ann mciver mchoes • ida m. flynn

understanding operating systems

sixth edition

COURSE TECHNOLOGY
CENGAGE Learning™

Understanding Operating Systems, Sixth Edition
Ann McIver McHoes and Ida M. Flynn

Executive Editor: Marie Lee

Acquisitions Editor: Amy Jollymore

Senior Product Manager: Alyssa Pratt

Editorial Assistant: Zina Kresin

Content Project Manager: Jennifer Feltri

Art Director: Faith Brosnan

Print Buyer: Julio Esperas

Cover Designer: Night & Day Design

Cover Photos: iStockphoto

Proofreader: Suzanne Huizenga

Indexer: Ann McIver McHoes

Compositor: Integra

Library of Congress Control Number: 2010920344

ISBN-13: 978-1-4390-7920-1

ISBN-10: 1-4390-7920-x

Course Technology
20 Channel Center Street
Boston, MA 02210
USA

Printed in the United States of America
1 2 3 4 5 6 7 16 15 14 13 12 11 10

Part One

Operating Systems Concepts

"So work the honey-bees,
Creatures that by a rule in nature teach
The act of order to a peopled kingdom."

—William Shakespeare (1564–1616; in Henry V)

All operating systems have certain core items in common: each must manage memory, processing capability, devices and peripherals, files, and networks. In Part One of this text we present an overview of these operating systems essentials.

- Chapter 1 introduces the subject.
- Chapters 2–3 discuss main memory management.
- Chapters 4–6 cover processor management.
- Chapter 7 concentrates on device management.
- Chapter 8 is devoted to file management.
- Chapters 9–10 briefly review networks.
- Chapter 11 discusses system security issues.
- Chapter 12 explores system management and the interaction of the operating system's components.

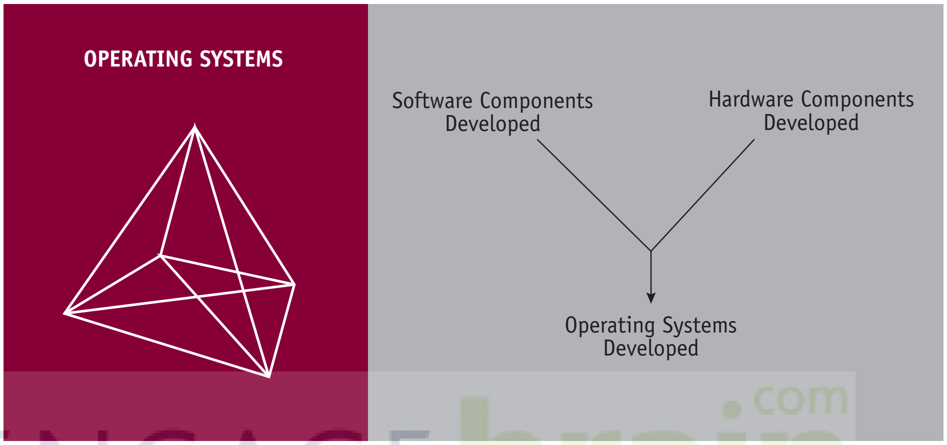
Then, in Part Two of the text (Chapters 13–16), we look at specific operating systems and how they apply the theory presented here in Part One.

Throughout our discussion of this very technical subject, we try to include definitions of terms that might be unfamiliar to you. However, 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, and at the end of the text is an extensive glossary for your reference. Items listed in the Key Terms are shown in boldface the first time they appear.

Throughout the book we keep our descriptions and examples as simple as possible to introduce you to the system's complexities without getting bogged down in technical detail. Therefore, be aware that for almost every topic explained in the following pages, there's much more information that can be studied. Our goal is to introduce you to the subject, and to encourage you to pursue your interest using other texts or primary sources if you need more detail.

Chapter 1

Introducing Operating Systems


OPERATING SYSTEMS
Software Components Developed
Hardware Components Developed
Operating Systems Developed

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

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

Learning Objectives

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

- Innovations in operating system development
- The basic role of an operating system
- The major operating system software subsystem managers and their functions
- The types of machine hardware on which operating systems run
- The differences among batch, interactive, real-time, hybrid, and embedded operating systems
- Multiprocessing and its impact on the evolution of operating system software
- Virtualization and core architecture trends in new operating systems

Introduction

To understand an operating system is to understand the workings of an entire computer system, because the operating system manages each and every piece of hardware and software. This text explores what operating systems are, how they work, what they do, and why.

This chapter briefly describes how simple operating systems work and how, in general, they've evolved. 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, you see how the pieces work harmoniously to keep the computer system working smoothly.

What Is an Operating System?

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

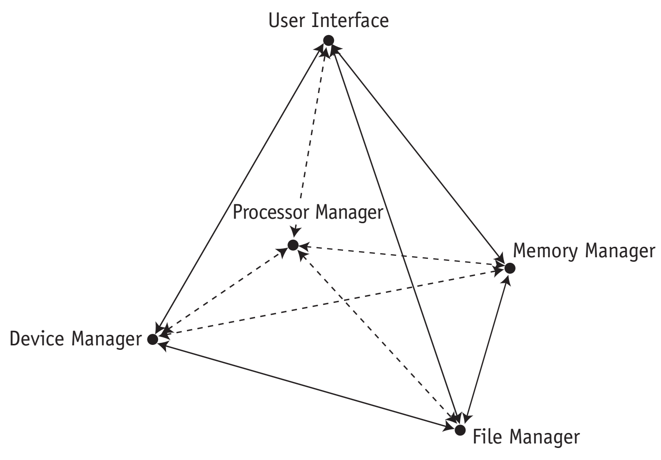
Therefore, each time the user sends a command, the operating system must make sure that the command is executed; or, if it's not executed, it must arrange for the user to get a message explaining the error. Remember: This doesn't necessarily mean that the operating system executes the command or sends 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 an operating system and demonstrates how its major components work together.

At the base of the pyramid are the four essential managers of every operating system: the **Memory Manager**, the **Processor Manager**, the **Device Manager**, and the **File Manager**. In fact, these managers are the basis of all operating systems and each is discussed in detail throughout the first part of this book. Each manager works closely with the other managers and performs its unique role regardless of which specific operating system is being discussed. At the top of the pyramid is the User Interface, from which users issue commands to the operating system. This is the component that's unique to each operating system—sometimes even between different versions of the same operating system.

Unless we mention networking or the Internet, our discussions apply to the most basic elements of operating systems. Chapters 9 and 10 are dedicated to networking.



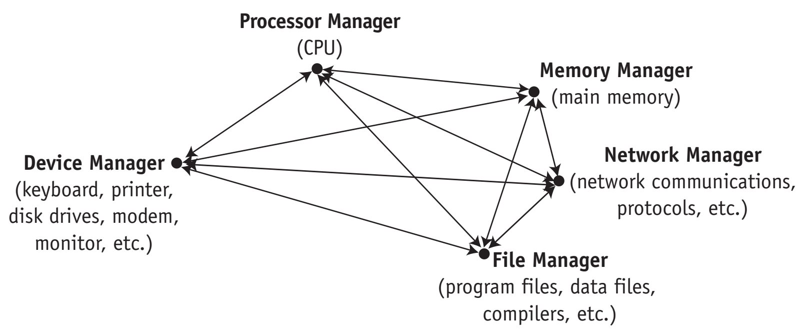
(figure 1.1)

This model of a non-networked operating system shows four subsystem managers supporting the User Interface.

A **network** was not always an integral part of operating systems; early systems were self-contained with all network capability added on top of existing operating systems. Now most operating systems routinely incorporate a **Network Manager.** The base of a pyramid for a networked operating system is shown in Figure 1.2.

