Chapter 1

What is an Operating System? An *operating system* acts as an intermediary between the user of a computer and the computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a *convenient* (suitable) *and efficient* manner.

Operating system goals:

- **Execute user programs and make solving user problems easier.**
- **Make the computer system convenient to use.**
- **Use the computer hardware in an efficient manner.**

Computer System Components:

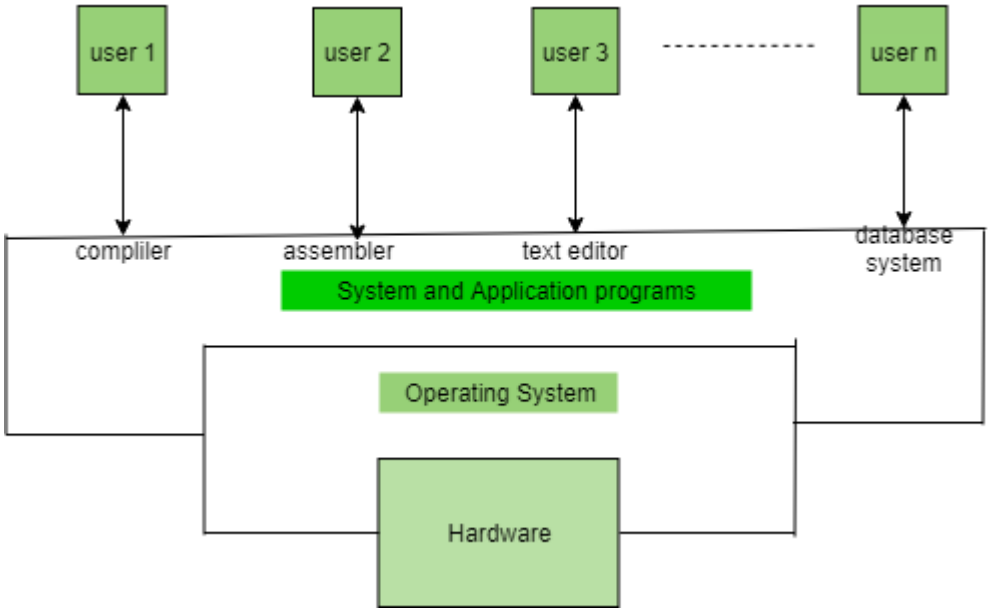
1. Hardware – provides basic computing resources (CPU, memory, I/O devices).

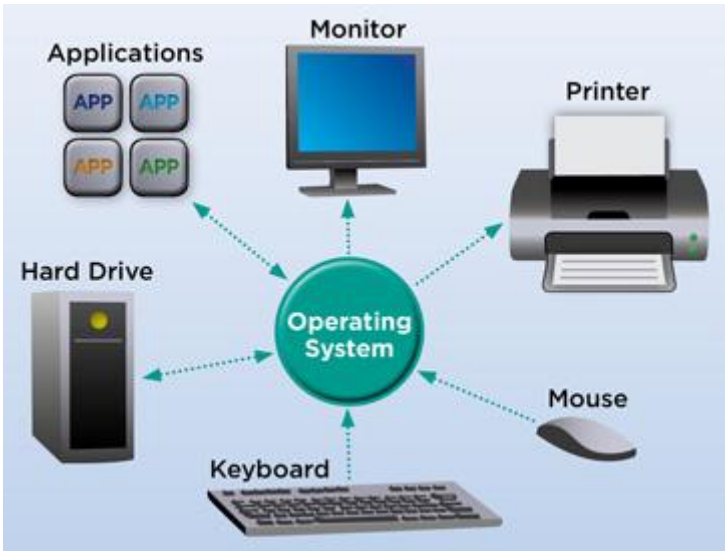
2. Operating system – controls and coordinates the use of the hardware among the various application programs for the various users.

3. Applications programs – define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).

4. Users (people, machines, other computers). **Abstract View of System Components** A computer system can be divided roughly into

four components: the *hardware*, the *operating system*, the *application programs*, and the *users.*





Operating System Functions

- **Resource allocator** – manages and allocates resources.
- **Control program** – controls the execution of user programs and operations of I/O devices.
- **Kernel** – the one program running at all times (all else being application programs).

