

1
国外著名高等院校
信息科学与技术优秀教材

Addison
Wesley

操作系统：现代观点(第二版 实验更新版)

Operating Systems: A Modern Perspective

Second Edition

Lab Update

Gary Nutt

3



英文版

人民邮电出版社

POSTS & TELECOMMUNICATIONS PRESS

国外著名高等院校信息科学与技术优秀教材

操作系统：现代观点（第二版 实验更新版）

Operating Systems: A Modern Perspective
Second Edition Lab Update

英文版

“本书采纳了最新的观点，并为实际操作系统的编程提供了清晰明了的方法。如我所料，这本书具有很强的竞争力。”

——David Binger 教授
中央大学

“该书目的明确，形式新颖，内容深邃广泛，具有一定的可读性，我非常欣赏此书。”

——Richard Guy教授
加利福尼亚大学

本书从全新的角度出发，在原则与实践之间找到了一个折中的办法，即通过实例阐述核心的操作系统概念。本书有以下特点：详细讨论了操作系统的原则，同时论述了编码、算法、实现等相关问题，增加了实验练习以便于大家更好地理解、掌握现代操作系统。

- ◆ Linux 和 Windows 2000 最新版本的全新资料。
- ◆ 操作系统设计的基本原理。
- ◆ In The Hangar 的例子表明这些原理在 Linux/UNIX 和 Windows 2000 操作系统中的实际应用。
- ◆ Performance Tuning 阐明系统设计者如何开发利用基本原理提高运行效率。
- ◆ Lab Exercise 使读者通过学习如何应用 Linux、UNIX 和 Windows 2000 获得实际经验。

限中国大陆地区销售

ISBN 7-115-10344-5



9 787115 103444 >



ISBN7-115-10344-5/TP·2903
定价：52.00 元

人民邮电出版社

http://www.ptpress.com.cn

TP316

43

国外著名高等院校信息科学与技术优秀教材

操作系统：现代观点
(第二版 实验更新版)
(英文版)

Operating Systems: A Modern Perspective

Second Edition

Lab Update ■

Gary Nutt

人民邮电出版社

图书在版编目 (CIP) 数据

操作系统: 现代观点: 第 2 版: 实验更新版: 英文版 / () 纳特 (Nutt, G.) 等编著.
—北京: 人民邮电出版社, 2002.8

国外著名高等院校信息科学与技术优秀教材

ISBN 7-115-10344-5

I. 操... II. 纳... III. 操作系统 (软件) —高等学校—教材—英文 IV. TP316

中国版本图书馆 CIP 数据核字 (2002) 第 055202 号

版 权 声 明

English Reprint Edition Copyright © 2002 by PEARSON EDUCATION NORTH ASIA LIMITED and POSTS & TELECOMMUNICATIONS PRESS.

Operating Systems: A Modern Perspective, Second Edition, Lab Update

By Gary Nutt

Copyright © 2002

All Rights Reserved.

Published by arrangement with the original publisher, Pearson Education, Inc., publishing as Addison Wesley.

This edition is authorized for sale only in People's Republic of China (excluding the Special Administrative Region of Hong Kong and Macao).

本书封面贴有 **Pearson Education** (培生教育出版集团) 激光防伪标签, 无标签者不得销售。

国外著名高等院校信息科学与技术优秀教材
操作系统: 现代观点 (第二版 实验更新版)
(英文版)

-
- ◆ 著 Gary Nutt
责任编辑 李 际
 - ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子函件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
读者热线 010-67180876
北京汉魂图文设计有限公司制作
北京朝阳展望印刷厂印刷
新华书店总店北京发行所经销
 - ◆ 开本: 800×1000 1/16
印张: 44.5
字数: 945 千字 2002 年 8 月第 1 版
印数: 1-3 000 册 2002 年 8 月北京第 1 次印刷
著作权合同登记 图字: 01-2002-3752 号

