8th EDITION

understanding Operating Systems

Ann McIver McHoes
Ida M. Flynn

CENGAGE brain .com

Buy. Rent. Access.

Access student data files and other study tools on **cengagebrain.com**.

For detailed instructions visit
http://s-solutions.cengage.com/ctdownloads/

Store your Data Files on a USB drive for maximum efficiency in organizing and working with the files.

Macintosh users should use a program to expand WinZip or PKZip archives. Ask your instructor or lab coordinator for assistance.

Understanding Operating Systems

Eighth Edition

Ann McIver McHoes
Ida M. Flynn

CENGAGE Learning®

Australia • Canada • Mexico • Singapore • Spain • United Kingdom • United States

CENGAGE Learning®

Understanding Operating Systems, Eighth Edition
Ann McIver McHoes & Ida M. Flynn

Senior Product Manager: Kathleen McMahon

Product Team Leader: Kristin McNary

Associate Product Manager: Kate Mason

Associate Content Development Manager: Alyssa Pratt

Production Director: Patty Stephan

Senior Content Project Manager: Jennifer Feltri-George

Manufacturing Planner: Julio Esperas

Art Director/Cover Design: Diana Graham

Production Service/Composition: SPi Global

Cover Photos: sumkin/Shutterstock.com

Library of Congress Control Number: 2016962900

ISBN: 978-1-305-67425-7

Cengage Learning
20 Channel Center Street
Boston, MA 02210
USA

Cengage Learning is a leading provider of customized learning solutions with employees residing in nearly 40 different countries and sales in more than 125 countries around the world. Find your local representative at **www.cengage.com.**

Cengage Learning products are represented in Canada by Nelson Education, Ltd.

To learn more about Cengage Learning Solutions, visit **www.cengage.com**

Purchase any of our products at your local college store or at our preferred online store **www.cengagebrain.com**

Printed in the United States of America

Print Number: 01 Print Year: 2017

Dedicated to two inspiring colleagues:

Ida Moretti Flynn, award-winning teacher and a wonderful friend; her love for teaching lives on.

Bob Kleinmann, superb editor and soul mate – not in that order.

AMM

Contents

Preface

Is this book for you? In these pages, we explain a very technical subject in a not-so-technical manner, putting the concepts of operating systems into words that many readers can quickly grasp.

For those who are new to the subject, this text demonstrates what operating systems are, what they do, how they do it, how their performance can be evaluated, and how they compare with each other. Throughout the textbook we describe the overall function of many unseen parts of the operating system and lead readers to additional resources where they can find more detailed information, if they so desire.

For readers with more technical backgrounds, this text introduces the subject concisely, describing the complexities of operating systems without going into intricate detail. One might say this book leaves off where other operating system textbooks begin.

To do so, we've made some assumptions about our audiences. First, we assume the readers have some familiarity with computing systems. Second, we assume they have a working knowledge of how to use an operating system and how it interacts with them. We recommend (although we don't require) that readers be familiar with at least one operating system. In the few places where, in previous editions, we used pseudocode to illustrate the inner workings of the operating systems, that code can be found in the Appendix. By moving these algorithms out of individual chapters, we have simplified our explanations of some complex events.

Although it is more difficult to understand how operating systems work than to memorize the details of a single operating system, gaining this understanding is a longer-lasting achievement, paying off in the long run because it allows one to adapt as technology changes—as, inevitably, it does.

Therefore, regardless of the level of expertise that the reader brings to the subject, the purpose of this book is to give computer users a solid background in the basics of operating systems, their functions and goals, and how they interact and interrelate.

Structure and Features

The organization of this book addresses a recurring problem with textbooks about technologies that continue to change—constant advances in evolving subject matter can make textbooks immediately outdated. To address this problem, our material is divided into two parts: first, the concepts, which do not change quickly, and second, the specifics of operating systems, which change dramatically over the course of years and even months. Our goal is to give readers the ability to apply their knowledge year after year, realizing that, although a command, or series of commands, used by one operating system may be different from another, the goals are the same and the functions of competing operating systems are also the same. It is for that reason, that we have structured this book in two parts.

Part One (the first 12 chapters) describes the concepts of operating systems by concentrating on several "managers" in turn, and then describing how these managers work together. In addition, Part One introduces network organization concepts, security, ethics, and system management.

Part Two examines actual operating systems: how they apply the theories presented in Part One and how they compare with each other.

Chapter 1 gives a brief introduction to the subject. The Memory Manager, described in Chapters 2 and 3, is the simplest component of the operating system to explain, and has been historically tied to the advances from one operating system to the next. We explain the role of the Processor (CPU) Manager in Chapters 4, 5, and 6, first discussing simple systems and then expanding the topic to include multiprocessing systems. By the time we reach the Device Manager in Chapter 7 and the File Manager in Chapter 8, readers will have been introduced to many key concepts found in every operating system. Chapters 9 and 10 introduce basic concepts related to networking. Chapters 11 and 12 discuss security, ethics, and system management, including some of the tradeoffs that operating systems designers consider when attempting to satisfy the needs of their user population.

In Part Two we explore four operating systems in the order of their first release: UNIX, Windows, Linux, and Android. Here, each chapter includes a discussion describing how that operating system applies the concepts discussed in Part One. Again, we must stress that this is a general discussion—an in-depth examination of an operating system would require details based on its current standard version, which can't be done in a textbook. We strongly suggest that readers use our discussion as a guide—a base to work from—when comparing the advantages and disadvantages of a specific operating system, and supplement our work with current academic research, which is readily available online.

Each chapter includes learning objectives, key terms, research topics, exercises, and a spotlight on industry experts who have left their mark in computer science. For technically oriented readers, the exercises at the end of each chapter include some problems

for advanced students. Please note that these advanced exercises assume knowledge of matters not presented in the book, but they're good for anyone who enjoys a challenge. We expect some readers who are new to the subject will cheerfully pass them by.

The text concludes with several reference aids. Within each chapter, important terms are listed at its conclusion as key terms. The Windows chapter also includes a table that briefly lists all acronyms or abbreviations used in that chapter. An extensive end-of-book Glossary includes brief reader-friendly definitions for hundreds of terms used in these pages; note that this glossary is specific to the way these terms are used in this textbook. The Bibliography can guide the reader to basic research on the subject. Finally, the Appendix features pseudocode algorithms referenced in several chapters, and a section of the ACM Code of Ethics.

In an attempt to bring the concepts closer to home, throughout the book we've added real-life examples to illustrate abstract concepts. However, let no one confuse our conversational style with our considerable respect for the subject matter. The subject of operating systems is a complex one and it cannot be covered completely in these few pages. Therefore, in this textbook we do not attempt to give an in-depth treatise of operating systems theory and applications. This is an overall view.

Not included in this text is a detailed discussion of databases and data structures, except as they are used to resolve process synchronization problems, or the work of specific operating systems. This is because these structures only tangentially relate to operating systems and are frequently the subject of other courses. We suggest that readers begin by learning the basics as presented in the following pages and pursue these complex subjects in their future studies.

Changes to this Edition

This edition has been thoroughly updated and features many improvements over previous editions:

- Renewed emphasis on the role of the talented people who designed and wrote operating systems, as well as their design decisions, which can affect how the resulting system works.
- Added more screenshots from a variety of operating systems, including Macintosh OS (which runs UNIX), Windows, Android phone and tablet, and Linux.
- Expanded our discussions of cloud computing and cloud storage.
- Revised networking discussions to reflect emerging designs and technology.
- Retained an emphasis on student understanding and original thinking in the exercises, rather than on memorization or cut-and-paste facts. This is because our book's answer key is often available online shortly after publication, so in these pages we have routinely asked students to use their own words to explain concepts.

- Expanded cross-references from Part Two to the concepts taught in Part One to help students link specific system features with the concepts discussed in the beginning chapters.
- Added emphasis on available command-mode options in each operating system for readers who want to explore their system more directly, without using the menus.
- Included online resources for more information about many of the highly technical subjects introduced in this text. Please remember that in the field of computer science, online links go bad frequently, but by providing these links to our readers, they will have a good starting place from which they can search for more current info.
- Updated artwork and references to the expanding influence of wireless technology.
- Removed examples in assembly language, which is not widely studied in introductory classes, and replaced them with pseudocode and prose descriptions.

Numerous other changes throughout the text include editorial clarifications, expanded captions, and improved illustrations.

A Note for Instructors

The following supplements are available when this text is used in a classroom setting. All supplements can be downloaded from the Instructor Companion Site. Simply search for this text at sso.cengage.com. An instructor login is required.

Instructor's Manual. The Instructor's Manual that accompanies this textbook includes additional instructional material to assist in class preparation, including Sample Syllabi, Chapter Outlines, Technical Notes, Lecture Notes, Quick Quizzes, Teaching Tips, and Discussion Topics.

Test Bank. Cengage Testing Powered by Cognero is a flexible, online system that allows you to:

- author, edit, and manage test bank content from multiple Cengage solutions;
- create multiple test versions in an instant;
- deliver tests from your LMS, your classroom, or wherever you want.

PowerPoint Presentations. This book comes with Microsoft PowerPoint slides for each chapter. These are included as a teaching aid for classroom presentations, either to make available to students on the network for chapter review, or to be printed for classroom distribution. Instructors can add their own slides for additional topics that they wish to introduce to the class.

Solutions. Selected solutions to Exercises are provided.

Order of Presentation

We have built this text with a modular construction to accommodate several alternative sequences, depending on the instructor's preference.

- For example, the syllabus can follow the chapters as listed from Chapter 1 through Chapter 12 to present the core concepts that all operating systems have in common. Using this path, students will learn about the management of memory, processors, devices, files, and networks, in that order.
- An alternative path might begin with Chapter 1, move next to processor management in Chapters 4 through 6, then to memory management in Chapters 2 and 3, touch on systems security and management in Chapters 11 and 12, and finally move to device and file management in Chapters 7 and 8. Because networking is often the subject of another course, instructors may choose to bypass Chapters 9 and 10, or include them for a more thorough treatment of operating systems.

We hope you find our discussion of ethics helpful in Chapter 11, which is here in response to requests by university adopters of the text who asked us to include this subject, even though it is sometimes the subject of a separate course.