(The **kernel** is a computer program that is the core of a computer's operating system, with complete control over everything in the system) it is one of the first programs loaded on start-up (after the bootloader). It handles the rest of start-up as well as input/output requests from software,

translating them into data-processing instructions for the central processing unit. It handles memory and peripherals like keyboards, monitors, printers, and speakers.

Computer-System Operation

A modern general-purpose computer system consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory (Figure 1.2). Each device controller is in charge of a specific type of device (for example, disk drives, audio devices, or video displays). The CPU and the device controllers can execute in parallel, competing for memory cycles. To ensure orderly access to the shared memory, a memory controller synchronizes access to the memory.

For a computer to start running—for instance, when it is powered up or rebooted—it needs to have an initial program to run.

- This initial program, or bootstrap program, Typically, it is stored within the computer hardware in read-only memory (ROM) or electrically erasable programmable read-only memory (EEPROM), known by the general term firmware. It initializes all aspects of the system, from CPU registers to device controllers to memory contents.
- The bootstrap program must know how to load the operating system and how to start executing that system.
- To accomplish this goal, the bootstrap program must locate the operating-system kernel and load it into memory.

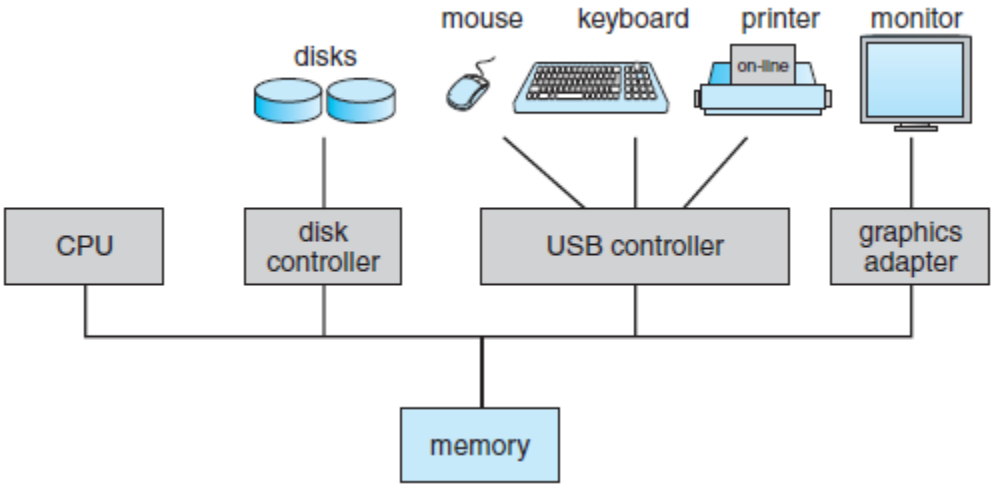


Figure 1.2 A modern computer system.

Storage Structure

The CPU can load instructions only from memory, so any programs to run must be stored there. General-purpose computers run most of their programs from rewritable memory, called main memory (also called **random-access memory**, or **RAM**). Main memory commonly is implemented in a semiconductor technology called **dynamic random-access memory (DRAM)**. Computers use other forms of memory as well. We have already mentioned read-only memory, ROM) and electrically erasable programmable read-only memory, EEPROM). Because ROM cannot be changed, only static programs, such as the bootstrap program described earlier, are stored there. EEPROM can be changed but cannot be changed frequently and so contains mostly static programs. For example, Smartphone's have EEPROM to store their factory-installed programs.

All forms of memory provide an array of bytes. Each byte has its own address. Ideally, we want the programs and data to reside in main memory permanently. This arrangement usually is not possible for the following two reasons:

1. Main memory is usually too small to store all needed programs and data permanently.

2. Main memory is a **volatile** storage device that loses its contents when power is turned off or otherwise lost. Thus, most computer systems provide **secondary storage** as an extension of main memory. The main requirement for secondary storage is that it be able to hold large quantities of data permanently

