default miiia CHAP minimum is in the range 2(M0 inge 45-345, depending on the t and the default initial maximum i.s in ihe range 45-345, depending on the mtul amount ot RAM. The system administrator can change these defaults, however It a page fault occurs and the working sel is smaller man ihe minimum, the page is added On ihe other hand, it a page fault occurs and the working set is larger than the maximum, a page is evicted from ihe working set (hut not trom rnemory) to make room for the new page. This algorithm means rhat Windows 2000 u ging . H nplt. it il ob not), the system time, ihe aUonth il algorithm, to prevent one process trom hurting others by hog- r. the system does try to tune itself lo some extent. For increase the size ot its maximum working set. so that over a nnx of local and global. There 15 an absolute limit on the working sei size, however' even if there is onl\ one process running, it may not lake the last 512 pages, lo leave Some s>lack tor new processes So far. so good, bui Hie story is not over jet. OnCc a second, a dedicated ker- kernel daemon thread, me balance set manager, checks to see if there are enough free pages. If there are nof enough, it starts the working sel manager thread to examine the working sets and recover more pages The working set manager first determines the order to examine the processes in Large processes lhai havt.- been idle for a long time arc considered before small aclive processes and the fore- foreground process is considered last. The working set manager then starts inspecting processes in ihe chosen ordei If a process' working set is currently less than its OHiumum

or if h.is incurred more than a certain number of page faults since the last inspection, it is passed over. Otherwise, one or more pages are removed The target number ot pages to remove is a complicated function ot the total RAM size, how tight memory is. how the current working set size compares to ihe process' minimum and max- maximum, and other parameters- All the pages are examined m mm On a uniprocessor, it ,1 page's reference bit is cle.li, a counter associated with the page is incremented. If ihe reference bii is set. the counter is set to zero After the scan, the pages with the highest counters art rem ned from the vvoikim; set. The thread continues examining procts es until u his recovered enough pages, then it stops. It a complete pass through all processes still has not recovered enough pages, it makes another pass tnmn in" mere aggressively, even reducing working sets below lheir minimum if lect ss irv On a multiprocessor, looking at the reference bit does not work because although the current CPU may not have touched the page recently, some other one may have. Examining another CPU's reference hits is too expensive to do Con- sequently. the reference bii is not examined and the oldest pages are removed. It should be noted that for page replacement purposes, the operating system itself is regarded as a process It owns pages and also has a woiking set. Thi1. working sel can be trimmed. However, parts ot the code and the nonpaged pool are locked in memory and cannot be paged out under any circumstances

SEC. IJ 5 MEMORY MANAGEMENT »21 Physical Memory Management Above we mentioned that there were iictu,ill> four free list*. Now it is time to see what all ot them are for Even, page- m memory is either m one or more work- working sets or on exactly one of these tour lists which are illustrated in Fig 11-27. The standby (clean) and modified (dirtv) lisis hold page* that have recently been pio< .sing them The difference between them is that li the i page-, rty page- do not have an up-to-dalu- copy on disk The fiee Iisl consists- or clean pages that arc no longer associated with any procc-sS The pages on the 7eroed page list are not associated with any process and are also filled with zeros. A fifth list holds any physically detective RAM pages that m.iy exist to make sure lhai they are not used for anything Figure 1-27. The Pages are moved between the working sets and the various lists by the work- working set manager and other kernel daemon threads Let us examine the transitions. When the working set manager removes a page trom d working set. the page goes on ihe bottom of the standby or modified list, depending on its Slate ot

cleanliness. This transition is shown as A). Pages on both lists are still valid pages, so it a page faull occurs and one of these pagcs is needed it is removed from the list and faulted back inio the working set withoui any disk I/O B) When a process exits, its nonshared pages cannot be faulted baek io it, so they go on the free list C) These pages are no longei associated with any process Other transitions are caused by other daemon threads, livery 4 seconds the swapper thread runs and looks for processes all of whose threads have been tdie

822 CASE STUDY 2 WINDOWS 2CXX) CHAP. 11 for a certain number of seconds It it finds any such processes, itieir kernel stacks are unpinned and their pages an? moved to the standby or modified lists. aUo shown as A) Two other daemon (dreads, ilie mapped page writer and the modified page writer, wake up periodically to -see if there are enough clean page;,. If there are noi, they take pages from the top ol the modified list, write them back to disk, and then move them 10 the siandby list D). The former handles write* to mapped files and the latter handles writes to the paging files The result of these writes is to transform dirty pages, int(> clean pages. The reason for having two threads is th.u .i mapped file might have to grow as a result ot the write, and growing it requires access lo on-disk data structures to allocate a tree disk hlock If rhere is no room in memory to bring them in when a page has to be written, a deadlock could result The other thread can solve ihe problem by writing out pages to a paging file, which never grows Nobody ever said Windows 2000 was simple The other transitions in Fig. 11-27 are as follows. It a process unmaps a page, the page is no longer associated with a process and can go on the tree list (>). exeepi for the case that it is shared. When a page fault requires a page frame [ci hold (he page about to be read in. the page frame is taken rrom the tree list F) if possible It does not matter that the page may still contain confidential informa- information because it is about id be overwritten in its, entirety The situation is different when a stack grows. In that case, an empty page frame is needed and the security rules require ihe page to contain all zeros. For this reason, another kernel daemon thread, the zero page thread, runs at the lowest priority (see hg 11 -19). erasing pages that are on the free list and putting them on the zeroed page list G] Whenever the CPU is idle and there are free pages, ihey might as well be zeroed since a zeroed page is potentially more useful than a free page The existence of all these lists leads to some subtle policy choices For exam- example, suppose that a page has to be

brought in from disk and the free list is empty. The system is now forced to choose between taking a clean page from the standby Itst (which might otherwise have been faulted back in later) or an empty page from the zeroed page list (throwing away the work done in zeroing it) Which is better? If the CPU is idle a lot and the zero page thread gets to run often, taking a zeroed page is better because there is no shortage ot rhem However, if the CPU is always busy and the disk is mostly idle, it is better to take a page trotn the standby iisf to avoid the CPU co.st of having to zero another page later if a stack grows. Another puzzle. How aggressively should the daemons move pages from the modif]ed list to the standby list? Having clean pages around is better than having dirty pages firound (since tnev can De reused instanlly'X out an aggressive (.leaning policy means more disk I/O and there is some chance lhat a newly-cleaned page may be faulted back into a working set and dirtied again.

SEC 11 5 MEMORY MANAGEMENT e kinds 823 .mplex s through < complex, thai the thing somewhere In general. Windows 2000 reso.v^s i.«.-^ *.,.r heuristics, guesswork, historical prec-edent, rules of thumb, controlled parameter seitmgs. Furthermore, the code designers are loathe to touch parts of it for tear of breakin else in the sysiem that nobody really understand** any more To Keep track of all the pages and all the lists. Windows maintains a page frame database with as many enines as there are RAM pages, as" shown in Fig 11-28. This table is indexed by phyMcal page frame number. The entries are hxecl Icngih, but different formats are used for different kinds ot entries (e.g.. valid versus invalid). Valid entries maintain the page's state and a count of how many page tables point to the page, so the system L-an tell when the page is no longer in use. Pages ihat are in a working set tell whieh one. There is ako a pointei to the page table pointing 10 the page, if any (shared pages are handled specially). ,i link to the next page on the list (if any), and \arious, other fields and flags, such as read in piogress, write in progress, etc. Zeroed Active Zeroed Figure 11-28. Some ol ihe major fields in i|ic p.ige Ii page mor\ rnanagei algorithms and heuristics It , but there are aKo man\ knobs that admimstrat. formance A numbei ot these knob and the ... „„„,. ,.t ,._„..„ ing tools in the \ariou tool kils mentioned earlier Probdblv the mo t important thing to remember here i [hat memory management in real in just one simple paging algorithm like UoUc (mpts to be largelj se|t mi t*eak to iffect s\sl iated counters Lan he \it a lot more ■ aging

824 CASF STUDY 2 WINDOWS 2000 t'HAP II 11.6 INPUT/OUTPUT IN WINDOWS 2000 The goal of the Windows 2000 I/O system k to provide a framework fur effi- efficiently handling a very wide variety of I/O devices. Current input devices include various kinds of kevboards mice touch pads, loystf ck^, scanners, sti U cameras, television cameras, bar code readers, microphones, and laboratory rats Current outpui devices include monitors, primers, ploiters, beamers, CD-recorders, and sound cards. Storage devices include floppy disks. IDE and SCSI hard disks. CD-ROMs, DVDs. Zip drives, and iape drives Finally, other devices include clocks, network\ telephones, and camcorders. No doubt many new I/O devices will be invented in the years to come, so Window 2000 has been designed with a general framework to vvhich new devices can easily be attached. In the following sections we will examine some ol the issues relating to I/O. 11.6.1 Fundamental Concepts The I/O manager is on intimate terms with the plug-<md-play manager. The basic idea behind plug and p|ay is that ol an enumerable bus Many buses, includ- including PC Card, PCI. USB, IEEE 1394, and SCSI, have been designed so that the plug-and-play manager can send a request to each slot and ask the device (here to identify itself. Having discovered what is out there, the p|ug-and-p|ay manager allocates hardware resources, such as interrupt levels, locates the appropriate drivers, and loads them into memory. As each one [s loaded, a driver object is created for it- For some buses, such as SCSI, enumeration happens only at boot time, but for other buses. $ucli as USB and IEEE 1304. it can happen at any moment, requiring close contact between ihe plug-and-play manager, ihe bus driver (which actually does the enumeration), and the I/O manager. The I/O manager is also closely connected with the power manager. The power manager can put the computer into any of six states, roughly described as: 1. Fully operational. 2. Sleep-1: CPU power reduced. RAM and cache on. instant wake-up 3 Sleep-2' CPU and RAM on; CPU cache ofi; continue from currem PC 4. Sleep-3: CPU and cache off; RAM on, restart from lixed address 5. Hibernate: CPU, cache, and RAM oif; restart from saved disk file 6. Off: Everything off; full reboot required. I/O devices can also be in various power states. Turning them on and oft t,s han- handled by the power manager and I/O manager together. Note that states 2 through 6 are only used when the CPU has been idle for a shorter or longer time interval

 Somewhat surprisingly, all the file systems are technically I/O drivers. Requests tor data

blocks from user processes are initially sent 10 the cache managei. It the cache manager cannot satisfy the request from the cache, it has the I/O manager call the proper tile system driver to go get the block it needs from disk An interesting feature ot Windows 2000 is Us support for dynamic disks These disks may span multiple partitions and even multiple disks and may be reconfigured on the fly. without even having to reboot- In this way. logical \olumes arc no longer constrained to a single partition or even a single disk so that a single file system may span multiple drives in a transparent way Another interesting aspect of Windows 2000 is its support for asynchronous I/O II is possible for a thread to start an I/O operation and then continue execut- executing in parallel with the I/O This feature is especially important on servers. There are various ways the thread can find out that the I/O has completed. One is to specify an event object at the-lime the call is made and men wait on it eventually Another is to specify a queue to which a completion event will be posted by the system when the I/O is done. A third is to provide a callback procedure that the system calls when the I/O has completed 11.6.2 Input/Output API Calls Windows 2000 has over 100 separate APIs for a wide variety of I/O devices, including mice, sound cards,, telephones, tape drives,, etc. Probably the mosl important is the graphics system, for which there are thousands of Win32 API calls. |n Sec. 5 7.3 we began our disciibston of the Window graphical system. Here we will continue, mentioning a few ot the Win32 API categories, each ol which has many calls. A brief summary of the categories arc given in Fig 11-29. As- we mentioned in Chap. 5. multiple |500-page books have been written on [he graphics portion of the Win32 API Win32 calls exist to Create, destroy, and manage windows Windows have a vast number of styles and options that can be specified, including titles, borders, colors', sizes, and scroll liars. Windows can be fixed or movable, of constant si/e or resizable. Their properties can be queried and messages can be sent to them. Many windows contain menus, so there are Win32 calls for creating and deleting menus and menu bars. Dynamic menus can be popped up and removed Menu items can be highlighted, dimmed out, or cascaded. Dialog boxes are popped up to inform the user of some event or ask a ques- question. They may contain buttons, sliders, or text fields to be filled in. Sounds cm also be associated with dialog boxes, for example for warning messages There are hundreds of drawing and painting functions available, ranging from .selling a .single pixel to doing complex region clipping operations. Many calls' are provided for drawing lines and closed geometric figures of various kinds, with

detailed control over textures, colors, widths, and many other attributes.

r\SIiSTL'DY2 WINDOWS 2000 CMA APIgrouP t Description Window management | Create destroy, and manage windows destroy, and append to menus and menu bars Pop up a dialog box and collect jnlor matron Display points, Lnes. and geometric figures Display lexi in some lont, size, and color Placement of bitmaps and icons on the screen Colors and palettes I Manage the set of colors available The clipboard I Pass information from one application to anothe Input __ _ Get information from the mouse and keyboard Another group ot calls relates to displaying text. Actually, the text display tall. TextOut, is straightforward. It in the management of the color, point sizes, typefaces, character widths, glyphs, kerning, and other typesetting details where the complexity comes in. Fortunately, the rasterization ol text (conversion to bit-bitmaps) is generally done automatically. Bitmaps are small rectangular blocks of pixels that tan be placed en the screen using the BitBIt Win32 call They are i.sed for icons .ind occasionally text. Various calls arc provided for creating, destroying, and managing icon objects. Many displays use a color mode with only 256 or 65.536 of the 214 possible colors in order to represent each pixel with only 1 or 2 bytes, respectively. In these cases a color palette is needed to determine which 256 or 65,536 colors are available. The calls in this group create, destroy, and manage palettes, select ihe nearest available color 10 a given color, and try to make colors on the screen match colors on color printers Many Windows 2000 programs allow tlte user to select some data (eg., a block of text, pan of a drawing, a set of cells in a spreadsheet), pui it on the clip- clipboard, and allow it io be pasted into another application The clipboard is, gen- generally used for (his transmission. Many clipboard formats are defined, including text, bitmaps, objects, and metafiles. The latter are sets cd Win32 calls that wheii executed draw something, allowing arbitrary drawings to be cut and pasted. This group of calls put* things on the clipboard, takes things off the clipboard, and gen- generally manages it Finally, we come to input. There are no Win32 calls for GUI applications lor reading input from the keyboard because GUI applications are event driven. The main program consists ot'a big loop getting input messages. When the usei types something interesting, a message is sent to the program telling it what jusl came in. On the other hand, there are calls relating to the mousd such as reading its U, V) position and the state of its buttons. Some of the input calls are actually output

SF,C 11 6 INPUT/OUTPUT IN WINDOWS 2000 »27 cills. though, such as selecting a mouse cursor icon and moving u around the screen (basically, this is output to the screen) For nonCjUl applications, it is pos- possible to kmiI Iron, the keyboard. 11.6.3 Implementation of I/O Wt^ could co on iiHirc-or-less indefinirely about the Win32 graphics calls, bui now it is time to look at how the I/O maiuger implements graphics and othui I/O functions The main function ol the I/O manager is to create a framework ,n which dtilerent I/O dc-viccs can operate The basic structure of the framework is a set ot deviLC-mdependcnt procedure* tor certain aspects of I/O plus a set of loaded device drivers for communicating w ifh the devices- 11.6.4 Device Drivers To make sure that device drivers work well with the n-st of Windows 2000. Microsoft has defined a Windows Driver Model that device drivers are expected to conform with. Furthermore, it also has provided a tool kit that is designed to help driver writers produce conformant drivers In this section we will briefly examine this model. Conformant drivers must meet all ot the Jollowmg requnc-ments as well as some others. 1. Handle incoming I/O requests- which arrive m ;i standard format 2. Be as object based as the rest ot Windows 2000. 3. Allow p|ug-and-play devices to be dynamically added or removed 4. Permit power management, where applicable. 5. Be configurable in terms ot resource usage 6. Be reentrant for use on multiprocessors. 7. Be portable across Windows 98 a,id Windows 2000 I/O Requests are passed to drivers m the form ot a standardized packet called an IRP (I/O Request Packet) Conformant driven; must be able to handle them. Drivers must be object based in the setise of supporting a specific list ot methods that the rest of the system can call. They must also correctly deal with othei Win- Windows 2000 objects when given an object handle to deal with Conformant drivers must fully support plug and play, which means thai it a device managed by the driver is suddenly added or removed from the system. Ihe driver must be prepared to accept thi.s infomiauon and act on n, even in the ease that the device currently being accessed is suddenly removed. Power manage- management must also be supported for devices for which this i.s relevant. For example, if the system decides it is now time to go into a low-power hibernation mode, all

 devices thai are capable of doing this must do so to save- energy. They must also wake up when told to do so. Drivers must be configurable, which means not having any built-in assump-assumptions about which interrupt lines or I/O ports certain devices use Tor

example, the primer port on the IBM PC ;ind its successors has been at address 0x378 tor more than 20 years and it is unlikely to change now. Bui a printer driver that has this addiess hard coded into it is not conformant Being multiprocessor safe is also a requirement as Windows 2t)(X) was designed for use on multiprocessors. Concretely, this requirement means while a driver is actively running and processing one request on behalf of one CPU. a second request may come in on behalf of a different CPU. 'I he second CPU may beein executing the dmer code simultaneously witli the first one. Tbe driver must function correctly even when being executed concununtly by ty.(> or more CPUs, which implies that all sensitive data slrucmics may only he accessed from the current one is finished !.s not permitted. Finally, eonformani drivers must work not only an Windows 2000 hut also on Windows 98 It may be necessary to recompile the driver on each system how- however, and use of C preprocessor commands lo isolate platform tie pendencies is permitted. In UNIX, drivers are located by using their major device numbers Windows. 2000 uses a diffeient scheme Ai booi time, or when a new hot pluggable plug-and-play device is attached to the computei Windows 200A automaticall\ detects it and calls the plug-and-play manager The manager queries the device to find oui what the manufaciuicr and model number arc F.uuipped with this knowledge, it looks on the hard disk in a certain directory' to ic-e if it has the driver If n doc not. it displays a dialog box asking the user to insert a floppy disk or CD-ROM with the driver. Once the driver is located, it is loaded into memory Each driver must supply a set of procedures that can be called to set us ser- services. The first one, called Dm'i'rEntn; initialises the dnvci It is called just after the driver is loaded. It may create uibles and data structures, but must not touch the device yet. It atso fills in «oine of the fields of the driver object created by the I/O manager when the driver was loyded The fields in ihc drivei object include pointers to a|l the other procedure* that drivers must supply. In addition, for each device controlled by the driver (e.g . each IDE disk com rolled by the Iph disk driver), a device object is1 created and initialized to point to the driver ob|ecl. These driver objects aru entered into a special directory, \'? Given a device object, the driver objeel can be located easily and hence its methods can be called. A second required procedure is AddDeuce. which is called once |hy the plug-and-play manager) for each device to be added Once this has been accom- accomplished, the driver is called with the first 1RP, which sets up the interrupt vector and actually initializes the hardware. Other procedures that drivers must contain

SEC. 11.6 INPUT/OUTPUT IN WINDOWS 2000 829 are the interrupt service procedure, various timer management procedures, a fast I/O path, DMA control, a way to cancel currently executing requests, and many more. All in all, Windows 2000 drivers are so complex that multiple books have been written about them (Cant, 1999; Oney. 1999; and Viscarola and Mason, 1999). A driver in Windows 2000 may do all the work by itself, as the printer driver does in Fig. 11-30 (just as an example). On the other hand, drivers may also be stacked, which means that a request may pass through a sequence of drivers, each doing part of the work. Two stacked drivers are also illustrated in Fig. 11-30. user process 1 1 (Monolithic 1 Controller V 1 1 I /"useTN f program * V Win32 J \ Rest Of windows FuncSon || Bus f \ ) I [1 Hardware abstraction layer 11 Controller 1 ♦ II Filter t Function t Bus } Controller 1 ■ Driver stack Figure 11-30. Windows 2000 allows drivers to be slackcd. One common use for stacked drivers is to separate out the bus management from the functional work of actually controlling the device. Bus management on the PCI bus is quite complicated on account of many kinds of modes and bus tran- transactions, and by separating this work from the device-specific part, driver writers are freed from learning how to control the bus They can just use the standard bus driver in their stack. Similarly, USB and SCSI drivers have a device-specific part and a generic part, with common drivers used for the generic part.

830 CASE STUDY 2-WINDOWS 2000 CHAP II Another use ot stacking drivers is to be able to insert filter drivers inio the stack. A filler driver performs some transformation on the data on the way up or down- For example, a filter driver could compress data on ihe way to the disk or encrypt data on the way to the network Putting the filler here means that neilher the application prograrn nor the true device driver have to be aware of it and it works automatically for all data going to (or coming from) the dev]ce. 11.7 THE WINDOWS 2000 FILE SYSTEM Windows 2000 supports several lile systems, the most important ot which are FAT-16, FAT-32. and NTFS (NT File System) FAT-16 is the old MS-DOS file system. It uses 16-bit disk addresses, which limits it to disk partitions no larger than 2 GB. FAT-32 uses 32-bit disk addresses and supports disk partitions up to 2 TB. NTFS is a new file system developed specifically for Windows NT and car- carried over to Windows 2000. li uses 64-bn disk addresses and can (theoretically) .support disk partitions up to 2M bytes, although other considerations limit n to smaller

size<. Windows 2000 also supports read-only file systems for CD-ROMs and DVDs. It is possible (even common) to have the same running system have access to multiple file system iypes available at the same time In this chapter we will treat the NTFS file system because it is a modern tile sysiem unencumbered by the need to be fully compatible with the MS-DOS tile system, which was based on the CP/M die system designed for 8-inch floppy disks more than 20 years ago Times have changed and 8-inch floppy disks are not quite state of the art any more. Neither are iheir file systems Also, NTFS differs both in user interface and implementation in a number of w.iys rrom [he UNIX file system, which makes it a good serond example to study. NTFS is ii large and complex system and space limitations prevenl us from covering all of it* features, but the material presented below should give a reasonable impression of 11.7.1 Fundamental Concepts Individual file names in NTFS are limited to 255 characters-, full paths .ire limned to 32,767 characters. File names are in Unicode, allowing people in coun- countries not using the Latin alphabet (e g.. Greece, Japan, India. Russia, and Israel) to write file names in their native language. For example. <>ite ,s a perfectly legal file name. NTFS fully supports case sensitive names (so/oo is different from Fo,> and FOO). Unfortunately, ihe Win32 API does not fully support case-sensitivity tor file names and noi at all tor directory names, so this advantage is lost to pro- programs restricted to using Win32 (e.g.. for Windows 98 compatibility).

SEC. 1I 7 THE WINDOWS 2000 FILE SYSTKM 831 An NTFS file is not juRl a linear sequence of bytes, as FAT-32 and UNIX files are. Instead, a file consists of multiple attributes, each of which is represented by a stream of bytes. Most files have a few short streams, such as the name of the file and its 64-bit object ID. plus one long (unnamed) stream with the data. How- However, a file can also have two or more (long) data streams as well. Each stream has a name consisting ot the file name, a colon, and the stream name, as in pry stream I. Each stream has its own size and is lockable independently of all the other streams,. The idea of multiple streams in a file was borrowed from the Apple Macintosh, m which files have two streamt;, the data fork and the resource tork. This concept was incorporated into NTFS to allow an NTFS server be able File streams can be used for purposes olher than Macintosh compaiibiltty. For example, a photo ediling program could use the unnamed stream for the main image and a named stream for a small thumbnail version. This scheme is simpler than the traditional way of putting them in the same file

one after another. Another use of streams is in word processing These programs often make two versions of a document, a temporary one for use during ediling and a final one when the user is done. By making the temporary one a named stream and the final one the unnamed stream, both versions automatically share a file name, security information, timestamps. etc. with no extra work The maximum stream length is 2M bytes To get some idea of how big a 2h4-byte stream is. imagine that the stream were written out in binary, with each of the 0s and Is in each byte occupying 1 mm of space The 2'l7-mm listing would be 15 light-years long, reaching far beyond the solar system, to Alpha Ccn- tauri and back. File pointers are used to keep track ot where a process is in each stream, and these are 64 bits wide lo handle the maximum length stream, which is about 18.4 exabytes The Win32 API function calls for file <md directory manipulation are roughly similar to their UNIX counterparts, except most have more parameters .ind the security model is different. Opening a file returns a handle, which is then Used tor reading and writing the lile. For graphical applications, no file handles are prede- predefined. Standard input, standard output, and standard error have to be acquired explicitly if needed; in Lonsole mode they aic preopened. however, Win32 also has a number of additional calls not present in UNIX 11.7.2 File System API Calls in Windows 2000 The principal Win32 API functions for file management are listed in Fig. 11- 31. There are actually many more, but these give a reasonable first impression ot the basic ones. Let us now examine these calls briefly. CreateFtle can be used to create a new file and return a handle to it This API function must a|so be used to open existing files as there is no FileOpen API function. We have not listed the

832 CASE STUDY 2: WINDOWS 2000 Description _J Create a We or open an existing file, return a handle i Set the file pointer to a specific place in the file Return the file properties _ _ _ Lock a region of the file to provide mutual exclus fcntl j Unlock a previously locked region of the file I Figure 11-31. The principal Win32 API functiiins for file I/O The second column gives the nearesl UNIX equivalent parameters tor the API functions because they are so voluminous. As an example. CreateFile has seven parameters, which art: roughly summarized as follows:]. A pointer to the name of the file to create or open 2 Flags telling whether the file can he read, written, or both 3. Flags telling whether multiple processes can open (he file at once 4. A pointer to the security descriptor, telling who can access the file 5. Flags telling what to do if

the file enists/does not exist 6. Flags dealing with attributes such as archiving, compression, etc 7. The handle of a file whose attributes should be cloned for the new fi|e The next six API functions in Fig. 11-31 are fairly similar to the correspond- corresponding UNIX system calls. The last two allow a region of a file to be locked and unlocked to permit a process to get guaranteed mutual exclusion to it Using ihese API functions, it is possible to write a procedure to copy a file, analogous to the UNIX version of Fig 6-5 Such a code fragment (without any error checking) is shown in Fig. I 1-32. It has been designed to mimic our UNIX version. In practice, one would not have to program a copy file program since CopyFile is an API function (which executes something close to this program as a library procedure). Windows 2000 NTFS is a hierarchical file system, similar to the UNIX file sil inherited from MS-DOS. There is a concept of a currenl working directory and path names can be relative or absolute. Hard and symbolic links are supported.

THE WINDOWS 2000 Fll>: SYSTEM I* Open files for input and output */ mhandle = CreateRefdata", GENERIC .READ, 0, NULL. OPEN EXISTING, 0, NULL), outhandle = CreateFile("newf", GENERIC .WRITE, 0. NULL, CREATE ALWAYS, FILE .ATTRIBUTE NORMAL, NULL); /* Copy the file •/ do{ a = ReadFile{mhandle. buffer, 8UF_SIZE, &count, NULL), if (s && count > 0] WnteFile(outhandle, buffer, count. Socnt, NULL];) while (s > 0 && counf > 0); /* Close the files -/ CloseHandiefinhandle); CloseHandle(outhandle). ragmeni for copying j fife umi e Windows 2000 the former implemenied by having multiple directory entries, as in UNIX, and tlie latter implemented using reparse points (discussed later in this chapter). In addi- addition, compression, encryption, and fault tolerance are also .supported Thoe features and their implementations will be discussed later in this chapter. The major directory management API functions are given in Fig I I -3\ again along wi(b their nearest UNIX equivalents. The functions should be s,elf explana- explanatory. Description I lew directory __ _ I n em pry di rectory Initialize to start reading the entries in a directory , Read the next directory entry > ^oveFile_ SetCt ile | rename Move a file from one directory to another rentDiredocy ' chdlr \ Change the current working directory Figure 11-33, The pnnupal ' m32 API functio iresl UNIXcqun 11.7.3 Implementation of the Windows 2000 File System NTFS is a highly complex and sophiuicaied file system. It was designed from scratch, rather than being an attempt to improve the old MS-DOS tile s>stem Below we will examine a number of its features, starting with

its structure, then moving on to file name lookup, file compression, and file encryption.

g34 CASE STUDY 2 WINDOWS 2000 CHAP I i Pile System Structure Each NTFS volume (e.g., disk partition) contains files, directories, bitmaps, and other data structures. Each volume is organized as a linear sequence of blocks (clusters in Microsoft's terminology), with [he block size being ftxetl for each volume and ranging from 512 bytes to 64 KB. depending on the volume size- Most NTFS disks use 4-KB blocks as a compromise between large blocks (for efficient transfers) and small blocks (for low internal fragmentaiion) Blocks are referred to by their offset from the start ot the volume using 64-bit numbers. The main data structure in each volume is the MFT (Master File Table). which i* a linear sequence ot fixed-size |-KB records. Each MFT record des- cribes one file or directory ll contains the file's aitnbutes, such as Us name and tunestamps. and the list of disk addresses where its blocks are located. If a file is extremely large, if is sometimes necessary to use two or more MFT records to contain the [ist of all the blocks, in which case the first MFT record, called the base record, points to the other MFT records This overflow scheme dates back to CP/M. where each directory entry was called an extent. A bnmap keeps track of which MFT entries are free. The MFT is itself a file and as such can be placed anywheie within the volume, thus eliminating the problem with defective sectors in the first track. Furthermore, the file can grow as needed, up to a maximum size of 24* records. The MFT is shown in F,g. 11-34. Each MFT record consists ot a sequence of (attribute header, value) pairs. Each attribute begins with a header telling which attribute this is and how long the value is because some aitribute values are vari- variable length, such as the file name and ihe data. If the attribute value is short enough to fit in the MI-T record, it is placed there If u is too long, tt is placed elsewhere on the disk and a pointer to it is placed in the MFf record. The first 16 MFT records are reserved for NTFS metadata files, as shown in Fig. 11-34. Each of the records describes a normal file that has attributes and data blocks, just like any other file. Each of these files has a name that begins with d dollar sign to indicate that it is a metadata file. The first record describes ihe MFT file itself. In particular, it tells where the blocks of the MFT file are located so the system can find the MFT file. Clearly, Windows 2000 needs a wav to find the first block Of the MFT file in order to find the rest of the tile system informa- information. The way it finds the first block of the MFT file is to look in the

hoot block, where its address is installed at sysiem installation time Record J is a duplicate of the early part of the MFT fi|e. This Hi formation is so precious that having a second copy can be critical in the event one of the first blocks of the MFT ever goes bad. Record 2 is the log file. When structural chan- changes are made to the file system, such as adding a new directory or removing an existing one. the action is logged here before it is performed, m order to increase the chance of correct recovery in the event of a failure during the operation. Changes to file attributes are also logged here- In fact, the only changes not

THE WINDOWS 2000 FILB SYSTEM First user file $B0Ql $6i(map Bootstrap loader Bitmap o[blocks u; Root directory Figure 11-34. The NTFS rr ITIe lahle logged here are changes to user data. Record 3 contains information about the volume, such as its size, label, and version. As mentioned above, each MFT record contains a sequence of (attribute header, value) pairs. The SAtlrDeffWe is where the attributes are defined. Infor- Information about this file is in MFT record 4. Next comes the rooi directory, which itself is a file and can grow to arbitrary length It is described by MFT record 5. Free space on the volume is kept track of with a bitmap. The bitmap is itself a file and its attributes and disk addresses are given in MFT record 6. The ne\i MFT record points to the bootstrap loader file. Record 8 is used to link all the bad blocks together to make sure they never occur in a file. Record 9 contains the security information. Record |0 is used for case mapping For the Latin letters A-Z case mapping is obvious (at least for people who speak Laun). Case map- mapping for other languages, such as Greek. Armenian, or Georgian (the country, not the state), is less obvious to Latin speakers, so this file tells how to do ii. Finally, record 11 is a directory containing miscellaneous files for things like dtsk quotas, object identifiers, reparse pomts. and so on The last 4 MFT records are reserved for future use. Each MFT record consists of a record header followed by a sequence of (attri- (attribute header, value) pairs. The record header contains a magic number used for validity checking, a sequence number updated each time the record is reused for a new file, a count of references to the file, the actual number of bytes in the record

 used, the identifier (index, sequence number) of the base record (used only for extension records), and

some other miscellaneous fields. Following the record header conns the header of the first attribute, then the first attribute value, the second attribute header, the second attribute value, and so on NTr-S defines 1? attributes that can appear in MFT records. These are listed in Kg. 11-35. Each MFT record consists ot a sequence of attribute headers, each of which idennfies the attribute ii is heading and gives the length and location of the value field along with a variety of flags and other information. Usually, attri- attribute values follow their attnbule headers directly, but if a value is too long to fit in the MFT record, it may be put in a separate disk block. Such an attribute is said io be a nonresident attribute. The data attribute Ls an obvious candidate- Some attributes, such as the name, may be repeated, but all attributes must appear in a fixed order in the MFT record. The headers for resident attributes are 24 bytes long; those for nonresident attributes are longer because they contain infor- information aboul where to find the attribute on disk Attribute Standard information File name Security descriptor Attribute list Description Flag bits, timestamps, etc File name in Unicode, may be repeated lor MS-DOS n Obsolete Security information is now in $Extend$Sec Location of additional MFT records, if needed 64-bit file identifier unique to this volume Used for mounting and symbolic »nks Nameof this volume (used only in $Volume) ne version (used only in $VotumeJ Used lor directories Used for vecy large directories Used tor very large directories Controls logging to $LogFile Stream data, may be repeated Figure 11-35. The attributes used in MFT re The standard information field contains the fi|e The standard infomiation field contains the fi|e owner, security information, the timestamps needed by POSIX, the hard Jink count, the read-only and archive bits, eic. It is a fixed-length field and is always present. The file name is variable length in Unicode. In order to make files wtth nonMS-DOS names accessible to old 16-bit programs, files can also have an 8 + 1 MS-DOS name If the actual file name conforms to the MS-DOS 8 + 3 naming ru|e. a secondary MS-DOS name i«; not used In NT 4.0, security information could be put in an attribute, but in Windows 2000 it a‖ goes into a single file so that multiple files can share the same security

SEC. 11 7 THE WINDOWS 2000 FILE SYSTEM 837 descriptions The attribute list is needed in case Ihe attributes, do not fit in the MFT record. This attnbule Ihen tells where to find the extension records. Each entry id the list contain* a 48-biI index inlo the MFT telling where the extension record is and a 16-biI sequence number to allow verification that the extension record and

base records malch up. The object ID atlnbule gives the file a unique name. This is sometimes needed internally. The reparse point tells the procedure parsing the file name lo do somelhing special. This mechanism is used for mouniing and symbolic links. The two volume altribuies are only used for volume identiftcalion. The next Ihree altnbulcs deal with1 how direclories are implemented. Small ones are just lisls of files but large ones are implemented using B+ Irees. The logged ulility slream atinbuie is used by the encrypting file system. Finally, we come to the allribule Ihal everyone has been wailing ton Ihe data The slream name, if presenl, goes in this atlnbute header. Following the header is eilher a list of disk addresses lelling which blocks Ihe file contains, or for files of only a few hundred byles (and there are many of these), the file itself. Putting the actual file dala in Ihe MFT record is called an immediate file (Mullender and Tanenbaum, 1987). Of course, mosi of Ihe time the dala does nol fit in the MFT record, so this altribule is usually nonresidenl. Lei us now take a look al how NTFS keeps Irack of the location of nonresident atinbules, in particular data. The model for keeping Irack of disk blocks is that Ihey are assigned in runs of consecutive blocks, where possible, for efficiency reasons For example, if ihe firsl logical block of a file is placed in block 20 on the disk, ihen the system will Iry hard to place Ihe second logical block in block 21, the Ihird logical block in 22, and so on. One way to achieve these runs is lo allocate disk slorage several blocks al a lime, if possible. The blocks in a file are described by a sequence of records, each one describ-describing a sequence of logically contiguous blocks. For a file witb no holes in it. there will be only one sucb record. Files thai are wmten in order from beginning to end all belong in Ihis category. For a file with one hole in H (e.g., only blocks 0--49 and blocks 60-79 are defined), Iherc will be iwo records Such a file could be produced by wnling the first 50 blocks, then seeking forward to logical block 60 and wnling anolher 20 blocks. When a bole is read back, all the missing bytes arc zeros. Each record begins wilb a header giving Ibe offset of Ihe ftrsi block witbm ibe file. Next comes Ihe offsel of the firsl block not covered by ihe record. In the example above, the firsl record would bave a header of @. 50) and would provide the disk addresses for these 50 blocks. The second one would have a header of F0,80) and would provide Ihe disk addresses for these 20 blocks Each record header is followed by one or more pairs, each giving a disk address and run lenglh. The disk address is Ibe offset of the disk block from the start of its partition; the run lenglh is the number of blocks in the run. As many

