

Operating System

Unit – 1

(Part-A)

Introduction



Mayank Mishra

School of Electronics Engineering

KIIT-Deemed to be University

Introduction

- An operating system is software that manages a computer's hardware.
- A program that acts as an intermediary between a user of a computer and the computer hardware
- Operating systems are everywhere, from cars and home appliances that include “Internet of Things” devices, to smart phones, personal computers, enterprise computers, and cloud computing environments.
- 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

- In order to explore the role of an operating system in a modern computing environment, it is important first to understand the organization and architecture of computer hardware.
- This includes the CPU, memory, and I/O devices, as well as storage. A fundamental responsibility of an operating system is to allocate these resources to programs.

Computer System (Contd.)

- A computer system can be divided roughly into four components: *the hardware, the operating system, the application programs, and a user.*
- **Hardware:** Central Processing Unit (CPU), memory, and the input/output (I/O) devices, etc.

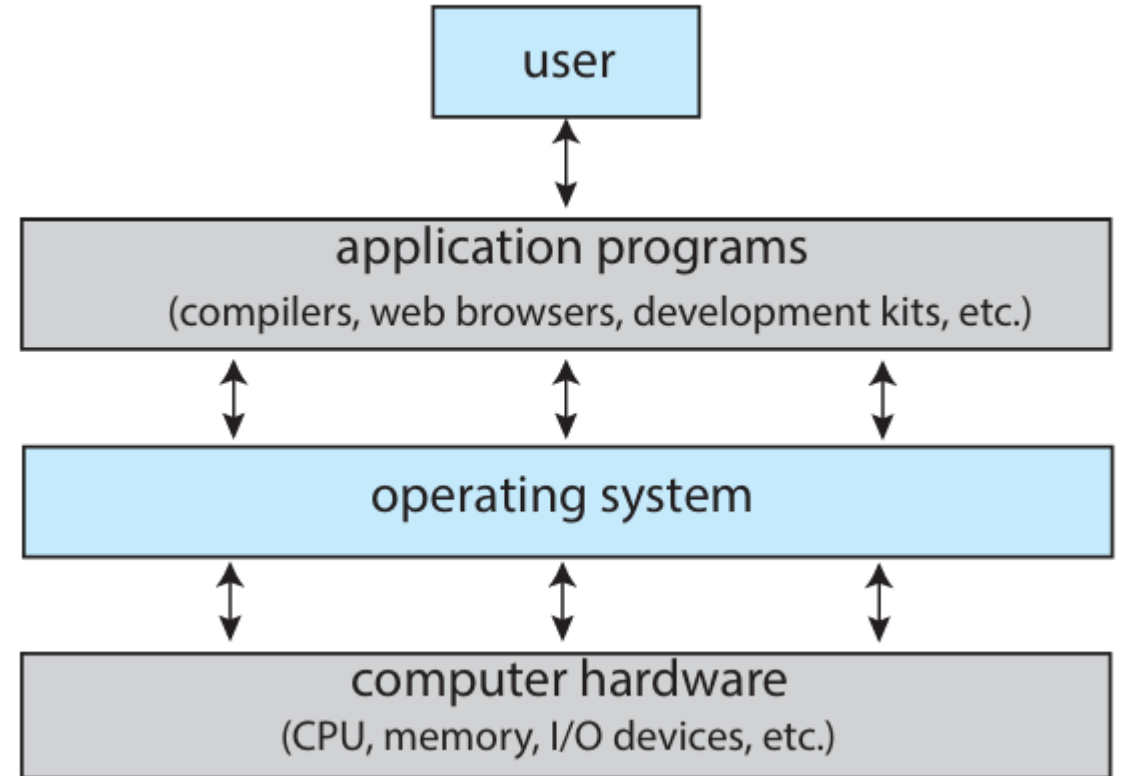


Fig. Abstract view of the components of a computer system

Computer System (Contd.)

- The application programs—such as word processors, spreadsheets, compilers, and web browsers—define the ways in which these resources are used to solve users' computing problems.
- To understand more fully the operating system's role, we can explore operating systems from two view points: *that of the user and that of the system.*

User View:

- The user's view of the computer varies according to the interface being used. Many computer users sit with a laptop or in front of a PC consisting of a monitor, keyboard, and mouse. Such a system is designed for one user to monopolize its resources.
- The goal is to maximize the work (or play) that the user is performing.
- In this case, the operating system is designed mostly for ease of use, with some attention paid to performance and security and none paid to resource utilization—how various hardware and software resources are shared.

- Increasingly, many users interact with mobile devices such as smartphones and tablets—devices that are replacing desktop and laptop computer systems for some users.
- These devices are typically connected to networks through cellular or other wireless technologies. The user interface for mobile computers generally features a touch screen, where the user interacts with the system by pressing and swiping fingers across the screen rather than using a physical keyboard and mouse.
- Many mobile devices also allow users to interact through a voice recognition interface, such as Apple's Siri.

- Some computers have little or no user view.
- *For example*, embedded computers in home devices and automobiles may have numeric keypads and may turn indicator lights on or off to show status, but they and their operating systems and applications are designed primarily to run without user intervention.

System View:

- From the computer's point of view, the operating system is the program most intimately involved with the hardware.
- In this context, we can view an operating system as a resource allocator. A computer system has many resources that may be required to solve a problem: CPU time, memory space, storage space, I/O devices, and so on.
- The operating system acts as the manager of these resources. Facing numerous and possibly conflicting requests for resources, the operating system must decide how to allocate them to specific programs and users so that it can operate the computer system efficiently and fairly.

