Chapter 1 – Introduction

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 and efficient manner. It is a software that manages the computer hardware and provides a basis for the application programs. Operating systems vary greatly in design for serving different computing environments. The design of a new operating system is a major task.

To understand the role of an operating system in a modern computer system, first understanding the organization and architecture of computer hardware (CPU, Memory, I/O devices and Storage) is important. In this chapter we provide a general overview of the major components of a contemporary computer system and the functions provided by the operating system.

Contents

1.5 Resource Management

1.5.1 Process Management
1.5.2 Memory Management
1.5.3 File-System Management
1.5.3 Mass-Storage Management
1.5.4 Cache Management
1.5.6 I/O System Management

1.6 Security and Protection

1.7 Virtualization

1.8 Distributed Systems

1.9 Kernel Data Structures

1.9.1 Lists, Stacks and Queues
1.9.2 Trees
1.9.3 Hash Functions and Maps
1.9.4 Bitmaps

1.10 Computing Environments

1.10.1 Traditional Computing
1.10.2 Mobile Computing
1.10.3 Client-Server Computing
1.10.4 Peer-to-Peer Computing
1.10.5 Cloud Computing
1.10.6 Real-Time Embedded Systems
1.11 Free and Open-Source Operating Systems

Chapter Objectives

- Describe the general organization of a computer system and the role of interrupts
- Describe the components in a modern multiprocessor computer system
- Illustrate the transition from user mode to kernel mode
- Discuss how operating systems are used in various computing environments
- Provide examples of free and open-source operating systems

1.1 What Operating Systems Do

A computer system can be divided roughly into **four** components: the **hardware**, the **operating system**, the **application programs** and the **user**.

The **hardware** (the central processing unit-CPU, the memory and the input/output devices) provides the basic computing resources for the system. The **application programs** (word processors, spreadsheets, compilers, web browsers etc.) define the ways in which these resources are used to solve users' computing problems. The **operating system** controls the hardware and coordinates its use among the various **application programs** for different users.

The role of an operating system (OS) can be viewed from following **two** viewpoints.

1.1.1 User View

The user's view of the computer varies according to the interface being used. The system in a laptop/desktop/mobile device is designed for the user to monopolize its resources in order to maximize the work done. The OS is designed mostly for ease of use, good performance, security and not merely for resource utilization.

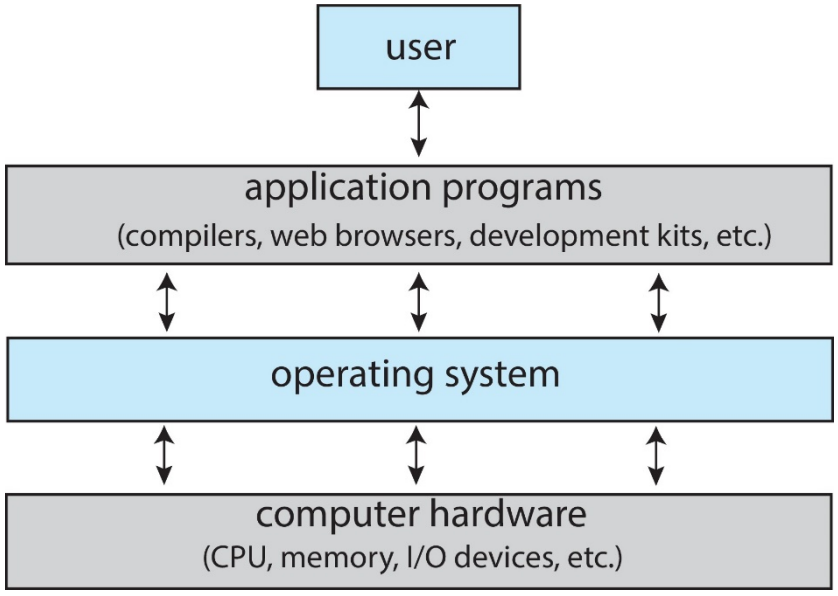

Figure 1 Abstract view of the components of a computer system

1.1.2 System View

To solve a problem in a computer system many resources (CPU time, memory space, storage space, I/O devices etc.) are required. OS is the program that is mostly involved with the hardware. So, from system's perspective, OS acts as a resource allocator. The OS must deal with numerous and possibly conflicting requests for resources by efficiently and fairly allocating them to specific programs and users. An operating system can also be viewed as a control program that manages the execution of user programs to prevent errors and improper use of the computer.

1.1.3 Defining Operating Systems

The term operating system covers many roles and functions because of the myriad designs and uses of computers. There is no completely adequate definition of an operating system.

"Everything a vendor ships as an operating system."

Operating system is the program running at all times on the computer (usually called **kernel**). Along with the kernel, there are two other types of programs: **system programs** (associated with the operating system but are not necessarily part of the kernel) and **application programs** (all programs not associated with the operation of the system).

Mobile operating systems often include a set of software frameworks with the core kernel that provide additional services (called **middleware**).

1.2 Computer System Organization

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. Each device controller controls a specific type of device (e.g., a disk drive, audio device or graphics display). It has some local buffer storage and a set of special purpose registers. It is responsible for moving the data between the peripheral device that it controls and its local buffer storage.

