INTRODUCTION

Operating System is a program that acts as an interface between a user of a computer and the computer hardware. Eg. Windows (Microsoft), iOS (Apple), Android (Google), Linux/Ubuntu (open source)

Goals of Operating System:

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

COMPUTER SYSTEM

OA computer system can be divided roughly into four components: the **hardware,** the

operating system, the **application programs,** and a **user**

Hardware: The Central Processing Unit (CPU), the Memory, and the Input/output (I/O) devices provides the basic computing resources for the system.

Application Programs: Such as word processors, spreadsheets, compilers, and web browsers define the ways in which these resources are used to solve users' computing problems.

Operating System: Controls the hardware and coordinates its use among the various application programs for the various users.

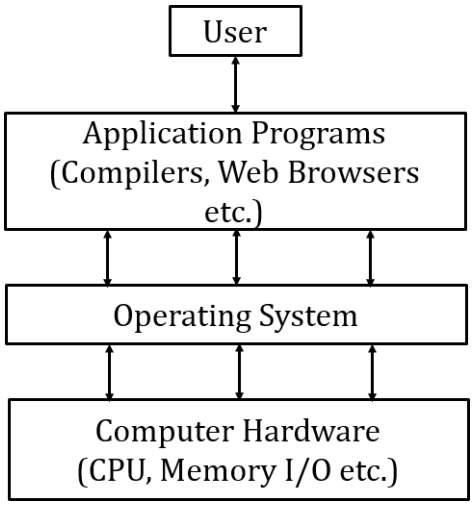


Figure : Computer System

The operating system provides the means for proper use of these resources in the operation of the computer system.

Operating Systems from two viewpoints:

- User View
- System View

User View

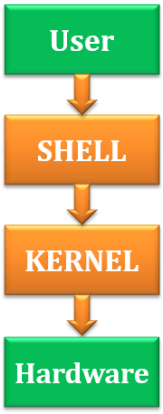
The user's view of the computer varies according to the interface being used. Many computer users sit with a laptop or in front of a PC consisting of a monitor, keyboard, and mouse. Such a system is designed for one user to control its resources. For this, the operating system is designed mostly for **ease of use**, performance and security and not to **resource utilization.**

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

System View

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

Components of Operating System



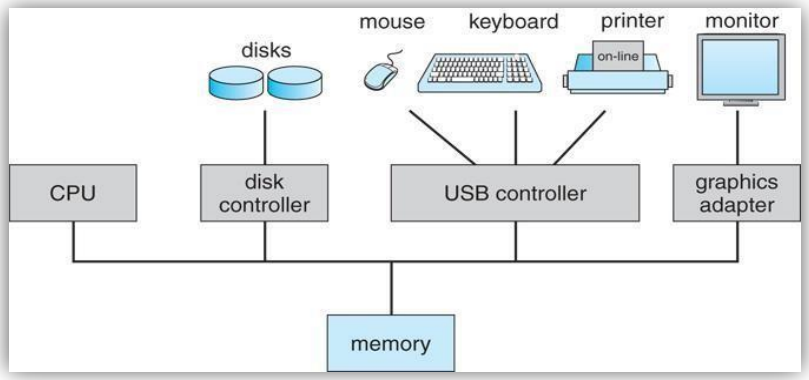
Shell :

- Environment that gives a user an interface to access the operating system's services.
- Launch Applications
- User Mode

Kernel :

- Keep track of the hardware and computer's operations.
- Kernel Mode

COMPUTER SYSTEM – ELEMENTS AND ORGANIZATION



A computer system consists of CPU and a number of device controllers connected through a common **bus** that provides access between components and shared memory. Each device controller is in charge of a specific type of device, more than one device may be attached. For example, one system USB port can connect to a USB hub, to which several devices can be connected. A device controller maintains some local buffer storage and a set of special purpose registers. The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage.

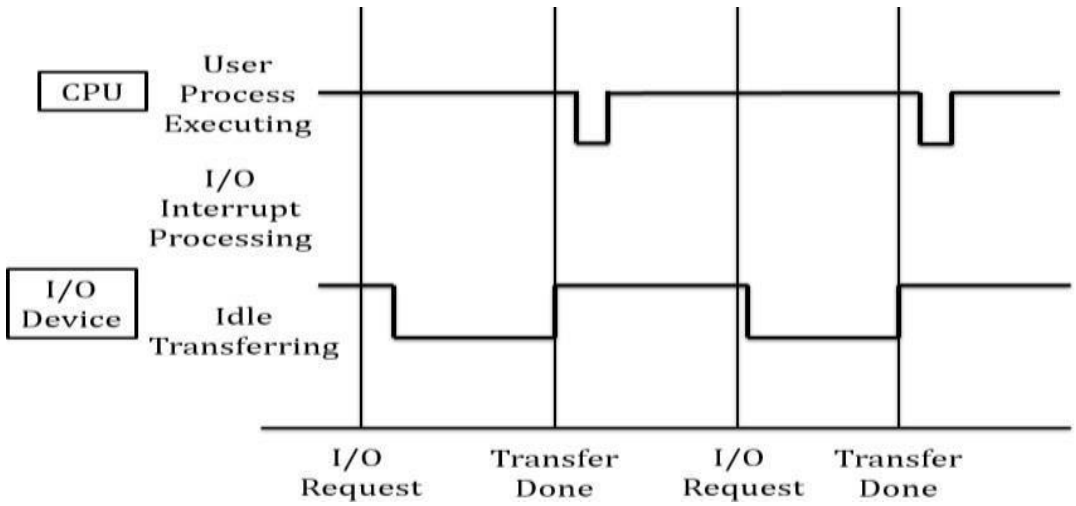
Operating systems have a **device driver** for each device controller. This device driver understands the device controller and provides the rest of the operating system with a uniform interface to the device. The CPU and the device controllers can execute in parallel.