Computer System Organization

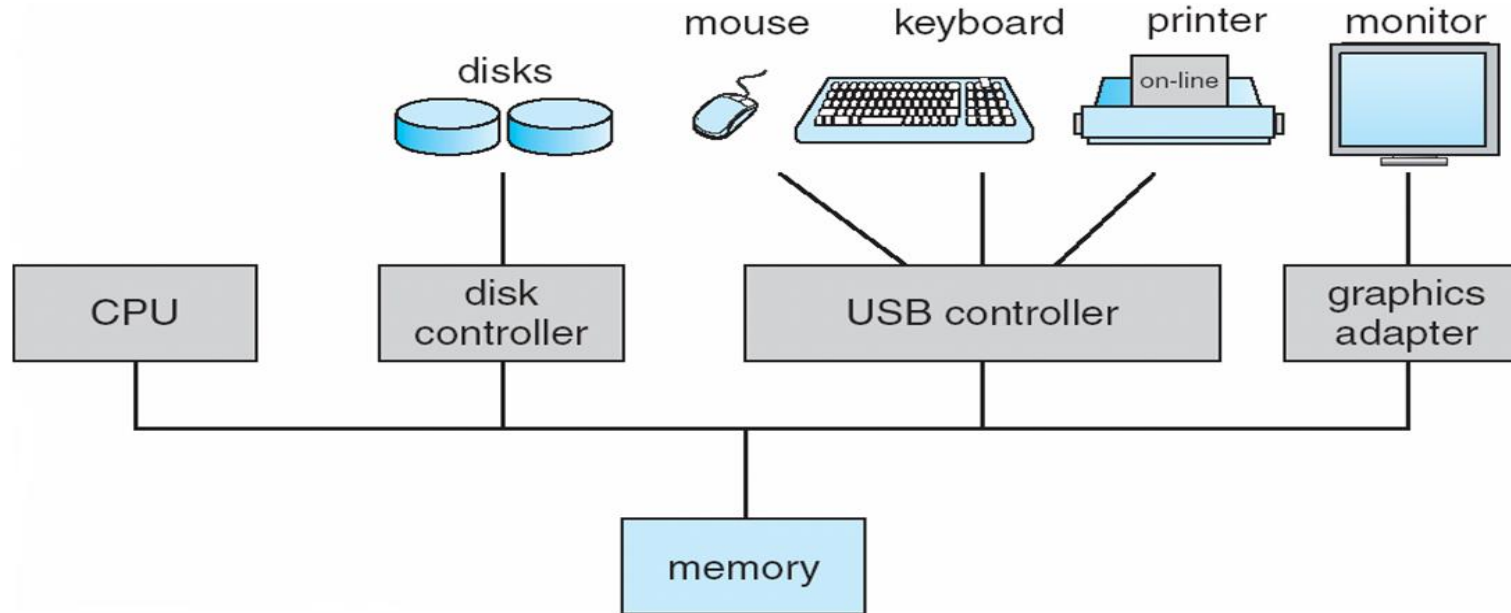


Fig. A typical PC computer system

- 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 between components and shared memory (as shown in Figure). Every driver contains local buffer.

- Each device controller is in charge of a specific type of device. Depending on the controller, more than one device may be attached. For instance, one system USB port can connect to a USB hub, to which several devices can connect.
- Typically, operating systems have a device driver for each device controller. This device driver understands the device controller and provides the rest of the operating system with a uniform interface to the device.
- 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.
- A device controller maintains some local buffer storage and a set of special-purpose registers. The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage.

- To know how such system operates, three key aspects of the system which are, *interrupts* (which alert the CPU to events that require attention), *storage structure* and *I/O structure* can be useful.

Bootstrap Program:

- A bootstrap program is the first code to run when a computer starts, and it's essential for the operating system to work.
- The bootstrap program, also known as the **bootstrap loader**, loads the operating system into the computer's memory and initializes system components (for example, every device will have driver file, so all those driver file will be loaded).
- The bootstrap program is typically stored in the computer's Read Only Memory (ROM). The bootstrap program runs automatically when the computer starts or restarts. Without the bootstrap program, the computer wouldn't know how to load the operating system or start applications.

Interrupts :

- An interrupt is a signal emitted by a device attached to a computer or from a program within the computer. It requires the operating system (OS) to stop and figure out what to do next.
- Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the **system bus**. (There may be many buses within a computer system, but the system bus is the main communications path between the major components.)
- Interrupts are used for many other purposes as well and are a key part of how operating systems and hardware interact.
- An interrupt signal might be **planned** (i.e., specifically requested by a program) or it may be **unplanned** (i.e., caused by an event that may not be related to a program that's currently running on the system).

- When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location. The fixed location usually contains the starting address where the service routine for the interrupt is located.
- The **Interrupt Service Routine (ISR)** executes; on completion, the CPU resumes the interrupted computation.
- Interrupts are an important part of a computer architecture. Each computer design has its own interrupt mechanism, but several functions are common.
- The interrupt must transfer control to the appropriate interrupt service routine.
- The straightforward method for managing this transfer would be to invoke a generic routine to examine the interrupt information.

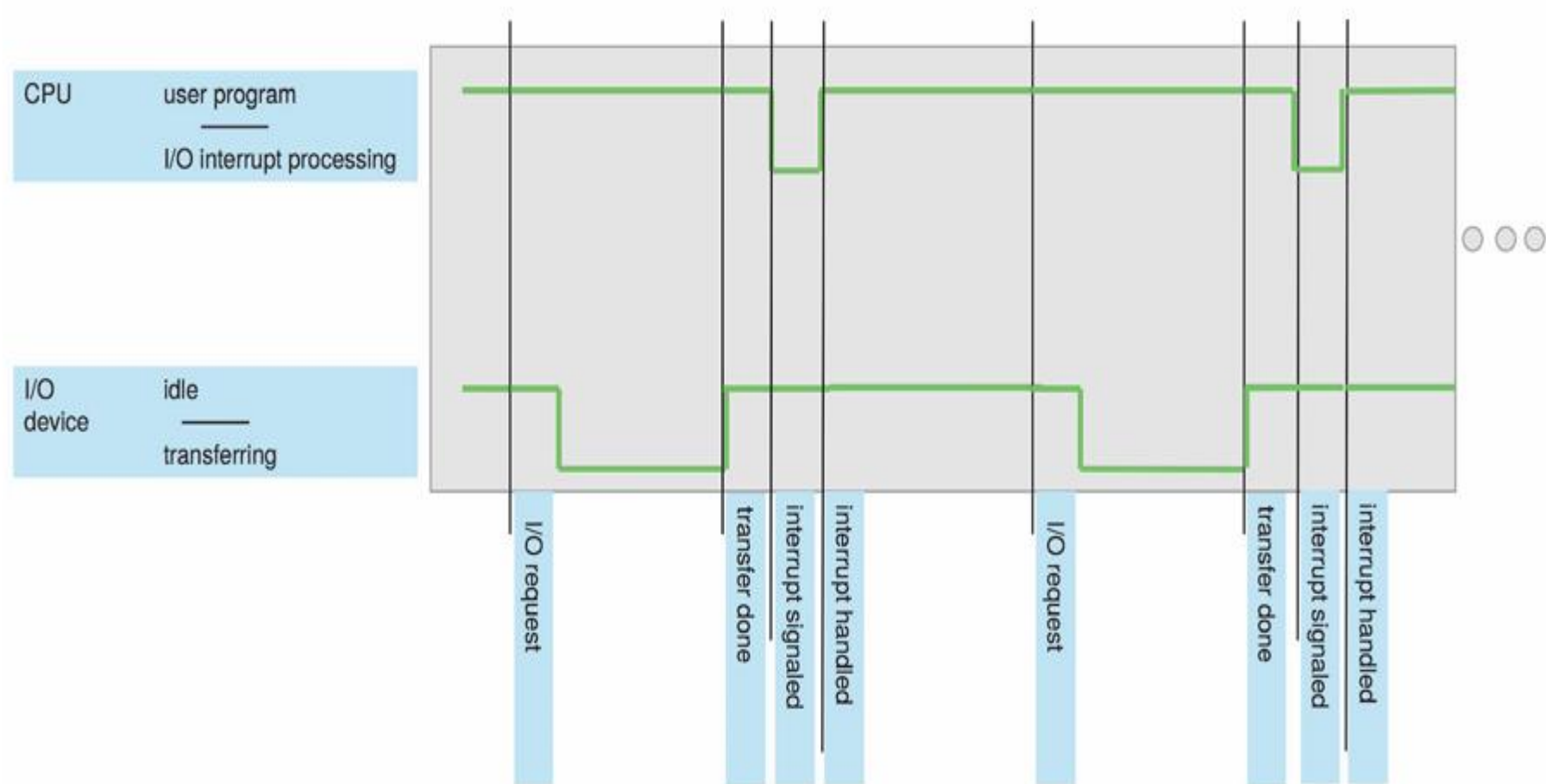


Fig. Interrupt timeline for a single program doing output.