CASE STUDY 2. WINDOWS 2(HK) irs as needed can be in (he run record le-block file is illustrated in Fig. 11-36 cd cm Blocks numbers 20-23 64-65 80-92 Figure 11-36. An Mt-T record (i>r a t hire-run nme-blotk tik- In this figure we have an MFT record tor a short file (short here means ihat al] the information about the file blocks fits i» one MFT record). It consists of the three runs of consecutive blocks on the disk. The first run is blocks 20-23, the second is blocks 64-65. and the third is blocks 80-82 Each of these runs is recorded in the MFT record as a (disk address, block count) pair. How many runs there are depends on how good a job the disk block allocator did in finding runs of consecutive blocks when the file was created. For a n-block tile, the number of runs can be anything from 1 up to and including n Several comments are worth making here First, there is no upper limit to the size of files that can be represented this way In the absence of address compres-compression, each pair requires two 64-bit numbers in ihe pair for a total of 16 bytes. However, a pair could lepresent 1 million or more consecutive disk blocks. In fact, a 20-MB file consisting of 20 separate runs of 1 million t-KB blocks each fits easily in one MFT record, whereas a 60-KB file scattered into 60 isolated blocks does not. Second, white the straightforward way of representing each pair takes 2x8 bytes, a compression method is available to reduce the size of the pairs below 16. Many disk addresses have multiple high-order ?ero-bytes. These can be omitted. The data header tells how many are omitted, that is. how many bytes are actually used per address. Oiher kinds of compression are also used. In practice, the pairs are often only 4 bytes. Our first example v.as easy: all the file information fit m one MFT record. What happens if the file is so targe or highly fragmented that the block informa-information does not fit m one MFT record The answer is simple: use two or more MFT records. In Fig! 11-37 we see a file whose base record is in MFT record !02. It

THE WINDOWS 200A RLE SYSTEM lias too many runs tor one MFT record, so it computes fiow many extension records it needs, say, two. and puts their indices in the base record. The rest oi the record is used loi I he first A data runs. - Second extension re Note lhat Fig. lt-37 contains some redundancy. In theory, it should not he necessary to specify the end of a sequence ot runs because this information can he calculated from the run pairs The reason lor "overspecifying" this information is to make seeking more efficient: to find the block at a given file offset, it is only necessary to examine the record headers,

not the run pairs. When all the space in record !02 has heen used up. storage of the runs contin- continues with MFT record 105. As many runs are packed in this record as fit When this record is also full, the rest of the runs go in MFT record 108. In this way many MFT records can be used to handle large fragmented tiles A problem arises if so many MFT records are needed that there is im room m the hase MFT to list all theii indices. There is also <i solution to this problem the list of extension MFT records is made nonresident (i e., stored on disk instead of in the base MFT record). Then it can grow as large as needed- An MFT entry for a small directory is shown in Fig I]-3S The record con- contains a number of directory entries, each ot which describes one file or directory Each entry has a fixed-length sfructure followed by a variable-length file name The fixed part contains the index of the MFf entry for the file, the length of the file name, and a variety of other fields und flags Looking li>r an entry in a direc- directory consists of examining all the file names in turn Large directories use a different format. Instead ot listing the file* linearly, a B+ tree is used to make alphabetical lookup possible and to make it easy to insert new names in the directory in the proper place File Name Lookup We now have enough information to see how file iser prugram wants to open a file, it typically makes CreateFileCC^mariaXweb.htm", ..) ime lookup o call like

CASb STUDY 2: WINDOWS 2000 leader cord \ i A directory entry contains the the length of the file name (hi and various fields and flags Figure tt-38. The MFT record f(>r er-level shared library, kernel32.dll. where \?.' This call goes to the pended to ihe file name gtvmg V"\C:\mana\web.ritm It is this name that is passed as a parameter to the system call NtFlleCreate. Then the operating system starts the search at the root of the object manager's name space (see Fig. 11-12). It then looks in the directory \'r> to find C:, which it will find. This file is a symbolic link to another part of the object manager's name space, the directory SDevice. The hnk typically ends at an object whose name is something like \Dev\ce\HarddiskVolumel This object corresponds to the first partition of the first hard disk. From this object it is possible to determine which MFT to use, namely the one on this partition. These steps are shown in Fie 11- 39. g' V? Directory C: D \ 2 FtRure \Devlces Harddisk Volume 1 Follow symbolic link to get disk portion 1-39. Step.s in tooki (\ eupt i^FTforHD volume weBTitm Root dlrectorv Look up path name ie hie C 'vtuina\iveh hlrr 5 Relurn Handle to casing ptoces HHandle 1 \ \ \ t Vr-i T I 4

Create new The parsing of the file name continues now at the root directory, whose blocks can be found from ents-y 5 in the MFT (see Fig. ! 1-34). The siring "maria' is now looked up in (he root directory, which returns the index into (he MFT for the directory maria. This directory is (hen searched for the string "web htm" If

SEC. 11.7 THE WINDOWS 2000 FILE SYSTEM 841 successful the result is a new object created by the object manager The object, which is unnamed, contains the index of the MFT record for the file. A handle to this object is returned to the calling process On subsequent ReadFile calls, the handle is provided, which allows the object manager to find the index and then the contents of the MFT record for the file. If a (bread in a second process opens the file again, it gets a handle to a new file object. In addition to regular files and directories. NTFS supports hard links in the UNIX sense, and also symbolic links using a mechanism called re parse points. It is possible to tag a File or directory as a reparse point and associate a block of data with it. When the file or directory is encountered during a file name parse, excep- exception processing is triggered and the block of data is interpreted, It can do various things, including redirecting the search to a different part of the directory hierar- hierarchy or even to a different partition. This mechanism is used to support both sym- holic links and mounted file systems. File Compression NTFS supports transparent File compression A file can be created in compressed mode, which means that NTFS automatically tries to compress the blocks as they are written to disk and automatically uncompresses them when they are read back. Processes that read or write compressed Files are completely unaware of the fact that compression and decompression are going on Compression works as follows. When NTFS writes a file marked for compression to disk, it examines the first 16 (logical) block.s in the File, irrespec- irrespective of how many runs they occupy. It then runs a compression algorithm on them. If the resulting data can be stored in 15 or lewer blocks, the compressed data are written to the disk, preferably in one run, if possible. If the compressed data still take 16 blocks, the 16 blocks are wntten in uncompressed form. Then blocks 16-31 are examined to see if they can be compressed to]5 blocks or less, and so on. Figure ll-40(a) shows a file in which the first 16 blocks have successfully compressed to eight blocks, the second 16 blocks failed to compress, and the third 16 blocks have also compressed by 50% The three parts have been written as three runs and stored

in the MFT record The "missing" blocks are stored m the MFT entry with disk address 0 as shown in Fig. 1 l-40(b) Here the header @.48) is followed by five pairs, two for the first (compressed) run, one for the uncompressed run, and two for the final (compressed) run. When the file is read back, NTFS has to know which runs aie compressed and which are not. It sees that based on the disk addresses. A disk address of 0 indi- indicates that it is the final part of 16 compressed blocks. Disk block 0 may not be used for storing data, to avoid ambiguity. Since it contains the boot sector, using it for data is impossible anyway.

CASH STUDY 2 WINDOWS 2000 k file h M hlixkv Random access to compressed files (s possible, hut tricky Suppose that a process docs a seek to block 35 in Fig 11-40. How does NTFS locate biock 35 ,n a compressed file? The answer is thut it has to read and decompress the entire run first. Then it knows where block ?5 is and can pass i] to any process that reads it. The choice of 16 blocks for the compression unit was a compromise. Making it shorter would have made the compression less effective Making if longer would have made random access more expensive File Encryption Computers are used nowadays to store all kinds of sensitive data, including plans tor corporate takeovers, tax information, and love letters (love email"}, which the owners do not especially want revealed to anyone Information loss can happen when a laptop computer is lost or stolen, a desktop system is rebooted using an MS-DOS floppy disk to bypass Windows 2000 security, or a hard disk is physically removed from one computer and installed on another one with an insecure operating system. Even the simple act of going to the bathroom and leaving the computer unattended and logged m can be a huge security breach Windows 2000 addresses these problem by having an option to encrypt files, so even in the event the computer is stolen or rebooted using MS-DOS.'the files will be unreadable. The normal way to uie Windows 2000 encryption is to mark certain directories as encrypted, which causes all the files in them to be encrypted, and new files moved to them or created in them io be encrypted as well Thu actual encryption and decryption is not done by NTFS itself, but by a driver called EFS (Encrypting File System), which is positioned between NTFS and the user

SEC. 1 THE WINDOWS 2000 FILE SYSTEM varc of c 843 cryption and NTFS process In this way. application programs itself is only partially

involved in it To understand how the encrypting tilt system woiks. it is necessary to under- understand how modern cryptography works For this purpose, a brief review was given in Sec. 9.2 Readers not tdmiliar with the basics of cryptography should read that section before continuing. Now let us see how Windows 2000 encrypts Tiles When the user asks a file to be encrypted, a random 128-blt file key is generated and used to encrypt the file block by bloLk using a symmetric algorithm parameinzed by this key. Each new file encrypted gets a different 128-bit random file key. so no two files use the same encryption key. which increases secunly in case one key is compromised. The current encryption algorithm is a valiant ot DES (Data Encryption Stan- Standard), but ihe FEFS architecture supports ihe addition o] new algorithms in the future Encrypting each block independently of all the others is necessary to make random access still possible. The file key has to be stored somewhere so the tile cm be decrvpted later If it were just stored on the disk in plaintext, then someone who stole or tound the computer could easily decrypt the File, defeating the purpose of encrypting the files. For this reason, the file keys must all be encrypted before Ihey are stored on ihe disk. Public-key cryptography is used for this purpose. After ihe file is encrypted, ihe location of the user's public key is looked up using information in ihe registry There is no danger ot storing the public key's location in ihe registry because it a chief sieals the computer and finds ihe public key, there is no way to deduce ihe pnvate key from it. The 128-bit random file key is now encrypted with ihe public key and the result stored on disk along with Ihe file, as shown in Fi 1141 To decrypt a tile. However, io decrypi i rypied 128-bit random iile kev is fetched from disk. Btrieve the file key, ihe user must present ihe private

 key. Ideally, this should be stored on a smart card, external to the computer, and only inserted in a reader when a file has to be decrypted. Although windows 2000 supports smart cards, it docs not siore private keys on them. Instead, the first time a user encrypts a file using EFS, Windows 2000 gen- generates a (private key. public key) pair and siores the private key on disk encrypted using a symmetric encryption algorithm. The key used for the symmetric algo- algorithm is derived either from the user's login password or from a key stored on the smart card, if smart card login is enabled. In this way, EFS can decrypt the private key at login time and keep it within its own virtual address space during normal operation so it can decrypt the 128-bit file keys as needed without further disk accesses.

When the computer is shut down, ihe private key is erased from EFS' virtual address space so anyone stealing the computer will not have access to the private key. A complication occurs when multiple users need access to the same encrypted file. Currently the shared use of encrypted files by multiple users i.s not sup- supported. However, the EFS architecture could support sharing in the future by encrypting each file's key multiple times, once with the public key or each author- authorized user. All of these encrypted versions of the file key could be attached to the file. The potential need to share encrypted files is one reason why this two-key system is used. If all files were encrypted by lheir owner's key, there would be no way to share any files. By using a different key to encrypt each file, this problem can he solved. Having a random file key per file but encrypting it wnh the owner's sym- symmetric key does not work because having the symmetric encryption key just lying around in plain view would ruin the security.— generating the decryption key from the encryption key is too easy. Thus (slow) public-key cryptography is needed to encrypt the file keys. Because the encryption key is public- anyway, having it lying around is not dangerous. The other reason the two-key system is used is performance. Using public- key cryptography to encrypt each file would be loo slow II is much more effi- efficient to use symmetric-key cryptography to encrypt the data and public-key cryp- cryptography to encrypt the symmetric file key. 11.8 SECURITY IN WINDOWS 2000 Having just looked at encryption in the file system, this is a good tune to examine security in general. NT was designed to meet the U S. Department ot Defense's C2 security requirements (DoD 5200 28-STD). the Orange Book, which we studied in Chap. 9. This standard requires operating systems to have certain properties in order to be classified as secure enough for certain kinds of

SEC 11.8 SECURITY IN WINDOWS 2000 845 military work. Although Windows 2000 was not specifically designed for C2 compliance, it inherits many security properties from NT, including the following: 1. Secure login with antispoofing metres. 2. Discretionary access controls. 3. Privileged access controls. 4. Address space proteciion per process. 5. New pages must be zeroed before being mapped in. 6. Security auditing. Let us review these items briefly (none ol which are met by Windows 98. inciden- incidentally). Secure login means that the system administrator can require all users io have a password in order to log m. Spoofing is when a malicious user writes a program that displays the login prompt or screen and then walks away from the

computer in the hope that an innocent user will sit down and enter a name and password. The name and password are then written to disk and the user is told that login has failed. Windows 2000 prevents this attack by instructing user-, to hit CTRL- ALT-DEL io iog in This key sequence is always capmred by the keyboard driver, which then invokes a system program that puts up the genuine login screen. This procedure works because lhere is no way for user processes to dis- abie CTRL-ALT-DEL processing in the keyboard driver Discretionary access controls allow the owner ol a file or other object to say who can use it and in what way. Privileged access controls allow the system administrator (supeniser) to override them when needed. Address space protec- protection simply means thai each process has its own protected virtual address space not accessible by any unauthorised process. The next item means that when a stack grows, the pages mapped in are initialized to zero so processes cannot find any old information put there by the previous owner (hence the zeroed page lisi in Fig. i 1-27, which provides a supply of zeroed pages lor this purpose). Finally, security auditing allows the administrator to produce a log of certain securuy-related events In the next section we will describe the basic concepts behind Windows 2000 security. After that we will look at the security system calls Finally, we will conclude b\ seeing how setunt> is implemented. 11.8.1 Fundamental Concepts Every Windows 2000 user (and group) is identified by a SID (Security ID). SIDs are binary numbers with a short beader followed by a long random com- component Each SID)s intended to be unique worldwide. When a user starts up a

846 CASH STUDY 2 WINDOWS 2000 CHAP II undi-r the ustr's SID Most ol the security ich object can be accessed onh by threads process, the process and its threads n system is designed to make sure thai with authored SIDs. Each process has an access token that specifies its SlD and otlier properties. It is normally assigned at login time b> \unlogon and is shown in hig 1 \-41, although processes should call GelTokenlnjomhUion to acquire this inforrnatin inge n infor- The expiration time field could teil \vhcn the token ceases to be vatid, but it is currently not used The Croups fietds specify the groups to which the process helongs: this is needed for POSIX conformance. The default DACL (Discretion- (Discretionary ACL) is the access con trot list assigned to objects created by the process if no other ACL is specified. The user SID tell;, who owns the process. The restricted S1DS are to allow untrustworthy processes to lake part in jobs with trustworthy processes but with less power to do damage

Finatly, the privileges listed, if any. give the process special powers, such as the right to shut (he machine down or access fites to which access would othei-wise be denied. In effect, the privileges sprit up (he power of the superuser into several rights that can be assigned to processes individuatty. * In this way. a user am be given some superuser power, but not all of it. In summary, ihe access token tctls who owns the process and which defaults and powers are associated Header Err Groups Default CACL User SIP Group SID Restrrctod SiDS Prn/ileges When a user logs in. winlogon gives the initial process an access token. Sub- Subsequent processes normally inherit this token on down the tine. A process' access token initialty appties to all the threads in the process. However, a thread can acquire a different access token during execution, in which case case the thread's access token overrides the process" access token. In particular, a client thread can pass its access token to a server thread to allow the server to access ihe client's protected files and other objects. This mechanism is called impersonation. Another basic concept is the security descriptor Every object has a security descriptor associated with it that tells who can perform which operations on it. A security descriptor consists of a header followed by a DACL with one or more ACEs (Access, Control Elements) The two main"kinds ot elements are Allow and Deny. An allow element specifies a SID and a bitmap that specifies which operations processes with that SlD may perform on the object A deny element works ihe same way, except a match means the caller may not perform the opera-operation For example, Ida has a file whose security descriptor specifies that everyone

SEC 11.8 SFCURITY IN WINDOWS 2000 847 has read access Elvis has no access. Cath> has read/write access, and Ida herself has full access. This, simple example ,s illustrated in Fig. 1 1-43. The SID Every- Everyone refers to the .set of all users, but it is overridden by any explicit ACEs that tol- Figure In addition lo the DACL, a security descriptor also has a SACL (System Access Control list), which is like a DACL except that it specifies not who may use the object, but which operations on the object are recorded in the system-wide security event log. In Fig. 11-43, every operation that Marilyn performs on the file will be logged. Windows 2000 provides additional auditing features to log sensitive accesses. 11.8.2 Security API Calls Most of the window* 2000 access control mechanism is based on security descriptors. The usual pattern is that when a process creates an object, it provides a

security descriptor as one of the parameters to the Create Process, CreateFile, or other object creation call. Thts security descriptor then becomes the security descriptor attached to the object, as we saw in Fig. 11-43- Tf no security descrip- descriptor is provided in the object creation call, the default security in the caller's access token (see Fig. I1 -42) is used instead. Many of the Win32 AP] security calls relate to the management ot security descriptors, so we will focus on those here The most important calls are hsted in Fig. 11 -44. To create a security descriptor, storage for it is first allocated and then initialized using InitializeSecurityDescriptor. This call fills in the header, If the

 tl owner SID is not known, it can be looked "P by name using LookupAccountSid. It can then be inserted into the security descriptor The same holds for the group SID, if any. Normally, these will be the caller's own SID and one of the caller's groups, but the system administrator can fill in any SIDs. Win32 API function ^ _ Description inrtializeSecuntyDescriptor Prepare a new secunty descriptor for us< LookupAci '■ SetSecuntyDescriptorOwner SetSecurrtyDescriptorGroup Look up the S]D tor a given user na S]D in the security descriptor group S]D in the secunty descriptor a DaCL or SACL .._. __orSACL allowing access a DACL or SACL denying access I DeleteAce __ Remove an ACE from a DaCL or SACL j SetSecuntyDescnptorPac] | Attach a DaCL to a security descnptor Figure 11-44. The principal Win32 APt funciions for ■.euiniy At this point the security descriptor's DACL (or SACL) can be initialized with InitializeAcl. ACL entries can be added using Add Access Allowed Ace, and AddAocessDeniedAce. These calls can be repeated multiple times to add at. many ACE entries as are needed. DeleteAce can be used to remove an entry, more like on an existing ACL than on one being constructed for the first time. When the ACL is ready, SetSecurityDescnptorDact can be used to attach it to the security descriptor. Finally, when the object is created, the newly minted security descrip- descriptor can be passed as a parameter to have it attached to (he object. 11.8-3 Implementation of Security Security in a standalone Windows 2000 system is implemented by a number of components, most of which we have already seen (networking is a whole other story and beyond the scope of this book) Logging in is handled by winlogon and authentication is handled by huss and msgina.dll as discussed in Sec. 11 4.5 The result of a successful login is a new shell with its associated access token This process uses (he SECURITY and SAM keys in the registry. The former sets the general

security policy and the latter contains the security information for the individual users, as discussed in Sec. 11.2.3. Once a user is logged in, security operations happen when an object is opened for access. Every OpenXXX ca]J requires the name of the object being opened and the set of rights needed. During processing of the open, the security manager (see Fig.]]-7) checks to see it the caller has all the rights required. It performs (his check by looking at the caiier's access token and the DACL associated with (he

SEC 11 H SECURITY IN WINDOWS 2000 849 object It goes down the list of entries in the ACL in order As soon as it finds an entry that matches (he caller's SID or one of the caller's groups, (he access found there is taken as definitive. If all the rights the caller needs are available, the open succeeds; otherwise it fails. DACl-s can have Deny entries as well as Allow entries, as we have seen. For this reason, it is usual 10 put entries denying access ahead of entries granting access in the ACL, so that a user who is specifically denied access cannot get in via a back door by being a member of a group that has legitimate access. After an object has been opened, a handle to it is returned to (he caller. On subsequent calls, the only check thai is made is whether the operation now being tried was in the set of operations requested at open time, to prevent a caller from opening a file for reading and then trying to write on it. Any log entries required by the SACL arc made 11.9 CACHING IN WINDOWS 2000 The Windows 2000 cache manager does caching for performance reasons, conceplually similar to caches in oiher operating systems. However, its design has some unusual properties that are worth looking at briefly The cache manager's job is to keep file system blocks that have been used recently in memory to reduce access time on any subsequent reference Windows 2000 has a single integrated cache that works for all the file systems in use, including NTFS, FAT-32, PAT-I6, and even CD-ROM file systems. This means thai the file systems do not need to maintain their own caches. As a consequence ol the design goal to have a single integrated cache despite the presence of multiple file systems, the L-adie manager is located in an unusual position in the system, as we saw in Fig. 11-7 It is not pan of the tile system because there arc multiple independent file systems thai may have nothing >n common. Instead it operates at a higher level than the file systems, which are technically drivers under control of the I/O manager. The Windows 2000 cache is organized hy virtual block, not physical block. To see what this means, remember that traditional file caches keep track o|" blocks by two-part addresses of the form

(partition, block), where the first member denotes a device and partition and the second member is a block number within that partition. The Windows 2000 cache manager doe, not do that Instead, it uses (file, offset) to reier to a block (Technically, streams are cached rather than files, but we will ignore that detail below.) The reason for this unorthodox arrangement is that when a request comes in to (he cache manager, it is specified as (file, offset) because that is all the calling process knows. If cache blocks had been labeled by a (partition, block) tag, the cache manager would have no way of knowing which (file, offset) block cor-

850 CASE STUDY 2: WINDOWS 2CHK) s the file s CHAP s lhat mainl tile responds lo which (partition, block) block since n those mappings. Let us now examine how the cache manager works. When enced, ihe cache manager maps a 256-KB c-hunk of kernel virtual address space onto the file. If the file is larger than 256KB, only a portion of the file is mapped The total amount of virtual address space the cache manager can use is deter- determined at boot lime and depends on the amount of RAM present if the cache manager runs out of 256-KB chunks of virtual addiess space, it must unmap an old file before mapping in a new one. Once lapped, t blocks by just copying from kernel virtual address space to the user buffer. If the bit* copied is not in physical memory, a page fault will occur and the memo manager will satisfy ihe fault in the usual away. The cache manager is not ev aware of whether ihe block was in the cache or nol. The copy always succeeds. The operation of the cache manager is shown in Fig I 1-45 in NTFS file system on a SCSI disk and a FAT-32 file system on an IDE disk. When a process docs a read on a file, the request is directed to the cache manager. If ihe block needed is in the cache, ii is copied to the user immediately. m the cache, ihe cache manager gets a page fault when trying to copy the page fault has been handled, the block is copied to the calling proa Kiser) | ntdll.cttl) | Cache manger) | I/O manager | | NTFS | | FAT^-32 | | SCSI |j 1OE | liser P Call is rocess rea handed in Actual sysiem ca" Cell is If tot* Call to Calf to caught by is absent. file system disk driver Shared library is made ache manager page fault fo get block o read block Figure 11-45. The path [hrough ihe ojche l<>the hardv. jre As j consequence of this design, the cache manager does not know how many of its mapped pages are in physical memory or even how large its cache is. Only

SEC. 119 CACHING IN WINDOWS 2000 85 * the memory manager know-,

for sure. This approach allows the memory manager to dynamically trade off the si/* of the cache against memory for user pages II there is little tile activity but there are many processes active, the memory inanagei can use most of physical memor> for process" pages. On Ihe cither hand, if there is a lot of file activity and few processes, more physical memory can be devoted to the cache. Another property the cache manager has is that coherence is maintained hetween memory-mapped files and tiles lhaf are open lor reading and writing. Consider, for example, a situation in which one process opens some file for read- reading and writing and a second process, maps that file onto its address space. What happens if the second process writes nn the file directly and imniediaiely (hereafter the first one reads in a block that was just changed7 Does it get stale data' The answer is no. In both cases—open files and mapped files— the cache manager maps a 256-KB piece of its virtual address space onto the file. The file is only mapped once, no matter how many processes have it open or mapped. In the case of a mapped file, both the cache manager and the user process share pages in memory. When a read request has to be satisfied, the cache manager iuM copies the page from memory to the user buffer, which always reflects the exact current State of the file because the cache manager is using the same pages as the process that has the file mapped in 11.10 SUMMARY Windows 2000 is structured in the HAL, the kernel, the executive, and a thin system services layer that catches incoming system calls In addition, there aic a variety of device drivers, including the file systems and GDI. The HAL hides cer- certain differences in hardware from the upper layers. The kernel attempts hi hide the remaining differences from the executive to make it almost completely machine independent The executive is based on memory ohjects User processes create them and gel back handles to manipulate them later Executive components can also create objects. The object manager maintains a name space into which objects can be inserted tor subsequent lookup Windows 2000 supports processes, jobs, threads, and fibers. Processes have virtual address spaces and are containers for resources Threads are the unit of execution and arc scheduled hy the operating system I-ibers are lightweight threads that arc scheduled entirely in user space. Jobs arc collections of processes and are used for assigning resource quotas Scheduling is done using a priority algorithm in which the highest priority ready thread rons nest. Windows 2000 supports a demand-paged virtual memory. The paging algo- algorithm is based on the working set concept The system maintains several lists of

852 CASESTUDY2 WINDOWS 2000 CHAR I! free pages, so thai when a page fault occurs, there is generally a free page avail- available. The free page lists are fed by trimming the working sets using complex for- formulas that try to throw out pages that have not been used in a long time. I/O is done by device drivers, which follow the Windows Device Model. Each driver starts out by initializing a driver object that contains, the addresses of the procedures that the system can call to add devices or perform I/O. Drivers can be stacked to act as filters. The NTFS file system is based on a master file table, which has one record per file or directory. Each file has multiple attributes, which can either be in the MFT record or nonresident, on the disk. NTFS supports compression and encryp- encryption, among other features. Security is based on access control lists Each process1 has an access control token that tells who it is and what special privileges it has, if any. Each object has a security descriptor associated with it. The security descriptor points to a discre- discretionary access control list that contains access control entries that can allow or deny access to individuals or groups Finally. Windows 2000 maintains a single system-wide cache for all the tile systems. It is a virtual cache rather than a physical one. Requests for disk blocks go first to the cache, If they cannot be satisfied, then the appropriate file system is called to fetch the needed blocks. 1. Give one advantage and one disadvantage of tile registry versus having individual Am files? 2. A mou.sc can have 1, 2, or 3 buttons. AH three types are in use Dues the HAL hide this difference from ihe rest of'the operating system1 Why or why not9 3. The HAL keeps track of time siartmg in the year 1601 Give an example ol an appli- application where this feature is useful 4. The POS1X subsystem needs to implement LNIX-style signals. If j u.sei hus ihc ke> for the Quit signal, is this scheduled as a DPC or an APC1 5. Many components of the executive (Fig. II-7) call other components of the executive Give three examples of one eomponem calling another one, but use (sn) different components in all 6. Win32 does, not have signals. !f they were to be introduced, they could be per pnicess per thread, both, or neither Make a proposal and explain why n is a good idea 7. An alternative to using DLLs is to statically link each program wnh precisely those library procedures it actually calls, no more and no less If this scheme were to be introduced, would it make more sense on client machines or on server machined'

CHAP I! PROBLEMS 853 8. The file ntdll.dl] exports 1179 function calls while nwtkmUxe exports 1209 function calls Is this a bug" What might be the

cause of this discrepancy'1 9. Objects managed by ihe object manager are van able-wed, with different objects hav- having different sues Can an object start at an arbitrary byte in the nonpaged poo!7 Hwr You do not need any information about Windows 2000 other than what was given m Ihe text. 10. Is there any limit on the number of different operations ihai can be deiined on an exe- executive object? If so. where does this limit come from9 If not, why not9 11. The Win32 API call WaitForMuItipleObjecis allows a thread to block on a set pt syn- synchronization objecis whose handles are passed is parameters As soon a;> any one of them is signaled, the calling thread is released Is it possihle tn have the set of syn- synchronization objects include two semaphores, one mutex, and one critical section? Why or why not? Hint. Thi.s is not a trick question but u does require some careful thought 12. Name three reasons v>hy a proc-ess, might be terminated 13. Consider the situation of Fig 11-19 in which Itk system is ahout to schedule a thread Assuming that each thread is compute bound, how long does it take before some thread at priority 3 gets to run on Windows 2000 Professional1 14. Suppose that the quantum is sei to 20 msec and the current thread, at priority 24, has just started a quantum. Suddenly an IfO operation completes and a priority 28 thiead is made ready. About how long does it have to wait to get serviced1? 15. In Windows 2000, the current priority is always greater than or equal to ihe base prior- priority Are there any circumstances in which it would make sense to have the current priority be lower than ihe base priority? If so, give an example If not. why not'' 16. Some MS-DOS programs were written in assembly language using instructions such as IN port and OUT port Can such programs be run under Windows 20001 If not, can you think of a way they could be supported' 17. Name two ways to give better response time to important processes. 18. Posiiion-mdependent code was briefly discussed in ihe teM It is a technique for allowing two processes to share the same procedure at different virtual addresses Can this problem be solved by just setting the page tables of the two processes appropri- appropriately? Explain your answer 19. Shared libraries in Windows 2000 are contained m dll files that multiple processes may map in at liie same ome If iwo processes need to map in the same shared library at different virtual addresses, a problem will occur. How could this problem be solved on the Pentium using a property of its memory architecture'1 If the solution requires changes to Windows 2000 to implement, state what changes are needed 2W. it a region of virtual address space is reserved but not committed* do you think a VAD is created for if Defend your answer 21.

Wliich of the transitions shown in Fig 11-27 are policy decisions, as opposed to required moves forced by system events (e g.. a process exiting and freeing its pages)?

854 CASE STUDY 2. WINDOWS 2000 CHAP 11 22. Suppose that a page 1S shared and in two working sets at once If ti is evicted from one of the working sets, where does ugom Fig. n-271 23. When a process unmaps 4 clean page, it nwKes the transition E) m Fig H-27 Why i-, there no transition to the modified list when a dmy page is unmapped' 24. Assuming thdt both RAM and video RAM each call read or write a 32-bil word in 10 nsec. how long does it take to paint the hac kg round (in jn X£IA screen m tlie besi 25. A File has the following mapping Gtve the MFT run entries Offset OI2 345 67 89IO Disk address 50 51 52 22 24 25 26 53 54 - 60 26- Consider the MFT record of" Fig. f I -36. Suppose that the Fiic grew and a I Oih block was assigned to the end of the file The number of this block is 66 What would the MFT record lookhke now? 27. In Fig I t-40(b). the first two runs are each ot length 8 blocks. If, tt just dn accident they are equal or does this have to do wtth the way compression works" Explain your 28. Suppose thai you wanted to build Windows 2000 Lite Which of the fields of Fig. J1-42 could be removed without weakening [he security of (he system' 29. The regedit command can be used to export part Or all of the registry to d text file under all current versions of Windows. Save the registry several times during a work session and see what changes. If you have access to a Windows computer on which you can lastall software or hardware, find out what changes when a program or devit e are added or removed 30. Write a UNIX program that simulates writing ^n NTFS file with multiple streams ii should accept a list of one or more files as drgumenls and wnle an ouiput fife ihat am- contents of each of the arguments Now write a second program tor rerxirtmg on the attributes and streams and extracting all the component. 31. Write a program that generates an MS-DOS 8 + 1 filename gi\en any file njme Use the Windows 2000 algorithm

12 OPERATING SYSTEM DESIGN In the past 11 chapters, we have covered a lot of ground and taken a look at many concepts and examples relating TO operating systems. But studying existing operating systems is different from designing a new one. In this chapter we are going to take a quick look at some of the issues and trade-offs that operating sys- systems designers have to take into account when designing and implementing a new system. There is a certain

amount of folklore about what is good and what is bad float- floating around in the Operating systems community, but surprisingly hide has been written down. Probably the most important book is Fred Brooks' classic A975) book The Mythical Man Month in which he relates his experiences in designing and implementing IBM"s OS/360. The 20th anniversary edition revises some of that material and adds four new chapters (Brooks, 1995). Probably the only operating systems textbook dealing wtth design in a serious way is Operating Svs- tems: A Design-Oriented Approach (Crowiey, 1997). Three classic papers on operating sysiem design are "Hints for Computer Sys- System Design" (Lampson, 1984), "On Building System^ lhat Will Fail" (Corbdtd, 1991), and -End-to-End Arguments in System Design" (Salizer et al., 1984). Like Brooks' book, all three papers have survived the years extremely well; most of their insights are still as valid now as when they were first published. This chapter draws upon these sources, plus the author's personal experience as designer or co-designer of three systems: Amoeba (Tanenbaum et al., 1990), MIN1X (Tanenbaum and Woodhull, 1997), and Globe (Van Steen et al., 1999a).

856 OPERATING SYSTEM DESIGN CHAP U Since no consensus exists among operating system designers about the best way to de-sign an operating system, this chapter will thus be more personal, specula- speculative, and undoubtedly more controversial than the previous ones. 12.1 THE NATURE OF THE DESIGN PROBLEM Operahng system design is more of an engineering project than an exact sci- science. Ii is much harder to set clear goals and to meei them. Let us look at aart with these points. 12.1.1 Goals In order to design a successful operating system, the designers have to have a clear idea of what they want. Lack of a goal makes it very hard to make subse- subsequent decisions. To make this point clearer, it is instructive to take a look at two programming languages PL/I and C, PL/1 was designed by IBM in the 1960s because it was a nuisance to have to support both FORTRAN and COBOL, and embarrassing to have academics yapping in the background that Algol was better than both of them. So a committee was set up to produce a language that would be all things to all people; PL/I, it had a little bit of FORTRAN, a little btt of COBOL, and a littie bit of Algol. It failed because it lacked any unifying vision. It was simply a collection of features at war wtth one another, and too cumber- cumbersome to be compiled efficiently, to boot. Now consider C. It was designed by one person (Dennis Ritchie) for one pur-

purpose (system programming). It was a huge success, in no small pan because Ritchie knew what he wanted and did not want. A.s a result, it is still in wide-widespread use decades after its appearance. Having a clear vision of what you want is crucial. What do operating system designers want? It obviously vanes from system to system, being different for embedded systems than for server systems. However, for general-purpose operating syslein.s fourmatn items come to mind; 1. Define abstractions. 2. Provide primitive operations. 3. Ensure isolation. 4. Manage the hardware Each of these items will he discussed below The most important, but probably hardest task of an operating systeni is to define the right abstractions. Some of them, such as processes and files, have been around so long that they may seem obvious. Other ones, such as threads, are

SEC f 2.1 THE NATURE OF THE DESIGN PROBLEM 857 newer, and are less mature. For example, if a multithread process thai has one Ihread blocked wailing for keyboard input forks, is there a thread ,n ihe new proc- process also waiting for keyboard input? Other abstractions relate to synchronization, signals, the memory model, modeling of I/O and many other areas, Each of the abstractions can be instantiated in the form of concrete data struc- structures. Users can create processes, files, semaphores, etc, The primitive opera-operations manipulate these data structures. For example, users can read and write files. The primitive operations are implemented in the form of system calls. From the user's point of view, the heart of the operating system is formed by the abstractions and the operations on them available via the system calls. Since multiple users can be logged into a computer at the same time, the operaung system needs to provide niechaniMns to keep them separated. One user may nof interfere with another. The process concept is widely used to group resources together for protection purposes. Files and other data structures gen- generally are protected as Well Making sure each user can perform only authorized operations on authorized data is a key goal of system design. However, users also want to share data and resources, so the isolation has to be selective and under user control. This makes it much harder. Closely related to this point is the need to isolate failures. If some part of the syslem goes down, most commonly a user process, it should not be able to take the rest of the syslem down with it The system design should make sure that the various parts are well isolated from one another. Ideally, pans of the operaung system should also be isolated from one another to allow independent failures.

Finally, the operating system has to manage the hardware. In particular, it has to take care of all the low-level chips, such as interrupt controllers and bus con- controllers. It also has to provide a framework for allowing device drivers to manage the larger I/O devices, such as disks, printers, and the display. 12.1.2 Why is it Hard to Design an Operating System? Moore's law says that computer hardware improves by a factor of 100 every decade. Nobody has a law saying that operating systems improve by a factor of 100 every decade. Or even get better at all. In fact, a case can be made thai some of them are worse in key respects (such as reliability) than UNIX Version 7 was back in the 1970s. Why? Inertia and the desire for backward compalibiliiy often gel much of the blame, and the failure to adhere to good design principles is also a culprit. But there is more to it. Operating systems are fundamentally different in certain ways from small application programs sold in computer stores for $49. Let us look at eight of the issues that make designing an operating system much harder than designing an application program. First, operating systems have become extremely large programs. No one per- person can sit down at a PC and dash off a serious operating system in a few months.

