Introduction, Evolution and Structures of Operating System

Background of operating system

- A modern computer consists of one or more processors, some main memory, disks, printers, a keyboard, a mouse, a display, network interfaces, and various other input/output devices.
- If every application programmer had to understand how all these things work in detail, no code would ever get written. Furthermore, managing all these components and using them optimally is an exceedingly challenging job.
- For this reason, computers are equipped with a layer of software called **the operating system,** whose job is to provide user programs with a better, simpler, cleaner, model of the computer and to handle managing all the resources just mentioned.
- An **operating system** *is system software that manages a computer's hardware*. It also provides a basis for application programs and acts as an *intermediary between the computer user and the computer hardware.*
- Operating system is a program that acts as an intermediary between a user of a computer and the computer hardware. The goals of operating system are :
    - 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 Structure

- Computer system can be divided into four components:
    - Hardware – provides basic computing resources
        - CPU, memory, I/O devices
    - Operating system
        - Controls and coordinates use of hardware among various applications and users
    - Application programs – define the ways in which the system resources are used to solve the computing problems of the users
        - Word processors, compilers, web browsers, database systems, video games
    - Users
        - People, machines, other computers

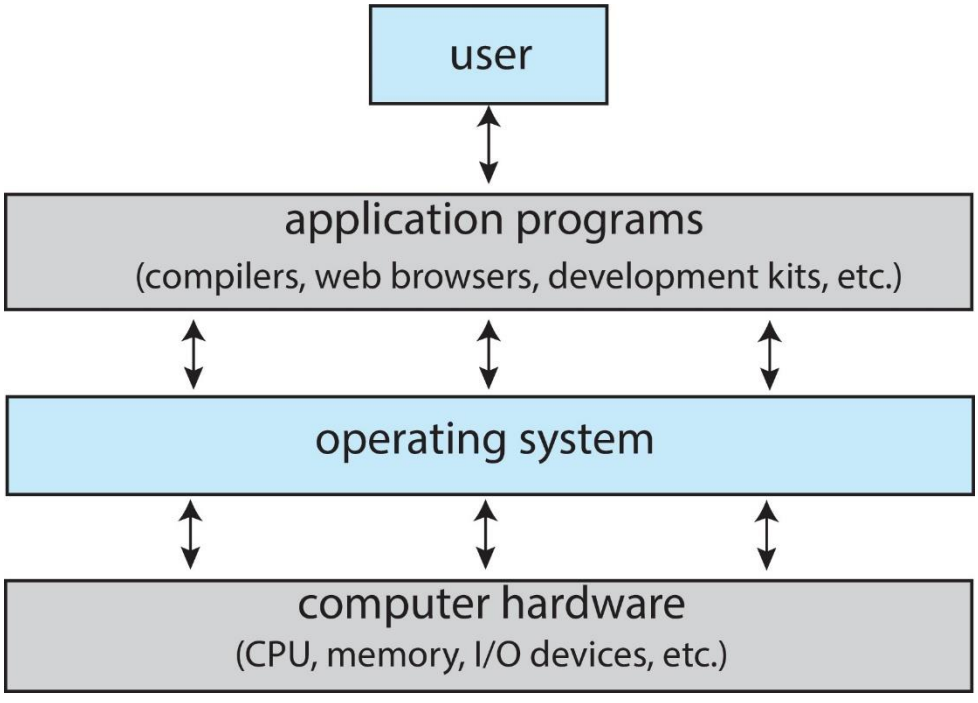

Figure:: Abstract view of the components of a computer system.

The place of Operating system

- A simple overview of the main components under discussion here is given in the figure below.

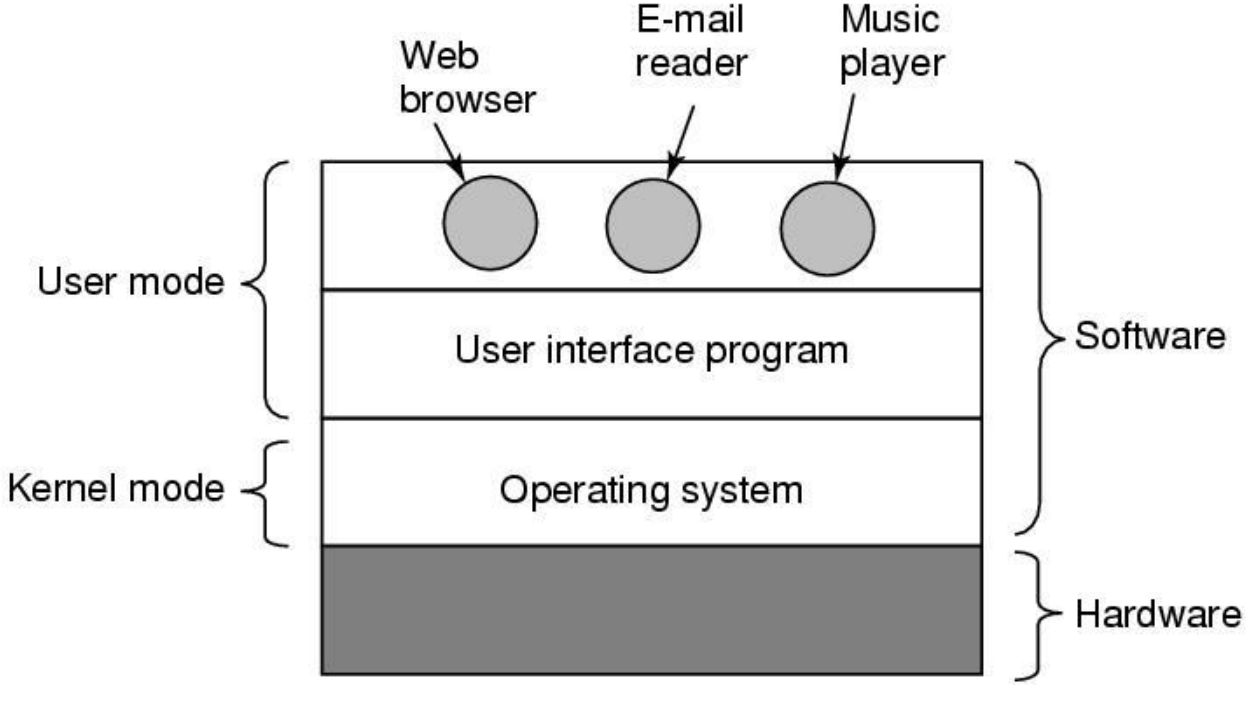

Figure: Where the operating system fits in.

- Here we see the hardware at the bottom. On top of the hardware is the software.
- Most computers have two modes of operation: **kernel** mode and **user mode**.

  - — The operating system, the most fundamental piece of software, runs in kernel mode (also called **supervisor** mode).
  - — In this mode it has complete access to all the hardware and can execute any instruction the machine is capable of executing.
- The rest of the software runs in user mode, in which only a subset of the machine instructions is available.
- In particular, those instructions that affect control of the machine or do I/O) Input/Output" are forbidden to user-mode programs.
- The user interface program, shell or GUI, is the lowest level of user-mode software, and allows the user to start other programs, such as a Web browser, email reader, or music player. These programs, too, make heavy use of the operating system.
- The placement of the operating system is shown in above figures. **It runs on the bare hardware and provides the base for all the other software.**

An Operating System (OS) is a system software which acts as an interface between a computer user and computer hardware. It performs all the basic tasks like memory management, file management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Roles of OS

- — Implementing the user interface
- — Sharing hardware among users
- — Allowing users to share data among themselves
- — Preventing users from interfering with one another
- — Scheduling resources among users
- — Facilitating input/output
- — Recovering from errors
- — Accounting for resource usage
- — Facilitating parallel operations
- — Organizing data for secure and rapid access
- — Handling network communications

It is hard to pin down what an operating system is other than saying it is the software that runs in kernel mode—and even that is not always true.

- No universally accepted definition of OS
- Operating systems perform two essentially unrelated functions:
    1. Providing application programmers (and application programs, naturally) a clean Abstract set of resources instead of the messy hardware ones (functioning as an Extended Machine)
    2. Managing the hardware resources (functioning as a Resource Manager)

The Operating System as an Extended Machine

- At the Machine level the structure of a computer's system is complicated to program, mainly for input or output. Programmers do not deal with hardware. They will always mainly focus on implementing software. Therefore, a level of abstraction is supposed to be maintained.
- Operating systems provide a layer of abstraction for using disk such as files.
- OS creates a high level of abstraction for the programmer
- Example—(Disk I/O Operation)
    - Disk contains a list of named files
    - Each file can be opened for read write
    - After read write is complete, close that file
    - No any detail to deal
- The abstraction hides the truth about the hardware from the programmer and presents a simple view of the named files that can be read and written.
- This abstraction is the key to managing all this complexity. Good abstractions turn a nearly impossible task into two manageable ones. The first is defining and implementing the abstractions. The second is using these abstractions to solve the problem at hand. One abstraction that almost every computer user understands is the file, as mentioned above.
- OS prevents the user with the equivalent of an extended machine or virtual machine that is easier to program than the underlying hardware.
- It should be noted that the **operating system's real customers are the application programs** (via the application programmers, of course). They are the ones who deal directly with the operating system and its abstractions.

- In contrast, **end users deal with the abstractions provided by the user interface, either a command- line shell or a graphical interface**. While the abstractions at the user interface may be similar to the ones provided by the operating system, this is not always the case. To make this point clearer, consider the normal Windows desktop and the line-oriented command prompt. Both are programs running on the Windows operating system and use the abstractions Windows provides, but they offer very different user interfaces.

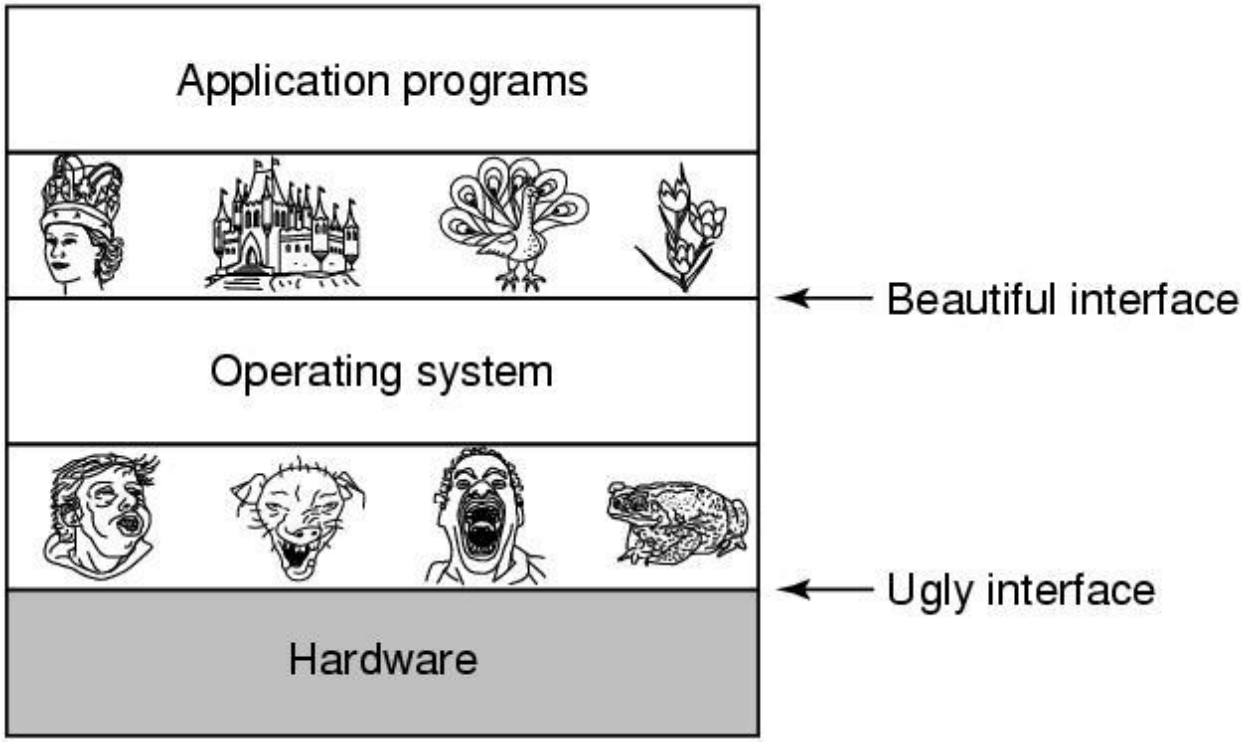

Figure: Operating systems turn ugly hardware into beautiful abstractions.