- The routine, in turn, would call the **interrupt-specific handler**. However, interrupts must be handled quickly, as they occur very frequently.
- A table of pointers to interrupt routines can be used instead to provide the necessary speed. The interrupt routine is called indirectly through the table, with no intermediate routine needed.
- Generally, the table of pointers is stored in low memory (the first hundred or so locations). These locations hold the addresses of the interrupt service routines for the various devices.
- This array, or **interrupt vector**, of addresses is then indexed by a unique number, given with the interrupt request, to provide the address of the interrupt service routine for the interrupting device.

- The interrupt architecture must also save the state information of whatever was interrupted, so that it can restore this information after servicing the interrupt.
- The basic interrupt mechanism works as follows. The CPU hardware has a wire called the **interrupt-request line** that the CPU senses after executing every instruction.
- When the CPU detects that a controller has asserted a signal on the interrupt-request line, it reads the interrupt number and jumps to the interrupt-handler routine by using that interrupt number as an index into the interrupt vector. It then starts execution at the address associated with that index.

- The interrupt handler saves any state it will be changing during its operation, determines the cause of the interrupt, performs the necessary processing, performs a state restore, and executes a **return_from_interrupt** instruction to return the CPU to the execution state prior to the interrupt.
- The device controller *raises* an interrupt by asserting a signal on the interrupt request line, the CPU *catches* the interrupt and *dispatches* it to the interrupt handler, and the handler *clears* the interrupt by servicing the device.

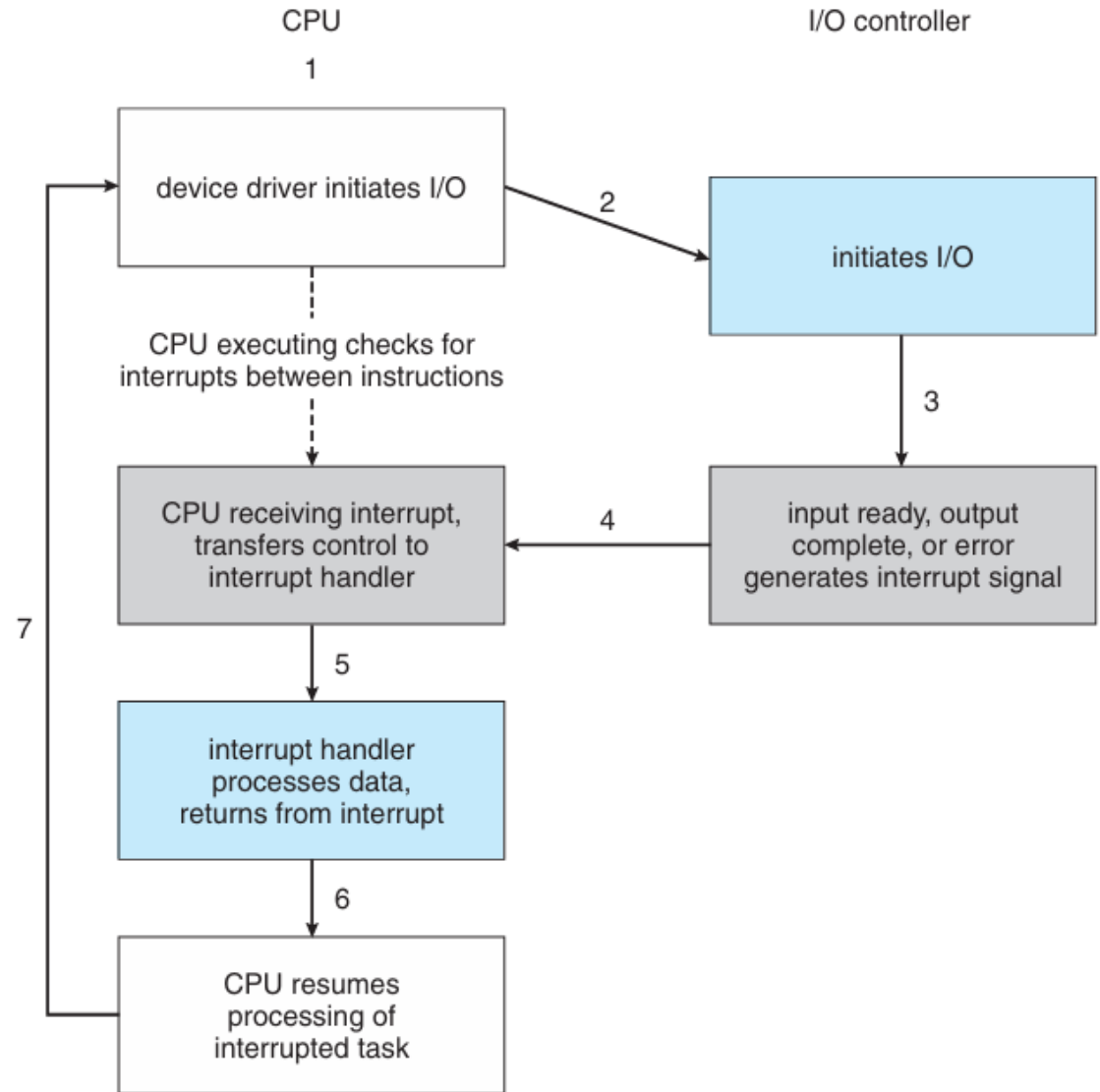


Fig. Interrupt-driven I/O cycle.

Types of Interrupts :

➤ There are two main types of interrupts:

1. **Hardware Interrupts:** The interrupt signal generated from external devices and i/o devices are made interrupt to CPU when the instructions are ready. Hardware interrupts are classified into two types which are as follows –
 - **Maskable Interrupt** – The hardware interrupts that can be delayed when a highest priority interrupt has occurred to the processor.
 - **Non Maskable Interrupt** – The hardware interrupts that cannot be delayed and immediately be serviced by the processor.
2. **Software Interrupts:** The interrupt signal generated from internal devices and software programs need to access any system call then software interrupts are present. Software interrupt is divided into two types. They are as follows –
 - **Normal Interrupts** – The interrupts that are caused by the software instructions are called software instructions.
 - **Exception** – Exception is nothing but an unplanned interruption while executing a program. For example – while executing a program if we got a value that is divided by zero is called an exception.

Storage Structure :

- 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. For example, the first program to run on computer power-on is a **bootstrap program**, which then loads the operating system.
- Since RAM is **volatile**—loses its content when power is turned off or otherwise lost—we cannot trust it to hold the bootstrap program. Instead, for this and some other purposes, the computer uses **electrically erasable programmable read-only memory (EEPROM)** and other forms of **firmwar** —storage that is infrequently written to and is nonvolatile. EEPROM can be changed but cannot be changed frequently.

(Some basic information)

STORAGE DEFINITIONS AND NOTATION

The basic unit of computer storage is the **bit**. A bit can contain one of two values, 0 and 1. All other storage in a computer is based on collections of bits. Given enough bits, it is amazing how many things a computer can represent: numbers, letters, images, movies, sounds, documents, and programs, to name a few. A **byte** is 8 bits, and on most computers it is the smallest convenient chunk of storage. For example, most computers don't have an instruction to move a bit but do have one to move a byte. A less common term is **word**, which is a given computer architecture's native unit of data. A word is made up of one or more bytes. For example, a computer that has 64-bit registers and 64-bit memory addressing typically has 64-bit (8-byte) words. A computer executes many operations in its native word size rather than a byte at a time.

