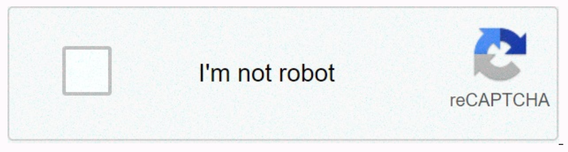
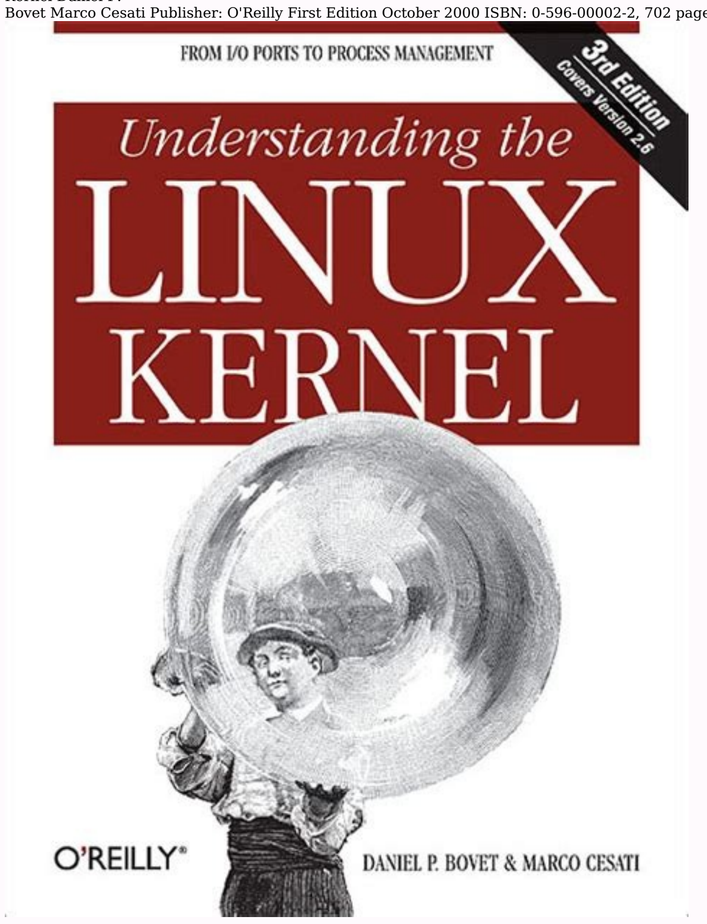


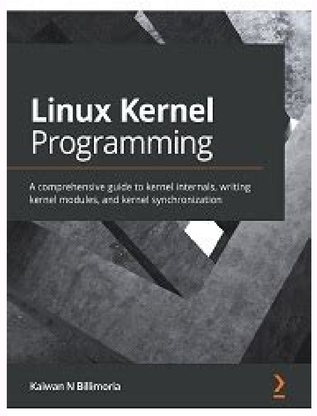
I'm not robot!