Typically, operating systems have a **device driver** for each **device controller**. The driver understands the device controller and provides with a uniform interface to the device. The CPU and the device controllers can execute in parallel, competing for memory cycles. There is a memory controller which ensures orderly access to the shared memory.

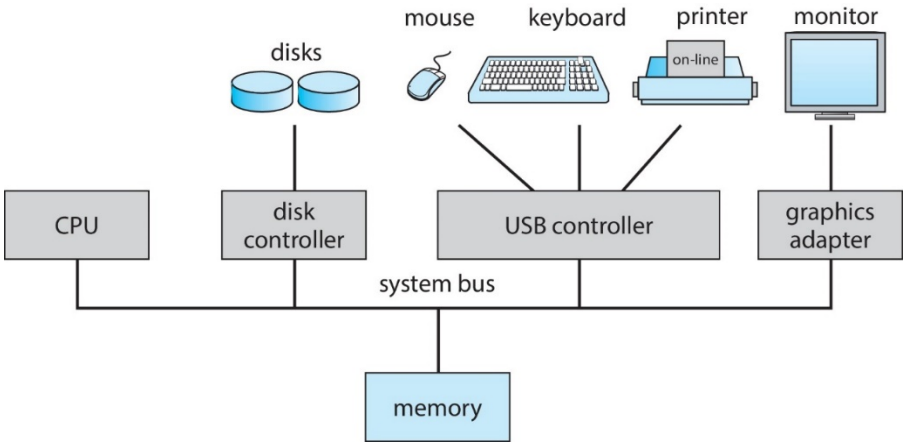

Figure 2 A typical PC computer system

In following subsections, we describe how such a system operates, focusing on **three** key aspects of the system: **Interrupts** (which alert the CPU to events that require attention), **Storage structure** and **I/O structure**.

1.2.1 Interrupts

Interrupts are hardware or software generated signal to the CPU, through system bus (it is the main communications path between the major components). This is how operating systems and hardware

interact. Interrupts are used throughout modern operating systems to handle asynchronous events. Device controllers and hardware faults raise interrupts.

Let's consider a program's typical I/O operation. To start an I/O operation, the device driver loads the appropriate registers in the device controller. The device controller examines the contents of these registers and determine what action to take (such as "read a character from the keyboard"). Then it starts the transfer of data from the device to its local buffer. Up on completion, the device controller notifies the device driver via an interrupt. The device driver then gives the control to other parts of the operating system, possibly returning the data or a pointer to it.

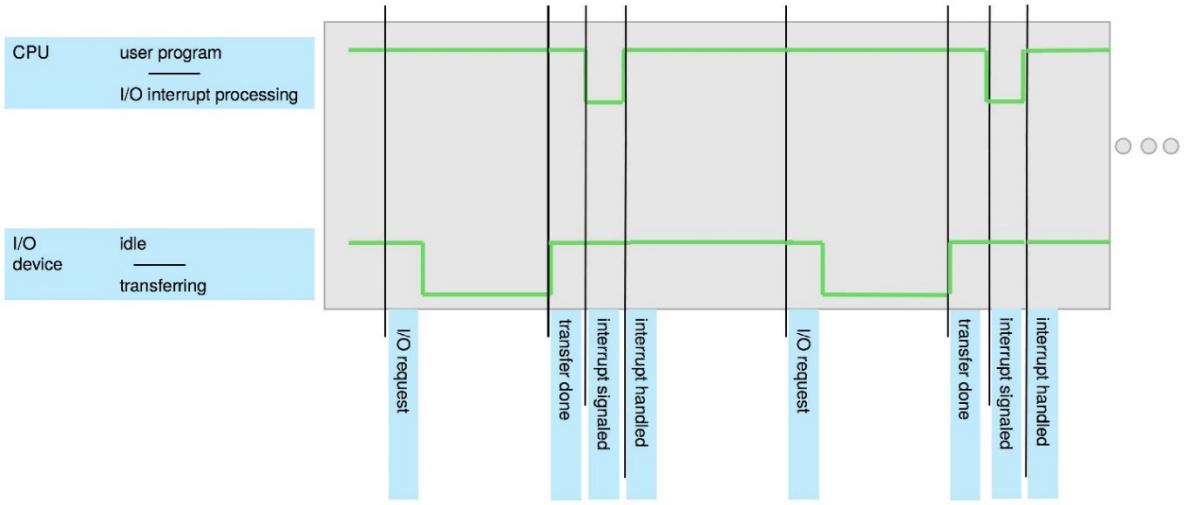

Figure 3 Interrupt timeline for a single program doing output

Interrupts are an important part of the computer architecture. When the CPU is interrupted, it stops what it is doing and immediately transfers execution to **interrupt service routine**. The interrupt service routine executes; on completion the CPU resumes the interrupted computation. The state information of whatever was interrupted must be saved beforehand in order to resume after servicing the interrupt.

As the interrupts must be handled quickly, the locations of interrupt service routines are stored in a table in low memory. This array is called **interrupt vector**, which indices denote specific interrupt requests.