The Operating System as a Resource Manager

- Now-a-days all modern computers consist of processors, memories, timers, network interfaces, printers, and so many other devices.
- The operating system provides for an orderly and controlled allocation of the processors, memories, and I/O devices among the various programs in the bottom-up view.
- Operating system allows multiple programs to be in memory and run at the same time.
- Resource management includes multiplexing or sharing resources in two different ways: in **time** and in **space.**
- When a resource is **time multiplexed**, different programs or users take turns using it. First one of them gets to use the resource, then another, and so on.

- For example, with only one CPU and multiple programs that want to run on it, the operating system first allocates the CPU to one program, then, after it has run long enough, another program gets to use the CPU, then another, and then eventually the first one again.
- Determining how the resource is time multiplexed—who goes next and for how long—is the task of the operating system.
- Another example of time multiplexing is sharing the printer.
    - When multiple print jobs are queued up for printing on a single printer, a decision has to be made about which one is to be printed next.

- In **space multiplexing**, instead of the customers taking a chance, each one gets part of the resource.
    - For example − Main memory is divided into several running programs, so each one can be resident at the same time.
    - Another resource that is space multiplexed is the disk. In many systems a single disk can hold files from many users at the same time. Allocating disk space and keeping track of who is using which disk blocks is a typical operating system task.

Operating system is a software that manages resources and provides abstractions.

Hardware review (Processors, Memory, I/O devices, Buses)

- An operating system is intimately tied to the hardware of the computer it runs on. It extends the computer's instruction set and manages its resources. To work, it must know a great deal about the hardware, at least about how the hardware appears to the programmer.
- Conceptually, a simple personal computer can be abstracted to a model resembling that of following Figure. The CPU, memory, and I/O devices are all connected by a system bus and communicate with one another over it. Modern personal computers have a more complicated structure, involving multiple buses.

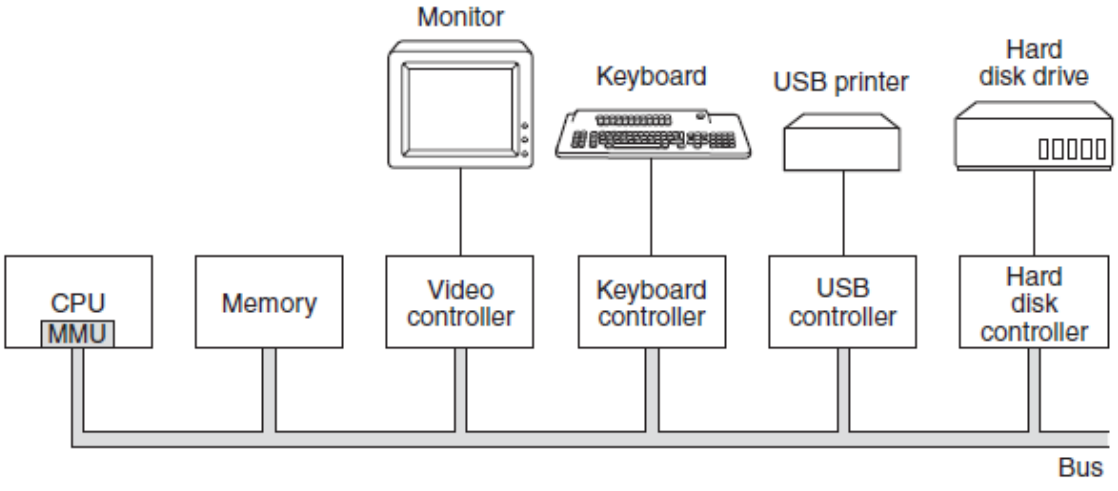

Figure: Some of the components of a simple personal computer.

Processors

- The ‘‘brain’’ of the computer is the CPU. It fetches instructions from memory and executes them.
- The basic cycle of every CPU is to fetch the first instruction from memory, decode it to determine its type and operands, execute it, and then fetch, decode, and execute subsequent instructions. The cycle is repeated until the program finishes.
- Each CPU has a specific set of instructions that it can execute. Thus an x86 processor cannot execute ARM programs and an ARM processor cannot execute x86 programs.
- Because accessing memory to get an instruction or data word takes much longer than executing an instruction, all CPUs contain some registers inside to hold key variables and temporary results.
- Thus the instruction set generally contains instructions to load a word from memory into a register, and store a word from a register into memory.
    - Other instructions combine two operands from registers, memory, or both into a result, such as adding two words and storing the result in a register or in memory
- In addition to the general registers used to hold variables and temporary results, most computers have several **special registers** that are visible to the programmer.
    - One of these is the **program counter**, which contains the memory address of the next instruction to be fetched.
    - After that instruction has been fetched, the program counter is updated to point to its successor.

  - Another register is the **stack pointer**, which points to the top of the current stack in memory. The stack contains one frame for each procedure that has been entered but not yet exited. A procedure's stack frame holds those input parameters, local variables, and temporary variables that are not kept in registers.
  - Another register is the **PSW (Program Status Word).** This register contains the condition code bits, which are set by comparison instructions, the CPU priority, the mode (user or kernel), and various other control bits.
- The operating system must be fully aware of all the registers. When time multiplexing the CPU, the operating system will often stop the running program to (re)start another one. Every time it stops a running program, the operating system must save all the registers so they can be restored when the program runs later,
- A **multicore processor** refers to a single physical chip that contains multiple independent processing units known as cores. Each core is capable of executing tasks independently, and they share various resources such as cache and memory controllers. In a multicore system, multiple cores are integrated into a single processor chip, which allows for parallel processing and improved performance for multitasking and multithreaded applications.
- A **multiprocessor system**, on the other hand, refers to a system that consists of multiple physical processors. Each processor typically contains its own set of resources, including instruction execution units, cache, memory controllers, and other components. These individual processors can work together to execute tasks simultaneously, enabling parallel processing.

Memory

- The second major component in any computer is the memory. Ideally, a memory should be extremely fast (faster than executing an instruction so that the CPU is not held up by the memory), abundantly large, and dirt cheap. No current technology satisfies all of these goals, so a different approach is taken.
- The memory system is constructed as a hierarchy of layers, as shown in following figure.

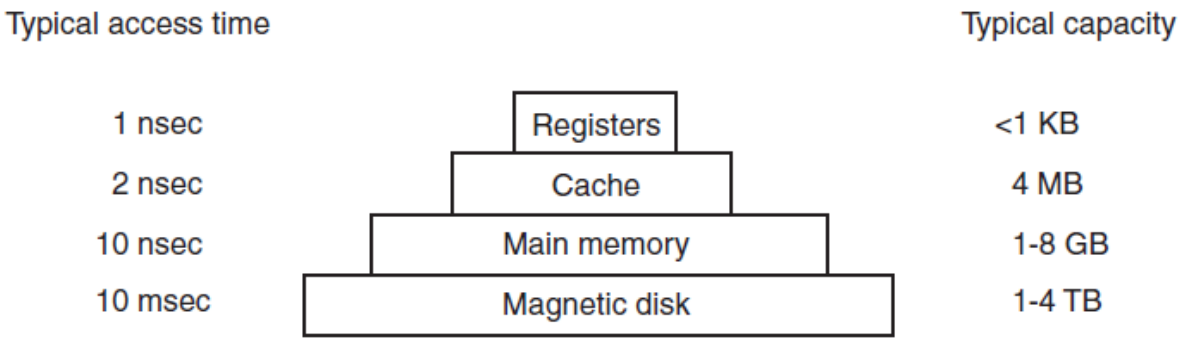

Figure: A typical memory hierarchy. The numbers are very rough approximations.

- The top layers have higher speed, smaller capacity, and greater cost per bit than the lower ones, often by factors of a billion or more
- The top layer consists of the **registers** internal to the CPU. They are made of the same material as the CPU and are thus just as fast as the CPU. Consequently, there is no delay in accessing them.
- Next comes the **cache** memory, which is mostly controlled by the hardware.
    - Main memory is divided up into **cache lines**, typically 64 bytes, with addresses 0 to 63 in cache line 0, 64 to 127 in cache line 1, and so on. The most heavily used cache lines are kept in a high-speed cache located inside or very close to the CPU.
    - When the program needs to read a memory word, the cache hardware checks to see if the line needed is in the cache. If it is, called a **cache hit**, the request is satisfied from the cache and no memory request is sent over the bus to the main memory
    - Cache **misses** have to go to memory, with a substantial time penalty.
    - Cache memory is limited in size due to its high cost.
    - Some machines have two or even three levels of cache, each one slower and bigger than the one before it.

- **Main memory** comes next in the hierarchy. Main memory is usually called RAM (Random Access Memory).
    - Currently, memories are hundreds of megabytes to several gigabytes and growing rapidly.
    - All CPU requests that cannot be satisfied out of the cache go to main memory.
    - In addition to the main memory, many computers have a small amount of **nonvolatile random-access memory.** Unlike RAM, nonvolatile memory does not lose its contents when the power is switched off.
        - **ROM (Read Only Memory)** is programmed at the factory and cannot be changed afterward. It is fast and inexpensive.
        - On some computers, the bootstrap loader used to start the computer is contained in ROM.
        - Also, some I/O cards come with ROM for handling low-level device control.
        - **EEPROM (Electrically Erasable PROM)** and **flash memory** are also nonvolatile, but in contrast to ROM can be erased and rewritten.
        - **Flash memory** is also commonly used as the storage medium in portable electronic devices. It serves as film in digital cameras and as the disk in portable music players, to name just two uses. Flash memory is intermediate in speed between RAM and disk. Also, unlike disk memory, if it is erased too many times, it wears out.
    - Another kind of memory is CMOS **(Complementary Metal-Oxide-Semiconductor),** which is volatile. Many computers use CMOS memory to hold the current time and date. The CMOS memory and the clock circuit that increments the time in it are powered by a small battery, so the time is correctly updated, even when the computer is unplugged. The CMOS memory can also hold the configuration parameters, such as which disk to boot from. CMOS is used because it draws so little power that the original factory-installed battery often lasts for several years.
- Next in the hierarchy is **magnetic disk** (hard disk). **Disk storage** is two orders of magnitude cheaper than RAM per bit and often two orders of magnitude larger as well.

The only problem is that the time to randomly access data on it is close to three orders of magnitude slower.