Understanding the linux kernel 6th edition pdf

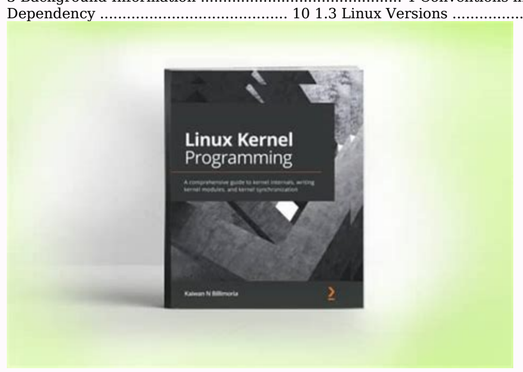
Get full access to Linux in a Nutshell, 6th Edition and 60K+ other titles, with a free 10-day trial of O'Reilly. There are also live events, courses curated by job role, and more. Everything you need to know about Linux is in this book. Written by Stephen Figgins, Ellen Siever, Robert Love, and Arnold Robbins -- people with years of active participation in the Linux community -- Linux in a Nutshell, Sixth Edition, thoroughly covers programming tools, system and network administration tools, the shell, editors, and LILO and GRUB boot loaders. This updated edition offers a tighter focus on Linux system essentials, as well as more coverage of new capabilities such as virtualization, wireless network management, and revision control with git. It also highlights the most important options for using the vast number of Linux commands. You'll find many helpful new tips and techniques in this reference, whether you're new to this operating system or have been using it for years.
Get the Linux commands for system administration and network managementUse hundreds of the most important shell commands available on LinuxUnderstand the Bash shell command-line interpreterSearch and process text with regular expressionsManage your servers via virtualization with Xen and VMwareUse the Emacs text editor and development environment, as well as the vi, ex, and vim text-manipulation toolsProcess text files with the sed editor and the gawk programming languageManage source code with Subversion and git Get full access to Understanding the Linux Kernel and 60K+ other titles, with a free 10-day trial of O'Reilly. There are also live events, courses curated by job role, and more.
Why is Linux so efficient? Is it the right operating system for a particular application? What can be learned from looking at the kernel source code? These are the kinds of questions that Understanding the Linux Kernel takes in stride in this guided tour of the code that forms the core of all Linux operating systems. Linux is presented too often as a casual hacker experiment. It has increasingly become not only a mission-critical part of many organizations, but a sophisticated display of programming skill. It incorporates many advanced operating system concepts and has proven itself extremely robust and efficient for a wide range of uses. Understanding the Linux Kernel helps readers understand how Linux performs best and how it meets the challenge of different environments. The authors introduce each topic by explaining its importance, and show how kernel operations relate to the utilities that are familiar to Unix programmers and users. Major topics include: Memory management, including file buffering, process swapping, and Direct Memory Access (DMA)The Virtual File System and the Second Extended File SystemProcess creation and schedulingSignals, interrupts, and the essential interfaces to device driversTimingSynchronization in the kernelInter-Process Communication (IPC)Program execution Get Mark Richards’s Software Architecture Patterns ebook to better understand how to design components—and how they should interact. It’s yours, free. Get it now Dive in for free with a 10-day trial of the O’Reilly learning platform—then explore all the other resources our members count on to build skills and solve problems every day. Start your free trial Become a member now Page 2 Understanding the Linux Kernel Daniel P.
Bovet Marco Cesati Publisher: O'Reilly First Edition October 2000 ISBN: 0-596-00002-2, 702 pages Understanding the Linux Kernel helps readers understand how Linux performs best and how it meets the challenge of different environments.



The authors introduce each topic by explaining its importance, and show how kernel operations relate to the utilities that are familiar to Unix programmers and users. Page 3 Table of Contents Preface ..... 1 The Audience for This Book ..... 1 Organization of the Material ..... 1 Overview of the Book .....

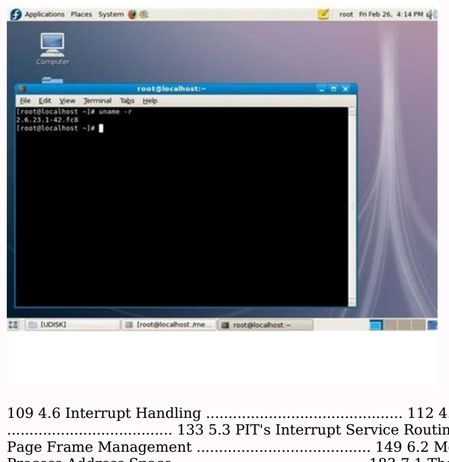


3 Background Information ..... 4 Conventions in This Book ..... 4 How to Contact Us ..... 4 Acknowledgments ..... 5 1. Introduction ..... 6 1.1 Linux Versus Other Unix-Like Kernels ..... 6 1.2 Hardware Dependency ..... 10 1.3 Linux Versions ..... 11 1.4 Basic Operating System Concepts ..... 12 1.5 An Overview of the Unix Filesystem ..... 16 1.6 An Overview of Unix Kernels .....