The CPU hardware has a wire called the interrupt request line. The CPU checks it after executing every instruction. When a device controller **raises** an interrupt by asserting a signal on the interrupt request line, the CPU **catches** the interrupt by reading the interrupt number and **dispatches** it to the interrupt handler by executing the instructions stored for that number and the handler **clears** the interrupt by servicing the device.

Most CPUs have two interrupt request lines, **non-maskable** (reserved for critical events, cannot be turned off) and **maskable** (can be turned off by the CPU before execution of critical instructions, used by device controllers to request service).

In the interrupt vector table for Intel Processors, the events from 0-31 are non-maskable and used to signal various error conditions. The events from 31-255 are maskable and used for purposes such as device generated interrupts.

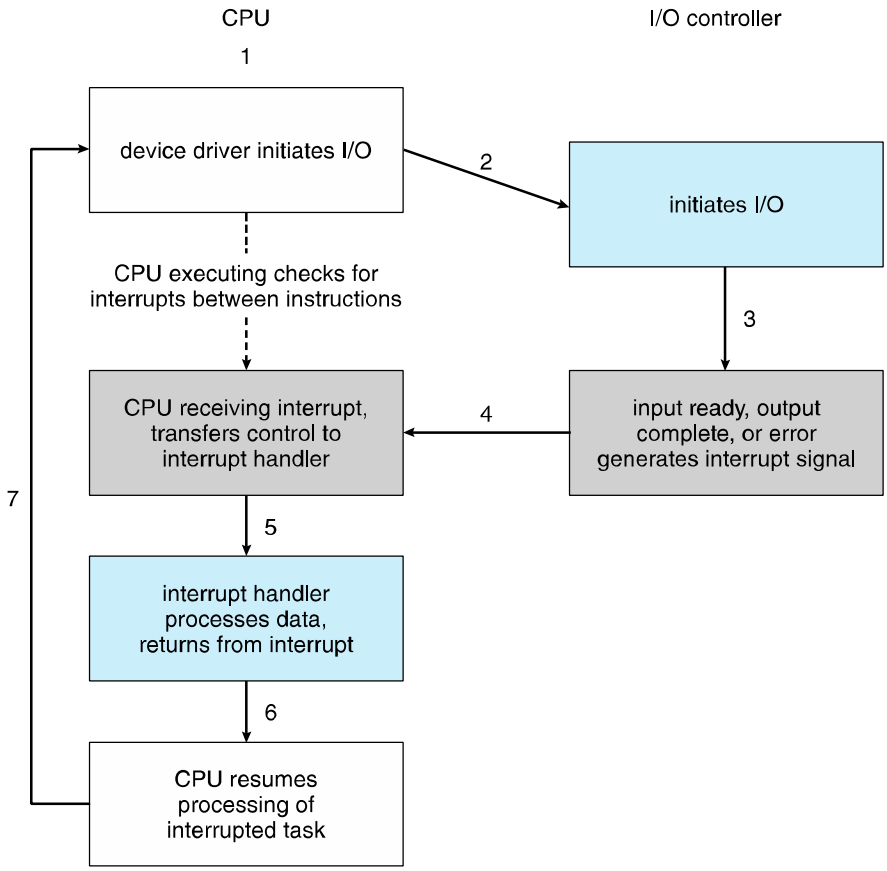

Figure 4 Interrupt driven I/O cycle

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Figure 5 Intel Processor event-vector table

1.2.2 Storage Structure

In order to run, a program must first be loaded into memory as the CPU can load instructions only from main memory (rewritable, random access memory - **RAM**). RAM is volatile (loses its content when power is turned off). Computers use other forms of memory for various purposes.

The first program that runs on computer power-on is a **bootstrap program**, which then loads the operating system. Since RAM is volatile, it cannot be stored there. For this, computers use electrically erasable programmable read only memory (**EEPROM**) and other forms of firmware (storage that is infrequently written to and is nonvolatile).

All forms of memory provide an array of bytes and each byte has its own address. Interaction is achieved through a sequence of load and store instructions to specific memory addresses. The load instruction moves a byte or word from main memory to an internal register within the CPU and the store instruction moves the content of a register to main memory.

As the main memory (RAM) is volatile and too small to store all the programs and data permanently, most computer systems use secondary storage (hard disk drives- HDDs and nonvolatile memory-NVM) for that purpose. Nonvolatile storage retains its contents when power is lost. Most programs are stored in secondary storage until they are loaded into memory. Secondary storage is slower than the main memory. In general, there is a trade-off between size and speed, with smaller and faster memory closer to the CPU.