- A disk consists of one or more metal **platters** that rotate at 5400, 7200, 10,800 RPM or more. A mechanical **arm** pivots over the platters from the corner, similar to the pickup arm on an old 33-RPM phonograph for playing vinyl records

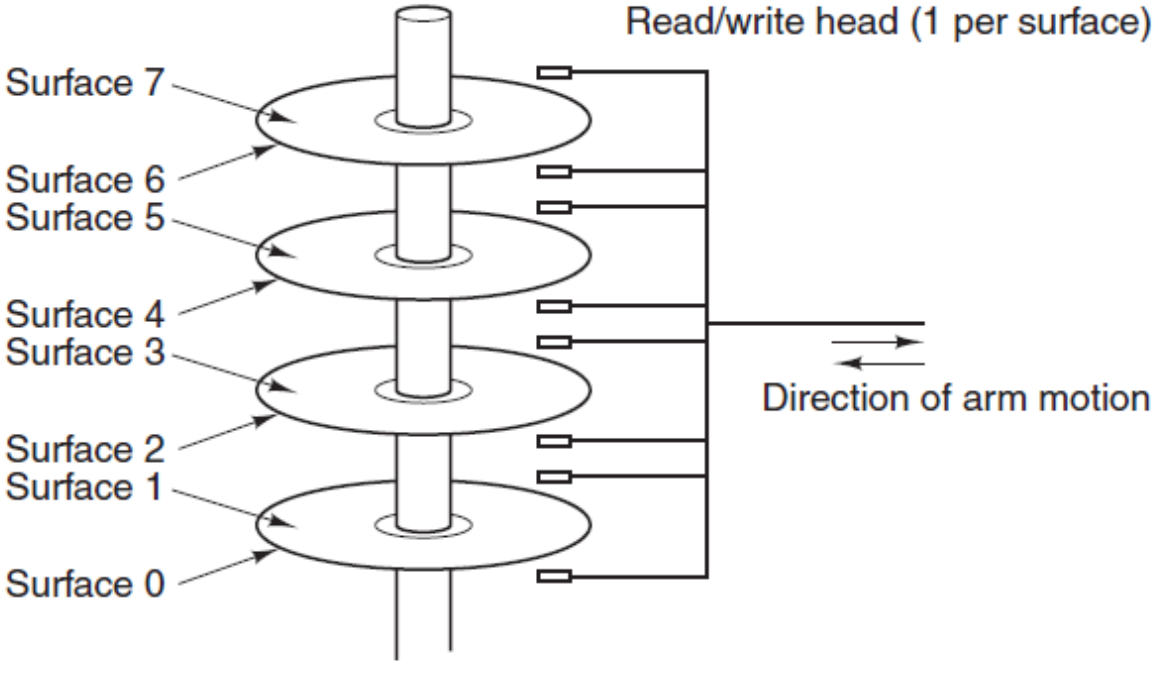

Figure: Structure of a disk drive.

- Information is written onto the disk in a series of concentric circles. At any given arm position, each of the heads can read an annular region called a **track**. Together, all the tracks for a given arm position form a **cylinder**.
- People talk about disks that are really not disks at all, like **SSDs, (Solid State Disks).** SSDs do not have moving parts, do not contain platters in the shape of disks, and store data in (Flash) memory. The only ways in which they resemble disks is that they also store a lot of data which is not lost when the power is off.

I/O devices

- The CPU and memory are not the only resources that the operating system must manage. I/O devices also interact heavily with the operating system

- I/O devices generally consist of two parts: a **controller** and the **device itself**. The controller is a chip or a set of chips that physically controls the device. It accepts commands from the operating system, for example, to read data from the device, and carries them out.
    - In many cases, the actual control of the device is complicated and detailed, so it is the job of the controller to present a simpler (but still very complex) interface to the operating system.
    - The other piece is the actual device itself. Devices have fairly simple interfaces, both because they cannot do much and to make them standard. (Example: SATA disk controller can handle any SAT A disk)
        - SATA is currently the standard type of disk on many computers. Since the actual device interface is hidden behind the controller, all that the operating system sees is the interface to the controller, which may be quite different from the interface to the device.
    - Because each type of controller is different, different software is needed to control each one. The software that talks to a controller, giving it commands and accepting responses, is called a device driver. Each controller manufacturer has to supply a driver for each operating system it supports. Thus a scanner may come with drivers for Windows 7, Windows 8, Windows 10, and Linux, for example

Buses

- In the context of computer architecture, a bus refers to a communication pathway or channel that allows different components of a computer system to transmit data between them. It serves as a shared medium for transferring information, control signals, and addresses among various hardware components.
- The operating system must be aware of all the buses used in the computer system for configuration and management.
- In the beginning there was one bus-couldn’t handle the traffic when cpu and memories got faster and bigger
- Some examples of buses are:
    - **PCIe (Peripheral Component Interconnect Express)** is a high-speed serial bus standard used for connecting peripheral devices to a computer's motherboard. It is the successor to the older **PCI (Peripheral Component**

Interconnect) bus and offers significant improvements in terms of bandwidth, speed, and scalability.

- **DMI (Direct Media Interface)** serves as the primary communication pathway between the CPU and the chipset. It enables high-speed data transfer and communication between the CPU and various peripherals and components connected to the chipset, such as memory, graphics cards, storage devices, and other input/output devices.
- **USB (Universal Serial Bus**) provides a simple and versatile interface for data transfer, power supply, and device connectivity. It provides a simple and versatile interface for data transfer, power supply, and device connectivity.
- **SCSI (Small Computer System Interface)** is a bus technology that was widely used in earlier computer systems to connect peripheral devices such as hard disk drives, tape drives, scanners, and optical drives

History of operating system

Operating systems have been evolving through the years. Since operating systems have historically been closely tied to the architecture of the computers on which they run, we will look at successive generations of computers to see what their operating systems were like.

The First Generation (1945–55): Vacuum Tubes

- The earliest electronic digital computers had no operating systems.
- Machines of the time were so primitive that programs were often entered one bit at time on rows of mechanical switches (plug boards).
- Programming languages were unknown (not even assembly languages).
- Operating systems were unheard of.

The Second Generation (1955–65): Transistors and Batch Systems

- By the early 1950's, the routine had improved somewhat with the introduction of punch cards.
- The General Motors Research Laboratories implemented the first operating systems in early 1950's for their IBM 701.

- The system of the 50's generally ran one job at a time. These were called single-stream batch processing systems because programs and data were submitted in groups or batches.

The Third Generation (1965–1980): ICs and Multiprogramming

- The systems of the 1960's were also batch processing systems, but they were able to take better advantage of the computer's resources by running several jobs at once.
- So operating systems designers developed the concept of **multiprogramming** in which several jobs are in main memory at once; a processor is switched from job to job as needed to keep several jobs advancing while keeping the peripheral devices in use.
    - For example, on the system with no multiprogramming, when the current job paused to wait for other I/O operation to complete, the CPU simply sat idle until the I/O finished.
    - The solution for this problem that evolved was to partition memory into several pieces, with a different job in each partition. While one job was waiting for I/O to complete, another job could be using the CPU.
- Another major feature in third-generation operating system was the technique called **spooling** (simultaneous peripheral operations on line).
    - In spooling, a high-speed device like a disk interposed between a running program and a low-speed device involved with the program in input/output.
    - Instead of writing directly to a printer, for example, outputs are written to the disk.
    - Programs can run to completion faster, and other programs can be initiated sooner when the printer becomes available, the outputs may be printed.
- Another feature present in this generation was **time-sharing** technique, a variant of multiprogramming technique, in which each user has an on-line (i.e., directly connected) terminal. Because the user is present and interacting with the computer, the computer system must respond quickly to user requests, otherwise user productivity could suffer. Timesharing systems were developed to multiprogram large number of simultaneous interactive users.
- The first general-purpose timesharing system, CTSS (Compatible Time Sharing System), was developed at M.I.T.

- MULTICS (MULTiplexed Information and Computing Service)

The Fourth Generation (1980–Present): Personal Computers

- With the development of LSI (Large Scale Integration) circuits, chips, operating system entered in the system entered in the personal computer and the workstation age.
- Microprocessor technology evolved to the point that it becomes possible to build desktop computers as powerful as the mainframes of the 1970s.
- Two operating systems have dominated the personal computer scene: MS-DOS, written by Microsoft, Inc. for the IBM PC and other machines using the Intel 8088 CPU and its successors, and UNIX, which is dominant on the large personal computers using the Motorola 6899 CPU family.
- MAC OSX , Windows 95, Windows NT, Windows 98, Windows 200, Windows Me Windows XP, Windows 7, Windows 8, Windows 10, Windows11

The Fifth Generation (1990–Present): Mobile Computers

— The first real mobile phone appeared in 1946 and weighed some 40 kilos. You could take it wherever you went as long as you had a car in which to carry it.

— The first true handheld phone appeared in the 1970s and, at roughly one kilogram, was positively featherweight. It was affectionately known as ‘‘the brick.’’

— Pretty soon everybody wanted one. Today, mobile phone penetration is close to 90% of the global population. We can make calls not just with our portable phones and wrist watches, but soon with eyeglasses and other wearable items. Moreover, the phone part is no longer that interesting. We receive email, surf the Web, text our friends, play games, and navigate around heavy traffic— and do not even think twice about it.

— While the idea of combining telephony and computing in a phone-like device has been around since the 1970s also, the first real smartphone did not appear until the mid-1990s when Nokia released the N9000, which literally combined two, mostly separate devices: a phone and a PDA (Personal Digital Assistant). In 1997, Ericsson coined the term smartphone for its GS88 ‘‘Penelope.’’

— Mobile phones, Smartphones

— Symbian OS, Blackberry OS, Android OS, iOS

Evolution of operating system

Operating systems have been evolving through the years. Operating systems have been around now for over half a century. During this time, quite a variety of them have been developed, not all of them widely known. Some types of operating systems are discussed as follows:

Batch system

- A batch system is a type of operating system that handles and processes jobs in batches or groups, without requiring user interaction during their execution.
- In a batch system, users submit their jobs to the system, and the operating system executes them one after another in a batch mode.
- Users do not need to interact with the system during the execution of their jobs.
- Batch systems were prevalent in the early days of computing when computer resources were expensive and needed to be shared among multiple users.
- The primary goal of a batch system is to maximize the utilization of resources by executing jobs back-to-back, minimizing idle time.
- Example: UNIVAC 1100/2200, IBM OS/360, etc

Multiprogramming System:

- A multiprogramming system is an operating system that allows multiple programs to run concurrently on a computer.
- Unlike a batch system where jobs are executed sequentially, a multiprogramming system allows several programs to be loaded into memory simultaneously. The operating system switches between these programs, providing each program with a fair share of CPU time. This switching is done rapidly, giving an illusion of parallel execution.
- Multiprogramming systems aim to maximize CPU utilization and improve system responsiveness by overlapping I/O operations and CPU computations.
- Multiprogramming Operating Systems can be simply illustrated as more than one program is present in the main memory and any one of them can be kept in execution. This is basically used for better execution of resources.
- Example: UNIX, Linux, Windows NT

Time-Sharing System

- A time-sharing system, is an operating system that allows multiple users to interact with the computer simultaneously.

- In a time-sharing system, the CPU time is divided into small time intervals called time slices or time quanta.
- Each user or process is allocated a time slice during which they can execute their tasks. The operating system rapidly switches between different users or processes, providing the illusion of simultaneous execution.
- Time-sharing systems aim to provide interactive computing by giving each user or process a fair share of CPU time and allowing them to perform tasks concurrently.
- Time-sharing is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.
- In such system multiple users can interact with the system simultaneously. The OS rapidly switches between users. Users can share the system's resources.
- Time-sharing OSes improve system efficiency, user experience, and security.
- Time-sharing OSes are used in a variety of applications like web servers, mail servers, and database servers.
- Example: UNIX, Linux, Windows NT, Mac OS X, etc.

Mainframe Operating Systems

- Operating systems for mainframe computers (big size, thousands of disks)
- The operating systems for mainframes are heavily oriented toward processing many jobs at once, most of which need lots of I/O
- They typically offer three kinds of services: **batch**, **transaction processing**, and **timesharing**.
- A **batch system** is one that processes routine jobs without any interactive user present.
    - Claims processing in an insurance company or sales reporting for a chain of stores is typically done in batch mode.
- **Transaction-processing systems** handle large numbers of small requests, for example, check processing at a bank or airline reservations.
    - Each unit of work is small, but the system must handle hundreds or thousands per second.
- **Timesharing systems** allow multiple remote users to run jobs on the computer at once, such as querying a big database.

— These functions are closely related; mainframe operating systems often perform all of them.

— An example mainframe operating system is OS/390, a descendant of OS/360. However, mainframe operating systems are gradually being replaced by UNIX variants such as Linux.