When teaching one or more operating systems from Part Two, keep in mind that we structured each of these four chapters the same way we presented concepts in the first 12 chapters. That is, they discuss the management of memory, processors, files, devices, networks, and systems, in that order, with a special section demonstrating the user interfaces for each operating system. To illustrate the use of graphical user interfaces in UNIX systems, we include screenshots from the Macintosh OS X operating system.

By including the Android operating system, which is specifically designed for use in a mobile environment using phones and tablets, we are able to explore the challenges unique to these computing situations.

Acknowledgments

Our gratitude goes to all of our friends and colleagues who were so generous with their encouragement, advice, and support over the two decades of this publication. Special thanks go to Bob Kleinmann, Eleanor Irwin, and Roger Flynn for their assistance.

As always, thanks to those at Cengage, Brooks/Cole, and PWS Publishing who have made significant contributions to all eight editions of this text, especially Alyssa Pratt, Kallie Swanson, Mike Sugarman, and Mary Thomas Stone.

And to the many students and instructors who have sent helpful comments and suggestions since publication of our first edition in 1991, we thank you. Please keep them coming.

Ann McIver McHoes, mchoesa@duq.edu

Ida Moretti Flynn (1945–2004)

Part One

Operating Systems Concepts

This text explores the core mechanisms of operating systems, which manage a computing system's hardware and software. That includes its memory, processing capability, devices, files, and networks—and how to do all of this in an appropriate and secure fashion. Here, in Part One, we present an overview of an operating system's essentials.

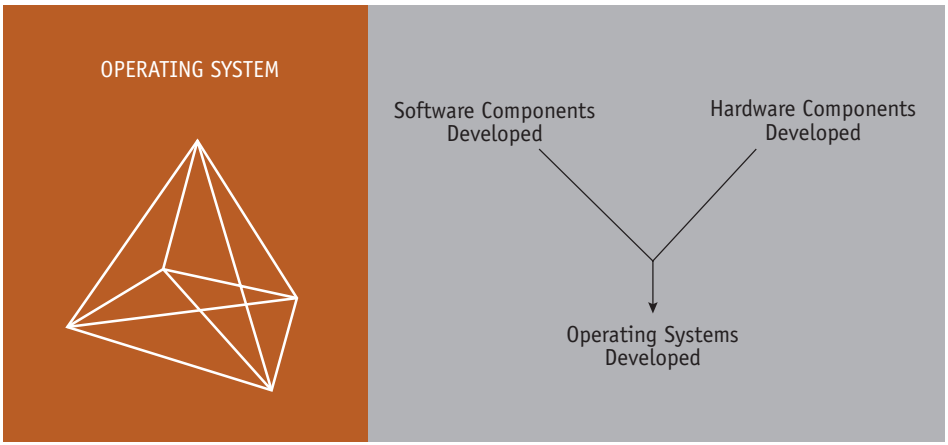
- Chapter 1 introduces the subject of operating systems.
- Chapters 2 and 3 discuss the management of main memory resources.
- Chapters 4 through 6 cover single processor and multiprocessor management.
- Chapter 7 concentrates on managing available devices without conflicts.
- Chapter 8 is devoted to the management of files, including those that hold system instructions as well as your data.
- Chapters 9 and 10 briefly review operating systems for networks.
- Chapter 11 discusses system security.
- Chapter 12 explores system management.

In Part Two (Chapters 13 through 16), we look at four specific operating systems and how they apply the overall concepts presented in the first 12 chapters.

Throughout our discussion of this very technical subject, we try to include definitions of terms that might be unfamiliar, but it isn't always possible to describe a function and define the technical terms while keeping the explanation clear. Therefore, we've put the key terms with definitions at the end of each chapter, as well as in the glossary at the end of the text. Items listed in the Key Terms are shown in **boldface** the first time they are mentioned significantly.

Throughout this book we keep our descriptions and examples as simple as possible to introduce the system's complexities without getting bogged down in technical detail. Therefore, remember that for almost every topic explained in the following pages, there's much more information that's readily available for study. Our goal is to introduce the subject and to encourage our readers to independently pursue topics of special interest. Enjoy.

Chapter 1 Introducing Operating Systems



"I think there is a world market for maybe five computers."

—Attributed to Thomas J. Watson (1874–1956; chairman of IBM 1949–1956)

Learning Objectives

After completing this chapter, you should be able to describe:

- How operating systems have evolved through the decades
- The basic role of an operating system
- How operating system software manages it subsystems
- The role of computer system hardware on the development of its operating system
- How operations systems are adapted to serve batch, interactive, real-time, hybrid, and embedded systems
- How operating systems designers envision their role and plan their work

To understand an **operating system** is to begin to understand the workings of an entire computer system, because the operating system software manages each and every piece of hardware and software. In the pages that follow, we explore what operating systems are, how they work, what they do, and why.

This chapter briefly describes the workings of operating systems on the simplest scale. The following chapters explore each component in more depth, and show how its function relates to the other parts of the operating system. In other words, we see how the pieces work together harmoniously to keep the computer system working smoothly.

What Is an Operating System?

A computer system typically consists of software (programs) and hardware (the tangible machine and its electronic components). The operating system is the most important software— it's the portion of the computing system that manages all of the hardware and all of the other software. To be specific, the operating system software controls every file, every device, every section of main memory, and every moment of processing time. It controls who can use the system and how. In short, the operating system is the boss.

Therefore, each time the user sends a command, the operating system must make sure that the command is executed; or, if it's not executed, it must arrange for the user to get a message explaining the error. This doesn't necessarily mean that the operating system executes the command or sends the error message, but it does control the parts of the system that do.

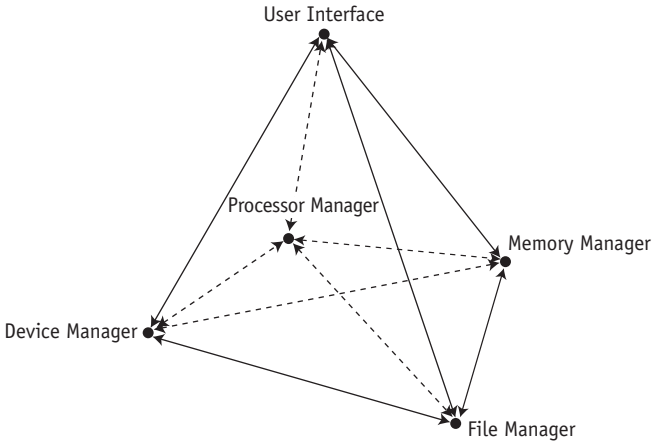
Operating System Software

The pyramid shown in Figure 1.1 is an abstract representation of the operating system in its simplest form, and demonstrates how its major components typically work together.

At the base of the pyramid are the four essential managers of every major operating system: **Memory Manager**, **Processor Manager**, **Device Manager**, and **File Manager**. These managers, and their interactions, are discussed in detail in Chapters 1 through 8 of this book. Each manager works closely with the other managers as each one performs its unique role. At the top of the pyramid is the User Interface, which allows the user to issue commands to the operating system. Because this component has specific elements, in both form and function, it is often very different from one operating system to the next—sometimes even between different versions of the same operating system.

(figure 1.1)

This pyramid represents an operating system on a stand-alone computer unconnected to a network. It shows the four subsystem managers and the User Interface.

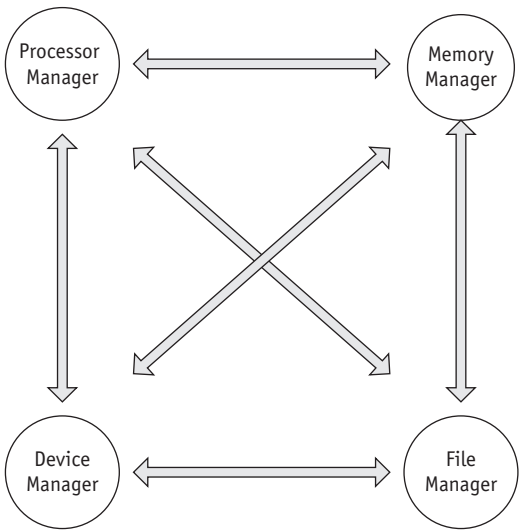


Regardless of the size or configuration of the system, the four managers, illustrated in Figure 1.2, must, at a minimum, perform the following tasks while collectively keeping the system working smoothly:

- Monitor the system's resources
- Enforce the policies that determine what component gets what resources, when, and how much
- Allocate the resources when appropriate
- Deallocate the resources when appropriate

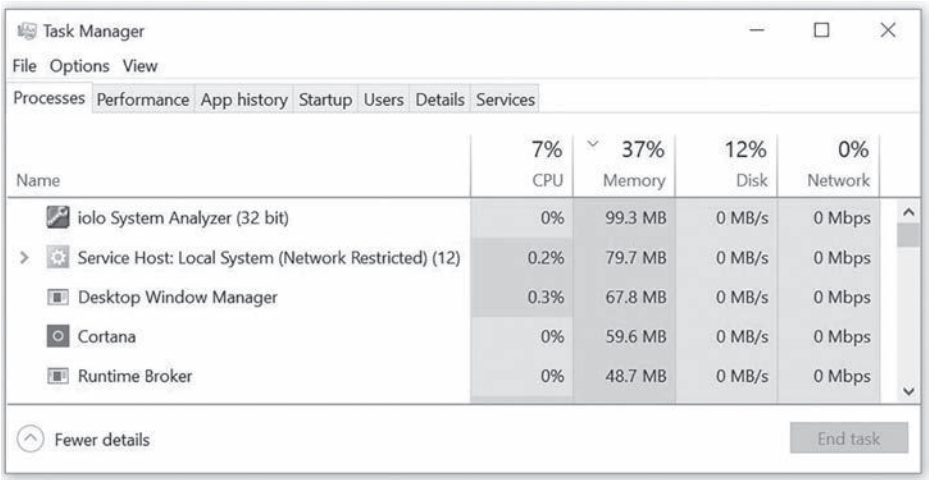
(figure 1.2)

Each manager at the base of the pyramid takes responsibility for its own tasks while working harmoniously with every other manager.



For example, the Memory Manager must keep track of the status of the computer system's main memory space, allocate the correct amount of it to incoming processes, and deallocate that space when appropriate—all while enforcing the policies that were established by the designers of the operating system.

