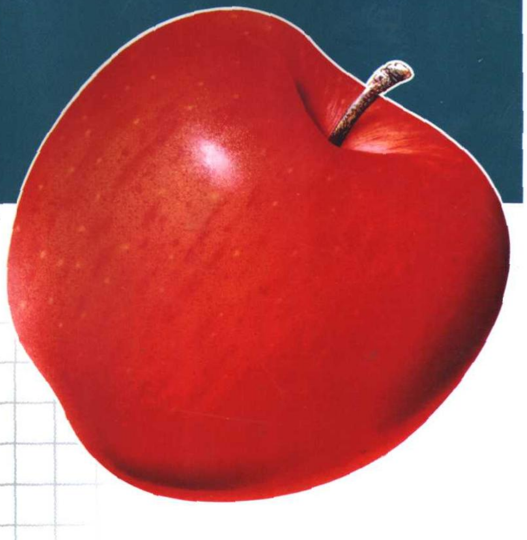
大学计算机教育国外著名教材系列（影印版）

PEARSON Prentice Hall

OPERATING SYSTEMS PRINCIPLES

操作系统原理



Lubomir F. Bic
Alan C. Shaw 著

Pearson Education

清华大学出版社

大学计算机教育国外著名教材系列（影印版）

Operating Systems Principles

操作系统原理

Lubomir F. Bic

University of California, Irvine

Alan C. Shaw

University of Washington, Seattle

清华大学出版社

北　京

图书在版编目(CIP)数据

操作系统原理 = Operating Systems Principles / 比奇（Bic, L. F.），肖（Shaw, A. C.）著. —影印本.
—北京：清华大学出版社，2004
（大学计算机教育国外著名教材系列）
ISBN 7-302-07724-X

Ⅰ. 操…　Ⅱ. ①比…　②肖…　Ⅲ. 操作系统－高等学校－教材－英文　Ⅳ. TP316

中国版本图书馆 CIP 数据核字（2003）第 111620 号

出 版 者：	清华大学出版社	**地　　址**：	北京清华大学学研大厦
	http://www.tup.com.cn	**邮　　编**：	100084
	社总机：（010）6277 0175	**客户服务**：	（010）6277 6969

责任编辑：周维焜
印 刷 者：清华大学印刷厂
装 订 者：三河市新茂装订有限公司
发 行 者：新华书店总店北京发行所
开　　本：185×230　**印张**：35.25
版　　次：2004 年 1 月第 1 版　2004 年 1 月第 1 次印刷
书　　号：ISBN 7-302-07724-X/TP・5652
印　　数：1~5000
定　　价：50.00 元

出 版 说 明

进入 21 世纪，世界各国的经济、科技以及综合国力的竞争将更加激烈。竞争的中心无疑是对人才的争夺。谁拥有大量高素质的人才，谁就能在竞争中取得优势。高等教育，作为培养高素质人才的事业，必然受到高度重视。目前我国高等教育的教材更新较慢，为了加快教材的更新频率，教育部正在大力促进我国高校采用国外原版教材。

清华大学出版社从 1996 年开始，与国外著名出版公司合作，影印出版了“大学计算机教育丛书（影印版）”等一系列引进图书，受到了国内读者的欢迎和支持。跨入 21 世纪，我们本着为我国高等教育教材建设服务的初衷，在已有的基础上，进一步扩大选题内容，改变图书开本尺寸，一如既往地请有关专家挑选适用于我国高校本科及研究生计算机教育的国外经典教材或著名教材，组成本套“大学计算机教育国外著名教材系列（影印版）”，以飨读者。深切期盼读者及时将使用本系列教材的效果和意见反馈给我们。更希望国内专家、教授积极向我们推荐国外计算机教育的优秀教材，以利我们把“大学计算机教育国外著名教材系列（影印版）”做得更好，更适合高校师生的需要。

清华大学出版社
2002 年 10 月

Preface

Operating systems bridge the gap between the hardware of a computer system and the user. Consequently, they are strongly influenced by hardware technology and architecture, both of which have advanced at a breathtaking pace since the first computers emerged in the 1940s. Many changes have been quantitative: the speed of processors, memories, and devices has been increasing continuously, whereas their size, cost, and power consumption have been decreasing. But many qualitative changes also have occurred. For example, personal computers with sophisticated input, output, and storage devices are now omnipresent; most also are connected to local area networks or the Internet. These advances have dramatically reshaped the world within which operating systems must exist and cooperate. Instead of managing a single processor controlling a collection of local memories and I/O devices, contemporary operating systems are required to manage highly parallel, distributed, and increasingly more heterogeneous configurations.