858 OPERATING SYSTbM DESKiN CHAP. 12 All current versions of UNIX exceed I million line* ot code. Windows, 2000 is 29 million lines of code. No one person can understand even 1 million lines of code, let alone 29 million hntis of code When >ou have a product that none of ihe designers can hope to fully understand, it should be no surprise that the results are often far trom optimal Operating systems are not the most complex systems around. Aircraft carriers arc far more complicated, for example, but they partition into isolated subsystems better. The people designing the toilets on a aircraft earner do not have to worry about the radar system. The two subsystems do not interact much In an operat- operating system, the file system often interacts with the memory system in unexpected and unforeseen ways- Second, operating system* have to deal with concurrency. There are multiple users and multiple I/O devices all active at once Managing concurrency is inherently much harder than managing a single sequential activity. Race condi- conditions and deadlocks are just two of the problems that come up. Third, operating systems have to deal with potentially hostile users—users who want to interfere with system operation or do things that are forbidden from doing, such as stealing another user's files. The operating system needs to take measures to prevent these users from hehaving improperly. Word processing pro- programs and photo editors do not

have this problem. Fourth, despite the tact that not all users trust each other, many users do want to share some of their information and resources with selected other users. The operating system has to make this possible, but in such a way that malicious usurs cannot interfere. Again, application programs do not face anything like this chal- challenge. Fifth, operating systems live for a very long time. UNIX has been around for a quarter of a century; Windows has been around tor over a decade and shows no signs of vanishing. Consequently, the designers have to think about how hard- hardware and applications may change in the distant future and how they should prepare for it. Systems that are locked too closely into one particular vision of the world usually die off, Sixth, operating system designers really do not have a good idea of how their systems will he used, so they need to provide for considerable generality Neither UNIX nor Windows was designed with email or Web browsers in mind, yet many computers running these systems do little else. Nobody tells a ship designer to build a ship without specifying whether they want a frshing vessel, a cruise ship, or a battleship. And even fewer change their minds after the product has arrived Seventh, modem operating systems are generally designed to be portable, meaning they have to run on multiple hardware platforms. Tliey also have to sup- support hundreds, even thousands of I/O devices, all of which are independently designed with no regard to one another. An example ot where this diversity causes prohlems is the need for an operating system to run on both little-endian and btg-endian machines. A second example was seen constantly under MS-DOS

SEC 12 I THF NATURE OF THE DESIGN PROfil.HM 859 when users attempted to install, say. a sound card and a modem that used the same I/O p»rt< or interrupt request lines Few progr.ims other than operating systems have to deal with sorting out problems caused by conflicting pieces of hardware Eighth, and last in our hst, is the frequent need to be backward compatible witl) some previous operating system- That sysicm may have restrictions on word leneths. file names, or other aspects thiit the designers now regard as obsolete, hut are "stuck with It is like converting a factory to produce next year's cars instead nf this jear's cars, but while continuing tci produce this year's cars at full capa- capacity. 12.2 INTERFACE DESIGN It should be clear by now that writing a modem operating system is not easy. But where does one begin? Probably the best place to begin is to tbmk about the interfaces it provides. An operating system provides a set of services, mostly

data types (e.g., files) and operations on them (e.g., read). Together, these torm the interface to its users. Note that in this context the users of the operating system arc programmers wbo write code that use system calls, not people lunning appli- application programs. In addition to the main system call interface, most operating systems have additional interfaces, t-or example, some programmers need to write device drivers to insert into the operating system. These drivers s,ee certain features and can make certain procedure calls. These features and calls also define and inter- interface, but a very different one from one application programmers see All of these interfaces must be carefully designed if the system is to succeed 12.2.1 Guiding Principles Are their any principles that can guide interface design'? We believe tht-re arc. Briefly summarized, they arc simplicity, completeness, and tbe ahility to be implemented efficiently. Principle 1: Simplicity A simple interface is easier to understand and implement in a bug-free way All system designers should memorize this famous quote from the pioneer French aviator and writer, Antoine de St Exupery. Perfection is reached not when there n no longer anything in add, but whan there m no longer uri\thw}> to tuke ttwt.t\. This principle says that less is hetter than more, at least in the operating s>s-tem itself. Another way lo say this is the KISS principle: Keep It Simple, Stupid.

860 OPERATING SYSTEM DESIGN CHAP 12 Principle 2: Completeness Ot course, the interface must make it possible to do everything that the users need to do. that is, it must be complete. This brings us to another famous quote, this one from Albert Einstein: Everything should be as simple as possible, but no simpler. In other words, the operating system should do exactly what is needed of it and no more. If users need to store data, it must provide some mechanism for storing data If users need to communicate with each other, the operating system has to provide a communication mechanism, and so on. In his 1991 Turing Award lec- lecture. Fernando Corbato. one of the designers of CTSS and MULT1CS, combined the concepts ot simplicity and completeness and said -SP 0 25 for complexity has a way of compounding difficulties and as we have seen, creating mistakes. My definition of elegance is the achievement of a given functionality with a minimum of tnechamstn and ct maximum of clar- uy. The key idea here is minimum of mechanism. In other words, every teature, func- function, and sysiem call should carry its own weight. It should do one thing and do it well. When a member of the design team proposes extending a

system call or adding some new feature, the others should ask the question; "Would something awful happen if we left it out1.1" If the answer is: "No, but somebody might find this feature useful some day" put it in a user-level library, not in the operating system, even if it is slower that way. Not every feature has to be faster than a speeding bullet. The goal is lo preserve what Corbato called minimum of mech- mechanism. Let us briefly consider two examples from1 my own experience MtNIX and A- moeba. For all intents and purposes, MINIX has three system calls, send, receive, and sendrec. The system is structured as a collection of processes with the memory manager, the file system, and each device driver being a separate sehcd- ulable process. To a first approximation, all the kernel does is schedule processes and handle message passing between them Consequently, only two system calls are needed: send, to send a message, and receive to receive one. The third call, sendrec is simply an optimization for efficiency reasons to allow a message to be sent and the reply to be requested with oniy one kernel trap, Everything else is done by requesting some other process (for example, the file system process or the disk driver) to do the work. Amoeba is even simpler It has only one system call: perform remote pro- procedure call. This call sends a message and waits tor a reply. It is essentially the same ss MINIX' sendrec. Everything else is built on this one call.

SEC. 12.2 INTERFACE DESIGN 861 Principle 3: Efficiency The third guideline is efficiency of implementation. If a feature or system call cannot be implemented efficiently, it is probably not worth having. It should also t>e intuitively obviously to the programmer about how much a system call costs. For example, UNIX programmers expect the Iseek system call to be cheaper than the read system call because the former just changes a pointer in memory while the latter performs disk I/O If the intuitive costs are wrong, programmers will write inefficient programs. 12.2.2 Paradigms Once the goals have been established, the design can begin. A good starting place is thinking about how the customers will view the system. One of the must important issues is how to make all the feature-, of the system hang together well and present what is often called architectural coherence. In this regard, it is important to distinguish two kinds of operating system "customers." On the one hand, there are the users, who interact with application programs; on the other are the programmers, who write them. The former mostly deal with the GUI; the latter mostly deal with the system call interface. If the intention is to have a sin- single GUI that

pervades the complete system, such a*, m the Macintosh, the design should start there. If, on the other hand, the intention is to support many possible GUIs, such as in UNIX, lhe system call interface should be designed first. Doing the GUI first is essentially a lop-down design. The issues are what features will it have, how will the user interact with it, and how should the system be designed to support it? For example, if most programs display icons on the screen and then wait for the user to click on one of them, lhis suggests an event-driven model for the GUI and probably also for the operating system. On lhe other hand, if the screen is mostly full of lext wmdows, then a model in which processes read from the keyboard is probably better. Doing the system call interface first is a bottom-up design. Here the issues are what kinds of features programmers in general need. Actually, not many spe- special features are needed to support a GUI. For example, lhe UNIX windowing Sys- lem, X, is just a big C program that does reads and writes on the keyboard, mou^e, and screen. X was developed long after UNIX and did not require many changes to the operating system to get tt to work. This experience validated thai laet that UNIX was sufficiently complete. User Interface Paradigms For both the GUMevel interface and the system-call interface, the moi.1 important aspect is having a good paradigm (sometimes called a metaphor) to pio- vide a way of looking at the interface. Many GUIs for desktop machines use the

862 OPERATING SYSTEM DESIGN CHAP <2 WIMP paradigm that we discussed in Chap. 5 This paradigm uses point-and- click, point-and-doubie-chck, dragging, and other idioms throughout the interface to provide an architectural coherence to the whole Often (here are additional requirements for programs, such as having a menu bar with FILE, EDIT, and oilier entries, each of which has eenain well-known menu items. In this way, users who know one program can quickly learn another one However, the WIMP user interface is not the only possible one. Some palm- palmtop computers use a stylized handwriting interface Dedicated multimedia de\ices may use a VCR-hke interface. And of couise, voice input has a com- completely different paradigm Whai is important is not so much the paradigm chosen, but the fact that there is a single overriding paradigm that unifies the enure user interface. Whatever paradigm is chosen, n is important lhat a!l application programs use it Consequently, the system designers need to piovide libraries and tool kits to application developers that give them access to procedures that pioducc lhe uni- uniform look-and-feel. User interface design is very important, but it is

(tot the sub- subject of this book, so we will now drop back down to the subject of the operating system interface. Execution Paradigms Architectural coherence is important at the user level, but equally important at the system call interface level Here tt is frequently useful to distinguish between n paradigm and lhe data paradigm, si> we will do both, starting with > time to get user input, obtain operating systei approach is outlined in Fig. I2-I(a). The other execution paradigm is the event-driven paradigm of Fig. I2-I(b) Here the program performs some kind of initialization, tor example by displaying a certain screen, and then waiting for the operating system to tell it about the firsi -vent. The event is often a key being struck or a mouse movement. This design ts that are consumed by process

INTERFACE DESIGN main() int . ; do something)); read{..), do_ something else(), wn(e(..), keep goingf); exil@): (a) Figure 12-1. U) Mgortfhtllti Data Paradigms mam() mess tmsg; whiie'(ge!_message(&msg)) { switch (msg-type) { case 1: .. : case 2-..., case 3- , (''> Hide (hi Lvcnt driven todc The execution paradigm is itot the only one exported hy the operating system An equally important onr is the data paradigm. The key question here is how are system structures and devices presented to the programmer. In early FORTRAN batch systems, everything was modeled as a sequential magnetic tape. Card decks read in were treated as input (apes, card decks to be punched were treated as out- output tapes, and output for the printer was ireated as an output tape. Disk files were also treated as lapes. Random access to a file was possihle only hy rewinding the tape corresponding to the file and reading it again. The mapping was done using job control cards like these MOUNT(TAPE08, REEL781) RUN{INPUT, MYDATA, OUTPUT, PUNCH, TAPE08) The firsi card instiucicd the operator t*' go get tape reel 781 trcmi the lape rack and mount it on tape drive 8 The second card instructed ihe operating system to run the just compiled FORTRAN program, mapping INPUT (meaning the card reader) to logical tape I. disk file MYDAJA to logical tape 2. the printer tcalled OUTPUT) to logical tape 3, the card punch (called PUNCH) to logical tape 4. and physical tape drive 8 to logical tape 5. FORTRAN had a syntax tbr reading and writing logical tapes By reading from logical tape 1. the program got card input. B> writing to logical tape 3. out- output would later appear tin the printer- By reading from logical tape 5, tape reel 781 could be read in, and so on. Note that the tape idea was just a paradigm to integrate the card reader, printer, punch, disk files, and tapes. In this example, only logical tape 5 was a

physical tape; the rest were ordinary (spooled) disk files. It was a primitive paradigm, but it was a start in the right direction.

864 OPERATING SYSTEM DESIGN CHAP [2 Later tame LJNiX, which goes much funhcr using the model of "everything is a file." lining this paradigm, all I/O devices are treated as tiles and can he opened and manipulated as ordinary files The C statements fd1 5 0pen<'f]le1\O_RDWR); Id2 = openGdev/tty", O.RDWR)' open a true disk file and ihe user's terminal. Subsequent statements can use fdl and fd2 to read and *nte them, respectively From ihai point on, there is no diffeience between accessing the file and accessing ihe terminal, except thai seeks on the terminal are not allowed. Noi only docs UNIX unify files and I/O device-,, but it also allows oiher processes to be accessed over pipes as files Furthermore, when mapped files are supported, a process can gel at its own virtual memory as though n were a nle. Finally, in versions of UNIX that support the /prof file system, the C statement 1d3 = OpenGproc/501", O_RDWR); allows the process to (try to) access process 501\ memory for reading and writing Using file descriptor/il?, something useful for, say. a debugger. Windows 2000 goes further still and tries to make everything look like an object. Once a process has acquired a valid handle to a file, process, semaphore, mailbox, or other kernel object, it can perform operations on it. This paradigm is even more general than that of UNIX and much more general than that of FOR- FORTRAN. Unifying paradigms occur in other contexts as well One of them is worth mentioning here: the Web. The paradigm behind the Web is that cyberspace is full ot documents, each of which has a URL. By typing in a URL or clicking on an entry backed hy a URL, you gel the document. In reality, many "documents" are not documents at all, but are generated by a program or shell script when a request comes in. For example, when a user asks an online store for a list of CDs by a particular artist, the document is generated on-the-fly by a program, it cer- certainly did not exist before the query was made. We have now seen four cases: namely, everything is a tape, file, object, or document. In all four cases, the intention is to unify data, devices, and other resources to make them easier to deal with Every operating system should have such a unifying data paradigm. 12.2.3 The System Call Interface If one believes in Corbato's dictum of minimal mechanism, then the operating system should provide as, few system calls as it can get away with, and each one should be as simple as possihle (but no simpler). A unifying data paradigm can play a major role in helping here. For

example, if files, processes, I/O devices, and much more all look like files or objects, then they can all be read with a single

SEC. 12.2 INTERFACE DESIGN 865 read system call. Otherwise it may be necessary to have separate calls for read_file. read_proc, and read_tty, among others. In some cases, system calls may appear to need several variants, but it is, often better practice to have one system call that handles the general case, with different library procedures to hide this fact from the programmers. For example, UNIX has a system call for overlaying a process virtual address space, exec. The most general call is exec(name, argp, envp); which loads Ihe executable file name and gives it arguments pointed to by argp and environment variables pointed to by envp Sometimes it is convenient to list Ihe arguments explicitly, so the library contains procedures lhat are called as fill-execl(name, argO, arg1,..., argn, o); execie(name, argO, arg1 argn, envp}; All these procedures do is stick the arguments m an array and then call exec lo do the work. This arrangement is the best of both worlds: a single straightforward system call keeps the operating sysiem simple, yet the programmer gets the con- convenience of various ways to call exec. Of course, trying lo have one call to handle every possible case can easily gel out of hand, in UNIX creating a process requires two calls: (ork followed by exec. The former has no parameters; the latter has three parameters. In contrast, (he Win32 API call for creating a process, CreateProcess, has 10 parameters, one of which is a pointer to a structure with an additional 18 parameters A long time ago, someone should have asked the question* "Would somelhing awful happen if we left some of these out'." The truthful answer would have been. "In some cases programmers might have to do more work to achieve a particular effect, but the net result would have been a simpler, smaller, and more reliable operating system." Of course, the person proposing ihe 10+ 18 parameter version might have added: "But users like all these features." The rejoinder to that might have beenr "They like systems that use little memory and never crash even more." Trade-offs between more functionality at the cost of more memory aro at least visible and can be given a price tag (since the price of memory is known). However, it is hard to estimate the additional crashes per year some feature will add and whether the users would make the same choice if they knew the hidden price. This effect can be summarized in Tanenbaum's first law of software- Adding mere code adds more bugs Adding more features adds more code and thus adds more bugs. Programmers who believe adding new

features does not add new bugs are either new to comput- computers or believe the tooth fairy is out there watching over them.

866 OPFRATING SYSTEM DFSIGN CHAP 12 Simplicily is not the only issue thai comes <HJ| when designing system calls. An important consideration is Lampson's A984) slogan- Don'(hide power If the hardware has an extremely efficient way ot doing something, n should be exposed to the programmers in a simple way and not buried inside some other attraction The purpose of abstractions is tit hide undesirable properties, not hide desirahle ones. For example, .suppose that the hardwaiv has a special wa> to indve large bitmaps around the screen (i.e.. the video RAM) at high speed It would be juslified to have a new system call to gel at ihis mechanism, lather than lust providing ways to read video RAM into main memory and write it buck again The new call shou|d |Ust move bils and nothing else If a system call is fast, users can ylways build more convenient interlaces on top of u If it is slow, nobody will use it Another design issue is connect] on-(men ted versus connectionless calls. The standard UNIX and Win 12 system calls for reading a file are connection-oriented First you open a file, then >ou read it, finally you close it Some remote t,|e access protocols are alsij connect ion-onen ted. For example, to use 1=TP, the user first logs in to the remote machine, reads the files, and then logs out. On the other hand, some remote file access protocols are connectionless NFS is connectionless, as we saw in Chap. 10 h.ich NFS call stands on its own, so files are not opened before reading or writing And. of course, they do not ha\e to be closed afterward, either. The Web is also connectionless to read a Web page you just ask for it; there is no advance setup required (a TCP connection is required, but this is at a lower level of protocol: the HTTP protocol for accessing the Web ,tself is connect tonl ess). The trade-off between an) connect] on-on en ted mechanism and a connection-connectionless one is the additional work required to set up the mechanism (eg . open tlie file), and the gain from not having to do it on (possibly many) subsequent calls. For file I/O on a single machine, where the setup cost is low. probably the stan- standard way {first Open, then use) is the best way For remote file systems, a case can be made both ways. Another issue relating to the sy.slem call interface is its visibility. The Us! of POS|X-mandated system calls is easy to find. A.IJ UNIX syslems support ihese, .is well as a small number of other calls, but the complele list is always public. In contrast, Microsoft has never made the list of Windows 2000 system calls public.

Instead the Win32 API and other APIs have heen made pubhc, bul these contain vast numbers of library calls (over 13,000 fur Windows 2000). but only a small number are true system calls. The argument for making ail ihe system cn)U public is that il lets programmers know what is cheap (functions performed m user space) and what K expensive (kernel calls). The argumenl lor not making them public is thai it gives Ihe implemented the flexibility of changing the actual underlying system calls to make them better without breaking user programs.

SEC 12 1 IMPLEMENTATION ™' 12.3 IMPLEMENTATION Turning away from ihe user and system oil inierfaces. lei us now lake a look al how to implc-menl an operating sy.siem Jn the nexl eighi seciions we wilJ i-x- amine some general conceptual issues relating In implementation strategies. After lhal we will look ai some low-level techniques Ihat are otien helpful 12,3.1 System Structure Probably I hi: firs I decision Ihe implemenlers have to make is whal (he sysinit simclure should be We examined Ihe main possibilities in Sec. I 7, but Mill rcv>en Ihcm here- An un si ruptured monohltoc design is really not a good idea, except maybe for a tii)j operating system in, s.ty. a refrigenilor, but e\en there >l is arguable. Layered Systems A reasonable approach thai has been well established over ihe years is ;i lay- layered system. Dijkstra'n THE system (Fig. 1-25) was the firsi layered operating sysiem. UNIX (Fig. 10-3) and Windows 2000 (Fig 11-71 also ha\e a layered siructure, but the layering in both of them >s more a way of irymg lo describe ihe sysiem |han a real guiding principle lhat was used in building the system For a new sysiem, designers choosing lo go ihis route should fini very caie- fully choose the layers and define ihe functionality of each one. The bottom layer should always try to hide the worst idiosyncracies of the hardware, as ihe HAL does in Fig, 11-7 Prohably the next layer should handle imeirupts, context switching, and the MMU. so above this level, the code is mostly machine independent. Above this, different designers will have different tastes (and biases). One possibility is to have layer 3 manage threads, including scheduling and mterthread synchronization, as shown m Fig 12-2. The idea here is thai starl- starling ai layer 4 we have proper threads thai are scheduled normally and synchronise using a standard mechanism (e.g., mutexes) In layer 4 we might find the device drivers, each one luniting as a -.epanite |hread, with its own slate, program counter, regislcrs, etc., possibly (but nol neces- necessarily) within the kernel address space. Such a design can gruatly simplify |he I/O structure because

when an interrupt occurs, ii can he converted into an unlock on a mutex and a call lo |he scheduler to (poientially) schedule the newly readied Ihread |hat was blocked on the mutex. M1N1X uses this approach, but m UNIX, Linux, and Windows 2000, the intermpt handlers run m a kind of no-man's land, ra(her than as proper threads that can be scheduled, suspended, eic. Since a huge amount of |he complexity of any operating system is in the I/O, any technique for making it more tractable and encapsulated is worth considering.

OPERATING SYSTEM DESIGN Driven Driver 2 s. thread scheduling, thread synchjonizalion Hide the tow-level hardware rn layere Figure 12-2. One possihtc design for 4 modet Above layer 4, we would expect to find virtual memory, one or more file sys- systems, and the system call handlers. If the virtual meniory is at a lower level lhan the file systems, then the block cache can be paged out, allowing the virtual memory manager lo dynamically determine how the real memory should be di- divided among user pages and kernel pages, including the cache. Windows 2000 works this way. Exokernels While layering has its supporters among system designers, there is also another camp that has precisely the opposite view (Engler et al, 1995). Their view is based on the end-to-end argument (Saltzer et al.. 1984). This concept says that if something has to be done by the user program itself, it is wasteful to do it in a lower layer as well. Consider an application of [hat principle to remote file access. If a system is worried about data being corrupted in transit, it should arrange for each file to be checksummed at the time it is written and the checksum stored along with the file. When a file is transferred over a network from the source disk to the destination process, the checksum is transferred, too, and also recomputed at |he receiving end. If the two disagree, the file is discarded and transferred again This check is more accurate than using a reliable network protocol since it also catches disk errors, memory errors, software errors in the routers, and other errors besides bit transmission errors. The end-to-end argument says that using a reliable network protocol is then not necessary, since the end point (the receiving process) has enough information to verify the correctness of the file itself. The only reason for using a reliable network protocol in this view is for efficiency, that ii, catching and repairing transmission errors earlier. The end-to-end argument can be extended to almost all of the operating sys- system. It argues for not having the operating system do anything that the user

SEC 12.3 IMPLEMENTATION 869 program can do itseif. For example, why have a file system? JuKt let the user read and write a port,on of the raw disk in a protected way. Of course, most users like having files, but the end-to-end argument says, thai the tile t>ystem should be a library procedure linked with any program thai needs to use files. This approach allows different programs to have different file systems- This line of reasoning says Mat all the operating system should do if, securely allocate resources (e.g.. the CPU and the disks) among the competing users The Exokemei is an operat- operating system built according to the end-to-end argument (Engier et al., 1995). Client-Server Systems A compromise between having the operating system do everything and the operating system do nothing is to have the operating system do a little hit. This design leads to a microkernel with much of the operating s>stem running as user- level server processes as illustrated in Fig 1-27. This is the most modular ;ind flexible of all the designs. The ultimate in flexibility is to have each device driver also run as a user process, fully protected against the kernel and other drivers. Getting the drivers out of the kernel would eliminate the largest source of instahii- ity in any operating system—buggy third-party drivers—and would be a tremen- tremendous win in terms of reliability. Of course, device drivers need to access the hardware device registers, so some mechanism is needed to provide this If the hardware permits, each driver process could be given access to only those I/O devices it needs For example, with memory-mapped I/O, each driver process could have the page for its device mapped in, but no other device pages. If the I/O port space can be partially pro- protected, the correct portion of it could be made available to each driver Even if no hardware assistance is available, ihe idea can still be made to work. What is then needed is a new system call, available only to device driver processes, supplying a list of (port, value) pairs. Whai the kernel does k t'irsi check lo see if the process owns aii ihe ports in Ihe list. If so, it then copies the corresponding values to the ports to initiate device I/O. A similar cali can be used to read I/O ports in a proiected way. This approach keeps device drivers from examining (and damaging) kernel data structures, which is (for the mos! pan) a good thing- An analogous set of calls could be made available !o allow driver processes to read and write kernel tables, bui only in a controlled way and with the approval of the kernel The mam problem wilh (his approach, and microkernels in general, is the per- performance hit all the extra context swilches cause However, virtually all work on microkernels was done many years ago when CPUs were much slower Nowa- Nowadays,

applications, that use every drop of CPU power and cannot tolerate a small loss of performance, are few and far between. After all, when running a word processor or Web browser, the CPU is probably idle 90% of ihe time. If a microkernel-based operaiing system turned an unreliable 900-MHz system into a

870 OPERATING SYSTEM DESIGN CHAP 12 reiiabie 800-MHz system, probably few users would complain- After all, most of them were quite happy only a few years ago when they go! their previous com- computer, a! !he ihen-stupendous speed of 100 MHz. Extensible Systems With the clienl-server systems, discussed above, the idea was !o get as much our of the kcmei as possible. The opposite approach is to put more modules into the kernel, but in a protected way. The key word here i.s protected, of course. We studied some protection mechanisms in Sec. 9.5.6 that were initially intended for importing applets over ihe Internet, bu! are equally appiicable to inserting foreign code into kernel, The most important ones are sandboxing and code signing a interpretation is not really practical for kernel code. Of course, an extensible system by itself is not a way to structure an operating system. However, by starting wilh a minima! syslem consisting of iittle more lhari a protection mechanism and then adding protected modules to the kernel one at a time until reaching the functionality desired, a minimal system can be built for the application a! hand. In this view, a new operating system can be tailored 10 each application by including only Ihe parts it requires. Paramecium is an example of such a system (Van Doom, 200]). Kernel Threads Another issue relevant here is that of system threads, no matter which struc- structuring model is chosen. It is sometimes convenieni lo allow kernel threads to exist, separate from any user process. These threads can run in the background, writing dirty pages to disk, swapping processes between main memory and disk, and so on. In fact, the kernel itself can be structured enlirely of such threads, so that when a user does a system call, instead of ihe user's thread executing in ker- kernel mode, the user's thread blocks and passes control to a kernel thread thai lakes over to do the work. In addition to kernel threads running in the background, mot.! operaling sys- lems star! up many daemon processes in the background, too. While these are not part of the operating system, they often perform "system" type activities- These might including getting and sending email and serving various kinds of requests for remote users, such as FTP and Web pages, 12.3.2 Mechanism versus Policy Another principle (hat helps

architeclural coherence, along with keeping things small and well structured, is that of separating mechanism from policy. By pulting ihe mechanism in the operating system and leaving the policy (o user processes, the system itself can be left unmodified, even if there is a need to change policy. Even if the policy module has to he kept in Ihe kernel, it should be

isolated ft not affect the mechan ;han IMPLEMENTATION f ™ bl o th t 1 871 n th p 1 > nod 1 d imxl To make the split betw n pi 1 > dnhn Ie real-world examples- As a (r t a npl o d 1 G payroll department, which h ^ fp ne th nplo> computers, software, blank h ck g en nt wth b nk I t _, lly paying out the 1 r H th pi 1 . paid how much—is coniplet b, pa t n 1 d d d b> n anae roil department just does wh t t t Id to d As the second example on d t nt 1 h th n h ing diners, including table plat w t kit h t II ot q ments with credit card conpant nd n Th pi I > namely, what is on the men If the h t d are in, this new policy can be h ndled bj th Now let us consider s< ne op t ne scheduling. The kernel co Id h d The mechanism entrj p n\ th t h 1 It ha i ti han n i n n^ ho eet n nt Th p j i t the chef. Fig. 11-19. Each The scheduler ju selecting the firs may have differen ides lha! ! ! ng n e h n n n e npt F i n de thre d h -d Ie w th k p ! Ie el ly, indexed by priority level, as shown in Fig. 10-11 or the head of a lis! of ready liireads ai thai priority lev h h f hh i searches (he rea cla , but t g It ry lr i ai prioy vl array from highest priority to lowest priority The policy is setting the priorities. The system y re rs, each with a different priority, for example I might also allow user processes to set the relative priority of its threads Priorities might be increased atter completing I/O or decreased after using up a quantum. There are numerous other policies that could be followed, but the idea here is the separation between setting policy and carrying it out A second example is paging. The mechanism involves MMU management, keeping lists of occupied pages and free pages, and code for shuttling page from disk. The policy is deciding'what (o do when a page fault be local or global, LRU-based or FIFO-based, or something el, nthm can (and should) be completely separate from the mechanics (managing the pages. A third example is allowing modules to be loaded into the ke mechanism concerns how ihey are inserted, how ihey are linked, what can make, and what calls can be made on them. The policy is determing allowed to load a module into the kernel and which modules. Maybe only the superuser can load modules, but maybe any user can load a module that has been digitally signed by the appropriate authorit) 12.3.3 Orthogonality Good sysiem design consists of separate concepts thai can be combined

independently. For example, in C, there are primitive data types including inte- integers, characters, and floating-point numbers. There are also mechanisms for com- is algo ctualh lls they ho is

872 OPERATING SYSTEM DESIGN CHAP. 12 brning data lypes, including arrays, s Theie tdca.s combtne independently, allowing arrays of integers, arrays of characiers, struciure and union members that are float ing-po mi numbers, etc. In fact, once a new data type has been defined, such as an array of integers, it can be used as if it were a pnmi- nve data type, for example as a memher of a structure or a union. The ability to combine separate concepts independently is called orthogonality, li is direct consequence of the simplicity and completeness principles. The concept of orthogonality also occurs in operating systems in various dis- disguises. One example is the Linux done system call, which creates a new thread. The call has a bitmap as a parameter, which allows the address space, working directory, file descriptors, and signals to shared or copied individually. If every- everything is copied, we have a new process, the same as fork, If nothing is copied, a new thread is crcaied in the current process. However, it is also possible 10 create intermediaie forms of sharing not possible in traditional UNIX systems. By separating out the various features and making them orthogonal, a finer degree of control is possible. Another use of orthogonaliiy is the separation of the process concept from the thread concept in Windows 2000. A process is a container for resources, nothing more and nothing less. A thread is a schedulable entity. When one process is given a handle for another process, it does not matter how many threads it has. When a thread is scheduled, it does not matter which process it hetongs to These concepts are orthogonal. Our last example of orthogonality comes from UNIX. Process creation there is done in two steps: fork plus exec. Creating the new address space and loading it with a new memory image are separate, allowing thing* to be done in between (such as manipulating file descriptors). In Windows 2000, ihese two steps cannot he separated, that is. the concepts of making a new address space and filling it in are not orthogonal there. The Linux sequence of done plus exec is yet more orthogonal, since there are even more fine-grained building blocks available. As a general rule, having a small number of ortiogonal elements that can be com- combined in many ways leads t nail npt n eleg n > te 12.3.4 Naming Most long-lived data structures used by an operating system have some kind of name or identifier by which they can be referred. Obvious examples are login names,

file names, device names, process IDs, and so on. How these names are constructed and managed is an important issue in system design and lmplemenia- tion. Names designed for people to use are character-string names in ASCII or Unicode and are usually hierarchical. Directory paths, for example one such as /usr/astA>ooks/mos2/chap-I2, are clearly hierarchical, indicating a senes of direc- directories to search starting at the root. URLs are also hierarchical. For example,

src 12 i IMPLEMFMATION 873 www.ii.vu.nl/~aM/ indiaac u specific machine (hvvm) in a specific department <ci) at spectttc university (m) in J specific country («/). The pan after ihe slash indicates a specific file on the designated machine, in this case, by convention, *ww/index html in ast's home directory. Noie ihai URU (and DNS addresses in general including email addresses) are "backward." sianing ai the bottom of ihe tree and going up, unlike file rwmes, which start ai ihe lop ot the Iree and go down Another way ot looking at this is whether the tree is written from the top starling at the left and going right or sianing at ihe nghi and going lefi Ohen naming is done at two levels' txiernjl and itrtenva! For example, liSes always have a character-string nunie for people to use. In addition, there is almost always zn internal name thai the system uses In UNIX, ihe real name of a file is its i-node number; the ASCI name is noi used ai all internally. In tact, it is noi even unique since a file may have muliiple links 10 it. The analogous internal name in Windows 2000 is ihe file's index in the MFT. The job ot the directory Ls 10 provide the mapping between the external name and ihe internal name as shown in Fig. 12-3. »■ Directory /usr/asVbooiis/mos2 r ONO-11 Oiap-12 | ^ 3B xample given above), the internal n.nne is an unsigned integer (hat serves us an index into a kerne] table Other examples of table-index names are tile descriptors in UNIX and objecl handles in Windows 2000. Note that neither of (hese has any external representtion They are strictly for use by the system and running processes In general, using table indices for transient names that are lost when the system is rebooted is a good idea Operating systems often support multiple name spaces, holh external and internal For example, in Chap] I we looked ai three external name spaces sup- supported by Windows 2000; file names, object names, and regisiry names (and there is also the Active Directory name space, whieh we did not look at). In addition, there are innumerable internal name spaces using unsigned integers, for example, object handles, MFT entries, etc. Although the names in the external name spaces

874 OPERATING SYSTEM DESIGN CHAP. B are a]] Unicode sinngs, looking up a file name in the registry will not work, just as using an MFT index in the object table will not work. In a good design, consider- considerable ihought is given to how many names spaces are needed, what the syntax of names is in each one, how (hey can be told apart, whether absolute and relative names exist, and so on. 12.3.5 Binding Time As we have (us! seen, opeialiug systems use various kinds of names to refer !o ohjecit,. Sometimes ihe mapping helween a name and an object is fixed, but sometimes it is not. In Ihe latter case, it may matter when the name is bound to the objeci. In general, early binding is simple, but is no! flexible, whereas late binding is more complicated but often more flexible. To clarify the concep! of binding time, let us look at some real-world exam- examples An example of early binding is the praclice of some colleges to allow parents to enroll a baby at hirth and prepay (he curreni luition. When the siudem shows up 18 years later, the luition m fully paid up, no mailer how high it may be at lhat moment- In manufacturing, ordering parts in advance and maintaining an inventory of them is eariy binding, in contrast, jusl-in-iime manufacturing requires suppliers to be able to provide parts on the spot, with no advance notice required. This is late binding. Programming languages often support multiple binding times for variables. Global variables are bound to a particular virtual address by the compiler This exemplifies early binding. Variables local to a procedure are assigned a virtual address (on the stack) at the time the procedure is invoked. This is intermediate binding. Variables stored on the heap (those allocated by mallot in C or new in Java) are assigned virtual addresses only ai the time they are actually used. Here we have late binding. Operating systems often use early binding for most data structures, bui occa- occasionally use late binding for flexibility Memory allocation is a case in point. Early multiprogramming systems on machines lacking address relocation hardware had to load a program at some memory address and relocate it to run there. If it was ever swapped out, it had to he brought back at the same memory address or it would fail. In contrast, paged virtual memory is a form of laie bind- binding. The actual physical address corresponding to a given virtual address is not known until the page is touched and actually broughi into memory. Another example of late binding is window placement in a GUI. In contrast to the early graphical systems, in which the programmer had to specify the abso- absolute screen coordinates for all images on the screen, in modem GUJs, the software uses coordinates relative to the window's origin, but that is not determined until the

window is put on the screen, and it may even be changed later.

SEC- 12.3 IMPLEMENTATION » 12.3.6 Static versus Dynamic Structures Operating system designers are constantly forced to choose between static and dynamic data structures. Static ones are always simpler to understand, easier to program, and faster in use; dynamic ones are more flexible. An obvious example is the process table. Early systems simply allocated a fixed array of per-process structures. If the process table consisted of 256 entries, then only 256 processes could exist at any one instant. An attempt to create a 257th one would fail for lack of tabje space. Similar considerations held for the table of open files (both per user and system wide), and many other kerne] tables. An alternative strategy is lo build the process table as a linked list of minit- ables. initially just one. If this table fill!, up, another one is allocated from a glo- global storage pool and linked to the first one. In this way, the process table cannot fill up until all of kernel memory is exhausted. On the other hand, the code for searching the table becomes more compli- complicated. For example, the code for searching a static process table to see if a given PID, pid, is given in Fig. 12-4. It is simple and efficient. Doing the same thing for a linked list of minitables is more work. found = 0; for (p = &proc_table[0]; p < &proc_ table[PROC .TABLE _S12Ej; p++) < it (p->proc _pid = pid) [found = 1; break- Figure t2-4. Code for searching Lhe process (ab]e for a given PtD Static tables are bes! when there is plenty of memory or table ulilizalions can be guessed fairly accurately. For example, in a single-user system, it is unlikely that the user will start up more than 32 processes at once and it is not a total disas- disaster if an attempt to start a 33rd one fails. Yet another alternative is lo use a fixed-size table, but if it fills up, allocate a new fixed-size table, say, twice as big. The current entries are then rapied over to the new table and the old table is returned to the free storage pool. In this way, the table is always contiguous rather than linked. The disadvantage here is that some storage management is needed and the address of the table is now a variable instead of a constant. A similar issue holds for kernel stacks. When a thread switches 10 kernel mode, or a kemel-mode thread is run, it needs a stack in kernel space. For user threads, the slack can be initialized to nin down from the top of the virtual address space, so the size need not be specified in advance. For kernel threads, the size must be specified in advance because the stack takes up some kernel virtual

876 OPERATING SYSTEM DESIGN CHAP.]2 address space and there may be many slacks. The question is; how much space should each one gel? The trade-offs here are similar to those for the process table. Another static-dynamic trade-off is process scheduling. In some systems, especially real-time ones, the scheduling can be done statically in advance For example, an airline knows what time its flights will leave weeks before their departure. Similarly, multimedia systems know when to schedule audio, video, and olher processes in advance. For genera]-purpose use. these considerations do not hold and scheduiing muse be dynamic. Yet another static-dynamic issue is kernel structure. It is much simpler if the kernel is built as a single binary program and loaded into memory to run The consequence of this design, however, is that adding a new I/O device requires a relinking of the kernel with the new device driver. Early versions of UNIX worked this way, and it was quite satisfactory in a minicomputer environment when adding new I/O devices was a rare occurrence. Nowadays, most operating systems allow code to be added to the kerne] dynamically, with al] the additional complexity that entails.