22 2. Memory Addressing ..... 36 2.1 Memory Addresses ..... 36 2.2 Segmentation in Hardware ..... 37 2.3 Segmentation in Linux ..... 41 2.4 Paging in Hardware ..... 44 2.5 Paging in Linux ..... 52 2.6 Anticipating Linux 2.4 .....

63 3. Processes ..... 64 3.1 Process Descriptor ..... 64 3.2 Process Switching ..... 78 3.3 Creating Processes ..... 86 3.4 Destroying Processes ..... 93 3.5 Anticipating Linux 2.4 ..... 94 4. Interrupts and Exceptions ..... 96 4.1 The Role of Interrupt Signals ..... 96 4.2 Interrupts and Exceptions ..... 97 4.3 Nested Execution of Exception and Interrupt Handlers ..... 106 4.4 Initializing the Interrupt Descriptor Table ..... 107 4.5 Exception Handling .....



109 4.6 Interrupt Handling ..... 112 4.7 Returning from Interrupts and Exceptions ..... 126 4.8 Anticipating Linux 2.4 ..... 129 5. Timing Measurements ..... 131 5.1 Hardware Clocks ..... 131 5.2 The Timer Interrupt Handler ..... 133 5.3 PIT's Interrupt Service Routine ..... 134 5.4 The TIMER_BH Bottom Half Functions ..... 136 5.5 System Calls Related to Timing Measurements ..... 145 5.6 Anticipating Linux 2.4 ..... 148 Page 4 6. Memory Management ..... 149 6.1 Page Frame Management ..... 149 6.2 Memory Area Management ..... 160 6.3 Noncontiguous Memory Area Management ..... 176 6.4 Anticipating Linux 2.4 ..... 181 7. Process Address Space ..... 183 7.1 The Process's Address Space ..... 183 7.2 The Memory Descriptor ..... 185 7.3 Memory Regions ..... 186 7.4 Page Fault Exception Handler ..... 201 7.5 Creating and Deleting a Process Address Space ..... 212 7.6 Managing the Heap ..... 214 7.7 Anticipating Linux 2.4 ..... 216 8. System Calls ..... 217 8.1 POSIX APIs and System Calls ..... 217 8.2 System Call Handler and Service Routines ..... 218 8.3 Wrapper Routines ..... 229 8.4 Anticipating Linux 2.4 ..... 230 9. Signals ..... 231 9.1 The Role of Signals ..... 231 9.2 Sending a Signal ..... 239 9.3 Receiving a Signal ..... 242 9.4 Real-Time Signals ..... 251 9.5 System Calls Related to Signal Handling ..... 252 9.6 Anticipating Linux 2.4 ..... 257 10. Process Scheduling ..... 258 10.1 Scheduling Policy ..... 258 10.2 The Scheduling Algorithm ..... 261 10.3 System Calls Related to Scheduling ..... 272 10.4 Anticipating Linux 2.4 ..... 276 11. Kernel Synchronization ..... 277 11.1 Kernel Control Paths ..... 277 11.2 Synchronization Techniques ..... 278 11.3 The SMP Architecture ..... 286 11.4 The Linux/SMP Kernel ..... 290 11.5 Anticipating Linux 2.4 ..... 302 12. The Virtual Filesystem ..... 303 12.1 The Role of the VFS ..... 303 12.2 VFS Data Structures ..... 308 12.3 Filesystem Mounting ..... 324 12.4 Pathname Lookup ..... 329 12.5 Implementations of VFS System Calls ..... 333 12.6 File Locking ..... 337 12.7 Anticipating Linux 2.4 ..... 342 Page 5 13. Managing I/O Devices ..... 343 13.1 I/O Architecture ..... 343 13.2 Associating Files with I/O Devices ..... 348 13.3 Device Drivers ..... 353 13.4 Character Device Handling ..... 360 13.5 Block Device Handling ..... 361 13.6 Page I/O Operations ..... 377 13.7 Anticipating Linux 2.4 ..... 380 14. Disk Caches ..... 382 14.1 The Buffer Cache ..... 383 14.2 The Page Cache ..... 396 14.3 Anticipating Linux 2.4 ..... 398 15. Accessing Regular Files ..... 400 15.1 Reading and Writing a Regular File ..... 400 15.2 Memory Mapping ..... 408 15.3 Anticipating Linux 2.4 ..... 416 16. Swapping: Methods for Freeing Memory ..... 417 16.1 What Is Swapping? ..... 417 16.2 Swap Area ..... 420 16.3 The Swap Cache ..... 429 16.4 Transferring Swap Pages ..... 433 16.5 Page Swap-Out ..... 437 16.6 Page Swap-In ..... 442 16.7 Freeing Page Frames ..... 444 16.8 Anticipating Linux 2.4 ..... 450 17. The Ext2 Filesystem ..... 451 17.1 General Characteristics ..... 451 17.2 Disk Data Structures ..... 453 17.3 Memory Data Structures ..... 459 17.4 Creating the Filesystem ..... 463 17.5 Ext2 Methods ..... 464 17.6 Managing Disk Space ..... 466 17.7 Reading and Writing an Ext2 Regular File ..... 473 17.8 Anticipating Linux 2.4 ..... 475 18. Process Communication ..... 476 18.1 Pipes ..... 477 18.2 FIFOs ..... 483 18.3 System V IPC ..... 486 18.4 Anticipating Linux 2.4 ..... 499 19. Program Execution ..... 500 19.1 Executable Files ..... 500 19.2 Executable Formats ..... 512 19.3 Execution Domains ..... 514 19.4 The exec-like Functions ..... 515 19.5 Anticipating Linux 2.4 ..... 519 Page 6 A. System Startup ..... 520 A.1 Prehistoric Age: The BIOS ..... 520 A.2 Ancient Age: The Boot Loader ..... 521 A.3 Middle Ages: The setup( ) Function ..... 523 A.4 Renaissance: The startup_32( ) Functions ..... 523 A.5 Modern Age: The start_kernel( ) Function ..... 524 B. Modules ..... 526 B.1 To Be (a Module) or Not to Be? ..... 526 B.2 Module Implementation ..... 527 B.3 Linking and Unlinking Modules ..... 529 B.4 Linking Modules on Demand ..... 531 C. Source Code Structure ..... 533 Colophon ..... 536 Page 7