Three key aspects of the system are

- Interrupts
- Storage Structure
- I/O Structure

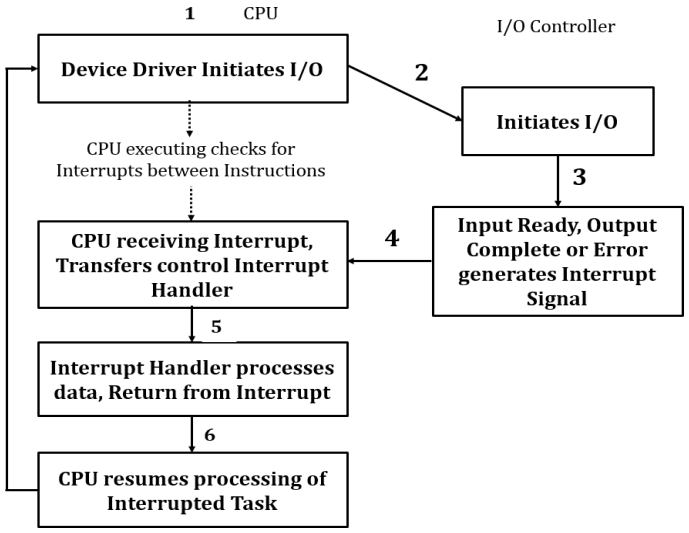
❖ Interrupts

Hardware may trigger an interrupt by sending a signal to the CPU. Software may trigger an interrupt by executing a special operation called a **system call.** When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location. The fixed location usually contains the starting address where the service routine for the interrupt is located. The interrupt service routine executes; on completion, the CPU resumes the interrupted computation. A time line of this operation is shown below



The interrupt routine is called indirectly through the table, The table holds the addresses of the interrupt service routines for the various devices. This array, or **interrupt vector,** of addresses is then indexed by a unique device number, given with the interrupt request, to provide the address of the interrupt service routine for the interrupting device.

The interrupt architecture must also save the address of the interrupted instruction. After the interrupt is serviced, the saved return address is loaded into the program counter, and the interrupted computation resumes as though the interrupt had not occurred. The device controller raises an interrupt by asserting a signal on the interrupt request line, the CPU catches the interrupt and dispatches it to the interrupt handler, and the handler clears the interrupt by servicing the device.



Most CPUs have two interrupt request lines. One is the **nonmaskable interrupt**, which is reserved for events such as unrecoverable memory errors. The second interrupt line is **maskable**: it can be turned off by the CPU before the execution of critical instruction sequences that must not be interrupted. The maskable interrupt is used by device controllers to request service.

❖ Storage Structure

- KB : 1024 Bytes
- MB : 1024^2 Bytes
- GB : 1024^3 Bytes (1 Million Bytes)
- TB : 1024^4 Bytes (1 Billion Bytes)
- PB : 1024^5 Bytes

The CPU load the instructions from memory. General-purpose computers run most of their programs from rewritable memory, called main memory (**RAM**).

The first program to run on computer to 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. So the bootstrap program cannot be stored in RAM. So the computer uses electrically erasable programmable read-only memory (EEPROM) and other forms of **firmware,** 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.

The CPU automatically loads instructions from main memory for execution from the location stored in the program counter. First fetches an instruction from memory and stores that instruction in the **instruction register**. The instruction is then decoded and may cause operands to be fetched from memory and stored in some internal register. After the instruction on the operands has been executed, the result may be stored back in memory.

The programs and data must be in main memory permanently. This arrangement 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, is volatile, it loses its contents when power is turned off or otherwise lost.

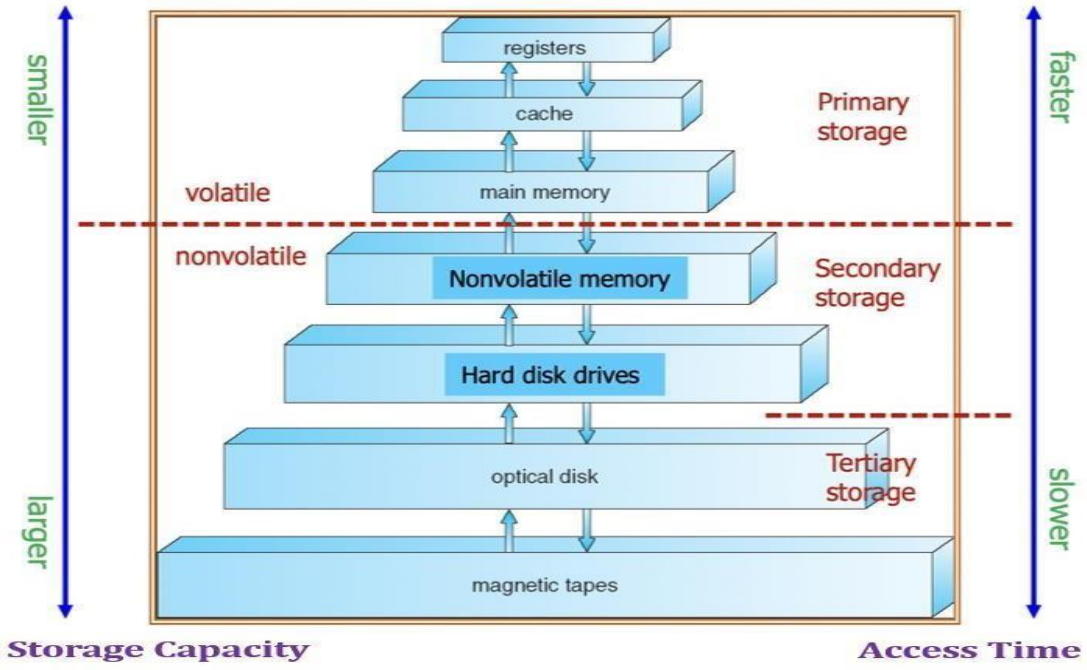
Most computer systems provide **secondary storage** as an extension of main memory. The main requirement for secondary storage is that it be able to hold large quantities of data permanently. The most common secondary-storage devices are **hard-disk drives** (**HDDs**) and **nonvolatile memory** (**NVM**) **devices**, which provide storage for both programs and data. Most programs are stored in secondary storage until they are loaded into memory.

Secondary storage is also much slower than main memory. Other possible components include cache memory, CD-ROM, magnetic tapes, and so on. Those that are slow enough and large enough that they are used only for special purposes, to store backup copies of material stored on other devices, are called **tertiary storage**.

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

Smaller and faster memory closer to the CPU. Various storage systems are either volatile or nonvolatile. Volatile storage, loses its contents when the power to the device is removed, so data must be written to nonvolatile storage for safekeeping.

