10.1 Roles of an Operating System

In Chapter 1, we talked about the changing role of the programmer. As early as the end of the first generation of software development, there was a split between those programmers who wrote tools to help other programmers and those who used the tools to solve problems. Modern software can be divided into two categories, application software and system software, reflecting this separation of goals. Application software is written to address specific needs—to solve problems in the real world. Word processing programs, games, inventory control systems, automobile diagnostic programs, and missile guidance programs are all application software. Chapters 12 through 14 discuss areas of computer science that relate to application software.

Application software Programs that help us solve real-world problems

System software Programs that manage a computer system and interact with hardware

Operating system System software that manages computer resources and provides an interface for system interaction

System software manages a computer system at a more fundamental level. It provides the tools and an environment in which application software can be created and run. System software often interacts directly with the hardware and provides more functionality than the hardware itself does.

The operating system of a computer is the core of its system software. An operating system manages computer resources, such as memory and input/output devices, and provides an interface through which a human can interact with the computer. Other system software supports specific application goals, such as a library of graphics software that renders images on a display. The operating system allows an application program to interact with these other system resources.

Figure 10.1 shows the operating system in its relative position among computer system elements. The operating system manages hardware resources. It allows application software to access system resources, either directly or through other system software. It provides a direct user interface to the computer system.

A computer generally has one operating system that becomes active and takes control when the system is turned on. Computer hardware is wired to initially load a small set of system instructions that is stored in permanent memory (ROM). These instructions load a larger portion of system software from secondary memory, usually a magnetic disk. Eventually all key elements of the operating system software are loaded, start-up programs are executed, the user interface is activated, and the system is ready for use. This activity is often called *booting* the computer. The term "boot" comes from the idea of "pulling yourself up by your own bootstraps," which is essentially what a computer does when it is turned on.

A computer could have two or more operating systems from which the user chooses when the computer is turned on. This configuration is often

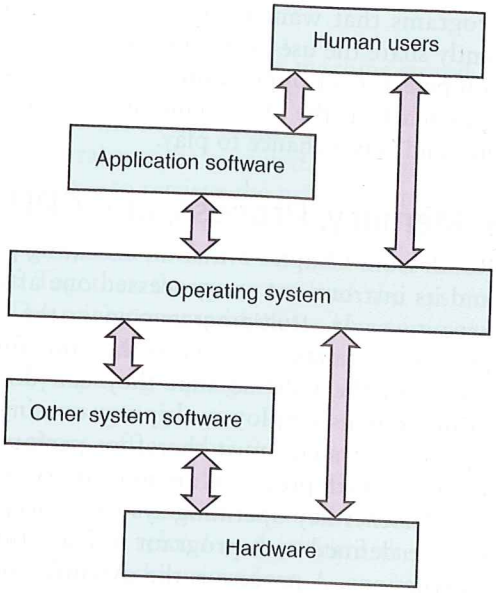

FIGURE 10.1 An operating system interacts with many aspects of a computer system.

called a *dual-boot* or *multi-boot* system. Only one operating system controls the computer at any given time, however.

You've likely used at least one operating system before. The various versions of Microsoft® Windows® (Windows NT, Windows XP, Windows Vista, Windows 7) are popular choices for personal computers. The different versions of these operating systems indicate how the software evolves over time as well as how changes occur in the way services are provided and managed. The Mac OS family is the operating system of choice for computers manufactured by Apple® Computer. UNIX has been a favorite of serious programmers for years, and a version of UNIX called Linux is popular for personal computer systems.

Any given operating system manages resources in its own particular way. Our goal in this chapter is not to nitpick about the differences among operating systems, but rather to discuss the ideas common to all of them. We occasionally refer to the methods that a specific OS (operating system) uses, and we discuss some of their individual philosophies. In general, however, we focus on the underlying concepts.

The various roles of an operating system generally revolve around the idea of "sharing nicely." An operating system manages resources, and these resources are often shared in one way or another among the various

?

Who is Blake Ross?