Understanding the Linux Kernel Preface In the spring semester of 1997, we taught a course on operating systems based on Linux 2.0. The idea was to encourage students to read the source code. To achieve this, we assigned term projects consisting of making changes to the kernel and performing tests on the modified version.
We also wrote course notes for our students about a few critical features of Linux like task switching and task scheduling. We continued along this line in the spring semester of 1998, but we moved on to the Linux 2.1 development version. Our course notes were becoming larger and larger. In July, 1998 we contacted O'Reilly & Associates, suggesting they publish a whole book on the Linux kernel. The real work started in the fall of 1998 and lasted about a year and a half. We read thousands of lines of code, trying to make sense of them. After all this work, we can say that it was worth the effort. We learned a lot of things you don't find in books, and we hope we have succeeded in conveying some of this information in the following pages. The Audience for This Book All people curious about how Linux works and why it is so efficient will find answers here. After reading the book, you will find your way through the many thousands of lines of code, distinguishing between crucial data structures and secondary ones—in short, becoming a true Linux hacker. Our work might be considered a guided tour of the Linux kernel: most of the significant data structures and many algorithms and programming tricks used in the kernel are discussed; in many cases, the relevant fragments of code are discussed line by line. Of course, you should have the Linux source code on hand and should be willing to spend some effort deciphering some of the functions that are not, for sake of brevity, fully described. On another level, the book will give valuable insights to people who want to know more about the critical design issues in a modern operating system.
It is not specifically addressed to system administrators or programmers; it is mostly for people who want to understand how things really work inside the machine! Like any good guide, we try to go beyond superficial features. We offer background, such as the history of major features and the reasons they were used. Organization of the Material When starting to write this book, we were faced with a critical decision: should we refer to a specific hardware platform or skip the hardware-dependent details and concentrate on the pure hardware-independent parts of the kernel? Others books on Linux kernel internals have chosen the latter approach; we decided to adopt the former one for the following reasons: • Efficient kernels take advantage of most available hardware features, such as addressing techniques, caches, processor exceptions, special instructions, processor control registers, and so on.
If we want to convince you that the kernel indeed does 1 Page 8 Understanding the Linux Kernel quite a good job in performing a specific task, we must first tell what kind of support comes from the hardware. • Even if a large portion of a Unix kernel source code is processor-independent and coded in C language, a small and critical part is coded in assembly language. A thorough knowledge of the kernel thus requires the study of a few assembly language fragments that interact with the hardware. When covering hardware features, our strategy will be quite simple: just sketch the features that are totally hardware-driven while detailing those that need some software support. In fact, we are interested in kernel design rather than in computer architecture. The next step consisted of selecting the computer system to be described: although Linux is now running on several kinds of personal computers and workstations, we decided to concentrate on the very popular and cheap IBM-compatible personal computers—thus, on the Intel 80x86