12.3.7 Top-Down versus Bottom-Up Implementation While it is bes,t to design the system top down, in theory it can be imple- implemented top down or bottom up. In a top-down implementation, the implemented start with the system call handlers and see what mechanisms and d"ta structures are needed to suppon them. These procedures are written and so on until the hardware is reached. The problem with this approach is that it is hard to test anything with only the top-level procedures available For this rea->on. many developers find it more practical to actually build the system bottom up. This approach entails first writ- writing code that hides the low-]evel hardware, essentially the HAL in Fig II-7. Interrupt handling and the clock driver are also needed early on Then multiprogramming can be tackled, along with a simple scheduler (eg., round-robin scheduling). At this point it should be possible to test the system to see if it can run multiple processes correctly. If that works, it is now time to begin the careful definition of the various tables and data structures needed throughout the system, especially those for process and thread management and later memory management. I/O and the file system can wait initially, except for a primitive way to read the keyboard and write to the screen tor testing and debugging. In some cases, the key]ow-]evel data structures should be protected by allowing access only through specific access procedures—in ettect, object-on en ted programming, no matter what the programming language is. As lower layers are completed, they can be tested thoroughly. In

this way, the system advances from the bottom up, much the way contractors build tall office buildings If a large team is available, an alternative approach is to first make a detailed design of the whole system, and then assign different groups to write different

SEC 12? 1MPLFMFNTATI0N 877 modules. Each one tests, its own work in isolation. When all the pieces arc ready, they are integrated and tested. The problem with this line of attack is that if noth- nothing" works initially, it may be hard to isolate whether one or more modules are malfunctioning, or one group misunderstood what some other module was sup- supposed to do. Nevertheless, with large teams, this approach is often used to max- maximize ihe amouni of parallelism in the programming effort 12,3.8 Useful Techniques We have just looked at some abstraci ideas for system design and implemen- implementation. Now we wi]] examine a number oi useful concrete techniques for system implementation. There are numerous other ones, of course, but space limitations restrict us' to just a few of them Hiding the Hardware A lot of hardware is ugly. It has to be hidden early on (unless it exposes power, which most hardware does not). Some ot the very low-]eve] details can be hidden by a HAL-type layer of the type shown in Fig. 12-2. However, many hardware details cannot be hidden this way. One thing (hat deserves early attention is how to deal with interrupts. They make programming unpleasant, but operating systems have to deal with them. One approach is to turn them into something else immediately. For example, every interrupt could be turned into a pop-up thread instantly. At that point we are dealing with threads, rather than interrupts A second approach is to convert each interrupt into an unlock operation on a mutex that the corresponding driver is waiting on Then the only effect of an interrupt is to cause some thread to become ready. A third approach is convert an interrupt into a message to some thread. The low-level code just builds a message telling where the interrupt came from, enqueues it, and calls the scheduler to (potentially) run the handler, which was probably blocked waiting for the message AH these techniques, and other ones like them, all try to convert interrupts into thread synchronization operations. Having each interrupt handled by a proper thread in a proper context is easier to manage than running a handler in the arbitrary context that it happened to occur in. Of course, this must be done efficiently, but deep within the operating system, everything must be done efficiently. Most operating systems are designed to run on multiple hardware platforms. These platforms can differ in terms of the CPU chip,

MMU, word length, RAM size, and other features that cannot eaKily be masked by the HAL or equivalent. Nevertheless, it is highly desirable to have a single set of source files that are used to generate all versions; otherwise each bug that later turns up must be fixed mul- multiple times in multiple sources, with the danger that the sources drift apart

878 OPERATING SYSTEM DESIGN CHAP. 12 Some hardware differences, such a.s RAM size, can be dealt with by having the operating system determine the value at boot time and keep it in a variable. Memory allocators, for example, can use the RAM size variable to determine how big to make the block cache, page tables, etc. Even static tables such a;, the proc- process table can be sized based on the total memory available. However, other differences, such as different CPU chips, cannot be solved by having a single binary that determines at run time which CPU it is running on. One way to tackle the problem of one source and multiple targets is to use condi- conditional compilation. In the source files, certain conipile-time flags arc defined for the different configurations and these are used to bracket code that is dependent on the CPU, word length, MMU, etc. For example, imagine an operating system that is to run on the Pentium and UltraSPARC chips, which need different initiali- initialization code. The mil procedure could be written as illustrated in Fig. 12-5(a). Depending on the value of CPU, which is defined in the header file config.h, one kind of initialization or olher is done. Because the actual binary contains only the code needed for the target machine, there is no loss of efficiency this way. #include "config.h" #inciude "config.h" init() #if (WORD .LENGTH =- 32) { typedef int Register; #if (CPU == PENTIUM) #endif /• Pentium initialization here. */ #endif #if (WORD .LENGTH == 64) typedef long Register; #if (CPU == ULTRASPARC) #endif /* UilraSPARC initialization here. ■/ #endif Register RO, B1, R2, R3, Figure 12-5. (a) CPU-dependsnt conditional compilation |bi Word-ienglh impendent conditional compilation As a second example, suppose there is a need for a data type Register, which should be 32 bits on the Pentium and 64 bits on the UltraSPARC This could be handled by the conditional code of Fig. l2-5(b> (a suming that the compiler pro- produces 32-bit ints and 64-bit longs). Once this definition ha been made (probably in a header file included everywhere), the programmer can ju t declare variables to be of type Register and know they will be the right length The header file, config.h. has to be defined correUly, oi course For the Pen- Pentium it might be something like this: #define CPU

PENTIUM #define WORD.LENGTH 32

SEC 121 IMPLEMENTATION 879 To compile (he sysiem tor the UltraSPARC, a different ,onfif>£ would be used, with (he correct values tor (he UltraSPARC, probably something like #defme CPU ULTRASPARC #define WORD LENGTH 64 Some readers may be wondering why CPU and WORD .LENGTH are han- handled by different macros We could easily have bracketed (he definition ot'flegu- ici with a test on CPU, setting it to 32 bits for (he Pentium and 64 bits for the UltraSPARC. However, this is not a good idea. Consider what happens when we later port the system to (he 64-bi(ln(cl l(anmm. We would have to add a third conditional to Fig. !2-5(b) for the Itanium. By doing it as we have, all we ha\e to do is include the line #define WORD .LENGTH 64 to the config.h file for the I an n This example illustrates the orthogonality principle we discussed earlier Those items that are CPU-dependent should be conditionally compiled based on the CPU macro and those things that are word-length dependent should use the WORD ^LENGTH macro. S n I on de n hold for many other parameters. Indirection It is sometimes said that there is no problem in compmer science that cannot be solved with another level of indirection. While something of an exaggeration there is definitely a grain of truih here. Let us consider some examples. On Pentium-based systems, when a key is depressed, the hardware generates an mttr nipt and puts the key number, rather than an ASCII character code, in a device register, Furthermore, when the key is released later, a second interrupt is gen erated, also with the key number. This indirection allows the operating system (he possibility of using the key number to index into a table to get the ASCII charac- character, which makes it easy to handle the many keyboards used around the world in different countries. Getting both the depress and release information makes it possible to use any key as a shift key since the operating system knows the exact sequence the keys were depressed and released. Indirection is also used on output. Programs can write ASCII characters to the screen, but these are interpreted as indices in(o a table for the current output font. The table entry contains the bitmap for (he character. This indirection makes it possible to separate characters from fonts. Another example of indirection is the use of major device numbers in UNtX. Within the kernel there is a table indexed by major device number for the block devices and another one for the character devices. When a process opens a s~pe- cial file such as /dev/hdO, the system extracts the type (block or character) and

3$Q OPERATING SYSThM DESIGN CHAP 12 major and minor device numbers from the i-node and indexes into the appropriate driver table to find the driver. This indirection makes it easy to reconfigure the system, because programs deal with symbolic device names, not actual driver names Yet another example of indirection occurs in message-passing systems that name a mailbox rather than a process as the message destination. By indirecting through mailboxes (as opposed to naming a process as the destination), consider- considerable flexibility can be achieved (e.g.. having a secretary handle her boss1 mes- messages). In a sense, the use of macros, such as #define PROC. TABLE .SIZE 256 is alt,o a form of indirection, since the programmer can write code without having to know how big ihe table really is. It is good practice to give symbolic names to all constants (except sometimes -I, 0, and i). and put these in headers with com- comments explaining what they are for- Reusability It is frequently possihle to reuse the same code in slightly different contexts. Doing so is a good idea as it reduces the size of ihe hinary and means (hat the code has to be debugged only once. For example, suppose that hitmaps are used to keep track of free blocks on the disk Disk block management can be handled by having procedures alloc and free that manage the bitmaps As a bare minimum, these procedures should work for any disk. But we can go further than that. The same procedure- can also work tor managing memory blocks, blocks in the file system's block cache, and i-nodes In fact, they can be used to allocate and deallocate any resources that can be numbered linearly, Reentrancy Reentrancy refers for the ability of code to be executed two or more times simultaneously. On a multiprocessor, there is. always the danger than while one CPU is executing some procedure, another CPU will start executing it as well, before the first one has finished. In this case, two (or more) threads on different CPUs might be executing the same code at ihe same time. This situation must bo protected against by using mutexes or Hime other means to protect critical regions. However, the problem also exists on a uniprocessor In particular, most of any operating system runs with interrupts enabled. To do otherwise, would lose many interrupts and make the system unreliable While the operating system is busy executing some procedure, p. it is entirely possible that an interrupt oc-curs and that the interrupt handler also calls p. If the data structures of P were in an

SEC. 12.3 IMPLEMENTATION 881 inconsistent state at the time of (he interrupt, the handler will see them m an inconsistent state and tail. An obvious

example where this can happen is if P is the scheduler. Suppose that some process used up its quantum and the operating system was. moving it to the end ot its queue. Part way through the list manipulation, the interrupt occurs, makes some process ready, and runs the scheduler With <he queues in an incon- inconsistent stale, the system will probably crash. As. a consequence even on a unipro- uniprocessor, it is best (hat most ot ihe operating sysiem is reentrant, critical data struc- structures are protected by miitexcs. and interrupis are disabled at moments when they cannot be ioferated Brute Force Using brute force to solve a problem has acquired a bad name over the years, but it is often the way to go in the name of simplicity Every operating system has many procedures that aie rarely called or operate with so little data that optimiz- optimizing them is not worthwhile. For example, it is frequently necessary to search vari- various tables and arrays within the sysiem. The brute force algorithm is |ust leavt ihe table in the order the entries are made and search it linearly when something has to be looked up. If ihe number of entries is small (say, under 100), the gain from sorting the table or hashing it is small, but the code is far more complicated and more likely to have bugs in it. Of course, for funciions thai are on ihe enncai paih. say. contexl switching, everything should be done to make them tasl. possibly even wniing them in (heaven forbid) assembly language. But large parls of the system are not on the critical path. For example, many sysiem calls are rarely called. If then, is ont tork every 10 sec, and it takes 10 msec to carry out, then even optimizing it 10 0 wins only 0.1 %. if the optimized code is bigger and buggier, a case can be m;ide not to boiher with the optimization Cbeck for Errors First Many system calls can poteniiaiiy fail for a vanely of reasons: the file to be opened belongs to someone else; process creation fails because the process iabje is full; or a signal cannot be sent because the target process does not exist, The operating system must painstakingly check for every possible error before carry- carrying out the call. Many system calls also require acquiring resources such as process table slots, i-node table slots, or file descriptors A general piece of advice that can save a ioi of grief is to first check to see if the system call can actually he carried oui before acquiring any resources This means putting all the tests at the beginning of the procedure that executes the system call. Each test should be of the form it (error .condition) return(ERROR _CODE);

 If the call gets all the way through the gauntlet ot tests, then it is certain that it will succeed. At that point

resources can be acquired. Interspersing the lets with resource acquisition means that if some lest fails ;i(ong the way. all the resources acquired up to that point must be returned. If an error k made here and some resource is not returned, no damage ts done immedi- immediately. For example, one process table cniry may just become permanently una- unavailable However, over a period of time, this bug may be triggered multiple times Eventually, most or all the process tahle entries may become unavailable, leading 10 a system crash in an extremely unpredictable and difficult to debug WdJ Many sj stems sutler trom this problem in the form of memory leaks. Typi- Typically, the program calls malloc to allocate space bui forgcis to call free later to re- release it Ever so gradually, all of memory disappears until ihc system is rebooted. Engler et al- B000) have proposed an interesting way to check for some of these errors at compile time. They observed that the programmer knows many invariants that the compiler does not know, such as when you lock a mutex, ail paths starting at ihe lock must contain an unlock and no more locks of the same mulex. They have devised a way tor the programmer to tell the compiler this fact and instruct it to check all the paths at compile time for violations of the invariant. The programmer can also specify thai allocated memory muit be released on all paths and many other conditions as well. 12.4 PERFORMANCE All things being equal, a fast operating system is better than a slow one. However, a fast unreliable operating system is not as good as a reliable slow one Since complex optimizations often lead to bugs, it is important to use them spar- sparingly. This notwithstanding, there are places where performance is critical and optimizations are well worth the effort. In (he following sections, we will look at some general techniques that can be used to improve performance in places where lhat is called for. lZ4.t Why Are Operating Systems Slow? Before talking about optimization techniques, it is worth pointing out that the slowness of many operating systems is to a large extent self-inflicted. For exam pie, older operating systems, such as MS-DOS and UNIX Version 7, booted within a few seconds. Modem UNIX systems and Windows 2000 can take minutes to boot, despite running on hardware that is IO() times as fast. The reason is that they are doing much more, wanted or not. A case in point. Plug and play makes it somewhat easier to install a new hardware device, but the price paid is that on

 every boot, the operating system has to go out and inspect ail the hardware to see if there is anything new out there. This

bus scan takes time. An alternative (and in the author's opinion, better) approach would be to scrap plug-and-play altogether and have an icon on the screen labeled "Install new hardware.'" Upon installing a new hardware device, the user would click on this icon to start the bus scan, instead of doing it cm every boot. The designers of current systems were well aware of this option, of course. They rejected it, basi- basically, because they assumed that the users were too stupid to be able to do this correctly (although they would word it more kmdiy). This is only one example, but there are many more where the desire to make the system "user-lriendly" (or ■"idiot-proof," depending on your viewpoint) slows the system down all the time for everyone. Probably the biggest single thing syslem designers can do to improve perfor- performance is to he much more selective about adding new features. The question to ask is notr "Will the users like this?" but "Is this feature worth the inevitable perience shows this view to he a bit optimistic. Another factor that plays a role is product marketing. By the time version 4 or 5 of some product has hit the market, probably all the features that are actually useful have been included and most of the people who need the product already have it. To keep sales going, many manufacturers nevertheless continue to pro- produce new versions, with more features, just so they can sell their existing custo- customers upgrades. Adding new features just for the sake of adding new features may help sales but rarely helps performance. 124.2 What Should Be Optimized? As a general rule, the first version of the system should he as straightforward as possible. The only optimizations should be things that are so obviously going to he a problem that they are unavoidable. Having a block cache for the file sys- system is such an example. Once the system is up and running, careful measure- measurements should he made to see where the time is really going. Based on these numhers. optimizations should he made where they will help most. Here is a true story of where an optimization did more harm than good. One of the author's students (who shall here remain nameless) wrote the MJNJX tnkfi program. This program lays down a fresh file system on a newly formatted disk. The student spent about 6 monihs optimizing it, including putting in disk caching. When he turned it hi, it did not work and it required several additional months of debugging. This program typically runs once on the hard disk during the life of the computer, when the system is installed. It also runs once for each floppy disk that is formatted. Each ran takes about 2 sec. Even if the unopiimized version

884 OPERATING SYSTEM DFSfGN CHAP 12 had taken 1 minute, it was a poor use of resources to spend so much time optimiz- optimizing a program mat is used so infrequently A slogan thai has considerable applicability to performance optimization is Good enough ii good enough. By this we mean that once performance has achieved a reasonable level, it is probably not worth the effort and complexity to squeeze out the last few percent. If the scheduling algorithm is reasonably fair and keeps the CPU busy 90% of the time, it is doing its job. Devising a far more complex one that is 5% hetter is probably a bad idea. Similarly, if the page rate is low enough that it k not a bot- bottleneck, jumping through hoops to get optimal performance is usually not worth it. Avoiding disaster is far more important than getting optimal performance, espe- especially since what is optimal with one load may not be optimal with another- 12.4.3 Space-Time Trade-offs One general approach to improving performance is to irade off time versus space. Jt frequently occurs in computer science that there is a choice between an algorithm that uses little memory but is slow and an algorithm that uses much more memory but is faster. When making an important optimization, it is worth looking for algorithms that gain speed by using more memory or conversely save precious memory by doing more computation One technique that is often helpful is to replace small procedures by macros. Using a macro eliminates the overhead normally associated with a procedure call. The gain is especially significant if the call occurs inside a loop. As an example, suppose we use bitmaps to keep track of resources and frequently need to know how many units are free in some portion of the bitmap. For this purpose we need a procedure, t>it_couni, that counts the number of I bits in a byte. The straightfor- straightforward procedure is given in Fig. l2-6(a). It loops over the bits in a byte counting them one at a time. This procedure has two sources of inefficiency First, it must be called, stack space must be allocated for it, and it mUst return. Every procedure call has this overhead. Second, it contains a loop, and there is always some overhead associ- associated with a Joop. A completely different approach is to use the macro of Fig. i2-6(b). It is an inline expression (hat computes the sum of the bits by successively shifting the argument, masking out everything but the low-order bit, and adding up the eight terms. The macro is hardly a work of art, but it appears in the code only once. When the macro is called, for example, by sum = bit_count(table[i]), the macro call looks identical to the call of the procedure. Thus other than one somewhat messy definition, the code does not look any worse in the macro case

#define BYTE^SlZE 8 /'A byte contains 8 Nts 'I mi bit-count(ini byie) (/* Count the bus in a byte. */ int i, coum - 0, for (i = 0-1< BYTE^SIZE; i++) /* loop over lhe bits in a byte */ i! ((byte >? i) & i)count++, /* if lhis bit is a 1, add to count*/ return(count). /* return sum */ /•Macro to add up trie bus in a byie and return ihe sum. #definebiccouni(b)(b&l) + ((b»D&1) + ((b»2)&f) + ((b4&1) ((b5)&1) ((b6 /*Macro to rook up ihe bit count in a table. V char bits[256] = {0, 1, 1. 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4, 1, 2, 2, 3, 2, 3, 3, . .); #define bit^count(b) (int) bits[b] than in the procedure case, but ii is much more efficient since n eliminates both ihe procedure call overhead and the loop overhead. We can take ihis example one step further. Why compute the hi count at ail" Why not look it up in a table? After all, there are only 256 different bytes, each with a unique value between 0 and 8. We can declare a 256-entry lable, hits, with each entry initialized (at compile time) to the bit count corresponding to that byte value. With this approach no computation at all is needed at run Ume, just one indexing operation. A macro to do the job is given m Fig. J 2-6(c). This is a clear example of trading computation time against memory How- However, we could go still further. If the hit counts for whoje 32-bit words are needed, using our bit. count macro, we need to perform four lookups per word. If we expand the table to 65.536 entries, we can suffice with two lookups per word, at the price of a much bigger table. Looking answers up in tables can be used in other ways. For example, in Chap. 7 we saw how JPEG image compression works, with fairly complex dis- discrete cosine transformations. An alternative compression technique. GIF, uses table lookup to encode 24-bit RGB pixels. However, GIF only works on images with 256 or fewer colors. For each image to be compressed, a palette of 256 en- entries is constmcted, each entry containing one 24-bit RGB value. The compressed

OPERATING SYSTEM DESIGN CHAP. image then consists of an 8-bit index for each pixel instead ot .1 24-bit color value, a gain of a factor of three This idea is illustrated for u 4 x 4 section of an image in Fig. 12-7. The original compressed image u shown in Fig l2-7(a) Each value here is a 24-bit value, with 8 bits each giving the intensity for red. green, and blue The GIF image is shown in Fig. |2-7(b). Here each value is an 8-bit index into the color palette. The color palette is stored as part of the image file, and is shown in Fig. I2-7(C). Actually, there is more to GIF, but the core of the idea is table lookup. 8 Bus 3,8 13 3.8.13 4,6,9 10,11,5 3,8,13 3*13 10 30,8 4,2.17 36,4,9 4.19,20

5,8,1 W..3 90,2 6 4,6,9 22.2,0 66,4,43 7 < a r 7 5 9 2 3 10 2 6 - 0 Figure 12-7. (a) Pan of an uncompressed image with 24 bits per pixel (b) The same pan compressed with GIF, with 8 bus per pixel. (l) The color paleue There is another way to reduce image size, and it illustrates a different trade- tradeoff. PostScript is a programming language that can be used to describe images. (Actually, any programming language can describe images, but PostScript is tuned for this purpose.) Many printers have a PostScript interpreter built into them to be able to run PostScript programs sent to them. For example, if there is a rectangular block of pixels all the same color in an image, a PostScript program for the image would carry instructions to place a rec- rectangle at a certain location and fit! it with a certain color. Only a handful of bits are needed to issue this command. When the image is received at the printer, an interpreter there must ran the program to construct the image. Thus PostScript achieves data compression at the expense of more computation, a different trade- tradeoff than table lookup, but a valuable one when memory or bandwidth is scarce Other trade-offs often involve data structures. Doubly-jinked lists take up more memory than singly-linked lists, but often allow faster access to items- Hash tables are even more wasteful of space, but faster still. Jn short, one of the main things to consider when optimizing a piece of code is whether using dif- different data structures would make the best time-space tradeoff.

SEC 12 4 PERFORMANCE 887 12.4.4 Caching A well-known technique for improving performance is, caching. It is applica- applicable any time it is likely the same result will be needed multiple times. The gen- general approach is to dJHo the full work [he first time, and then save the result in a cache On subsequent attempts, the cache is first checked If the result is there, it is used. Otherwise. ;he full work is done again We have already seen the use of caching within the file system to hold iome number ot recently used disk blocks, thus saving a disk read on each hit. How- However, caching can be used for many other purposes as well- For example, parsing path names is surprising expensive. Consider the UNtX example of Fig. 6-39 again. To look up/usr/ast/mbox requires tbe following disk accesses: I. Read the i-node for the root directory (i-node I) 2 Read the root directory {block !) 3 Read the i-node fat/usr (i-node 6). 4. Read the /usr directory {block 132). 5. Read the i-node for /usr/asi (i-node 26). 6. Read the /usr/ast directory (block 406). It takes six disk accesses just to discover the i-node number of tbe file. Then the

i-node itself has to be read to discover the disk bjock numbers, Jf the file is smaller than the block size (e.g., 1024 bytes), it takes eight disk accesses to read the data. Some syslems optimize path name parsing by caching (path, i-node) combina- combinations. For the example of Fig. 6-39, the cache wi!l certainly hold the first three entries of Fig. 12-8 after parsing /usr/ast/mbox. The last three entries come from parsing other paths. Figure 12-8. Part oi the i-n,)de cache fur Fig 0-^ When a path has to be looked up, the name parser first consults the cache and arches it for the longest substring present in the cache. For example, if the path

jj88 OPERATING SYSTEM DESIGN CHAP. 12 /w,r/atf/gHtui\/stw is presented, the cache returns tbe tact that /usr/cut is i-node A problem with caching paths is that the mapping between file name and i- node number is not fixed for all time. Suppose that the file /uir/usl/>nbox is lemoved trom tbe system and its i-node reused for a different file owned by a dif- lerenl user. Later, the file /usr/a.st/mbox is created again, and tbis time it gets j- node 106 If nothing is d,)nc lo prevent it. the cache entry Will now be wrong and subsequent lookups will lcturn the wrong (-node number For this reason, wben a tile or directory is deleted, its cache en(ry and (it it is a directory! all the entries below it must be purged tiom the cache. Disk blocks and path names are not the only items (hat can be cached, l- nodes can be cached, too It pop-up threads are used to handle interrupts, each (me of them requires a sl.ick and some additional machinery These previously- used threads can also be (.ached, since refurbishing a used one is easier than creat- creating a new one from scratch (to avoid having to allocate memory) Just about any- anything that is hard to produce can he cached 12.4.5 Hints Cache entries are always correct A cache search may fail. bu(if it finds an entry, tliat entry is guaranteed to be correct and can be used without further ado In some systems. i(is. convenient to have a table ot hints. These are suggestions about the solution, but they are not guaranteed (o be correct. The caller must ver- verify the result uselr. A well-known example of hints are the URLs embedded on Web pages. Clicking on a link does no(guarantee that the Web page pointed to is there. In fact, the page pointed ;o may have been remo\ed years ago. Thus the information on the pointing page is really on!y a hint Hints are also used in connection with remote tiles The information is the hin(tells something ahou((he remote file. s,uch as where n is located. However, the file may have moved or heen deleted smce the bmi was recorded, so a check is always needed to see ll it n, correct-

12.4.6 Exploiting Locality Processes and programs do not act at rand locality in time and space, and this infomiatio to improve performance. One we!!-known ex. (ha(processes do no(jump around at random tend td use a relatively small number of pages pages that a process is actively using can be noted as its working set and the operating system can make sure that when the process is allowed to run, its work- working set is in memory, thus reducing the number of page faults. pl< th They i be e? :of sj in tbej neap exh,bi(a t (ploued in lana! local ir address liven (ime air va ■ity SDE int. amount of nous ways is the fact [ces. They erval. The

SEC 12.4 PERFORMANCE ™=* The locality principle also holds for files. When a process has selected a par- licujar working directory, it is likely that many of its future file references will be to files in that directory. By putting ail the i-nodes and files for each directory close together on the disk, performance improvements can be obtained. This prin- principle is what underlies the Berkeley Fast File System (McKusick et al., 1984). Another area in which locality plays a role is in thread scheduling in multipro- multiprocessors. As we saw in Chap. 8, one way to schedule threads on a multiprocessor is to try ;o run each thread on the CPU it las; used, in hopes that some of its mem-memory blocks will still be in the memory cache. 12.4.7 Optimize the Common Case ft is frequently a good idea to distinguish between the most common case and the worst possible case and treat them differently. Often (he code for the two is quite different. It is important to make the common case fast. For the worst case, if it occurs rarely, it is sufficient to make it correct. As a first example, consider entering a critical region. Most of the time, the entry will succeed, especially if processes do not spend a lot of time inside critical regions. Windows 2000 takes advantage of this expectation by providing a Win32 API call EnterCriticalSectlon that atomically tests a flag in user mode (using TSL or equivalent). If the test succeeds, the process just enters the critical region and no kernel call is needed. If the test fails, the library procedure does a down on a semaphore to block the process. Thus in the normal case, no kernel call is needed. As a second example, consider setting an alarm (using signals in UNIX), If no alarm is currently pending, il is straightforward to make an entry and put it on the timer queue. However, if an alarm is already pending, it has to be found and removed from the limer queue. Since the alarm call does not specify whether (here is already an alarm set, the sysiem has to assume worst case, that there is. However, since most of the lime

there is no alarm pending, and since removing <tn existing alarm is expensive, it is a good idea to distinguish these two cases. One way to do this is to keep a bit in the process table that lells whether an alarm is pending, if the bit is off, the easy path is followed (just add a new timer queue entry without checking). If the bit is on, the timer queue must be checked. 12.5 PROJECT MANAGEMENT Programmers are perpetual oplimisrs. Most of them think that the way 10 write a program is to ran to the keyboard and start typing. Shortly thereafter the fuily debugged program is finished. For very large programs, it does not quite work like that, In the following sections we have a bit to say about managing large software projects, especially large operating sysiem projects.

890 OPERATING SYSTEM DESIGN CHAP. 12 125.1 The Mythical Man Month In his classic book, Fred Brooks, one of the designers of OS/360, who later moved to academia, addresses the question of why it is so hard to build fug operating systems (Brooks, 1975, 1995). When most programmers see his claim that programmers can produce only 1000 lines of debugged code per year on large projects, they wonder whether Prof. Brooks is living in outer space, perhaps on Planet Bug. After all, mosi of them can remembher an all nighter when they pro- produced a 1000-line program in one night. How could this he the annual output of anybody with an]Q > 50? What Brooks pointed out is that large projects, with hundreds of program- programmers, are completely different than small projects and that the results obtained from small projects do not scale to large ones, in a large project, a huge amount of time is consumed planning how to divide the work into modules, carefully specifying the modules and their interfaces, and trying to imagine how ihe modules will interact, even hefbre coding hegins. Then the modules have to he coded and debugged in isolation. Finally, the modules have to he integrated and system as a whole has to he tested. The normal case is that each module works perfectly when tested by iiself, but the system crashes instantly when all the pieces are put together. Brooks estimated the work as heing 1/3 Planning 1/6 Coding 1/4 Module testing 1/4 System testing In other words, writing the code is the easy part. The hard part is figuring out what the modules should he and making module A correctly talk to module B. In a small program written by a single programmer, all that is left over is the easy part. The title of Brooks' book comes from his assertion that people and time are not interchangeable. There is no such unit as a man-month (or a person-month), if a project takes 15 people 2

years to build, it is inconceivable that 360 people could do ii in one month and probably not possible to have 60 people do it in 6 months. There are three reasons for this effect. First, the work cannot he fully paral- parallelized. Until the planning is done and it has been determined what modules are needed and what their interfaces will he, no coding can he even started, On a two-year project, the planning alone may take 8 months. Second, to fully utilize a large numher of programmers, the work must he par- partitioned into large numhers of modules so everyone has something lo do. Since every module may potentially interact with every other module, the numher of module-module interactions that need to he considered grows as the square of the numher of modules, that is, as the square of the number of programmers. This

SEC 12 5 PROJECT MANAGEMENT »9l complexity quickly gets out of hand. Careful measurements of 63 software pro- projects have confirmed thai the trade-off between people and months is far from linear on large projecls (Boehm, 1981). Third, debugging is highly sequential. Selling ten debuggers on a problem doe-, not find the bug len times as fast, in fact, ten debuggers are probably slower lhan one because they will waste so much time talking to each other. Brooks sums up his experience wilh trading-off people and time in Brooks' law. Adding manpower to a late software prvject makes it later. The problem with adding people is that they have to he trained in the project, ihe modules have to be redivided to match ihe larger numher of programmers now available, many meetings will be needed to coordinate all the efforts and so on. Abdel-Hamid and Madmck A991) confirmed this law experimentally. A slightly irreverent way of restating Brooks law is It lakes 9 months to bear a child, no matter how many women you assign to the job. 12.5.2 Team Structure Commercial operating systems are iarge software projects and invariably require large teams of people. The quality of the people mailers immensely. It has heen known for decades that top programmers are lOx more productive than bad programmers (Sackman et al., 1968). The trouble is, when you need 200 pro- programmers, it is hard to find 200 lop programmers; you have to settle for a wide spectrum of qualities. What is also important in any large design project, software or otherwise, is the need for architectural coherence. There should he one mind controlling the design, Brooks ciles the Reims cathedral as an example of a large project that took decades to build, and in which the architects who came later subordinated their desire to put their stamp on the project lo carry out Ihe initial architect's plans. The result is an

architectural coherence unmatched in other European cathedrals. In the 1970s, Harlan Milis combined the observation that some programmers are much better than others with the need for architectural coherence to propose the chief prognunmer team paradigm (Baker, 1972). His idea was to organize a programming team like a surgical team rather than like a hog-butchering team. Instead of everyone hacking away like mad, one person wields the scalpel- Everyone else is there to provide support. For a]0-person project, Mills sug- suggested the team structure of Fig. 12-9. Three decades have gone by since this was proposed and put into production. Some things have changed (such as the need for a language iawyer—C is simpler

OPERATING SYSTEM DESIGN Tft]e 9}*ieB. - — — - - 1 Chief programmer T~Performs Ihe architectural design and writes Ihe code i "copilot _ j_!J5iP? tne ch'el Programmer and serves as a sounding board j ^Administrator I Manages the people, budget, space, equipment, reporting, etc . 1 Editor ' Edits the documentation, which must be written by the chie] programmer I Secretaries The administrator and editor each need a secretary _ I | Program cierk f Maintains the code and documentation archives I Toolsmith , Provides any tools the chief programmer needs ! .Tester I Tests the chief programmer's code | \ Language lawyer \ Part timer who can advise the chief programmer on the language Figure 12-9. Mills" proposal for populating a 10- perw chief programmer Ihan PL/1), but the need to have only one mind conn oiling the design is still true. And ihai one mind should be able io work 100 ^ on designing and programming, hence ihe need for ihe support staff, although with help from the computer, a smaller staff will suffice now. But in its essence, ihe idea ts still valid. Any large project needs to he organized as u hierarchy. At lhe bottom level are many small teams, each headed by a chief programmer. At the next level, groups of teams must be coordinated by a manager. Experience shows that each person you manage costs you 10% of your time, so a rull-time manager is needed for each group of 10 teams. These managers must be managed, and so on up the tree. Brooks observed that bad news does not travel up the tree well Jerry Saltzer of M.i.T. called this effect the had-news diode. No chief programmer or manager wants to tei! his boss that the project is 4 months late and has no chance whatsoever of meeting the deadline hecause there is a 1000-year old tradition of heheading the messenger who bnng.s bad news. As a consequence, top manage- management is generally in the dark about the state of the project. When it become;, obyiou;. that the

deadline cannot be met, top management responds by adding peo- people, at which time Brooks' law kicks in. in practice, large companies, which have had long experience producing software and know what happens if it is produced haphazardly, have a tendency to at least try to do it right, in contrast, smaller, newer companies, which are in a huge rush to get to market, do not always take the care to produce their software carefully. This haste often leads to far from optimal results. Neither Brooks nor Mills foresaw the growth of the open source movement. Although it has had some successes, (t remains to be seen if this is a viable mode! for producing large amounts of quality software once the novelty wears off. Recall that m iis early days, radio was dominated by ham radio operators, but that

SEC PROJECT MANAGEMENT 893 soon gave way lo commercial radio and later to commercial television. What is noticeable is that the open source software projects that have been most successful have clearly used the chief programmer model of having one mind control (he architectural design (e,g, Linus Torvalds for the Linux kernel and Richard Stall- man for the GNU C compiler). 12.5.3 The Role of Experience Having experienced designers is critical to an operating systems project. Brooks points out thai mo.st of the errors are not in the code, but in the design. The programmers correctly did whal ihey were told to do. What they were told to do was wrong. No amount of tesl software will catch bad specifications. Brooks' solution is to abandon the classical development model of Fig. 12- 10fa> and use the model of Fig. 12-10(b), Here the idea is to first write a main program thai merely calls the top-level procedures, which are initially dummies. Starting on day 1 of the project, the system will compile and run, although it does nothing. As time goes on, modules are inserted inio the fuil system. The result of ihis approach is that sysiem integration lesting is performed continuously, >i> errors in the design show up much earlier. In effect, the learning process caused by bad design decisions starts much earlier in (he cycle. Procedure 1 / Main program procedure 2 Dummy procedure 3 Figure 12-10. (a) Traditional software design progresses in stages (bl Alterna- Alternative design produces i working sysiem (dial does nothing) starting on day I A little knowledge is a dangerous (hing. Brooks observed what he called the second system effect. Often the first product produced by a design team is minimal because the designers are afraid it may not work at all. As a result, they are hesitant to put in many features, it the project succeeds, they build a follow- up system. Impressed by their own

success, the second time the designers include all ihe bells and whistles that were intentionally ieft out the first time, As a result,