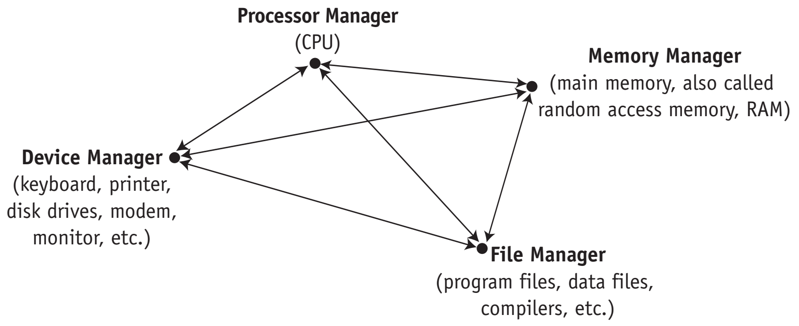
Regardless of the size or configuration of the system, each of the subsystem managers, shown in Figure 1.3, must perform the following tasks:

- Monitor its resources continuously
- Enforce the policies that determine who gets what, when, and how much
- Allocate the resource when appropriate
- Deallocate the resource when appropriate


Processor Manager
(CPU)
Memory Manager
(main memory)
Device Manager
(keyboard, printer, disk drives, modem, monitor, etc.)
Network Manager
(network communications, protocols, etc.)
File Manager
(program files, data files, compilers, etc.)

(figure 1.2)

Networked systems have a Network Manager that assumes responsibility for networking tasks while working harmoniously with every other manager.


Processor Manager
(CPU)
Memory Manager
(main memory, also called random access memory, RAM)
Device Manager
(keyboard, printer, disk drives, modem, monitor, etc.)
File Manager
(program files, data files, compilers, etc.)

(figure 1.3)

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

Main Memory Management

The Memory Manager (the subject of Chapters 2–3) is in charge of main memory, also 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, it allocates a portion of memory that isn't already in use. In a multiuser environment, the Memory Manager sets up a table to keep track of who is using which section of memory. Finally, when the time comes to reclaim the memory, the Memory Manager deallocates memory.

✓ **RAM is the computer's main memory and was called "primary storage" in early systems.**

A primary responsibility of the Memory Manager is to protect the space in main memory occupied by the operating system itself—it can't allow any part of it to be accidentally or intentionally altered.

Processor Management

The Processor Manager (the subject of Chapters 4–6) 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 process. A process is defined here as an instance of execution of a program.

The Processor Manager monitors whether the CPU is executing a process or waiting for a READ or WRITE command to finish execution. Because it handles the processes' transitions from one state of execution to another, it can be compared to a traffic controller. Once the Processor Manager allocates the processor, it sets up the necessary registers and tables and, when the job is finished or the maximum amount of time has expired, it reclaims the processor.

Think of it this way: The Processor Manager has two levels of responsibility. One is to handle jobs as they enter the system and the other is to manage each process within those jobs. The first part is handled by the Job Scheduler, the high-level portion of the Processor Manager, which accepts or rejects the incoming jobs. The second part is handled by the Process Scheduler, the low-level portion of the Processor Manager, which is responsible for deciding which process gets the CPU and for how long.

Device Management

The Device Manager (the subject of Chapter 7) monitors every device, channel, and control unit. Its job is to choose the most efficient way to allocate all of the system's devices, printers, ports, disk drives, and so forth, based on a scheduling policy chosen by the system's designers.

The Device Manager does this by allocating each resource, starting its operation, and, finally, deallocating the device, making it available to the next process or job.

File Management

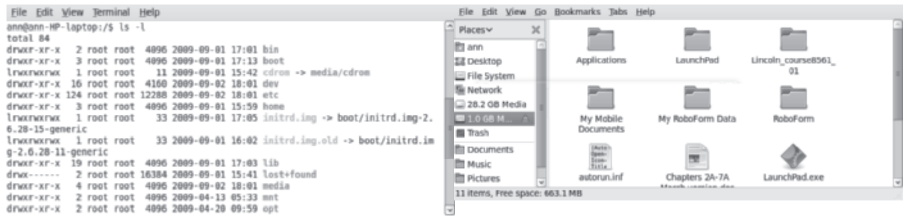
The File Manager (the subject of Chapter 8) keeps track of every file in the system, including data files, program files, compilers, and applications. By using predetermined access policies, it enforces restrictions on who has access to which files. The File Manager also controls what users are allowed to do with files once they access them. For example, a user might have read-only access, read-and-write access, or the authority to create and delete files. Managing access control is a key part of file management. Finally, the File Manager allocates the necessary resources and later deallocates them.

Network Management

Operating systems with Internet or networking capability have a fifth essential manager called the Network Manager (the subject of Chapters 9–10) that provides a convenient way for users to share resources while controlling users' access to them. These resources include hardware (such as CPUs, memory areas, printers, tape drives, modems, and disk drives) and software (such as compilers, application programs, and data files).

User Interface

The user interface is the portion of the operating system that users interact with directly. In the old days, the user interface consisted of commands typed on a keyboard and displayed on a monitor, as shown in Figure 1.4. Now most systems allow users to choose a menu option from a list. The user interface, desktops, and formats vary widely from one operating system to another, as shown in Chapters 13–16 in Part Two of this text.



```
ann@ann-HP-laptop:/$ ls -l
total 84
drwxr-xr-x   2 root root  4096 2009-09-01 17:01 bin
drwxr-xr-x   3 root root  4096 2009-09-01 17:13 boot
lrwxrwxrwx   1 root root    11 2009-09-01 15:42 cdrom -> media/cdrom
drwxr-xr-x  16 root root  4160 2009-09-02 18:01 dev
drwxr-xr-x 124 root root 12288 2009-09-02 18:01 etc
drwxr-xr-x   3 root root  4096 2009-09-01 15:59 home
lrwxrwxrwx   1 root root    33 2009-09-01 17:05 initrd.img -> boot/initrd.img-2.
6.28-15-generic
lrwxrwxrwx   1 root root    33 2009-09-01 16:02 initrd.img.old -> boot/initrd.im
g-2.6.28-11-generic
drwxr-xr-x  19 root root  4096 2009-09-01 17:03 lib
drwx------   2 root root 16384 2009-09-01 15:41 lost+found
drwxr-xr-x   4 root root  4096 2009-09-02 18:01 media
drwxr-xr-x   2 root root  4096 2009-04-13 05:33 mnt
drwxr-xr-x   2 root root  4096 2009-04-20 09:59 opt
```

(figure 1.4)

Two user interfaces from Linux: a command-driven interface (left) and a menu-driven interface (right).

Cooperation Issues

However, it is not enough for each manager to perform its individual tasks. It must also be able to work harmoniously with every other manager. Here is a simplified example. Let's say someone chooses an option from a menu to execute a program. The following major steps must occur in sequence:

1. The Device Manager must receive the electrical impulses from the mouse or keyboard, form the command, and send the command to the User Interface, where the Processor Manager validates the command.
2. The Processor Manager then sends an acknowledgment message to be displayed on the monitor so the user realizes the command has been sent.
3. When the Processor Manager receives the command, it determines whether the program must be retrieved from storage or is already in memory, and then notifies the appropriate manager.
4. If the program is in storage, the File Manager must calculate its exact location on the disk and pass this information to the Device Manager, which retrieves the program and sends it to the Memory Manager.
5. The Memory Manager then finds space for it and records its exact location in memory. Once the program is in memory, the Memory Manager must track its location in memory (even if it's moved) as well as its progress as it's executed by the Processor Manager.
6. When the program has finished executing, it must send a finished message to the Processor Manager so that the processor can be assigned to the next program waiting in line.
7. Finally, the Processor Manager must forward the finished message to the Device Manager, so that it can notify the user and refresh the screen.

Although this is a vastly oversimplified demonstration of a complex operation, it illustrates some of the incredible precision required for the operating system to work smoothly. So although we'll be discussing each manager in isolation for much of this text, no single manager could perform its tasks without the active cooperation of every other part.