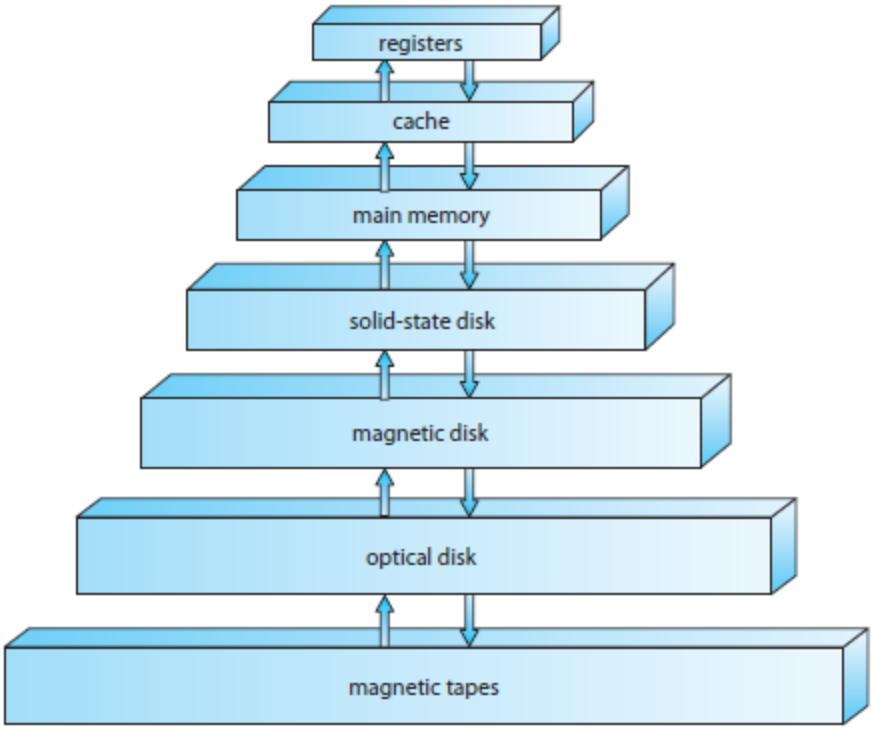


Figure 1.4 Storage-device hierarchy.

Operating-System Structure

Now that we have discussed basic computer-system organization and architecture, we are ready to talk about operating systems. An operating system provides the environment within which programs are executed. Internally, operating systems vary greatly in their makeup (structure), since they are organized along many different lines. There are, however, many commonalities (sharing features), which we consider in this section. One of the most important aspects of operating systems is the ability to multiprogramming. A single program cannot, in general, keep either the CPU or the I/O devices busy at all times. Single users frequently have multiple programs running. **Multiprogramming** increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute. The idea is as follows: The operating system keeps several jobs in memory simultaneously (Figure 1.9). Since, in general, main memory is too small to accommodate all jobs, the jobs are kept initially on the disk in the **job pool**. This pool consists of all processes residing on disk awaiting allocation of main memory. The set of jobs in memory can be a subset of the jobs kept in the job pool. The operating system picks and begins to execute one of the jobs in memory.

Eventually, the job may have to wait for some task, such as an I/O operation, to complete. In a non multiprogrammed system, the CPU would sit idle. In a multiprogrammed system, the operating system simply switches to, and executes, another job. When *that* job needs to wait, the CPU switches to *another* job, and so on. Eventually, the first job finishes waiting and gets the CPU back. As long as at least one job needs to execute, the CPU is never idle.

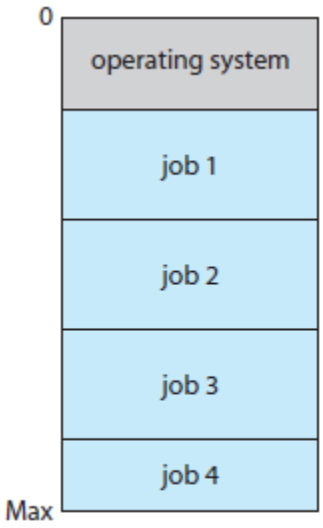
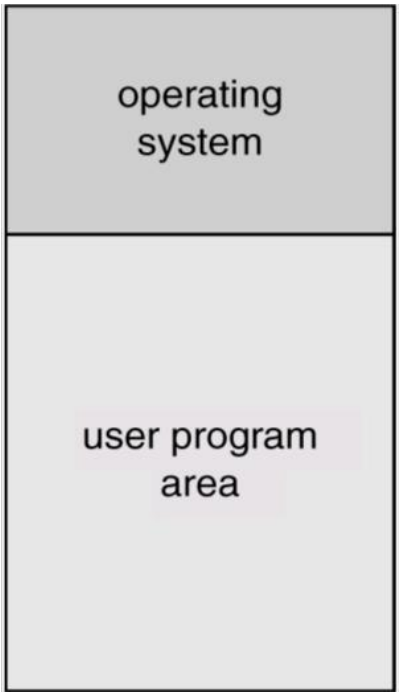