This book is an introduction to operating systems, appropriate for computer science or computer engineering majors at the junior or senior level. One objective is to respond to a major paradigm shift from single-processor to distributed and parallel computer systems, especially in a world where it is no longer possible to draw a clear line between operating systems for centralized environments and those for distributed ones. Although most of the book is devoted to traditional topics, we extend and integrate these with basic ideas in distributed computing.

The authors express their sincere appreciation to Gary Harkin, Montana State University; Mukkai Krisnimoorthy, Rensselaer Polytechnic Institute; Scott Cannon, Utah State University; John Hartman, University of Arizona; Gopal Lakhani, Texas Tech; Herb Mayer, Portland State University; and Chung Kuang-Shene, Michigan Technological University for their review of the book.

CONTENTS

After the introductory chapter, the book is organized into four main sections: Process Management and Coordination, Memory Management, File and I/O Management, and Protection and Security. At the end of each chapter, there is a list of the key concepts, terms, and abbreviations defined in the chapter; the back of the book contains a glossary.

Processes and Threads

Processes and, more recently, threads, are the basis of concurrency and parallelism, and have always been prominent parts of the study of operating systems. This area can be subdivided into two components: the creation of processes or threads, and their coordination. In Chapters 2 and 3, we treat the topic from the programming point of view, presenting a spectrum of constructs for expressing concurrency and for coordinating the execution of the resulting processes or threads. This includes the coordination of processes in a distributed environment, which must be based ultimately on message-passing rather than shared variables. In Chapters 4 and 5, we examine the problem from the implementation point of view by presenting the necessary data structures and operations to implement and manage processes and threads at the operating systems level. This discussion also includes issues of process and threads scheduling, interrupt handling, and other kernel functions. Chapter 6 is concerned with the important problem of deadlocks in both centralized and distributed systems.

Main Memory

Main memory has always been a scarce resource, and much of the past operating systems research has been devoted to its efficient use. Many of these results have become classical topics of operating systems; these are covered in Chapters 7, 8, and 9. Among these topics are techniques for physical memory allocation, implementation of virtual memory using paging or segmentation, and static and dynamic sharing of data and code. We also present the principles of distributed shared memory, which may be viewed as an extension of virtual memory over multiple computers interconnected by a communication network.

File Systems and I/O

Files were devised in the early days of computing as a convenient way to organize and store data on secondary storage devices. Although the devices have evolved dramatically, the basic principles of files have not. In Chapter 10, we discuss file types and their representations on disks or tapes. We also present ways of organizing and implementing file directories. In recent years, the most significant developments in the file systems area have been driven by the proliferation of networking. Many systems today do not maintain their own file systems on local drives. Instead, a more typical configuration is a network of machines, all accessing dedicated file servers. Frequently, the file systems are distributed over multiple servers or multiple networks. The last section of the chapter addresses file systems issues in such distributed environments.

Hiding the details of individual I/O devices by supporting higher-level abstractions has always been one of the main tasks of operating systems. Modern systems must continue to provide this essential service, but with a larger variety of faster and more sophisticated devices. Chapter 11 is devoted to this topic, presenting the principles of polling, interrupts, and DMA, as employed by various device drivers. Also discussed are device-independent aspects of I/O processing, including buffering and caching, error-handling, and device scheduling.

Protection and Security

Protecting a computing facility from various attacks requires a broad spectrum of safeguards. Chapter 12 focuses on the protection and security interface of the system, which guards the system access. This requires authentication of users, remote services, and clients. Despite many technological breakthroughs, user authentication still relies largely on passwords presented by users at the time of login. But the existence of computer networks has again stimulated the most dramatic developments in protection and security: the vulnerability of communication lines makes it necessary to employ techniques in secret or public key cryptography. We discuss the application of cryptographic methods both to protect information transmitted between computers and to verify its authenticity.

Once a user has entered the system, the system must control the set of resources accessible to that user. This is accomplished by hardware mechanisms at the instruction level and by access or capability lists at the software level. In addition, mechanisms to prevent unauthorized flow of information among different users also must be provided. Chapter 13 discusses such internal protection mechanisms.

EXERCISES AND PROGRAMMING PROJECTS

Each chapter ends with a set of exercises reflecting the presented topics. The exercises have been chosen carefully to satisfy the needs of different teaching styles. Each exercise set contains both analytical and constructive exercises, where students must apply conceptual knowledge acquired from the chapter to solve specific problems. We also have included questions that lend themselves to discussion or speculative analysis. A solutions manual is available to professors; they can obtain a copy from their local Prentice-Hall representative.

The set of five large programming projects and several smaller programming exercises at the end of the book are designed to complement the conceptual understanding gained from the book with practical hands-on experience. They may be used selectively as term projects or can serve as the basis for a separate laboratory component in operating systems.