An additional management task, networking, has not always been an integral part of operating systems. Today the vast majority of major operating systems incorporate a **Network Manager**, see Figure 1.3, to coordinate the services required for multiple systems to work cohesively together. For example, the Network Manager must coordinate the workings of the networked resources, which might include shared access to memory space, processors, printers, databases, monitors, applications, and more. This can be a complex balancing act as the number of resources increases, as it often does.



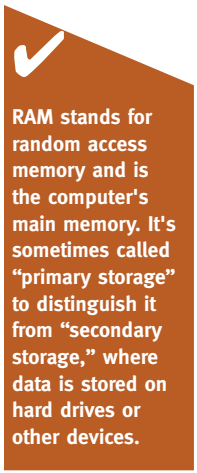
(figure 1.3)

The Windows 10 Task Manager displays a snapshot of the system's CPU, main memory, disk, and network activity.

Main Memory Management

The Memory Manager (the subject of Chapters 2 and 3) is in charge of main memory, widely known as **RAM** (short for random access memory). The Memory Manager checks the validity of each request for memory space, and if it is a legal request, allocates a portion of memory that isn't already in use. If the memory space becomes fragmented, this manager might use policies established by the operating system's designers to reallocate memory to make more useable space available for other jobs that are waiting. Finally, when the job or process is finished, the Memory Manager deallocates its allotted memory space.

A key feature of RAM chips—the hardware that comprises computer memory—is that they depend on the constant flow of electricity to hold data. If the power fails or is turned off, the contents of RAM is wiped clean. This is one reason why computer system

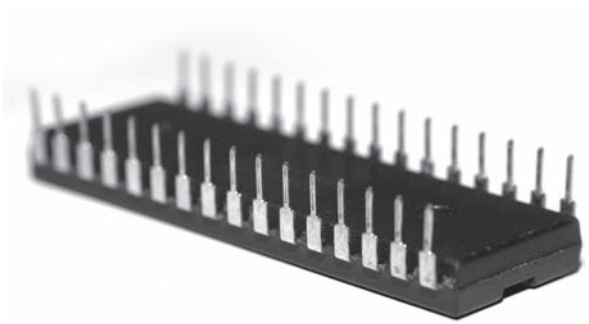


RAM stands for random access memory and is the computer's main memory. It's sometimes called "primary storage" to distinguish it from "secondary storage," where data is stored on hard drives or other devices.

designers attempt to build elegant shutdown procedures, so that the contents of RAM can be stored on a nonvolatile device, such as a hard drive, before the main memory chips lose power during computer shutdown.

A critical responsibility of the Memory Manager is to protect all of the space in main memory, particularly the space occupied by the operating system itself—it can't allow any part of the operating system to be accidentally or intentionally altered because that would lead to instability or a system crash.

Another kind of memory that's critical when the computer is powered on is read-only memory (often shortened to ROM), shown in Figure 1.4. This ROM chip holds software called **firmware**: the programming code that is used to start the computer and perform other necessary tasks. To put it in simplest form, it describes, in prescribed steps, when and how to load each piece of the operating system after the power is turned on, up to the point that the computer is ready for use. The contents of the ROM chip are nonvolatile, meaning that they are not erased when the power is turned off, unlike the contents of RAM.



(figure 1.4)

A computer's relatively small ROM chip contains the firmware (unchanging software) that prescribes the system's initialization every time the system's power is turned on.

Processor Management

The Processor Manager (discussed in Chapters 4 through 6) decides how to allocate the central processing unit (CPU); an important function of the Processor Manager is to keep track of the status of each job, process, thread, and so on. We will discuss all of these in the chapters that follow, but for this overview, let's limit our discussion to a **process** and define it as a program's "instance of execution." A simple example could be a request to solve a mathematical equation: This would be a single job consisting of several processes, with each process performing a part of the overall equation.

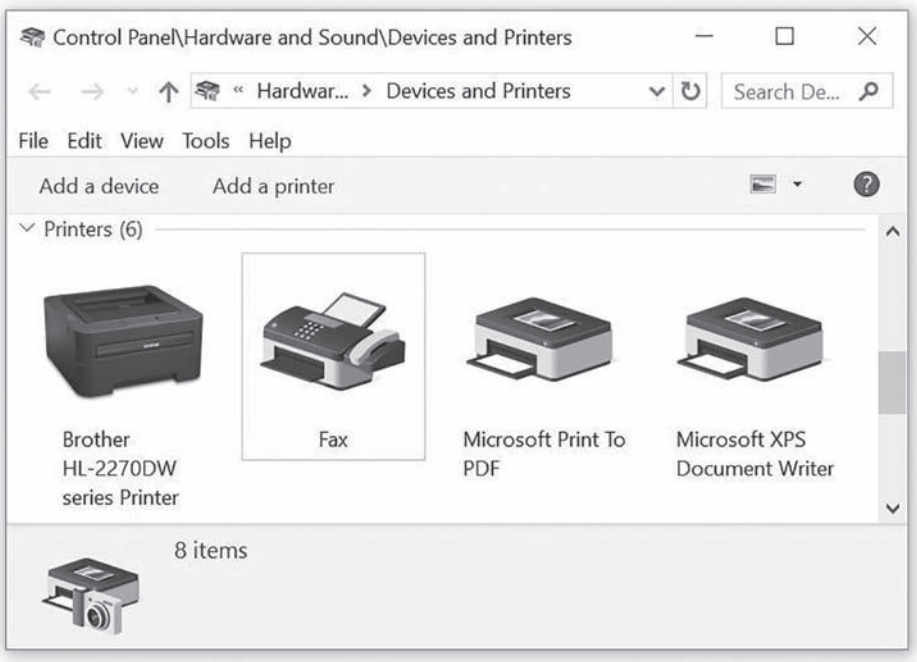
The Processor Manager is required to monitor the computer's CPU to see if it's busy executing a process or sitting idle as it waits for some other command to finish execution. Generally, systems are more efficient when their CPUs are kept busy. The Processor

Manager handles each process's transition, from one state of execution to another, as it moves from the starting queue, through the running state, and, finally, to the finish line (where it then tends to the next process). Therefore, this manager can be compared to a traffic controller. When the process is finished, or when the maximum amount of computation time has expired, the Processor Manager reclaims the CPU so it can allocate it to the next waiting process. If the computer has multiple CPUs, as with a multicore system, the Process Manager's responsibilities are greatly complicated.

Device Management

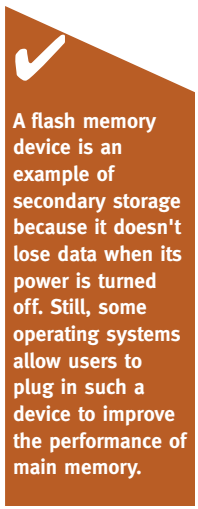
The Device Manager (the subject of Chapter 7) is responsible for connecting with every device that's available on the system, and for choosing the most efficient way to allocate each of these printers, ports, disk drives, and more, based on the device scheduling policies selected by the designers of the operating system.

Good device management requires that this part of the operating system uniquely identify each device, start its operation when appropriate, monitor its progress, and, finally, deallocate the device to make the operating system available to the next waiting process. This isn't as easy as it sounds because of the exceptionally wide range of devices that can be attached to any system, such as the system shown in Figure 1.5.



(figure 1.5)

This computer, running the Windows 10 operating system, has device drivers loaded for all of the printing devices shown here.



A flash memory device is an example of secondary storage because it doesn't lose data when its power is turned off. Still, some operating systems allow users to plug in such a device to improve the performance of main memory.

For example, let's say you're adding a printer to your system. There are several kinds of printers commonly available (laser, inkjet, inkless thermal, etc.) and they're made by manufacturers that number in the hundreds or thousands. To complicate things, some devices can be shared, while some can be used by only one user or one job at a time. Designing an operating system to manage such a wide range of printers (as well as monitors, keyboards, pointing devices, disk drives, cameras, scanners, and so on) is a daunting task. To do so, each device has its own software, called a device driver, which contains the detailed instructions required by the operating system to start that device, allocate it to a job, use the device correctly, and deallocate it when it's appropriate.

File Management

The File Manager (described in Chapter 8), keeps track of every file in the system, including data files, program files, utilities, compilers, applications, and so on. By following the access policies determined by the system designers, the File Manager enforces restrictions on who has access to which files. Many operating systems allow authorized individuals to change these permissions and restrictions. The File Manager also controls the range of actions that each user is allowed to perform on the files after they access them. For example, one user might have read-only access to a critical database, while the systems administrator might hold read-and-write access with the authority to create and delete files in the same database. Access control is a key part of good file management and is tightly coupled with system security software.

When the File Manager allocates space on a secondary **storage** device, such as a hard drive, flash drive, archival device, and so on, it must do so knowing the technical requirements of that device. For example, if it needs to store an archival copy of a large file, it needs to know if the device stores it more efficiently as one large block or in several smaller pieces that are linked through an index. This information is also necessary for the file to be correctly retrieved later. Later, if this large file must be modified after it has been stored, the File Manager must be capable of making those modifications as accurately and efficiently as possible.

Network Management

Operating systems with networking capability have a fifth essential manager called the Network Manager (the subject of Chapters 9 and 10) that provides a convenient way for authorized users to share resources. To do so, this manager must take overall responsibility for every aspect of network connectivity, including the requirements of the available devices as well as files, memory space, CPU capacity, transmission connections, and types of encryption (if necessary). Networks with many available resources require management of a vast range of alternative elements, which enormously complicates the tasks required to add network management capabilities.