ISBN 7-115-10344-5/TP · 2903

定价: 52.00 元

本书如有印装质量问题, 请与本社联系 电话: (010) 67129223

内 容 提 要

本书是一本操作系统课程的学习教材，全书共分十八章。第一至第四章是介绍性内容，是学习操作系统的基础，其中第一章介绍操作系统是什么和操作系统是如何发展到今天的现状的；第二章介绍如何使用操作系统，特别是多进程操作系统；第三章介绍操作系统的基本组织和实现策略；第四章介绍计算机的组织。从第五章开始介绍了操作系统的各个部分，首先在第五章介绍设备管理，特别是基本技术、缓冲区和设备驱动。第六章至第十章介绍进程管理，其中第六章介绍基本的任务概念、进程组织和资源管理；第七章介绍进程调度；第八章介绍基本的同步方法；第九章介绍高层的同步方法；第十章介绍死锁问题。在第十一章介绍了内存管理，第十二章介绍虚拟内存管理，第十三章介绍文件管理，第十四章介绍保护机制和安全策略。第十五章至十七章介绍了支持分布式计算的操作系统技术，其中第十五章介绍网络管理；第十六章介绍远程文件管理；第十七章介绍分布计算，包括分布式进程管理、消息传递机制、远程过程调用和分布式内存管理。最后在第十八章介绍了一些操作系统实例，包括 UNIX、Linux、Windows NT、Mac OS 和 Chorus 微内核操作系统。

本书是作为计算机科学和工程类专业教材编写的，也可供有关科技人员参考。

出版说明

2001年,教育部印发了《关于“十五”期间普通高等教育教材建设与改革的意见》。该文件明确指出,“九五”期间原国家教委在“抓好重点教材,全面提高质量”方针指导下,调动了各方面的积极性,产生了一大批具有改革特色的新教材。然而随着科学技术的飞速发展,目前高校教材建设工作仍滞后于教学改革的实践,一些教材内容陈旧,不能满足按新的专业目录修订的教学计划和课程设置的需要。为此该文件明确强调,要加强国外教材的引进工作。当前,引进的重点是信息科学与技术 and 生物科学与技术两大学科的教材。要根据专业(课程)建设的需要,通过深入调查、专家论证,引进国外优秀教材。要注意引进教材的系统配套,加强对引进教材的宣传,促进引进教材的使用和推广。

邓小平同志早在1977年就明确指出:“要引进外国教材,吸收外国教材中有益的东西。”随着我国加入WTO,信息产业的国际竞争将日趋激烈,我们必须尽快培养出大批具有国际竞争能力的高水平信息技术人才。教材是一个很关键的问题,国外的一些优秀教材不但内容新,而且还提供了很多新的研究方法和思考方式。引进国外原版教材,可以促进我国教学水平的提高,提高学生的英语水平和学习能力,保证我们培养出的学生具有国际水准。

为了贯彻中央“科教兴国”的方针,配合国内高等教育教材建设的需要,人民邮电出版社约请有关专家反复论证,与国外知名的教材出版公司合作,陆续引进一些信息科学与技术优秀教材。第一批教材针对计算机专业的主干核心课程,是国外著名高等院校所采用的教材,教材的作者都是在相关领域享有盛名的专家教授。这些教材内容新,反映了计算机科学技术的最新发展,对全面提高我国信息科学与技术的教学水平必将起到巨大的推动作用。

出版国外著名高等院校信息科学与技术优秀教材的工作将是一个长期的、坚持不懈的过程,我社网站(www.ptpress.com.cn)上介绍了我们陆续推出的图书的详细情况,敬请关注。希望广大教师和学生将使用中的意见和建议及时反馈给我们,我们将根据您的反馈不断改进我们的工作,推出更多更好的引进版信息科学与技术教材。