A Brief History of Machine Hardware

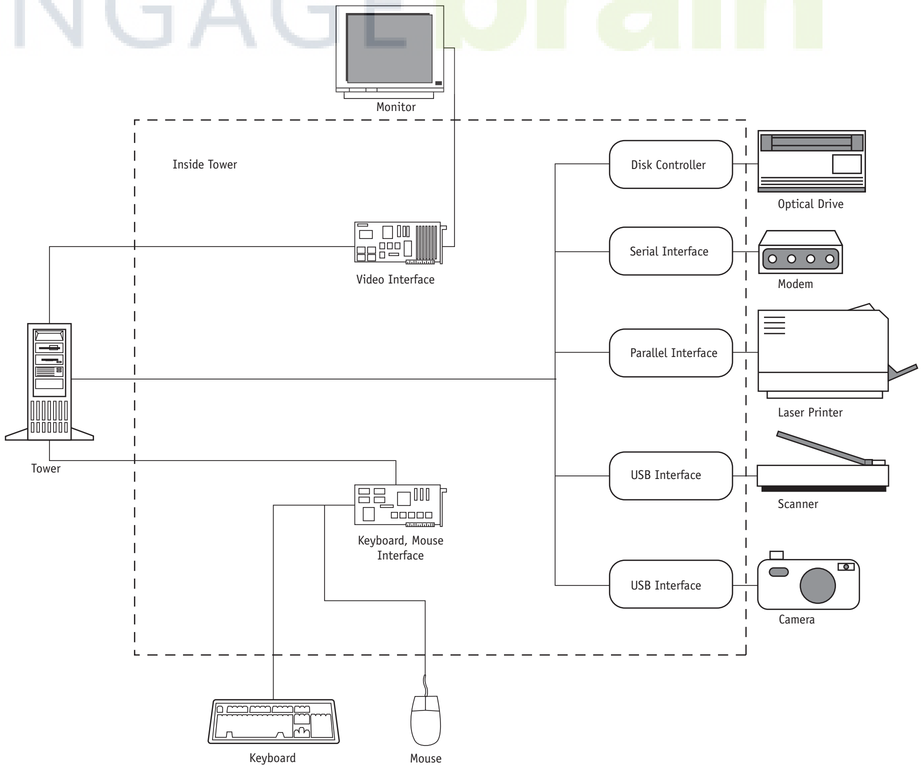
To appreciate the role of the operating system (which is software), we need to discuss the essential aspects of the computer system's hardware, the physical machine and its electronic components, including memory chips, input/output devices, **storage** devices, and the central processing unit (CPU).

- **Main memory** (random access memory, RAM) is where the data and instructions must reside to be processed.
- I/O devices, short for input/output devices, include every peripheral unit in the system such as printers, disk drives, CD/DVD drives, flash memory, keyboards, and so on.
- The **central processing unit** (CPU) is the brains with the circuitry (sometimes called the chip) to control the interpretation and execution of instructions. In essence, it controls the operation of the entire computer system, as illustrated in Figure 1.5. All storage references, data manipulations, and I/O operations are initiated or performed by the CPU.

Until the mid-1970s, computers were classified by capacity and price. A **mainframe** was a large machine—in size and in internal memory capacity. The IBM 360, introduced in

(figure 1.5)

A logical view of a typical computer system hardware configuration. The tower holds the central processing unit, the arithmetic and logic unit, registers, cache, and main memory, as well as controllers and interfaces shown within the dotted lines.



1964, is a classic example of an early mainframe. The IBM 360 model 30 required an air-conditioned room about 18 feet square to house the CPU, the operator's console, a printer, a card reader, and a keypunch machine. The CPU was 5 feet high and 6 feet wide, had an internal memory of 64K (considered large at that time), and a price tag of $200,000 in 1964 dollars. Because of its size and price at the time, its applications were generally limited to large computer centers belonging to the federal government, universities, and very large businesses.

The **minicomputer** was developed to meet the needs of smaller institutions, those with only a few dozen users. One of the early minicomputers was marketed by Digital Equipment Corporation to satisfy the needs of large schools and small colleges that began offering computer science courses in the early 1970s. (The price of its PDP-8 was less than $18,000.) Minicomputers are smaller in size and memory capacity and cheaper than mainframes. Today, computers that fall between microcomputers and mainframes in capacity are often called midrange computers.

The **supercomputer** was developed primarily for government applications needing massive and fast number-crunching ability to carry out military operations and weather forecasting. Business and industry became interested in the technology when the massive computers became faster and less expensive. A Cray supercomputer is a typical example with six to thousands of processors performing up to 2.4 trillion floating point operations per second (2.4 teraflops). Supercomputers are used for a wide range of tasks from scientific research to customer support and product development. They're often used to perform the intricate calculations required to create animated motion pictures. And they help oil companies in their search for oil by analyzing massive amounts of data (Stair, 1999).

The **microcomputer** was developed to offer inexpensive computation capability to individual users in the late 1970s. Early models featured a revolutionary amount of memory: 64K. Their physical size was smaller than the minicomputers of that time, though larger than the microcomputers of today. Eventually, microcomputers grew to accommodate software with larger capacity and greater speed. The distinguishing characteristic of the first microcomputer was its single-user status.

Powerful microcomputers developed for use by commercial, educational, and government enterprises are called **workstations**. Typically, workstations are networked together and are used to support engineering and technical users who perform massive mathematical computations or computer-aided design (CAD), or use other applications requiring very powerful CPUs, large amounts of main memory, and extremely high-resolution graphic displays to meet their needs.

✓ **HP-UX, Sun Solaris, and Macintosh OS X are only three of many operating systems based on UNIX.**

Servers are powerful computers that provide specialized services to other computers on client/server networks. Examples can include print servers, Internet servers, e-mail servers, etc. Each performs critical network tasks. For instance, a file server, usually a powerful computer with substantial file storage capacity (such as a large collection of hard drives), manages file storage and retrieval for other computers, called clients, on the network.

(table 1.1)

A brief list of platforms and sample operating systems listed in alphabetical order.

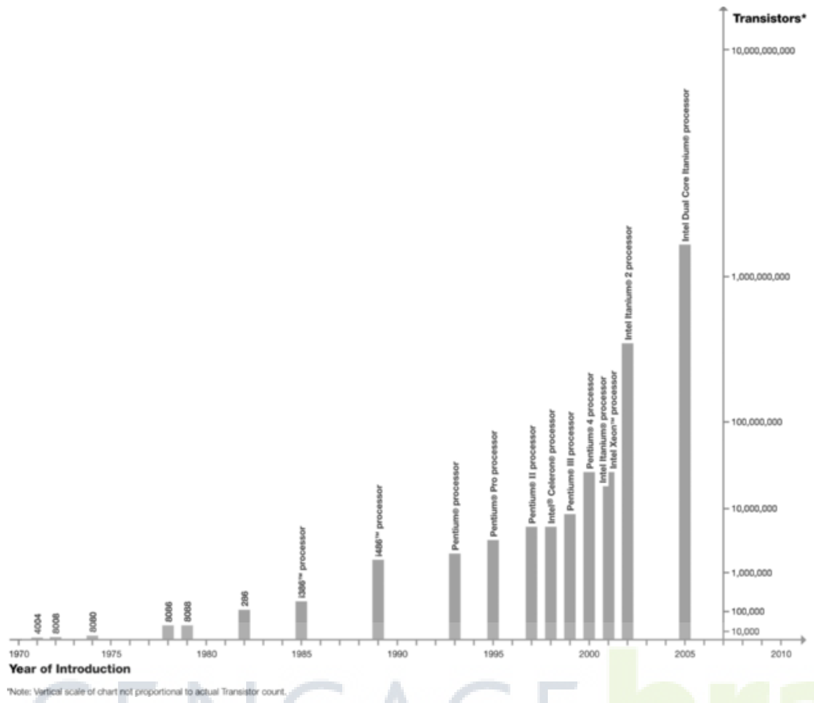
Platform	Operating System
Microcomputers	Linux, UNIX (includes Mac), Windows
Mainframe computers	IBM z/390, Linux, UNIX
Supercomputers	IRIX, Linux, UNICOS
Workstations, servers	Linux, UNIX, Windows
Networks	Linux, NetWare, UNIX, Windows
Personal digital assistants	BlackBerry, Linux, Palm OS, Windows Mobile

Some typical operating systems for a wide variety of platforms are shown in Table 1.1. Since the mid-1970s, rapid advances in computer technology have blurred the distinguishing characteristics of early machines: physical size, cost, and memory capacity. The most powerful mainframes today have multiple processors coordinated by the Processor Manager. Simple mainframes still have a large main memory, but now they're available in desk-sized cabinets.