Server Operating Systems

— They run on servers, which are either very large personal computers, workstations, or even mainframes.

— They serve multiple users at once over a network and allow the users to share hardware and software resources. Servers can provide print service, file service, or Web service.

— Typical server operating systems are Solaris, FreeBSD, and Linux and Windows Server 201 x.

Multiprocessor Operating Systems

— An increasingly common way to get major-league computing power is to connect multiple CPUs into a single system. Depending on precisely how they are connected and what is shared, these systems are called **parallel computers, multicomputers, or multiprocessors.**

— They need special operating systems, but often these are variations on the server operating systems, with special features for communication, connectivity, and consistency.

— With the recent advent of multicore chips for personal computers, even conventional desktop and notebook operating systems are starting to deal with at least small-scale multiprocessors and the number of cores is likely to grow over time.

— Many popular operating systems, including Windows and Linux, run on multiprocessors.

Personal Computer Operating Systems

— All modern PC operating systems support multiprogramming, often with dozens of programs started up at boot time.

— Their job is to provide good support to a single user.

— They are widely used for word processing, spreadsheets, games, and Internet access.

— Common examples are Linux, FreeBSD, Windows 8, Windows 10, and Apple's OS X.

Handheld Computer Operating Systems

- Continuing on down to smaller and smaller systems, we come to tablets, smartphones and other handheld computers. A handheld computer, originally known as a PDA (Personal Digital Assistant), is a small computer that can be held in your hand during operation.
- Smartphones and tablets are the best-known examples.
- Market is currently dominated by Google's Android and Apple's iOS, but they hav e many competitors.
- Most of these devices boast multicore CPUs, GPS, cameras and other sensors, copious amounts of memory, and sophisticated operating systems.

Embedded Operating Systems

- Embedded systems run on the computers that control devices that are not generally thought of as computers and which do not accept user-installed software
- Typical examples are microwave ovens, TV sets, cars, DVD recorders, traditional phones, and MP3 players.
- The main property which distinguishes embedded systems from handhelds is the certainty that no untrusted software will ever run on it.
- You cannot download new applications to your microwave oven—all the software is in ROM. This means that there is no need for protection between applications, leading to design simplification.
- Systems such as Embedded Linux, QNX and VxWorks are popular in this domain.

Sensor-Node Operating Systems

- Networks of tiny sensor nodes are being deployed for numerous purposes. These nodes are tiny computers that communicate with each other and with a base station using wireless communication.
- Sensor networks are used to protect the perimeters of buildings, guard national borders, detect fires in forests, measure temperature and precipitation for weather forecasting, glean information about enemy movements on battlefields, and much more.
- Each **sensor node is a real computer**, with a CPU, RAM, ROM, and one or more environmental sensors. It runs a small, but real operating system, usually one that is event driven, responding to external events or making measurements periodically based on an internal clock.

— *The operating system has to be small and simple because the nodes have little RAM and battery lifetime is a major issue.*

— Also, as with embedded systems, all the programs are loaded in advance; users do not suddenly start programs they downloaded from the Internet, which makes the design much simpler.

— **TinyOS** is a well-known operating system for a sensor node.

Real-Time Operating Systems

— These systems are characterized by having time as a key parameter. For example, in industrial process- control systems, real-time computers have to collect data about the production process and use it to control machines in the factory.

— Often there are **hard deadlines** that must be met. For example, if a car is moving down an assembly line, certain actions must take place at certain instants of time.

- If, for example, a welding robot welds too early or too late, the car will be ruined.
- If the action absolutely must occur at a certain moment (or within a certain range), we have a **hard real-time system**.
- These systems must provide absolute guarantees that a certain action will occur by a certain time.

— A **soft real-time system**, is one where missing an occasional deadline, while not desirable, is acceptable and does not cause any permanent damage.

— Digital audio or multimedia systems fall in this category. Smartphones are also soft real-time systems.

— Examples of real time OSes: eCos, VxWorks, QNX Neutrino, FreeRTOS,etc.

NOTE: The embedded and real-time systems run only software put in by the system designers; users cannot add their own software, which makes protection easier. The handhelds and embedded systems are intended for consumers, whereas real-time systems are more for industrial usage.

Smart Card Operating Systems

— The smallest operating systems run on smart cards, which are credit-card-sized devices containing a CPU chip.

— They have very severe processing power and memory constraints.

- Some are powered by contacts in the reader into which they are inserted, but contactless smart cards are inductively powered, which greatly limits what they can do.
- Some of them can handle only a single function, such as electronic payments, but others can handle multiple functions.
- Often these are proprietary systems.
- MULTOS is an example.

Distributed Systems

- A distributed system is a collection of physically separate, possibly heterogeneous computer systems that are networked to provide users with access to the various resources that the system maintains.
- Access to a shared resource increases computation speed, functionality, data availability, and reliability.
- Some operating systems generalize network access as a form of file access, with the details of networking contained in the network interface's device driver .Others make users specifically invoke network functions. Generally, systems contain a mix of the two modes—for example FTP (File Transfer Protocol) and NFS (Network File System). The protocols that create a distributed system can greatly affect that system's utility and popularity.
- In distributed system, the different computer communicate closely enough to provide the illusion that only a single operating system controls the network
- A distributed system refers to a network of interconnected computers that work together to achieve a common goal. In a distributed system, multiple computers, often referred to as nodes or hosts, are connected through a communication network and collaborate to perform tasks or provide services. These systems are designed to improve performance, reliability, scalability, and fault tolerance by distributing the workload across multiple machines.
- Operating systems play a crucial role in managing and coordinating distributed systems.
- Operating systems in distributed systems are responsible for managing resources, facilitating communication, ensuring synchronization, handling fault tolerance, and providing security. They act as the backbone of the distributed system, orchestrating the activities of multiple nodes and enabling them to work together seamlessly toward achieving common objectives.

— In a distributed operating system, users access remote resources in the same way they access local resources. Data and process migration from one site to another is under the control of the distributed operating system. Depending on the goals of the system, it can implement data migration, computation migration, process migration, or any combination thereof.
— Examples of distributed OS: Plan 9, Distributed UNIX, etc.

Operating System Concepts

— Most operating systems provide certain basic concepts and abstractions such as processes, address spaces, and files that are central to understanding them.

Booting Computer

— Booting a computer refers to the process of starting it up and loading the operating system into memory so that the computer becomes functional and ready for use.
— Very briefly, the boot process is as follows.
  - Every PC contains a parentboard (formerly called a motherboard before political correctness hit the computer industry). On the parentboard is a program called the system **BIOS (Basic Input Output System).** The BIOS contains low-level I/O software, including procedures to read the keyboard, write to the screen, and do disk I/O, among other things. Nowadays, it is held in a flash RAM, which is nonvolatile but which can be updated by the operating system when bugs are found in the BIOS.
  - When the computer is booted, the BIOS is started. It first checks to see how much RAM is installed and whether the keyboard and other basic devices are installed and responding correctly. It starts out by scanning the PCIe and PCI buses to detect all the devices attached to them. If the devices present are different from when the system was last booted, the new devices are configured.
  - The BIOS then determines the boot device by trying a list of devices stored in the CMOS memory. The user can change this list by entering a BIOS configuration program just after booting. Typically, an attempt is made to boot from a CD-ROM (or sometimes USB) drive, if one is present. If that fails, the system boots from the hard disk.

- The first sector from the boot device is read into memory and executed. This sector contains a program that normally examines the partition table at the end of the boot sector to determine which partition is active.
- Then a secondary boot loader is read in from that partition. This loader reads in the operating system from the active partition and starts it.
- The operating system then queries the BIOS to get the configuration information. For each device, it checks to see if it has the device driver. If not, it asks the user to insert a CD-ROM containing the driver (supplied by the device's manufacturer) or to download it from the Internet. Once it has all the device drivers, the operating system loads them into the kernel. Then it initializes its tables, creates whatever background processes are needed, and starts up a login program or GUI

Steps in Computer Booting

Powering on the computer: When the power button is pressed, the computer's power supply provides electricity to the various components of the computer, initiating the booting process.

Power-on self-test (POST): The computer's hardware components, such as the CPU, memory, and storage devices, are tested to ensure they are functioning properly. The system firmware (BIOS – [Basic Input Output System**]** or UEFI [Unified Extensible Firmware Interface]) performs these tests and checks for any errors or issues.

Boot loader initialization: The boot loader, which is typically stored in the computer's storage device (e.g., hard drive or SSD), is initialized. The boot loader is responsible for loading the operating system.

Operating system loading: The boot loader locates the operating system's kernel (the core component of the OS) and loads it into memory. The kernel initializes essential system components and prepares the operating system for user interaction.

Operating system initialization: Once the kernel is loaded, the operating system initializes various services, drivers, and system processes. This includes setting up the graphical user interface (GUI), network connections, device drivers, and other necessary components.

User login: After the initialization process, the computer presents the user with a login screen or desktop environment. The user can then enter their credentials to log in and begin using the computer.

NOTE: The booting process can vary depending on the computer's hardware architecture, firmware (BIOS or UEFI), and the specific operating system being used. Additionally, certain devices or configurations may introduce additional steps or variations in the booting process.

Address Spaces

- An address space refers to the range of memory addresses that a process or program can access.
- Every computer has some main memory that it uses to hold executing programs. In a very simple operating system, only one program at a time is in memory. To run a second program, the first one has to be removed and the second one placed in memory.
- More sophisticated operating systems allow multiple programs to be in memory at the same time. To keep them from interfering with one another (and with the operating system), some kind of protection mechanism is needed. While this mechanism has to be in the hardware, it is controlled by the operating system
- Normally, each process has some set of addresses it can use, typically running from 0 up to some maximum. In the simplest case, the maximum amount of address space a process has is less than the main memory. In this way, a process can fill up its address space and there will be enough room in main memory to hold it all. However, on many computers addresses are 32 or 64 bits, giving an address space of 2^{32} or 2^{64} bytes, respectively. What happens if a process has more address space than the computer has main memory and the process wants to use it all? In the first computers, such a process was just out of luck. Nowadays, a technique called **virtual memory exists**, in which the operating system keeps part of the address space in main memory and part on disk and shuttles pieces back and forth between them as needed.
- Each process running on a computer system has its own address space, ensuring that one process cannot interfere with the memory of another process. The operating system manages and allocates the address space for each process.
- Address spaces provide several benefits, including memory isolation, memory protection, and efficient memory utilization. They enable processes to run independently, without interfering with each other's memory, and provide security by preventing unauthorized access to memory locations. Address spaces also allow for more efficient memory allocation by utilizing virtual memory techniques such as demand paging and memory swapping.
- In essence, the operating system creates the abstraction of an address space as the set of addresses a process may reference. The address space is decoupled from the machine’s physical memory and may be either larger or smaller than the physical memory.

— Address spaces play a crucial role in the organization and management of memory in a computer system, enabling processes to effectively utilize and access memory resources while maintaining isolation and protection

Files

— Another key concept supported by virtually all operating systems is the file system. As noted before, a major function of the operating system is to hide the peculiarities of the disks and other I/O devices and present the programmer with a nice, clean abstract model of device-independent files

— System calls are needed to create files, remove files, read files, and write files

— Before a file can be read, it must be located on the disk and opened, and after being read it should be closed, so calls are provided to do these things

— To provide a place to keep files, most PC operating systems have the concept of a **directory** as a way of grouping files together. A student, for example, might have one directory for each course he is taking (for the programs needed for that course), another directory for his electronic mail, and still another directory for his World Wide Web home page. System calls are then needed to create and remove directories. Calls are also provided to put an existing file in a directory and to remove a file from a directory. Directory entries may be either files or other directories.

— This model also gives rise to a hierarchy—the file system—as shown in following figure.

— File hierarchy is organized as tree.