Computer storage, along with most computer throughput, is generally measured and manipulated in bytes and collections of bytes. A **kilobyte**, or **KB**, is 1,024 bytes; a **megabyte**, or **MB**, is $1,024^2$ bytes; a **gigabyte**, or **GB**, is $1,024^3$ bytes; a **terabyte**, or **TB**, is $1,024^4$ bytes; and a **petabyte**, or **PB**, is $1,024^5$ bytes. Computer manufacturers often round off these numbers and say that a megabyte is 1 million bytes and a gigabyte is 1 billion bytes. Networking measurements are an exception to this general rule; they are given in bits (because networks move data a bit at a time).

- 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 on most systems for two reasons:*
 1. *Main memory is usually too small to store all needed programs and data permanently.*
 2. *Main memory, as mentioned, is volatile.*
- The most common secondary-storage devices are **hard-disk drives(HDDs)** and nonvolatile memory (NVM) devices, which provide storage for both programs and data. Most programs (system and application) are stored in secondary storage until they are loaded into memory.

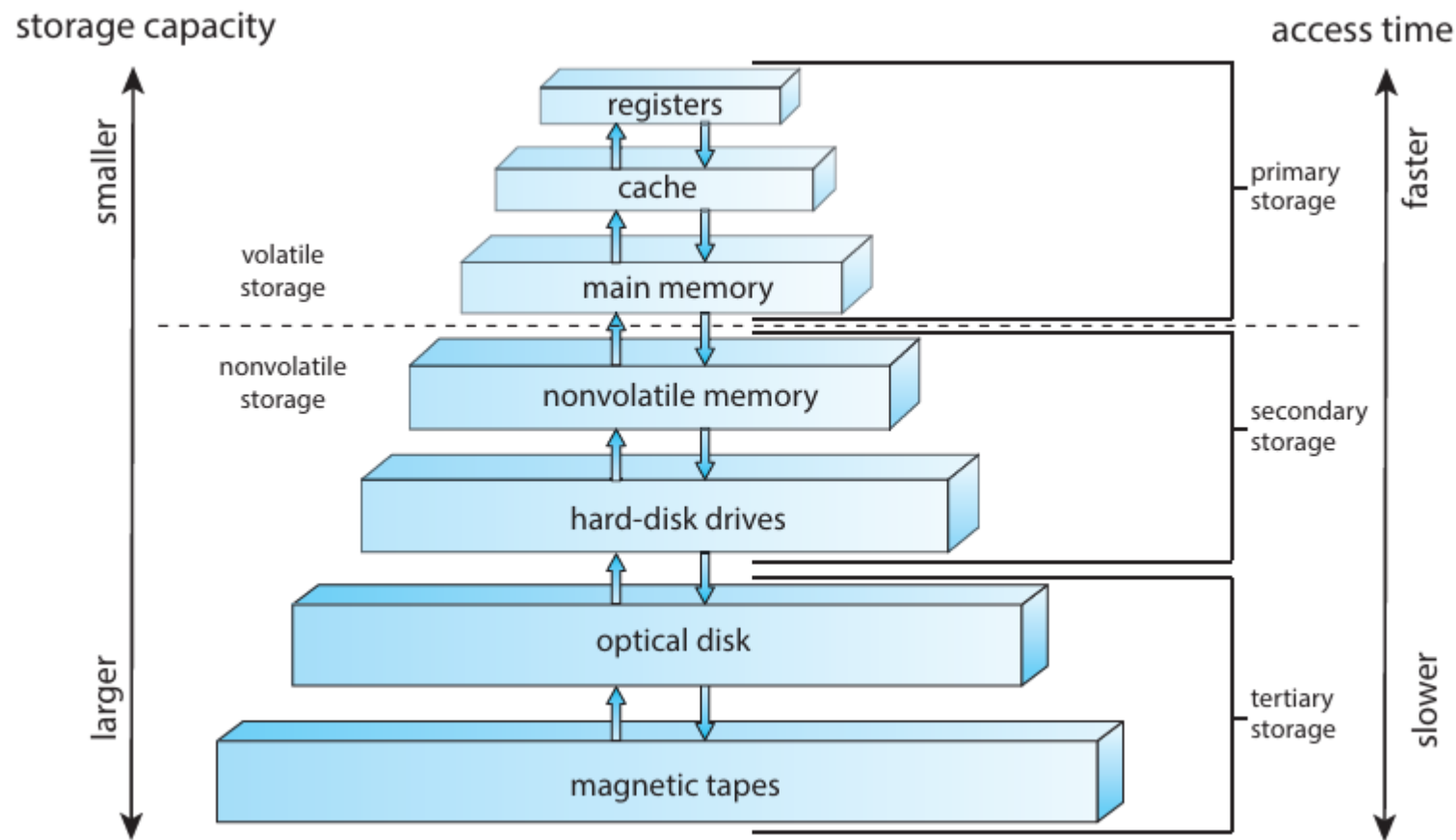


Fig. Storage-device hierarchy.

➤ As a general rule, there is a **trade-off** between size and speed, with smaller and faster memory closer to the CPU. As shown in the figure, in addition to differing in speed and capacity, the various storage systems are either volatile or nonvolatile.

➤ The top four levels of memory in the figure are constructed using semiconductor memory, which consists of semiconductor-based electronic circuits. NVM devices, at the fourth level, have several variants but in general are faster than harddisks. The most common form of NVM device is **flash memory**, which is popular in mobile devices such as smartphones and tablets. Increasingly, flash memory is being used for long-term storage on laptops, desktops, and servers as well.

Computer-System Architecture

A computer system can be organized in a number of different ways, which we can categorize roughly according to the number of general-purpose processors used.

Single-Processor Systems:

- Many years ago, most computer systems used a single processor containing one CPU with a single processing core. The core is the component that executes instructions and registers for storing data locally.
- The one main CPU with its core is capable of executing a general-purpose instruction set, including instructions from processes. These systems have other special-purpose processors as well. They may come in the form of device-specific processors, such as disk, keyboard, and graphics controllers.
- All of these special-purpose processors run a limited instruction set and do not run processes. Sometimes, they are managed by the operating system, in that the operating system sends them information about their next task and monitors their status.