Figure 1.9 Memory layout for a multiprogramming system

Simple Batch System

A batch system is when a computer is programmed to batch together a number of transactions for processing at a specific time. For example a bank may run batch jobs to update all payments into customer accounts at midnight. The main idea behind this is to save processing time and resource so these are available for more urgent transactions that need to be processed quicker. Early computers were run from console. The card readers and tape drives were input devices. Line printers, tape drives and card punched were common output devices. The user did not interact directly with computer system. **Memory Layout for a Simple Batch System**

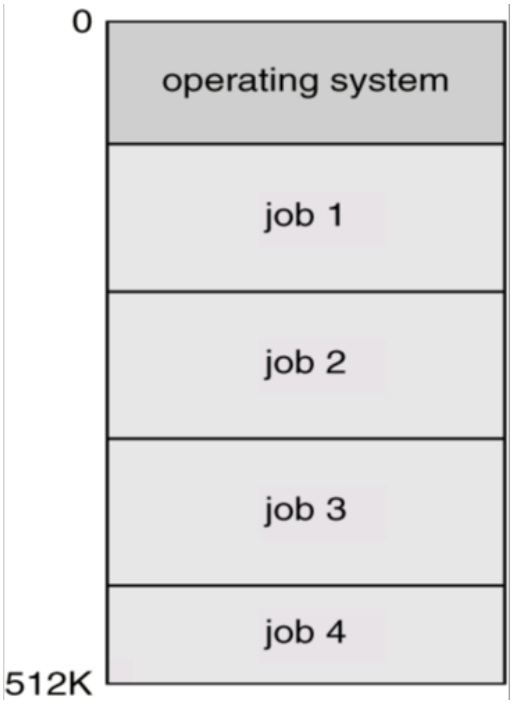


Problems

1. How does the monitor know about the nature of the job (e.g., FORTRAN versus Assembly) or which program to execute?

2. How does the monitor distinguish (a) job from job? (b) Data from program? **Solution** Introduce control cards

Spooling A **spool** is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. Although a printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixed together. The operating system solves this problem by intercepting all output to the printer. Each application's output is spooled to a separate disk file. When an application finishes printing, the spooling system queues the corresponding spool file for output to the printer. The spooling system copies the queued spool files to the printer one at a time. In another words (Overlap I/O of one job with computation of another job. While executing one job, the OS). Reads next job from card reader into a storage area on the disk (job queue). Outputs printout of previous job from disk to printer. *Job pool* – data structure that allows the OS to select which job to run next in order to increase CPU utilization **Multiprogrammed Batch Systems** Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them.



OS Features Needed for Multiprogramming

- I/O routine supplied by the system.
- Memory management – the system must allocate the memory to several jobs.
- CPU scheduling – the system must choose among several jobs ready to run.
- Allocation of devices.

Time-Sharing Systems–Interactive Computing

The CPU is multiplexed among several jobs that are kept in memory and on disk (the CPU is allocated to a job only if the job is in memory). A job is swapped in and out of memory to the disk. On-line communication between the user and the system is provided; when the operating system finishes the execution of one command, it seeks the next ―control statement‖ not from a card reader, but rather from the user's keyboard. On-line system must be available for users to access data and code.