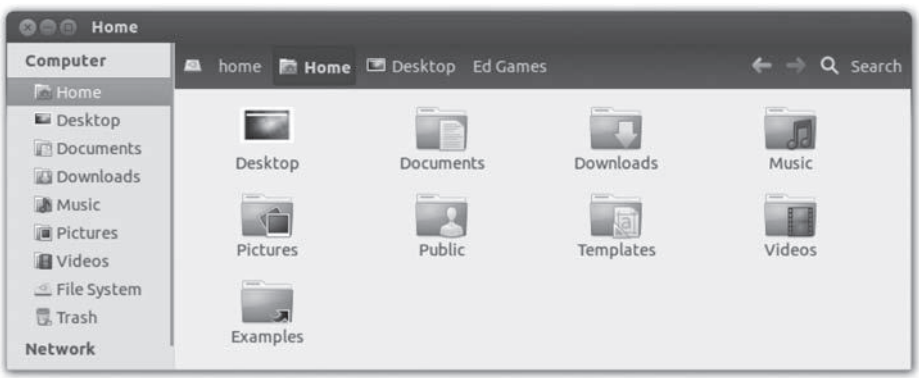
Networks can range from a small wireless system that connects a game system to the Internet; to a private network for a small business; to one that connects multiple computer systems, devices, and mobile phones to the Internet. Regardless of the size and complexity of the network, these operating systems must be prepared to properly manage the available memory, CPUs, devices, and files.

User Interface

The **user interface**—the portion of the operating system that users interact with directly—is one of the most unique and most recognizable components of an operating system. Two primary types are the graphical user interface (**GUI**), shown in Figure 1.6, and the command line interface. The GUI relies on input from a pointing device, such as a mouse or the touch of your finger. Specific menu options, desktops, and formats often vary widely from one operating system to another, and, sometimes, from one version to another.

The alternative to a GUI is a command line interface, which responds to specific commands typed on a keyboard and displayed on the monitor, as shown in Figure 1.7. These interfaces accept typed commands, and offer skilled users powerful additional control because, typically, the commands can be linked together (concatenated) to perform complex tasks with a single multifunctional command that would require many mouse clicks to duplicate using a graphical interface.

While a command structure offers powerful functionality, it has strict requirements for every command: Each must be typed accurately, each must be formed in the correct syntax, and combinations of commands must be assembled correctly. In addition, users need to know how to recover gracefully from any errors they encounter. These command line interfaces were once standard for operating systems and are still favored by power users, but have largely been supplemented with simple, forgiving, graphical user interfaces.

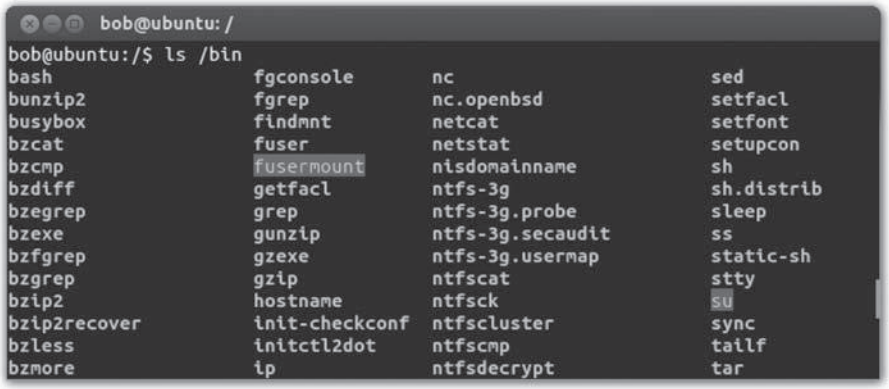


(figure 1.6)

An example of the graphical user interface (GUI) for the Ubuntu Linux operating system.

(figure 1.7)

This is the Linux command line user interface showing a partial list of valid commands for this operating system. Many menu-driven operating systems also support a command-line interface similar to this one.



Cooperation Issues

None of the elements of an operating system can perform its individual tasks in isolation—each must also work harmoniously with every other manager. To illustrate this using a very simplified example, let's follow the steps as someone chooses a menu option to open a program. The following series of major steps are typical of the discrete actions that would occur in fractions of a second as a result of this choice:

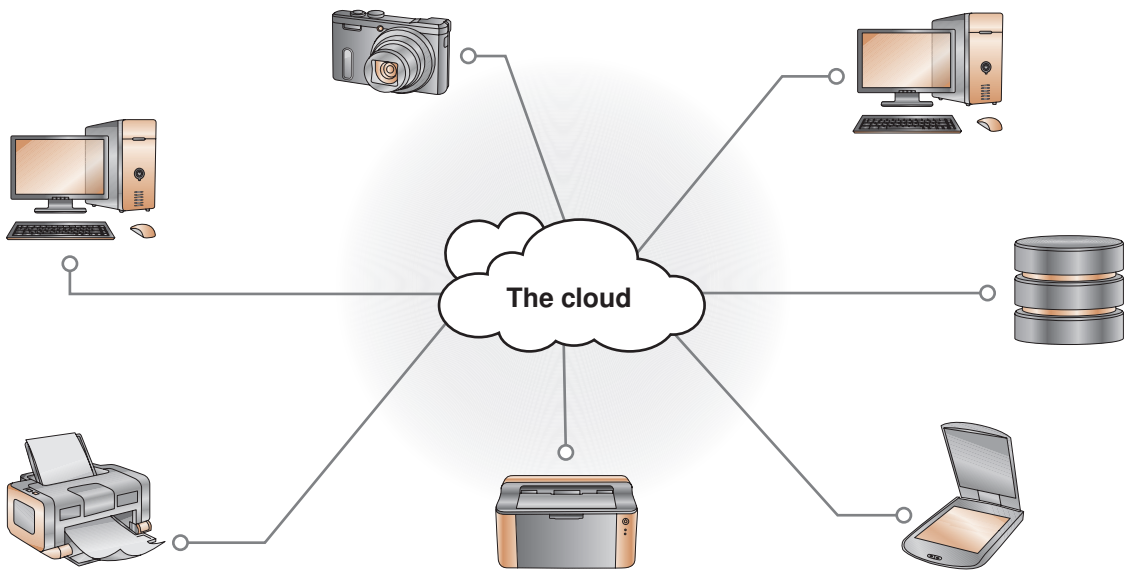
1. The Device Manager receives the electrical impulse caused by a click of the mouse, decodes the command by calculating the location of the cursor, and sends that information through the User Interface, which identifies the requested command. Immediately, it sends the command to the Processor Manager.
2. The Processor Manager then sends an acknowledgment message (such as "waiting" or "loading") to be displayed on the monitor so that the user knows that the command has been sent successfully.
3. The Processor Manager determines whether the user request requires that a file (in this case a program file) be retrieved from storage, or whether it is already in memory.
4. If the program is in secondary storage (perhaps on a disk), the File Manager calculates its exact location on the disk and passes this information to the Device Manager, which retrieves the program and sends it to the Memory Manager.
5. If necessary, the Memory Manager finds space for the program file in main memory and records its exact location. Once the program file is in memory, this manager keeps track of its location in memory.
6. When the CPU is ready to run it, the program begins execution via the Processor Manager. When the program has finished executing, the Processor Manager relays this information to the other managers.

7. The Processor Manager reassigns the CPU to the next program waiting in line. If the file was modified, the File Manager and Device Manager cooperate to store the results in secondary storage. If the file was not modified, there's no need to change the stored version of it.
8. The Memory Manager releases the program's space in main memory and gets ready to make it available to the next program that requires memory.
9. Finally, the User Interface displays the results and gets ready to take the next command.

Although this is a vastly oversimplified demonstration of a very fast and complex operation, it illustrates the incredible precision required for an operating system to work smoothly. The complications increase greatly when networking capability is added. Although we'll be discussing each manager in isolation for much of this text, remember that no single manager could perform its tasks without the active cooperation of every other manager.

Cloud Computing

One might wonder how **cloud computing** changes the role of operating systems. In simplest terms, cloud computing is the practice of using Internet-connected resources to perform processing, storage, or other operations, as shown in Figure 1.8. Generally,



(figure 1.8)

A map showing a few of the numerous system resources that can be connected via the cloud. Cloud-connected devices can be located anywhere in the world if they can access the network.

cloud computing allows operating systems to accommodate remote access to system resources, and provides increased security for these transactions. However, at its roots, the operating system still maintains responsibility for managing all local resources, and coordinating data transfer to and from the cloud. Also, the operating system that is managing the far-away resources is responsible for the allocation and deallocation of all its resources, this time, on a massive scale. Companies, organizations, and individuals are moving a wide variety of resources to the cloud, including data management, file storage, applications, processing, printing, security, and so on. One can expect this trend to continue. But regardless of where the resource is located—in the box, under the desk, or on the cloud—the role of the operating system is the same: to access those resources and manage the entire system as efficiently as possible.

An Evolution of Computing Hardware

To appreciate the role of the operating system (which is software), it may help to understand the computer system's **hardware**, which is the tangible, physical machine and its electronic components, including memory chips, the central processing unit (CPU), the input/output devices, and the storage devices.

- **Main memory** (**RAM**) is where the data and instructions must reside to be processed.
- The **central processing unit** (**CPU**) is the "brains" of the computer. It has the circuitry to control the interpretation and execution of instructions. All storage references, data manipulations, and input/output operations are initiated or performed by the CPU.
- Devices, sometimes called I/O devices for input/output devices, include every peripheral unit attached to the computer system, from printers and monitors to magnetic disks, optical disc drives, flash memory, keyboards, and so on.

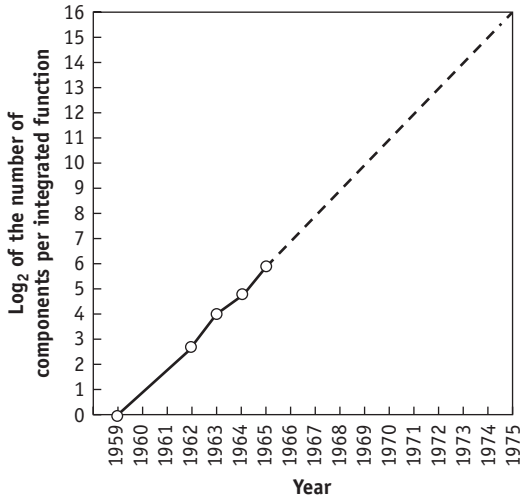
A few of the operating systems that can be used on a variety of platforms are shown in Table 1.1.

(table 1.1)

A brief list of platforms, and a few of the operating systems designed to run on them, listed here in alphabetical order.