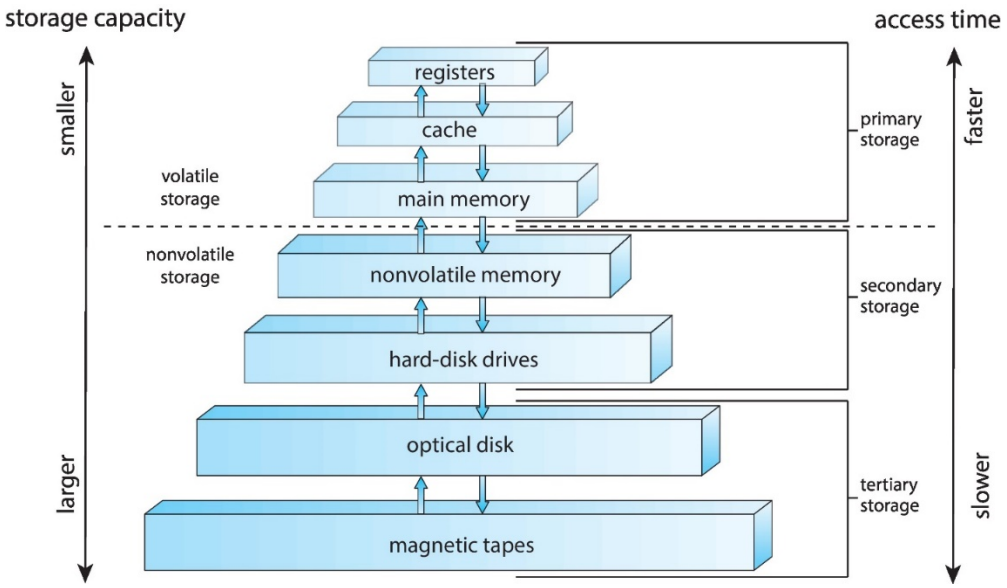

Figure 6 Storage device hierarchy

1.2.3 I/O Structure

A large portion of operating system code is dedicated to managing I/O to ensure reliable performance across different types of devices.

As mentioned earlier, the device components in a general-purpose computer system exchange data via a common bus. Interrupt driven I/O is fine for moving small amounts of data, but it creates high overhead for bulk data movement. To solve this, direct memory access (DMA) is used. After setting up

buffers, pointers and counters for the I/O device, the device controller transfers an entire block of data directly to or from the device and main memory, with no intervention by the CPU. Only one interrupt is generated per block as the notification of completion, rather than generating one interrupt per byte. While the device controller performs I/O, the CPU is available for executing other instructions.

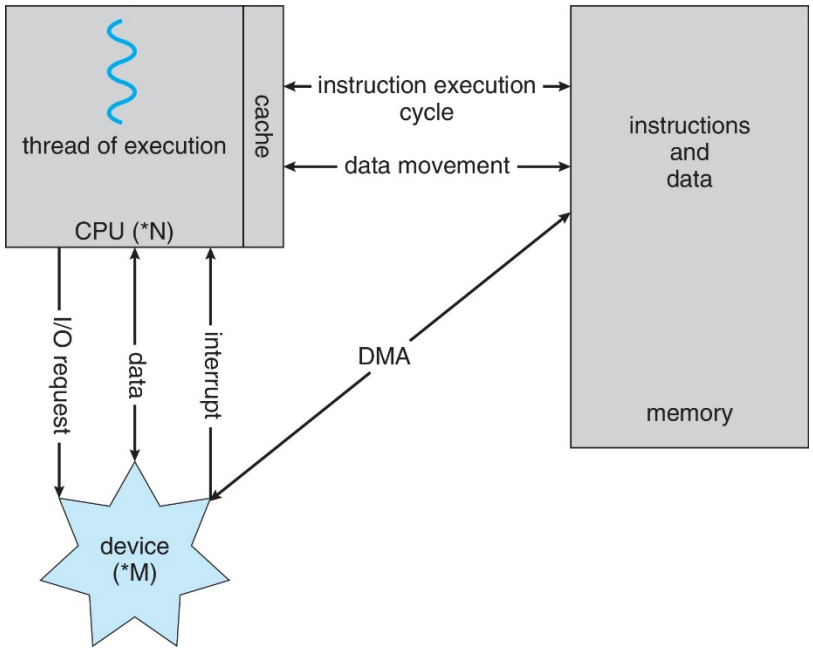

Figure 7 How a modern computer system works

1.3 Computer System Architecture

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

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 (executes instructions, stores data locally in registers)*
- *Multicore – Multiple cores on the same CPU*
- *Multiprocessor – Multiple processors*

1.3.1 Single Processor Systems

These systems usually contain one CPU with a single processing core. The one main CPU with its core is capable of executing a general-purpose instruction set. These systems may have other special-purpose device specific processors.

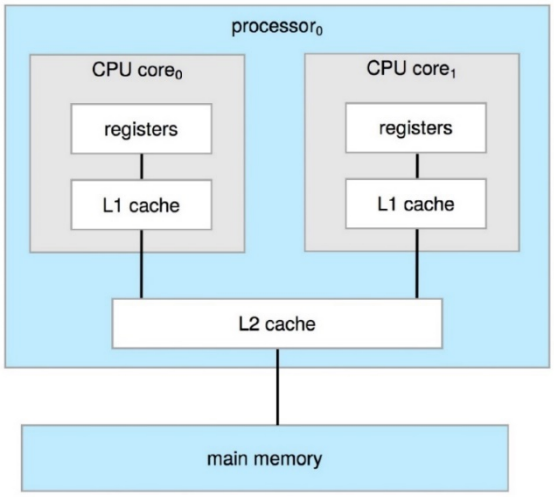

Figure 8 A dual-core design with two cores on the same chip.

