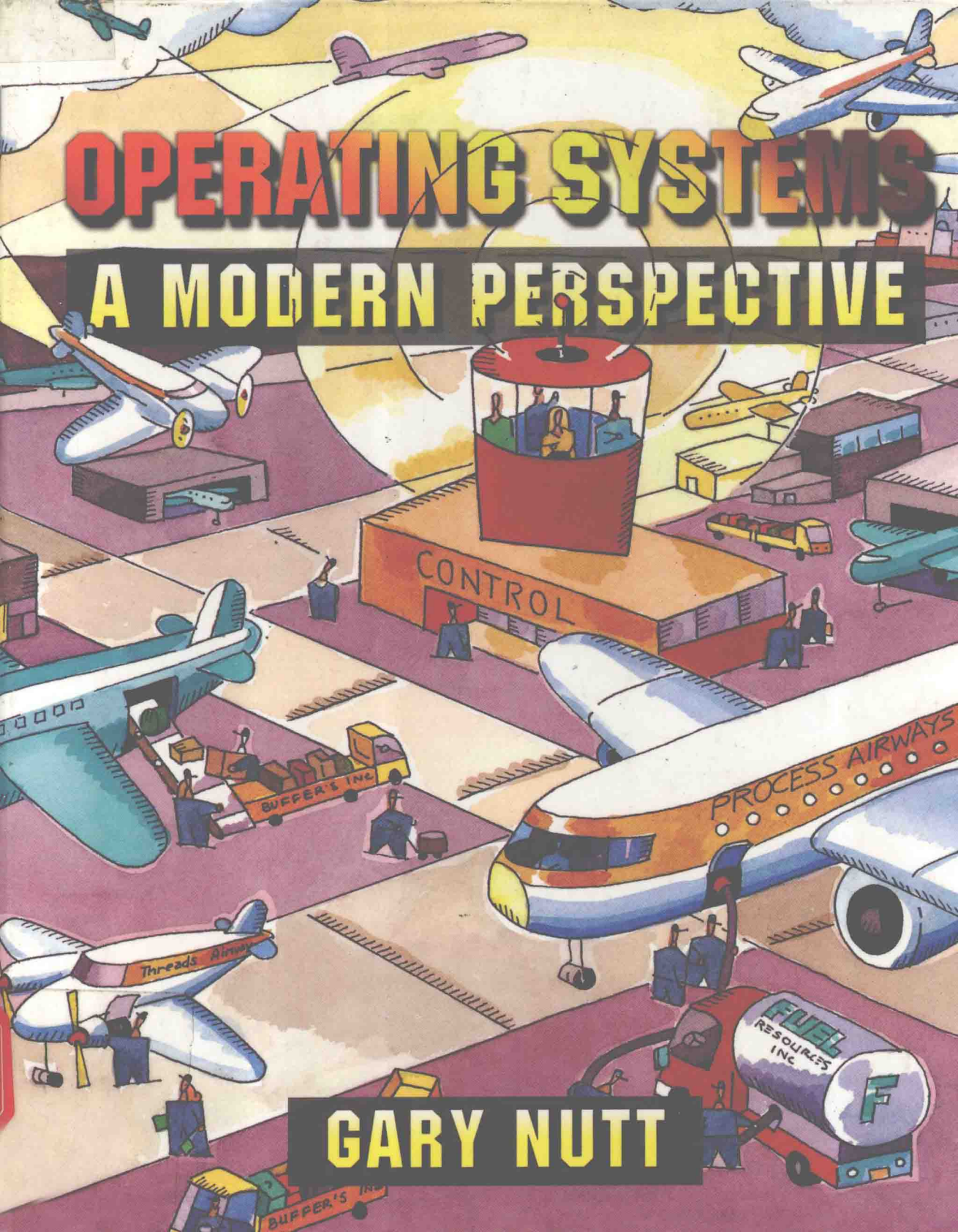
OPERATING SYSTEMS

A MODERN PERSPECTIVE



GARY NUTT

Operating Systems

A MODERN PERSPECTIVE

Gary J. Nutt

UNIVERSITY OF COLORADO

An imprint of Addison Wesley Longman, Inc.