Platform	Operating System
Laptops, desktops	Linux, Mac OS X, UNIX, Windows
Mainframe computers	Linux, UNIX, Windows, IBM z/OS
Supercomputers	Linux, UNIX
Telephones, tablets	Android, iOS, Windows
Workstations, servers	Linux, Mac OS X Server, UNIX, Windows

In 1965, Intel executive Gordon Moore observed that each new processor chip contained roughly twice as much capacity as its predecessor (number of components per integrated function), and that each chip was released within 18–24 months of the previous chip. His original paper included a graph (shown in Figure 1.9) predicting that the trend would cause computing power to rise exponentially over relatively brief periods of time, and it has. Now known as Moore's Law, the trend has continued and is still remarkably accurate. Moore's Law is often cited by industry observers when making their chip capacity forecasts.



Number of components per integrated function for minimum cost per component extrapolated vs time.

(figure 1.9)

Gordon Moore's 1965 paper included the prediction that the number of transistors incorporated in a chip will approximately double every 24 months (Moore, 1965).

Courtesy of Intel Corporation.

Types of Operating Systems

Operating systems fall into several general categories distinguished by the speed of their response, and the method used to enter data into the system. The five categories are batch, interactive, real-time, hybrid, and embedded systems.

Batch systems feature jobs that are entered as a whole, and in sequence. That is, only one job can be entered at a time, and once a job begins processing, then no other job can start processing until the resident job is finished. These systems date from early computers, when each job consisted of a stack of cards—or reels of magnetic tape—for input, and were entered into the system as a unit, called a batch. The efficiency of a batch system is measured in throughput which is the number of jobs completed in a given amount of time (usually measured in minutes, hours, or days.)

Interactive systems allow multiple jobs to begin processing, and return results to users with better response times than batch systems; but interactive systems are slower than the real-time systems that we will talk about next. Early versions of these operating systems allowed each user to interact directly with the computer system via commands entered from a typewriter-like terminal. The operating system used complex algorithms to share processing power (often with a single processor) among the jobs awaiting processing. Interactive systems offered huge improvements in responsiveness with turnaround times in seconds or minutes, instead of the hours or days of batch-only systems.

Hybrid systems, widely used today, are a combination of batch and interactive. They appear to be interactive because individual users can enter multiple jobs or processes into the system and get fast responses, but these systems also accept and run batch programs in the background when the interactive load is light. A hybrid system takes advantage of the free time between high-demand usage of the system and low-demand times.

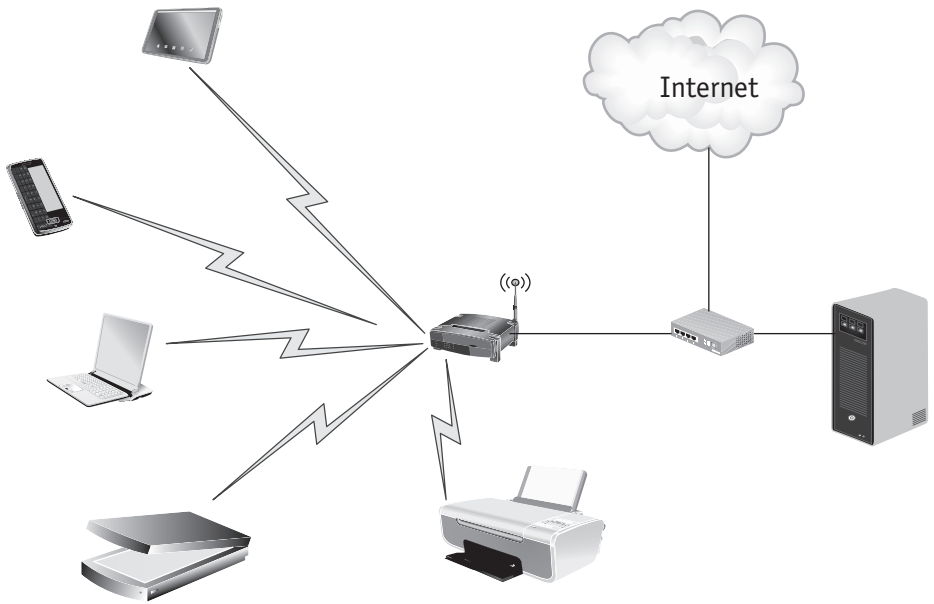
Real-time systems are used in time-critical environments where reliability and deadlines are critical. This time limit need not be ultra-fast, though it often is; however, system response time must meet the deadline because there are significant consequences for not doing so. They also need to provide contingencies to fail gracefully—that is, preserving as much of the system's capabilities and data as possible, throughout system failure, to facilitate recovery. Examples of real-time systems are those used for spacecraft, airport traffic control, fly-by-wire aircraft, critical industrial processes, and medical systems, to name a few. There are two types of real-time systems, depending on the consequences of missing the deadline: hard and soft systems.

- Hard real-time systems risk total system failure if the predicted deadline is missed.
- Soft real-time systems suffer performance degradation, but not total system failure, as a consequence of a missed deadline.

Although it's theoretically possible to convert a general-purpose operating system into a real-time system by merely establishing a deadline, the need to be extremely predictable is not part of the design criteria for most operating systems; so they can't provide the guaranteed response times and graceful failure that real-time performance requires. Therefore, most embedded systems (described in the following paragraphs) and real-time environments require operating systems that are specially designed to meet real-time needs.

Networks allow users to manipulate resources that may be located over a wide geographical area. Network operating systems were originally similar to single-processor operating systems in that each machine ran its own local operating system and served its own local user group. Now, network operating systems make up a special class of software that allows users to perform their tasks using few, if any, local resources. One example of this phenomenon is cloud computing.

As shown in Figure 1.10, wireless networking capability is a standard feature in many computing devices: cell phones, tablets, and other handheld Web browsers.



(figure 1.10)

An example of a simple network. The server is connected by a cable to the router, and the remaining devices connect to it wirelessly.

An **embedded system** is a computer that is physically placed inside the products that it operates to add very specific features and capabilities. For example, embedded systems can be found in automobiles, digital music players, elevators, and pacemakers, to name a few.

Operating systems for embedded computers are very different from those for general computer systems. Each one is designed to perform a set of specific programs, which are not interchangeable among systems. This permits the designers to make the operating system more lean and efficient to take best advantage of the computer's limited resources, typically with slower CPUs and smaller memory resources.

Before a general-purpose operating system, such as Linux, UNIX, or Windows, can be used in an embedded system, the system designers must select which operating system components are required in that particular environment and which are not. The final version of this operating system generally includes redundant safety features, and only the necessary elements; any unneeded features or functions are dropped. Therefore, operating systems with a small **kernel** (the core portion of the software) and other functions that can be mixed and matched to meet the embedded system requirements have potential in this market.

One example of software available to help developers build an embedded system is Windows Embedded Automotive.

Grace Murray Hopper

Grace Hopper developed one of the world's first compilers: intermediate programs that translate human-readable instructions into zeros and ones (the language of the target computer). She then went on to write compiler-based programming languages. A mathematician, she joined the U.S. Navy Reserves in 1943. She was assigned to work on computer systems at Harvard, where she did groundbreaking work, which included her development of the widely adopted COBOL language. In 1969, the annual Data Processing Management Association awarded Hopper its "Man of the Year Award," and in 1973, she became the first woman of any nationality, and the first person from the United States, to be made a Distinguished Fellow of the British Computer Society. She retired from the Navy as Rear Admiral Hopper.

National Medal of Technology and Innovation (1991): "For her pioneering accomplishments in the development of computer programming languages that simplified computer technology and opened the door to a significantly larger universe of users."

Timeline of Operating Systems Development

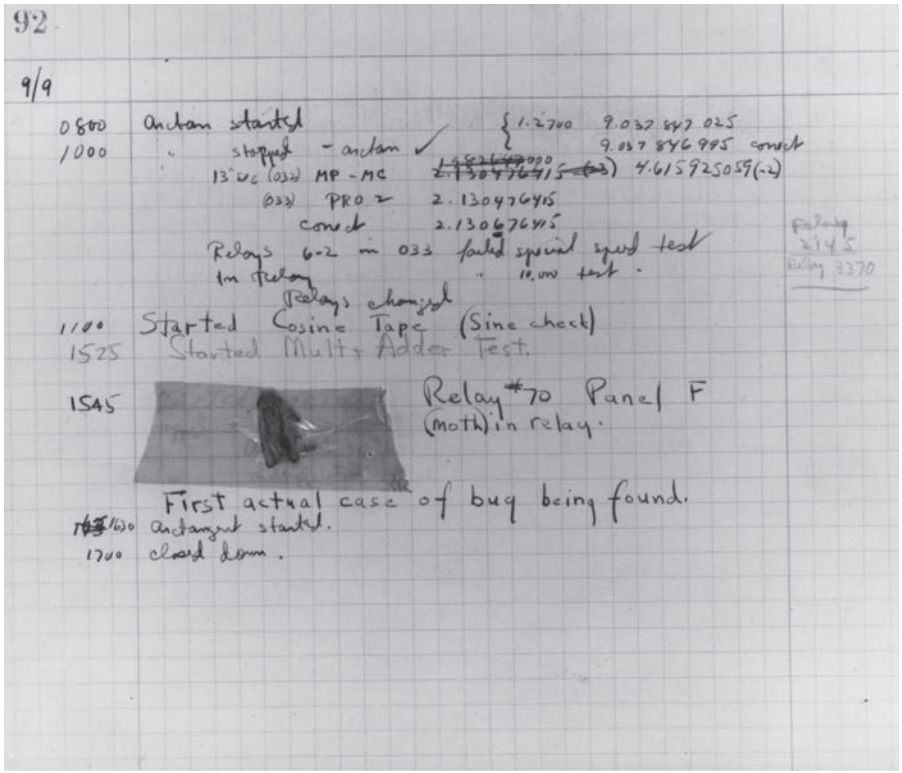
The evolution of early operating system software parallels the evolution of the computer hardware it was designed to control.

1940s

To run programs on early computers, the programmers would reserve the entire machine for the entire time they estimated it would take for the system to execute their program. Then, the computer would sit idle between reservations.

Many things could go wrong with these early computers. For example, when Harvard's Mark I computer stopped working one day in 1945, technicians investigating the cause discovered that a flying moth had worked its way inside Relay 70 in Panel F and short-circuited it. The researcher, Grace Murray Hopper, duly placed the dead insect in the

system log, as shown in Figure 1.11, noting, "First actual case of bug being found." The incident spawned the industry-wide use of the word "bug" to indicate that a system is not working correctly, and the term is still commonly used today.