894 OPERATING SYSTEM DESIGN CHAP 12 ihe second system is bloated and performs poorly. The third time around they are sohered by the failure of Ihe second syslem and are cautious again. The CTSS-MULTICS pair is a clear case in poini. CTSS was the first general- purpose timesharing system and was a huge success despite having minimal func- ijonalily. iis successor, MULT1CS was too ambitious and suffered badly for it. The ideas were good, bul (here were loo many new things so (he syslem per- performed poorly for years and was never a big commercial success. The third sys- system in this line of development, UNIX, was much more cautious and much more successful. 12.5.4 No Silver Buiiet In addition to The Mythical Man Month, Brooks also wrote an influential paper called "No Siiver Buiiet" (Brooks, 1987), in it, he argued thai none of the many nostrums heing hawked by various peopie at the time was going lo generate an order-of-magnitude improvement in software productivity within a decade. Experience shows that he was right. Among the siiver bullets thai were proposed were betler high-ievel languages, object-oriented programming, artificial intelligence, expert systems, automatic programming, graphical programming, program verification, and programming environments. Perhaps the next decade will see a silver bullet, but mayhe we wiii have to settie for gradual, incremental improvements. 12.6 TRENDS IN OPERATING SYSTEM DESIGN Making predictions is always difficult—especially about the future. For example, in i899, the head of the U.S. Patent Office, Charles H. Duell, asked then-President McKiniey to aboiish the Patent Office (and his job!), because, as he put it: "Everything that can he invented, has been invented" (Cerf and Nava- sky, i984). Nevertheless, Thomas Edison showed up on his doorstep within a few years with a coupie of new items, including the electric iight, the phonograph, and the movie projector. Let us put new batteries in our crystal bail and venture a guess on where operating systems are going in the near future. 12.6.1 Large Address Space Operating Systems As machines move from 32-bit address spaces to 64-bit address spaces, major shifts in operating system design become possible. A 32-bit address space is not yy n earth tes to go around in c r personal

SEC 12 6 TRENDS IN OPERATING SYSTEM DESIGN 895 What could we do wilh an address space of 2 x I0iy byles? For starters., we could eliminate

the file system concept- Instead, all files could be conceptually held in (virtual) memory all the time. After all, there is enough room in there for over I billion full-length movies, each compressed lo 4 GB, Anolher possible use is a persistent object store. Objects could be created in the address space and kept there until all references to them were gone, at which time they would be automatically deleted. Such objects would be persistent in the address, space, even over shutdowns and reboots of ihe computer. With a 64-bit address space, objects could be created at a rate of 100 MB/sec for 5000 years, before we run out of address space. Of course, to actually store this amount of data, a lot ot disk storage would be needed for the paging traffic, but for the first time in history, the limiting factor would be disk storage, not address space. With large numbers of objects in the address space, it becomes interesting to allow multiple processes to run in the s#me address space at ihe same time, to share the objects in a general way Such a design would dearly lead to very dif- different operating systems than we now have. Some thoughts on this concept ;ire contained in (Chase et a!, 1994). Another operating system issue that will have to be rethought with 64-bit addresses is virtual memory. With 2M bytes of virtual address space and 8-KB pages we have 251 pages. Conventional pages tables do not scale well to this size, so something else is needed. Inverted page tables are a possibility, but other ideas have been proposed as well (Tallun et al., 1995). In any event there is plenty of room for new research on 64-bit operating systems. 12.6.2 Networking Current operating systems were designed for standalone computers. Network- Networking was an afterthought and is generally accessed through special programs, such as Web browsers, FTP, or telnet. In the future, networking will probably be the basis for all operating systems. A standalone computer without a network con- connection will be as rare as a telephone without a connection lo the network. And it is likely that multimegabit/sec connections will be the norm Operating systems will have to change to adapt to this paradigm shift. The difference between local data and remote data may blur to the point that virtually no one knows or cares where data are stored. Computers anywhere may be able to treat data anywhere as local data. To a limited extent, this is already true with NFS, but it is likely to become far more pervasive and better integrated. Access to the Web, which now requires special programs (browsers) may also become completely integrated into the operating system in a seamless way. The standard way to store information may become Web pages, and these pages may contain a wide variety of nontext items, including audio, video, programs,

and more, all managed as the operating system's fundamental data.

jj9<> OPERATING SYSTEM DESIGN CHAP. 12 t2,6,3 Parallel and Distributed Systems Another area thai is up and coming is parallel and distributed systems. Current operating systems for multiprocessors and multicomputer are just stan- standard unipr°cessor operating systems with minor adjustments to the scheduler to handle parallelism a bit better, in the future, we may see operating systems where parallelism is much more central than it now is. This effect will be enormously stimulated if desktop machines soon have two, four, or more CPUs in a multipro- multiprocessor configuration. This may lead to many application programs being designed for multiprocessors, with the concurrent demand for better operating system sup- support for them. Multicomputer are likely to dominant large-scale scientific and engineering supercomputers in the coming years, but the operating systems for them are still fairly primitive. Process placement, load balancing, and communication need a lot of work. Current distributed systems are often built as middleware because existing operating systems do not provide the right facilities, for distributed applications. Future ones may be designed with distributed systems in mind, so all the neces- necessary features are already present in the operating system from the start. 12.6.4 Multimedia Multimedia systems are clearly a rising star in the computer world. It would surprise no one if computers, stereos, televisions, and telephones all merged into a single device capable of supporting high-quality still images, audio and video, and connected to high-speed networks so these files could easily be downloaded, exchanged, and accessed remotely. The Operating systems for these devices, or even for standalone audio and video devices, will have to be substantially dif- different from current ones. In particular, real-time guarantees will be needed, and these will drive the system design. Also, consumers will be very intolerant of hourly crashes of their digital television sets, so better software quality and fault tolerance will be required. Also, multimedia files tend to be very long, so file sys- systems will have to change to be able to handle them efficiently. 12.6.5 Battery-powered Computers Powerful desktop PCs, probably with 64-bit address spaces, high-bandwidth networking, multiple processors, and high-quality audio and video wili no doubt be commonplace soon. Their operating systems will have to be appreciably dif- different from current ones to handle all these demands. However, an even faster growing segment of the market is battery-powered computers, including laptops, palmtops, Webpads, and

various telephone hybrids. Some of these will have wireless connections to the outside world; others will run in disconnected mode

SEC. 12 6 TRENDS IN OPERATING SYSTEM DESION 897 when not docked at home. They will need different operating systems that are smaller, faster, more flexible, and more reliable than current ones. Various kinds of microkernel and extensible systems may form the basis here These operating systems will have to handle fully connected (i c, wired) weakly connected (i.e.. wireless), and disconnected operation, including data hoarding before going offline and consistency resolution when going back online better than current systems. They will also have to handle the problems of mobil ity better than current systems (e.g., find a laser printer, log onto it, and send it a file by radio). Power management, including extensive dialogs between [he operating system and applications about how much batiery power is lefi and how it can be best used, will be essential. Dynamic adaptation of applications 10 tan die the limitations of liny screens may become important. Finally, ntw input and output modes, including handwriting and speech, may require new techniques in the operating system to improve the quality. It is unlikely the operating system for a battery-powered, handheld wireless, voice -ope rated computer will h;ve much in common with that of a desktop 64-bii foar-CPU multiprocessor with d gigabit fiber-optic network connection. And of coarse, there will be innumerable hybrid machines with their own requirement*,. 12.6,6 Embedded Systems One final area in which new operating sysiems will proliterate is embedded systems. The operating systems inside washing machines, microwave ovens, dolls, transistor (Internet?) radios, MP3 players, camcorders, elevators, and pacemakers will differ from all of the above and most likely from each other. Each one will probably be carefully tailored for its specific application, since it is unlikely anyone will ever stick a PCI card into a pacemaker io turn it into an elevator controller. Since all embedded sysiems run only a limited number of pro- programs, known at design time, it may be possible to make optimisations not possi- possible in general-purpose systems. A promising idea for embedded systems are the extensible operating systems (e.g., Paramecium and Exokernel). These can be made as lightweight or heavy- heavyweight as the application in question demands, but in a consistent way across applications. Since embedded systems will be produced by the hundreds of mil- millions, this will be a major market for new operating systems. 12.7 SUMMARY Designing an operating system starts with

determining what it should do. The interface should be simple, complete, and efficient, it should have a clear user interface paradigm, execution paradigm, and data paradigm.

898 OPERATING SYSTEM DESIGN CHAP 12 The system should be well structured, using one of several known lechnlques, such as layering or client-server. The internal components should be orthogonal (o one another and clearly separate policy from mechanism. Considerable though! should be given (o issues such as static versus dynamic data structure, naming, binding time, and order ot implementing modules. Performance is important, bu(optimizations should be chosen carefully so as not lo ruin (he system's structure Space-time (rade-offs, caching, hints, exploit- exploiting locality, and optimizing (he common case are often worth doing. Writing a system with a couple of people is different than producing a big sys- system witf> 300 people. In the latter case, team structure and project management play a crucial role in the success or failure of the project. Finally, operating systems will have to change in the coming years to follow new trends and meet new challenges. These may include 64-bil address spaces, massive connectivity, desktop multiprocessors, multimedia, handheld wireless computers, and a huge variety of embedded systems. The coming years will be exciting times for operating system designers. 1. Moore's law describes a phenomenon of exponential growth similar to Ihe population growth of an animal species introduced into a new environment wnh abundant lnod and no natural enemies. In nature, an exponential growth curve is likely eveniuall} to become a sigmoid curve wnh an asymptotic limit when food supplies become limiting or predaiors team to take advantage of new prey. Discuss some factors thai may even- eventually limit Ihe rate of improvement ol computer hardware 2. In Fig. 12-1. two paradigms are shown, algorithmic and event driven. For each ol ihe following kinds, ot programs, which paradigm is likely to be easiest lo use. (a) A compiler. (h) A photo editing program. (c) A payroll program. 3. On some of ihe early Apple Macintoshes, ihe GUI code was in ROM Why1' 4. Hierarchical file names atway.s start at the top of the tree. Consider, for example, the file name /usr/a<it/t)o<>L\/mo&/chap-l2 rather than ckap-l2/moi2/bookb/tisi/Uir In contrast. DNS names start at the bottom of the tree and work up. Is there some funda- fundamental reason tor this difference? 5. Corbattf's dictum is thai ihe system should provide minimal mechanism Here is a list of POSIX calls that were also present in tIMX Version 7 Which ones are

redundant, that is, could be removed with no loss of functionality hccau.se simple combinations of other ones could do the ,ame job with aboui ihe same performance*1 Access, alarm, chdir, chmod, chown, chrool, close, creat, dup. exec, exit, fcntl, fork, (slat, ioctl, kill, link, lseek, mkdir, mknod, open, pause, pipe, read, slat, lime, times, umask, unlink, unme, waif, and write.

CHAP 12 PROBLEMS 899 6. Suppose (ha(layers 3 and 4 in Fig 12 2 were exchanged What implications would ihai have for the design of (he system? 7. In a microkernel-based clien(-server system, (he microkernel]us(docs message pjss- mg and nothing else, Is i(possible for user processes (o nevertheless creaie and ii»e semaphores? If so. how? If not. why not9 8. Careful opuniiiaiion can improve system call performance Consider (fie case in which one sysiem call is made every tO msec The average lime of a call is 2 msec If (he sy.s(em calls can be speeded up by a factor of (wo, how long docs a process that (ook 10 sec to run now cake? 9. Give a short discussion of mechanism versus policy in Che con(ex(of retail stores 10. Operahng systems often do naming a((wo different levels: external and internal Wha(are (he differences be(ween these names with respec(lo (a) Leng(h (b) Uniqueness (c) Hierarchies 11. One way (o handle tables who*e size is not known in advance is (o make them fixed, bui when one fills up, to replace K with a bigger one, copy (he old en(nes over to (he new one, then release the old one What are (he advantages and disadvantages of making the new one 2x ihe size of the original one, as compared to making it only I 5x as big? 12. In Fig. 12-4. a flag, found, it. used (o tell wheiher the PID was located. Would k have been possible to forget about found and just ies(p a(the end of (he loop (o see whether it got to the end or not7 13. In Fig. 12-5, Ihe differences between Ihe Pentium and (he UltraSPARC are hidden by conditional compilation. Could (his same approach be used (o htde (he difference be(ween Pentiums wi(h an IDE disk as ihe only disk and Pentium1, with a SCSI disk as (he only disk? Would it be a good idea1' 14. Indirection is a way of making an algorithm more flexible. Does it have any disadvan- disadvantages, and if so, which ones? 15. Can reentrant procedures have private stalic global variables? Discuss your answer 16. The macro of Fig. l2-6(b) is dearly mm Fig. l2-6(a). One disadvantage, however, other disadvantages? If so, what are they'' 17. Suppose thai we need a way of computing whether ihe number of bits in a 32-bit word is odd or even. Devise an algorithm for performing (his computation as tast as possi- possible. You may use up (o 256 KB of RAM for tables if need be. Write a macro (o carry

out your algori(hm. Extra Credit: Wn!e a procedure lo do ihe computation by looping over (he 32 bits. Measure how many (imes faster your macro is ihan (he procedure 18. In Fig. 12-7, we saw how GIF files use 8-bit values to index into a color palette The same idea can be used with a 16-bii-wide color paletfe. Under wha(circumstances, if any, might a 24-bit color palette be a good idea9

900 OPERATING SYSTEM DESfGN C"HAP l2 19. One disadvantage of GIF is that ihe image must include the color palette, which palette breaks even'1 Now repeat this question for a 16-bii-wide color palette. 20. fn the lexl we showed how caching path names can result in a significant speeifup when looking up path names Another technique that is sometimes used is having a daemon program ihat opens all the fifes m the mot directory and keeps them open per- permanently, m order lo force their i-nodes to be in memory all the time Does pinning iIil* i-nodes like this improve the path lookup even more11 21. t\cn if a remote file has not been removed since a hint was recorded, n may have been changed since the fast nme il v.as reterenccd Wh.« olhcr informalion might il be 22. Consider a system ihai hoards references to remote fifes as hints, for example as Inline, remote-hosi. remote-name). It is possible that a remote file will quietly be removed and then repfaced The hint may ihen retrieve the wrong fife How can lhb probfem be made fess fikcfy to occur9 23. In the text It is staled ihat loeaflly can often he expfoited to improve performance But consider a case where a program reads input from one source and continuousfy out- outputs to two or more fifes, Can an aitempt to lake advaniage of focaljty jn the fife sys- system fead to a decrease in efficiency here'1 fs there a way around this9 24. Fred Brooks cfajms thai a programmer can write fOOO fines of debugged code per veiir, yet ihe first version of MINIX A3.000 lirces of code) was produced by (ine person in under three years How do you exp fain this discrepancy' 25. Using Brooks' figure of 1000 fines of code per programmer per year, make an esti- estimate of the amount of money it took lo produce Windows 3000 Assume that a pro- programmer costs SfOO.OOo per year (incfuding overhead, such as computers, office space, secreianaf support, and management overhead) Do you believe this answer? ff not, what might be wrong with it? 26. As memnry gets cheaper and cheaper, one coufd imagine a computer with a big battery-backed up RAM instead of a hard disk At current prices, how much would a fow-end RAM-dnly PC cost'.' Assume that a f-GB RAM-disk is sufficient for a low- end machine Is this

machine fikefy to be competitive^ 27. Name some features of a conventionaf operating svstem that are not needed in an embedded system used inside an appfiance 28. Write a procedure m C to do a doubfe precision addition (in two given parameters Write the procedure using conditionaf compilation in ^uch a way that it works on f6- bit machines and afso on 32-bit machines 29. Write programs that enter randomfy generated short strings into an array and then can search the array for a given siring using (a) a simple linear search (brute force), and (b) a more sophisticated melhod of your choice Recompile your programs for array sizes raffing from ^niall [o as large as you can handle on your system Evaluate (he performance of both approaches) Where is the breakeven point0 30. Write a program to simulate an m-memory file system.

13 READING LIST AND BIBLIOGRAPHY In the previous 12 chapters we have touched upon a variety oi topics. This chapier is intended as an aid lo readers interested in pursuing (heir study of op- operating systems further Section 13.1 is a list of suggested readings. Section 13.2 is an alphabetical bibliography of all books and articles cited in (his book. In addition to (he references given below, the Proceedings of the n-th ACM Symposium on Operating Svstems Prim ipies (A CM) held every o(her year and the Proceeding* of the n-th International Conjvrence an Distributed Computing Sv*- lems (IEEE) held every year are good places lo look for recen(papers on operat- operating systems. So is the USEN1X Svmpouttm on Operating System* Design u,ui Implementauon Furthermore, ACM Transaction on Computer SsMem and Operating Systems Rei tew are two journals that often have relevant articles 13.1 SUGGESTIONS FOR FURTHER READING In the following sections, we give some suggestions for further reading. Unlike the papers cited m the sections emitted "RESEARCH ON ." in the texl, which were about current research, these references arc mostly introductory or tutorial in nature They can serve lo present material present in this book from a different perspective oi with a different emphasis, however.

902 RF.AD1NG LIST AND BIBLIOGRAPHY CHAP. 13 13.1.1 Introduction and General Works Kavi et al., ••Computer Systems Research: The Pressure is On" Where is systems research going? Whai is importanl now'.' How about developing robust, predictable systems? How about billion-transistor chips and hillion-user worldwide systems? These are some of the questions and answers presented in this paper. Milojicic, "Operating Systems: Now and in

the Future," Suppose you were to ask six of the world's leading experts in operaiing sys- systems a series of questions about the field and where it was going. Would you get the same answers? Hint'. No. Find oul what they said here. Silberschatz et al., Applied Operating System Concepts A general lextbook on operaiing sysiems. H covers processes, storage management, distributed systems, and protection. Three case studies are given: UNIX, Linux, and Windows NT. The cover is full of dinosaurs. Whai, if any- anything, this has to do with operaiing systems anno 2000 is unclear. Stalling;., Operating Systems, 4th Ed. Still another textbook on operaiing systems. It covers all (he traditional topics, and also includes a small amount of maieria] on distributed systems. Stevens, Advanced Programming in the UNIX Environment This book lells how to write C programs that use the UNIX system call inter- interface and the standard C library. Examples are based on the System V Release 4 and the 4.4BSD versions of UNIX. The relationship of these implementations to POSIX is described in detail 13.1.2 Processes and Threads Andrews and Schneider, "Concepts and Notations for Concurrent Programming" A tutorial and survey of processes and interprocess communication, including busy waiting, semaphores, monitors, message passing, and other techniques. The article also shows how these concepts are embedded in various programming languages Ben-Ari, Principles of Concurrent Programming This little book is entirely devoted to the problems of interprocess communi- communication. There are chapters on mutual exclusion, semaphores, monitors, and the dining philosophers problem, among others.

SEC- 13.1 SUGGESTIONS FOR FARTHER READING 903 Silberschatz et al., Applied Operating System Concepts Chapters 4 through 6 cover processes and interprocess communication, including scheduling, critical sections, semaphores, monitors, and classical inter- interprocess communication problems. 13.1.3 Deadlocks Coffman et al,, "System Deadlocks" A short introduction to deadlocks, what causes them, and how they can be prevented or detected. Holt, "Some Deadlock Properties of Computer Systems" A discussion of deadlocks. Holt introduces a directed graph model that can be used to analyze some deadlock situations- Isloor and Marsland, "The Deadlock Problem: An Overview" A tutorial on deadlocks, with special emphasis on database systems. A variety of models and algorithms are covered. 13.1.4 Memory Management Denning, "Virtual Memory" A classic paper on many

aspects of virtual memory. Denning was one of the pioneers in this field, and was the inventor of the working set concept. Denning, "Working Sets Past and Present" A good overview of numerous memory management and paging algorithms. A comprehensive bibliography is included. Knuth, The Art of Computer Programming Vol 1 First fit, best fit, and other memory management algorithms are djscussed and compared in this book. Silberschatz et al, Applied Operating System Concepts Chapters 9 and 10 deal with memory management, including swapping, pag- paging, and segmentation. A variety of paging algorithms are meniioned 13.1.5 Input/Output Chen el al., "RAID; High Performance Reliable Secondary Storage" The use of multiple disk drives in parallel for fasi I/O is a trend in high-end systems. The authors discuss this idea and examine different organizaiions in terms of performance, tost, and reliabilny

904 READING LIST AND BIBLIOGRAPHY CHAP 13 Computer, March 1994 This issue of Computer coniams eighi articles on advanced I/O, and covers simulaiion, high performance storage, caching. I/O for parallel compuiers, and multimedia. Geis] and Daniel. "AConiinuumoiDisk Scheduling Algorithms" A generalized disk arm scheduling algonihm is presented. Extensive Simula- lion and ex pen menial results are given. Gibson and Van Meter, -'Neiwork Aitached Storage" The Internet is driving the need for vasi amounts ot storage for Web, data- database, and other servers This situation has led 10 various designs in which free- freestanding siorage can be attached to a network In this paper, several architectures to accomplish lhis goal are discussed Ng, "Advances mDi.sk Technology: Performance Issues'" The various faciors in disk performance, such as linear bil density, rpm, number of heads, and number of seciors/track can be traded Dffin various ways io affeci system performance. These issue.1- and lheir inipaci on performance are covered in this easy-to-read paper on disk technology Ruemmler and Wilkes, "An Imroduction to Disk Drive Modeling" The firy part of this article K about modern disk drive* and how they work inside, including topics such as seeking, zoning, track skewing, sparing, caching, read ahead, and much more The second part is about modeling disk drives. Walker and Cragon, -Interrupt Processing in Concurrent Processors" Implementing precise interrupts on superscalar computers is a challenging activity. The trick is io serialize the state and do it quickly. A number of the design issues and trade-offs are discussed here 13.1.6 File Systems Harbron, File Systems A book on file

system design, applications, .ind performance. Both structuie and algorithms are covered. McKusick et al,, "A Fast File System for UNIX" The UNIX file system was completely redone for 4 2 BSD This paper describes the design of the new file system, with emphasis on its performance. Silberschalz et al.. Applied Operating System Com eptt Chapter 11 is aboui tile systems. It coven, file operations, access methods, directories, and implementation, among other topics

SEC 13.1 SUGGESTIONS FOR FURTHER READING 905 Sellings, Operating Systems, 2nd Ed. Chapter 14 contains a fair amount of material about the security environment especially about hackers, viruses, and other threats. 13.1.7 Multimedia Operating Systems ACM Computing Suneys, Dec. 1995 This issue of ACM Computing Surveys contains 21 short papers on vanoun aspects of multimedia, ranging from low-level technical issues to high-level appli- applications issues. Computer, May 1995 The theme of this issue of Computer is multimedia and it contains six articles on the subject. After a brief introduction, the papers cover interactive television, multimedia servers, research management, and applications in medicine and edu- education. Lee, "Parallel Video Servers: A Tutorial" Many organizations want to offer video on demand, which creates a need for scalable, fault-tolerant parallel video servers. The major issues of how to build them are covered here, including server architecture, striping, placement policies, load balancing, redundancy, protocols, and synchronization- Leslie et a]., "The Design and Implementation of an Operating System to Support Distributed Multimedia Applications," Many attempts at implementing multimedia have been based on adding features to an existing operating system. An alternative approach is to start all over again, as described here, and build a new operating system for multimedia from scratch, with no need to be backward compatible for anything- The result is a fairly different design than conventional svstems. Reddy, "I/O Issues in a Multimedia System" When people talk about computer performance, they usually mean CPU per- performance. However, for multimedia, I/O performance is at least as important and that is the subject of this paper. Disk scheduling, buffer space trade-offs, and admission control are among the topics covered here. Sitaram and Dan, "Multimedia Servers- Multimedia servers have many differences with regular file servers. The authors discuss the differences in detail, especially in the areas of scheduling, the storage subsystem, and caching.

906 READING LIST AND BIBLIOGRAPHY CH4P. 13 13.1.8 Multiple Processor Systems Ahmad, -'Gigantic Ousters- Where Are They and What Are They Doing?"' To get an idea of the state-of-the-art in large minicomputers, this is a good place to look, h describes the idea and gives an overview of some of the larger system;, currently in operation Given the working of Moore's law, it is a reason- reasonable bet thai the sues mentioned here will double abom ever} 2 years or so Bhoedjang et al.. "User-Level Network Interface Protocols- More and more multicomputers are putting the network interface board in user space to improve performance- Doing so raises many design issues, eleven of which are discussed here A number of actual systems are also compared. Computer, Dec 1996 This issue of Computer contains eight papers on multiprocessors One of them is a tutorial on the semantics of shared memory, hut the other seven deal with multiprocessor applications and performance Dubois et al., "Synchronization, Coherence, and Event Ordering in Multiproccs- A tutonal on synchronization in shared-memory multiprocessor systems However, some of ihe ideas are equally applicable to single processor and distri- distributed memory systems as, well. Kwok and Ahmad "Static Scheduling Algorithms for Allocating Directed Task Graphs lo Multiprocessors" Optimal job scheduling of a multicomputer or multiprocessor is possihle when the characteristics of all the jobs are known in advance The problem is that optimal scheduling lakes far too long to compule. In this paper, the authors dis- discuss and compare 27 known algorithms for attacking this problem in different ways. Langendoen et al., "Models for Asynchronous Message Handling" High performance in multicompulers depends criiically on the performance of the message-passing system, especially how incoming messages are handled. The main choices are active messages, upcalls. and pop-up threads. The authors describe all three of them in this paper and then present experimental results com- comparing them on the same hardware plalform Protic et al.. Distributed Shared Memory Concepts and Systems For an introduction to distributed shared memory, this collection ot 28 previ- previously published papers is a good start, It collects many of the classic papers on models, algorithms, and implementations inlo a single convenient book.

SEC 13 I SUGGESTIONS FOR FURTHER READING 907 Stenstrom et al., "Trends in Shared Memory Multiprocessing," Which way are multiprocessors going? The authors helieve the future lies more in small-scale multiprocessors

than large-scale ones. They also discuss models, architectures, and parallel software. Waldo, "Alive and Well Jini Technology Today" bar a quick introduction to Jim, its components, and how they fit together, this paper is a good starling place Perhaps indicative of how information will be disseminated in the future, it has no bibliography of published papers, but instead provides the URLs of many Weh resources on Jini. 13.1.9 Security Computer, Feb 2000 The theme of this issue of Computer is btometncs, with six papers on the sub- subject. They range from an introduction to the subject, through various specific- technologies, to a paper dealing with the legal and privacy issues. Denning, "The United States vs. Craig Neidorf" When a young hacker discovered and published information about how the telephone system works, ho was indicted for computer fraud. This article desenhes the case, which involved many fundamental issues, including freedom of speech. The article is followed hy some dissenting views and a rebuttal by Denning. Denning, Information Warfare and Secuni\ Information has become a weapon of war. both military jnd corporate. The participant!; not only try to attack the other side's information systems, but safe- safeguard their own as well. In this fascinating book, the author covers every con- conceivable topic relating to offensive and defensive strategy, from data diddling to packet sniffers. A must read for anyone seriously interested in computer security. Hafner and Markoff, Cyberpunk Three compelling tales of young hackers breaking into computers around the world are told here by the New York Time-, computer reporter who broke the Inter- Internet worm story (Markoff) Computer, Feb 2000 Johnson and Jajodia, "Exploring Steganograpliy- Seeing the Unseen"' Steganography has a long history, going back to the days when the writer would shave the head of a messenger, write a message on the shaved head, and send him off after the hair grew back. Although current techniques are often hairy, they are also digital. For a thorough introduction to the subject as currently practiced, this paper is the place to start.

 Ludwig, The Giant Black Book of Computer Viruses. 2nd ed If you want to write antivirus software and need to understand how viruses work down to the bit level, this is the book for you Every kind of virus is dis- discussed at length and aciual code for most of them is supplied on a floppy disk. A thorough knowledge of programming the Pentium in assembly language is a mu.st. however. Milojicic, "Security and Privacy," Security has many facets including operating systems,

networks, implications for privacy, and more In this article, six security experts are interviewed on their thoughts on the subject. Nachenberg, "Computer Virus-Antmrus Coevolution" As soon as the antivirus developers find a way to detect and neutralize some class of computer virus, the virus writers go them one better and improve the virus. The cat-and-mouse game played by the viru.s and antivirus sides is dis- discussed here. The author is not optimistic about the antivirus writers winning the war, which is bad news for computer users. Pfleeger, Security in Computing, 2nd ed. Although a number of books on computer security have been published, most of them cover only network security. This book docs that, but also has three chapters on operating systems security, which make it a good reference for more background on that subject. 13.1.10 UNIX and Linux Bovet and Cesati, Understanding the Linux Kernel This book is probably the best overall discussion of the Linus kernel. It cov- covers processes, memory management, file systems, signals, and much more IEEE, Information Technology—Portable Operating System Interface (POSIX), Part I: System Application Program Interfm e (API) fC Language} This is the standard, Some parts are actually quite readable, especially Annex B, "Rationale and Notes," which often sheds light on why things are done as they are, One advantage of referring to the standards document is that, by definition, there are no errors. If a typographical error in a macro name makes it through the editing process it is no longer an error, it is official Lewine, POSIX Programmers Guide This book describes the POSIX standard in more detail than the standards document, and includes discussions on how to convert older programs to POSIX and how to develop new programs for the POSIX environment, with examples.

ShC 13 I SUGGESTIONS FOR FURTHER READING 909 Maxwell, Linux Core Kernel Commentary The first 400 pages of this' book contain a subset of the Linux kernei code The last 150 pages consists of comments on the code, very much in the style ot John Lions' classic book A996). If you want to understand the Linux kernel in all its gory detail, this is the place to begin, but be warned: reading 40.000 lines of C is not for everyone. McKusick et al., The Design and Implementation of the 4.4 BSD Operating Sys- The title says it all But actually, the book is more general than that since all UNIX systems are fairly similar internally. This is an excellent reference for any- anyone wanting to learn about the internal workings of UNIX in general. 13.1.11 Windows 2000 Cusumano and Selby, "How Microsoft Builds Software" Have you ever

wondered how anyone could write a 29-million-lme program (like Windows 2000) and have it work at all? To find out how Microsoft's build and test cycle is used to manage very large software projects, take a look at this Norton et al.. Complete Guide to Windows 2000 Professional If you are looking for a book that discusses how to set up and use Windows 2000 but also discusses many of the advanced features in some detail, such a.s the registry, FAT and nTFS file systems, ActiveX, DCOM, and networking this is a good choice. It lies in between the mass of books that tell wbere to click to get whicb effect and Solomon and Russinovich's book Rector and Newcomer, Win32 Programming If you are looking for one of those 1500-page books giving a summary of how to write Windows programs, this is not a bad start. Ii covers windows, devices, graphical output, keyboard and mouse input, printing, memory management, libraries, and synchronization, among many other topics'. It requires knowledge of C or C++. Solomon and Russinovich. Inside Windows 2000, 3rd ed. If you want to learn how to use Windows 2000, there are hundreds of books out there. If you want to know how Windows 2000 works inside, this is your best bet. It covers numerous internal algorithms and data structures and in consider- considerable technical detail. No other book comes close.

9J0 READING LIST AND BIBLIOGRAPHY CHAP 13 -13.1.12 Design Principles Brooks, -The Mythical Man Month Essays on Software Engineering" Fred Brooks was one of the designers of IBM's OS/360 He learned ihe hard way what works and what does not work. The advice given in this witty, amus- amusing, and informative book is as valid now as it was a quarter of a century ago when he first wrote it down. Cooke et al., "UNIX and Beyond: An Interview with Ken Thompson" Designing an operating system is much more ol an art than a science. Conse- Consequently, listening to experts m the field is a good way to learn about Ihc subject. They do noi come much more expert lluu Ken Thompson, «i-designer of UNIX, Infema, and Plan 9. In this wide-ranging imerview, Thompson gives his lhoughis on where we came from and where we are going in ihe field. Corbaio. "On Building Sy.sicms Thai Will hul" In his Turing Award leciure, ihc faiher of timesharing addresses many of ihc same concerns lhai Brooks does in the Mythical Man-Month. His conclusion is thai all complex sysiem<> will uhimaiely tail, and lhiit hi have any chance tor suc- and elegance in design Crowley. Operating System.-,: A Deiign-Onented Approa*. h Most lextboojts on operating sysiems lusi describe

the basic concepis (processes, virtual memory, utc.) and a few examples, but say nothing about how to design an Operating system This one is unique in devoting tour Chapters to the subject. Lampson. "Hints for Computer System Design"' Butler Lampsdn, one of the world's lending designers ot innovative operating systems,, has collected many hints, suggestions, and guidelines frcim his years of experience and put them together In this entertaining and informative article. Like Brooks' book, this is required reading for every aspiring operating system designer Wirth, "'A Plea for Lean Sciflware'" Niklaus Wirth, a famous and experienced system designer, makes the ease here for lean and mean software based on a few simple ccineepls, instead of the bloated mess that much commercial software is. He makes his paint by discuss- discussing his Oberon system, a network-oriented, GUI-based operating system that fits in 200 KB. including the Oberon compiler and text editor.

SEC 112 ALPHABETICAL BIBLIOGRAPHY 1 13.2 ALPHABETICAL BIBLIOGRAPHY ABDF.I.-HAMID, T., and MADNICK, S.: Software Project Dynamics. An Integrated Approach, Upper Saddle River, NJ: Prentice Hall, 1991 ABRAM-PROFETA, E.I-., and SHIN. K.G.: "Providing Unrestricted VCR Functions in Multicast VJdeo-<>n-Demand Servers," Proc. lit7! Canf. on Multimedia Comp. Syst, IEEE, pp. 66-75, 1998 ABLTALEB, A., and LI, V.O.K.: "Paging Strategy Optimization in Personal Commilnjca- lion Systems Wireless Networks,'" Wireless Networks, vol 3, pp 195-204, Aug. 1997 ACCETTA, M., BARON, R., GOLUB, D., RASH1D, R., TEVANIAN, A., and YOUNG, M.: "Mach A New Kernel Foundation for UNIX Development," Proc Summer 1986 USEN1X Conf, USt.NIX. pp 93-112. 19R6 ADAMS. GB. Ill, AGRAWAL, D.P., and S1EGEL. HJ.: 'A Survey and Comparison of Fault-Tolerant Multistage Interconnection Networks," Computer, vol 20, pp. 14-27. June 1987. ADAMS, A., and SASSE. MA.: "Taming the Wnli in Sheep's Clothing," Proc. Seventh Infl Conf. cm Multimedia, ACM, pp. 101-107, 1999. AHMAD, 1.: "Gigantic Clusters' Where Are They and What Are ITiey Doing'"" IEEE Concurrency, vo| 8. pp 83-85, April-June 2000 ALEXANDROV, AJX, 1BEL, M., SCHAUSER. K.E., and SCHE1MAN, CJ-: 'UFO a Per- Personal Global File System Based im User-Level Extensions tn (he Operating System," Trans on Computer Systems, vol. 16, pp 207-233. Aug 1998 ALFIEKI, R.A.: "An Efficient Kernel-Based Implementation t>i POS1X breads," Proc Summer 1994 USENIX Tech Conf., USENIX, pp. 59-72, June

1994 ALVAREZ, G.A., BURKHARD. W.A.. and CRISTIAN. Vj Tolerating Multiple failures in RAID Architectures with Optimal Storage And Uniform Dee 1 ustering," Proc 24th inflSymp. on Computer Architecture, ACM, pp. 62-72, 1997. ANDERSON, T.F,.; "The Performance oTSpin Lock Alternatives for Shared-Memory Mul- Multiprocessors." IEEE Trans, on Parallel and Distr System?, vol 1, pp. 6-16. Jan. 1990 ANDERSON, T.E., BERSHAD. B.N., LAZOWSKA, E.D., and LEVY, H.M.; 'Scheduler Activations: Effective Kernel Support for the User-level Management of Parallelism." ACMTrans. on Computer Systems, vol. 10. pp 53-79. Feb 1992 ANDREWS, G.R.: Concurrent Programming—Principles and Practice. Redwood City. CA: Benjamin/Cummings, 1991 ANDREWS, G.R, and SCHNEIDER, F.B,: "Concepts and Notations for Concurrent Pro- gratnmmg." Computing Surveys, vol 15, pp. 3-43, March 1983 ARON, M., and DRUSCHEL, P,: "Soft Timers: Efficient Microsecond Software Timer Support for Network Processing,-' prnc. 17th Symp on Operating Systems Principles, ACM, pp. 223-246, 1999

912 READING LIST AND BIBLIOGRAPHY CHAP. 13 ARORA, A,S., BLUMOPE, R.D., and PLAXTON, CO.: "Thread Scheduling for Mulllpro-grammed Multiprocessors/' Proc. Tenth Svmp on Paia/lel Algorithms and Architec- Architectures, ACM, pp. 119-129,1998 BAKER, F.T.: "Chief Programmer Team Management of Production Programming," IBM Seisms Journal, vol II,pp. 1.1972 BAKER-HARVEY, M,: "ET1 Resource Distnbutor. Guaranteed Resource Allocation and Scheduling m Multimedia Systems," Proc. Third Symp on Operating Systems Design and Implementation, USEN1X, pp. 131-144. 1999 laiion Lookaside Buffers," Proc, First Symp. on Operating System Design and Imple- Implementation, USENIX, pp. 243-254, 1994 BALLINTUN, G,, VAN STEEN, M,, and TANENBAUM, A.S.: "'Scalable Naming in Global Middleware," Proc. I3lh Infl Conf. on Parallel and Distributed Systems, 1SCA, pp. 624-631,2000. BAYS, C: "A Comparison nf Next-Fit, First-Fit, and Best-Fit." Commun. of the ACM, vol 20,pp. 191-192, March 1977 BECK. M^ BOHME, H., DZ1ADZKA, M., KUN1TZ, U., MAGNUS, K., VERWORNRR. D,: Linux Kernel Internals, 2nd ed, Reading, MA. Add,son- Wesley 1998 BELADY, LA., NELSON, R.A,, and SHEDLER, G.S.: " Characteristics of Certain Programs Running m a Pagi ACM, vol 12, pp. 349-353, June 1969. BELL, D., and LA PADULA, L.; "Secure Computer Syste and Model," Technical Report MTR 2547 v2. Mure Co BEN-ARl, M.: Principles of Concurrent Programming, Up Hall International, 1982.