— Every file within the directory hierarchy can be specified by giving its path name from the top of the directory hierarchy, the **root directory.** Such absolute path names consist of the list of directories that must be traversed from the root directory to get to the file, with slashes separating the components

— At every instant, each process has a current working directory

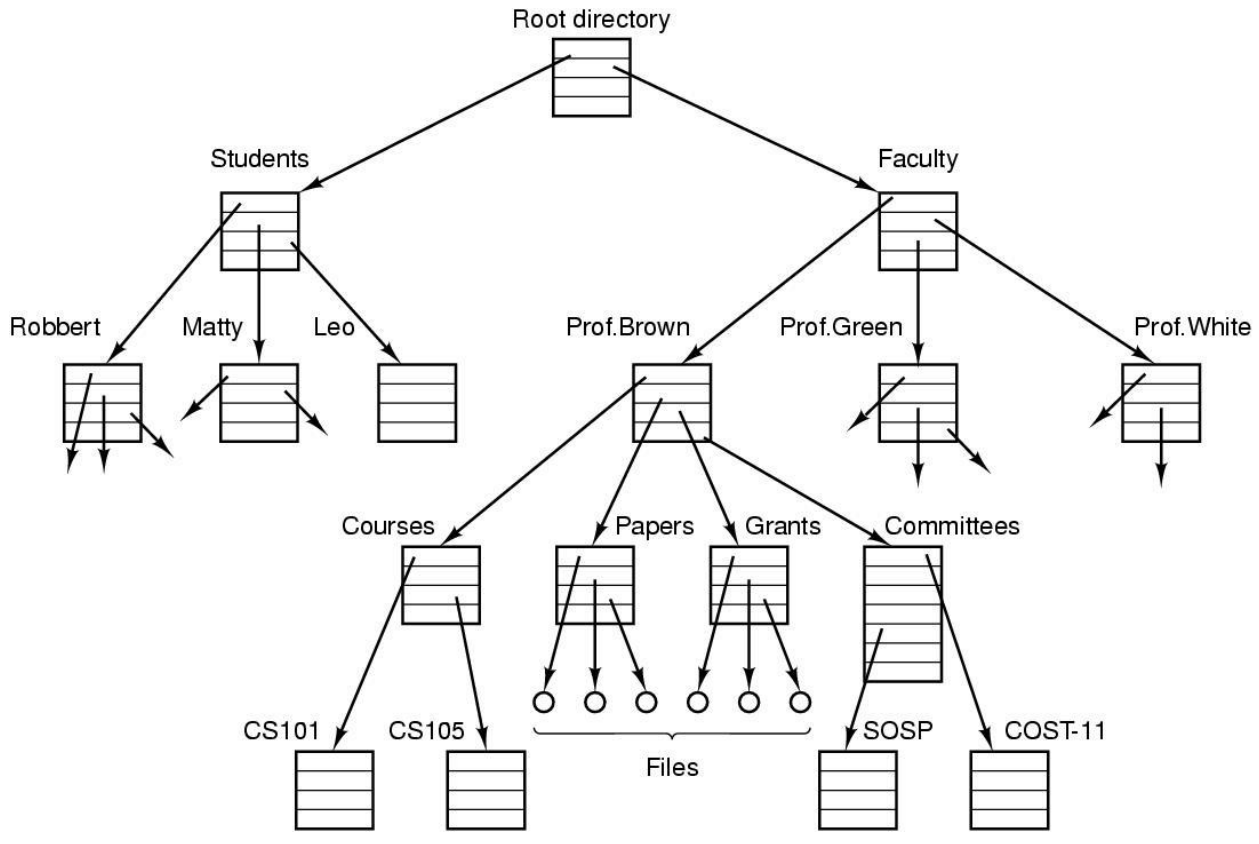

Figure: A file system for a university department

Protection

- Computers contain large amounts of information that users often want to protect and keep confidential. This information may include email, business plans, tax returns, and much more. It is up to the operating system to manage the system security so that files, for example, are accessible only to authorized users.
- Files in UNIX(UNiplexed Information Computing System) are protected by assigning each one a **9-bit** binary protection code.
    - The protection code consists of three 3-bit fields, one for the owner, one for other members of the owner's group (users are divided into groups by the system administrator), and one for everyone else.
    - Each field has a bit for read access, a bit for write access, and a bit for execute access. These 3 bits are known as the **rwx** bits.
    - For example, the protection code **rwxr-x--x** means that the owner can read, write, or execute the file, other group members can read or execute (but not write) the file, and everyone else can execute (but not read or write) the file. For a directory,

x indicates search permission. A dash means that the corresponding permission is absent.

— In addition to file protection, there are many other security issues. Protecting the system from unwanted intruders, both human and nonhuman (e.g., viruses) is one of them.

Operating System Structures

— There are different structures of operating system. The six designs we will discuss here are **monolithic systems**, **layered systems, microkernels, client-server systems, virtual machines,** and **exokernels**.

Monolithic Systems

— By far the most common organization, in the monolithic approach the entire operating system runs as a single program in kernel mode.

— The operating system is written as a collection of procedures, linked together into a single large executable binary program.

— The OS in it is written as a collection of procedures, each of which can call any of the other ones whenever it needs to.

— In this technique each procedure is free to call any other one, if the latter provides some useful computation than the former needs.

— To construct the actual object of the operating system when this approach is used, one first compiles all the individuals' procedures, or files containing the procedures and then binds them all together into a single object file using the system linker.

— In terms of information hiding, there is essentially none – every procedure is visible to every other procedure.

— Even in monolithic systems, however, it is possible to have some structure.

- The services (system calls) provided by the operating system are requested by putting the parameters in a well-defined place (e.g., on the stack) and then executing a trap instruction.
- This instruction switches the machine from user mode to kernel mode and transfers control to the operating system

- The operating system then fetches the parameters and determines which system call is to be carried out. After that, it indexes into a table that contains in slot k a pointer to the procedure that carries out system call k

This organization suggests a basic structure for the operating system:

1. A main program that invokes the requested service procedure.
2. A set of service procedures that carry out the system calls.
3. A set of utility procedures that help the service procedures.

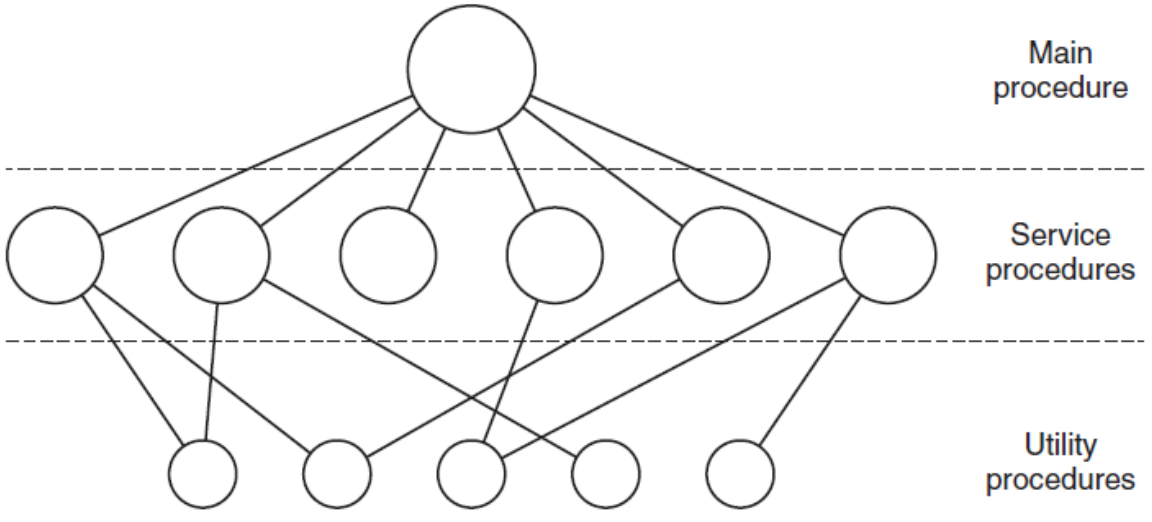

Figure : A simple structuring model for a monolithic system

Layered Systems

— To generalize the monolithic system into the hierarchy of layers, the first system called THE was built by E.W. Dijkstra in 1986.

— This system had 6 layers.

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Figure: Structure of THE operating system

— An OS can be broken into pieces and retain much more control on system. In this structure the OS is broken into number of layers (levels)

Microkernels

- With the layered approach, the designers have a choice where to draw the kernel- user boundary. Traditionally, all the layers went in the kernel, but that is not necessary.
    - In fact, a strong case can be made for putting as little as possible in kernel mode because bugs in the kernel can bring down the system instantly.
    - In contrast, user processes can be set up to have less power so that a bug there may not be fatal.
- The basic idea behind the microkernel design is to achieve high reliability by splitting the operating system up into small, well-defined modules, only one of which—the microkernel—runs in kernel mode and the rest run as relatively powerless ordinary user processes.
    - In particular, by running each device driver and file system as a separate user process, a bug in one of these can crash that component, but cannot crash the entire system.
    - Thus a bug in the audio driver will cause the sound to be garbled or stop, but will not crash the computer.

- In contrast, in a monolithic system with all the drivers in the kernel, a buggy audio driver can easily reference an invalid memory address and bring the system to a grinding halt instantly

Example: MINIX system

- The process structure of MINIX 3 is shown in the figure, with the kernel call handlers labeled Sys. The device driver for the clock is also in the kernel because the scheduler interacts closely with it. The other device drivers run as separate user processes.
- Outside the kernel, the system is structured as three layers of processes all running in user mode.
  - The lowest layer contains the **device drivers**. Since they run in user mode, they do not have physical access to the I/O port space and cannot issue I/O commands directly. Instead, to program an I/O device, the driver builds a structure telling which values to write to which I/O ports and makes a kernel call telling the kernel to do the write. This approach means that the kernel can check to see that the driver is writing (or reading) from I/O it is authorized to use. Consequently (and unlike a monolithic design), a buggy audio driver cannot accidentally write on the disk.
  - Above the drivers is another user-mode layer containing the **servers**, which do most of the work of the operating system. One or more file servers manage the file system(s), the process manager creates, destroys, and manages processes, and so on.
  - **User programs** obtain operating system services by sending short messages to the servers asking for the POSIX system calls..

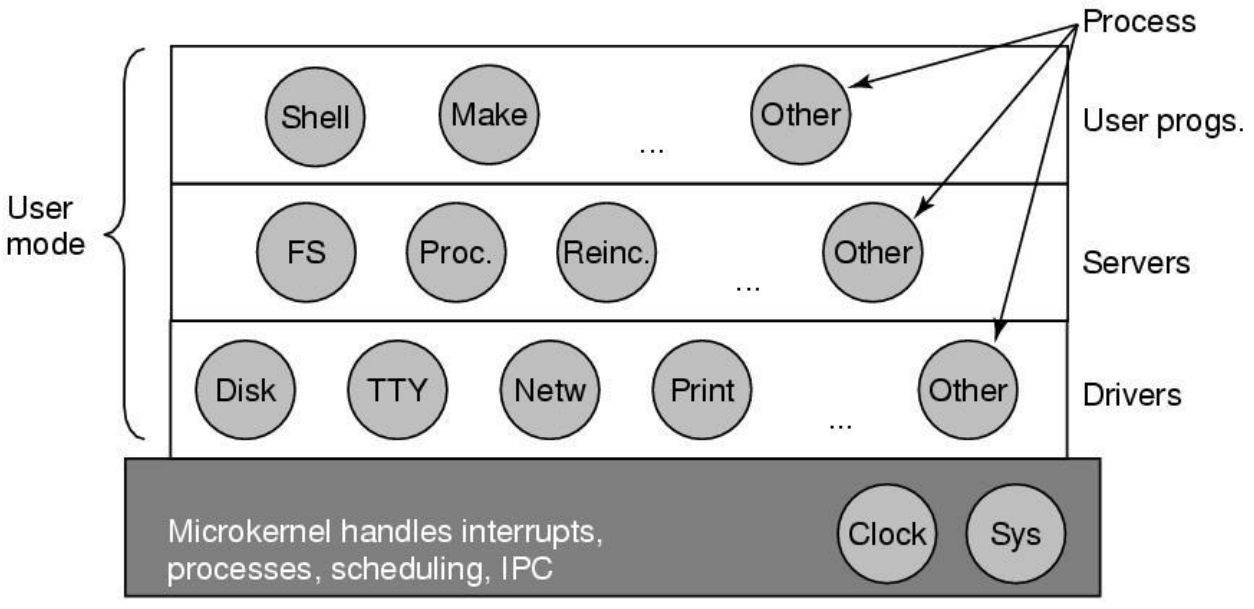

Figure: Simplified structure of the MINIX system

- An idea somewhat related to having a **minimal kernel** is to put the **mechanism** for doing something in the kernel but not the **policy**.
  - To make this point better, consider the scheduling of processes. A relatively simple scheduling algorithm is to assign a numerical priority to every process and then have the kernel run the highest-priority process that is runnable.
  - The **mechanism**—in the kernel—is to look for the highest-priority process and run it.
  - The **policy**—assigning priorities to processes—can be done by user-mode processes.
  - In this way, policy and mechanism can be decoupled and the kernel can be made smaller