(figure 1.11)

Dr. Grace Hopper's research journal included the first computer bug. Taped to the page are the remains of a moth that became trapped in the computer's relays, causing the system to crash. [Photo © 2002 IEEE]

1950s

Mid-century improvements included professional computer operators (instead of individual programmers) who were assigned to maximize the computer's operation and schedule the incoming jobs as efficiently as possible. Hardware improvements included:

- faster speeds of input/output devices, such as storage drives and disks systems;
- the practice of grouping records into blocks before they were stored or retrieved in order to use more of the available storage area in the devices;
- the introduction of a control unit to manage data flow in spite of the speed discrepancy between the slow I/O devices and the faster CPU.

During this time, programs were still assigned to the processor one-at-a-time in sequence. The next step toward better use of the system's resources was the move to shared processing.

1960s

Computers in the mid-1960s were designed with faster CPUs, but they still had problems interacting directly with the relatively slow printers and other I/O devices. The solution was called **multiprogramming**, which introduced the concept of loading many programs at one time and allowing them to share the attention of the single CPU.

One mechanism developed to help implement multiprogramming was the introduction of the concept of the interrupt, whereby the CPU was notified of events needing operating systems services. For example, when a program issued a print command, called input/output (I/O) command, it generated an interrupt, which signaled the release of the CPU from one job so it could begin execution of the next job. This was called *passive multiprogramming* because the operating system didn't control the interrupts, but, instead, waited for each job to end on its own. This was less than ideal because if a job was CPU-bound, meaning that it performed a great deal of nonstop in-memory processing before issuing an interrupt, it could monopolize the CPU for a long time while all other jobs waited, even if they were more important.

To counteract this effect, computer scientists designed *active multiprogramming,* which allowed the operating system a more active role. Each program was initially allowed to use only a preset slice of CPU time. When time expired, the job was interrupted by the operating system so another job could begin its execution. The interrupted job then had to wait until it was allowed to resume execution at some later time. Soon, this idea, called time slicing, became common in many interactive systems.

1970s

During this decade, computers were built with faster CPUs, creating an even greater disparity between their rapid processing speed and slower I/O device times. However, schemes to increase CPU use were limited by the small and expensive physical capacity of the main memory. For example, the first Cray supercomputer was installed at Los Alamos National Laboratory in 1976 and had only 8 megabytes (MB) of main memory, a mere fraction of what can be found in many computing devices today.

A solution to this physical limitation was the development of virtual memory, which allowed only a portion of multiple programs to reside in memory at the same time. In other words, a virtual memory system could divide each program into parts, keeping them in secondary storage and bringing each part into memory only as it was needed. Virtual memory soon became standard in operating systems of all sizes, and paved the way toward a much better use of the CPU.

1980s

Computer hardware during this time became more flexible, with logical functions that were built on easily replaceable circuit boards. And because it had become cheaper to create these circuit boards, more operating system functions were made part of the hardware itself, giving rise to a new concept—firmware, a word used to indicate that a program is permanently held in read-only memory (ROM), as opposed to being held in secondary storage.

The evolution of personal computers and high-speed communications sparked the move to networked systems and distributed processing, enabling users in remote locations to share hardware and software resources. These systems required a new kind of operating system—one capable of managing multiple sets of subsystem managers, as well as hardware that might reside half a world away.

1990s

The overwhelming demand for Internet capability in the mid-1990s sparked the proliferation of networking capability. The World Wide Web was first proposed in a paper by Tim Berners-Lee (http://info.cern.ch/Proposal.html); his original concept is shown in Figure 1.12. Based on this research, he designed the first Web server and browser, making it available to the general public in 1991. While his innovations sparked increased connectivity, it also increased demand for tighter security to protect system assets from Internet threats.

The decade also introduced a proliferation of multimedia applications demanding more power, flexibility, and device compatibility for most operating systems, as well as large amounts of storage capability, longer battery life, and cooler operation. These technological advances required that the operating system be equipped with commensurate advancements.

2000s

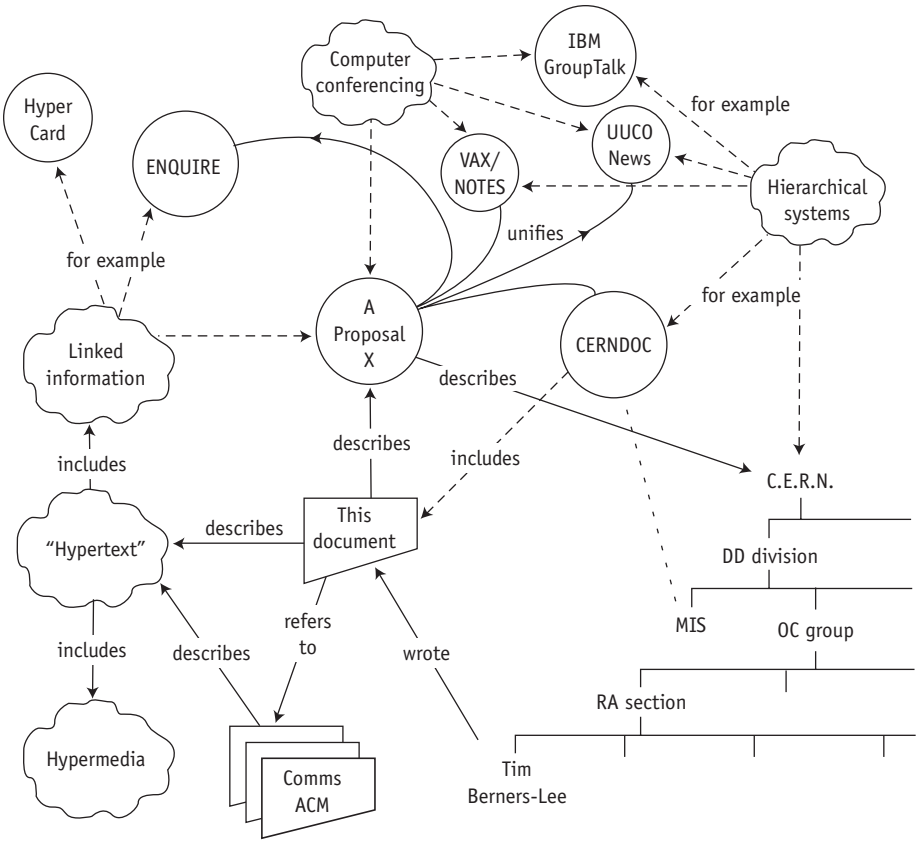
The new century emphasized the need for improved flexibility, reliability, and speed. Virtualization allowed separate partitions of a single **server** to support different operating systems. In other words, it turned a single physical server into multiple virtual servers, often with multiple operating systems. Virtualization required the operating system to have an intermediate manager to oversee the access of each operating system to the server's physical resources.

Processing speed enjoyed a similar advancement with the commercialization of multicore processors, which contained many cores working cooperatively together. For example,

(figure 1.12)

Illustration from the proposal by Tim Berners-Lee describing his revolutionary "linked information system," which became the World Wide Web (www).

(http://info.cern.ch/Proposal.html)

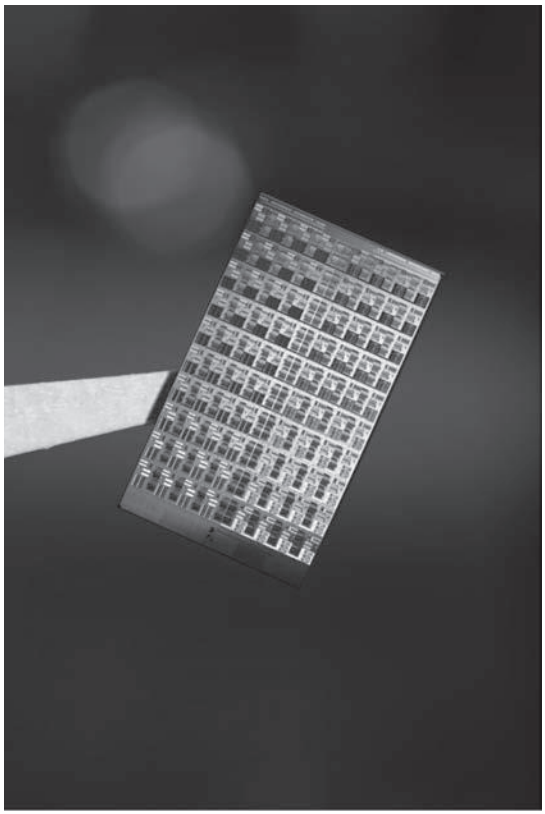


a chip with two CPUs allows two sets of calculations to run at the same time, which sometimes leads to faster job completion (though not always). It's almost as if the user has two separate computers, and, thus, two processors, cooperating on a single task. Designers have created chips that have dozens of cores; one is shown in Figure 1.13.

Does this hardware innovation affect the operating system software? Absolutely. This is because the operating system must now manage the work of each of these processors and be able to schedule and manage the processing of their multiple tasks.

2010s

Increased mobility and wireless connectivity spawned a proliferation of multicore CPUs (a processor is also called a core) on one computer chip. Multicore engineering was driven by the problems caused by nano-sized transistors and their ultra-close placement on a computer chip. Although chips with millions of transistors placed very close



(figure 1.13)

This single piece of silicon can hold 80 cores, which, to put it in the simplest terms, can perform up to 80 calculations at one time, and do so without overheating.

(Courtesy of Intel Corporation)

together helped increase system performance dramatically, the close proximity of these transistors also allowed current to "leak," which caused the buildup of heat, as well as other problems. With the development of multi-core technology, a single chip (one piece of silicon) could be equipped with two or more processor cores. In other words, they replaced a single large processor with two half-sized processors (called dual core), four quarter-sized processors (quad core), and so on. This design allowed the same sized chip to produce less heat and offered the opportunity to permit multiple calculations to take place at the same time.