人民邮电出版社
2001年12月

序 言

操作系统是计算机系统最基本的系统软件，它管理着系统内的各种软件、硬件资源，为用户提供一个方便、有效、安全、可靠地使用计算机的工作环境。

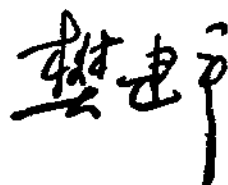
本书是一本操作系统课程的教材，是美国科罗拉多大学计算机系教授 Gary Nutt 撰写的教材《操作系统：现代观点》的第二版之实验更新版，是作者在长期的操作系统教学中逐渐形成和完善的。这本教材阐述了操作系统的基本概念、工作原理和实践技术，尤其在概念的介绍上十分透彻深入。本书十分注意理论和实践的结合，培养学生分析和设计操作系统的的能力，提供了大量的 UNIX、Linux、Windows 的实现代码、算法、实现中的问题和课程练习题。

本书作为教材，提供了 4 种非常有特点的内容：第一，分析练习题，每章后面都提供了学生练习题，用于激发学生对操作系统原理的理解和思考；第二，性能分析部分，用于解释操作系统设计者使用的提高操作系统性能的基本方法，这些通用的、非正式、非理论化的解释加深了学生对概念的理解；第三，代码实例，这些来自实际操作系统的代码实例帮助学生理解操作系统的理论是如何实现的；第四，实习题，每个实习题提出一个问题，同时提供一些相关背景信息，并建议解决问题的方法和计划，学生可以将它们用作课程实习，从中获得如何使用 Windows 和 UNIX 的细节和经验。

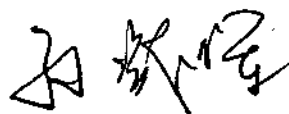
本书共分十八章。第一至第四章是介绍性内容，是学习操作系统的基础，其中第一章介绍操作系统是什么和操作系统是如何发展到今天的现状的；第二章介绍如何使用操作系统，特别是多进程操作系统；第三章介绍操作系统的基本组织和实现策略；第四章介绍计算机的组织。从第五章开始介绍了操作系统的各个部分，首先在第五章介绍了设备管理，特别是基本技术、缓冲区和设备驱动。第六章至第十章介绍进程管理，其中第六章介绍基本的任务概念、进程组织和资源管理；第七章介绍进程调度；第八章介绍基本的同步方法；第九章介绍高层的同步方法；第十章介绍死锁问题。在第十一章介绍内存管理，第十二章介绍虚拟内存管理，第十三章介绍文件管理，第十四章介绍保护机制和安全策略。第十五章至十七章介绍了支持分布式计算的操作系统技术，其中第十五章介绍网络管理；第十六章介绍远程文件管理；第十七章介绍分布计算，包括分布式进程管理、消息

传递机制、远程过程调用和分布式内存管理。最后在第十八章介绍了一些操作系统实例，包括 UNIX、Linux、Windows NT、Mac 和 Chorus 微内核操作系统。

本书是作为计算机科学和工程类专业教材编写的，也可供有关科技人员参考。为了获得比较满意的教学效果，建议教师贯彻精讲多练的原则，教师讲透概念、原理和方法，学生多对照实际系统的实现，加强书中的分析练习和实习题的实践。尤其是计算机专业的学生应加强课程实习，以做到融会贯通。



中国科学院计算技术研究所
副所长



中国科学院计算技术研究所
高性能计算机研究室主任

To my wife and best friend, Mary, and my
grandson and best buddy, Scott

Preface

To the Student

Operating systems is an exciting software area because the design of an operating system (OS) exerts a major influence on the overall function and performance of the entire computer. When studying operating systems for the first time, I believe that it is important to understand the *principles* behind the designs of all operating systems, and also 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 OS principles, supplemented with code, algorithms, implementation issues, and lab exercises to provide you with an understanding of contemporary OS practice. I have attempted to differentiate the conceptual material from the applied material by discussing the principles in the main flow of the text, and placing much of the practice material in supplemental discussions and lab exercises.