Microkernels

- Small number of processes are allowed in the kernel
- Minimizes effects of bugs
  - Don't want bug in driver to crash system
- Put mechanism in kernel and policy outside the kernel
  - Mechanism- schedule processes by priority scheduling algorithm
  - Policy-assign process priorities in user space
- Policy and mechanism can be decoupled and the kernel can be made smaller

Client-Server Model

- A slight variation of the microkernel idea is to distinguish two classes of processes, the **servers,** each of *which provides some service*, and the **clients**, *which use these services*. This model is known as the **client-server model**.
- Since clients communicate with servers by sending messages, the clients need not know whether the messages are handled locally on their own machines, or whether they are sent across a network to servers on a remote machine.
  - As far as the client is concerned, the same thing happens in both cases: requests are sent and replies come back.
  - Thus the client-server model is an abstraction that can be used for a single machine or for a network of machines.
- .Increasingly many systems involve users at their home PCs as clients and large machines elsewhere running as servers. In fact, much of the Web operates this way. A PC sends a request for a Web page to the server and the Web page comes back. This is a typical use of the client server model in a network.

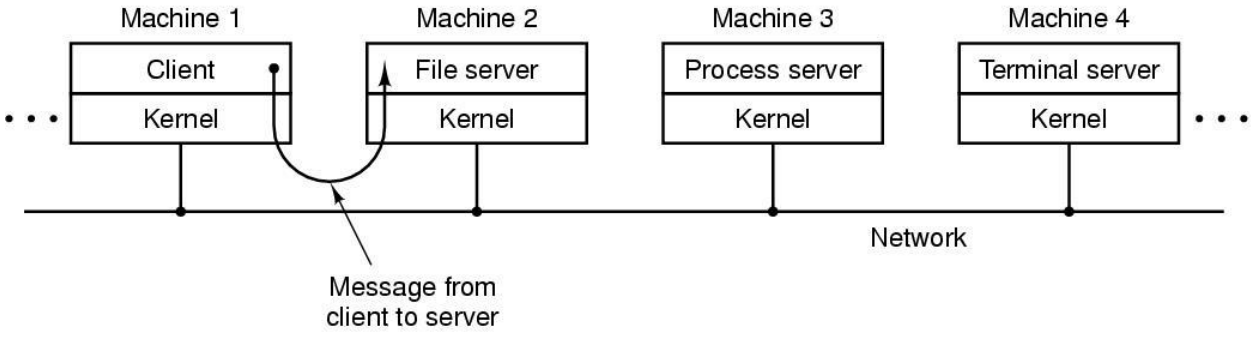

Figure: The client-server model over a network.

Virtual Machines

- The concept of a virtual machine is to provide an interface that looks like independent hardware, to multiple different OSes running simultaneously on the same physical hardware. Each OS believes that it has access to and control over its own CPU, RAM, I/O devices, hard drives, etc.
- One obvious use for this system is for the development and testing of software that must run on multiple platforms and/or OSes.
- One obvious difficulty involves the sharing of hard drives, which are generally partitioned into separate smaller virtual disks for each operating OS.

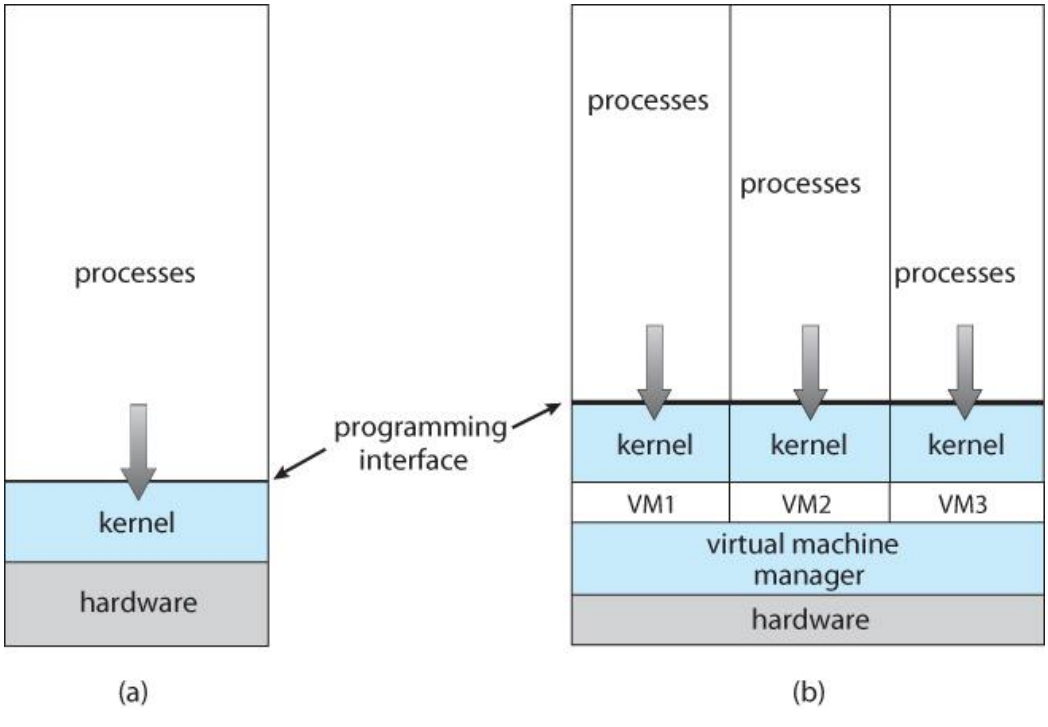

Figure: System models. (a) Nonvirtual machine. (b)Virtual machine

- Examples of Virtual Machines: VMWare, JVM, CLR ,etc.

VM/370 (First Virtual Machine)

- Initial release of OS/360 were strictly batch systems
- Many 360 users wanted to have time sharing and IBM developed a machine originally called CP/CMS and later named VM/370
- The heart of the system, known as the virtual machine monitor, runs on the bare hardware and does the multiprogramming, providing not only one, but several virtual machines to the next layer up.
- The virtual machines are not the extended machines but are the exact copies of the bare hardware including kernel/user mode, I/O, interrupts and everything else the real machine has.
- The CMS (Conversational Monitor System) is for interactive sharing users.
- Because each virtual machine is identical to the true hardware, each one can run any operating system that will run directly on the bare hardware. Different virtual machines can, and frequently do, run different operating systems.

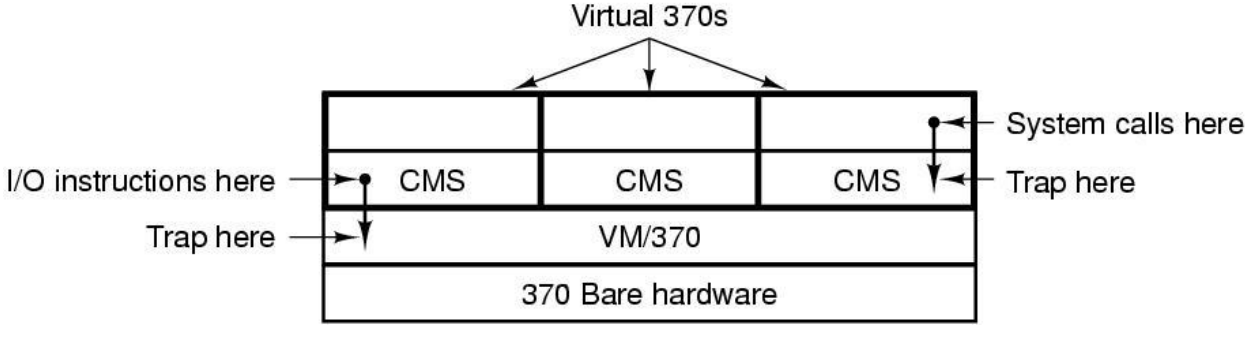

Figure: The structure of VM/370 with CMS.

NOTES:

- Virtual machines are made possible through virtualization technology. Virtualization uses software to simulate virtual hardware that allows multiple VMs to run on a single machine. The physical machine is known as the host while the VMs running on it are called guests.
- This process is managed by software known as a hypervisor. The hypervisor is responsible for managing and provisioning resources—like memory and storage—from the host to guests. It also schedules operations in VMs so they don't overrun each other when using resources. VMs only work if there is a hypervisor to virtualize and distribute host resources

Assignment: Discuss about type1 and type2 hypervisors

Exokernels

- Rather than cloning the actual machine, as is done with virtual machines, another strategy is partitioning it, in other words, giving each user a subset of the resources. Thus one virtual machine might get disk blocks 0 to 1023, the next one might get blocks 1024 to 2047, and so on. At the bottom layer, running in kernel mode, is a program called the **exokernel.** Its job is to allocate resources to virtual machines and then check attempts to use them to make sure no machine is trying to use somebody else's resources.
- The advantage of the exokernel scheme is that it saves a layer of mapping. In the other designs, each virtual machine thinks it has its own disk, with blocks running from 0 to some maximum, so the virtual machine monitor must maintain tables to remap disk addresses (and all other resources). With the exokernel, this remapping is not needed. The exokernel need only keep track of which virtual machine has been assigned which resource. This method still has the advantage of separating the multiprogramming (in the exokernel) from the user operating system code (in user space), but with less overhead, since all the exokernel has to do is keep the virtual machines out of each other's hair.

Research task: The structures of OSes (latest stable versions) like iOS, Android, Ubuntu, Windows 10 /11 , Mac OS

Operating system Components

As we have seen, an 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 operating system must manage. Different components of OS are responsible for managing such resources.

Process Management

- A program in execution, as mentioned, is a process.
- A process needs certain resources—including CPU time, memory, files, and I/O devices—to accomplish its task.
- These resources are typically allocated to the process while it is running. In addition to the various physical and logical resources that a process obtains when it is created, various initialization data (input) may be passed along.

- The operating system is responsible for the following activities in connection with process management:
    - Creating and deleting both user and system processes
    - Scheduling processes and threads on the CPUs
    - Suspending and resuming processes
    - Providing mechanisms for process synchronization
    - Providing mechanisms for process communication

I/O managements

- One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user. For example, in UNIX, the peculiarities of I/O devices are hidden from the bulk of the operating system itself by the I/O subsystem. The I/O subsystem consists of several components:
    - A memory-management component that includes buffering, caching, andspooling
    - A general device-driver interface
    - Drivers for specific hardware devices
- Only the device driver knows the peculiarities of the specific device to which it is assigned.

File and Directory management

- To make the computer system convenient for users, the operating system provides a uniform, logical view of information storage. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the file. The operating system maps files onto physical media and accesses these files via the storage devices.
- File management is one of the most visible components of an operating system.
- The operating system implements the abstract concept of a file by managing mass storage media and the devices that control them. In addition, files are normally organized into directories to make them easier to use.
- When multiple users have access to files, it may be desirable to control which user may access a file and how that user may access it (for example, read, write, append).