The top four levels of memory are constructed using **semiconductor memory**, which consists of semiconductor based electronic circuits. Non Volatile Memory devices, at the fourth level, 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.



Volatile storage will be referred to simply as **memory**. Nonvolatile storage retains its contents when power is lost. This type of storage can be classified into two distinct types:

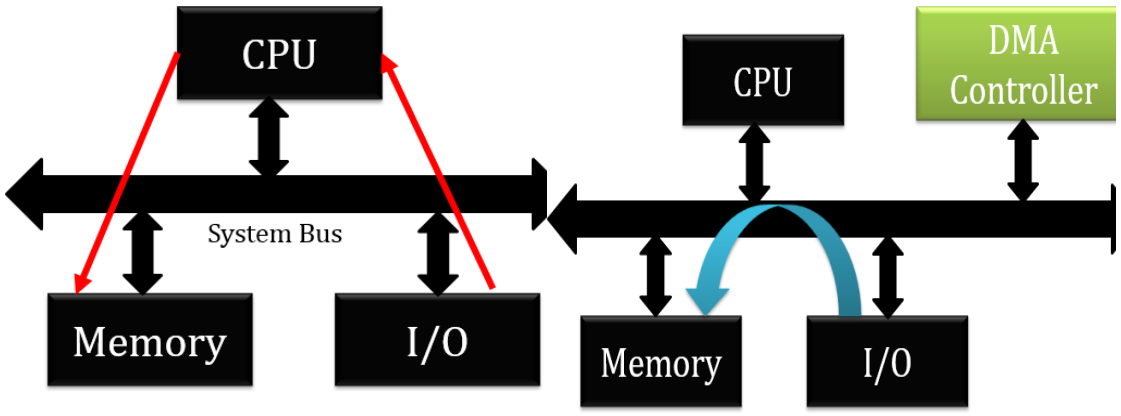
- **Mechanical** : Storage systems are HDDs, optical disks, holographic storage, and magnetic tape.
- **Electrical : S**torage systems are flash memory, FRAM, NRAM, and SSD.

Mechanical storage is generally larger and less expensive per byte than electrical storage.

Conversely, electrical storage is typically costly, smaller, and faster than mechanical storage.

❖ **I/O Structure**

A large portion of operating system code is dedicated to manage I/O. 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, **direct memory access** (**DMA**) is used for bulk data transfer.

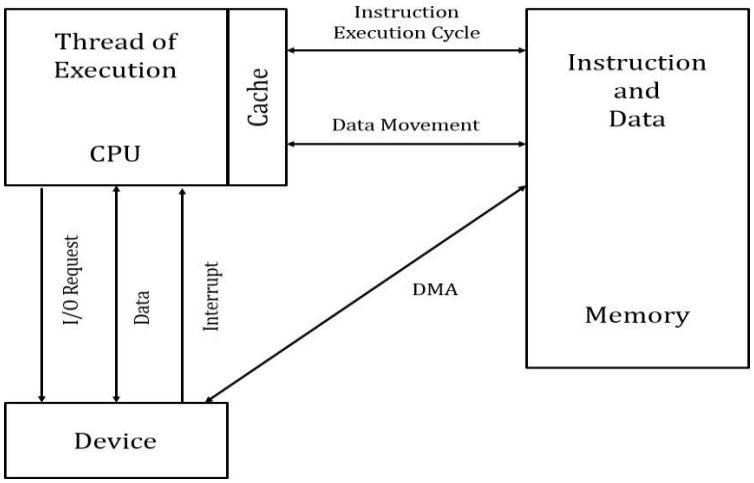


Normal Data Transfer

DMA Data Transfer

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.



EVOLUTION OF OPERATING SYSTEM

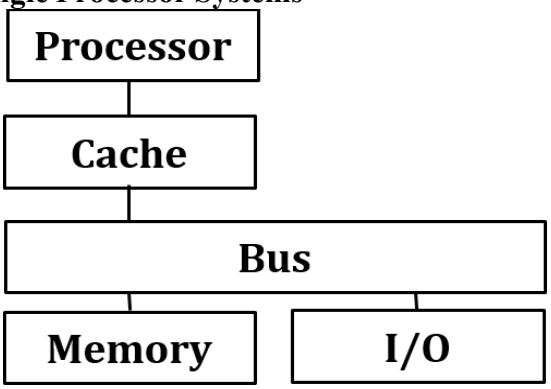
CPU : The hardware that executes instructions. Processor : A physical chip that contains one or more CPUs. Core : The basic computation unit of the CPU.

Multicore : Including multiple computing cores on the same CPU. Multiprocessor : Including multiple processors.

According to the number of general-purpose processors used it is divided into

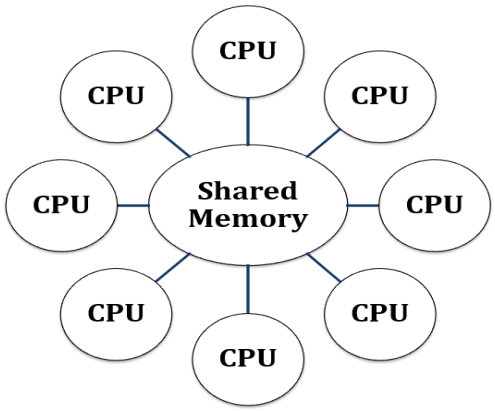
- ❖ Single Processor Systems
- ❖ Multiprocessor Systems
- ❖ Clustered Systems

❖ **Single Processor Systems**



On a single-processor system, there is one main CPU capable of executing the instructions. Most computer systems use a single processor containing one CPU with a single processing core. The **core** is the component that executes instructions and registers for storing data locally. The one main CPU with its core is capable of executing a general-purpose instruction set, including instructions from processes.