Networking is an integral part of modern computer systems because it can connect workstations, servers, and peripheral devices into integrated computing systems. Networking capability has become a standard feature in many computing devices: personal organizers, personal digital assistants (PDAs), cell phones, and handheld Web browsers.

At one time, computers were classified by memory capacity; now they're distinguished by processor capacity. We must emphasize that these are relative categories and what is large today will become medium-sized and then small sometime in the near future.

In 1965, Intel executive Gordon Moore observed that each new processor chip contained roughly twice as much capacity as its predecessor, and each chip was released within 18–24 months of the previous chip. He predicted that the trend would cause computing power to rise exponentially over relatively brief periods of time. Now known as Moore's Law, shown in Figure 1.6, the trend has continued and is still remarkably accurate. The Intel 4004 chip in 1971 had 2,300 transistors while the Pentium II chip 20 years later had 7.5 million, and the Pentium 4 Extreme Edition processor introduced in 2004 had 178 million transistors. Moore's Law is often used by industry observers to make their chip capacity forecasts.



(figure 1.6)

Demonstration of Moore's Law. Gordon Moore's 1965 prediction has held up for more than three decades.

Copyright © 2005 Intel Corporation

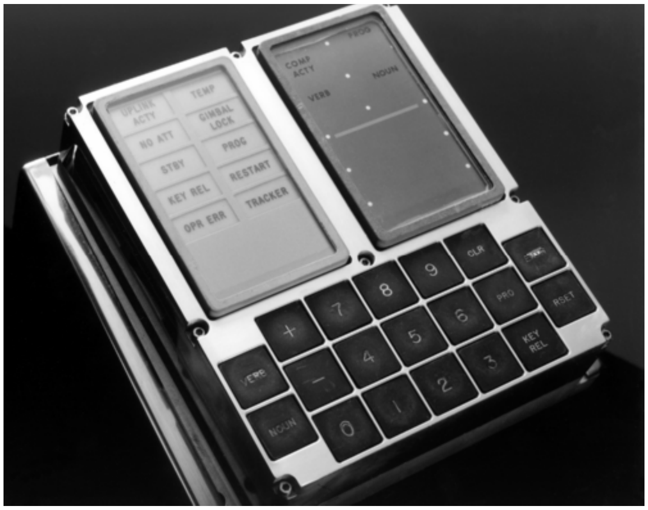
Types of Operating Systems

Operating systems for computers large and small fall into five categories distinguished by response time and how data is entered into the system: batch, interactive, real-time, hybrid, and embedded systems.

Batch systems date from the earliest computers, when they relied on stacks of punched cards or reels of magnetic tape for input. Jobs were entered by assembling the cards into a deck and running the entire deck of cards through a card reader as a group—a batch. The efficiency of a batch system is measured in **throughput**—the number of jobs completed in a given amount of time (for example, 550 jobs per hour).

Interactive systems give a faster turnaround than batch systems but are slower than the real-time systems we talk about next. They were introduced to satisfy the demands of users who needed fast turnaround when debugging their programs. The operating system required the development of time-sharing software, which would allow each user to interact directly with the computer system via commands entered from a type-writer-like terminal. The operating system provides immediate feedback to the user and response time can be measured in fractions of a second.

Real-time systems are used in time-critical environments where reliability is key and data must be processed within a strict time limit. The time limit need not be ultra-fast


(figure 1.7)

The state-of-the-art computer interface box for the Apollo spacecraft in 1968. The guidance computer had few moving parts and no vacuum tubes, making it both rugged and compact.

Courtesy of NASA

(though it often is), but system response time must meet the deadline or risk significant consequences. These systems also need to provide contingencies to fail gracefully—that is, preserve as much of the system's capabilities and data as possible to facilitate recovery. For example, real-time systems are used for space flights (as shown in Figure 1.7), airport traffic control, fly-by-wire aircraft, critical industrial processes, certain medical equipment, and telephone switching, to name a few.

There are two types of real-time systems depending on the consequences of missing the deadline:

- Hard real-time systems risk total system failure if the predicted time 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 unpredictability of these systems can't provide the guaranteed response times that real-time performance requires (Dougherty, 1995). Therefore, most embedded systems and real-time environments require operating systems that are specially designed to meet real-time needs.

Hybrid systems are a combination of batch and interactive. They appear to be interactive because individual users can access the system and get fast responses, but such a system actually accepts and runs 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. Many large computer systems are hybrids.

Embedded systems are computers placed inside other products to add features and capabilities. For example, you find embedded computers in household appliances, automobiles, digital music players, elevators, and pacemakers. In the case of automobiles, embedded computers can help with engine performance, braking, and navigation. For example, several projects are under way to implement "smart roads," which would alert drivers in cars equipped with embedded computers to choose alternate routes when traffic becomes congested.

✓ **One example of a software product to help developers build an embedded system is Windows Automotive.**

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 efficient and take advantage of the computer's limited resources, such as memory, to their maximum.

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 components, from the entire operating system, are needed in that particular environment. The final version of this operating system will include only the necessary elements; any unneeded features or functions will be 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 will have potential in this market.

Brief History of Operating System Development

The evolution of operating system software parallels the evolution of the computer hardware it was designed to control. Here's a very brief overview of this evolution.

1940s

The first generation of computers (1940–1955) was a time of vacuum tube technology and computers the size of classrooms. Each computer was unique in structure and purpose. There was little need for standard operating system software because each computer's use was restricted to a few professionals working on mathematical, scientific, or military applications, all of whom were familiar with the idiosyncrasies of their hardware.

A typical program would include every instruction needed by the computer to perform the tasks requested. It would give explicit directions to the card reader (when to begin, how to interpret the data on the cards, when to end), the CPU (how and where to store the instructions in memory, what to calculate, where to find the data, where to send

(figure 1.8)

Dr. Grace Hopper's research journal from her work on Harvard's Mark I computer in 1945 included the remains of the first computer "bug," a moth that had become trapped in the computer's relays causing the system to crash. Today's use of the term "bug" stems from that first moth.

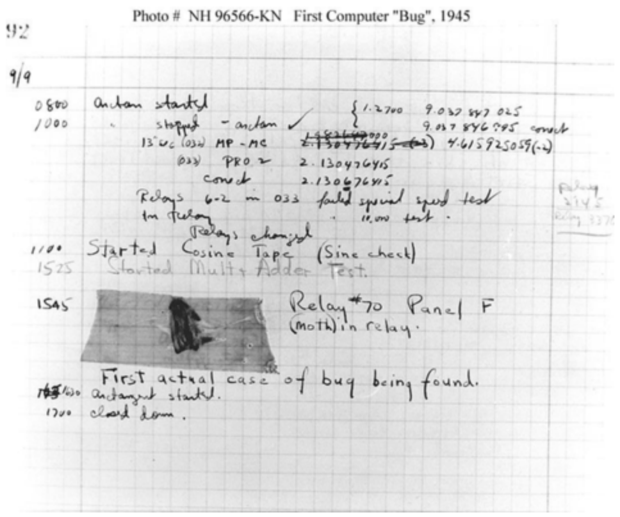


Photo # NH 96566-KN First Computer "Bug", 1945

92

9/9

```
0800   antan started
1000     "   stopped - antan ✓      {1.2700    9.037 847 025
                                    {          9.037 846 995 conect
       13"UC (032) MP-MC   ~~1.982147000~~
                           ~~2.130476415 (-3)~~  4.615925059(-2)
            (033) PRO 2    2.130476415
                  conect   2.130676415
       Relays 6-2 in 033 failed special speed test
       in relay              "  11,000 test .
                 Relays changed
1100   Started Cosine Tape (Sine check)
1525   Started Mult + Adder Test.
1545                       Relay #70 Panel F
                           (moth) in relay.
          First actual case of bug being found.
1630   antangent started.
1700   closed down.
```

Relay 2145
Relay 3370

the output), and the output device (when to begin, how to print out the finished product, how to format the page, and when to end).