The heart of the matter is the conceptual material. Many OS principles can be described in formal (mathematical) terms or in informal discussion. Informal descriptions are relatively easy to read, but formal descriptions are more precise. For example, an informal discussion of a dictionary might explain that it is “a list of terms with their definitions,” whereas a formal description might indicate that a dictionary is “a mechanism, f , to map a term, x , to its definition, $f(x)$.” The first explanation is intuitive, the second focuses on the

Features at a Glance

This book provides a comprehensive description of OS principles, supplemented with the following:

- *Analytic exercises* to stimulate thinking about OS principles.
- *In the Hangar* sections to show how the principles are applied in practice in the UNIX family and Windows operating systems.
- *Performance Tuning* sections to explain how system designers have exploited the basic principles to achieve higher performance than could otherwise be achieved.
- *Laboratory exercises* to allow students to gain hands-on experience with the details of how to use Windows and UNIX. Each lab exercise begins with a problem statement, followed by a Background section, and an Attacking the Problem section. The Background section is a detailed discussion of information needed to gain comprehensive understanding of the problem and to create a design for the solution. The Attacking the Problem section provides specific guidance for solving the problem. In early lab exercises, the background and solution design discussions are more comprehensive than they are in later exercises. This allows students to get a significant amount of help in solving the early exercises but requires them to develop their design skills with each subsequent exercise.

logical intent of the dictionary. The first description suggests a list or table implementation, the second admits implementations ranging from tables, to lists, to associative memories, to databases, to network servers, and so on. The informal definition connotes word dictionaries, but the formal definition applies just as well to compiler symbol tables. My goal is to explain general OS principles so you will have a deep understanding of how an OS is designed. This goal is best supported using formal descriptions because they focus on the logical intent of the concept rather than on an example of how the concept is implemented. This has motivated me to describe OS concepts using informal or specific descriptions in the early chapters, but with increasing amounts of formal discussion as you progress through the book. In Chapter 7 you will see some formal discussion about scheduling coupled with the informal discussion, then more formal discussion about deadlock in Chapter 10, and even more in the discussion of virtual memory in Chapter 12. The formal discussion of concepts is always accompanied by informal discussion and examples.

Operating systems are designed around performance issues. However, detailed discussions of performance tend to obscure the concepts. In this case I decided to forgo extensive coverage of analysis and performance theory in favor of a generally informal explanation of performance issues. This will encourage you to develop your intuition on performance issues so that you can study them formally later. If the comments about performance fit naturally into the description of the concept, I have included them with the discussion of the concept. However, in those cases where the performance issues add a level of complication to understanding the principle, I have separated that discussion from the conceptual material by placing it in a distinct *Performance Tuning* section.

As I mentioned earlier, experimentation with real OS code provides you with an understanding of how OS concepts are implemented in real systems. I have also provided two other types of material to help you learn about current OS practice: In the Hangar examples and Laboratory Exercises.

7.3 ■ NONPREEMPTIVE STRATEGIES 203

PERFORMANCE TUNING

Predicting Wait Times for FCFS

It is not difficult to predict analytically a process's wait time under FCFS scheduling. Suppose we know the service rate, μ . Let L be the length of the queue at the time process p arrives. We can then estimate the time that the new process, p , will have to wait before it begins to receive service:

$$W(p) = L(1/\mu) + 1/2(1/\mu)$$

$$= L/\mu + 1/(2\mu).$$

Here is the rationale for this expression: If each job in the queue uses an average of $1/\mu$ time units for service, $L(1/\mu)$ will be the amount of time for

all L of them to be processed. The average time for the process that is already using the CPU is half of its service time, or $1/2(1/\mu)$. According to the FCFS policy, only the load that is present when process p arrives is relevant, since any subsequent processes will be served after process p .