microprocessors and on some support chips included in these personal computers. The term Intel 80x86 microprocessor will be used in the forthcoming chapters to denote the Intel 80386, 80486, Pentium, Pentium Pro, Pentium II, and Pentium III microprocessors or compatible models. In a few cases, explicit references will be made to specific models. One more choice was the order followed in studying Linux components. We tried to follow a bottom-up approach: start with topics that are hardware-dependent and end with those that are totally hardware-independent. In fact, we'll make many references to the Intel 80x86 microprocessors in the first part of the book, while the rest of it is relatively hardware- independent. Two significant exceptions are made in Chapter 11, and Chapter 13.

In practice, following a bottom-up approach is not as simple as it looks, since the areas of memory management, process management, and filesystem are intertwined; a few forward references—that is, references to topics yet to be explained—are unavoidable.

Each chapter starts with a theoretical overview of the topics covered. The material is then presented according to the bottom-up approach. We start with the data structures needed to support the functionalities described in the chapter. Then we usually move from the lowest level of functions to higher levels, often ending by showing how system calls issued by user applications are supported. Level of Description Linux source code for all supported architectures is contained in about 4500 C and Assembly files stored in about 270 subdirectories; it consists of about 2 million lines of code, which occupy more than 58 megabytes of disk space. Of course, this book can cover a very small portion of that code. Just to figure out how big the Linux source is, consider that the whole source code of the book you are reading occupies less than 2 megabytes of disk space. Therefore, in order to list all code, without commenting on it, we would need more than 25 [1] books like this! [1] Nevertheless, Linux is a tiny operating system when compared with other commercial giants. Microsoft Windows 2000, for example, reportedly has more than 30 million lines of code. Linux is also small when compared to some popular applications; Netscape Communicator 5 browser, for example, has about 17 million lines of code. So we had to make some choices about the parts to be described. This is a rough assessment of our decisions: 2 Page 9 Understanding the Linux Kernel • We describe process and memory management fairly thoroughly. • We cover the Virtual Filesystem and the Ext2 filesystem, although many functions are just mentioned without detailing the code; we do not discuss other filesystems supported by Linux. • We describe device drivers, which account for a good part of the kernel, as far as the kernel interface is concerned, but do not attempt analysis of any specific driver, including the terminal drivers. • We do not cover networking, since this area would deserve a whole new book by itself. In many cases, the original code has been rewritten in an easier to read but less efficient way. This occurs at time-critical points at which sections of programs are often written in a mixture of hand-optimized C and Assembly code. Once again, our aim is to provide some help in studying the original Linux code.