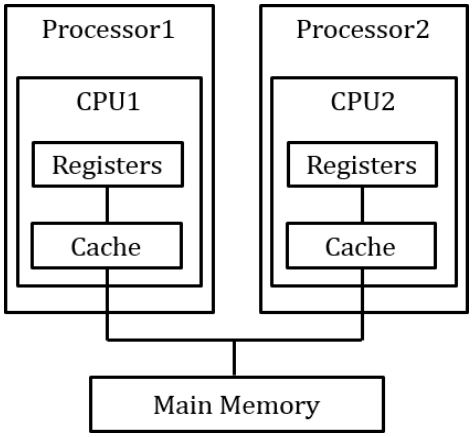
❖ **Multiprocessor Systems**



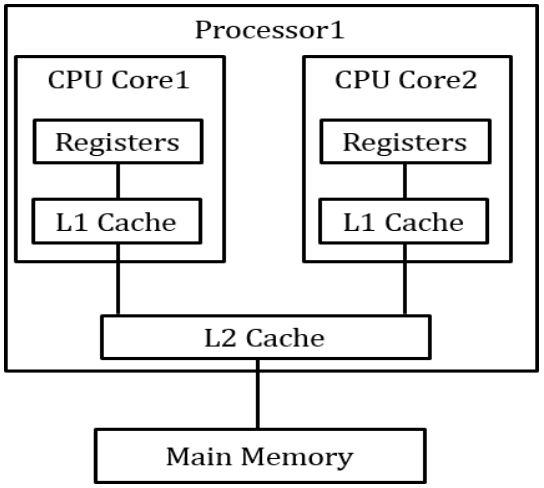
They have more processors, each with a single-core CPU. The processors share the computer bus and sometimes the clock, memory, and peripheral devices. The main advantages are

- Increased throughput
- Economy of scale
- Increased reliability – graceful degradation or fault tolerance

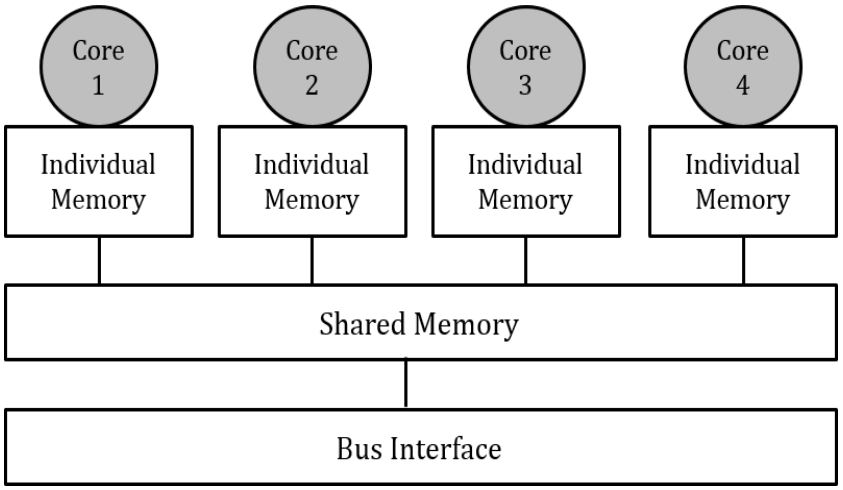
The advantage of multiprocessor systems is increased throughput. By increasing the number of processors, more work will be done in less time. The most common multiprocessor systems use **symmetric multiprocessing** (**SMP**), in which each peer CPU processor performs all tasks, including operating-system functions and user processes.



The Figure illustrates a typical SMP architecture with two processors, each with its own CPU. Each CPU processor has its own set of registers, and local cache. Main Memory is shared. The benefit is that many processes can run simultaneously *N* processes can run if there are *N* CPUs without causing performance to deteriorate significantly. However, since the CPUs are separate, one may be sitting idle while another is overloaded, resulting in inefficiencies. These inefficiencies can be avoided if the processors share certain data structures. The definition of **multiprocessor** has evolved over time and now includes **multicore** systems, in which multiple computing cores reside on a single chip. Multicore systems can be more efficient than multiple chips with single cores.

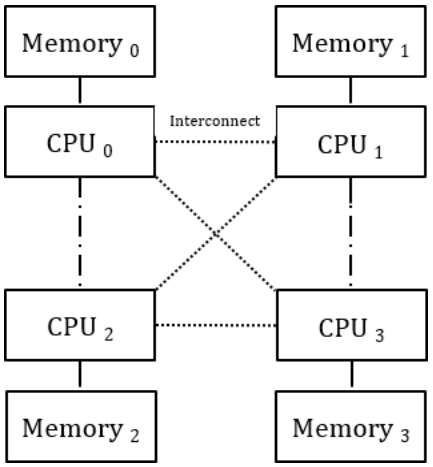


In a dual-core design with two cores on the same processor chip. In this design, each core has its own register set, as well as its own local cache, often known as a level 1, or L1, cache. Notice, too, that a level 2 (L2) cache is local to the chip but is shared by the two processing cores.



NUMA Multiprocessing Architecture

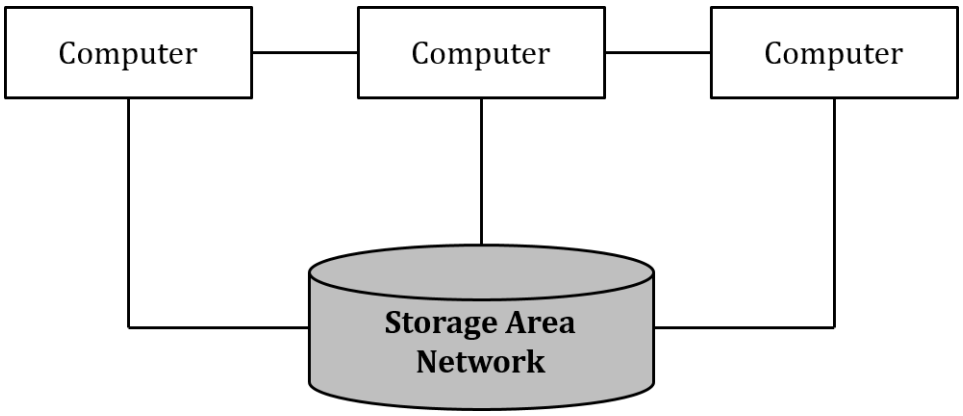
Adding additional CPUs to a multiprocessor system will increase computing power; and adding too many CPUs, becomes a bottleneck and performance begins to degrade. Instead to provide each CPU (or group of CPUs) with its own local memory that is accessed via a small, fast local bus. The CPUs are connected by a **shared system interconnect**, so that all CPUs share one physical address space. This approach known as **Non-Uniform Memory Access**, or **NUMA**



The advantage is that, when a CPU accesses its local memory, it is fast, and no contention over the system interconnect. Thus, NUMA systems can scale more effectively as more processors are added. A drawback is **increased latency**. For example, CPU0 cannot access the local memory of CPU3 as quickly as it can access its own local memory, slowing down performance.