- The operating system is responsible for the following activities in connectionwith file management:
    - 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

Memory Management

- The main memory is central to the operation of a modern computer system. Main memory is a large array of bytes, ranging in size from hundreds of thousands to billions.
- Each byte has its own address. Main memory 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 main memory during the data-fetch cycle (on a von Neumann architecture)
- For a program to be executed, it must be mapped to absolute addresses and loaded into memory. As the program executes, it accesses program instructions and data from memory by generating these absolute addresses. Eventually, the program terminates, its memory space is declared available, and the next program can be loaded and executed.
- To improve both the utilization of the CPU and the speed of the computer's response to its users, general-purpose computers must keep several programs in memory, creating a need for memory management. Many different memory management schemes are used. These schemes reflect various approaches, and the effectiveness of any given algorithm depends on the situation.
- In selecting a memory-management scheme for a specific system, we must take into account many factors—especially the hardware design of the system. Each algorithm requires its own hardware support.
- The operating system is responsible for the following activities in connection with memory management:
    - Keeping track of which parts of memory are currently being used and which process is using them
    - Allocating and deallocating memory space as needed

- Deciding which processes (or parts of processes) and data to move into and out of memory

Operating system services

- An Operating System supplies different kinds of services to both the users and to the programs as well. It also provides application programs (that run within an Operating system) an environment to execute it freely. It provides users the services run various programs in a convenient manner.
- The user program requests various resources through the operating system. The operating system gives several services to utility programmers and users. Applications access these services through application programming interfaces or system calls. By invoking those interfaces, the application can request a service from the operating system, pass parameters, and acquire the operation outcomes.Following are a few common services provided by an operating system
- Following are a few common services provided by an operating system −

  Program execution

  Operating systems handle many kinds of activities from user programs to system programs like printer spooler, name servers, file server, etc. Each of these activities is encapsulated as a process.

  A process includes the complete execution context (code to execute, data to manipulate, registers, OS resources in use). Following are the major activities of an operating system with respect to program management −

  - Loads a program into memory.
  - Executes the program.
  - Handles program's execution.
  - Provides a mechanism for process synchronization.
  - Provides a mechanism for process communication.
  - Provides a mechanism for deadlock handling.

  I/O Operation

  An I/O subsystem comprises of I/O devices and their corresponding driver software. Drivers hide the peculiarities of specific hardware devices from the users.

An Operating System manages the communication between user and device drivers.

- I/O operation means read or write operation with any file or any specific I/O device.
- Operating system provides the access to the required I/O device when required.

File system manipulation

A file represents a collection of related information. Computers can store files on the disk (secondary storage), for long-term storage purpose.

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions. Following are the major activities of an operating system with respect to file management −

- Program needs to read a file or write a file.
- The operating system gives the permission to the program for operation on file.
- Permission varies from read-only, read-write, denied and so on.
- Operating System provides an interface to the user to create/delete files.
- Operating System provides an interface to the user to create/delete directories.
- Operating System provides an interface to create the backup of file system.

Communication

In case of distributed systems which are a collection of processors that do not share memory, peripheral devices, or a clock, the operating system manages communications between all the processes. Multiple processes communicate with one another through communication lines in the network.

The OS handles routing and connection strategies, and the problems of contention and security. Following are the major activities of an operating system with respect to communication −

- Two processes often require data to be transferred between them
- Both the processes can be on one computer or on different computers, but are connected through a computer network.
- Communication may be implemented by two methods, either by Shared Memory or by Message Passing.

Error handling

Errors can occur anytime and anywhere. An error may occur in CPU, in I/O devices or in the memory hardware. Following are the major activities of an operating system with respect to error handling −

- The OS constantly checks for possible errors.
- The OS takes an appropriate action to ensure correct and consistent computing.

Resource Management

In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job. Following are the major activities of an operating system with respect to resource management −

- The OS manages all kinds of resources using schedulers.
- CPU scheduling algorithms are used for better utilization of CPU.

Protection

Considering a computer system having multiple users and concurrent execution of multiple processes, the various processes must be protected from each other's activities.

Protection refers to a mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system. Following are the major activities of an operating system with respect to protection −

- The OS ensures that all access to system resources is controlled.
- The OS ensures that external I/O devices are protected from invalid access attempts.
- The OS provides authentication features for each user by means of passwords.

System Call

- A system call is a mechanism that provides the interface between a process and the operating system. It is a programmatic method in which a computer program requests a service from the kernel of the OS.
- The interface between a process and an operating system is provided by system call
- System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.

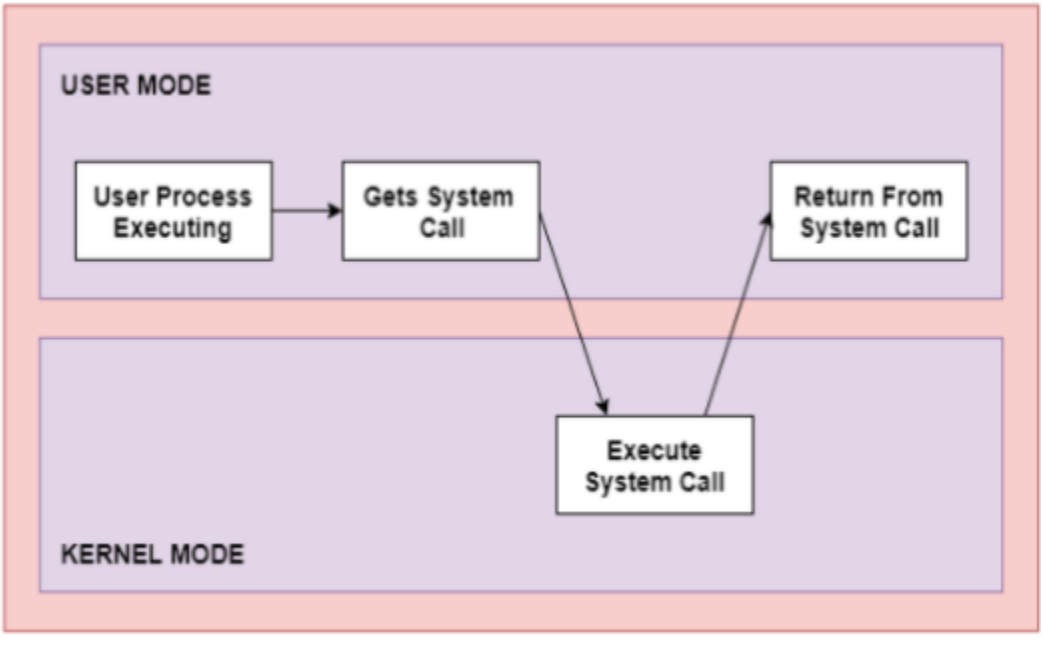

Figure: Execution of system call

As can be seen from above diagram, the processes execute normally in the user mode until a system call interrupts this. Then the system call is executed on a priority basis in the kernel mode. After the execution of the system call, the control returns to the user mode and execution of user processes can be resumed.

In general, system calls are required in the following situations −

- If a file system requires the creation or deletion of files. Reading and writing from files also require a system call.
- Creation and management of new processes.
- Network connections also require system calls. This includes sending and receiving packets.

— Access to a hardware devices such as a printer, scanner etc. requires a system call.

Understanding system call mechanism in Read System Call (in UNIX)

Like nearly all system calls, it is invoked from C programs by calling a library procedure with the same name as the system call: read. A call from a C program might look like this:

```
count = read(fd, buffer, nbytes);
```

It has three parameters: the first one specifying the file, the second one pointing to the buffer, and the third one giving the number of bytes to read.

System calls are performed in a series of steps. The steps in **read** system call are as follows :

— In preparation for calling the read library procedure, which actually makes the read system call, the calling program first pushes the parameters onto the stack, as shown in **steps 1–3** in following figure.

— The first and third parameters are called by value, but the second parameter is passed by reference, meaning that the address of the buffer (indicated by &) is passed, not the contents of the buffer. Then comes the actual call to the library procedure **(step 4).** This instruction is the normal procedure- call instruction used to call all procedures.

— The library procedure, possibly written in assembly language, typically puts the system-call number in a place where the operating system expects it, such as a register **(step 5).**

— Then it executes a TRAP instruction to switch from user mode to kernel mode and start execution at a fixed address within the kernel **(step 6).**

- The TRAP instruction is actually fairly similar to the procedure-call instruction in the sense that the instruction following it is taken from a distant location and the return address is saved on the stack for use later.
- Nevertheless, the TRAP instruction also differs from the procedure-call instruction in two fundamental ways.
  - First, as a side effect, it switches into kernel mode. The procedure call instruction does not change the mode.
  - Second, rather than giving a relative or absolute address where the procedure is located, the TRAP instruction cannot jump to an arbitrary address.

— The kernel code that starts following the TRAP examines the system-call number and then dispatches to the correct system-call handler, usually via a table of pointers to system-call handlers indexed on system-call number **(step 7).**

— At that point the system-call handler runs **(step 8).**

— Once it has completed its work, control may be returned to the user-space library procedure at the instruction following the TRAP instruction **(step 9).**

— This procedure then returns to the user program in the usual way procedure calls return **(step 10).**

— To finish the job, the user program has to clean up the stack, as it does after any procedure call **(step 11).**

- Push parameter into the stack (1-3)
- Calls library procedure (4)
- Pass parameters in registers (5)
- Switch from user mode to kernel mode and start to execute (6)
- Examine the system call number and then dispatch to the correct system call handler via a table of pointer (7)
- Run System call Handlers (8)
- Once the system call handler completed its work, control return to the library procedure (9)
- This procedure then return to the user program in the usual way (10)
- Increment SP before call to finish the job. (11)

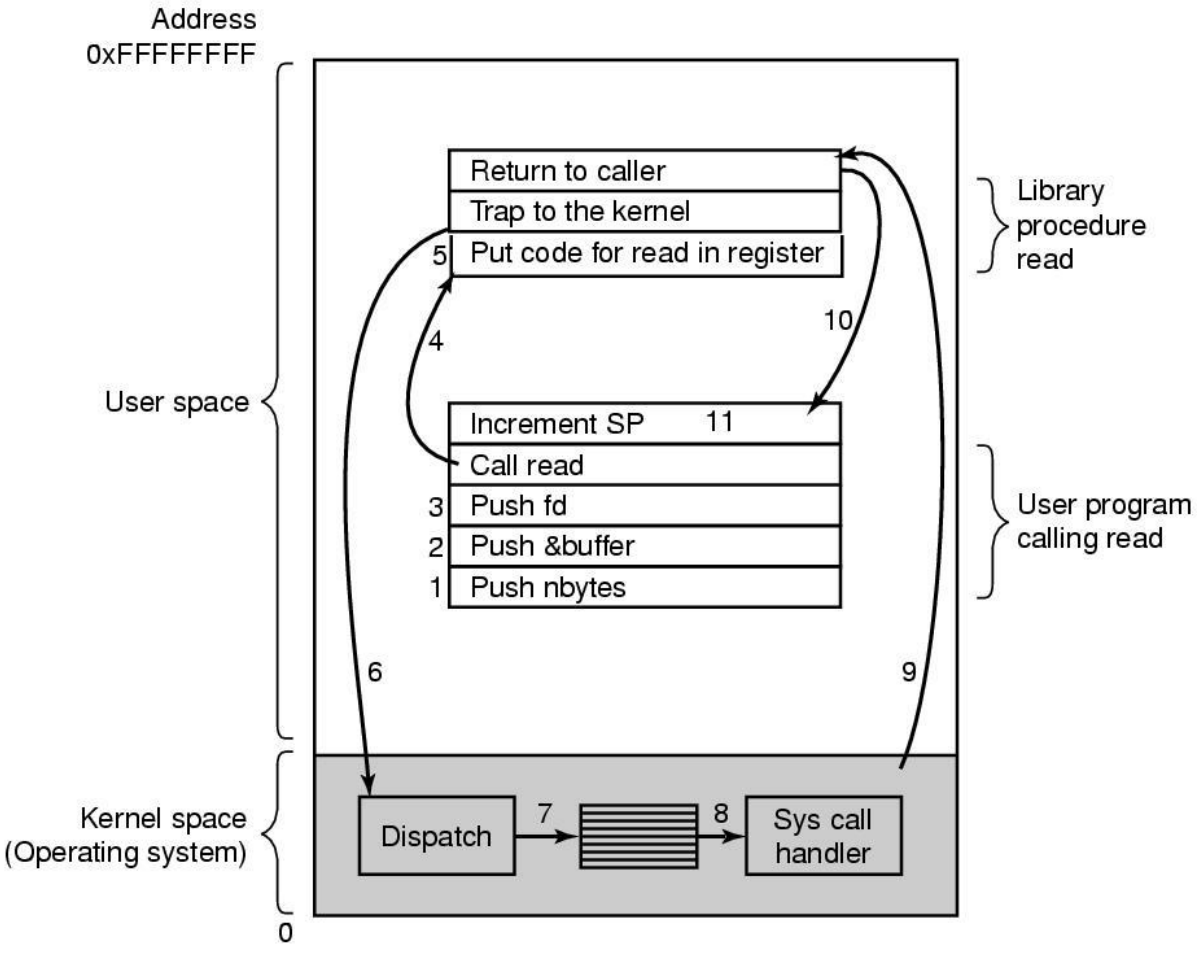

Figure : The 11 steps in making the system call read(fd, buffer, nbytes).