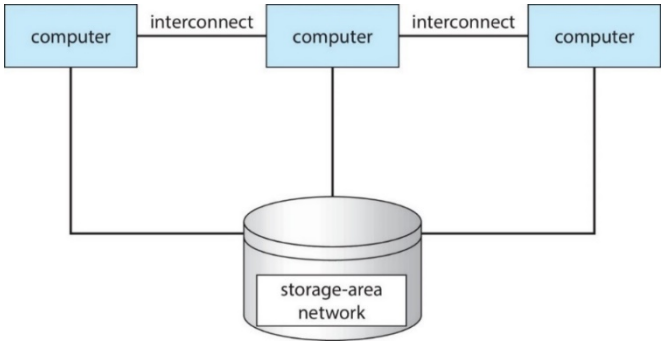

Figure 9 General Structure of a clustered system.

1.3.2 Multiprocessor Systems

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. These systems provide increased throughput and reliability. Now a days these include multicore systems in which multiple computing cores reside on a single chip. These are faster than single cores because on chip communication is faster than between chip communication, as well as these consumes less power.

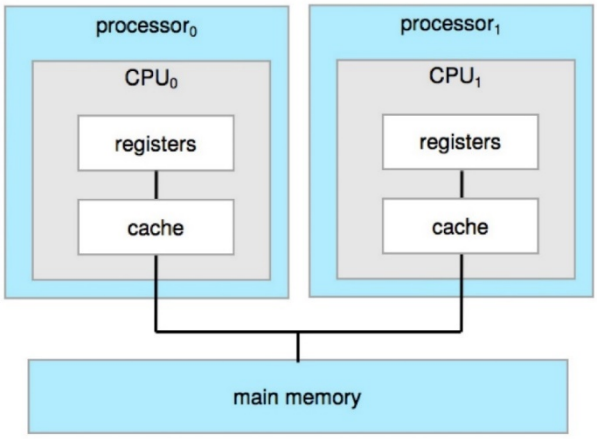

Figure 10 Symmetric multiprocessing architecture

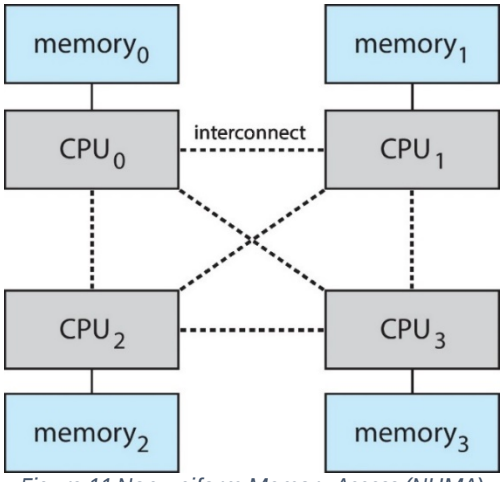

Figure 11 Non-uniform Memory Access (NUMA) multiprocessing architecture

1.3.3 Clustered Systems

These systems gather together multiple CPUs. These differ from multiprocessor systems as they are composed of two or more individual systems – or nodes joined together where each node is typically a multicore system. Such systems are considered loosely coupled and used to provide high availability services.

1.4 Operating System Operations

When a computer is powered up or rebooted, a simple program called **bootstrap** (stored in firmware) initializes all aspects of the system - from CPU registers to device controllers to memory contents. It loads the OS kernel into memory and start executing it. After initialization, the system usually waits for interrupts. When a hardware or software interrupt occurs, an operating system service is performed by executing a special operation called a **system call**.

Some other system programs (other than the kernel) called **system daemons** are loaded into memory at boot time to provide some services; these run the entire time the kernel runs.

1.4.1 Multiprogramming and Multi-tasking

A program in execution is called a process. In a **multi-programmed** system, the OS keeps several processes in memory simultaneously and starts executing one of them. When this process waits for something such as an I/O operation, instead of keeping the CPU idle, the OS switches to another process and starts executing that. 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. Multiprogramming increases CPU utilization (as a single program cannot keep the CPU or I/O devices busy all the time) and user satisfaction (as they typically need more than one program at a time to run).

Multitasking is a logical extension of multiprogramming, where the CPU executes multiple processes by frequently switching among them to provide the user with a fast response time.

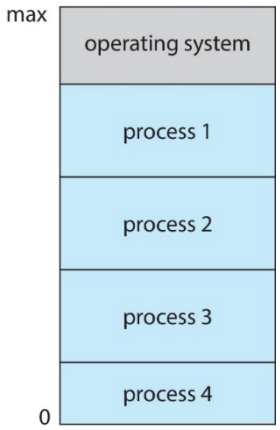

Figure 12 Memory layout for a multiprogramming system.