BERNHARDT, C, and B1ERSACK, E.W.: -The Server Array: A Scalable Video Server Architecture." in High-Speed Networking for Multimedia Applications AmsteidaiTi' Kluwer Publishers, 1996. BERSHAD, B.N., SAVAGE, S., PARDYAK, P., BECKER, D., F1UCZYNSKI, M, and SIRER, E.G.: "Protection is a Software Issue,- Proc Hot Topics ,n Operating Sys-Systems V. IEEE, pp 62-65, 1995a. BERSHAD, B.N., SAVAGE, S., PARDYAK, P., SIRER, E.G.., FILCZYNSK1, M_ BECKER, D,, CHAMBERS, C., and EGGERS, S.: "Extensibility. Safety, and Performance in the SPIN Operating Sysiem," Proc ISth Symp. on Operating Systems Principles. ACM. pp. 267-284, 1995b. BHOEDJANG, R.A.F.: Communication Architectures for parallet-Piogramming Systems, Ph D Thesis, Vrije Universiteii, Amsterdam. The Netherlands. 2000 BHOEDJANG, RA.F., RUHL, T., and BAL, H.E.: 'User-Level Network Interface Proto cols," Computer, vol. 31, pp. 53-60, Nov 1998. Anoir tfachin Mathe Nov 1 Saddle laly Cot 973 Riv ■er Space Found -Til NJ. Prcnt the „ ice

SEC 13.2 ALPHABETICAL BIBLIOGRAPHY 913 BHUYAN, L.N., YANG, Q., and AGRAWAL, D.P.; -performance of Multiprocessor Inter-Interconnection Neiworks," Computer, vol. 22, pp. 25-37, Feb. 1989. B1BA, K..' -Integrity Considerations for Secure Compufer Systems," Technical Report 76-371, US. Air Force Electronic Systems Division, 1977. B1RREIX, A,D,, and NELSON, BJ.: "implementing Remoie Procedure Calls," ACM Tram on Computer Systems, vol 2, pp 39-59, Feb 1984. BISDIKIAN, C.C., and PATEL, B.V.: "issues on Movie Allocation in Distributed Video- on-Demand Systems," Proc. Infl Conf. on Commun., IEEE, pp, 250-255,1995 Bl.AUM, M., BRADY, J., BRUCK, J., and MENON, J.: "EVENODD: An Optimal Scheme for Tolerating Double Disk Failures in RAID Architectures," Proc. 21st Infl Symp. on Computer Architecture. ACM, pp 245-254, 1994. BIXMOFE, R.D., and LEISERSON, C.E.: "Scheduling Multithreaded Computations by Work Stealing," ProC. 35th Annual Symp on Foundations of Computer Science, IEEE. pp. 356-368, Nuv. 1994 BOEHM, B.W.: Software Engineering Economics, Upper Saddle River, NJ Prentice Hall. 1981. BOLOSKY, W.J., FITZGERALD, R.P., and DOUCEUR, J.R.: "Distributed Schedule Management in the Tiger Video Fileserver." Proc. 16th Symp on Operating Systems Principles, ACM, pp 212-223,1997. BORN, G,,: Inside the Microsoft Windows 98 Registry, Redmond WA: Microsoft Press, 1998 BOVET, D.P., and CESATI, M.: Understanding the Linux Kernel Sebastopol, CA. O'Reilly &

Associates, 2000 BRANDWEIN, R,, KATSEFF, H., MARKOW1TZ, R., MORTENSON, R., and ROBINSON. B.: "Nemesis; Multimedia information Delivery " Prot. Second ACM Infl Conf. on Multimedia, ACM, pp 473-481, 1994. BRICKER, A., GIEN, M., GUILLEMONT, M., LIPKIS, J., ORR, D,, and ROZIfcR, M.: "A New Look at Microkernel-Based UNIX Operating Systems: Lessons in Performance and Compatibility,'- Proc. EurOpen Spring '91 Conf. EurOpen, pp. 13-32, 1991. BRINCH HANSEN.P.: 'The Programming Language Concurrent Pascal," IEEE Trans on Software Engineering, \o\.SE-\, pp. 199-207, June 1975. BROOKS, F. P., Jr.; Tke Mythical Man-Month' Essays on Software Engineering, Reading MA: Addison-Wesley, 1975 BROOKS, F.P., Jr, "No Silver Bullet—Essence and AccLdeni Ln Software Engiheenng," Computer, vol, 20, pp 10-19. April 1987 BROOKS, F, P., Jr.: The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary edition, Reading, MA. Add ison-Wesley, 1995.

 BUCHANAN, M-, and CHIfcN, A.: "Coordinated Thread Scheduling for Workstation Clus- Clusters Under Windows NT." The USENIX Windows NT Workshop, USEN1X, pp 47-"?. 1997 BUGNION, E., DEVINE, S., GOV1L, K., and ROSENBLUM, M.; "'Disco: Running Commo- djiy Operating Systems on Scalable Multiprocessors," Trans, on Computer Systems, vol. 15, pp 412-447, Nov 1997. BUZZARD, G., JACOBSON, D., MACKEY, M., MAROVICH, S., and W1LKES, J.: "An Implementation of the Hamlyn Sender-Managed Inlerface Architeciure." Proc. Second Symp. on Operating System Design and Implementation, USENIX, pp 245- 259, Oci. 1996. CANT, C: Writing Windows WDM Device Drh-erS, Lawrence, KS: R&D Books, 1999. CAO, P., LIN, S.B., VENKATARAMAN, S., and WlLKES, J.: The TickerTA|P Parallel RAID Architecture," Trans, on Computer Systems, vol. 12, pp, 236-269, Aug. 1994 CAO, P_ FELTEN, E.W., KARLIN, A.R., and LI, K.; "A Si»dy of Iniegrated Prefetching and Caching Strategies," Proc. SIGMETRICS Joint Infl Conf. on Measurement and Modeling of Computer Systems, ACM, pp 188-197, 1995. CARLEY, L.R., GANGER, G.R., and NAGI.E, D.F.: "MEMS-Based Iniegraied CircuH Mass Storage Systems," Commun. of the ACM. vol 43. pp. 73-80, Nov. 2000. CARR, R.W., and HENNESSY, JX.: "WSClock—A Simple and Effective Algonlhm fi,r Virtual Memory Management" Proc Eighth Symp. on Operating Systems Principle!, ACM. pp. 87-95, 1981 CARRIERO, N, and GEl.ERNTER, D.: "The S/Ne!\ Linda Kernel," ACM Trans on Computer

Systems, vol. 4, pp. 1 10-129, May 1986. CARRIERO, N., and GELERNTER, D.: "Linda in Context," Commun of the ACM, vol. 32, pp. 444-458, April 1989. CARTER, J.B., BENNETT, J.K., and 7.WAENEPOEL, W.: 'Techniques for Reducing Consistency-Related Communtcalion in Dtstnbuied Shared-Memory Systems," Trans on Computer Systems, vol. 13, pp 205-243, Aug 1995 CERF, C,and NAVASKY, V.: The Experts Speak, New York' Random House, 1984 CHANDRA, A., ADLER, M., GOYAL, P., and SHENOY, P.^IO "Surplus Fair Scheduling A Proportional-Share CPU Scheduling Algonlhm for Symmetric Multtpiocessor<.." Proc Fourth Symp on Operating S\stems Design and [mpkmenmlion, USENIX, pp 45-*>R. 2000 CHASE, J.S., LEVY, H.M., FEELEY, MJ., and LAZOWSKA, E.D.: "Sharing and Protec- Protection in a Single-Address-Spaoe Operaiing System " Trans, on Computer Systems vol 12, pp 271-307,Nov. 1994 CHEN, PM., LEE, E.K., GIBSON, G.A., KATZ, R.H., and PATTERSON, D.A.; -RAID High Performance Reliable Siorage," Comp. Surv., vol, 26, pp. 145-185. June 1994.

SEC. 13-2 ALPHABETICAL BIBUOGRAPHY 915 CHEN, P.M., NG, W.T., CHANDRA^., AVCOCK, C,, RAJAMANI, G., and LOWELL, D.: "The Rio File Cache: Surviving Operating System Crashes," Proc Seventh Int'i Conf. on Architectural Support for Programming Languages and Operating Systems, ACM, pp 74-83,1996. CHEN, S, and THAPAR, M.: "A Novel Video Layout Strategy for Near-Video-on - Demand Servers," Proc tori Conf. on Multimedia Computing and S^iems. IEEE, pp. 37-45,1997. CHEN, S,, and TOWSLEV, D.: "A Performance Evaluation of RAID Architectures," IEEE Trans, on Computers, vol. 45, pp III6-II30, Oct. 1996. CHERITON, D.R.: "An Experiment Using Registers for Fast Message-Based Interprocess Communication," Operating Systems Review, vol. 18, pp, 12-20, OcL 1984 CHERITON, D.; "The V Distributed System," Commun. of the ACM. vol. 31, pp. 314-333, March 1988. CHERVENAK, A., VELLANKI, V., and KURMAS, 7,.; "Protecting File Systems A Survey of Backup Techniques," Proc. I5th IEEE Symp on Mass Storage Systems, IEEE, 1998. CHOW, T.C.K., and ABRAHAM, JA.; "Load Balancing in Distributed Systems." IEEE Trans, on Software Engineering, vOl.SE-8,pp 401-412, July 1982. CLARk, P.C., and HOFFMAN, LJ,: "BITS: A Smartcard Protected Operating System," Commitn.oftheACM.vtX 37, pp. 66-70, Nov 1994. COFFMAN, F.G., ELPHICK, MJ, and SHOSHANI, A.; "System Deadlocks," Compunng Surveys, vol 3, pp. 67-78, June 1971. COMER, D.: Operating System Design, '[he Xinu

Approach, Upper Saddle River NJ Prentice Hall, 1984 COOKE, D., URBAN, J,, and HAMILTON, S.: "Unix and Beyond An Interview with Ken Thompson," Computer, vol. 32, pp 58-64. May 1999 CORBA1AN, J., MARTORELL, X., and LABARTA, J.: "Performance-Driven Processor Allocation," Pro, Fourth S\mp on Operating Sutemt DeiiRn and Implementation, USKNIX, pp. 59-71,2000. CORBATO, FJ.: "On Building Sysiems Thai Wi!! Pail," Convnun of the ACM, v«! 34, pp 72-81,June 1991. CORBATO, FJ., MERW1N-DAGGETT, M., and DALEY. R.C.; "An Experimental Time- Timesharing System," Proc. AFIPS Fall Joint Computer Conf., AFIPS. pp 335-344, 1962 CORBATO, FJ, SALTZER, J.II, and CIJNGEN, C.T.: "Ml.'LTICS—-The Firbt Seven Years," Proc. AFIPS Spring Joint Computer Conf. <\FIPS, pp. 57I-5S3 1972. CORBATO, FJ, and VVSSOTSKV, VA,: - Introduction and Overview of ihe MULTlCS System," Proc, AFIPS Fall Joint Computer Conf, AFIPS, pp. 185-196, 1965.

916 READING LIST AND BIBLIOGRAPHY CHAP. 13 COURTO1S, P.J., HEVMANS, F., and PARNAS. D.L.: "Concurrent Control with Readers and Writers," Commun ofthe ACM, vol. 10, pp 667-668, Oct 197) CRANOR, CD., and PARULKAR, G,M.: "The UVM Virtual Memory System," Proc IJSENIX Annual Tech Conf. USENIX, pp H7-130. 1999 CROWLEV, C:. Operating Systems: A Design-Onented Approach, Chicago; Irwin, I99"\ CUSUMANO, MA., and SELBY, B.W.: "How Microsoft Builds Software." Comntun of the ACM, vol. 40. pp. 53-61, June 1997 DALEV, R.C., and DENNIS, JB.; "Virtual Memory, Process, and Sharing in MULTICS." Commun. of the ACM, vol. 11, pp. 306-312. May 1968. DAN, A,, SIT ARAM, D., and SHAHABUDDIN, P: "Scheduling Policies for an On-Demand Video Server with Batching," Proc Second Int'i Conf on Multimedia, ACM, pp 15- 23, 1994 DANDAMUDI, S.P.; -Reducing Run Queue Contention in Shared Memory Multiproces-Multiprocessors," Computer, vol 30, pp. 82-89, March 1997 DE JONGE, W-, KAASHOEK, M.F,, And HSIEH. W.C: "The Logical Disk: A New Approach to Improving File Systems." P,oc 14th Svmp on Operating Systems Prin-Principles. ACM, pp. 15-28, 1993 DENNING, D.; "A Lattice Mode) of Secure Information now." Commun. of the ACM, vol 19, pp. 236-243, 1976 DENNING, D-: "The United Slates vs. Craig Neidorf," Commun of the ACM. vol. 34. pp. 22-43.March 1991. DENNING, D,: Information Warfare and Security, Reading, MA Addison-Wesley. 1999. DENNING, P.J.: "The Working Set Model for Program Behavior," Commun o/ the ACM, vol 11. pp 323-333,

1968a DENNING, PJ-: "Thrashing Its Causes and Prevention," Proc AFIPS National Compuur Con}., AFIPS, pp. 915-922, 1968b. DENNING, P.J.: "Virtual Memory," Computing Surveys, vol. 2. pp 153-189, Sept. 1970. DENNING, P.J.: "Working Seij, Past and Present.- IEEE Trans on Software Engineering vol. SE-6. pp. 64-84, Jan. 1980 DENNIS, J.B., and VAN HORN, E.C.: "Programming Semantics for Multiprogrammed Computations," Commun of the ACM, vol. 9, pp 143-155, Marth 1966 DEVARAKONDA, M., KISH, B., and MOHlNDRA, A.: "Recovery in the Ca|ypso File Sys- lenV' Trans, on Computer S:\ttems, vol. I4.pp 287-310,'Vug 1996 DIFF1E, W., and HELLMAN M.E.: "New Directions in Cryptography " IEEE Trans, on Information Theory, vol IT-22, pp. 644-654, Nov. 1976.

ALPHABETICAL BIBLIOGRAPHY DUKSTRA, E.W.: "Co-operating Sequential Processes..' in I ingiamming ijjngua^e\. Genuys. F. (Ed.), London-Academic Press, 1965 DUKSTRA, E.W.: -The Structure of THE Multiprogramming Swem." Cominun. of the ACM.vul 11, pp 341-346, May 1968 DUBOIS, M., SCHEURICH. L\, and BRICCS. F.A.: 'Synchronization. Coherence, lLnd Event Ordenng m Multiprocessors." Computei, vol 21. pp 9-21.Keb 1988. DOUCEUR, J.R., and BOLOSKV, wj.: "A Large-Scale Study ol File-System Contenis," Pn>c Int'l Conj on Meawtement and Modeling oj Computer S\tlem\. ACM pp 59- 70. 1999 DRUSCHEL, P,, PAI, V.S.. and ZWAENEPOEL, W.: Extensible Systems are Leading OS Research Astray," Proc Hot Topics in Operating Systems VI. IEEE. pp. 38-42, 1997 DUDA, KJ., and CHERITON, D.R.: "Borrowed-Virtual-Time (BVT) Scheduling Support-Supporting Latency-Sensitive Threads m a General-Purpose Scheduler," Pruc 17th S\mi> oil Operating Systems Principles, ACM. pp. 261 -276, 1999. EAGER, D.L., LAZOWSKA, E.D., and ZAHORJAN, J.: "Adaptive Load Sharing in Homo- Homogeneous Distributed Syslems,'" IEEE Tram, oil Software Engineering, vol. SF-12, pp 662-675, May 1986 EAGER, DA., VERNON, M, and ZAHORJAN, J.: ■Optimal and Efficient Merging Schedules fur Video-an-Demand Server1.." Proc Seventh Int'l Conf on Multimedia, ACM, pp. 199-202, 1099. EDLEK, J., LIPKIS, J,, and SCHONBERG, E.: Process Management lor Highly Parallel UNIX Systems,- Proc UStNlX Workshop on UNIX and Super*om(Wers, USENIX. pp. 1-17, Sept 1988 EGAN, J.L, and TE1XEIRA, Tj.: Writing a UNIX Deme Driver, 2nd ed . New York John Wiley, 1992. EL GAMAL, A.: "A Public Key Cryptosyncm and Signature Scheme Based on Discrete Logarithms," IEEE Tians on Information Fkeoi\, vul

IT-31. pp 469-472, July 198^ ELLIS, C.S.: "The Case lbr Higher-Level Power Management." Proc Hot Topic in Operating Systems VII, IEEE, pp 162-167 [999 ENGLER, D.R., GUPTA. S,K_ and KAASHOF.K, Mj.; "AVM. Application-Level Virtudl Memory," Proc Hot'lopics tn Operating Sntems V, lEEE.pp 72-77, 1995d ENGLER, D., CHELF, B., CHOU, A.. HALI,EM, S.: "Checking System Rules Using System-Specific Programmer- Written Cnmpiler Extensions." Pro< Fourth S\mp on Operating Systems Design and Implementation. USENIX, pp 1-16, 2000 ENGLER. D.R, and KAASHOEK, M.F.: "Exterminate All Operating SysLeni Abstrac- Abstractions," Proc. HotT,'pirs in Operating S^stfm* V, IEBE, pp. 78-83, 1995 ENGLER, D.R., KAASHOEK, M.F., and O'TOOLE, J. Jr.: "Exokernel An Operating Sys tern Architecture for Application-Level Resource Management,- Pro<- 15th Svmp.'oH Operating Systems Principle^, ACM, pp. 251-266, 1995

918 RFADING LIST AND BIBLIOGRAPHY CHAP 13 FAKN.S.: Graph Algorithms, Potomac. MD CtimpuLer Science Press. [979 FABRY. R.S.: "'Capability-Based Addressing, Commun. of the ACM, vol. 17, pp 403-4!2.July [974 tEF.LEY, M.J. MORGAN, W.E., PIGHIN, F.H., KARLlN, A.R., LEW, H.M., and THEK- KATH.C.A.-. "Implementing Global Memoij Management in a Workstation Cluster." Prix 15thSymp. on Operating S\stems Principles, ACM, pp 201-212, 1995 FERGUSON, D., VEMINI. Y, and N1KOLAOI. C: "Micraeconomic Algorithms lor Load Balancing in Distributed Computer Systems." Proc Eighth In! I Conf. on Distributed Computing Systems, IEEE, pp 49! -499, [988. FF.u.STAL, E.A.; "The R«,e Research Computer—A Tagged Architectuie,"' Proc AFIPS Conf, AFlPS. 1972 FIJNN, J., and SATVANARAVANAN, M.: "Energy-Aware Adaptation lor Mobile Appli- Applications." Proc 17th Symp. on Operating Systems Principles. ACM. pp 48-63. 1999. FLUCKIGER, F.: Understanding Networked Multimedia, tipper Saddle River, NJ' Pren- oce Hall, 1995 FORD, B., fflBLER, M., LEPREAU, J.. TULI.MAN, P^ BACK, C, CLAWSON, S-: "Mfcro- kemels Meet Recursive Viriual Machines," Proc Second S\mp on Operating S\s- terns Design and Implementation. VSEN\X,pp 137-151, 1996 FORD, B., and SUSARLA, S.: "CPU inheritance Scheduling," Proc Second Svmp on Operating Systems Design and Implementation, USENIX, pp 91-105, 1996 FORD, B., BACK, G., BENSO^, G., LEPREAL, J-, UN, A., SHIVERS, O.; The Mux OSkit A Substrate for Kernel and Unguage Research." Pr>J< 17th Symp on Operat- Operating Systems Principle!,, ACM. pp 38-51, 1997.

FOTHERINGHAM, J.: "Dynamic Storage Allocation in the Atlas Including an Automatic Use ofa Backing Store,- Commun ofthe ACM\ vol 4,pp 4M-436. Oct. 1961. GAFSI, J., and BIERSACK, E-W-: "A Nove) Replica Placement Strategy for Vtdeo Servers," Proc. Sixth Int'l Workshop on Inieraaive and Distrib Multimedia Systems, ACM, pp. 321-335 GEIST, R., and DANIEL, h.: "A Continuum o(Disk Schcdultng Algorithms " ACM Tram on Computer Systems, vol. 5, pp. 77-92. Feb 1987 GELERNTER, D.: -Generative Communicttion in l.tndd," ACM Trans on Piogrammmg Languages and Systems, vol. 7. pp. 80-112, Jan 19R5 GHORMLEY, D., PETROU, D., RODRIGfES, S., VAHDAT, A., and ANDERSON, T.E.: "SLIC An Extensible System for Commodity Operating Systems," Proc USE.VIX Annuallech Can}., USENIX, pp 39-46. 1998. GIBSON, G.A., and VAN METER, R.; -'Network Attached Storage," Commun. of rhe ACM, vol 43, pp, 37-45, Nov. 2000.

SEC 13 2 ALPHABETICAL BIBLIOGRAPHY 9 GILL DS,ZHOU, S-, and SANDHL'. H.S.: "A Case Study of File System Workload r Large-scale Distributed Environment." Pnx 1994 Couj on Mwuremema Modeling of Compute, Systems, ACM, pp 276-277. 1994 GOLDBERG, L.A., GOLDBERG, P.W., PHILLIPS, C.A,, and SORKIN, G.B.: 'Constructi Virus Phylogemes " Journal of Algorithms, \o\. 26, pp. 188-208^-lan 1998. GOLDEN, D., and PECHl'RA, M.: "The Structure of Microcomputer File Systems,,'" Cotn- mun. of the ACM. vol. 29. pp 222-230, March 1986 GONG, L-: Inside Java 2 Platform Security. Reading, MA. Addi son-Wesley, 1999 GOODHEART, B., and COX, J.: The Magic Gaiden Explained, Upper Saddle Rtver, NJ: Prentice HaJJ, J994. GOVIL, K., TEODOSIU, D, HUANG, V., and ROSENBLUM. M.: "CeJJuJar Dkco Resource Management Using Virtual Clusters on Shared-Memory Multiprocessors." Proc HthSymp on Operating Svstems Principles, ACM, J999, pp. J54-J69 GOYAL, P,, GUO, X., and VIN, H,M; "A Hierarchical CPU ScheduJer for MuJtimedia Operating Systems," Prot Second Symp i,n Operating Systems Design and Imple- Implementation, USENIX.pp 107-J2J.J996 GRAHAM, R.: "Use of High-LeveJ Languages lor System Programming," Project MAC Report TM-13.M.I.T, Sept 19711 GRIFFIN, J.L., SCHLOSSER, S.W., GANGER, G,R., and NAGLE, D.F.-. 'Operating System Management of MEMS-based Storage Devioes,,'" Proc Fourth S\mp on Operating Systems Design and Implementation. USENlX, pp 87-J02, 2000 GRIMM, R., and BERSHAD, B: "Security for Extensible Systems," Proc Hot Topii i in Operating Systems VL IEEE, pp 62-66, 1997

GR1WODZ, C, BAR. M-, and WOLF, L.C.: "Long-Term Movie PopuJanty Models m Video-on-Demand Systems," Proc. Fifth Int'l Conf on Multimedia, ACM. pp. 349- 357, 1997. GROPP, W., LUSK, E., and SKJELLUM, A,: Uvng Ml'!- Portable Parallel Programming with the Message Pasting Interface. Cambridge, MA M.IT.Press, 1994. GROSSMAN, D,, and SILVERMAN, H,; "PJactment of Records on a Secondary Storage Device to Minimize Access Time," Journal of the ACM, vol 20, pp 429-438, J973 HAFNER, K,,andMARKOFF,J,; Cyberpunk, New York" Simon and Schuster, 1991 HAND, S.M: -Self-Paging rn the Nemesis Operating System." Proc Third S\mp. on Operating Systems Design and Implementation. USENlX, pp. 73-86, 1999 HARBRON, T.R.: File System*. Upper Saddle River, NJ Prentice Hall, 19K8 HARCHOL-BALTER, M., and DOWNEY, A.B.; "Exploiting Process Lifetime Distributions, for Dynamic Load Balancing," Proc. S1GMETR1CS Con} on Measurement and Modeling of Computer Systems, ACM, pp. 13-24, 1996

920 READING LIST AND BIBLIOGRAPHY CHAP 13 HARRISON,M.A., RUZZO, W.L,, and ULLMAN, JJ).: -'Protection in Operating System.,," Common, of the ACM, vol. 19, pp. 461 -471, Aug. (976. HART, J,M.: Win32 System Programming, Reading, M<\: Add-on-Wesley, 1997. HARTIG, H., HOHMUTH, M-, UEDTKE, J., sind SCHONBERG, S,: "The Performance of Kernel-Based System*-." Proc 16th Syrup- on Operating Syilems Principles. ACM, pp 66-77, 1997 HARTMAN, J.H., and OUSTERHOL'T, J,K.; "The Zebra Striped Network File System." T,am. on Computer Systems, vol 13. pp 274-310, Aug. 1995 HAUSER. C, JACOBI, C, THEIMEB, M, WELCH, B., and WEISER, M.: "Using Thread*. in Interactive Systems A Case Study," Prot. 14th Symp. an Operating System Pnn- uplc,, ACM, pp 94-1A5. 1193, HAVENDER, J.W.: "Avoiding Deadlock in Multitasking Systems.'" IBM Systems Journal. vol. 7, pp. 74-84,1968 HEBBARD, B., el al.: "A Penetration Analysis ot the Michigan Terminal System," Operat- Operating System-, Review, vol 14, pp. 7-20, Jan 1980 HEIDEMANN, J.S., and POPEK. GJ.: "File-System Development with Stackable Layers," Trans, on Computer System/,, vol. 12. pp. 5X-89. Feb 1994. HEYBEY, A., SUI.UVAN, M., ENGLAND, P.: "Calliope A Distributed Scalable Mul- Multimedia Server,"/W USENIX Annual Tech Conf, USEN1X, pp 75-86. |996. HIPSON, P.D.: Mastering Windows 2001) Regism, Alameda, CA: Sybcx. 2000 HOARE, C.A.R.: "Monitor^., An Operating System Structuring Concept," Comnmti 'if

the ACM, vol. 17, pp. 549-557, Oct. 1974" Erratum in Commun of the ACM. vol 18, p. 95, Feb 1975. HOLT.R.C-: "Some Deadlock Properties of Computer Systems," Computing Survey, vol 4, pp. 179-196, Sept. 1972 HONEYMAN, P., ADAMSON, A., COFFMAN. K., JANAKIRAMAN, J., JERDONEK, R., and REES, J.: "Secure Videoconferencing," The Seventh USENIX Se(urttv 5ym;>, USENIX, pp. 123-H3, 1998. HOWARD, J.H., KAZAR, MJ., MENEES, S.G., NICHOLS, D.A., SATYANARAVANAN, M., SIDEBOTHAM, R.N., and WEST, MJ-: "Scale and Performance in a Distributed File System." ACM Trans on Computer Systems, vol. 6. pp. 55-81, Feb. 1988 HUCK, J., and HAYS, J.: -Architectural Support tor Translation Table Management m Large Address Space Machines " Proc. 20th Int'l Symp on Computer Architecture ACM.pp 39-50,1991 HUTCHINSON, N,C\, MANLEY, S., FEDERWI.SCH, M-, HARRIS, G, HlTZ, D., KLEI- MAN, S., and O'MAU-EY, S.: -Logical vs. Physical File System Backup," Proc Third Symp on Operating Systems Design and Implementation, USENIX pp 239-249, i«99.

SEC 13.2 ALPHABETICAL BIBLIOGRAPHY 921 IEEE: Information Technology—Portable Operating System interface (POS1X). Part I: System Application Program interface (API) [C language], New York: Institute of Eleurical and Electronics Engineers, Inc., 1990 ISLOOR, S.S., and MARSLAND, TA.: "The Deadlock Problem: An Overview," Com- Computer, vol. 13,pp.58-78, Sept. 1980. ITZKOVITZ, A., and SCHUSTER, A.: "MullrView and Millipage—Fine-Grain Sharing ill Page-Based DSMs," Prac. Third Symp. on Operating Systems Design and Implemen- Implementation, USENIX, pp. 215-228,1999. IVENS, K,: Optimizing the Windows Registry, Foster City, CA- IDG Books Worldwide, 1998. JOHNSON, KA., KAASHOEK, MJ, and WALLACH, DA,: "CRL: High-Performance All-Software Distributed Shared Memory," Proc. I5th Symp. on Operating Systems Principles, ACM, pp. 213-226,1995, JOHNSON, N.F., and JAJODIA, S,: "Exploring Steganograph)': Seeing the Unseen," Com- Computer, vol. 31, pp. 26-34, Feb. 1998. JONES, M.B., ROSU, D,, andROSU, M.-C: "CPU Reservations and Time Constraints Effi- Efficient, Predictable Scheduling of Independent Activities," Proc. 16th Symp. on Operating Systems Principles, ACM, pp. 198-211, 1997. KAASHOEK, MJ., ENGLER, D.R., GANGER, G.R., BRICENO, H,, HUNT, R,, MAZIERES, D-, PINCKNEY, T,, GRIMM, R., JANNOTTI, J., and MACKENZIE, K.: "'Application Performance and

Flexibility on Exokemel Systems," Proc. I6'h Symp, on Operating Systems Principles, ACM, pp. 52-65, 1997. KABAY, M-: -Flashes from the Past," Information Security, p. 17, 1997. KALLAHALLA, M., and VARMAN, PJ.: -Optimal Read-Once Parallel Disk Scheduling," Proc Sixth Workshop on I/O in Parallel and Distributed Systems, ACM pp 68-77, 1999, KARACALI, B,, TAI, K.C., and VOUK, M.A.: 'Deadlock Detection of EFSMs Using Simultaneous Reachability Analysis," Proc. Infl Conference on Dependable Systems and Networks (DSN2000), IEEE, pp. 315-324, 2000. KARUN, A.R., LI, K,, MANASSE, M.S., and OW1CKI, S,: '-Empirical Studies of Competi-Competitive Spinning for a Shared-Memory Multiprocessor," Prac- 13th Symp, On Operating Systems Principles, ACM, pp. 41-54, 1991. KARUN, A.IL, MANASSE,M,S,, McGEOCH, L,, andOW1CKI,S,: 'Competitive Random- Randomized Algorithms for Non-Uniform Problems," Proc. First Annual ACM Symp on Discrete Algorithms, ACM, pp. 301-309, 1989. KARPOVICH, J.F., GRIMSHAW, A.S., and FRENCH, J.C.: "Extensible File System (ELFSJ: An Object-Oriented Approach to High Performance File I/O," proc. Ninth Annual Conf. on Object-Oriented Programming Systems, Language, and Applications ACM, pp, 191-204, 1994.

922 READING LIST AND BIBLIOGRAPHY CHAP 13 KATCHER, DJ,, KETTLER, K.A., and STROSMDER, J.K.: "Real-Time Operating Sys-Systems for Multimedia Processing," Prac. Hot Topics tn Operating Systems V, IEEE, 1995 KAUFMAN, C, PERLMAN, R., and SPECINER, M-: Network Security, Upper Saddle River, NJ: Prentice Hall, 1995. KAVl, K., BROWNE, J.C., and TR1PATHI, A,: 'Computer Systems Research: The Pres- Pressure]S On," Computer, vol. 32, pp. 30-39, Jan 1999, KELEHER, P., COX, A., DWARKADAs, S,, and ZWAENEPOEL, W.: "TreadMarks: Distri- Distributed Shared Memory on Standard Workstations and Operating Systems," Prac. USENIX Winter 1994 Conf., USENIX, pp. 115-132, 1994. KERNIGHAN, B.W., and PIKE, R,: The UNIX Programming Environment, Upper Saddle River, NJ- Prentice Hall, 1984. KHALIDI, Y.A., and NELSON, MJV,! "Extensible File Systems in Spring," Prac. J4th Symp. on Operating Systems Principles, ACM, pp. 1-14, 1993, KLEIN, D.V.: "Foiling the Cracker: A Survey of, and Improvements to, Password Secu- Security," Proc. UNIX Security Workshop II, USENIX, Summer 1990. KLEINROCK, L.: Queueing Systems, Vol. I, New York: John Wiley, 1975. KLINE, BX-, and GLINERT, EJV; "Improving GUI Accessibility for People with Low Vision," Proc. Conf.

on Human Factors in Computing Systems, ACM, pp. 114-121, 1995. KNTJTH, D.E.: The Art of Computer Programming. Volume I: Fundamental Algorithms, 2nd Ed., Reading, MA: Addison-Wesley, 1973. KOCHAN, S.G., and WOOD, P.H.: UNIX Shell Programming, Indianapolis, IN: Hayden Books, 1990. KRAVETS, R., and KRISHNAN, P.: "Power Management Techniques for Mobile Com- Communication," prac. Fourth ACM/IEEE Int'i Conf on Mobile Computing and Net- Networking, ACM/IEEE, pp. 157-168,1998. KRISHNAN, R.: "Timeshared Video-on-Demand: A Workable Solution," IEEE Mul- Multimedia, vol. 6, Jan.-March 1999, pp. 77-79. KRUEGER, P., LAI, T.-H,, and DIXIT-RADIYA, V.A.: 'Job Scheduling is More Important than Processor Allocation for Hypercube Computers," IEEE Trans, on Parallel and Dtstr. Systems, vol. 5, pp. 488-497, May 1994. KUMAR, V.P., and REDDV, S.M.: "Augmented Shuffle-Exchange Multistage Interconnec- Interconnection Networks," Computer, vol, 20, pp, 30-40, June 1987. KWOK, Y.-K., AHMAD, I.: "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," Computing Surveys, vol. 31, pp. 406-471, Dec. 1999. LAMPORT, L.i '"Password Authentication with Insecure Communication,- Common of the ACM, vol. 24, pp. 770-772, Nov. 1981.

SEC. 13.2 ALPHABETICAL BIBLIOGRAPHY 923 LAMPSON, B.W.: "A Scheduling Philosophy for Muniprogramming Systems,'" Commun of the ACM, vol II,pp 147-360. May I96K LAMPSON, B.W.: "A Note on the Confinement Problem," Commun. of the ACM, vol 10, pp 613-615, Oct. 1971. LAMPSON, B.W.: "Hints lor Computer System Design.'" IEEE Software, vol. I, pp. II- 28, Jan 1984 LAMPSON, B.W., and STURG1S, H.E.: "'Crash Recovery in a Distributed Data Storage System.'" Xerox Palo Alto Research Center Technical Report. June 1979 LANDWEHR, CX.i "Formal Models, tit Computer Security," Computing Surveys, vol. 13, pp 247-278, Sept, 1981 LANGENDOEN, K., BHOEDJANG, R., and BAL, H.E,: "Models for Asynchronous Mes- Message Passing," IEEE Concurrency, vol 5, pp 28-37, April June 1997 LEBECK, A.R., FAN, X., ZENG, H., ELLIS. C.S.: "Power Aware Page Allocation.'" Proc. Ninth Int'l Conf. on Architectural Support jorProgramming Languages and Operat- Operating Systems, ACM. 2000 LEE, J.Y.B.! "Parallel Video Servers- A Tutorial," IEEE Multimedia vol. 5, pp 20-28, April-June 1998 LEE, W-, SU, D,, WLJESEKERA, D,, SRIVASTAVA, J-, KENCHAMMANA-HOSEKOTE, D., and FORESTL M.: "Experimental Evaluation ot PFS Continuous Media File Sys- System," Proc. Sixth Int'l Conj

on Information and Knowledge Management, ACM. pp 246-253, 1997. LESLIE, 1.. MtAULEV, D., BLACK, R., ROSCOE, T., BARHAM, P, EVERS, D., FA1R- BA1BNS, R., and HYOF.N, E.: "The Design and Implementation of an Operating Sys- System to Support Distributed Multimedia Applications,"' IEEE Journal on Selected Areas tn Commun.. vol. 14, pp 1280-1297, July 1996 LEVIN, R., COHEN, E.S., CORW1N, W.M., POLLACK, FJ., and v/ULV. W.A.: "Policy/Mechanism Separation in Hydra," Proc. Fifth Symp on Operating Synems Principles, ACM, pp. 132-140, 1975 LEW1NE, D.-. POS1X Programmer's Guide. Seba-stopol, CA. O'Reilly & Associates. 1991 Ll, K.: "Shared Virtual Memory on Loosely Coupled Multiprocessors." Ph.D. Thesis. YaleUniv, 1986 Ll, K., and HUDAK, P.: "Memory Coherence in Shared Virtual Memory Systems," ACM Trans, on Computer Systems, vo\ 7,pp 321-359, Nov 1989 LI, K., KUMPF, R.. HORTON, P., and ANDERSON, T,: "A Quantitative Analysis of Disk Drive Power Management in Portable Computers," Proc. 1994 Winter Conf. USENIX. pp. 279-291. 1994. LlEDTKE. J.: -Improving IPC by Kernel Design,- Proc I4ih Symp. on Operating Sys- Systems Principles, ACM, pp. 175-188, 1993.