Single-Processor Systems:

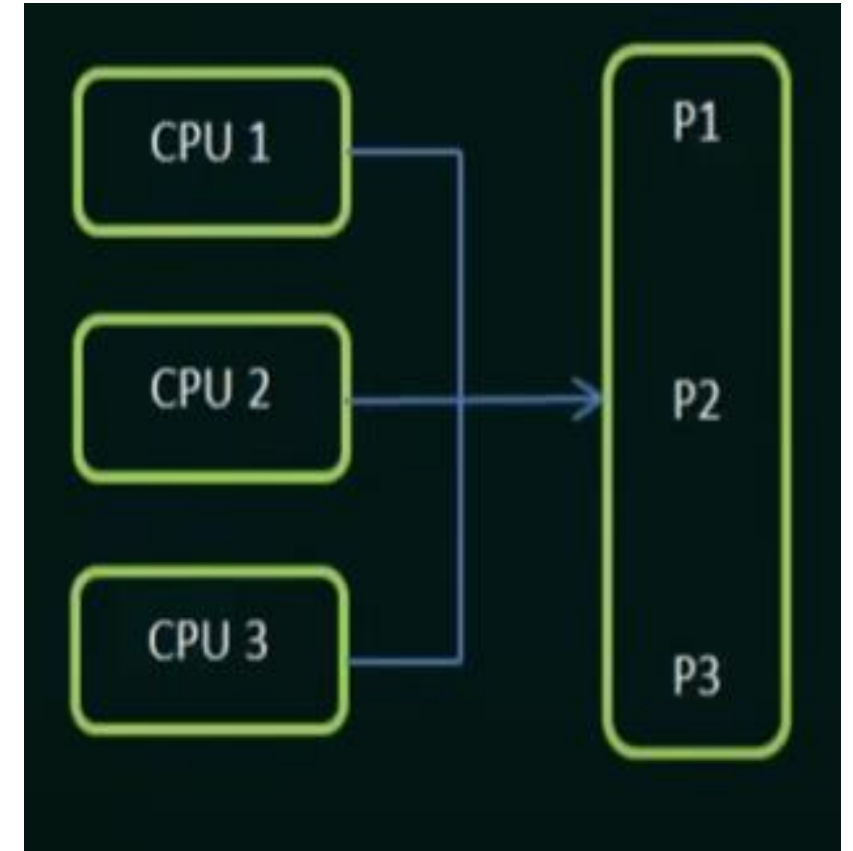
- The use of special-purpose microprocessors is common and does not turn a single-processor system into a multiprocessor. If there is only one general-purpose CPU with a single processing core, then the system is a single-processor system. According to this definition, however, very few contemporary computer systems are single-processor systems.

Multiprocessor Systems:

- On modern computers, from mobile devices to servers, multiprocessor systems now dominate the landscape of computing. Traditionally, such systems have two (or more) processors, each with a single-core CPU.
- The processors share the computer bus and sometimes the clock, memory, and peripheral devices.
- Multiprocessor Systems can be broadly classified into two categories: ***Symmetric Multiprocessing System, and Asymmetric Multiprocessing System.***
-

Symmetric Multiprocessing Operating System

- In a Symmetrical multiprocessing operating system, each processor executes the same copy of operating system every time. Each process makes its own decisions and works according to all other process to make sure that system works efficiently.
- Symmetrical multiprocessing operating system is also known as "**Shared Everything System**" because all the processors share memory and input-output bus.



Symmetrical Multiprocessing Operating System

➤ *Advantages:*

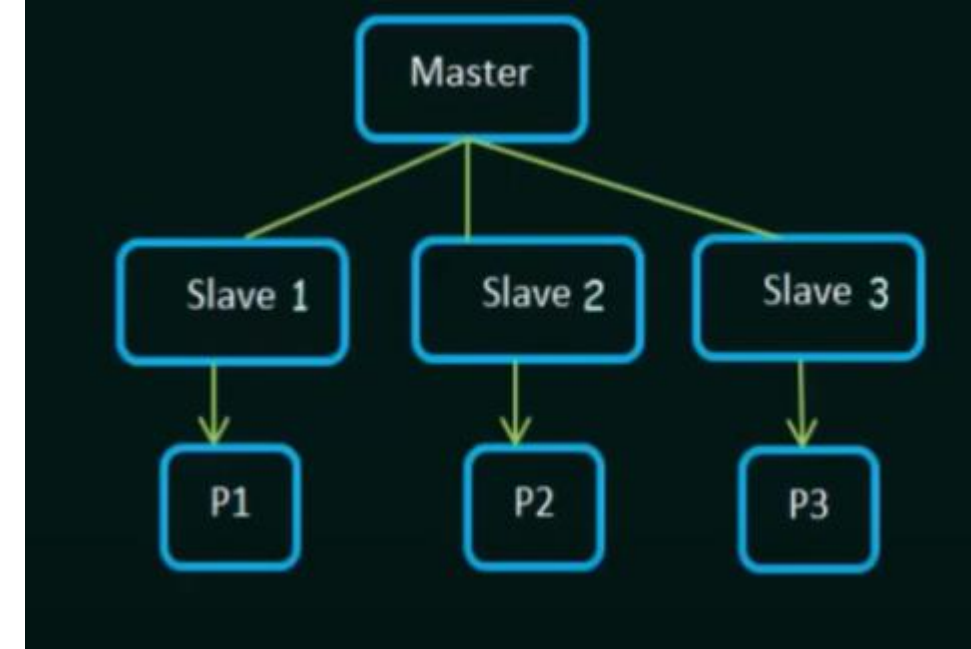
- ☐ Failure of one processor does not affect the functioning of other processors.
- ☐ It divides all the workload equally to the available processors.
- ☐ Makes use of available resources efficiently.

➤ *Disadvantages:*

- ☐ Symmetrical multiprocessing OS are more complex.
- ☐ Synchronization between multiple processors is difficult.
- ☐ Costly.

Asymmetric Multiprocessing Operating System

- In Asymmetrical multiprocessing operating system one processor acts as a master whereas remaining all processors act as slaves.
- Slave processors are assigned with ready to execute processes by the master processor. A ready queue is being maintained by master processor to provide with processes for slaves.
- In multiprocessing operating system a scheduler is created by master process that assigns processes to be executed to slave processors.



Asymmetric Multiprocessing Operating System

➤ *Advantages:*

- ☐ Asymmetrical multiprocessing operating system are cost-effective.
- ☐ They are easy to design and manage.
- ☐ They are more scalable.

➤ *Disadvantages:*

- ☐ There can be uneven distribution of workload among the processors.
- ☐ The processors do not share same memory.
- ☐ Entire system goes down if one process fails.

(For detailed information of Multiprocessor Systems)

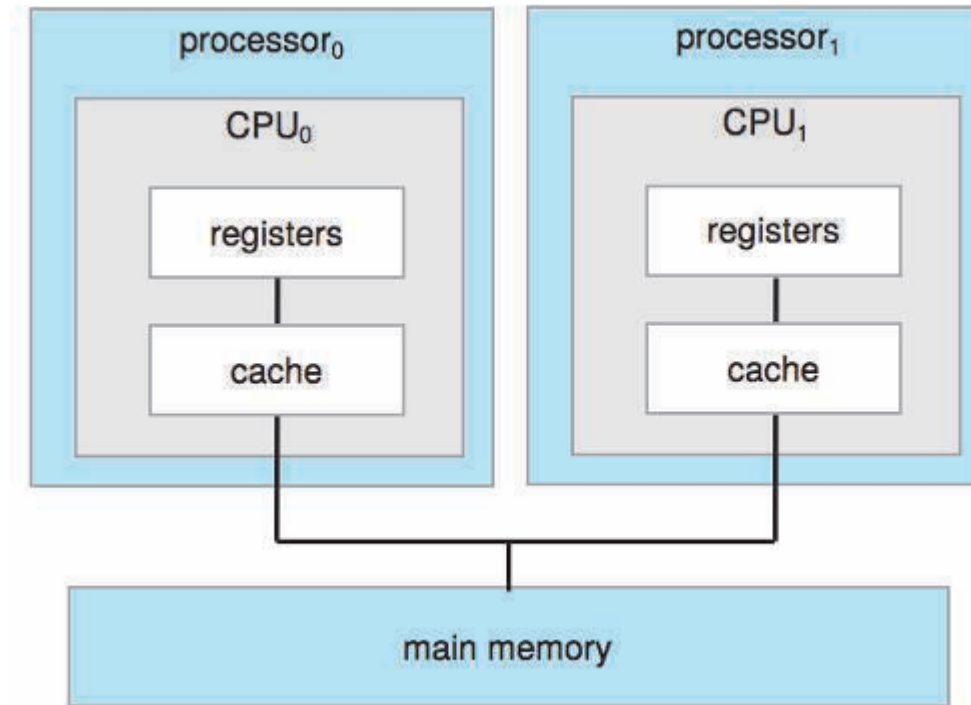


Fig. Symmetric multiprocessing architecture.