Blake Ross was designing Web pages by the age of 10. By 14, he was fixing bugs in Netscape's Web browser as a hobby. Before finishing high school, he helped develop Firefox®, an open-source Web browser. America Online® created a not-for-profit foundation to continue the development of Firefox®, which was officially released in November 2004. While in college, Ross continued fixing bugs in the browser. In 2005, he was nominated for *Wired* magazine's Renegade of the Year award. Ross joined up with another ex-Netscape employee to create Parakey, a new user interface designed to bridge the gap between the desktop and the Web. In 2007, Parakey was purchased by Facebook.

programs that want to use them. Multiple programs executing concurrently share the use of main memory. They take turns using the CPU. They compete for an opportunity to use input/output devices. The operating system acts as the playground monitor, making sure that everyone cooperates and gets a chance to play.

Memory, Process, and CPU Management

Recall from Chapter 5 that an executing program resides in main memory and its instructions are processed one after another in the fetch–decode–execute cycle. Multiprogramming is the technique of keeping multiple programs in main memory at the same time; these programs compete for access to the CPU so that they can do their work. All modern operating systems employ multiprogramming to one degree or another. An operating system must therefore perform memory management to keep track of which programs are in memory and where in memory they reside.

Another key operating system concept is the idea of a process, which can be defined as a program in execution. A program is a static set of instructions. A process is the dynamic entity that represents the program while it is being executed. Through multiprogramming, a computer system might have many active processes at once. The operating system must manage these processes carefully. At any point in time a specific instruction may be the next to be executed. Intermediate values have been calculated. A process might be interrupted during its execution, so the operating system performs process management to carefully track the progress of a process and all of its intermediate states.

Related to the ideas of memory management and process management is the need for CPU scheduling, which determines which process in memory is executed by the CPU at any given point.

Memory management, process management, and CPU scheduling are the three main topics discussed in this chapter. Other key operating system topics, such as file management and secondary storage, are covered in Chapter 11.

Keep in mind that the OS is itself just a program that must be executed. Operating system processes must be managed and maintained in main memory along with other system software and application programs. The OS executes on the same CPU as the other programs, and it must take its turn among them.

Before we delve into the details of managing resources such as main memory and the CPU, we need to explore a few more general concepts.

- **Multiprogramming** The technique of keeping multiple programs in main memory at the same time, competing for the CPU
- **Memory management** The act of keeping track of how and where programs are loaded in main memory
- **Process** The dynamic representation of a program during execution
- **Process management** The act of keeping track of information for active processes
- **CPU scheduling** The act of determining which process in memory is given access to the CPU so that it may execute

Influential computing jobs

There were many influential jobs in computing in the 1960s, but none more so than the computer operator. In his or her hands rested the decision of whose computer jobs ran and when. Many a graduate student was known to have bribed a weary operator with coffee and cookies in the wee hours of the morning for just one more run.

Batch Processing

A typical computer in the 1960s and 1970s was a large machine stored in its own frigidly air-conditioned room. Its processing was managed by a human *operator*. A user would deliver his or her program, usually stored as a deck of punched cards, to the operator to be executed. The user would come back later—perhaps the next day—to retrieve the printed results.

When delivering the program, the user would also provide a set of separate instructions regarding the system software and other resources needed to execute the program. Together the program and the system instructions were called a *job*. The operator would make any necessary devices available and load any special system software required to satisfy the job. Obviously, the process of preparing a program for execution on these early machines was quite time-consuming.

To perform this procedure more efficiently, the operator would organize various jobs from multiple users into batches. A batch would contain a set of jobs that needed the same or similar resources. With batch processing, the operator did not have to reload and prepare the same resources over and over. Figure 10.2 depicts this procedure.

Batch systems could be executed in a multiprogramming environment. In that case, the operator would load multiple jobs from the same batch into memory, and these jobs would compete for the use of the CPU and other shared resources. As the resources became available, the jobs would be scheduled to use the CPU.