Personal-Computer Systems

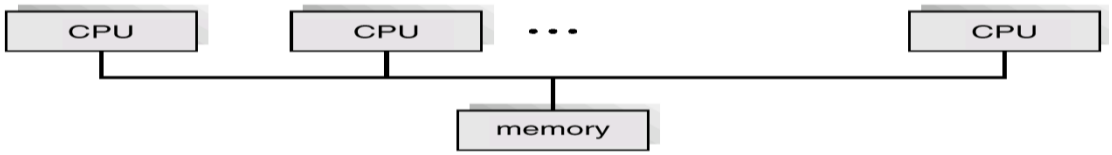
Personal computers – computer system dedicated to a single user. I/O devices – keyboards, mice, display screens, small printers. **Parallel Systems** Multiprocessor systems with more than one CPU in close communication. *Tightly coupled system* – processors share memory and a clock; communication usually takes place through the shared memory.

Advantages of parallel system:

- Increased *throughput*
- Economical
- Increased reliability
- fail-soft systems

Symmetric multiprocessing (SMP)	Asymmetric multiprocessing
- Each processor runs an identical copy of the operating system. - Many processes can run at once without performance deterioration (drop). - Most modern operating systems support SMP	- Each processor is assigned a specific task; master processor schedules and allocates work to slave processors. - More common in extremely large systems

Symmetric Multiprocessing Architecture



An SMP system:	An AMP system:
Multiple CPUs	Multiple CPUs
Each of which has the same architecture	Each of which may be a different architecture [but can be the same]
CPUs share memory space [or, at least, some of it]	Each has its own address space
Normally an OS is used and this is a single instance(copy) that runs on all the CPUs, dividing work between them	Each may or may not run an OS [and the OSes need not be the same]
Some kind of communication facility between the CPUs is provided [and this is normally shared memory]	Some kind of communication facility between the CPUs is provided

Real-Time Systems

Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial

control systems, and some display systems. Well-defined fixed-time constraints.

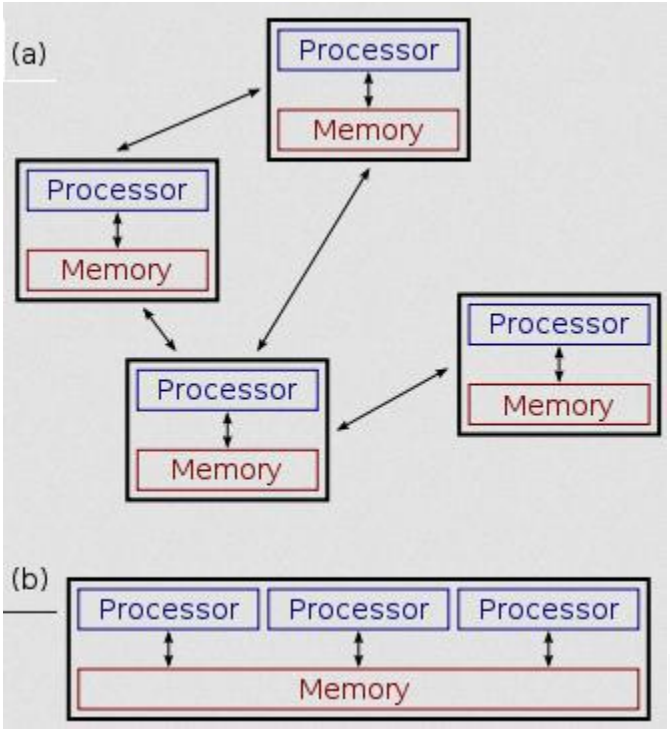
Hard real-time system. Secondary storage limited or absent, data stored in short-term memory, or read-only memory (ROM) Conflicts with time-sharing systems, not supported by general-purpose operating systems.

Soft real-time system Limited utility in industrial control or robotics

Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.

Distributed Systems

A *distributed system* is a model in which components located on networked computers communicate and coordinate their actions by passing messages.[1] The components interact with each other in order to achieve a common goal. Three significant characteristics of distributed systems are: concurrency of components, lack of a global clock, and independent failure of components. Distribute the computation among several physical processors. *Loosely coupled system* – each processor has its own local memory; processors communicate with one another through various communications lines, such as high-speed buses or telephone lines.



(a) : a distributed system.
(b): a parallel system.

Advantages of distributed systems.

- Resources Sharing
- Computation speed up – load sharing
- Reliability
- Communications
- Network Operating System
- provides file sharing
- provides communication scheme

- runs independently from other computers on the network

- Distributed Operating System

- less autonomy between computers

Gives the impression (looks like) there is a single operating system controlling the network.