- The most common multiprocessor systems use **symmetric multiprocessing (SMP)**, in which each peer CPU processor performs all tasks, including operating-system functions and user processes.

(Some basic information)

DEFINITIONS OF COMPUTER SYSTEM COMPONENTS

- **CPU**—The hardware that executes instructions.
- **Processor**—A physical chip that contains one or more CPUs.
- **Core**—The basic computation unit of the CPU.
- **Multicore**—Including multiple computing cores on the same CPU.
- **Multiprocessor**—Including multiple processors.

Although virtually all systems are now multicore, we use the general term *CPU* when referring to a single computational unit of a computer system and *core* as well as *multicore* when specifically referring to one or more cores on a CPU.

(For detailed information of Multiprocessor Systems)

- In Figure, a dual-core design with two cores on the same processor chip has been shown. In this design, each core has its own register set, as well as its own local cache, often known as a **level 1, or L1 cache**.
- Notice, too, that a **level 2(L2)** cache is local to the chip but is shared by the two processing cores. Most architectures adopt this approach, combining local and shared caches, where local, lower-level caches are generally smaller and faster than higher-level shared caches.
- Aside from architectural considerations, such as cache, memory, and bus contention, a multicore processor with N cores appears to the operating system as N standard CPUs. This characteristic puts pressure on operating-system designers—and application programmers—to make efficient use of these processing cores.

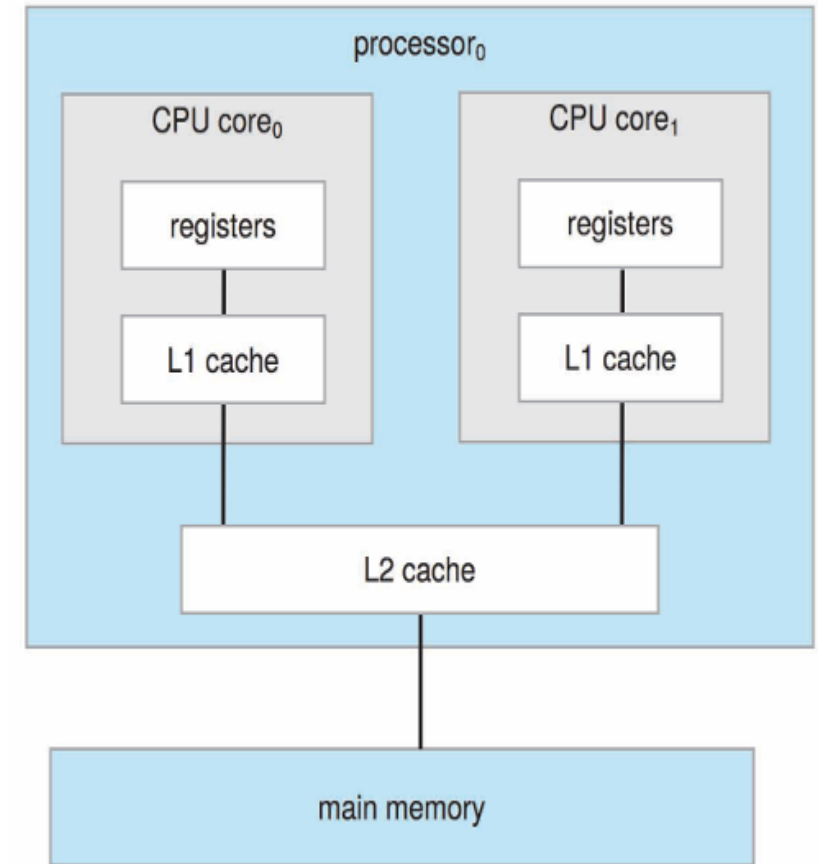


Fig. A dual-core design with two cores on the same chip.

(For detailed information of Multiprocessor Systems)

- Adding additional CPUs to a multiprocessor system will increase computing power; however, as suggested earlier, the concept does not scale very well, and once we add too many CPUs, contention for the system bus becomes a bottleneck and performance begins to degrade.
- An alternative approach is instead to provide each CPU (or group of CPUs) with its own local memory that is accessed via a small, fast local bus. The CPUs are connected by a shared system interconnect, so that all CPUs share one physical address space. This approach—known as **non-uniform memory access**, or **NUMA**.
- *A potential drawback with a NUMA system is increased latency when a CPU must access remote memory across the system interconnect, creating a possible performance penalty. In other words, for example, CPU0 cannot access the local memory of CPU3 as quickly as it can access its own local memory, slowing down performance. Operating systems can minimize this NUMA penalty through careful CPU scheduling and memory management.*

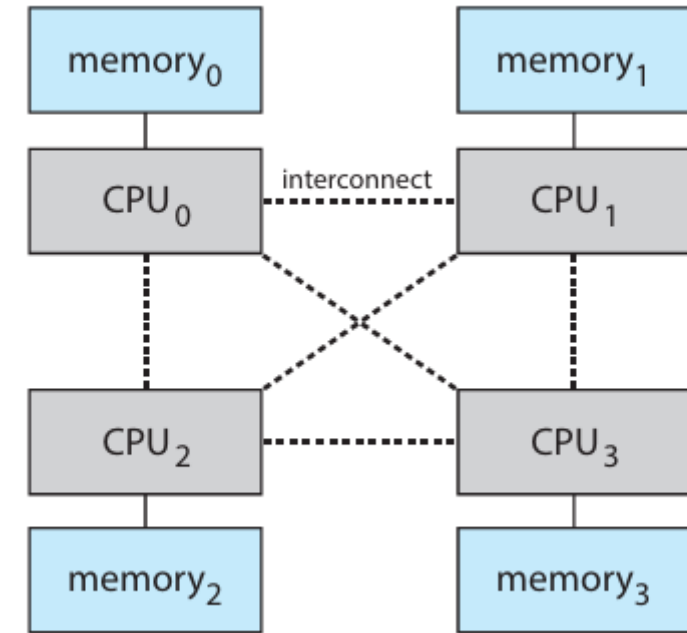


Fig. NUMA multiprocessing architecture.

Clustered Systems:

- Like Multiprocessor systems, clustered systems gather together multiple CPUs to accomplish computational work.
- It is composed of two or more individual systems coupled together.
- Clustering is usually used to provide high-availability service—that is, service that will continue even if one or more systems in the cluster fail. High availability provides increased reliability, which is crucial in many applications.
- Clustering can be structured asymmetrically or symmetrically.
 - ❑ In **asymmetric clustering**, one machine is in **hot-standby mode** while the other is running the applications. The hot-standby host machine does nothing but monitor the active server. If that server fails, the hot-standby host becomes the active server.
 - ❑ In **symmetric clustering**, two or more hosts are running applications and are monitoring each other. This structure is obviously more efficient, as it uses all of the available hardware. However, it does require that more than one application be available to run.