924 READING LIST AND BIBLIOGRAPHY CHAP 13 LJEDTKE, J.: -On Micro-Kernel Construction." Proc. 15th Svmp. on Operating Systems Principles, ACM, pp. 237-250, 1995 LIEDTKE, J.: -Toward Real Microkernels,- Commun of the ACM, vol 39. pp 70-77, Sept. 19%. UNDE, R.R.: "Operating System Penetration." Proc \FIPS National Computer Conf, AFIPS. pp. 361-368, 1975 LIONS, J-: Lions' Commentary on Unix 6th Edition, with Source Code. San Jose, C'A: Pecr-toPeeT Communications, 1996. LiU, CX., and l.AYLAND, J.W.: -Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," Journal of the ACM, vol 20, pp. 46-61, Jan. 1973. LO, V.M.: "Heuristic Algorithms for Task Assignment in Distributed Systems," Proc Fourth Int 7 Conf. on Distributed Computing Systems, IEEE, pp. 30-39, 1984 LORCH, J.R., and SMITH, A.J.: -Reducing Processor Power Consumption by Improving Processor Time Management In a Single-User Operating System," Proc. Second Int'I Conf. on Mobile Computing and Networking. ACM, pp 143-154, 1996. LORCH, J., and SMITH, AJ.i "Apple Macintosh's Energy Consumption." IEEE Micro. vol. 18, pp. 54-63, Nov/Dec I998 LOUGHER, P., PEGLER, D., and SHEPHERD, D.: 'Scalable Storage Servers for Digital Audio and Video," Proc IEE Int'I Conf on Storage and Recording Svttems, London: lEE.pp. 140-143, 1994. LU, Y.-H., SIMUNlC, T.,

DE MlCHELl, G.: "Software Controlled Power Management," Proc. Seventh Infl Workshop on Hardware/Software Codesign, ACM, pp 157-161, 1999. LUDWlG, M.A.: The Giant Black Book of Computer Viruses. 2nd ed, Show Low. AZ. American Eagle Poblicatiom, 1998 LUMB, C.R., SCHINDLER, J., GANGER, GJt., and NAGLE, D.F.i Towards Higher Drsk Head Utilization: Extracting Free Bandwidih from Busy Disk Drives," Proc. Fourth Symi>, on Operating Systems Design and Implementation. USENIX. pp 87-102, 2000. MAEKAWA, M., OLDEHOEFT, A.E., and OI.DEHOEFT, R.R.: Operating S\stems Advanced Concepts, Menlo Park. CA. Benjamm/Cummings, 1987 MALKEWITZ, R.: "Head Potnting and Speech Control as d hands-free interface to Desk- Desktop Computing," Proc Third Int'l Conf on Aisiw\e Technologies, ACM. pp 182- 188, 1998. MANARIs, B-, and HARKREADER, A.: "SUlTEKeys. A Speech Understanding Interface for the Motor-Control Challenged," Proc. Third Int'l Conf. on Assutive Technologies ACM. pp. 108-115, 1998, MARK, A.R.: "The Development of Destination-Specific Biometnc Authentication,' Proc Tenth Conf. on Computers, Freedom and Privacy, ACM pp. 77-80, 2000.

SEC 13 2 ALPHABETICAL BIBLIOGRAPHY 925 MARKOW1TZ, J.A.: "'Voice Biometrics," Commun. of the ACM. vol 43. pp 66-73, Sept 20(H MARSH, B.D., SCOTT, M.L., LEBLANC, T.I., and MARKATOS, E.P.: 'First-Class Lser- Level Threads," Proc I3th Symp on tyeraung Systems Principles, ACM, pp. H0- 121. 1991 MATTHEWS, J.N, ROSELL1, P., COSTELI.O. A.M., WAN(i, R.Y., and ANDERSON, T.E-: "Improving the Performance of Log-Structured File Systems with Adaplive Methods" Proc. I6ih Symp onOperatmK Systems Prm.,ACM.pp 23S-251, 199? MAXWELL, SX.: Linux Core Kerne! Commentary, Scottiddle, AZ. Conolis, 1999. MAZ1ERES, D., KAMINSKY, M., KAASHOEK, M.¥., and W1TCHEL, E,; "Separaling Key Management from File Sjstem Security.- Proc !7th Symp. on Operating Systems Principle, ACM. pp 124-139,1999. McDANiEL, T.: "Magneto-Optical Data Storage/' Commuii. of the ACM, vol 43. pp 57- 63, Nov 2000. MCKUSICK, M.K., BOST1C, K., KAKELS, MJ., and QUARTERMAN, J.S-: The Deugn and Implementation of the 4.4 BSD Operating Svstem. Reading, MA. AddUon-Wesley, 1996. McKUSICK, MJ, JOY, W.N., LEFFLER, SJ., and FABRY, R.S.i "A Fast File System for UNIX." ACM Trans, on Computer System^ ^(A 2. pp. 181-197, Aug 1984 MEDINETSJX: UNIX Shell Programming Tools, New York, NY: McGraw-Hill, 1999 MELLOR-CRUMMEY, JM, and SCOTT, M.L.:

"Algorithms for Scalable Synchroniza- Synchronization on Shared-Memory Multiprocessors," ACM Trans on Computer Systems, vol. 9, pp. 21-65, Feb. 1991 MERCER, C.W.: "Operating System Support for Muiiimedu Applications,-' Proc Second Int'l Confon Multimedia, ACM, pp 492-493. 1994. MILLER, F.W.: "pk: A POSIX Threads Kernel,- FREENIX Track. USENIX Annual Technical Conference. USENIX, pp. 179-182, 1999 M1LOJ1CIC, D.: "Operating Systems Now and in the Future," IEEE Concurrent^. vol 7. pp. 12-21, Jan.-March 1999 M1LOJ1CIC, D.: "Security and Privacy," IEEE Concurrence, vol. 8, pp 70-79, AptH-Jtme 2000 MONROSE, F., and RUBIN, A.: "Authentication Via Keystroke Dynamics," Proc Conf. on Computer and Communications Security. ACM. pp 48-56, 1997. MORRIS, J.H., SATYANARAYANAN, M.. CONNER, M.H., HOWARD, J,H., ROSENTHAL, D.S., and SMITH, F.D.: "Andrew. A Distributed Personal Computing Environment." Commtn oftheACM.vol 29, pp 184-201. Marth 1986 MORRIS, R., and THOMPSON, K.: "Password Security. A Case History,- Commun of the ACM. vol. 22, pp. 594-597, Nov. 1979.

926 READING LIST AND BIBLIOGRAPHY CHAP 13 MUL1XNDER, S.J.. and TANENBAUM, A.S.: "Immediate Files," Software—Practice and Experience, vol. 14, pp. 365-368, Apnl 1984 MYERS, A.C., and L1SKOV, B.: "A Decentralized Model for Information Flow Control," Pro,- 16th Symp. on Operating Systems Principles, ACM. pp. 129-142. 1997. NACHF.NBERG, C.: "Compute/ V*rus-Antivirus Coe volution" Commun. of the ACM, vol. 40. pp 46-51, Jan 1997 NEMETH, E., SNYDER, G, SEEBASS, S., and HE1N, T.R.: UNIX S\stem Administration Handbook, 2nd ed., Upper Saddle River, NJ Prentice Hall, 2000. NEWHAM, C, and ROSENBLATT, B.: Learning the Bash Shell. Sebastopol. CA; O'Reilly & Associates. 1998. NEWTON, G.: "Deadlock Prevention, Detection, and Resolution. An Annotated Bibliogra- Bibliography," Operating Systems Review, vol. 13. pp 33-44, April 1979 NIEH, J., and LAM, M.S,: "The Design, Implementation and Evaluation of SMART a Scheduler for Multimedia Applications." Proc I6th Symp on Operating Systems Principles, ACM, pp. 184-197, 1997. NIST (National Institute or SUndards and Technology)t RPS Pub 1801, 1995 NG, S.W.: "Advances in Disk Technology. Performance Issues," Computer, vol. 31, pp. 75-81, May 1998. NORTON, P., MUELLER, J., and MANSFIELD, R.: Complete Guide to Windows 2000 Professional, Indianapolis, IN Sams, 2000- OKI, B., PFLUEGL, M., SIEGEL, A., and SKEEN, D.; "The Information Bus

—An Archi- Architecture for Extensible Distributed Systems," Proc 14th Symp. on Operating Systems Principles, ACM, pp. 58-68, 1993. ONEY, W.: Programming the Microsoft Windows Driver Model, Redmond, WA. Micro-Microsoft Press, 1999. ORGAN1CK, EJ.; The Multtcs System, Cambridge. MA: M.I T. Press, 1972. ORLOV, S.S.: "Volume Holographic Data Storage," Commun. of the ACM, vol. 43, pp. 47-54, Nov. 2000- OUSTERHOUT, J.K.: -'Scheduling Techniques for Concurrent Systems," Proc Third Int'i Conf on Dlstrib- Computing Systems, IEEE. pp. 22-30, 1982 PAL V.S., DRUSCHEL, p., and ZWAENEPOEL, W.: "IO-Lite- A Unified I/O Buffering and Caching System," Trans. on Computer Systems, vol. 18, pp 37-66, Feb. 2000. PAKIN, S,, KARAMCHET1, V., CH1EN, A.A,; 'Fast Messages (FM): Efficient, Portable Communication for Workstation Clusters and Massively Parallel Processors," IEEE Concurrency,\ol 5, pp. 60-73, Apni-June 1997. PANKANT1, S., BOLLE, R,M., and JAIN, A.: "Biomeirics: The Future of Identification," Computer, vol. 33. pp. 46-49, Feb. 2000.

 PATE, S.D.; UNIX Internals A Practical Approach, Reading, MA; Addi son-Wesley, I996. PATTERSON, R.H., GIBSON, GX, GINTING, E., STODOLSKY, D., and ZELENKA, J.: "Informed Prefetehing and Caching," Proc. I5th Symp. on Operating Systems Princi- Principles, ACM, pp 79-95, 1995. PATTERSON, DA., GIBSON, G., and KATZ, R.; "A Case for Redundani Arrays of Inex-Inexpensive Disks (RAID)," Proc. ACM SIGMOD Im'i. Conf. on Management of Data, ACM, pp. 109-166,1988. PENTLAND, A., and CHOUDHURY, T.; "Face Recognition for Smarl Environments," Computer, voi 33, pp. 50-55, Feb 2000 PETERSON, G.L.: "Myth* aboul the Mutual Exclusion Problem," Information Processing Letters, voi, 12, pp. 115-116, June 1981 PETROU, IX. M1LFORD, J., and GIBSON, G.-. "Implementing Loitery Scheduling," Proc. USENIX Annual Tech. Conf., USENIX, pp. 1-14,1999. PETZOLD, C: Programming Windows, 5th ed., Redmond, WA: Microsoft Press, 1999 PFLEEGER, C.P.: Security in Computing, 2nd ed. Upper Saddle River, NJ Prentice Hall, 1997. PH1LBIN, J., EDLER, J., ANSHUS, O J., DOUGLAS, C.C., and LI, K.: 'Thread Scheduling for Cache Locality," Proc. Seventh fnfi Conf. on Architectural Support for Program- Programming Languages and Operating Systems, ACM, pp. 60-71, 1996. PRECHELT, L.: "An Empirical Comparison of Seven Programming Languages," Com- Computer, vol. 33, pp, 23-29, Oct. 2000. PHOTIC, J,, TOMASEV1C, M., and MILUTINOVIC, V.:

Distributed Snared Memory: Con- Concepts and Systems, Los Alamitos, CA: IEEE Computer Society, J998. RAWSON, F.L. m: "Experience with the Development of a Microkernel-Based, Multi- Server Operating System," Prac. Hot Topics in Operating Systems VI, IEEE pp. 2-7 1997. RECTOR, B.E., and NEWCOMER, J.M.: Win32 Programming, Reading, MA. Addison- Wesley, 1997. REDDY, A.LJJ., and WYLLIE, J.c.: "Disk Scheduling in a Multimedia I/O System," Proc. ACM Multimedia Conf, ACM, pp. 225-233,1992. REDDY, A.LJJ., and WYLLIE, J.C,; "I/O Issues m a Multimedia System,- Computer vol. 27, pp. 69-74, March 1994. RITCHIE, D.M.: "Reflections on Software Research," Commun. of the ACM voi 27. pp, 758-760, Aug. 1984, RITCHIE, D.M., and THOMPSON, K.: "The UNIX Timesharing System " Commun of the ACM, vol. 17, pp, 365-375, July 1974, RIVEST, RA.: "The MD5 Message-Digest Algorithm," RFC 1320, April 1992,

RlVEST, R.L., SHAMIR, A., and AD1.EMAN. L-: "On a Method lor Obtaining Digital Sig-Signatures and Public Key Cryptosy stems,- Cominurt. of the ACM, vol 21, pp 120-126, Feb 1978 KOBBINS, A.: UNIX in a Nutshell. A Desktop Quuk Reference far SVR4 and Salary 7, Sehastopol. CA" O'Reilly & Associates |W ROMPtMJIANNAKIS, Y., NERJES, G., MUTH. P., PATERAKIS, M.. TRIANT\F|LLOU, P., and WEIKDM, G.: "Disk Scheduling for Mixod-Media Workloads in a Multimedia Server,- Proc. Sixth Iitfl Conf. on Multimedia. ACM. pp 297 -302, 1998 KOSENBMIM. M., and OUSTERHOLT, J Jt.: "The Design and Implementation of j Uig- Structured File Sysiem,"' Proc 13th Symp. on Oper S}s Pnn . ACM. pp I-H.I99I ROSELLI, V., and LORCH, J.R.: "A Comparison of File System Workloads," Proc USENIX Annual Tech Conf, USENIX, pp 41-54, 200A ROZ1ER, M, ABBROSS1MOV, V^ ABMAND, F., BOULE, 1., GIEN, M-, GUILLEMONT, M-, HERRMANN, F., KAISER, C, LEONARD, P., IANGLO1S, S., and NEUHAUSFJl, W.: "Chorus Distributed Operating Systems/' Computing S\s-iems; vol. I, pp 305- 379. Oct. 1988. RUEMMLER, C.,and WlLKES, J.: "An Introducnon to Disk Drive Modeling," Computer. vol.27, pp. 17-28, March 1994 SACKMAN, H., ERIKSON, WJ., and GRANT, E.E.; "Exploratory Experimental Studies Comparing Online and Offline Programming Performance." Commun of the ACM. vol. II,pp. 3-11,Jan. I968. SALTZER, J.H>: "Protection and Conirol of Information Shanng in MULTICS," Com- mun. of the ACM, vol. 17, pp. 388-402. July 1974. SALTZER, J.H., REED, DJ-, and CLARK, DJJ.r "End-to-End Atgumenls in

System Design," Trans, on Computer Systems, vol. 2. pp 277-277, No% 1984. SALTZER, J.H., and SCHROEDER, M.D.: "The Protection of Inrorm.ition in Computer Systems," Proc. IEEE. vol. 63. pp 1278-1108. Sepi 1975. SALUS,P.H.: "UNIX At25."Bj(e,vol 19, pp 75-82, Oct. I994 SANDHU, R.S.: •'Lattice-Based Access Control Models," Computer vol. 26 pp 9-19. Nov. 1993 SANTRY, D., FEELEY, MJ., HDTCHlNSON, N£., VEITCH, A.C : "Elephant. The File Sysiem That Never Forgets," Proc Hot Topic in Operating Sy.stems V/l, IEEE, pp. 2-7. 1999a. SANTRY, D., FEELEY, M., HUTCHINSON, N_ VFJTCH, A.C, CARTON, R.W., and OFIR, J.: "Deciding When to Forget in the Elephant Hie System." Pro< 17th S\mp on Operating Systems Principles, ACM, pp JI0-121, 1999b. SATYANARAVANAN, M., HOWARD, J.H., NICHOLS, D.N, SjDEBOTHAM, R.N., SPEC- TOR, A.Z., and WEST, MJ.: "The JTC Distributed File Sysiem; Principles and Design." Proc. of the Tenth Symp. on Operating System Pnn.. ACM, pp 35-50, 1985.

SEC. 13 2 ALPHABETICAL BIBLIOGRAPHY 929 SAVAGE, S., BURROWS. M., NELSON, G., SOBALVARRO, P., and ANDERSON, T.: "Eraser: A Dynamic Data Race Detector for Multithreaded Programs," Tram on Computer Systems, w). 15. pp. 391-411. Nov. 1997. SCALES, D.J., and GHARACHORLOO, K.= "Towards Transparent and Efficient Soft*are Distributed Shared Memor>," Proc I6th S^mp on Operating Systems Principles, ACM, pp. J57-169. 1997 SCHMIDT, Bit., LAM, MS., and NORTHCUTT, J.D.: 'The Interactive Performance or Slim A Stateless, Thin-Client Archilecture," Proc /7th Symp. on Operating Systems Principles, ACM, pp 12-47. 1999 SCOTT, M., LeBLANC, T., and MARSH, B.: "Multi-model Parallel Programming in Psyche," Proc Second ACM Symp. on Print iplei and Practice of Parallel Program- Programming, ACM. pp. 70-78, 1990. SEAWR1GHT, L,H., and MACKINNON, R.A.-. "VM/37O—A Study of Multiplicity and Usefulness," IBM Systems Journal, vol. 18. pp. 4-17, 1979. SEGARRA, M.-T., and ANDRl, F.: "MFS: a Mobile File System Using Generic System Services," Proc. 1999 Symp. on Applied Computing, ACM, pp. 419-420, 1999 SELTZER, M,, ENDO, Y., SMALL, C, and SMITH, K,: 'Dealing with Disaster Surviving Misbehaved Kernel Exiensjonsm'" Proc, Second Symp on Operating System Design and Implementation. (JSENIX, pp 213-227. 1994. SHENOY, PJ,, and V1N, H.M.: "Efficient Striping Techniques for Variable Bit Rale Con- Continuous Media File Servers." Perf. Evai Journal, vol 38, pp. 175-199, 1999 SHENOY, P.J., GOYAL, P., and ViN.

H.M.: "Architectural Consideraiions for Next Gen- eraiion File Systems," Proc Seventh Int'l Cmf on Multimedia, ACM. pp 457-467 1999. SILBERSCHATZ, A,, GALV1N, P.B., and GAGNE, G: Applied Operating Svstem Concepts, New York: Wiley, 2000. SlMON, Rj.: Window? NT Win32 API SuperBibh. Corte Madera, CA- Sams Publishing 1997. SITARAM, D., and DAN, A.: Multimedia Servers, San Francisco CA M()rgan Kaufman 2000. SLAUGHTER, L., OARD, D.W., WARNlCK, V.L^ HARDING, J.L., and WILKERSON, GJ.: "A Graphical Interface for Speech-Based Retrieval,'- Proc Third ACM Conf. on Digital Libraries , ACM, pp. 305-306, 1998. SMALL, C, and SELTZER, M.; "MiSFIT; constructing Safe Extensible Systems." IEEE Concurrency, vol. 6, pp. 34-41, July-Sept 1998. SMITH, D.K., and ALEXANDER, R.C.; Fumbling rhe Future, How Xerox Invented, Then Ignored, the First Personal Computer, Ne* York William Morrow, 1988. SNIR, M,, OTTO, S.W., HUSS-LEDERMAN, S., WALKER, D.W., and DONGARRA, J.: MPI: The Complete Reference Manual, Cambridge, MA: M.i.t. Press, J996.

 SOLOMON, D.A., and RUSS1NOVICH, M.E.: Inside Windows 2000, 3rd ed, Redmond, WA: Microsoft Press, 2000 SPAFFORD, E., HEAPHY, K., and FERBRACHE, D.: Computer Viruses, Arlington, VA; ADAPSO, 1989. STALLINGS, W.: Operating Systems, 4th Ed, Upper Saddle River, NJ: Prentice Hall. 2001. STEENKISTE, P.A.: "A Systenialic Approach to Host Interface Design for High-Speed Networks,- Computer, vol. 27. pp 47-57, March J994. STE1NMETZ, R., and NAHRSTEDT, K.; Multimedia: Computing, Communications and Applications, Upper Saddle River, Nl: Prentice Hall, 1995. STENSTROM, P., HAGERSTEN, E., LOJA, DJ,, MARTONOSI, M., and VENUGOPAL, M-: "Trends in Shared Memory Multiprocessing," Computer, vol. 30, pp. 44-50, 1997. STETS, R., DWARKADAS, S., HARDAVELLAs, N., HUNT, G., KONTOTHANASSIS, L., PARTHASARATHY, S.. and SCOTT, M.: "Cashmere—2L Software Coherent Shared Memory on a Clustered Remote-Write Network," Proc. Ibth Symp on Operating Sys- Systems Principles, ACM, pp. 170-183,1997- STEVENS, W.R.: Advanced Programming in the WV/Jf Environment, Reading, MA: Addison-Wesley. 1992. STOLL,C.: The Cuckoos Egg: Tracking a Spy through the Maze of Computer Espionage, New York: Doubleday, 1989. STONE, H.S,,andBOKHARl,S.H.: "Control of Distributed Processes," Computer, vol. I

I, pp. 97-106, July 1978. TAI, K.C., and CARVER, R.H.: "VP: A New Operation for Semaphores." Operating Sys- Systems Review, vo) 30, pp. 5-11, July 1996. TALLURl, M., and HILL, M.D.: "Surpassing Ihe TLB Performance of Superpages with Less Operating System Support," Proc. Sixth Infi Conf on Architectural Support for Programming Languages and Operating $y\tems, ACM, pp 171-182, 1994. TALLUBI, M., H11.L, M.D., and KMALIDl, Y.X.: "A New Page Table for 64-bit Address Spaces," Prw iSlhSymp on Operating Systems Prin , ACM.pp. 184-200, 1995 TANENBAUM, A A: Operating Systems Design and Implementation, Upper Saddle River NJ: Prentiee Hall, 1987 TANENBAUM, A.S.: Computer Networks, Upper Saddle River, NJ: Prentiee Hall, 1996 TANENBAUM, A.S., VAN RENESSE, R., VAN STAVEREN, H., SHARP, GJ.. MUL- LENDER, SJ., JANSEN, J^ and VAN ROSSUM, G.: "F,xper,ences with the Amoeha Distributed Operating Sysiem," Commun, of the ACM, vol. 33, pp. 46-63, Dec. 1990 TANENBAUM, A.S., and VAN STEEN, M.R.: Distributed Systems, Upper Saddle River NJ- Prentice Hall, 2002.

SEC. 13.2 ALPHABETICAL BIBLIOGRAPHY 931 TANENBAUM, A.S., and WOODHULL, A.S.: Operating Systems. Des,gn and Implementa- Implementation, 2nded. Upper Saddle River. NJ: prentice Hall, 199? TAYLOR, R-N, MEDVIDOVIC, N, ANDERSON, K.M., wHjTEHEAD, EJ-, and ROBBINS, JE- "A Component- and Message-Based Architectural Style for GUI Software," Pmc 17ih Int'l Conf. on Software Engmeenng. ACM. pp 295-304, 1995. TEORY, TJ.: "Properties of Disk Scheduling Policies in Multiprogrammed Computer Systems." Proc AFIPS Fall Joint Computer Conf.. AFIPS, pp I-11, 1972. THEKKATH, C.A., MANN, T., and LEE, E.K.: "Frangipam A Scalable Dtslnbuted File Syslem." Proc !6rh Symp on Operating S\stems Principles, ACM, pp 224-237. 1997 THOMPSON, K.: ■'Reflections on Trusting Trust." Commun oj the ACM. vol 27. pp. 761-763, Aug 1984 TRONO, J-A.; "Comments on Tagged Semaphores,'" Operating Sviterrts Review. \o\ 3*. pp. 7-11, Oct. 2000. TUCKER, A., and GUPTA, A.; "Process Control and Scheduling Issues lor Multipro- grammed Shared-Memory Muliiprocessor\" Pmc 12th Symp. on Operating Systcm\ Principles. ACM, pp. 159-166, 1989. UHLIG, R., NAGLE, D., STANLEY, T., MUiMiE, T-, SECREST, S., and BROWN, R.: "Design Tradeoffs tor Software-Managed TLBs." ACM Trans, on Computer Systems, vol. 12, pp. 175-205, Aug. 1994 VAHAUA, U.t UNIX Internal's—The New Frontiers. Upper Saddle River NJ,

Premice Hall, 1996- VAN BUSKIRK, R., and LaLOMIA, M.: "A Comparison nf Speech and Mouse/Keyboard GUI Navigation." Conference Companion on Human Fa(ror-> in Computing Systems. ACM, p. 96, 1995 VAN DOORN, L.: The Design and Applicant* of ah Extensible Operating Syuem. Ph D. Thesis, Vnje Universiteit, Amsterdam, The Netherlands, 20A1 VAN DOORN, L, HOMBURG, P., and TANENBAUM. hS.-. -paramecium- An EKtcnSlbIe Object-Based Kernel." Ptoc Hot Tuples in Operating S\stem\ V (EEE pp 86 89 1995. VAN STKEN, M., HAUCK. FJ., BALLINTIJN, G,, and TANENBAUM. A.S.: 'Algorithmic Design of the Globe Wide-Area Locahnn Service," Computer Journal, vnl 41. pp 207-310, 1998a. VAN STEEN, M^ HALLK. Vj., HOMBURG, P., and T\NENBAUM, A.S.; Ucaiing Objects in Wide-Area Systems," IEEE Communications Mat>a?ine vol 36 pp 104- 109, Ian 1998b VAN STEEN, M., HOMBURG, P., and TANENBAUM, A.S.: "Globe: A Wide-Area DiMn- buled System," IEEE Concurred, vol 7. pp. 70-78. Ian-March 1999a

932 READING LIST AND BIBLIOGRAPHY CHAP. 13 VAN STEEN, M, TANENBAUM, A.S., KUZ, I, and SIPS, Hj,; "A Scalable Middleware Solution for Advanced Wide-Area Web Services," Distributed Systems Engineering, voi 7, pp. 34-42, 1999b VASWAN1, R., and ZAHORJAN, J.: "The Implications of Cache Affinity on Processor Scheduling for Mult [programmed Shared-Memory Multiprocessors," Proc. I3th Symp, on Operating Systems Principles, ACM, pp 26-40, 1991. VENKATASUBRAMANIAN, N., and RaMANATHAN, S: "Load Management in Distri- Distributed Video Servers," Proc I7th Int'l Conf. on Distrib. Computing Systems, IEEE, pp. 528-535, 1997. V1NOSK1, S.: "CORBA: Integrating Diverse Applications within Distributed Heterogene-Heterogeneous Environments," IEEE Communications Magazine, vol. 35, pp. 46-56, Feb 1997. VISCAROLA, P.C and MASON, W.A.: Windows NT Device Driver Development, Indi- Indianapolis, IN: Macmillan Technical Publishing, 1999 VOGELS, W-: "File System Usage in Windows NT 4.0," Proc. 17th Symp. on Operating Systems Principles, ACM, pp. 93-109, 1999 VON EICKEN, T., BASU, A.. BUCH, V., and VOGELS, W.: "U-Net a User-Level Network Interface for Parallel and Distributed Computing," Proc, I5th Symp. on Operating Systems Principles, ACM, pp. 40-53, 1995 VON EICKEN, T., CULLER, D,, GOLDSTEIN, S.C., SCHAUSER, K.E.; "Active Messages A Mechanism for integrated Communication and Computation," Proc

19th int'l Symp on Computer Architecture, ACM, pp. 256-266, 1992. WAHBE, R,, LUCCO, S,. ANDERSON, T.. and GRAHAM, S.; "Efficient Software-Based Fault Isolation," Proc. 14th Symp. on Operating Systems Principles ACM, pp. 203- 216, 1993. WALDO, J,: -The Jini Architecture for Network-Centric Computing," Commun. of the ACM, vol. 42, pp. 76-82. July 1999. WALDO. J,: "Alive and Well: Iini Technology Today " Computer, vol. 33 pp. 107-109, June 2000. "Lottery Scheduling; Flexible Proportional-Share Resource Management," Proc First Symp. an Operating System Design andImplemensauon, USENIX, pp. I-I2, 1994. WALKER, w., and CRAGON. H.G.; "Interrupt Processing in Concurrent Processors," Computer, vol. 28. pp. 36-46, June 1995. WAN, G., and LIN, E.; "A Dynamic Paging Scheme for Wireless Communication Sys- Systems," Proc, Third Int'l Conf. on Mobile Computing and Networking ACM/IEEE, pp 195-203, 1997 WANG, c, GOEBEL, v., and PLAGEMANN, T.: "Techniques to Increase Disk Access Locality in the Minorca Multimedia File System," Proc. Seventh Int'l Conf. on Mul- Multimedia, ACM, pp. 147-150,1999.

SEC. 13.2 ALPHABETICAL BIBLIOGRAPHY 933 WANG, R.Y., ANDERSON, T.E., and PATTERSON, D.A.: "Virtual Log Based File Sys- Systems for a Programmable Disk," Proc. Third Symp. on Operating Systems Design and Implementation, USENIX, pp. 29_43, 1999. WEISER, M., WELCH, B., DEMERS, A., and SHENKER, S.: "Scheduling for Reduced CPU Energy," Proc- First Symp on Operating System Design and Implementation, USENIX, pp. 13-23, 1994. W1LKES, J, GOLDlNG, R^ STAEL1N, C, and SULLIVAN, T.; 'The HP AutoRAID Hierarchical Storage Sysiem," ACM Trans, on Computer Systems, vol. 14, pp. 108- 136, Feb. 1996. WIRTH, N.i "A Plea for Lean Software,- Computer, vol. 28, pp. 64-68, Feb. 1995. WOLMAN, A., VOELKER, M, SHARMA, N., CARDWELL, N., KARLIN, A., and LEVY, H.M.; "On die Scale and Performance of Cooperattve Web Proxy Caching," Proc. DihSymp. on Operating Systems Principles, ACM, pp. 16-31, 1999. WONG, C.K.: Algorithmic Studies in Mass Storage Systems. New York: Computer Science Press. 1983. WONG, P.C., and LEe, Y.B.: "Redundant Amiy of Inexpensive Servers (RAIS)." Proc. Int'l Conf. on Commun,, IEEE, pp. 787-792,)997, WORTHINGTON, BJ... GANGER, G.R., and PATT, Y.N.: "Scheduling Algonlhms for Modem Disk Drives," Proc. 1994 Conf on Measurement and Modeling of Computer Systems, pp. 241 -251, 1994. WU, M., and SHU, w.: "Scheduling for Large-Scale Parallel Video

Servers," Proc Sixth Symp. on Frontiers of Massively Parallel Computation, IEEE, pp. 126-133, (996. WULF, W.A., COHEN, &S« CORW1N, W-M., JONES, A.K., LEvIN, R., PIERSON, C, and POLLACK, fj.: "HYDRA: The Kernel of a Multiprocessor Operaiing System," Commun. of the ACM, vol. \7, pp. 337-345, June 1974. YOUNG, M, TEVANIAN, A,, Jr., RASHID, R., GOLUB, D., EPP1NGER, J., CHEW, J., BOLOSKY, W., BLACK, D, and BARON, R.: "The Duality of Memory and Commun- Communication in the Implementation of a Multiprocessor Operating System," Proc. Ijth Symp. on Operating Systems Principles, ACM, pp. 63-76. 1987. ZACHARY, G,P.: Showstopper, New York" Maxwell MacMiHan, 1994. ZAHORJAN, J,, LAZOWSKA, E.D,, and EAGER, D.L.: "The Effect of Scheduling Discip- Discipline on Spin Overhead in Shared Memory Parallel Systems." IEEE Trans, on Parallel andDistr. Systems, vo] 2, pp. 180-198, April l99i. ZEKAUSKAS, M J., SAWDON, W.A., and BERSHAD, B.N.: "Software Write Detection for a Distributed Shared Memory," Proc. First Symp on Operating System Design and implementation, USENIX, pp. 87-100, 1994 ZOBEL, D.: "The Deadlock Problem: A Classifying Bibliography," Operating Systems Review, vol. 17, pp. 6-16, Oc(. 1983.

 ZUBERI, K.M., PILLA1, P., and SHIN, K.G.; "EMERALDS: A Small-Memory Real-T/me Microkernel," Proc t7th Symp. on Operating S\stems Principles. ACM, pp. 277-299, 1999 ZWICKV, E.D.: "Torture-Tesl/ng Backup jnil Archive Programs. Things You Ought to Know but Probably Would Ralher Not," Proc. Fifth Crinf. On Large Installation Svs- tems Admin.. VSENIX, pp 181-190, l99l.

