**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.

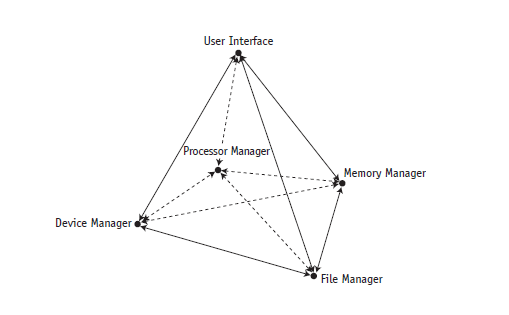
Therefore, each time the user sends a command, the operating system must make sure that the command is executed; or, if it’s not executed, it must arrange for the user to get a message explaining the error. Remember: This doesn’t necessarily mean that the operating system executes the command or sends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

**Operating System Software**

The pyramid 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.



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.

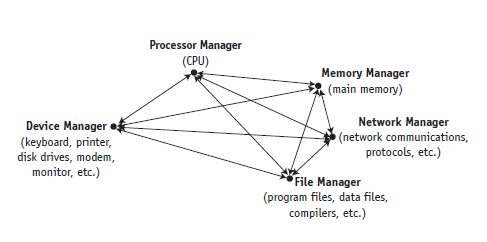
Regardless of the size or configuration of the system, each of the subsystem managers, must perform the following tasks:

• Monitor its resources continuously

• Enforce the policies that determine who gets what, when, and how much

• Allocate the resource when appropriate

• DE allocate the resource when appropriate



**Main Memory Management**

The Memory Manager 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 de-allocates memory.

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 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.

**The Processor Manager has two levels of responsibility.**

1. 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.
2. 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** 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, de-allocating the device, making it available to the next process or job.

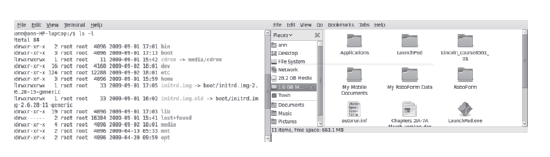
**The File Manager** keeps track of every file in the system, including data files, program files, compilers, and applications. By using predetermined access policies, it enforces restrictions on who has access to which files. The File Manager also controls what users are allowed to do with files once 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

De-allocates them.

**Network Management** Operating systems with Internet or networking capability have a fifth essential managercalled the Network Manager that provides a convenientway for users to share resources while controlling users’ access to them. Theseresources include hardware (such as CPUs, memory areas, printers, tape drives,modems, and disk drives) and software (such as compilers, application programs, anddata files).

**User Interface** is the portion of the operating system that users interact withdirectly. In the old days, the user interface consisted of commands typed on a keyboardand displayed on a monitor. Now most systems allow users tochoose a menu option from a list. The user interface, desktops, and formats varywidely from one operating system to another.



*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.

**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.

1. **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).
2. **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 typewriter-like terminal. The operating system provides immediate feedback to the user and response time can be measured in fractions of a second.
3. **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 (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 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.

1. **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.
2. **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.