Because NUMA systems can scale to accommodate a large number of processors, they are becoming increasingly popular on servers as well as high-performance computing systems.

- **Clustered Systems**

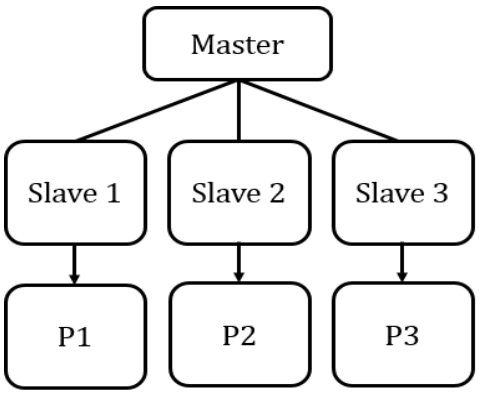


Clustered computers share storage and are closely linked via a local-area network LAN or a faster interconnect. Clustering is usually used to provide **high availability service** that is, service that will continue even if one or more systems in the cluster fail.

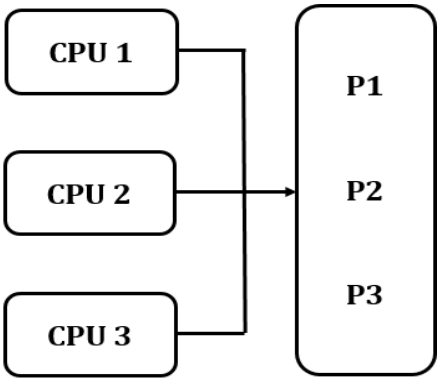
A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the others (over the network). If the monitored machine fails, the monitoring machine can take ownership of its storage and restart the applications that were running on the failed machine. High availability provides increased reliability, which is crucial in many applications. The ability to continue providing service proportional to the level of surviving hardware is called **graceful degradation**. Some systems go beyond graceful degradation and are called **fault tolerant**, because they can suffer a failure of any single component and still continue operation. Fault tolerance requires a mechanism to allow the failure to be detected, diagnosed, and, if possible, corrected.

Clustering can be structured asymmetrically or symmetrically.

In **Asymmetric Clustering**, one machine is in hot standby mode while the other is running the applications. The hot standby host machine does nothing but monitor the active server. If that server fails, the hot standby host becomes the active server.



Asymmetric Clustering



Symmetric Clustering

In **symmetric clustering**, two or more hosts are running applications and are monitoring each other. This structure is obviously more efficient, as it uses all of the available hardware.

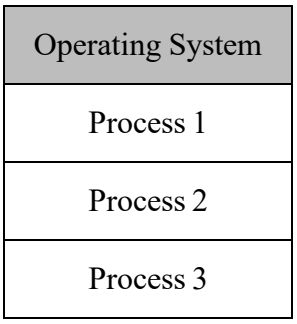
OPERATING SYSTEM OPERATIONS

- ❖ Multiprogramming and Multitasking
- ❖ Dual-Mode and Multimode Operation
- ❖ Timer

Multiprogramming

Multiprogramming increases CPU utilization, so that the CPU always has one to execute.

In a multiprogrammed system, a program in execution is termed as **process**.



Memory Layout of a Multiprogramming System

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

Multitasking

CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing

- **Response time** should be < 1 second
- Each user has at least one program executing in memory
- If several jobs ready to run at the same time [**CPU scheduling]**
- If processes don't fit in memory, **swapping** moves them in and out to run
- **Virtual memory** allows execution of processes not completely in memory

❖ **Dual-Mode and Multimode Operation**

The following are the modes

- **User Mode:**

Operating system running a user application such as handling a text editor. The transition from user mode to kernel mode occurs when the application requests the help of operating

system or an interrupt or a system call occurs. The mode bit is set to 1 in the user mode. It is changed from 1 to 0 when switching from user mode to kernel mode.

- **Kernel Mode**

The system starts in kernel mode when it boots and after the operating system is loaded, it executes applications in user mode. There are some privileged instructions that can only be executed in kernel mode. These are interrupt instructions, input output management etc. If the privileged instructions are executed in user mode, it is illegal and a trap is generated. The mode bit is set to 0 in the kernel mode. It is changed from 0 to 1 when switching from kernel mode to user mode.

The concept of modes of operation in operating system can be extended beyond the dual mode. Known as the **multimode** system. In those cases more than 1 bit is used by the CPU to set and handle the mode.

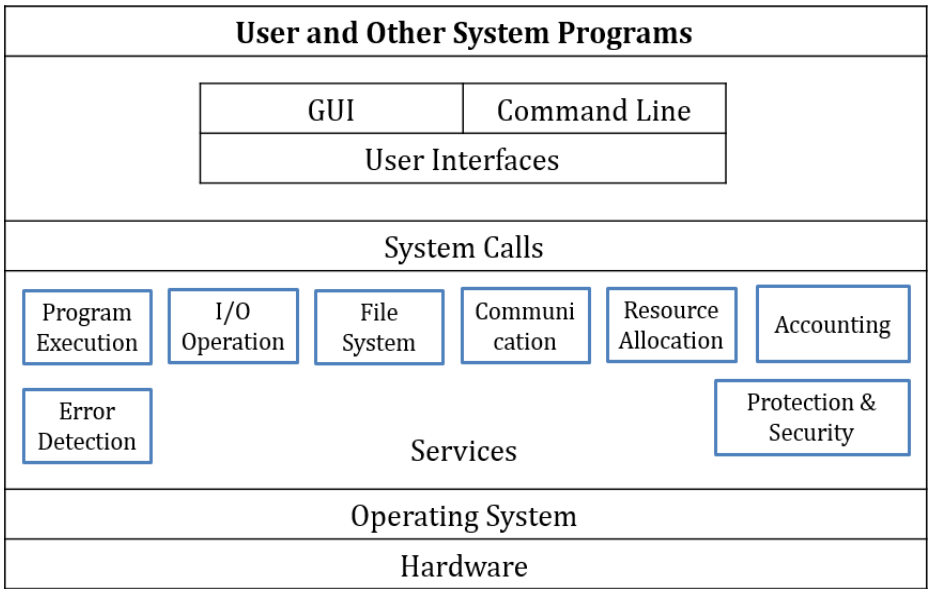
❖ **Timer**

A user program cannot get stuck in an infinite loop or to fail to call system services and never return control to the operating system. A timer is used to accomplish this goal. A timer can be set to interrupt the computer after a specified period. A **variable timer** is generally implemented by a fixed-rate clock and a counter. The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs.