The machines were operated by the programmers from the main console—it was a hands-on process. In fact, to debug a program, the programmer would stop the processor, read the contents of each register, make the corrections in memory locations, and then resume operation. The first bug was a moth trapped in a Harvard computer that caused it to fail, as shown in Figure 1.8.

To run programs, the programmers would have to reserve the machine for the length of time they estimated it would take the computer to execute the program. As a result, the machine was poorly utilized. The CPU processed data and made calculations for only a fraction of the available time and, in fact, the entire system sat idle between reservations.

In time, computer hardware and software became more standard and the execution of a program required fewer steps and less knowledge of the internal workings of the computer. Compilers and assemblers were developed to translate into binary code the English-like commands of the evolving high-level languages.

Rudimentary operating systems started to take shape with the creation of macros, library programs, standard subroutines, and utility programs. And they included device driver subroutines—prewritten programs that standardized the way input and output devices were used.

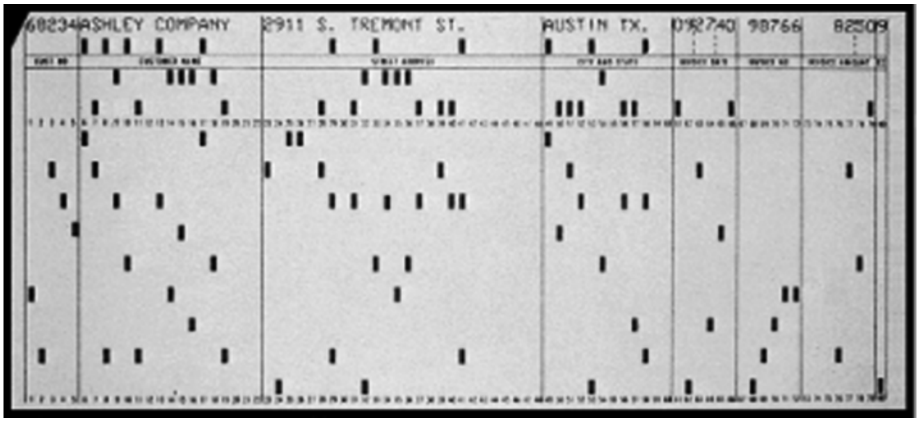
These early programs were at a significant disadvantage because they were designed to use their resources conservatively at the expense of understandability. That meant that many programs used convoluted logic that only the original programmer could understand, so it was nearly impossible for anyone else to debug or change the program later on.

1950s

Second-generation computers (1955–1965) were developed to meet the needs of new markets—government and business researchers. The business environment placed much more importance on the cost effectiveness of the system. Computers were still very expensive, especially when compared to other office equipment (the IBM 7094 was priced at $200,000). Therefore, throughput had to be maximized to make such an investment worthwhile for business use, which meant dramatically increasing the usage of the system.

Two improvements were widely adopted: Computer operators were hired to facilitate each machine's operation, and job scheduling was instituted. Job scheduling is a productivity improvement scheme that groups together programs with similar requirements. For example, several FORTRAN programs would be run together while the FORTRAN compiler was still resident in memory. Or all of the jobs using the card reader for input might be run together, and those using the tape drive would be run later. Some operators found that a mix of I/O device requirements was the most efficient combination. That is, by mixing tape-input programs with card-input programs, the tapes could be mounted or rewound while the card reader was busy. A typical punch card is shown in Figure 1.9.

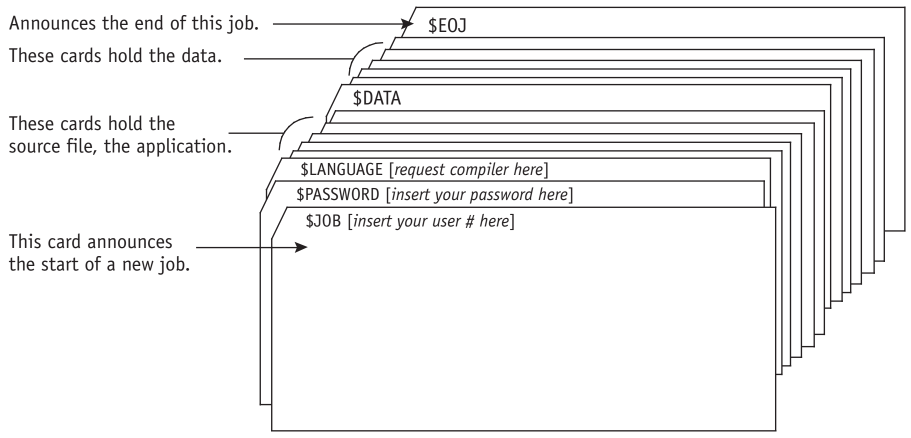
Job scheduling introduced the need for control cards, which defined the exact nature of each program and its requirements, illustrated in Figure 1.10. This was one of the first uses of a job control language, which helped the operating system coordinate and manage the system resources by identifying the users and their jobs and specifying the resources required to execute each job.



(figure 1.9)

Each letter or number printed along the top of the punch card is represented by a unique combination of holes beneath it.

From ibm.com


Announces the end of this job.
These cards hold the data.
These cards hold the source file, the application.
This card announces the start of a new job.
$EOJ
$DATA
$LANGUAGE [*request compiler here*]
$PASSWORD [*insert your password here*]
$JOB [*insert your user # here*]

(figure 1.10)

The Job Control Language (called JCL) program structure and the order of punch cards for the DEC-10 computer.

But even with batching techniques, the faster second-generation computers allowed expensive time lags between the CPU and the I/O devices. For example, a job with 1600 cards could take 79 seconds to be read by the card reader and only 5 seconds of CPU time to assemble or compile. That meant the CPU was idle 94 percent of the time and busy only 6 percent of the time it was dedicated to that job—an inefficiency that resulted in poor overall system use.

Eventually, several factors helped improve the performance of the CPU:

- First, the speeds of I/O devices such as drums, tape drives, and disks gradually increased.
- Second, to use more of the available storage area in these devices, records were grouped into blocks before they were retrieved or stored. (This is called blocking, meaning that several logical records are grouped within one physical record, and is discussed in detail in Chapter 7.)
- Third, to reduce the discrepancy in speed between the I/O and the CPU, an interface called the control unit was placed between them to act as a buffer. A buffer is an interim storage area that works as a temporary holding place. As the slow input device reads one record, the control unit places each character of the record into the buffer. When the buffer is full, the entire record is quickly transmitted to the CPU. The process is just the opposite for output devices: The CPU places the entire record into the buffer, which is then passed on by the control unit at the slower rate required by the output device.

  The buffers of this generation were conceptually similar to those now used routinely by Internet browsers to make video and audio playback smoother, as shown in Figure 1.11.

  If a control unit had more than one buffer, the I/O process could be made even faster. For example, if the control unit had two buffers, the second buffer could be loaded while the first buffer was transmitting its contents to or from the CPU. Ideally, by the time the first was transmitted, the second was ready to go, and so on. In this way, input or output time was cut in half.

(figure 1.11)

Three typical browser buffering progress indicators.

- Fourth, in addition to buffering, an early form of spooling was developed by moving offline the operations of card reading, printing, and "punching." For example, incoming jobs would be transferred from card decks to reels of magnetic tape offline. Then they could be read into the CPU from the tape at a speed much faster than that of the card reader. The spooler worked the same way as a buffer but, in this example, it was a separate offline device while a buffer was part of the main computer hardware.

Also during the second generation, techniques were developed to manage program libraries, create and maintain each data direct access address, and create and check file labels. Timer interrupts were developed to allow job sharing and to prevent infinite loops on programs that were mistakenly instructed to execute a single series of commands forever. Because a fixed amount of execution time was allocated to each program when it entered the system, and was monitored by the operating system, programs that were still running when the time expired were terminated.