Multiprogramming and Multitasking requires proper memory management, CPU scheduling, file system management, process synchronization and communication etc. (discussed in subsequent chapters)

1.4.2 Dual-Mode and Multimode Operation

As the OS and its users share the hardware and software resources of the computer system, in order to ensure proper execution and to prevent malicious usage, it is necessary to distinguish between the execution of operating system code and user defined code. Most computer systems provide hardware support for this purpose.

Usually there are two **modes** of operation: **user mode** and **kernel mode** (system/privileged mode). A **mode bit** in the hardware indicates the current mode. When the mode bit is set to 0 (kernel mode), a

task is executed on behalf of the operating system. When a user program requests a service from the operating system (via a system call), the system switches from kernel mode to user mode. When the computer system executes user applications, the mode bit is set to 1 that indicates user mode.

At system boot time, the hardware is in kernel mode. The OS then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (i.e., set mode bit to 0).

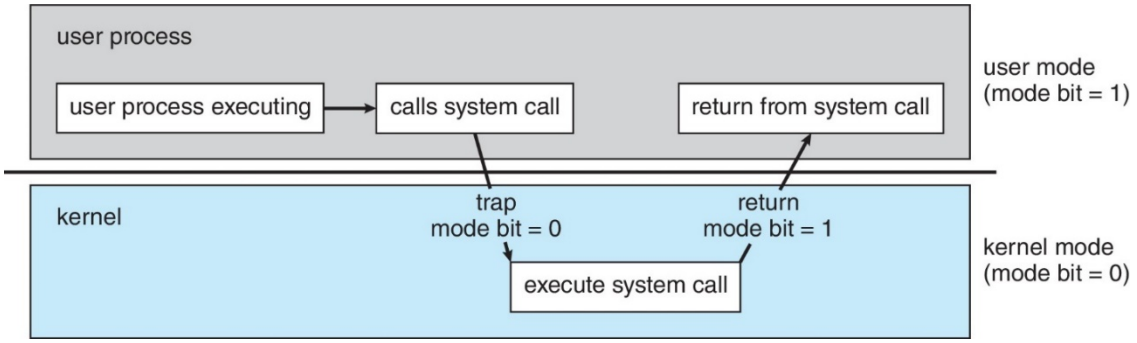

Figure 13 Transition from user to kernel mode.

The dual mode of operation protects the operating system from errant users. Some machine instructions (privileged instructions) can only be executed in kernel mode, so no user application can execute them illegally.

1.4.3 Timer is generally implemented by a fixed rate clock and a counter. It can be used to avoid infinite loops by interrupting the computer after a specified period.

1.5 Resource Management

The operating system is a resource manager. The system's CPU, memory space, file-storage space and I/O devices are among the resources that the OS must manage.

1.5.1 Process Management

A program in **execution** is called a **process**. A program is a passive entity (files stored on disk), whereas a process is an active entity, e.g., A compiler, a word processor, a social media app on a mobile device etc.

A process needs system resources as CPU time, memory, files and I/O devices to perform. These are allocated by the OS to the process while it is running. A **single threaded** process has **one** program counter specifying the next instruction to execute. The CPU executes the instructions sequentially one after another. A **multi-threaded** process has **multiple** program counters. A process is the unit of work in a system. A system consists of a collection of processes, some of them are operating system processes and the rest are user processes; all of them can execute concurrently. The OS performs the following activities for managing the processes:

- Creating and deleting both user and system processes
- Scheduling processes and threads on the CPUs
- Suspending and resuming processes
- Providing mechanism for process synchronization
- Providing mechanism for process communication

1.5.2 Memory Management

Main memory is central to the operation of a modern computer system. Main memory is a large array of bytes, each byte has its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices. The CPU reads instructions from main memory during the instruction-fetch cycle and both reads and writes data from it during the data-fetch cycle. It is generally the only large storage device that the CPU is able to address and access directly. For example, for the CPU to process data from disk, those data must first be transferred to main memory by CPU generated I/O calls. Instructions also must be in memory for the CPU to execute them.

In order to execute, a program must be mapped to absolute address and loaded into memory. During execution it accesses program instructions and data from memory by generating these absolute addresses. When it terminates, the memory is freed and the next program is loaded to execute.

The OS performs the following activities for managing the main memory:

- Keeping track of memory parts currently being used and which process is using them
- Allocating and de-allocating memory space as needed
- Deciding which processes and data to move into and out of memory

1.5.3 File-System Management

Operating system provides a uniform, logical view of information storage for the users' convenience. The OS abstracts from the physical properties of its storage devices to define a logical storage unit, the **file**. The OS maps files onto physical media and accesses these files via the storage devices. The storage devices have their own properties including access speed, capacity, data-transfer rate and access method etc. The OS implements the abstract concept of a file by managing mass storage media and the devices that control them.

The OS performs the following activities for managing the file-system:

- Creating and deleting files
- Creating and deleting directories to organize files
- Supporting primitives for manipulating files and directories
- Mapping files onto mass storage
- Backing up files on stable (nonvolatile) storage media

1.5.3 Mass-Storage Management

To back up main memory, secondary storage is must needed. Most modern computer systems use HDDs and NVM devices as the principal on-line storage media for both programs and data. Most programs are stored on these devices until loaded into memory for executing.