In the example, we could estimate $W(p_i)$, which is 1200 in the Gantt chart, by computing the average service time ($1/\lambda$ or τ) of the first four processes:

$$\tau = 350 + 125 + 475 + 250)/4$$

$$= 1200/4$$

$$= 300 \text{ time units.}$$

When p_i arrives, $L = 3$; the estimated waiting time for p_i is thus

$$W(p_i) = L/\mu + 1/(2\mu)$$

$$= 3/(1/300) + 150$$

$$= 1050.$$

Notice that the estimate assumes half of a job has already executed. (We did not assume this in the Gantt chart.)

times will never be served. This total starvation of large processes may be a serious liability of the scheduling algorithm.

Again suppose the ready list contains the processes shown in Table 7.1. The arrival order is irrelevant here: provided all of the processes are already in the queue at the time

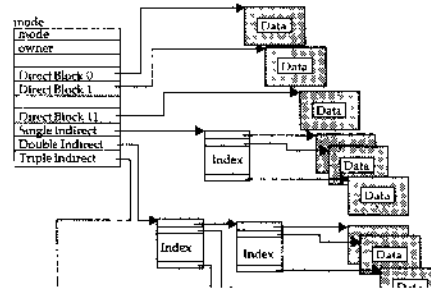
- In the *Hangar* examples explain how concepts are used or implemented in UNIX, Linux, Windows, or other operating systems. Many of the In the Hangar sections are code examples. The intent of including code examples is to provide you with insight into how an OS can implement the theory. A few of these 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 abbreviated descriptions deliberately omit fine 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 have experimented with using pseudocode 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.

- The book also includes several Laboratory Exercises; each exercise poses a problem then provides you with comprehensive background information needed to solve the problem, and a section to help you plan your solution. These exercises will give you valuable practice with various UNIX and Windows operating systems. Your instructor may have chosen for your course to go into more detail

IN THE HANGAR

UNIX File Structure

The UNIX file structure uses a variant of the indexed allocation scheme. The storage device detail part of the inode contains pointers to 15 different storage blocks of 4KB each (see Figure 13.13). The first 12 blocks of the file are indexed directly from the first 12 of 15 pointers in the inode. The last three pointers are used for indirect pointers through *index blocks*. If the file manager is configured with 4KB blocks, the 12 direct pointers in the inode accommodate files up to 48KB. Experience indicates this is an efficient mechanism for addressing the blocks (see [Ousterhout et al. 1985]). If a file requires more than 12 blocks, the file system



Lab Exercise: Observing OS Behavior

This exercise can be solved on Solaris and Linux systems.

In this chapter you have learned that the OS is a program that uses various data structures. Like all programs in execution, you can determine the performance and other behavior of the OS by inspecting its state—the values stored in its data structures. The goal of this exercise is to study some aspects of the organization and behavior of a Linux system by observing values in kernel data structures.

Write a program to report the behavior of the Linux kernel. Your program should have three different options: the default version should print the following values on `* * *`:

- CPU type and model
- kernel version
- Amount of time since the system was last booted

A second version of the program should print the same information as the first version, plus:

- The amount of time the CPU has spent in user mode, system mode, and the amount of time the system was idle
- The number of disk requests made on the system
- The number of context switches the kernel has performed
- The time at which the system was last booted
- The number of processes that have been created since the system was booted

The last version of the program should print the same information as the second version, plus (be sure to look at the relevant man pages for `/proc` to get more context for the requested information):

- The amount of memory configured into this computer
- The amount of memory currently available

with UNIX systems using the companion book on Linux internals [Nutt, 2001]. Alternatively, there is a Windows companion manual for Windows NT (that also applies to Windows 2000) [Nutt, 1999a].

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!

Topic Order

The order of presentation is based on the response to the first edition of the book, my experience teaching OS, and the input from many other instructors. This organization

thus reflects the combined knowledge and practice of many different teachers and I believe the result is logical, conducive to learning, and generally accepted by most OS instructors.