INDEX

Absolute path, 395, 733 Access eontrol element, 846-847 Access control list, 647-650, 846-847 Access loken, 846 ACE (see Access Control Element) Acknowledged datagram service, 555 Acknowledgement 121, 554 ACL (see Access Control List) ACPI, 370 Active message, 537 Adapter, 271 ADC (see Analog digital converter) Address space, 34 Address windowing extensions, 8l6 Admission control algorithm, 458 Admission scheduler, 140 ADSL (see Asymmetric Dig/tal Subscriber Lir Adversary, 585 Affinity scheduling, 523 AFS (see Andrew File System) Agent, 637 Aging, 146,220-222 Alarm signal, 36 Analog digital converter, 46O Andrew file system, 564-565 Antivirus techniques, 628-635 Behavioral checkers, 633 Integrity

checkers. 632 Virus scanners, 629-632 APC (see Asynchronous Procedure Call) Aperiodic process, 149 API (see Application Program Interface) Applet, 637 Application program interface, 54 Architectural coherence, 861 Architecture, 3 Associative memory, 211 Asymmetric digital subscriber 1/ne, 454, 727 Asynchronous call, 535 Asynchronous, procedure call, 783 Atomic action, IK) i) Attribute, file, 386-387 Audio encoding. 459-461 Authentication, l2l challenge-response, 600 user 591-606 Automounting 749 AV disk, 324

B Backing store, 246 Backup file system, 41E-421 Bad block, 322-324,418 Bad-news diode, 892 Balance ser manager, 820 Banker's algorithm, 178-180 muftipte resources, 179-180 single resource, 178-179 Bank switching 816 Barrier, 123-124 Base record, 834 Base register, 26, 196 Basic inpul outpui system, 33, 190 Batch system, 7 Battery management, 369 Battery-powered computers, 896-897 Behavioral checker, 633 Belady's anomaly, 229 Bell-La Padula model, 657-658 Berkeley fast file system, 744-745 Berkeley UNIX, 675-676 Best fit algoridim, 201 Biba model, 658-659 Binary exponential backoff, 552 Binary semaphore, m Binding time, 874 BIOS (see Basic Input Outpui System) Bitmap, 199-200,352-354 Bilmap graphics, 344 Block cache, 424-426,492-494 Block device, 270, 290 Blocking call, 535 Blocking network, 5l0 Blocking versus nonlocking call, 535-537 Block size, 298, 411-412 Block special file, 40, 383, 724 Bool block, 399 Boot sector, 810 Booting UNIX, 708-710 Booting Windows 2000, 810-811 Boiiom-up implementation, 876-877 Bounded-buffer problem, 108 Bridge, 552 Brute force solution 881 BSD, 13 BSS segment, 711 Buddy algorithm, 721 Buffer cache, 424-426,728 Buffering, 294-296 double, 295-296 Buffer overflow attack, 610-612 Bursi mode, 278 Bus, 31-34 Bus-based multiprocessor, 506-507 Busy waiting, 30, 104,285 Byte code, JVM, 643 Cache, 87 Cache-Coherent NUMA, 511 Cache hit, 24 Cache line, 23 Cache manager, Windows 2000, 785 Caching, 887-888 file, 424-426 multimedia, 492-494 Windows 2000, 849-851 Call gate, 26i-262 Canonical mode. 337-340 Capability, 650-653 Capability list, 650 CC-NUMA, 511 CD-recordabie, 310-312 CD-Rewritable 312-313 CD-ROM, 33-34, 306-310 file system, 430-435 multisession, 3l2 session, 312 VTOC, 3U-312 XA, 311 Central processing un/t, 21-23 CERT (see Computer Emergency Response T Certificate, 591 Challenge-response, 600 Character device, 270, 291 Character special file, 40, 383, 724 Checkerboarding, 253 Checkpointing, 174 Chief programmer learn, 891 Child process, 35,691 Chrominance signal, 462 Ciphertext, 587 Circuii switching, 529 Circular wail condition, 164-

165,182-183

Classical 1 PC problems, 124-132 dining philosophers, 125-128 readers and writers, 128-129 sleeping barber, 129-132 Cleaner, 430 Cleaning pofícy, 24i Client process, 62 Clieni-server model, 6(-63 Clienr-server system, 869-870 Client stub, 538 C-list 650, 730 Clock, 327-333 square-wave mode, 328 one-shot mode, 328 Clock hardware, 328-329 Clock paging algorithm, 218-219 Clock software, 329-332 Clock tick, 328 Cluster computer, 526 Cluster of workstations, 526 Cluster size, 440 CMOS memory, 25-26 CMS (see Conversational Monitor System) Code page, 347 Code review, 6lO Code signing, 64l-642 Color palette, 346 Command interpreter, 35 Compaction, memory, 197 Compatible Time Sharing System, 11 Composite video. 462 Computer first generation, 6-7 fourth generation, 14-16 second generation, 7-9 third generation, 9-13 Computer emergency response team, 637 Computer hardware, 20-34 Condition variable, 116,698 Configuration manager, Windows 2000, 785 Confinement problem, 661 -663 Constant data length, 485 Constaut time length, 485 Content switch, 28, 142 Control object, Windows 2000, 783 Conversational monitor system, 59-60 Cooked character stream, 731 Cooked mode, 336 Copy on write, 241,701,815 CORBA 565-567 ,x 937 Core image, 35 Core map, 7,6 Co-scheduling. 525 Counlermeasures 606 Coven channel, 661-665 COW (see Cluster Of Workstations) CP/M. 14,435-438 CPU (see Centra! Processing Unit) CPU scheduler (see Scheduling algondimj Cracker, 592 593-596 Crash, 290,417,678,814 Critical region, 102-103 Critical section, 102 Crossbar sWiich, 507 Crosspomt, 508 Cryptography, 587-591 CTSS (see Compatible Time Sharing System Cube, 527 Current directory, 396 Cycle slealing, 278 Cylinder, 24 Cylinder group, 745 Cylinder skew, 316 D DACL (see Discrelionary ACL) Daemon, 74, 299,690 primer, 101 DAG (see Directed Acyclic Graph) Data confidentiality, 584 Data encryption standard, 843 Datagram service, 555 Data integrity, 584 Data segment, 50, 711 Deadlock, 36-37, 159-185 conditions, 164 definition, 163 Deadlock avoidance, 175-180 multiple resource;,, 179-180 single resource, 178-179 Deadlock detection, 168-175 Deadlock modeling, 164-167 Deadlock prevention, 180-IS5 attacking circular wait, 182-183 attacking hold and wait, 181 attacking mutual exclusion, 180-181 attacking no preemption, 182

938 If Deadlock recovery, 173-175 using killing processes, 174-175 using preemption, 173-174 using rollback. 174 Decibel, 459 Dedicated I/O device,

297 Deferred procedure call, 783 Degree of multiprogramming, 141 Demand paging, 222 Denial of service attack, 584 DES (see Data Encryption Standard) Design, operating system, 855-898 Design principles, 859-861 Device context, 351 Device controller, 271-272 Device driver, 29,289-292 uniform interfacing, 293 Windows 2000, 827-%30 Device independence, 283 Device independent bitmap, 354 Device object, 82S Device register, 2 Diameter, network, 527 DIB (see Device independent hitmapj Digiial signature, 590-591 Digital versatile disk,3l3,454 Digital video, 467 Digital video disk, 313 Digram, 588 Dining philosophers problem, 125-128 Directed acyclic graph, 408 Direct memory access, 31, 276-279,287 Directory-based multiprocessor, 511 Directory 38.383 393-399 CP/M, 437-438 current, 396 distributed system, 560-561 hierarchical, 395 implementation 405-408 ISO 9660, 432-433 MS-DOS, 440 root, 393 single level, 393-394 UNIX version 7, 445-446 Windows 98,443^*45 Windows 2000, 839-840 working, 3% Directory operation, 398-399 DirectX, 781 Dirty hit, 210 Disabling interrupts. 103-104 Discretionary access control, 657 Discretionary ACL, 846 Disk, 300-327 CD-ROM, 306-310 CD-recordable 310-312 CD-rewritable, 312-313 DVD 313-315 IDE, 300 magnetic, 300-302 Disk arm scheduling, 318-322 elevator algorithm. 319 shortest seek first, 319 Disk backup, 416-421 Disk farm 490 Disk formatting, 315-318 Disk hardware, 300 Disk interleaving, 317-318 Disk quota, 415-416 Disk scheduling, 318-322 dynamic 496-498 multimedia, 494-498 static, 495-496 Disk space management, 410-411 Dispatcher, 87 Dispatcher object, 783 Distance string, 232 Distributed operating system. 16 Distributed shared memory, 242, 540-544, 568 Distributed system, 505, 549-577 DLL {see Dynamic Link Library) DMA (see Direct Memory Access) DNS (see Domain Name System) Domain name system, 557 Domain, protection, 645 DOS (see MS-DOS) Dotted decimal notation, 594 Double buffering, 295 Double indirect block 446 744 Double torus, 527 DPC (see Deferred Procedure Call) Driver interface, 370 Driver-Kernel Interface 728 Driver object, 824 DSM (see Distributed Shared Memory) D-space, 239 DVD (see Digital Versatile Disk) DV (see see Digital Video) Dynamic disk, 825 Dynamic link library, 794

E Earliest deadline first, 473-476 Early binding. 874 Echoing 337-338 HDF dee Earliest Deadline First) Effete UID, 755 EFS (u-e Encrypting File System) Eiectncally Erasable ROM, 25 blevalor algorithm, 320 Embedded systems, 897 Encryption, 587-589. 842-844 Encrypting file system, 842 End-to-end argumem. 868-869 Environment Subsystem, 792 Error-correcting code, 272

Error handling, 322-324 Error reporting, 297 Escape character, 340 Escape sequence, 341 Ediernet, 551-552 Event-dnven paradigm, 862 Event, Windows 2000, 801 -802 Exclusive lock, 736 Executive, Windows 2tXK) 784-787 Exokernel, 61, 868-869 Extezided machine, 4 Extensible system, 870 External fragmentation, 253 Fair shar scheduling, |4i FAT dee File Allocation Table) Fiber, Windows 2000, 798 Field, 461 FIFO page replacement algorithm, 217 File, 38-41 block size, 411-412 contiguous. 400-402 shared, 408 special, 40, 383 724 File access. 385-386 File allocation using r'AT,4U3-404 using linked list. 402 hie allocation table, 403-404 FAT-lfi tile system, 830 FAT -32 file system, 830 File caching, 494 File compression, 841-842 File descriptor, 39, 389. 737 File encryption. 842-844 File extension, 381 File handle, 749 File implementation, 400-405 File management 413-414 File name lookup. 839-841 File naming, 380-382 File operation, 387-188 File placement multimedia, 481-492 near video on demand, 486-487 File read-ahead, 427 File sharing, 562-564 File structure, 182-183 File system. 379-449 backup. 416-421 CD-ROM, 430-435 CP/M, 415-438 disk quotas. 415-416 distributed. 559-564 examples. 430-448 FAT-16 830 FAT-32. 830 implementation 399-430 ISO 9660, 430-435 layout, 399-400 Linux, 746-747 log structured 428-430 MD-DOS, 438-442 multimedia, 475-481 performance, 424-428 reliability,416-424 UNIX, 732-753 UNIX system calls, 736-740 UNIX Version 7,445-448 Win32 API calls,'831-832 Windows 2000, 830-844 Windows 98, 442-445 File type, 383-385

940 IND Filter, 685 Filter driver, 830 Finger daemon, 636 Finger length identification, 604 Fingerprint identification, 605 Finite-state machine. 89 "irst-come first-served scheduling. 139 -irsi fit algorithm, 200 ;irsi-m First-out page replacement. 217 ;ixed panitions, 19|-I92 i|ashRAM.25 rly-by mode, 278 FMS (see FORTRAN Monitor System) Folder, 393 Font, 354-355 Formal Models of Secure Systems, 655-647 FORTRAN monitor system, 8-9 Fragmentauon external, 253 internal. 237 Frame, 461 Free disk block management, 413-414 Gang scheduling, 524-526 GDI (see Graphics Device Interface) GDT (see Global Descriptor TaNeJ Generic rights, 652 Generic security attacks, 613 GID (see Group IDj Global descriptor table 257 Globe, 567-571 GNU public license, 679 Goal file, 629 GPL (see GNU Public License) Graphical user interface, 15, 342-355 input software, 347 output software, 347-355 Graphics adapter, 344 Graphics device interface, 351 786 Green book, 309 Grid,527 Group 648 Group ID, 36,753 Guaranteed

scheduling, 146-147 GUI (see Graphical User interface) H Hacker, 592. (see also cracker) HAL (see Hardware Abatracuon Layer) Handle, 77.773 Hard link, 398 Hard real time system, 148 Hardware abstraction layer, 779-782 Header file, 687 Head skew, 316 Hiding die hardware, 877-879 High sierra, 309 Him. 888 Hive, registry. 777 Host. 553 Hungarian notation 350 Hypercubc, 528 Hyperlink, 558 I IBM 1401,7-10 IBM 7084, 8-10 IBM PC. 14 IBM System/360 9-10 IDE Disk, 28 1DL (see Interface Definition Language) IEEE 1394.32 IEEE standard 1003.1,676 HOP (see Internet interOrb Protocol) Immediate fUe, 837 Impersonation, 846 Imprecise interrupt, 282 lacremental dump, 417 Indirect block, 446, 744 Indirection, 879-880 1-node. 51, 404-405,740 1-node table, 741 Input software GUI, 347 terminal. 336-340 Instruction backup, 244-245 Instruction set architecture, 2 Integrated drive electronics, 28 Integrity checker, 632 Interface definition language, 566 Interlaced video, 462 Interleaving, disk, 317-318

Internal fragmentation, 237 Internet, 552-553 Internet InterOrb Protocol, 566 Internet protocol, 556,726 Internet worm, 635-637 Interpretation, 640-641 Interprocess communication, 35, 100-124 Interrupt, 30, 279-282 imprecise, 282 precise, 281 ■Intemipt-driven I/O, 286 Interrupt handler, 287 Interrupt vector, 30,79, 280 Intrinsics 357 Intruder, 585 1/0,38,269-372 DMA, 287, 276-279 memory-mapped, 272-276 I/O device, 28-31,270-271 I/O hardware, 269-282 clock, 328-329 disk, 300-315 display, 344-346 keyboard. 335 mouse, 343 I/O manager, Windows 2000,784 I/O port, 273 I/O request packet, 827 I/O software, 282-287 buffering, 283-284, 294-296 device independent, 283, 292-298 ermr handling, 283 ermr reporting, 297 input. 336-340. 347 layers. 287 output, 341-342. 347-355 synchronous, 283 naming, 283 UNIX. 723-732 Windows 2000, 824-830 IP address, 556 IPC (see Interprocess communication) IP (see Internet Protocol) IRP (see I/O Request Packet) IS A bus, 31 ISA (see Instruction Set Architecture) ISO 9660 file system, 309-310, 430-435 Joliet extensions, 435 Rock Ridge extensions, 434-435 I-space, 239 virti ;Iopment kit, 643 :hine, 61,640-641 Kit) ce, 576 JDK (see Java Developm Jiffy, 707 Juii, 575-577 discovery protocol, 575 lease, 575 lookup service, 575 Jitter, 457 Job, 7 Windows 2000, 796 Joint photographic experts group, 464 Joliet extensions, 435 JPEG standard, 464-467 JVM (see Java Virtual Machine) K Kernel UNIX, 687-690 Wiadows 2000 782-784 Kernel mode, 2 Kernel thread, 870 Key cryptographic, 587 file, 383 registry, 775 Keymap, 347 Land, 307 LAN (see Local Area Network) Large address space,

894-895 Late binding, 874 Layered system, 57-59, 867 LDT (see Local Descriptor Table) Least recently used page replacemei Legacy device, 33 LFS (see Log-Structured File systen Lightweight process, 81 Limit register, 26, 196

942 INH I.mdi. 572-571 template. 573 tuple spate. 572 Lineal address. 219-260 Line discipline, 7?O Link 408. 734 i inked iisi. 200-202 Linux. 13. 678-680 fife system, 740-747 mcoiory management. 719-72'! scheduling algorithm. 7116-708 threads. 702-704 Loadable module. 730 Load balancing algorithm. *>45 bidding. 548 graph-theoretic. 545-546 receiver-initiated. 547-548 sender-initiated. 546-547 Local Area Network, 551 Local descriptor tahle 257 Locality, 888-889 Locality oi reference, 222 Local procedure call. Window 2000. 785 Location independence, 562 Location service. 571 Location transparency, 561 Lock, file, 735 Lock variable. 104 Logical block addressing. 302 Logical dump, 419 Logic bomb, 609 Login spoofing, 608-609 Log-structured file system. 428-430 Loosely-coupled system. 505 Lottery scheduling. 147 LRU page replacement algorithm, 218-220 Luminance signal, 462 M Machine language. 2 Macroblock, 468 Magic character, shell. 684 Magic number, 384 Mailbox. 122 Mainframe. 7 Major device, 293.724 Mandatory access controls. 657 Mapped page writer- 822 Master booi record. 317. jW9 M^ter file table, 834-839 MBR(w« Master Boot Record) MD5. 590 Mechanism versus policy, 62, [49-150. 247-249 870-R7I Memory. 23-28 Memory companion. 197 Memory hierarchy. 189 Memory management, 37. IK9-263 bitmaps 199-300 design issues. 234-242 implementation, 242-249 linked lists, 200-202 Linux, 719-723 system calls, 714-715 swapping, 196-202 UNIX. 710-723 virtual memory 202-214 Windows 20tK). 811-823 Memory management unit. 27, 203 Memory manager 1S9 Windnws 2000. 785 Memory-mapped file, 391'393.713 MciHory-mapped I/O, 272-276 Memory-.chedulcr, 141 Mesh. 527 Message digest, 590 Me3sage passing. 119-123 design issues, 121 producer-tons inner. 121-12! Mess age-pas sing interface, 123 Metafile *53 Method, 349, 565 Metric units. 66-67 MFT (see Master File Table) Microcomputer. 14 Microkernel, 62 Microprogram. 1 Middleware. 550 coordination-based, 572-577 document-based, 5.'i8-559 file-system-hdscd. 659-564 shared-object-based 565-571 MlNlX, 13.677-678 Minor device, 53. 294, 724 Missing disk block. 422

MMU (see Memory Management Unii) Mobile code. 637-642 Modeling paging systems, 228-234 Modified page writer, 822 Monitor, 115-119

Monoalphabelic substitution, 588 Monolithic system, 56-57 Monoprogramming, 190-191 Motif, 16, 357 Motion picture experts group, 467 Mounted file system, 52 MPEG standard, 467-469 MPI {see Message-Passing Interface) MS-DOS, 14,763-764 emulation on Windows 2000, 809-810 file system, 438-442 Multicomputer, 526-548 communication software, 531-540 hardware, 527-531 interconnection, 527-531 scheduling, 544-545 software, 531-548 MULTICS, 12, 672, 254-257 Multilevel security, 657-659 Muliimedia system, 453-499, 896 audio encoding. 459-461 disk scheduling, 494-498 file, 458-463 file placement, 481-492 file system paradigms, 475-481 process scheduling, 469-475 server, 475 video compression, 463-469 video encoding, 461-463 Multiple queue scheduling I44-U6 Multiprocessor, 71, 506-526 bus-based, 506-507 CC-NUMA, 511 crossbar-based, 507-508 direciory-based, 511 hardware. 506-513 mullistage network, 509-510 NC-NUMA.5I1 operating system, 513-516 scheduling, 521-526 synchronization. 516-521 Multiprogramming 10 72 degree, 193 fixed partitions, 191-192 modeling, I92-i94 Multistage swnching networks, 509 Multithreaded code, 81, 97-100 Mutation engine, 631 Mutex, 113-114 698 Mutual exclusion, 102, 103-108 busy waiting, 103 disabling interrupts, 103-104 lock variables 104 Peterson's algorithm, 105-106 strict alternation, 104-105 TSL instruction. 107-108 Mythical man month, 890-891 N Naming, 872-874 Naming transparency, 561-562 Narrow striping. 492 NC-NUMA, 511 Near video on demand 478^81 file placement, 486-487 Network file syslem, 747-753 architecture, 747-748 implementation, 750-753 protocols, 749-750 Network hardware, 551-553 Networking, 895 UNIX, 725-726 Network iniertace, 529 Network operating system. 16 Network" protocol, 555-558 Network service, 553-555 acknowledged datagram, 555 connectionless, 554 connection-oriented. 554 datagram, 555 request-reply service. 555 Network terminal, 355-363 SLIM, 360-363 Next fit algorithm, 201 NFS (see Network File System) NFU page replacement algorithm, 220 Nonlocking system call, 535 Nonblocking network, 508 Noncanonical mode, 337 Nonpreemptive scheduling, 135 Nonresident attribute, 836 Nonresource deadlocks, 184

944 INDEX Nominiform memory access mach Nonvolatile RAM. 327 Not frequently used page replacement, 220 Not recently used page replacement, 216-217 NRU page replacement algorithm, 216-217 NT Hie system, 830-844) NTFS (rpe NT File System) NUMA multiprocessor, 511-513 s, 511-513 OS/360. 10-11 OSF (see Open Software Foundation) Ostrich algonthm, 167-

168 Output software GUt 347-355 terminal, 341 Windows, 347 Overlapped seek, 300 Overlap 202 o Object. 565 secumy, 647 Windows 2000,787-792 Object adapter, 566 Object manager, Windows 2000, 784 Object name space, Windows 2000,791-792 Object request broker, 566 Omega neiwork, 509 One-irme password, 599-600 One-way function, 589 Oniogeny recapitulates phylogeny, 16-18 Open file description, 743 Open software foundation, 677 Operating system, 1 batch, 7 concepts, 34-44 design, 855-898 embedded. 20 extended machine, 3-4 history, 6-18 implementation, 867-882 multimedia, 453-499 multiprocessor, 513-516 performance, 882-889 resource manager. 5-6 structure, 56-05, 867-871 trends, 894-897 types, 18-20 Optical disk, 306-315 Optimal page replacement algorithm, 215-216 Optimization, common case, 889 Orange book. 311, 659-661 ORB (see Object Request Broker) Organ-pipe algorithm, 488 Orthogonality, 871-872 Pand V primitives III Packet, 528 Packet sniffer, 596 Page, 203-210 pinned, 246 Page daemon, 716 Page directory, 259-260 Page fault, 205 Jage fault frequency replacei "ige fault handling, 243-244 Windows 2000, 818-819 ige fault rate 233-234 ige frame, 203 ige replacement algoridim, 214-2 aging, 220-222 clock. 218-219 FIFO. 217 global, 234 Linux, 719-723 local 234 LRU, 218-220 NFU, 220 NRU. 216-217 optimal, 215-216 PFF, 235-236 second chance, 217-218 summary, 227-228 two-handed clock 718 UNlX,7i7-719 Windows 2000, 819-823 working set, 222-225 WSclock, 225-227 Page size, 237-238 Page table, 205-210 invened, 213-214 multilevel, 207-209 235-236

945 Page table entry, 209-210 Page taNe structure, 209-2H Paging, 202 demand, 222 design issues. 231-242 global, 234 implementation issues 242-249 load control, 236-237 local, 234 ' modeling, 228-23 UNIX. 716-717 Windows 2000, 819-823 Paging daemon, 241 Paradigm, data, 863-864 execution, 862-863 user interface, 861-862 Parallel and distributed systems, 896 Parent process, 691 Partition, disk, 53, 317 Password, 592-600 one-time, 599-600 Path name, 38, 395-398 absolute, 395 relative, 396 PCt bus, 31 PDA (see Personal Digital Assistant) PDP-11, 13 Penetration team, 613 Pentium, segmentation, 257-262 Perfect shuffle network, 509 Performance, operating system, 882-889 Periodic process, 149 Personal digital assistant, 20 Peterson's algorithm, 105-106 PFF page replacement algorithm, 235-236 Physical address, 27 Physical dump, 418 Physical memory management, 821 P1D (see Process ID) Pinned page, 246 Pipe, 40-41, 692 Pipeline, 22, 685 Pipe symbol 685 Pit, 307 Pixel, 344, 463 Plaintext, 587 Play point, 480 Plug

and play, 33 Plug-and-play manager, 785 Policy versus mechamsm, 62. 149-150. 247.249, 870-871 Polling. 285 Pop-up thread, 537 Portable C compiler, 674 Positron independent code, 816 POS1X 13,46,676 diread calls. 697-698 Power management, 363-371 degraded operation. 370-371 hardware issues, 364-365 operating system issues, 365-370 Windows 2000. 785 Preamble, disk, 272 Precise interrupt, 281 Preemptive scheduling, 135 Prepaging, 222 Present/absent bit, 205 Primary volume descriptor, 431 Principal, security, 647 Principle of least privilege, 653 Printer daemon, 101 Priority inversion, 108,808 Priority scheduling, 143-144 Privacy, 585 Process, 34-36,71-153 aperiodic. 149 child, 691 classical 1PC problems, 124-132 interprocess communication, 100-124 parent, 691 periodic, 149 UNIX. 690-710 Windows 2000, 796-811 Process control block, 79 Process creation, 73-75 Process group, 692 Process hierarchy, 76-77 Process ID, 48, 691 Process implementation, 79-8A Process management, UNIX, 692-698 Process manager. Windows, 2000. 784 Process model, 72-73 Process scheduling, 132-152 multimedia, 469-475 Process state, 77-79 Process switch, 142 Process table 35 79-80 UNIX, 699 Windows 2000, 796-8II

946 Process terminal ion, 75-76 Processor allocation algorithm, 545 Processor sharing, 155 Producer-consumer problem, 108-110 message passing, 121-123 monitors, 117-118 semaphores, 11-113 Program counter, 21 Programmed I/O, 284-286 Program status word, 21 Progressive video, 462 Project management, 889-894 Prompt, 42, 683 Proportionality, 138 Protection, memory, 194-196 Protection command, 655-656 Protection domain, 645-647 Protection matrix, 645-647 Protection mechanism, 584, 645-653 Protection ring, 262 Protocol, 356, 602 Protocol stack, 556 Pseudoparallelism,71 PSW (tee Program Status Word) Pubfjc-key Cryptography. 588-589 Publish/Subscribe, 574-575 Pull server, 475 Puise code modulation, 460 Push server, 475 Quality of service, 457, 554 Quantization noise, 460 Quantum, 142 Quick fit algorithm 201 Quota, disk. 4M-416 Race condition, 100-102 RAID, 302-306 striping, 303 RAM (see Random Access Memory) Random access file, 385 Random access memory, 24 Raster graphics, 344 Rate monotomc scheduling, 472-473 Raw mode, 336 Read ahead, file 427,752 Readers and writers problem, 128-129 Read-only memory, 25 Real time, 329 Real-time operating system, 20 Real-time Scheduling, 470-475 Receive primitive, 534-535 Recycle bin, 417 Recycling of concepts, 43-44 Red book, 306 Redirection, I/O, 42 692 Reentrancy, 880-881 Reentrant code. 292 Reference

monitor, 640, 645. 655 Reference unng, 230-231 Registry, Windows. 774-777 Regular file, 383 Relative path, 396, 733 Reliability, file system, 416-424 Relocation, 194-196 Remote procedure call, 537-540 Rendezvous, 123 Reparse point, 841 Request-reply service, 555 Resource, 160-163,358 nonpreernptable, 161 preemptable. 160 Resource acquisition, 161-163 Resource trajectories, 175-176 Resource vector, 171 Response time, 138 Reusability 880 Right. 645 RMS (.we Rate Monotomc Scheduling) R-node. 751 Rock Ridge extensions. 434^135 Rote, 648 ROM (see Read-Only Memory) Root. 754 Root directory. 38 Root file system, 40 Root key, registry 775 Round. 495 Round robin scheduling 142-143 Router, 553 RPC (see Remote Procedure Call) RSA algorithm, 589 Rwx bits, 41

SACl. (set System ACU Sate male. 176-178 Salt, 1i97 Suntiboxing, 638-640 SiMn-EDF algorithm. 4 Schedulablc system. 149 Scheduler, 132 Scheduler activations, 94 Scheduling- process, 132-152 Scheduling algorithm, 1.12-152 affinity. 523 batch systems. 13H-H9 categories, 135-136 earliest deadline first. 473-476 fair share, 148 first-come. first-served, 139 goals, 136-138 guaranteed. 146-147 interactive, 142-148 Linux. 706-708 lottery, 147 multicomputer, 544-545 multiprocessor, 521-526 multimedia, 469-475 multiple queues, 144-146 multiprocessor, 521-526 priority, 143-144 rate monotomc, 472-473 real-umc, 148449 real-time, 470-475 round rohrn, 142-143 shortest job first. 139-140 shortest process next. 146 shortest remaining time next 140 thread, E0-151 three-level scheduling 140-142 two-level, 523 UNIX, 704-706 Windows 2000, 804-309 Scheduling mechanism, 149 Scheduling polity. 149 Script kiddy, 596 SCSi bus. 32 Second cliance page replacement, 217-218 Second system effect. 89? Secret-key cryptography, 588 Secure hash algonthm, 590 Security, 41,583-666 Security attacks, 606-644 huffcr overflow. 610-612 logic bomb. 609 login spoofing, 608-609 trapdoor, 610 Trojan horse. 607-608 I'NIX 7S3-757 Win32 API calls,. 847-848 Windows 2000. 844-849 Security flaws, 614-616 OS/360.615 TENEX.614 UNIX, 614 Sccuril> by obscurity. 587 Security descriptor, 846 Security design principles. 616-617 Security environment. $84-^86 Security ID. 845-847 Security manager, Windows 2000, 785 Segment, 250-251 Sn, 249-262 M MULTICS, 254-257 Pentium, 257-262 Selector. Pentium, 258 Semantics of file sharing, 562 Semaphore, 110-113 binary. 111 Send primitive. 534-535 Sequencer, 56') Sequential access lite, 3K5 Sequential Lonbistency, 543-544. 562 Sequential process, 72 Serial line. t34 Server stub, 538 Session manager,

SI I Set-ttip box. 455 SRTUlD bit, 755 SHA (sec Secure Hash Algorithm) Shared library. 251 Shared lock. 736 Shared-memory multiprocessor. 5t>6-526 Shared pages. 239-241 Shared text segmem. 712 Shell, 15,41-42,683-686,693-694 Shell script. 686 Shortest job fust scheduling, 139-140 Shortest process next scheduling, 146

948 JNDE Shortest remaining time next, 140 Shortest seek first. 319 SID (see Security ID) S.gna). 692 Signature block, 590 Silver bullet lack of, 894 Single indirect block, 446, 744 Skeleton. 566 SLED, 303 Sleep primitive, 108 Sleeping barber problem, 129-132 SLIM network terminal, 360-363 Smart card. 20, 601 Smart scheduling, 522 SMP (see Symmetric Multiprocessor) Socket, 725-726 Soft real time system, 148 Soft timer, 332-333 Space sharing, 523 Space-time trade-off, 884-886 Special file, 40 383,724 Spin lock, 105,518-519 Spinning versus switching, 519-521 Spooler directory, 101 Spooling. 11,298 Spooling directory. 299 Stable storage, 324-327 Stack algorithm, 229-232 Stack pointer, 21 Stack segment, 50 Standard error, 684 Standard input, 684 Standard ourput, 684 Starvation, 126,184-185 Stateless file system, 750 Static versus dynamic structures, 875-876 Steganography, 664 Store-and-forward packet switching 528-529 Stored value card, 601 Stream, 730-732 Striping, disk, 492 Stub, 538 Subject, security, 647 Subkeys, registry 775 Superblock, 399 740 Superscalar CPU, 22 Superuser, 36,754 Supervisor mode, 2 SVID (see System V Interface Definition) Swapper, 715 Swapper thread, 821 Swapping, 196-202 UNIX, 715-716 Symbolic Imk, 409-410 Symmetric-key cryptography, 588 Symmetric multiprocessor, 515-516 Synchronization, 113 multiprocessor, 516-521 Synchronous call, 535 System/360,9-10 System ACL, 847 System call, 23,44-5E example of use, 389-391 UNIX file system 736-740 UNIX I/O, 726-727 UNIX memory management 714-715 UNIX process management, 692-696 UNIX secunty, 755-756 UNIX thread management 696-698 Win32 fiie system, 831-833 Win32 I/O, 825-827 Win32memory management, 816-817 Win32 process management, 799-804 Win32 security, 847-849 Win32 thread management, 799-804 System call interface, 864-866 System services, 786 System V. 13 System V interface definition, 676 Tagged architecture, 650-65! TCB (see Trusted Computing Base) TCP (see Transmission Control Protocol) TCP/IP, 676 Team structure, 891 -893 Template, 573 TENEX secunty flaw 614 Termcap, 341 Terminal character-oriented, 333-342 input software, 336-340 network, 355-363 output software, 341 RS-232, 334-336 Text segment, 50,711

Thermal management, 369

Thin diem. 360-363 U Thompson, Ken, 13 Thrashing. 222 UART. 335 Thread 81-100 UDP (see User Datagram Protocol) hybrid, 94 UID(wUwrlD) kernel, 93 UI (see UNIX International) Linux 702-704 UMA multiprocessor, 506-510 pop-up 96-97 UN1CS, 672-673 UNIX, 701-702 Uniform resource locator, 5s8 user. 90-93 Universal coordinated time, 329 Windows 2000. 804-809 Universal serial bus, 32 worker, 87 UNIX, 13 15-16, 671-758 Thread implementation. 90-93 Berkeley, 675-676 Thread jacket 92 booting, 708-710 Thread model, 81-85 file system, 732-753 Thread scheduling, 150-151 goals, 681-682 Thread system calls. 696-698 history- 672-680 Thread table, 90 interfaces, 682-683 Thread usage, 85-90 I/O, 723-732 Thread wrapper, 92 kernel structure, E87-690 Threat, 584-585 memory management, 710-723 Three-level scheduling, 140-142 overview, 681-690 Tigerteam, 613 paging, 716-717 Tightly-coupled system, 505 password security. 596-598 Timer, 327 PDP-11, 673-674 Timesharing, 11,521-523 portable. 674-675 Time-space trade-off, 884-8S6 processes, 690-710 TLB {see Translation Lookastde Buffer) scheduling algorithm, 704-706 Top-down implementation, 876-877 security, 753-757 TOPS-20, 384 security flaws. 614 Torvalds, Linus, 13 shell, 683-686, 693-694 Track, 24 standard, 676-677 Trampoline, 810 system calls (see System calls) Translation lookaside buffer, 211 threads, 701-702 software management. 212-213 utility programs, 686-688 Transmission controi protocol, 557,676, 726 version 7 file system, 445-448 TRAP door, 610 UNiX international, 677 Trap instruction. 45 Unsafe state, i76-178 Trends in operating system design, 894-897 Upcall. 95 Triple indirect block, 446, 744 URL (see Uniform Resource Locator) Trojan horse, 607-608, 663 USB (tee Universal serial busj Trusted computing base, 654-655 User authentication, 591-606 Trusted system, 653-665 using password, 592-600 TSL instruction, 107-lOS using biometrics. 603-605 Tupie, 572 using physical object, 601-603 Tuple space, 572 User datagram protocol, 726 Turnaround time, 137 User friendly, 15 Two-level directory. 394 User iD. 36, 753 Two-level scheduling, 523 User interface paradigm, 861 Two-phase locking. 183-184 User mode, 3

Usef-space I/O softwdie, 29N-300 User structure, UNIX, 699 UTC (see Universal Coordinated Time) VAP (see Virtual Address Descriptor Value, registry, 775 Vampire tap. 551 VCR coutrol function, 476 Vector graphics 344 Venus, 564 Vice. 564 Video compression, 463-469 Video controller, 344

Videti encoding, 461 Video on demand. 454 Video RAM, 344 Video server 454 Virtual address, 27, 203 Virtual address, space, 203 Virtual address descriptor. 818 Virtual machine, 4, 59-61 Virtual machine monitor 59 Virtual memory, 197, 202-214 Virtual memory interface, 241 -242 Virus, 617-635 avoidance, 633-634 bont sector 623-624 cavity. 622 companion, 619-620 damage scenarios, 618-619 denial of service attack, 618 device driver, 625 dropper, 619 executable program, 620-622 how spread, 626-628 how to fight, 628-635 macro. 625 memory resident, 623 overwriting, 620 parasitic, 622 payload.619 polymorphic, 631 recovery, 635 source code, 626 Virus scanner 629-632 V-node, 750-751 w Wakeup primitive, 108 Wakeup waiting bit, 110 WAN (nee Wide Area Network) War dialer, 594 Watchdog timer, 331 Web browser, 558 Web page. 558 Wide area network. 551 Wide striping, 492 Widget, 357 Wildcard. 649 Wildcard, shell, 684 Win32 API calls, 53-56, 772-774 file system. 8.31-833 I/O, 825-827 memory management. 816-817 process management. 799- S04 security, 847-849 thread management. 799-804 Window. 347 Window manager, 357 Windows 95 history 764-765 Windows 98, file system, 442-445 history, 764-765 Windows 2000. 763-852 booting, 810-811 critical section, 801 device drivers, 827-830 executive, 784-787 file system, 830-844 history, 767-771 I/O, 824-830 1PC. 800-802 kernel, 782-784 master Tile table, 834-839 memory management, 811-823 objects. 787-792 page fault handling, 818-819 page replacement algorithm, 819-823 processes, 796-811 programming, 771-777 registry. 774-777 scheduling. 804-809 >ecurity, 844-849 socket, 801 system calU (see Win32 API calls) system structure, 778-796

Windows driver model, 827 Windows Me, 15,764-765 Windows Millennium edition, 15, Windows NT, 15.765-766 Wireless communication, 368 VM/370. 59-60 Working directory, 39, 396, 733 Working set, 222-225 Working set manager, 820 Wormhole routing, 529 Worst'fit algorithm, 201 Wnte-through cache, 426 WSCbck, 225-227 X terminal, 356 X Window System, 16, 356-360 Xlib, 356 y Yellow book, 3i Zero page thread, 822 ZipFstaw,488 Zombie state, 695

ABOUT THE AUTHOR Andrew S. Tanenbaum has an SB. degree from M.I.T, and a Ph.D. from the University of California at Berkeley, He is currently a Professor of Computer Science at the Vrije Universiteit in Amsterdam, The Netherlands, where he heads the Computer Systems Group. He is also Dean of

the Advanced School for Com- Computing and Imaging, an internniversity graduate school doing research on advanced parallel, distributed, and imaging systems. Nevertheless, he is trying very hard to avoid turning into a bureaucrat. In the past, he has done research on compilers, operating systems, networking, and local-area distributed systems. His current research focuses primarily on the design of wide-area distributed systems that scale to a billion users. These re- research projects have led to over 85 refereed papers in journals and conference proceedings and five books. Prof. Tanenbaum has also produced a considerable volume of software. He was the principal architect of the Amsterdam Compiler Kit, a widely-used toolkit for writing portable compilers, as well as of MINIX, a small UNIX clone intended for use in student programming labs. Together with his Ph.D. students and pro- programmers, he helped design the Amoeba distributed operating system, a high- performance microkernel-based distributed operating system. The MINIx and Amoeba systems are now available for free via the Internet. His Ph.D. students have gone on to greater glory after getting their degrees, He is very proud of them. In this respect he resembles a mother hen. Prof. Tanenbaum is a Fellow of the ACM, a Fellow of the the IEEE, a member of the Royal Netherlands Academy of Arts and Sciences, winner of the 1994 ACM Karl V, Karlstrom Outstanding Educator Award, and winner of the 1997 ACM/SIGCSE Award for Outstanding Contributions to Computer Science Education. He is also listed in Who's Who in the World. His home page on the World Wide Web can be found at URL http://www.cs.va.nl/~ast/ .