Reading, Massachusetts ■ Harlow, England ■ Menlo Park, California ■ Berkeley, California
Don Mills, Ontario ■ Sydney ■ Bonn ■ Amsterdam ■ Tokyo ■ Mexico City

Senior Acquisitions Editor: J. Carter Shanklin
Editorial Assistant: Angela Buenning
Production Editor: Patricia A. O. Unubun
Compositor: TKM Productions
Text Designer: Paul Uhl, Design Associates
Cover Illustrator: Robin Jareaux
Cover Designer: Eileen Hoff

Access the latest information about Addison-Wesley titles from our World Wide Web site:
http://www.awl.com/cseng

Library of Congress Catalog Card Number: 97-70383

ISBN 0-8053-12951
1 2 3 4 5 6 7 8 9 10-MA-0100999897
First printing, March 1997

Preface

Operating systems is an exciting software area because the design of an operating system exerts a major influence on the overall function and performance of the entire computer. Today there are basically three types of books that introduce this topic to undergraduates:

- The first type provides a detailed description of one or more different operating systems. This approach is particularly useful if you just want to know how a particular operating system works without learning much about how *all* operating systems work.
- The second type is an evolution of the first in that it bases all its explanation around a single pedagogical system. It usually provides enough information to derive the underlying principles of good operating systems, but these principles often are obscured by the explanation of the pedagogical system.
- The third type is written to be independent of any particular operating system. It focuses on the theory behind all operating systems. It tends to leave you without that important link to how an operating system is really built.

I strongly believe that it is important to understand the principles behind the designs of all operating systems and to see how those principles are put into practice in real operating systems. The goal of this book is to provide a complete discussion of theory, along with an extensive set of coding and algorithm examples. The material is presented so that the student can easily differentiate among operating systems fundamentals and the detailed coding examples. Highlighted throughout are issues of performance, since this is a fundamental factor affecting how an operating system is designed. However, the coverage of performance is not intended to be comprehensive in nature. Additional emphasis is also placed on how various parts of the system are related to real-world demand and hardware constraints. I decided to forego extensive coverage of analysis and performance theory in favor of explanation of performance issues. Students should have plenty of time to study analysis techniques in a graduate-level operating system course.

The main part of the text presents operating systems fundamentals, along with general examples. It also includes three types of examples: *In The Cockpit* examples illustrate how to use concepts being explained in the chapter. *In The Hangar* examples focus on how these concepts can be implemented, and performance issues are highlighted in *Performance Tuning* boxes. If desired, the reader could ignore these examples and focus only on operating systems theory. Or one could browse the main text but focus primarily on these examples for a course that emphasizes a hands-on approach to operating systems.

The intent of including a variety of code examples is to describe operating systems more concretely and provide samples of the implementation of OS theory. Sometimes these examples include descriptions at the level of programs and algorithms. A few of these code examples are complete programs that have been compiled and executed. Most examples, however, are simply descriptions of algorithms or techniques using the C programming language. These descriptions deliberately omit detail that would be necessary in an actual implementation but that do not contribute to the understanding of the algorithm. The context in which the code appears should make clear when the code is an actual implementation; otherwise it should always be assumed to be a *description* of an algorithm or technique. I believe that detailed descriptions are mandatory for describing operating systems. I also believe that full implementations contain so many code-specific details that the essence of the ideas being illustrated are difficult to abstract and separate from the details of the language implementation. I have experimented with using pseudo code languages for these descriptions, but students and reviewers have consistently preferred the use of C. Be careful not to interpret the descriptions in C as complete implementations.

Topic Order

The order of presentation is based on my experience teaching operating systems courses with input from many experienced instructors. This organization thus reflects the combined knowledge and experience of many different teachers and I believe the result is logical, conducive to learning, and generally accepted by most operating system instructors.

Each chapter begins with a transition from the previous chapter and a preview of what is covered in the current chapter. Students can look at this material as well as the summary at the end of the chapter to get a quick idea of what a chapter is about.

Chapters 1–4 consist of important introductory material that provides a solid foundation for the study of operating systems. Teachers may decide to go over this material rather quickly, perhaps assigning it as outside reading material, especially if this was covered in other courses. However, understanding this material is critical before one dives into the further study of the meat of operating systems, starting in Chapter 5.

- Chapter 1 shows how operating systems fit into software technology. In earlier drafts, an historical perspective had been included; instructors tend to like a little history and context, but many students think it is boring, so we have dispensed with history.
- Chapter 2 is unique among operating system books in that it considers how to *use* an operating system, particularly how to use multiple processes. This chapter was added because my experience with computer science juniors and seniors is that they may have written considerable single-threaded code but are far less likely to have written or studied *multithreaded* software. This chapter offers an immediate opportunity to learn this new material.
- Chapter 3 describes the fundamental organization of operating systems, including implementation strategies.
- Chapter 4 finishes the preliminaries for studying operating systems—computer organization. For students who have already taken a computer organization class, the first half of Chapter 4 will be elementary. The second half describes interrupts, emphasizing the aspects that are critical to operating systems.