Each chapter begins with a transition from the previous chapter and a preview of what is covered in the current chapter. You 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 through 4 consist of important introductory material that provide 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 prerequisite courses. However, understanding this material is critical before you dive 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, a historical perspective had been included; instructors tend to like a little history and context, but many students think it is boring, so I have dispensed with a separate discussion of 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

Changes in the Second Edition

This edition is based on the constructive criticism I have received from people who used and/or reviewed the first edition. The goal in the previous edition was to separate the hands-on material from the principles; the comments I received indicated that instructors preferred better integration of simple examples. I have dropped the In the Cockpit examples that were used in the first edition, and greatly reduced the number of In the Hangar and Performance Tuning examples. Most of the material that appeared in the omitted examples has now been incorporated into the main text. Since the frequent examples in the first edition often made the mainline text difficult to follow, the book has been redesigned so that the remaining In the Hangar examples and Performance Tuning discussions are more easily distinguished from the main text.

Chapters 2 and 6 were reorganized and revised. The intent of Chapter 2 is for students to focus on *using* processes and resources, especially for concurrent applications. Chapter 6 is the cornerstone of the process management design discussion. The rewritten chapters now have better focus than they did in the first edition.

The most significant content change in the second edition is the addition of the laboratory exercises. A shortcoming of the first edition (and other conceptual OS textbooks) is the lack of material to support experimental lab exercises. This often forces students to purchase a second book to use with the laboratory component of the OS course. The second edition is a self-contained book including material for lectures on OS concepts and for recitations on OS practice.

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 review. 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, in the main part of the chapter I have resisted this temptation to focus instead on a macrolevel view of the purpose and general organization of interrupt-driven I/O. Included are extensive discussions of 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 through 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. The laboratory exercise provides a means for taking a closer look at the details of file 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 through 17 introduce OS 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.

To the Instructor

Operating systems continues to be an essential computer science course, yet as I have taught it over many years, I became increasingly dissatisfied with the OS texts that were available. I sought a book that had more content on principles than the existing ones. At the same time, I felt that if my students were not exposed to extensive lab practices, the principles would be difficult to absorb. The first edition described OS principles at a level I felt was necessary, and now this edition adds material to explicitly support the practice component that is so important to understanding operating systems.

The main thread of this book concentrates on OS concepts, usually illustrated with brief examples. The In the Hangar and Performance Tuning sections are more extensive examples or explanations of concepts, usually providing a practical perspective on the conceptual material that they follow. If you want your students to get an applied perspective of the OS, you should explicitly assign these supplementary sections as reading. The new Laboratory Exercises are accompanied by the applied material a student needs to solve the problem in a UNIX and/or Windows environment. The intent of including these exercises is to provide you with a single book that can be used to teach conceptual material as well as basic lab materials.

Many books begin with materials on process management. In my classes, I have found that it is necessary to provide background information of the type in Chapters 1 through 4. Specifically, my experience shows that it is really worth the time to lecture on the material in Chapter 2, since very few students have used `fork` and `exec` (or their analogs in non-UNIX systems) before they take an OS course. The Laboratory Exercises in Chapter 2 allows students to learn about basic concurrency concepts.

I start the detailed discussion of operating systems with device management. At first, you may find this approach unusual, 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 provides a sound foundation for introducing independent threads of execution (in the hardware and the software), concurrency, and synchronization. After you have finished the device material, it is natural to generalize these ideas into process and resource management, scheduling, synchronization, and deadlock.

Memory management is also important and another topic instructors usually want to address 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).

Any contemporary OS 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. Chapters 15 through 17 introduce distributed operating systems after all the discussion of traditional topics has been covered. Because of the nature of commercial systems and networks, an instructor would be remiss to completely ignore these topics in an OS course. In a one-semester course, I spend two to three weeks on this material.

Finally, in spite of all logical intentions 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 individual desires.