OPERATING SYSTEM SERVICES

Furthermore, the operating system, in one form or another, provides certain services to the computer system.

- ❖ User Interface
- ❖ Program Execution
- ❖ I/O Operations
- ❖ File System Manipulation
- ❖ Communications
- ❖ Error Detection
- ❖ Resource Allocation
- ❖ Accounting
- ❖ Protection and Security
- ❖

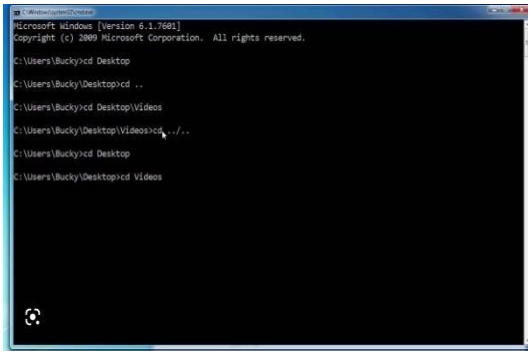


❖ **User Interface**

Almost all operating systems have a **user interface (UI)**. Two types of User Interface are Command Based Interface and Graphical User Interface

Command Based Interface

Requires a user to enter the commands to perform different tasks like creating, opening, editing or deleting a file, etc. The user has to remember the names of all such programs or specific commands which the operating system supports. The primary input device used by the user for command based interface is the keyboard. Command-based interface is often less interactive and usually allows a user to run a single program at a time. Examples of operating systems with command-based interfaces include MS-DOS and Unix.



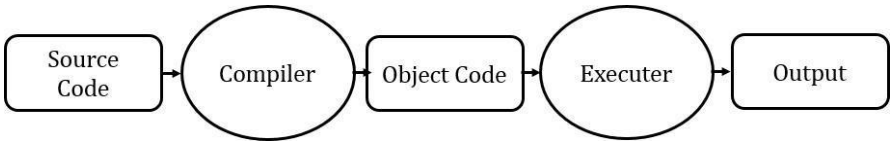
Command Based Interface **Graphical User Interface**

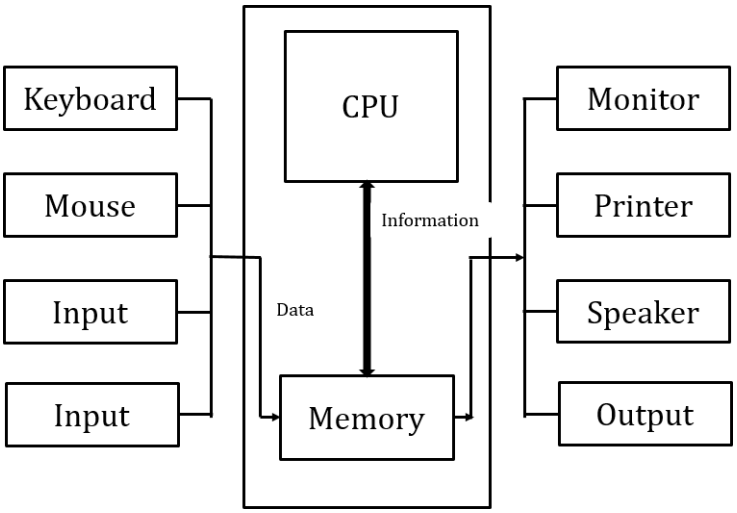
Graphical User Interface (GUI)

The interface is a window system with a mouse that serves as a pointing device to direct I/O, choose from menus, and make selections and a keyboard to enter text. Mobile systems such as phones and tablets provide a **touch-screen interface**, enabling users to slide their fingers across the screen or press buttons on the screen to select choices.

❖ **Program Execution**:

The OS is in charge of running all types of programs, whether they are user or system programs. The operating system makes use of a variety of resources to ensure that all types of functions perform smoothly.





❖ **Input/Output Operations**:The operating system is in charge of handling various types of inputs, such as those from the keyboard, mouse, and desktop. Regarding all types of inputs and outputs, the operating system handles all erfaces in the most appropriate manner.

For instance, the nature of all types of peripheral devices, such as mice or keyboards, differs, and the operating system is responsible for transferring data between them.

❖ **File System Manipulation**:

The OS is in charge of deciding where data or files should be stored, such as on a floppy disk, hard disk, or pen drive. The operating system determines how data should be stored and handled.

❖ **Communications :**

There are many circumstances in which one process needs to exchange information with another process. Such communication may occur between processes that are executing on the same computer or between processes that are executing on different computer systems tied

together by a network. Communications may be implemented via **shared memory**, in which two or more processes read and write to a shared section of memory, or **message passing**, in which packets of information in predefined formats are moved between processes by the operating system.

- **Error Detection:**

The operating system needs to be detecting and correcting errors constantly. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (such as a parity error on disk, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow or an attempt to access an illegal memory location). For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing. Sometimes, it has no choice but to halt the system. At other times, it might terminate an error-causing process or return an error code to a process for the process to detect and possibly correct.

- **Resource Allocation:**

The operating system guarantees that all available resources are properly utilized by determining which resource should be used by whom and for how long. The operating system makes all of the choices.

- **Accounting:**

The operating system keeps track of all the functions that are active in the computer system at any one time. The operating system keeps track of all the facts, including the types of mistakes that happened.

- **Protection and Security :**

The operating system is in charge of making the most secure use of all the data and resources available on the machine. Any attempt by an external resource to obstruct data or information must be foiled by the operating system.