Chapter 5 describes device management, specifically general techniques, buffering, and device drivers. It is tempting to become completely immersed in Linux device drivers. However, I resisted this temptation to focus instead on a macro-level view of the purpose and general organization of interrupt-driven I/O. Included are extensive boxed sections on device drivers, but these stop short of providing an actual Linux driver. The chapter examines devices before considering processes because devices provide an elementary case in which physical concurrency exists in the computer and the software must be carefully designed to control that concurrency. This provides a natural introduction to process management.

Chapters 6–10 are devoted to process management. They start from the basic tasks and organization of process and resource managers (Chapter 6) and move to scheduling (Chapter 7), synchronization (Chapters 8 and 9), and deadlock (Chapter 10).

Chapter 11 deals with traditional issues in memory management, while Chapter 12 covers the contemporary approach to memory managers using virtual memory. Because of the popularity of paging, most of the discussion is directed at this technology. However, with the current trends in memory technology, it would be a mistake to ignore segmentation. Thus part of this discussion deals with segmentation. Unfortunately, the best example of a robust segmentation system is still the (now obsolete) Multics system.

Chapter 13 describes file management. Less space is devoted to file management than is customary in operating systems books because it is not as difficult to understand as process management and memory management. This discussion is augmented in Chapter 16, which deals with remote files.

Chapter 14 provides a general discussion of protection mechanisms and security policies. It might be argued that this section belongs in the process management discussion, although much of the technology is just as closely related to files, memory, and other resources. It is much easier for someone to appreciate the need for protection and security after they have seen the process, memory, and file managers.

Chapters 15–17 introduce operating system technology to support distributed computing. Distributed computing is a dominant aspect of modern operating systems and I feel strongly that coverage of this important issue belongs in all introductory texts on operating systems.

The study of operating systems has traditionally been one of the most challenging and exciting software disciplines in computer science. I hope this book makes complex aspects of operating systems easy to understand and avoids making simple aspects boring. Good luck in your study of operating systems; I hope you enjoy it as much as I do!

To the Instructor

Thank you for choosing this book as your aide in teaching undergraduates about operating systems. Operating systems continues to be an essential computer science course, yet as I have taught it over many years, I have become increasingly dissatisfied with the OS texts that are available. The types of textbooks available do not fit the needs of my students. In my operating systems courses, I find that the students often have a variety of backgrounds, although most will have taken an introductory programming class, a data structures class, and perhaps a class on machine organization. At UC-Boulder, the operating systems course is also a popular class for electrical engineers and other majors in addition to computer science students. Increasingly I find that some students will have found ways to learn considerable amounts about system software by experimenting on their own. Given such a varied background and base of knowledge, one of the largest challenges in an operating systems class is to continue to encourage experimentation with specific systems while developing an understanding and appreciation for the fundamentals behind all operating systems.

The goal of this book is to meet that challenge. I advocate (and use) a balanced approach to studying operating systems, combining a hands-on approach with a study of theoretical issues and theory. Some reviewers took exception with the amount of introductory material, which is presented in Chapters 1–4. In my classes, I usually cover the material in Chapters 1, 3, and 4 very rapidly and rely on the student to read the material to fill in gaps. I have found that it is really worth the time to lecture on the material in Chapter 2. Very few students have used `fork` and `exec` (or their analogs in non-UNIX systems) before they take an operating systems course. I have students write simple shells at the beginning of the class so they can explore these basic concepts.

I start the detailed discussion of operating systems with device management. At first, you may find this approach awkward, although it follows the traditional evolution of operating systems. A natural segue exists from the discussion of interrupts in Chapter 4 to the discussion of device management in Chapter 5. This approach also allows you to get your students involved in hands-on operating system exercises early in the term, since device drivers are the easiest part of

the operating system for students to learn enough about to begin modifying code and writing new drivers. By addressing this first, you can prepare the student for more difficult kernel exercises presented later in the term. Most important, it provides a simplified environment for introducing independent threads of execution (in the hardware and the software), concurrency, and synchronization.