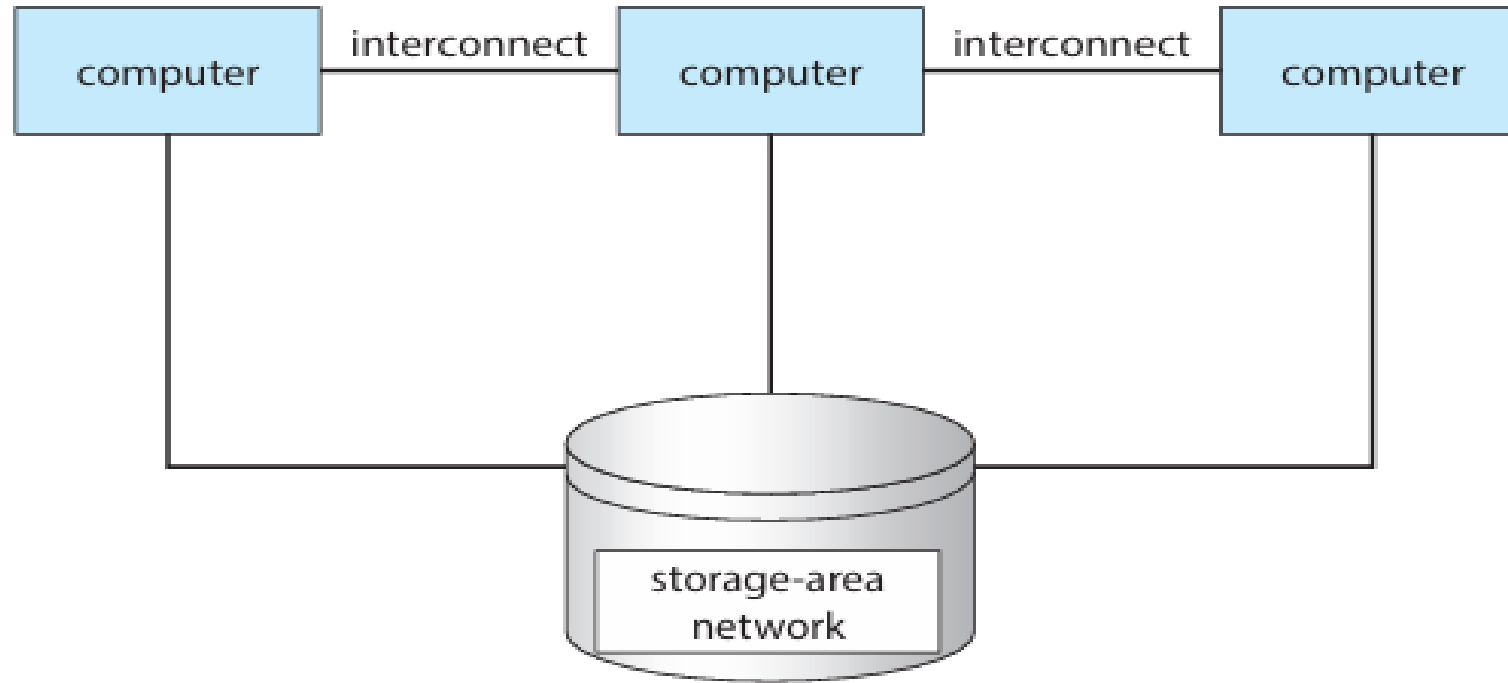



Fig. General structure of a clustered system.

- *Since a cluster consists of several computer systems connected via a network, clusters can also be used to provide high-performance computing environments.*

Can we have a definition of Operating System?

- *A more common definition, and the one that we usually follow, is that the operating system is the one program running at all times on the computer—usually called the **kernel**. Along with the kernel, there are two other types of programs: **system programs**, which are associated with the operating system but are not necessarily part of the kernel, and **application programs**, which include all programs not associated with the operation of the system.*
- Today, however, if we look at operating systems for mobile devices, we see that once again the number of features constituting the operating system is increasing.
- Mobile operating systems often include not only a **core kernel** but also **middleware** (a set of software frameworks that provide additional services to application developers).
- For example, each of the two most prominent mobile operating systems—Apple's iOS and Google's Android—features a core kernel along with middleware that supports databases, multimedia, and graphics (to name only a few)
- *In summary, for our purposes, the operating system includes the always running kernel, middleware frameworks that ease application development and provide features, and system programs that aid in managing the system while it is running.*

Operating-System Operations



As noted earlier, the initial program, or bootstrap program, tends to be simple. Typically, it is stored within the computer hardware in 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.

1. **Multiprogramming and Multitasking**
2. **Dual-Mode and Multimode Operation**
3. **Timer**

Multiprogramming:

- As the name suggests, **Multiprogramming** means more than one program can be active at the same time.
- One of the most important aspects of operating systems is the ability to run multiple programs, as a single program cannot, in general, keep either the CPU or the I/O devices busy at all times. Furthermore, users typically want to run more than one program at a time as well.
- **Multiprogramming** increases CPU utilization, as well as keeping users satisfied, by organizing programs so that the CPU always has one to execute. In a multiprogrammed system, a program in execution is termed a process.
- *The idea is as follows:* The operating system keeps several processes in memory simultaneously (as shown in Figure)

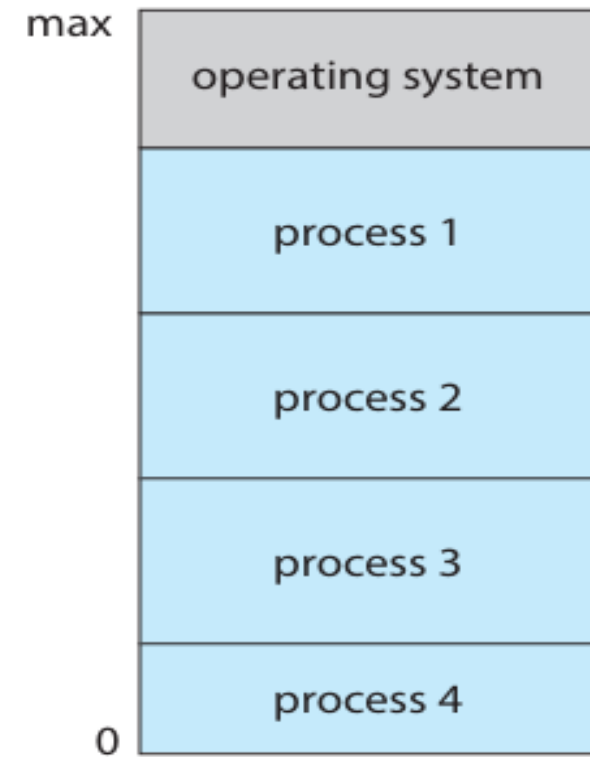


Fig. Memory layout for a multiprogramming system.