The OS performs the following activities for managing the secondary storages:

- Mounting and un-mounting
- Free space management
- Storage allocation
- Disk scheduling
- Partitioning
- Protection

1.5.4 Cache Management

Information is normally kept in some storage system (such as main memory). As it is used, it is copied into a faster storage system – the **cache**, on a temporary basis. When a particular piece of information is needed, first the cache is checked if it is there. If it does, then it is used directly. Otherwise, the information is used from the source and kept in the cache assuming that it will be needed again soon. This reduces the waiting time needed for fetching data from main memory.

Caches have limited size, so proper cache management is important. In a multitasking environment, where the CPU is switched back and forth among various processes, both hardware level and software level management are necessary to maintain the coherency.

Level	1	2	3	4	5
Name	registers	cache	main memory	solid-state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25-0.5	0.5-25	80-250	25,000-50,000	5,000,000
Bandwidth (MB/sec)	20,000-100,000	5,000-10,000	1,000-5,000	500	20-150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Figure 14 Characteristics of various types of storage

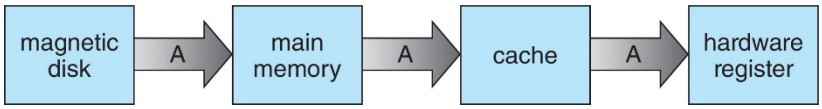

Figure 15 Migration of integer A from disk to register.

1.5.6 I/O System Management

One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user. For example, Unix has I/O subsystem to manage I/O, which consists of several components such as:

- A memory management component that includes buffering, caching and spooling.
- A general device driver interface
- Drivers for specific hardware devices.

The I/O subsystem is responsible for:

- Providing interfaces to other system components
- Managing devices
- Transferring data
- Detecting I/O completion

1.6 Security and Protection

The computer system has multiple users and allows the concurrent execution of multiple processes, so the access to data must be regulated. OS ensures that files, memory segments, CPU, and other resources can only be operated on by the processes that have proper authorization. For example, a process can execute only within its own address space, users cannot directly access the device control registers to maintain integrity of the various peripheral devices etc.

Protection is any mechanism for controlling the access of processes or users to the resources defined by the system. A protection-oriented system can distinguish between authorized and unauthorized usage.

It is the job of **security** to defend a system from external and internal attacks such as **viruses** and **worms, denial-of-service attacks** (which use all of a system's resource and so keep legitimate users out of the system), **identity theft**, and the **theft of service** (unauthorized use of a system). Prevention of these attacks can be done by an OS function, or policy or additional software.

Protection and security require the system to be able to distinguish among all its users. Most OS maintain a list of user names and associated user IDs. When a user logs in, the authentication stage determines its appropriate user ID. That ID is associated with all of the user's processes and threads.

1.7 Virtualization

Virtualization is a technology that allows us to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards etc.) into several different execution environment as if each environment is running on its own private computer. These can be viewed as different individual OS that runs at the same time and can interact with each other. It allows the OS run as applications within other operating systems. A user of a virtual machine can switch among the various OS in the same way a user can switch among various processes running concurrently in a single operating system.

Virtualization also relates to **emulation** - simulating computer hardware in software, when the source CPU type is different from the target CPU type. It allows an entire OS for one platform to run on another.

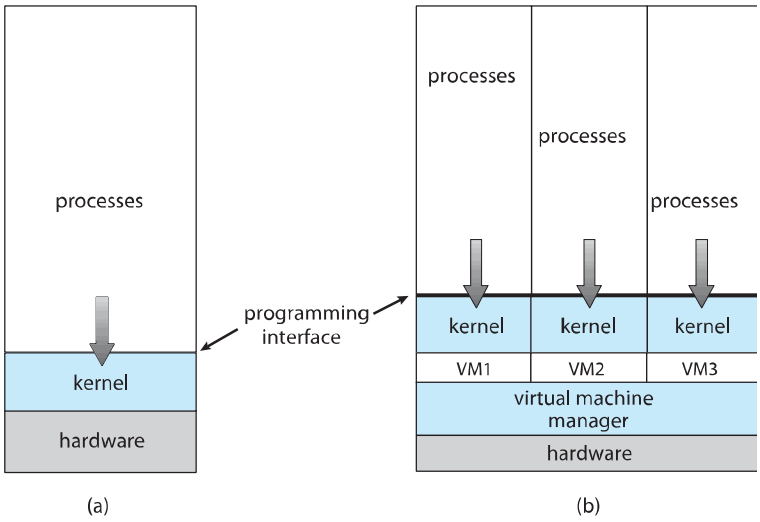

Figure 16 A computer running (a) a single operating system and (b) three virtual machines.

1.8 Distributed Systems

A distributed system is a collection of physically separate, possibly heterogeneous computer systems networked to provide users with access to the various resources. This increases computation speed, functionality, data availability and reliability.

A network is a communication path between two or more systems. **A network OS** is an operating system that provides features such as file sharing across the network and allows different processes on different computers to communicate. A computer running a network OS acts autonomously from all other computers. **A distributed OS** provides a less autonomous environment. Here different computers communicate closely enough to provide the illusion that only a single OS controls the network.