I have included two programming exercises at the end of Chapter 5 to experiment with Linux device drivers. You may find that for some students the second one is too complicated, since students must have a certain amount of knowledge of kernel operation before they can really understand how to solve it. Students also will require extra information from Linux sources, and perhaps some highly specific information before they can really write the FIFO device driver. Many instructors like to get to the details of process management as early in the course as possible. I believe that the device management chapter allows you to introduce these critical concepts early. Then you can elaborate on this material by considering the basic organization of process and resource management, scheduling, synchronization, and deadlock.

Memory management is also important and another topic that instructors usually want to get to as soon as possible. I choose to phase it in after process management and then move to file management. Then I finish the essential material with a discussion of protection and security, which is deferred until the student has had a chance to absorb the notions of process and various kinds of resources (generic resources, memory, and files).

Some reviewers were concerned about the inclusion of distributed computation in Chapters 15–17. Any contemporary operating system must be built to operate in (or be evolved to) distributed systems. All current research on operating systems is deeply influenced by distributed operating systems. In a one-semester undergraduate course, I spend two to three weeks on this material, although it is not covered at the same depth as the earlier material.

Because of the nature of commercial systems and networks, it would be unforgivable to completely ignore these topics in an operating systems course. It is impossible to organize this material so that it meets every instructor's desires. The organization I use in my course is reflected in the book. However, there is no particular harm caused by shuffling the material to suit your own desires.

If you do not like to talk about device management until after you talk about file management, then skip Chapter 5 until after you have completed Chapter 13.

I included several applied exercises that are challenging and relatively time consuming, especially if attempted without considerable guidance by you or your teaching assistant. These exercises are set off from the others in the chapters as sections on Programming Practices and Exploring Linux code-reading problems. Some reviewers felt these problems were too difficult to include without a warning. Most of the problems have been assigned previously in my own undergraduate operating system class. I have found them to be important in providing an opportunity for detailed hands-on experience in operating systems, even though they require extra effort by the instructor.

Today, there is a wealth of information on operating systems available on the Internet through `ftp` sites and now on the World Wide Web that I would encourage you to point your students towards. Because such sites change so frequently I am maintaining a Web page at:

http://www.cs.colorado.edu/~nutt/osamp.html

where I keep a current set of links to relevant operating systems information. If you have some material that should be shared with our readers, let me know; and I will add it to the page (email me at osamp@cs.colorado.edu). I also welcome your questions, comments, suggestions, and advice (and I will even try to accept your criticism in good humor `:-))`.

Many people have helped to edit and refine this book. First there are the students at the University of Colorado, especially Sam Siewart, Scott Brandt, and Don Lindsay. Addison-Wesley arranged to have additional students from other institutions look at the manuscript: Eric F. Stuckey, Shawn Lauzon, Dan Dartman, and Nick Tkach at Montana State University, and Jeffrey Ramin now at Berbee Information Networks Corporation.

Next were the many people who spent hours looking at drafts or otherwise suggesting ways to organize and improve it: Divy Agrawal (University of California at Santa Barbara), Vladamir Akis (California State University at Los Angeles), Kasi Anantha (San Diego State University), Charles J. Antonelli (University of Michigan), Lewis Barnett (University of Richmond), Lubomir F. Bic (University of California, Irvine), Paosheng Chang (Lucent Technologies), Randy Chow (University of Florida), Wesley J. Chun, Carolyn J. Crouch (University of Minnesota, Duluth), Peter G. Drexel (Plymouth State College), Joseph Faletti, Gary Harkin (Montana State University), Dr. Sallie Henry (Virginia Tech), Mark A. Holliday (Western Carolina University), Kevin Jeffay (University of North Carolina at Chapel Hill), Phil Kearns (The College of William and Mary), Qiang Li (University of Santa Clara), Darrell Long (University of California), Junsheng Long, Michael Lutz (Rochester Institute of Technology), Carol McNamee (Sacremento State University), Donald Miller (Arizona State University), Jim Mooney (West Virginia University), Ethan V. Munson (University of Wisconsin - Milwaukee), Douglas Salane (John Jay College), C.S. (James) Wong (San Francisco State University), and Salih Yurttas (Texas A&M University). Thank you all for sharing your experience, insight, and suggestions.

Finally, the editorial staff at Addison-Wesley and several freelance consultants have been invaluable in helping me produce this book, especially Christine Kulke, Angela Buenning, Rebecca Johnson, Dusty Bernard, Laura Michaels, Pat Unubun, Dan Joraanstad, Nate McFadden, and most of all Carter Shanklin.

The book has benefited immensely by these collective efforts, but of course the remaining errors are solely my responsibility.

Gary J. Nutt
Boulder, Colorado

Contents