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

1960s

Third-generation computers date from the mid-1960s. They were designed with faster CPUs, but their speed still caused problems when they interacted with printers and other I/O devices that ran at slower speeds. The solution was **multiprogramming**, which introduced the concept of loading many programs at one time and sharing the attention of a single CPU.

The first multiprogramming systems allowed each program to be serviced in turn, one after another. The most common mechanism for implementing multiprogramming was the introduction of the concept of the interrupt, whereby the CPU was notified of events needing operating system services. For example, when a program issued a print command (called an input/output command or an I/O command), it generated an interrupt requesting the services of the I/O processor and the CPU was released to begin execution of the next job. This was called *passive multiprogramming* because the operating system didn't control the interrupts but waited for each job to end an execution sequence. It was less than ideal because if a job was CPU-bound (meaning that it performed a great deal of nonstop CPU processing before issuing an interrupt), it could tie up the CPU for a long time while all other jobs had to wait.

To counteract this effect, the operating system was soon given a more active role with the advent of *active multiprogramming*, which allowed each program to use only a preset slice of CPU time, which is discussed in Chapter 4. When time expired, the job was interrupted and another job was allowed to begin execution. The interrupted job had to wait until it was allowed to resume execution later. The idea of time slicing soon became common in many time-sharing systems.

Program scheduling, which was begun with second-generation systems, continued at this time but was complicated by the fact that main memory was occupied by many jobs. To solve this problem, the jobs were sorted into groups and then loaded into memory according to a preset rotation formula. The sorting was often determined by priority or memory requirements—whichever was found to be the most efficient use of the available resources. In addition to scheduling jobs, handling interrupts, and allocating memory, the operating systems also had to resolve conflicts whenever two jobs requested the same device at the same time, something we will explore in Chapter 5.

Even though there was progress in processor management, few major advances were made in data management.

1970s

After the third generation, during the late 1970s, computers had faster CPUs, creating an even greater disparity between their rapid processing speed and slower I/O access time. The first Cray supercomputer was released in 1976. Multiprogramming schemes to increase CPU use were limited by the physical capacity of the main memory, which was a limited resource and very expensive.

A solution to this physical limitation was the development of virtual memory, which took advantage of the fact that the CPU could process only one instruction at a time. With virtual memory, the entire program didn't need to reside in memory before execution could begin. A system with virtual memory would divide the programs into parts and keep them in secondary storage, bringing each part into memory only as it was needed. (Programmers of second-generation computers had used this concept with the roll in/roll out programming method, also called overlay, to execute programs that exceeded the physical memory of those computers.)

At this time there was also growing attention to the need for data resource conservation. Database management software became a popular tool because it organized data in an integrated manner, minimized redundancy, and simplified updating and access of data. A number of query systems were introduced that allowed even the novice user to retrieve specific pieces of the database. These queries were usually made via a terminal, which in turn mandated a growth in terminal support and data communication software.

Programmers soon became more removed from the intricacies of the computer, and application programs started using English-like words, modular structures, and standard operations. This trend toward the use of standards improved program management because program maintenance became faster and easier.

1980s

Development in the 1980s dramatically improved the cost/performance ratio of computer components. Hardware was more flexible, with logical functions built on easily replaceable circuit boards. And because it was less costly 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 job of the programmer, as it had been defined in previous years, changed dramatically because many programming functions were being carried out by the system's software, hence making the programmer's task simpler and less hardware dependent.

Eventually the industry moved to **multiprocessing** (having more than one processor), and more complex languages were designed to coordinate the activities of the multiple processors servicing a single job. As a result, it became possible to execute programs in parallel, and eventually operating systems for computers of every size were routinely expected to accommodate multiprocessing.

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.

With network operating systems, users generally became aware of the existence of many networked resources, could log in to remote locations, and could manipulate files on networked computers distributed over a wide geographical area. Network operating systems were similar to single-processor operating systems in that each machine ran its own local operating system and had its own users. The difference was in the addition of a network interface controller with low-level software to drive the local operating system, as well as programs to allow remote login and remote file access. Still, even with these additions, the basic structure of the network operating system was quite close to that of a standalone system.

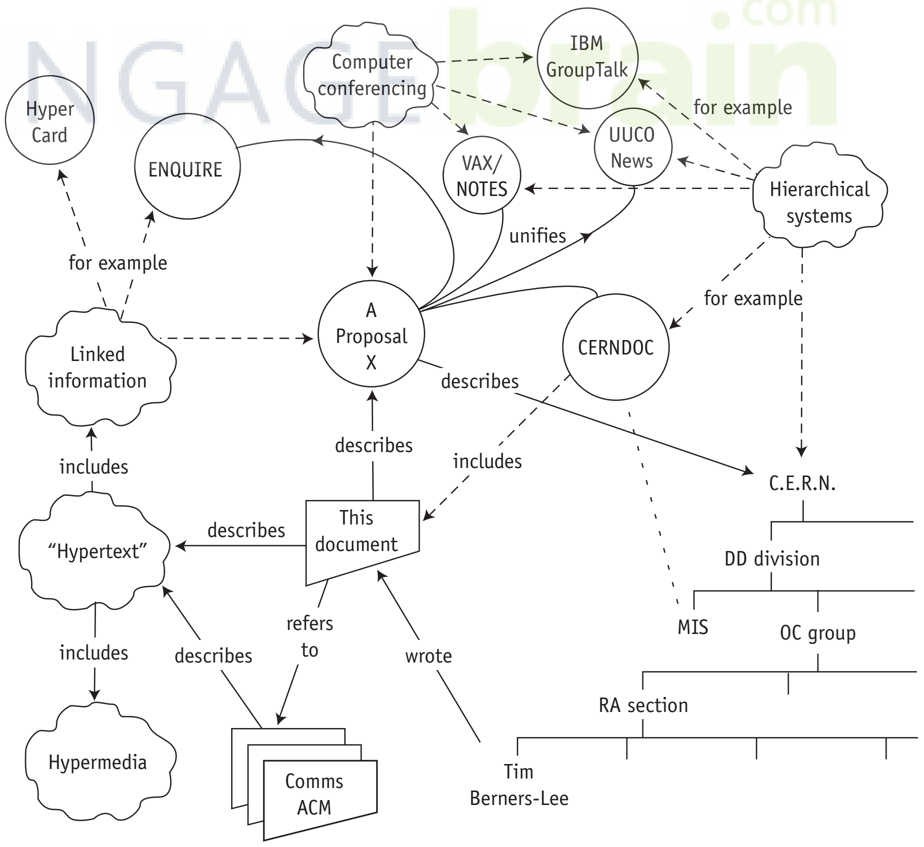
On the other hand, with distributed operating systems, users could think they were working with a typical uniprocessor system when in fact they were connected to a cluster of many processors working closely together. With these systems, users didn't need to know which processor was running their applications or which devices were storing their files. These details were all handled transparently by the operating system—something that required more than just adding a few lines of code to a uniprocessor operating system. The disadvantage of such a complex operating system was the requirement for more complex processor-scheduling algorithms. In addition, communications delays within the network sometimes meant that scheduling algorithms had to operate with incomplete or outdated information.

1990s

The overwhelming demand for Internet capability in the mid-1990s sparked the proliferation of networking capability. The World Wide Web, conceived in a paper, shown in Figure 1.12, by Tim Berners-Lee made the Internet accessible by computer users

(figure 1.12)

Illustration from the first page of the 1989 proposal by Tim Berners-Lee describing his revolutionary "linked information system." Based on this research, he designed the first World Wide Web server and browser, making it available to the general public in 1991.



worldwide, not just the researchers who had come to depend on it for global communications. Web accessibility and e-mail became standard features of almost every operating system. However, increased networking also sparked increased demand for tighter security to protect hardware and software.

The decade also introduced a proliferation of multimedia applications demanding additional power, flexibility, and device compatibility for most operating systems. A typical multimedia computer houses devices to perform audio, video, and graphic creation and editing. Those functions can require many specialized devices such as a microphone, digital piano, Musical Instrument Digital Interface (MIDI), digital camera, digital video disc (DVD) drive, optical disc (CD) drives, speakers, additional monitors, projection devices, color printers, and high-speed Internet connections. These computers also require specialized hardware (such as controllers, cards, busses) and software to make them work together properly.