Although the original concept of batch processing is not a function of modern operating systems, the terminology persists. The term "batch" has come to mean a system in which programs and system resources are

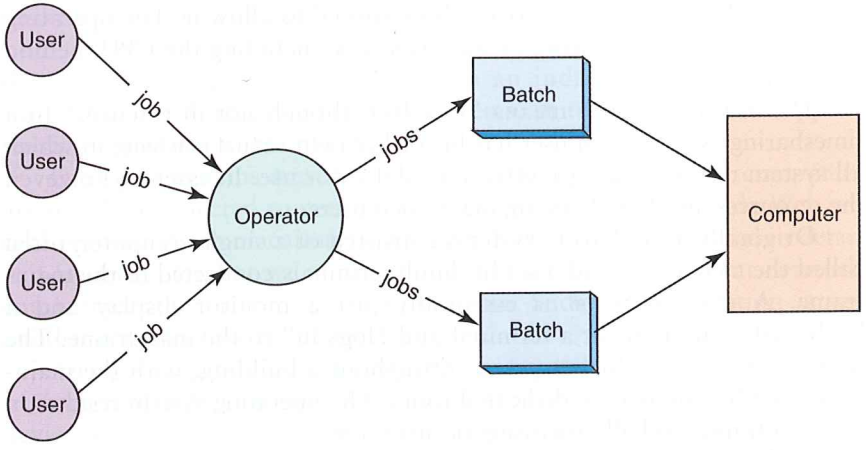

FIGURE 10.2 In early systems, human operators would organize jobs into batches

coordinated and executed without interaction between the user and the program. Modern operating systems incorporate batch-style processing by allowing the user to define a set of OS commands as a batch file to control the processing of a large program or a set of interacting programs. For example, files with the extension `.bat` in Microsoft Windows originated from the idea of batch control files; they contain system commands.

Although most of our computer use these days is interactive, some jobs still lend themselves to batch processing. For example, processing a corporation's monthly payroll is a large job that uses specific resources with essentially no human interaction.

Early batch processing allowed multiple users to share a single computer. Although the emphasis has changed over time, batch systems taught us valuable lessons about resource management. The human operator of early computer systems played many of the roles that modern operating system software handles now.

Timesharing

As we pointed out in Chapter 1, the problem of how to capitalize on computers' greater capabilities and speed led to the concept of *timesharing*. A timesharing system allows multiple users to interact with a computer at the same time. Multiprogramming allowed multiple processes to be active at once, which gave rise to the ability for programmers to interact with the computer system directly, while still sharing its resources.

Timesharing systems create the illusion that each user has exclusive access to the computer. That is, each user does not have to actively compete for resources, though that is exactly what is happening behind the scenes. One user may actually know he is sharing the machine with other users, but does not have to do anything special to allow it. The operating system manages the sharing of the resources, including the CPU, behind the scenes.

The word "virtual" means "in effect, though not in essence." In a timesharing system, each user has his or her own virtual machine, in which all system resources are (in effect) available for use. In essence, however, the resources are shared among many such users.

Originally, timesharing systems consisted of a single computer, often called the mainframe, and a set of dumb terminals connected to the mainframe. A dumb terminal is essentially just a monitor display and a keyboard. A user sits at a terminal and "logs in" to the mainframe. The dumb terminals might be spread throughout a building, with the mainframe residing in its own dedicated room. The operating system resides on the mainframe, and all processing occurs there.

Each user is represented by a *login process* that runs on the mainframe. When the user runs a program, another process is created (spawned

» Timesharing A system in which CPU time is shared among multiple interactive users at the same time

» Virtual machine The illusion created by a timesharing system that each user has a dedicated machine

» Mainframe A large, multiuser computer often associated with early timesharing systems

» Dumb terminal A monitor and keyboard that allowed the user to access the mainframe computer in early timesharing systems

by the user's login process). CPU time is shared among all of the processes created by all of the users. Each process is given a little bit of CPU time in turn. The premise is that the CPU is so fast that it can handle the needs of multiple users without any one user seeing any delay in processing. In truth, users of a timesharing system can sometimes see degradation in the system's responses, depending on the number of active users and the CPU capabilities. That is, each user's machine seems to slow down when the system becomes overburdened.