Role of the Software Designer

Who writes an operating system? Depending on the complexity of the computer, the design team could range from a single person to a multi-skilled team of people. Those who create an operating system are faced with many choices that can affect every part of the software and the resources it controls. Before beginning, designers typically start by asking key questions, using the answers to guide them in their work. The most common overall goal is to maximize the use of the system's resources (memory, processing,

The number of operating systems is uncountable. Some are specific only to a single type of device, and some are widely used on systems of all sizes and capabilities. The Linux operating system was originally written by a young student named Linus Torvalds in Finland. There's more about Torvalds in Chapter 15.

devices, and files) and minimize downtime, though certain proprietary systems may have other priorities. Typically, designers include the following factors into their developmental efforts: the minimum and maximum main memory resources; the number and brand of CPUs; the variety of storage devices likely to be connected; the types of files, networking capability, security requirements, and default user interfaces available; assumed user capabilities; and so on.

For example, a mobile operating system often needs to minimize the heat the device generates. Likewise, if it's a real-time operating system, designers need to aggressively manage memory usage, processor time, device allocation, and files so that urgent deadlines will not be missed. For these reasons, operating systems are often complex pieces of software.

As we might expect, no single operating system is perfect for every environment. Some systems can be best served with a UNIX system, others benefit from the structure of a Windows system, and still others work best using Linux, Mac OS, Android, or even a custom-built operating system.

Conclusion

In this chapter, we looked at the definition of an operating system as well as its overall function. What began as hardware-dependent operations software has evolved to run increasingly complex computers and, like any complicated subject, there's much more detail to explore, such as the role of the main memory resource, the CPU (processor), the system's input and storage devices, and its numerous files. Each of these functions needs to be managed seamlessly, as does the cooperation among them and other system essentials, such as the network it's connected to. As we'll see in the remainder of this text, there are many ways to perform every task, and it's up to the designers of the operating system to choose the policies that best match the environment and its users.

In the following chapters, we'll explore in detail how each portion of the operating system works, as well as its features, functions, and benefits. We'll begin with the part of the operating system that's the heart of every computer: the module that manages main memory.

Key Terms

batch system: a type of computing system that executes programs, each of which is submitted in its entirety, can be grouped into batches, and is executed without external intervention.

central processing unit (CPU): a component with circuitry that controls the interpretation and execution of instructions. See also *processor*.

cloud computing: a multifaceted technology that allows computing, data storage and retrieval, and other computer functions to take place via a large network, typically the Internet.

Device Manager: the section of the operating system responsible for controlling the use of devices. It monitors every device, channel, and control unit and chooses the most efficient way to allocate all of the system's devices.

embedded system: a dedicated computer system that is often part of a larger physical system, such as a jet aircraft or automobile. Often, it must be small, fast, and able to work with real-time constraints, fail-safe execution, and nonstandard I/O devices.

File Manager: the section of the operating system responsible for controlling the use of files.

firmware: software instructions, or data, that are stored in a fixed or "firm" way, usually implemented on some type of read-only memory (ROM).

hardware: the tangible machine and its components, including main memory, I/O devices, I/O channels, direct access storage devices, and the central processing unit.

hybrid system: a computer system that supports both batch and interactive processes.

interactive system: a system that allows each user to interact directly with the operating system.

kernel: the primary part of the operating system that remains in random access memory (RAM), and is charged with performing the system's most essential tasks, such as managing main memory and disk access.

main memory (RAM): the memory unit that works directly with the CPU, and in which the data and instructions must reside in order to be processed. Also called *primary storage, RAM,* or *internal memory.*

Memory Manager: the section of the operating system responsible for controlling the use of memory. It checks the validity of each request for memory space, and if it's a legal request, allocates the amount of memory required to execute the job.

multiprogramming: a technique that allows a single processor to process several programs residing simultaneously in main memory, and interleaving their execution by overlapping I/O requests with CPU requests.

Network Manager: the section of the operating system responsible for controlling the access to, and use of, networked resources.

network: a system of interconnected computer systems and peripheral devices that exchange information with one another.

operating system: the primary software on a computing system that manages its resources, controls the execution of other programs, and manages communications and data storage.

process: an instance of execution of a program that is identifiable and controllable by the operating system.

processor: (1) another term for the CPU (central processing unit); (2) any component in a computing system capable of performing a sequence of activities. It controls the interpretation and execution of instructions.

Processor Manager: a composite of two submanagers, the Job Scheduler and the Process Scheduler, that decides how to allocate the CPU.

RAM: short for random access memory. See *main memory*.

real-time system: a computing system used in time-critical environments that require guaranteed response times. Examples include navigation systems, rapid transit systems, and industrial control systems.

server: a node that provides clients with various network services, such as file retrieval, printing, or database access services.

storage: the place where data is stored in the computer system. Primary storage is main memory. Secondary storage is nonvolatile media, such as disks and flash memory.

user interface: the portion of the operating system that users interact with directly—is one of the most unique and most recognizable components of an operating system.

To Explore More

For additional background on a few of the topics discussed in this chapter, begin a search with these terms.

- Embedded computers aboard the International Space Station
- Operating systems for mainframes
- UNIX operating system in Apple devices
- Windows systems for sports stadiums
- Android operating system for phones

Exercises

Research Topics

A. Write a one-page review of an article about the subject of operating systems that appeared in a recent computing magazine or academic journal. Give a summary of the article, including the primary topic, your own summary of the information presented, and the author's conclusion. Give your personal evaluation of the article, including topics that made the article interesting to you (or not) and its relevance to your own experiences. Be sure to cite your sources.

B. In the computing environment the numerical value represented by the prefixes kilo-, mega-, giga-, and so on can vary depending on whether they are describing bytes of main memory or bits of data transmission speed. Research the actual value (the number of bytes) in a Megabyte (MB) and then compare that value to the number of bits in a Megabit (Mb). Are they the same or different? If there is a difference or uncertainty, explain why that is the case. Cite your sources.

Exercises

1. Give a real-world example of a task that you perform everyday via cloud computing. Explain how you would cope if the network connection suddenly became unavailable.
2. In your opinion, what would be the consequences if the Memory Manager and the File Manager stopped communicating with each other?
3. In your opinion, what would be the consequences if the Memory Manager and the Processor Manager stopped communicating with each other?
4. Gordon Moore predicted the dramatic increase in transistors per chip in 1965 and his prediction has held for decades. Some industry analysts insist that Moore's Law has been a predictor of chip design, but others say it is a motivator for designers of new chips. In your opinion, who is correct? Explain your answer.
5. Give an example of an organization that might find batch-mode processing useful and explain why.
6. Give an example of a situation that might need a real-time operating system and explain in detail why you believe that would be the case.
7. Name five current operating systems (other than those mentioned in Table 1.1) and identify the computers, platforms, or configurations where each is used.
8. Many people confuse main memory and secondary storage. Explain why this might happen, and describe how you would explain the differences to classmates so they would no longer confuse the two.
9. Name the five key concepts about an operating system that you think a typical user needs to know and understand.
10. Explain the impact of the continuing evolution of computer hardware and the accompanying evolution of operating systems software.
11. List three tangible, physical, resources that can be found on a typical computer system.
12. Select two of the following professionals: an insurance adjuster, a delivery person for a courier service, a newspaper reporter, a doctor (general practitioner),

or a manager in a supermarket. Suggest at least two ways that each person might use a mobile computer to work more efficiently.

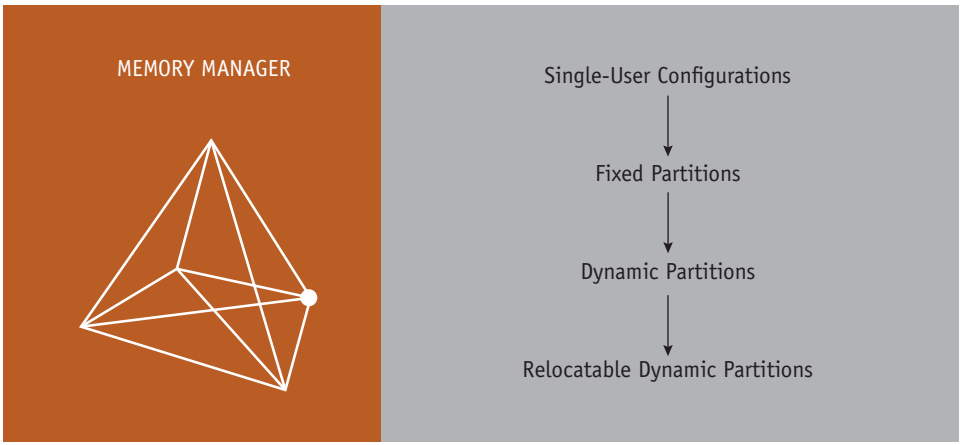
13. Give real-world examples of interactive, batch, real-time, and embedded systems and explain the fundamental differences that distinguish each one from the others.
14. Briefly compare active and passive multiprogramming and give examples of each.

Advanced Exercises

Advanced Exercises explore topics not discussed in this chapter and are appropriate for readers with supplementary knowledge of operating systems.

15. Give at least two reasons why a state-wide bank might decide to buy six networked servers instead of one mainframe.
16. Compare the development of two operating systems, described in Chapters 13–16 of this text, including design goals and evolution. Name the operating system each was based on, if any. Which one do you believe is more efficient for your needs? Explain why.
17. Draw a system flowchart illustrating the steps performed by an operating system as it executes the instruction to back up a disk on a single-user computer system. Begin with the user typing the command on the keyboard or choosing an option from a menu, and conclude with the result being displayed on the monitor.
18. In a multiprogramming and time-sharing environment, several users share a single system at the same time. This situation can result in various security problems. Name two such problems. Can we ensure the same degree of security in a time-share machine as we have in a dedicated machine? Explain your answers.
19. Give an example of an application where multithreading gives improved performance over single-threading.
20. If a process terminates, will its threads also terminate or will they continue to run? Explain your answer.
21. The boot sequence is the series of instructions that enable the operating system to get installed and running. In your own words, describe the role of firmware and the boot process for an operating system of your choice.
22. A "dual boot" system gives users the opportunity to choose from among a list of operating systems when powering on a computer. Describe how this process works. Explain whether or not there is a risk that one operating system could intrude on the space reserved for another operating system.

Chapter 2 Early Memory Management Systems



"Memory is the primary and fundamental power, without which there could be no other intellectual opeatrion."