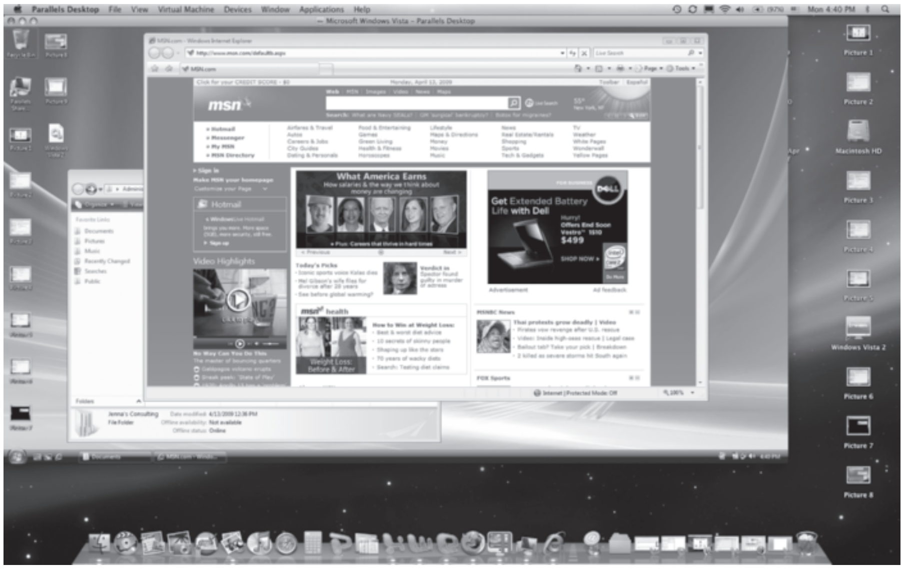
Multimedia applications need large amounts of storage capability that must be managed gracefully by the operating system. For example, each second of a 30-frame-per-minute full-screen video requires 27MB of storage unless the data is compressed in some way. To meet the demand for compressed video, special-purpose chips and video boards have been developed by hardware companies.

What's the effect of these technological advances on the operating system? Each advance requires a parallel advance in the software's management capabilities.

2000s

The new century emphasized the need for operating systems to offer improved flexibility, reliability, and speed. To meet the need for computers that could accommodate multiple operating systems running at the same time and sharing resources, the concept of virtual machines, shown in Figure 1.13, was developed and became commercially viable.

Virtualization is the creation of partitions on a single server, with each partition supporting a different operating system. In other words, it turns a single physical server into multiple virtual servers, often with multiple operating systems. Virtualization requires the operating system to have an intermediate manager to oversee each operating system's access to the server's physical resources. For example, with virtualization, a single processor can run 64 independent operating systems on workstations using a processor capable of allowing 64 separate threads (instruction sequences) to run at the same time.



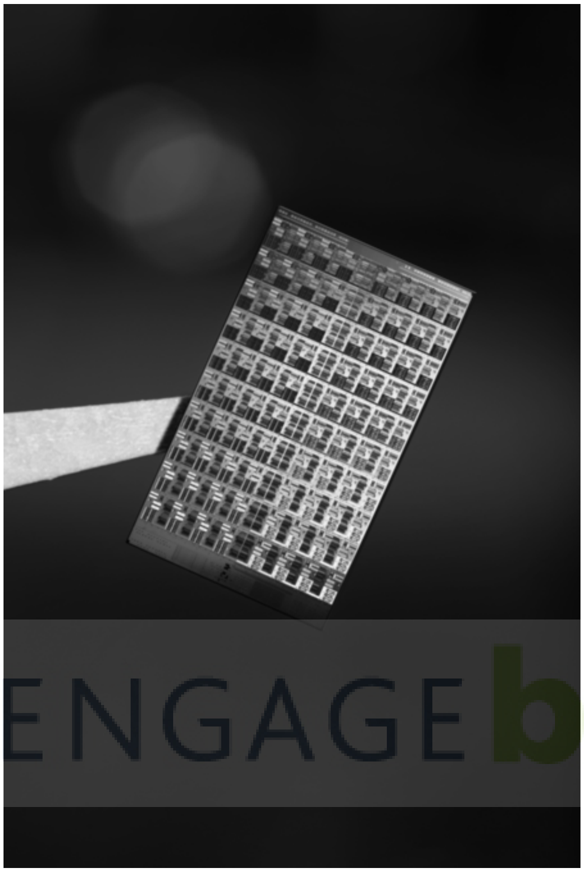
(figure 1.13)

With virtualization, different operating systems can run on a single computer.

Courtesy of Parallels, Inc.

Processing speed has enjoyed a similar advancement with the development of multi-core processors, shown in Figure 1.14. Until recent years, the silicon wafer that forms the base of the computer chip circuitry held only a single CPU. However, with the introduction of dual-core processors, a single chip can hold multiple processor **cores**. Thus, a dual-core chip allows two sets of calculations to run at the same time, which sometimes leads to faster completion of the job. It's as if the user has two separate computers, and two processors, cooperating on a single task. As of this writing, designers have created chips that can hold 80 simple cores.

Does this hardware innovation affect the operating system software? Absolutely, because it must now manage the work of these multiple processors and be able to schedule and manage the processing of their multiple tasks. We'll explore some of the complexities of this in Chapter 6.



(Figure 1.14)

A single piece of silicon can hold 80 cores, which (to put it in simplest terms) can perform 80 calculations at one time.

Courtesy of Intel Corporation

Threads

Multi-core technology helps the operating system handle threads, multiple actions that can be executed at the same time. First, an explanation: The Processor Manager is responsible for processing each job submitted by a user. Jobs are made up of processes (sometimes called tasks in other textbooks), and processes consist of multiple threads.

A process has two characteristics:

- It requires space in main memory where it resides during its execution; although, from time to time, it requires other resources such as data files or I/O devices.
- It passes through several states (such as running, waiting, ready) from its initial arrival into the computer system to its completion.

Multiprogramming and virtual memory dictate that processes be swapped between main memory and secondary storage during their execution. With conventional processes (also known as heavyweight processes), this swapping results in a lot of overhead. That's because each time a swap takes place, all process information must be saved to preserve the process's integrity.

A **thread** (or lightweight process) can be defined as a unit smaller than a process, which can be scheduled and executed. Using this technique, the heavyweight process, which owns the resources, becomes a more passive element, while a thread becomes the element that uses the CPU and is scheduled for execution. Manipulating threads is less time consuming than manipulating processes, which are more complex. Some operating systems support multiple processes with a single thread, while others support multiple processes with multiple threads.

✓ **Web browsers routinely use multithreading to allow users to explore multiple areas of interest on the Internet at the same time.**

Multithreading allows applications to manage a separate process with several threads of control. Web browsers use multithreading routinely. For instance, one thread can retrieve images while another sends and retrieves e-mail. Multithreading is also used to increase responsiveness in a time-sharing system to increase resource sharing and decrease overhead.