System Calls

- Interface between user programs and OS
- Varies from OS to OS
- System call issued by user program
- Call uses a library call of the same name
- Library routine puts machine into kernel modes (by issuing a special instruction)
- Finds actual routine for system call in a table
- Does the work involved in the call
- Returns to user program

Types of System Calls

The system calls can be classified in following types:

Process Management	☞ These system calls deal with processes such as process creation, process termination etc.
File Management	☞ These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.
Directory Management	☞ These system calls are responsible for system calls that are related to directories or the file system like creating new directory, mounting filesystem etc.
Miscellaneous	☞ Other different system calls for different tasks

Some system calls in POSIX system

Some of the major POSIX system calls are listed below. The return code *s* is −1 if an error has occurred. The return codes are as follows: *pid* is a process id, *fd* is a file descriptor, *n* is a byte count, *position* is an offset within the file, and *seconds* is the elapsed time.

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Directory- and file-system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

System calls for Process management

- System calls for processes are specific functions provided by the operating system that allow programs to interact with the underlying operating system kernel and perform various operations related to process management. These system calls provide an interface for processes to request services from the operating system.
- Some Process management system calls in Linux are listed below:

SystemCall	Description
fork()	Create a new process by duplicating the existing process. The new process is an exact copy of the parent process.
exec()	Replace the current process with a new process. It loads a new executable file into the current process's memory and starts its execution.
wait()	Suspend the execution of a parent process and wait for the termination of its child process. Retrieve the exit status of the child process.
exit()	Terminate the execution of the current process and return an exit status to the parent process.
getpid()	Retrieve the process ID (PID) of the calling process.
getppid()	Retrieve the parent process ID (PPID) of the calling process.
kill()	Send a signal to a specified process or group of processes. Signals are used to notify processes about events or request specific actions.
nice()	Adjust the scheduling priority of a process. Influence the amount of CPU time the process receives.

pipe()	Create a pipe, a unidirectional data channel that allows inter-process communication (IPC) between two related processes.
signal()	Manipulate signal handling by specifying the behavior of the process when a particular signal is received.
execve()	Replace the current process with a new process specified by the given executable file path. It allows passing command-line arguments and environment variables to the new process.
setuid()	Set the effective user ID of the calling process. Useful for changing the user context and permissions.
setgid()	Set the effective group ID of the calling process. Useful for changing the group context and permissions.
sched_yield()	Relinquish the CPU voluntarily, allowing other processes with the same priority to execute.
clone()	Create a new process (similar to fork()), but with more flexibility to specify different levels of sharing between parent and child processes.

Storage Structure

The CPU can load instructions only from memory, so any programs must first be loaded into memory to run. 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. In addition, it is low speed, and so it contains mostly static programs and data that aren't frequently used. For example, the iPhone uses EEPROM to store serial numbers and hardware information about the device.

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

register to main memory. Aside from explicit loads and stores, the CPU automatically loads instructions from main memory for execution from the location stored in the program counter.

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:

- Main memory is usually too small to store all needed programs and data permanently.
- Main memory, as mentioned, is volatile—it 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. Secondary storage is also much slower than main memory. Hence, the proper management of secondary storage is of central importance to a computer system.

Each storage system provides the basic functions of storing a datum and holding that datum until it is retrieved at a later time. The main differences among the various storage systems lie in speed, size, and volatility.

The wide variety of storage systems can be organized in a hierarchy as in following figure according to storage capacity and access time. 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.

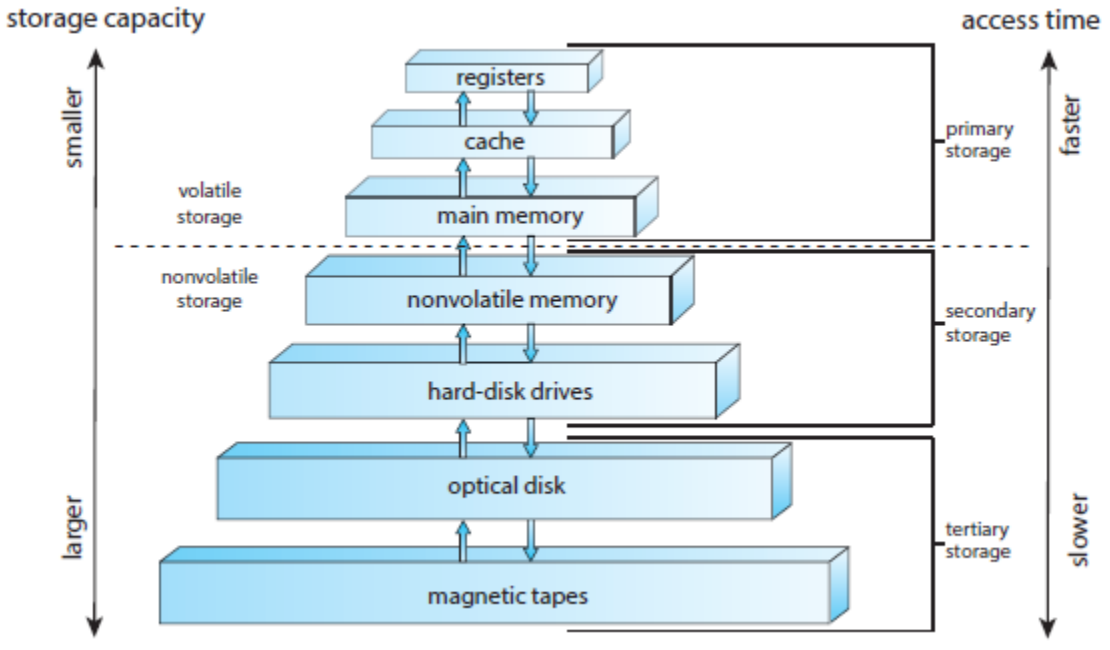

Figure: Storage-device hierarchy.

The top four levels of memory in the figure are constructed using semiconductor memory, which consists of semiconductor-based electronic circuits. NVM (nonvolatile memory) devices, at the fourth level, have several variants but in general are faster than hard disks. 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.

The design of a complete storage system must balance all the factors just discussed: it must use only as much expensive memory as necessary while providing as much inexpensive, nonvolatile storage as possible. Caches can be installed to improve performance where a large disparity in access time or transfer rate exists between two components.

I/O Structure

A large portion of operating system code is dedicated to managing I/O, both because of its importance to the reliability and performance of a system and because of the varying nature of the devices.

A general-purpose computer system consists of multiple devices, all of which exchange data via a common bus. The form of interrupt-driven I/O is fine for moving small

amounts of data but can produce high overhead when used for bulk data movement. To solve this problem, 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, to tell the device driver that the operation has completed, rather than the one interrupt per byte generated for low-speed devices. While the device controller is performing these operations, the CPU is available to accomplish other work.

Some high-end systems use switch rather than bus architecture. On these systems, multiple components can talk to other components concurrently, rather than competing for cycles on a shared bus. In this case, DMA is even more effective. Following figure shows the interplay of all components of a computer system.

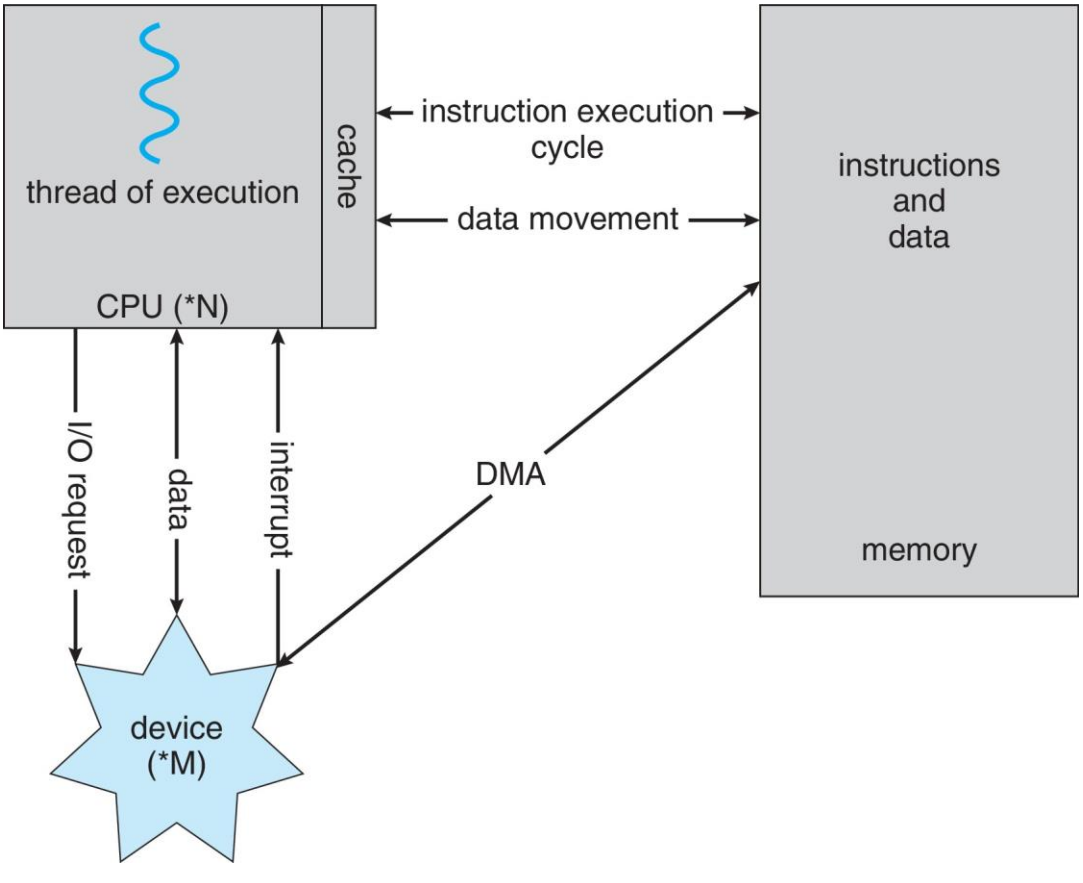

Figure : How a modern computer system works.

Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
    - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
    - User identities (**user IDs**, security IDs) include name and associated number, one per user
    - User ID then associated with all files, processes of that user to determine access control
    - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
    - **Privilege escalation** allows user to change to effective ID with more rights