Although mainframe computers are interesting now mostly for historical reasons, the concept of timesharing remains highly relevant. Today, many desktop computers run operating systems that support multiple users in a timesharing fashion. Although only one user is actually sitting in front of the computer, other users can connect through other computers across a network connection.

Other OS Factors

As computing technology improved, the machines themselves got smaller. Mainframe computers gave rise to *minicomputers,* which no longer needed dedicated rooms in which to store them. Minicomputers became the basic hardware platform for timesharing systems. *Microcomputers,* which for the first time relied on a single integrated chip as the CPU, truly fit on an individual's desk. Their introduction gave rise to the idea of a *personal computer* (PC). As the name implies, a personal computer is not designed for multiperson use, and originally personal computer operating systems reflected this simplicity. Over time, personal computers evolved in functionality and incorporated many aspects of larger systems, such as timesharing. Although a desktop machine is still often referred to as a PC, the term "workstation" is sometimes used and is perhaps more appropriate, describing the machine as generally dedicated to an individual, but capable of supporting much more. Operating systems, in turn, evolved to support these changes in the use of computers.

Operating systems must also take into account the fact that computers are usually connected to networks. Today with the World Wide Web we take network communication for granted. Networks are discussed in detail in a later chapter, but we must acknowledge here the effect that network communication has on operating systems. Such communication is yet another resource that an OS must support.

An operating system is responsible for communicating with a variety of devices. Usually that communication is accomplished with the help of a device driver, a small program that "knows" the way a particular device expects to receive and deliver information. With device drivers, every operating system no longer needs to know about every device with which it

might possibly be expected to communicate in the future. It's another beautiful example of abstraction. An appropriate device driver often comes with new hardware, and the most up-to-date drivers can often be downloaded for free from the manufacturing company's website.

One final aspect of operating systems is the need to support real-time systems. A real-time system is one that must provide a guaranteed minimum response time to the user. That is, the delay between receiving a stimulus and producing a response must be carefully controlled. Real-time responses are crucial in software that, for example, controls a robot, a nuclear reactor, or a missile. Although all operating systems acknowledge the importance of response time, a real-time operating system strives to optimize it.

» Real-time system A system in which response time is crucial given the nature of the application domain

» Response time The time delay between receiving a stimulus and producing a response

10.2 Memory Management

Let's review what we said about main memory in Chapter 5. All programs are stored in main memory when they are executed. All data referenced by those programs are also stored in main memory so that they can be accessed. Main memory can be thought of as a big, continuous chunk of space divided into groups of 8, 16, or 32 bits. Each byte or word of memory has a corresponding address, which is simply an integer that uniquely identifies that particular part of memory. See Figure 10.3. The first memory address is 0.

Earlier in this chapter we stated that in a multiprogramming environment, multiple programs (and their data) are stored in main memory at the same time. Thus operating systems must employ techniques to perform the following tasks:

- Track where and how a program resides in memory
- Convert logical program addresses into actual memory addresses

A program is filled with references to variables and to other parts of the program code. When the program is compiled, these references are changed into the addresses in memory where the data and code reside. But given that we don't know exactly where a program will be loaded into main memory, how can we know which address to use for anything?

The solution is to use two kinds of addresses: logical addresses and physical addresses. A logical address (sometimes called a virtual or relative address) is a value that specifies a generic location relative to the program but not to the reality of main memory. A physical address is an actual address in the main memory device, as shown in Figure 10.3.

When a program is compiled, a reference to an identifier (such as a variable name) is changed to a logical address. When the program is eventually loaded into memory, each logical address is translated into a specific physical address. The mapping of a logical address to a physical address is called address binding. The later we wait to bind a logical address to a

» Logical address A reference to a stored value relative to the program making the reference

» Physical address An actual address in the main memory device

» Address binding The mapping from a logical address to a physical address