USER AND OPERATING SYSTEM INTERFACE

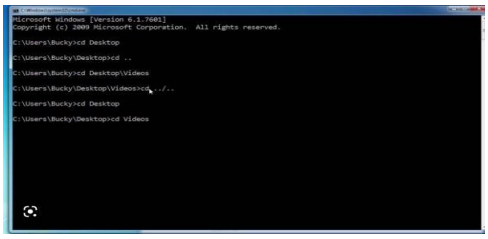
There are different types of user interfaces each of which provides a different functionality:

- Command Based Interface
- Graphical User Interface
- Touch Based Interface
- Voice Based Interface
- Gesture Based Interface

Command Based Interface

Command based interface requires a user to enter the commands to perform different tasks like creating, opening, editing or deleting a file, etc. The user has to remember the names of all such programs or specific commands which the operating system supports. The primary input device used by the user for command based interface is the keyboard. Command-based

interface is often less interactive and usually allows a user to run a single program at a time. Examples of operating systems with command-based interfaces include MS-DOS and Unix.



Command Based Interface **Graphical User Interface**

Graphical User Interface

Users run programs or give instructions to the computer in the form of icons, menus and other visual options. Icons usually represent files and programs stored on the computer and windows represent running programs that the user has launched through the operating system. The input devices used to interact with the GUI commonly include the mouse and the keyboard. Examples of operating systems with GUI interfaces include Microsoft Windows, Ubuntu, Fedora and Macintosh, among others.

Touch Based Interface

Today smartphones, tablets, and PCs allow users to interact with the system simply using the touch input. Using the touchscreen, a user provides inputs to the operating system, which are interpreted by the OS as commands like opening an app, closing an app, dialing a number, scrolling across apps, etc.

Examples of popular operating systems with touch-based interfaces are Android and iOS. Windows 8.1 and 10 also support touch-based interfaces on touchscreen devices.

Touch Based Interface **Voice Based Interface**

Voice Based Interface

Modern computers have been designed to address the needs of all types of users including people with special needs and people who want to interact with computers or smartphones while doing some other task. For users who cannot use input devices like the mouse, keyboard, and touchscreens, modern operating systems provide other means of human-computer interaction. Users today can use voice-based commands to make a computer work in the desired way. Some operating systems which provide voice-based control to users include iOS (Siri), Android (Google Now or "OK Google"), Microsoft Windows 10 (Cortana), and so on.

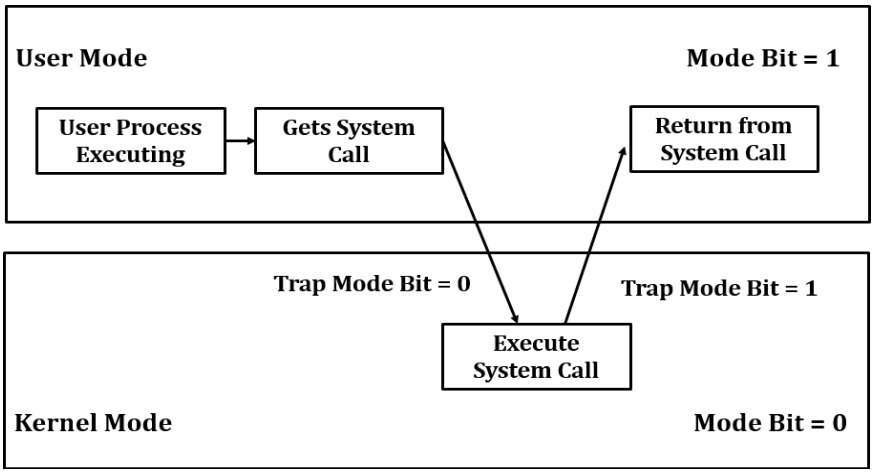
Gesture Based Interface

Some smartphones based on Android and iOS as well as laptops let users interact with the devices using gestures like waving, tilting, eye motion, and shaking. This technology is evolving faster and it has promising potential for application in gaming, medicine, and other areas.

SYSTEM CALLS

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. System call offers the services of the operating system to the user programs via API (Application Programming Interface). System calls are the only entry points for the kernel system.

Working of System Call



Step 1) The processes executed in the user mode till the time a system call interrupts it.

Step 2) After that, the system call is executed in the kernel-mode on a priority basis.

Step 3) Once system call execution is over, control returns to the user mode.,

Step 4) The execution of user processes resumed in Kernel mode.

Need of System Call

- Reading and writing from files demand system calls.
- If a file system wants to create or delete files, system calls are required.
- System calls are used for the creation and management of new processes.
- Network connections need system calls for sending and receiving packets.
- Access to hardware devices like scanner, printer, need a system call.

Example of how system calls are used.

For Example

Writing a simple program to read data from one file and copy them to another file.

The first input that the program will need is the names of the two files: the input file and the output file. These names can be specified in many ways, depending on the operating-system design.

One approach is to pass the names of the two files as part of the command For example, the UNIX cp command:

cp in.txt out.txt

This command copies the input file in.txt to the output file out.txt.

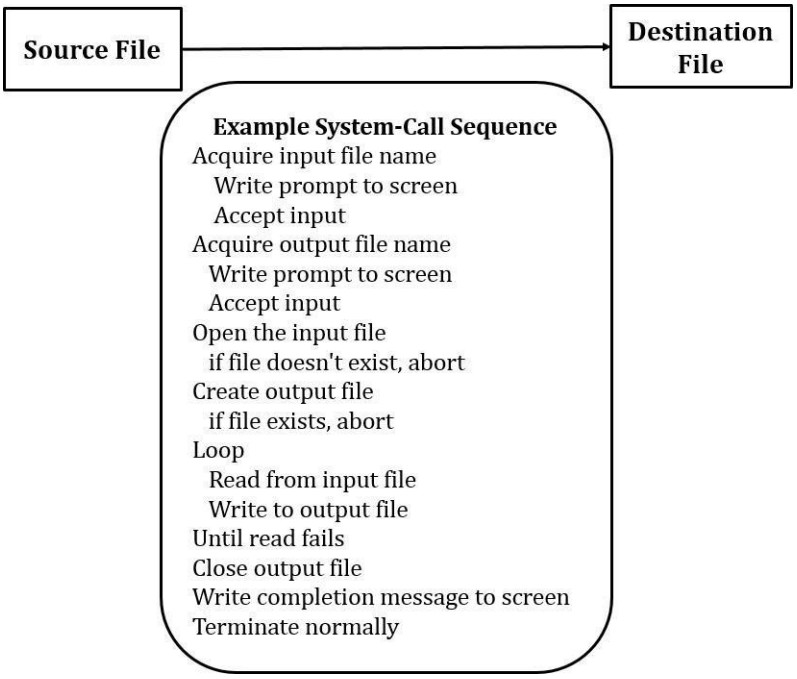
A second approach is for the program to ask the user for the names.