APPROACH AND PHILOSOPHY

As expected, we provide in-depth coverage of all standard topics in the field of operating systems. A conventional approach typically also includes separate chapters on operating systems support for distributed network-based environments, usually appearing at the end of the text. The problem with this organization is that it makes an artificial distinction between centralized and distributed systems. In reality, there is often no clear demarcation line between the two, and they have many issues in common. Concurrency and parallelism have always been a major topic of operating systems. Even the earliest mainframes of the 1950s and 1960s attempted to overlap CPU execution with I/O processing to achieve better utilization of both. Advanced programming techniques of the 1970s and 1980s made it necessary to support concurrent processes at the user level, leading operating systems designers to provide new process synchronization and scheduling techniques, many of which also apply to networked environments. The last two decades have forced software manufacturers to seriously consider networking and physical distribution, and to integrate the necessary tools and techniques into their operating systems products.

We have chosen to preserve the natural relationship and overlap between centralized and distributed operating systems issues by integrating them within each chapter. The main distributed operating systems topics presented include message-based synchronization and remote procedure calls, distributed deadlocks, distributed shared memory, distributed file systems, and secure communication using cryptography.

Following the above philosophy, we also have refrained from presenting case studies of existing operating systems in separate chapters. Instead, we have distributed and integrated all case studies—from Unix, Linux, Windows, and many other influential operating systems—throughout the chapters. They illustrate the relevance of each concept at the time of its presentation.

Lubomir Bic

September 2002 Alan Shaw

Contents

CHAPTER 1

Introduction

We begin by examining the gap between the requirements and expectations placed on computer systems by the user community and the low-level capabilities of existing hardware. This gap is bridged by the operating system (OS) and other utility and support programs. We then outline the overall organization of OS, including interfaces to the hardware, the application programs, and the user. The remainder of the chapter traces the evolution of key OS concepts in the context of changing technology and the increasing diversity and sophistication of the user community.

1.1 THE ROLE OF OPERATING SYSTEMS

1.1.1 Bridging the Hardware/Application Gap

Most computer systems today are based on the principles of a "stored-program computer" formulated by mathematician John von Neumann and others in the late 1940s. The basic components of a computer and their interconnections are shown schematically in Figure 1-1 in the form of a high-level block diagram. At the heart of this system is the computational engine consisting of a **central processing unit (CPU)** and executable **main memory**. The memory is a linear sequence of directly addressable cells; it holds programs (lists of executable machine instructions) and data. The CPU continuously repeats the following basic hardware cycle:

- Fetch the instruction pointed to by a special register called the program counter.
- Increment the program counter.
- Decode the current instruction, held in a special instruction register, to determine what must be done.
- Fetch any operands referenced by the instruction.
- Execute the instruction.

This cycle forms the basis of all computations on present-day computers.

For this computational scheme to be of any practical value, two fundamental components must be included. The first is a set of **communication devices** to allow data and commands to be exchanged between the user and the machine or between one computer

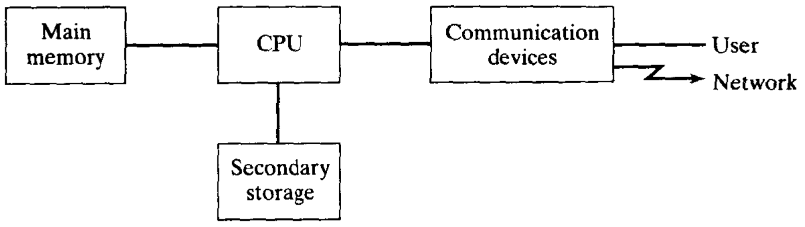


FIGURE 1-1. Main components of a computer system.

and another. It consists of **input/output (I/O) devices** (e.g., a keyboard and display terminal) and network interface devices. The second component is **secondary storage** to hold programs and data that are not currently loaded in main memory or that are only partially or temporarily loaded. This storage is needed because the system's main memory is volatile and thus loses its contents when power is turned off, and because it is also much smaller in size than secondary storage.

Sometimes the distinction between communication and storage devices is clear cut. For example, a CD-ROM drive is strictly an input device, whereas a hard disk is clearly a storage device. However, there are also common cases where such a distinction cannot be made easily. For example, a removable diskette can be viewed as storage, but it also can be used as an I/O device when moving information between different systems. From an operating system's perspective, CD-ROM, hard disk, diskette, and other devices are similar in nature, and many of the same techniques are employed to service them. We will refer to secondary storage and communication devices jointly as I/O devices.