—Samuel Johnson (1709–1784)

Learning Objectives

After completing this chapter, you should be able to describe:

- How four memory allocation schemes in this chapter manage incoming jobs
- How two memory allocation systems work: best-fit and first-fit
- How a memory list is used to keep the status of available memory
- The essential role of memory deallocation, and the consequences if it isn't performed
- How compaction can improve memory allocation efficiency

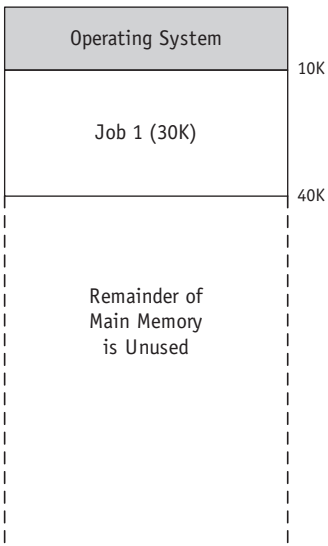
One of the most critical segments of the operating system is the part that manages the **main memory**. In fact, for early computers, the performance of the *entire* system was judged by the quantity of the available main memory resources, and how that memory was optimized while jobs were being processed. Back in those days, jobs were submitted serially, one at a time; and managing memory centered on assigning that memory to each incoming job, and then reassigning it when the job was finished.

Main memory has gone by several synonyms, including random access memory or **RAM**, core memory, and primary storage (in contrast with secondary storage, where data is stored in a more permanent way). This chapter discusses four types of memory allocation schemes: single-user systems, fixed partitions, dynamic partitions, and relocatable dynamic partitions. Let's begin with the simplest memory management scheme: the one used with early computer systems.

Single-User Contiguous Scheme

This memory allocation scheme works like this: before execution can begin, each job or program is loaded, in its entirety, into memory, and is allocated as much contiguous space in memory as it needs, as shown in Figure 2.1. The key terms here are *entirety* and *contiguous*. If the program is too large to fit into the available memory space, it cannot begin execution.

This scheme demonstrates a significant limiting factor of all computers—they have only a finite amount of memory. If a program doesn't fit, then either the size of main memory must be increased, or the program must be modified to fit, often by revising it to be smaller.



The earliest memory management scheme processed only a single job at a time. It did not allow two jobs to share main memory at the same time, even if a sufficient amount of memory was available.

(figure 2.1)

Only one program (this one is 30K in size) at a time is allowed to occupy memory, even if there is room to accommodate the next job that's waiting. Notice that the space that's reserved for the operating system must remain in main memory.

Single-user systems in a non-networked environment allocate to each user, access to all available main memory for each job, and jobs are processed sequentially, one after the other. To allocate memory, the amount of work required from the operating system's Memory Manager is minimal, as described in the following steps:

1. Evaluate the incoming job to see if it is small enough to fit into the available space. If it is, load it into memory; if not, reject it and evaluate the next incoming process.
2. Monitor the occupied memory space. When the resident job ends its execution and no longer needs to be in memory, indicate that the entire amount of main memory space is now available and return to Step 1, evaluating the next incoming job.

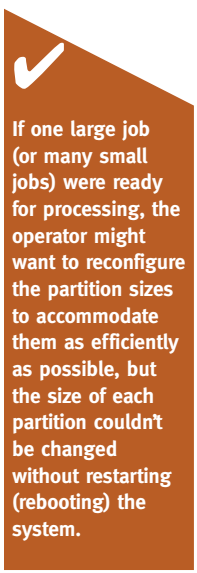
An "Algorithm to Load a Job in a Single-User System," using pseudocode and demonstrating these steps, can be found in Appendix A.

Once the incoming job is entirely loaded into memory, it begins its execution and remains there until the execution is complete, either by finishing its work, or through the intervention of the operating system, such as when an error is detected. When it finishes, the operating system prepares to load the next waiting job, and so on.

One major problem with this type of memory allocation scheme (first made available commercially in the late 1940s and early 1950s) is that it doesn't support multiprogramming (multiple jobs or processes occupying memory at the same time).

Fixed Partitions

The first attempt to allow for multiprogramming used **fixed partitions** (sometimes called static partitions) within main memory—that is, the entirety of each partition could be assigned to a single job, and multiple partitions allowed multiple jobs at the same time. This memory management system was naturally more complex to manage.



For example, a system with four partitions could hold four jobs in memory at the same time. One fact remained the same, however: these partitions were static, so the systems administrator had to turn off the entire system to reconfigure partition sizes, and any job that couldn't fit into the largest partition could not be executed. The role of the individual operating the computing system was essential.

An important factor was introduced with this scheme: protection of the job's assigned memory space. Once a partition was allocated to a job, the jobs in other memory partitions had to be prevented from invading its boundaries, either accidentally or intentionally. This problem of partition intrusion didn't exist in the single-user contiguous allocation scheme because only one job was present in the main memory space at any given time, that is, only the portion of main memory that held the operating system had to be protected. However, with the introduction of fixed partition allocation schemes,

protection was mandatory for each partition in main memory. Typically, this was the joint responsibility of the hardware of the computer and of the operating system.

The algorithm used to store jobs in memory requires a few more steps than the one used for a single-user system because the size of the job must be matched with the size of the available partitions in order to make sure it fits completely. ("An Algorithm to Load a Job in a Fixed Partition" is in Appendix A.) Remember, the fixed partitions scheme also required that the entire job be loaded into memory before execution could begin.

To do so, the Memory Manager would perform these steps (in a two-partition system):

1. Check the incoming job's memory requirements. If it's greater than the size of the largest partition, reject the job and go to the next waiting job. If it's less than the largest partition, go to Step 2.
2. Check the job size against the size of the first available partition. If the job is small enough to fit, see if that partition is free. If it is available, load the job into that partition. If it's busy with another job, go to Step 3.
3. Check the job size against the size of the second available partition. If the job is small enough to fit, check to see if that partition is free. If it is available, load the incoming job into that partition. If not, go to Step 4.
4. Because neither partition is available now, place the incoming job in the waiting queue for loading at a later time. Return to Step 1 to evaluate the next incoming job.

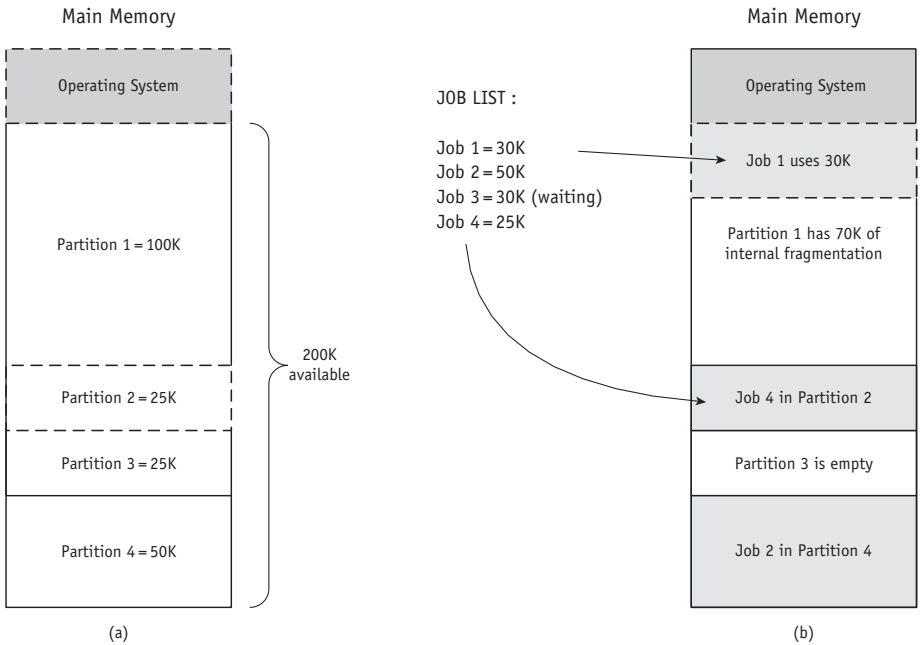
This partition scheme is more flexible than the single-user scheme because it allows more than one program to be in memory at the same time. However, it still requires that the *entire* program be stored *contiguously* and *in memory* from the beginning to the end of its execution.

In order to allocate memory spaces to jobs, the Memory Manager must maintain a table showing each memory partition's size, its **address**, its access restrictions, and its current status (free or busy). Table 2.1 shows a simplified version for the system illustrated in Figure 2.2. Note that Table 2.1 and the other tables in this chapter have been simplified. More detailed discussions of these tables and their contents are presented in Chapter 8, "File Management."

Partition Size	Memory Address	Access	Partition Status
100K	200K	Job 1	Busy
25K	300K	Job 4	Busy
25K	325K		Free
50K	350K	Job 2	Busy

(table 2.1)

A simplified fixed-partition memory table for several small jobs with the free partition shaded.



(figure 2.2) *As the small jobs listed in Table 2.1 are loaded into the four fixed partitions, Job 3 must wait, even though Partition 1 has 70K of available memory. These jobs are allocated space on the basis of "first available partition that accommodates the job's size."*

When each resident job terminates, the status of its memory partition is changed from busy to free in order to make it available to an incoming job.

The fixed partition scheme works well if all of the jobs that run on the system are of similar size, or if the sizes are known ahead of time and don't vary between reconfigurations. Ideally, this would require accurate, advance knowledge of all the jobs waiting to be run on the system in the coming hours, days, or weeks. However, under most circumstances, the operator chooses partition sizes in an arbitrary fashion, and, thus, not all incoming jobs fit in them.

There are significant consequences if the partition sizes are too small: large jobs will need to wait if the large partitions are already booked, and they will be rejected if they're too big to fit into the largest partition.

On the other hand, if the partition sizes are too big, memory is wasted. Any job that occupies less than the entire partition (the vast majority will do so) will cause the unused memory in the partition to remain idle. Remember that each partition is an indivisible unit that can be allocated to only one job at a time. Figure 2.3 demonstrates one such circumstance: Job 3 is kept waiting because it's too big for Partition 3, even though there is more than enough unused space for it in Partition 1, which is claimed by Job 1.

This phenomenon of less-than-complete use of memory space in a fixed partition is called **internal fragmentation** because it's inside a partition, and is a major drawback to this memory allocation scheme.