While discussing kernel code, we often end up describing the underpinnings of many familiar features that Unix programmers have heard of and about which they may be curious (shared and mapped memory, signals, pipes, symbolic links). Overview of the Book To make life easier, Chapter 1 presents a general picture of what is inside a Unix kernel and how Linux competes against other well-known Unix systems. The heart of any Unix kernel is memory management. Chapter 2 explains how Intel 80x86 processors include special circuits to address data in memory and how Linux exploits them. Processes are a fundamental abstraction offered by Linux and are introduced in Chapter 3. Here we also explain how each process runs either in an unprivileged User Mode or in a privileged Kernel Mode. Transitions between User Mode and Kernel Mode happen only through well-established hardware mechanisms called interrupts and exceptions, which are introduced in Chapter 4. One type of interrupt is crucial for allowing Linux to take care of elapsed time; further details can be found in Chapter 5. Next we focus again on memory: Chapter 6 describes the sophisticated techniques required to handle the most precious resource in the system (besides the processors, of course), that is, available memory. This resource must be granted both to the Linux kernel and to the user applications. Chapter 7 shows how the kernel copes with the requests for memory issued by greedy application programs. Chapter 8 explains how a process running in User Mode makes requests to the kernel, while Chapter 9 describes how a process may send synchronization signals to other processes. Chapter 10 explains how Linux executes, in turn, every active process in the system so that all of them can progress toward their completions. Synchronization mechanisms are needed by the kernel too: they are discussed in Chapter 11 for both uniprocessor and multiprocessor systems. Now we are ready to move on to another essential topic, that is, how Linux implements the filesystem. A series of chapters covers this topic: Chapter 12 introduces a general layer that supports many different filesystems. Some Linux files are special because they provide 3 Page 10 Understanding the Linux Kernel trapdoors to reach hardware devices; Chapter 13 offers insights on these special files and on the corresponding hardware device drivers.

Another issue to be considered is disk access time; Chapter 14 shows how a clever use of RAM reduces disk accesses and thus improves system performance significantly. Building on the material covered in these last chapters, we can now explain in Chapter 15, how user applications access normal files. Chapter 16 completes our discussion of Linux memory management and explains the techniques used by Linux to ensure that enough memory is always available. The last chapter dealing with files is Chapter 17, which illustrates the most-used Linux filesystem, namely Ext2. The last two chapters end our detailed tour of the Linux kernel: Chapter 18 introduces communication mechanisms other than signals available to User Mode processes; Chapter 19 explains how user applications are started. Last but not least are the appendixes: Appendix A sketches out how Linux is booted, while Appendix B describes how to dynamically reconfigure the running kernel, adding and removing functionalities as needed.

Appendix C is just a list of the directories that contain the Linux source code. The Source Code Index includes all the Linux symbols referenced in the book; you will find here the name of the Linux file defining each symbol and the book's page number where it is explained. We think you'll find it quite handy. Background Information No prerequisites are required, except some skill in C programming language and perhaps some knowledge of Assembly language. Conventions in This Book The following is a list of typographical conventions used in this book: Constant Width Is used to show the contents of code files or the output from commands, and to indicate source code keywords that appear in code. Italic Is used for file and directory names, program and command names, command-line options, URLs, and for emphasizing new terms. How to Contact Us We have tested and verified all the information in this book to the best of our abilities, but you may find that features have changed or that we have let errors slip through the production of the book. Please let us know of any errors that you find, as well as suggestions for future editions, by writing to: O'Reilly & Associates, Inc. 101 Morris St. Sebastopol, CA 95472 (800) 998-9938 (in the U.S. or Canada) (707) 829-0515 (international/local) (707) 829-0104 (fax) 4 You can’t perform that action at this time.