Multiprogramming:

- The operating system picks and begins to execute one of these processes. Eventually, the process 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 process.
- When that process needs to wait, the CPU switches to another process, and so on. Eventually, the first process finishes waiting and gets the CPU back. As long as at least one process needs to execute, the CPU is never idle.
- *This idea is common in other life situations. A lawyer does not work for only one client at a time, for example. While one case is waiting to go to trial or have papers typed, the lawyer can work on another case. If she has enough clients, the lawyer will never be idle for lack of work.*

Multitasking (Time Sharing):

- Multitasking is a logical extension of multiprogramming. In multitasking systems, the CPU executes multiple processes by switching among them, but the switches occur frequently, providing the user with a fast response time.
- **Multitasking** means working on multiple tasks simultaneously, such as using your computer while listening to music. Also, using a browser, search for something on the internet and create a word document that is your assignment.
- Multitasking is similar to multiprogramming in that the CPU is assigned to a process for a specified period of time, i.e., '**time slice**', after which the CPU 'Context switches' to another process. It runs various programs at the same time.

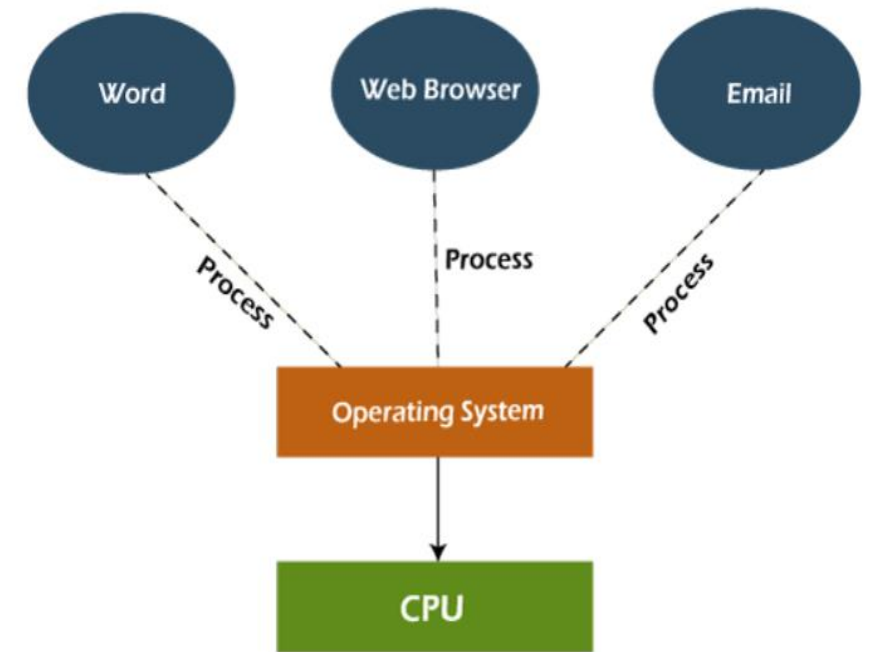


Fig. Multitasking

Multitasking (Time Sharing):

- The PC requires a huge memory to execute multitasking (RAM or ROM). Its primary goal is to improve the timing of the CPU's response. Users can engage with the system during multitasking, for example, by typing a letter while the printing process is running.
- Multitasking is a highly complicated system. It is based on the time slice principle, which assigns a fixed amount of time to each activity to be completed. It is especially useful when a program requires a high level of parallelism. It provides a set amount of time for each program to run.

Dual-Mode and Multimode Operation:

- Since the operating system and its users share the hardware and software resources of the computer system, a properly designed operating system must ensure that an incorrect (or malicious) program cannot cause other programs—or the operating system itself—to execute incorrectly.
- In order to ensure the proper execution of the system, we must be able to distinguish between the execution of operating-system code and user-defined code. The approach taken by most computer systems is to provide hardware support that allows differentiation among various modes of execution.
- At the very least, we need two separate modes of operation: **user mode** and **kernel mode** (also called **supervisor mode**, **system mode**, or **privileged mode**).
- A bit, called the **mode bit**, is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1).
- With the mode bit, we can distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user.

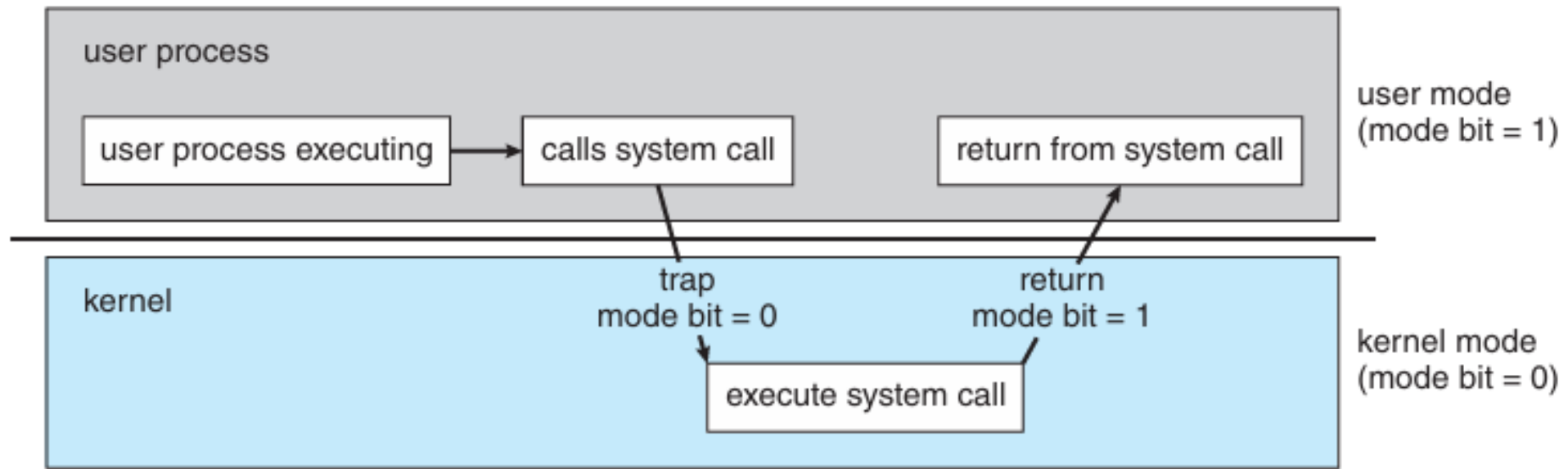


Fig. Transition from user to kernel mode.

- When the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a system call), the system must transition from user to kernel mode to fulfill the request.
- At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0).

Timer:

- We must ensure that the operating system maintains control over the CPU. We cannot allow a user program to get stuck in an infinite loop or to fail to call system services and never return control to the operating system.
- To accomplish this goal, we can use a timer. A timer can be set to interrupt the computer after a specified period. The period may be fixed (for example, $1/60$ second) or variable (for example, from 1 millisecond to 1 second).
- A **variable timer** is generally implemented by a fixed-rate clock and a counter. The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs.
- Before turning over control to the user, the operating system ensures that the timer is set to interrupt. If the timer interrupts, control transfers automatically to the operating system, which may treat the interrupt as a fatal error or may give the program more time. Clearly, instructions that modify the content of the timer are privileged instructions.

References

1. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, “Operating System Concepts,” Eleventh Edition (Willey).
2. <https://www.geeksforgeeks.org/>.
3. <https://www.javatpoint.com/>.
4. <https://www.tutorialspoint.com/>.