Another degree of complexity is added when the computer system consists of more than one CPU. This can take several different forms, depending on the sharing level of the system's hardware components. Figure 1-2 shows three possible architectures that extend the basic single-CPU architecture of Figure 1-1 in different ways. In the first case (Fig. 1-2a), the two CPUs share a common main memory. The secondary storage and communication devices are typically shared. The presence of multiple CPUs poses new challenges for the OS. One of these is caching. If each CPU maintains its own local memory cache, the system must ensure that two caches do not contain different values for the same memory element. With a shared memory, this problem, referred to as **cache coherence**, is handled by the hardware and is transparent to the OS. Another important problem is the scheduling of processes. With a single CPU, scheduling is a matter of controlling the order in which processes execute. With multiple CPUs, the OS (or the application) also must decide on which CPU a given task should run. Synchronization and communication among processes running on different CPUs is performed through the shared memory; the approaches are similar to those for coordinating processes on a single CPU.

Figure 1-2b shows an architecture where each CPU has its own main memory. The secondary storage and other devices could still be shared. However, the communication subsystem must include an interconnection network that allows the CPUs to interact with each other, since no shared memory is available. There is a broad range of interconnection networks, ranging from a simple shared bus to dedicated connections arranged in a

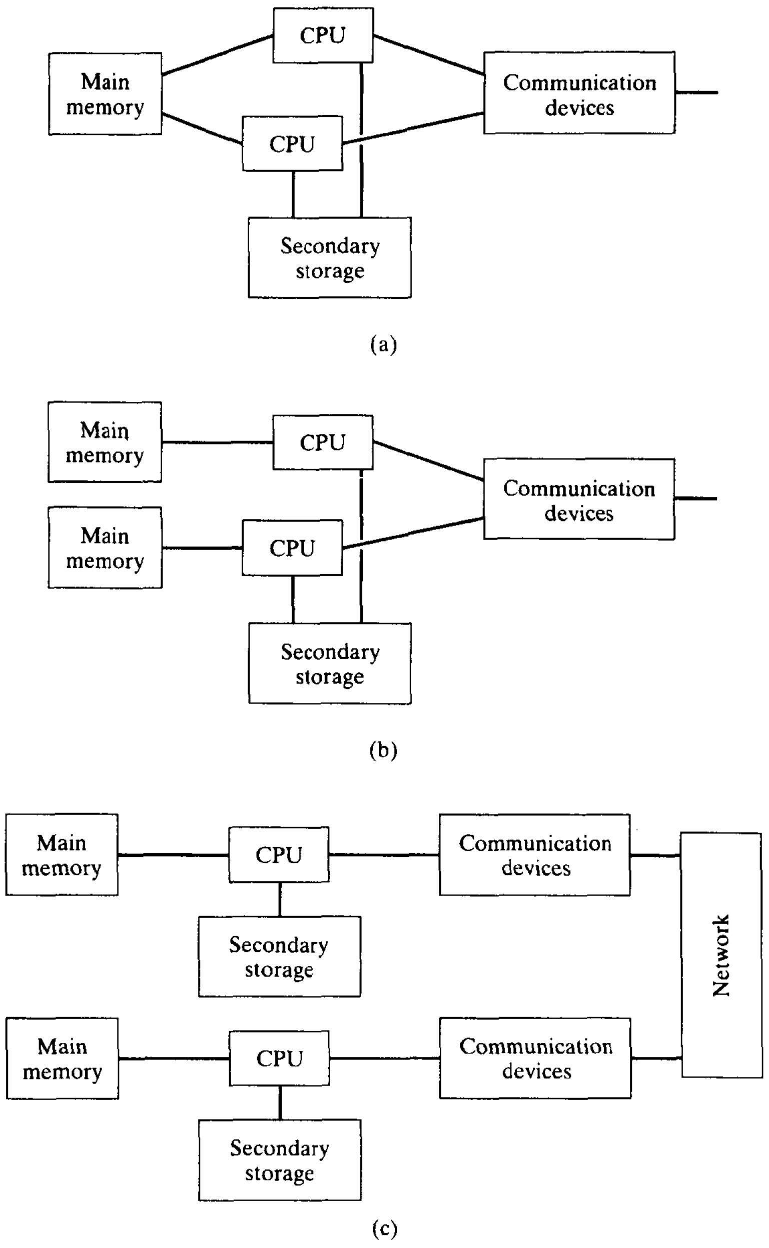


FIGURE 1-2. Systems with multiple CPUs: (a) Shared-memory multiprocessor; (b) Distributed-memory multiprocessor; and (c) Multicomputer.

variety of topologies. In the absence of shared memory, both scheduling and process coordination become more complicated. Scheduling involves not only the assignment of processes to different CPUs but also the assignment of data, some of which may be needed by multiple processes, to the disjoint local memory modules. Given that such