Today, there is a wealth of information on operating systems available on the Internet through ftp sites and on the World Wide Web. I would encourage you to point your students toward them. Because such sites change so frequently, I maintain 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 (email me at osamp@cs.colorado.edu) and I will add it to the page. I also welcome your questions, comments, suggestions, and advice (and I will even try to accept your criticism in good humor :-)).

About the Laboratory Environment

There are only a handful of widely used commercial operating systems. While studying these systems is valuable, there are practical barriers to experimenting with any of them in the classroom. First, commercial operating systems are by definition very complex since they must offer full support to commercial applications. It is impractical to experiment with such complex software because it is sometimes difficult to see how specific issues are addressed within the software. Small changes to the code may have unpredictable effects on the behavior of the overall operating system. Second, the OS software sometimes has distinct proprietary value to the company that implemented it. As a consequence, the company may be reluctant to provide OS source code to anyone wishing to study and learn how the implementation was done.

I have experimented with two approaches to this problem in the classroom [Nutt, 1999b]:

- Base the course on an *external view* of real operating systems; this is essentially the approach in the ACM/IEEE 1991 curriculum recommendation.
- Base the course on an *internal view* of some “manageable” OS.

I have also discussed this problem with numerous OS instructors (including participants at a Birds of a Feather session at the Operating Systems Design and Implementation meeting in New Orleans in February 1999). There is general confusion about choosing the right laboratory component for the undergraduate OS course. However, at the OSDI session, those in attendance unanimously agreed that the external view of an OS should be used in the first OS course.

This book provides materials to study the external view of Linux and Windows 2000. If you want to use Windows 2000 to teach the external view, the companion lab manual [Nutt, 1999a] provides more than enough exercises for a one-semester course. All of the lab exercises have students write user space code that allows them to get specific insight into the way the kernel works. The dependence on “crashable” lab facilities was my primary consideration in deciding not to include a device driver lab exercise.

While there is general consensus that teaching OS internals in the first course is too difficult, there is nevertheless a strong desire to offer an OS internals course as early as possible in the curriculum. If you decide to teach an internals course—as the first or

second course—your choices are limited: Linux or FreeBSD if you want to study a real OS, or one of the pedagogical systems otherwise. Another companion lab manual [Nutt, 2001] provides more than enough kernel internals for a semester course.

Acknowledgments

Many people have helped to edit and refine this book. First there are the students at the University of Colorado: Jason Casmira, Don Lindsay, Ann Root, and Sam Siewert were great teaching assistants who created laboratory exercises and solutions, and generally helped make the book better. Scott Brandt provided comments and insight into how the material should be presented. Adam Griff spent many hours helping me with my Linux system. Scott Morris set up my Windows NT machine and offered insider tips about how it worked.

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. There were many people who spent hours looking at drafts or otherwise suggesting ways to organize and improve the material: 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); Marty Humphrey (University of Virginia); 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 at Santa Cruz); Junsheng Long, Michael Lutz (Rochester Institute of Technology); Carol McNamee (Sacramento State University); Donald Miller (Arizona State University); Jim Mooney (West Virginia University); Ethan V. Munson (University of Wisconsin – Milwaukee); Deborah Nutt, Douglas Salane (John Jay College); Henning Schulzrinne (Columbia University); C. S. (James) Wong (San Francisco State University); and Salih Yurttas (Texas A&M University). The second edition was reviewed by Toby Berk (Florida International University); David Binger (Centre College); Richard Guy (UCLA); Zhiyuan Li (Purdue University); John Noll (University of Colorado, Denver); Kenneth A. Reek (Rochester Institutes of Technology); Joseph J. Pfeiffer, Jr. (New Mexico State University); and Irene Tseng (Gallaudet 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. In the first edition, Christine Kulke, Angela Buenning, Rebecca Johnson, Dusty Bernard, Laura Michaels, Pat Unubun, Dan Joraanstad, and Nate McFadden provided invaluable help and direction. Carter Shanklin, acquisition editor for the first edition, had a vision for how the book