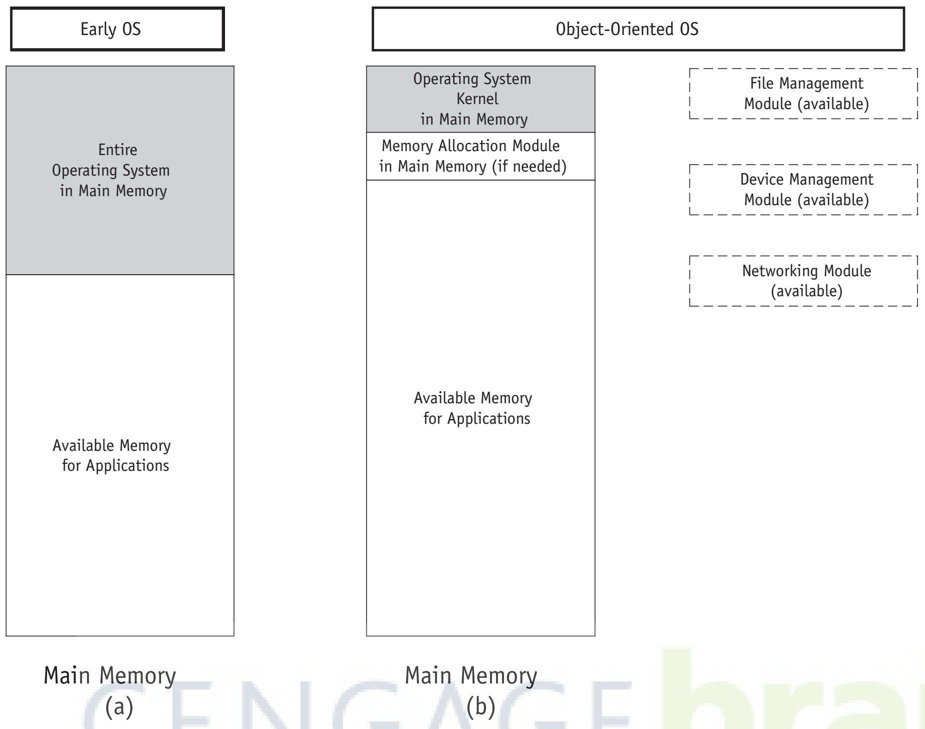
Object-Oriented Design

An important area of research that resulted in substantial efficiencies was that of the system architecture of operating systems—the way their components are programmed and organized, specifically the use of **object-oriented** design and the reorganization of the operating system's nucleus, the kernel. The kernel is the part of the operating system that resides in memory at all times, performs the most essential operating system tasks, and is protected by hardware from user tampering.

The first operating systems were designed as a comprehensive single unit, as shown in Figure 1.15 (a). They stored all required elements of the operating system in memory such as memory allocation, process scheduling, device allocation, and file management. This type of architecture made it cumbersome and time consuming for programmers to add new components to the operating system, or to modify existing ones.

Most recently, the part of the operating system that resides in memory has been limited to a few essential functions, such as process scheduling and memory allocation, while all other functions, such as device allocation, are provided by special modules, which are treated as regular applications, as shown in Figure 1.15 (b). This approach makes it easier to add new components or modify existing ones.

Object-oriented design was the driving force behind this new organization. Objects are self-contained modules (units of software) that provide models of the real world and can be reused in different applications. By working on objects, programmers can modify and customize pieces of an operating system without disrupting the integrity of the remainder of the system. In addition, using a modular, object-oriented approach can


Early OS
Entire Operating System in Main Memory
Available Memory for Applications
Main Memory
(a)
Object-Oriented OS
Operating System Kernel in Main Memory
Memory Allocation Module in Main Memory (if needed)
Available Memory for Applications
File Management Module (available)
Device Management Module (available)
Networking Module (available)
Main Memory
(b)

(figure 1.15)

Early operating systems (a) loaded in their entirety into main memory. Object-oriented operating systems (b) load only the critical elements into main memory and call other objects as needed.

make software development groups more productive than was possible with procedural structured programming.

Conclusion

In this chapter, we looked at the overall function of operating systems and how they have evolved to run increasingly complex computers and computer systems; but like any complex subject, there's much more detail to explore. As we'll see in the remainder of this text, there are many ways to perform every task and it's up to the designer of the operating system to choose the policies that best match the system's environment.

In the following chapters, we'll explore in detail how each portion of the operating system works, as well as its features, functions, benefits, and costs. 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 system developed for the earliest computers that used punched cards or tape for input, which were entered in a batch.

central processing unit (CPU): the component with the circuitry, the "chips," to control the interpretation and execution of instructions.

core: the processing part of a CPU chip made up of the control unit and the arithmetic logic unit (ALU).

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, often small and fast, that resides in a larger physical system such as jet aircraft or ships.

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 read-only memory (ROM).

hardware: the physical 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 via commands entered from a keyboard.

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: 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 or internal memory.

mainframe: the historical name given to a large computer system characterized by its large size, high cost, and high performance.

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 needed to execute the job.

microcomputer: a small computer equipped with all the hardware and software necessary to perform one or more tasks.

minicomputer: a small to medium-sized computer system, also called a midrange computer.

multiprocessing: when two or more CPUs share the same main memory, most I/O devices, and the same control program routines. They service the same job stream and execute distinct processing programs concurrently.

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

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

object-oriented: a programming philosophy whereby programs consist of self-contained, reusable modules called objects, each of which supports a specific function, but which are categorized into classes of objects that share the same function.

operating system: the software that manages all the resources of a computer system.

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

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

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

software: a collection of programs used to perform certain tasks. Software falls into three main categories: operating system programs, compilers and assemblers, and application programs.

storage: a place where data is stored in the computer system. Primary storage is main memory and secondary storage is nonvolatile media.

supercomputer: the fastest, most sophisticated computers made, used for complex calculations.

thread: a portion of a program that can run independently of other portions. Multithreaded application programs can have several threads running at one time with the same or different priorities.

throughput: a composite measure of a system's efficiency that counts the number of jobs served in a given unit of time.

virtualization: the creation of a virtual version of hardware or software. Operating system virtualization allows a single CPU to run multiple operating system images at the same time.

workstation: a desktop computer attached to a local area network that serves as an access point to that network.

Interesting Searches

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

- Computer History Museum
- NASA - Computers Aboard the Space Shuttle
- IBM Computer History Archive
- History of the UNIX Operating System
- History of Microsoft Windows Products

Exercises

Research Topics

Whenever you research computer technology, make sure your resources are timely. Notice the date when the research was published. Also be sure to validate the authenticity of your sources. Avoid any that might be questionable, such as blogs and publicly edited online (wiki) sources.

A. Write a one-page review of an article about operating systems that appeared in a recent computing magazine or academic journal. Be sure to cite your source. Give a summary of the article, including the primary topic, the information presented, and the author's conclusion. Give your personal evaluation of the article, including the author's writing style, inappropriate use of jargon, topics that made the article interesting to you, and its relevance to your own experiences.

B. Research the Internet or current literature to identify an operating system that runs a cell phone or handheld computer. (These are generally known as mobile operating systems.) List the key features of the operating system and the hardware it is designed to run. Cite your sources.

Exercises

1. Name five current operating systems (not mentioned in this chapter) and the computers or configurations each operates.
2. Name the five key concepts about an operating system that you think a novice user needs to know and understand.

3. Explain the impact of the evolution of computer hardware and the accompanying evolution of operating system software.
4. In your opinion, has Moore's Law been a mere predictor of chip design, or a motivator for chip designers? Explain your answer.
5. Explain the fundamental differences between interactive, batch, real-time, and embedded systems.
6. List three situations that might demand a real-time operating system and explain why.
7. Give an example of an organization that might find batch-mode processing useful and explain why.
8. List three tangible (physical) data storage resources of a typical computer system. Explain the advantages and disadvantages of each.
9. Briefly compare active and passive multiprogramming.
10. Give at least two reasons why a multi-state bank might decide to buy six server computers instead of one more powerful computer. Explain your answer.
11. Select one 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 such a person might use a handheld computer to work more efficiently.

Advanced Exercises

12. Compare the design goals and evolution of two operating systems described in Chapters 13–16 of this text.
13. 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 clicking the mouse and conclude with the display of the result on the monitor.
14. Identify the clock rates of processors that use (or used) 8 bits, 16 bits, 32 bits, and 64 bits. Discuss several implications involved in scheduling the CPU in a multiprocessing system using these processors.
15. In a multiprogramming and time-sharing environment, several users share the system simultaneously. 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 can in a dedicated machine? Explain your answers.
16. Give an example of an application where multithreading gives improved performance over single-threading.
17. If a process terminates, will its threads also terminate or will they continue to run? Explain your answer.
18. If a process is suspended (put into the "wait" state by an interrupt), will its threads also be suspended? Explain your answer and give an example.