1.9 Kernel Data Structures

Here are several fundamental data structures that are used extensively in different OS implementations.

1.9.1 Lists (and Stacks/Queues) are used by the kernel algorithms directly or for constructing more powerful data structures, such as **stacks** and **queues**. An OS often uses a stack when invoking function calls. Parameters, local variables and the return addresses are pushed onto the stack when a function is called. Queues are also common in OS – tasks that are waiting to be run on an available CPU are often organized in queues.

1.9.2 Trees: Linux uses a balanced binary search tree as a part of its CPU-scheduling algorithm.

1.9.3 Hash Functions and Maps are used extensively in operating systems for their O(1) search complexity (depending on the implementation). One use of a hash function is to implement a hash map. For example, a user name can be mapped to a password; can be used in password authentication.

1.9.4 Bitmaps are commonly used to represent the availability of a large number of resources.

1.10 Computing Environments

Here we discuss how operating systems are used in a variety of computing environments.

1.10.1 Traditional Computing refers to standalone general-purpose machines where these interconnect with others via internet using web portals. Both user processes and system processes are managed accordingly so that each frequently gets a slice of computer time.

1.10.2 Mobile Computing refers to computing on handheld smartphones and tablet computers.

1.10.3 Client-Server Computing refers that server systems satisfy requests generated by client systems over the network.

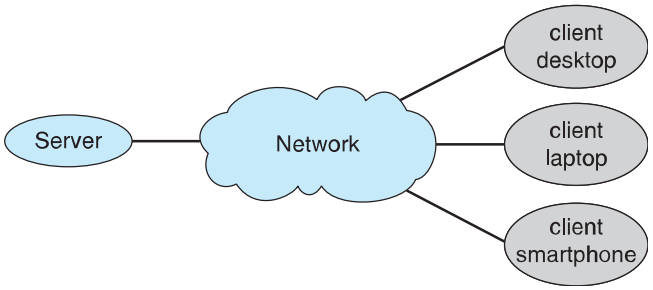

Figure 17 General structure of a client-server system.

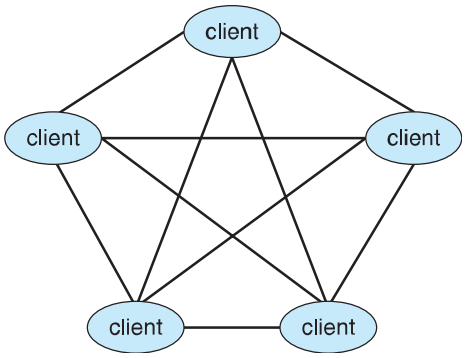

Figure 18 Peer-to-peer system with no centralized service.

1.10.4 Peer-to-Peer Computing refers to distributed systems where clients and servers are not distinguished from one another. Instead, all nodes within the system are considered peers and each may act as either a client or a server, depending on whether it is requesting or providing a service. In a client server system, the server is a bottleneck; but in a peer-to-peer system, services can be provided by several nodes distributed throughout the network.

1.10.5 Cloud Computing is a type of computing that delivers computing, storage and even applications as a service across a network.

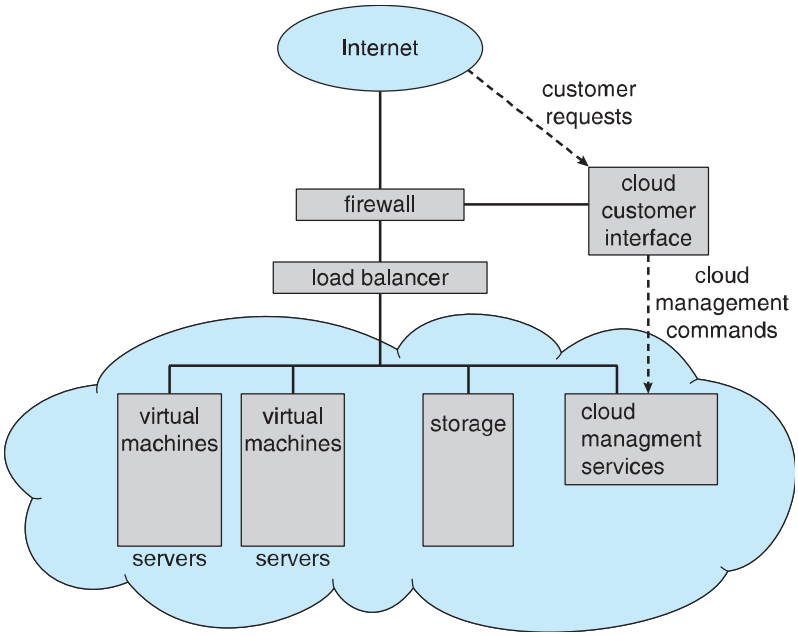

Figure 19 Cloud computing

1.10.6 Real-Time Embedded Systems are most prevalent, special and limited purpose systems used in various devices.

1.11 Free and Open-Source Operating Systems: GNU/Linux, BSD Linux, Solaris etc.