In an interactive system, this approach will require a sequence of system calls, first to write a prompting message on the screen and then to read from the keyboard the characters that define

the two files. On mouse-based and icon-based systems, a menu of file names is usually displayed in a window. The user can then use the mouse to select the source name, and a window can be opened for the destination name to be specified. This sequence requires many I/O system calls.

Once the two file names have been obtained, the program must open the input file and create and open the output file. Each of these operations requires another system call. Possible error conditions for each system call must be handled. For example, when the program tries to open the input file, it may find that there is no file of that name or that the file is protected against access.

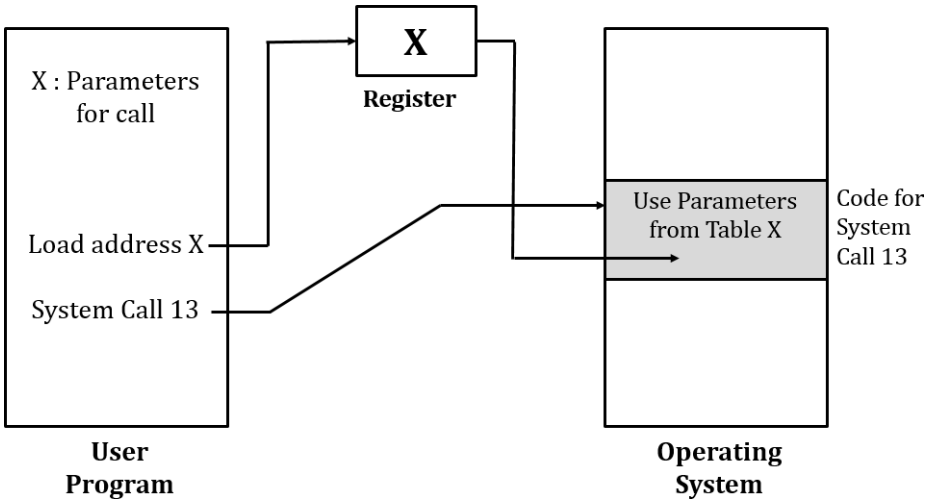
In these cases, the program should output an error message (another sequence of system calls) and then terminate abnormally (another system call). If the input file exists, then we must create a new output file. We may find that there is already an output file with the same name. This situation may cause the program to abort (a system call), or we may delete the existing file (another system call) and create a new one (yet another system call). Another option, in an interactive system, is to ask the user (via a sequence of system calls to output the prompting message and to read the response from the terminal) whether to replace the existing file or to abort the program.



When both files are set up, we enter a loop that reads from the input file (a system call) and writes to the output file (another system call). Each read and write must return status information regarding various possible error conditions. On input, the program may find that the end of the file has been reached or that there was a hardware failure in the read (such as a parity error). The write operation may encounter various errors, depending on the output device (for example, no more available disk space).

Finally, after the entire file is copied, the program may close both files (two system calls), write a message to the console or window (more system calls), and finally terminate normally (the final system call).

Passing of Parameters as a table



Rules for passing Parameters for System Call

Here are general common rules for passing parameters to the System Call:

- Parameters should be pushed on or popped off the stack by the operating system.
- Parameters can be passed in registers. For five or fewer parameters, registers are used.
- More than five parameters, the block method is used. The parameters are generally stored in a block, or table, in memory, and the address of the block is passed as a parameter in a register

Types of System calls

Here are the five types of System Calls in OS:

- ❖ Process Control
- ❖ File Management
- ❖ Device Management
- ❖ Information Maintenance
- ❖ Communications

Process Control

This system calls perform the task of process creation, process termination, etc.

Functions:

- End and Abort
- Load and Execute
- Create Process and Terminate Process
- Wait and Signal Event
- Allocate and free memory

File Management

File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc.

Functions:

- Create a file
- Delete file
- Open and close file
- Read, write, and reposition
- Get and set file attributes

Device Management

Device management does the job of device manipulation like reading from device buffers, writing into device buffers, etc.

Functions:

- Request and release device
- Logically attach/ detach devices
- Get and Set device attributes

Information Maintenance

It handles information and its transfer between the OS and the user program.

Functions:

- Get or set time and date
- Get process and device attributes

Communication:

These types of system calls are specially used for interprocess communications.

Functions:

- Create, delete communications connections
- Send, receive message
- Help OS to transfer status information
- Attach or detach remote devices

Important System Calls Used in

OS wait()

A process needs to wait for another process to complete its execution. This occurs when a parent process creates a child process, and the execution of the parent process remains suspended until its child process executes. The suspension of the parent process automatically occurs with a wait() system call. When the child process ends execution, the control moves back to the parent process.

fork()

Processes use this system call to create processes that are a copy of themselves. With the help of this system Call parent process creates a child process, and the execution of the parent process will be suspended till the child process executes.

exec()

This system call runs when an executable file in the context of an already running process that replaces the older executable file. However, the original process identifier remains as a new process is not built, but stack, data, head, data, etc. are replaced by the new process.

kill():

The kill() system call is used by OS to send a termination signal to a process that urges the process to exit. However, a kill system call does not necessarily mean killing the process and can have various meanings.

exit():

The exit() system call is used to terminate program execution. Specially in the multi- threaded environment, this call defines that the thread execution is complete. The OS reclaims resources that were used by the process after the use of exit() system call.

SYSTEM PROGRAMS

System programs provide a convenient environment for program development and execution. It can be divided into:

- File manipulation
- Status information
- File modification
- Programming language support
- Program loading and execution
- Communications
- Application programs

- **File management**.

These programs create, delete, copy, rename, print, list, and generally access and manipulate files and directories.

- **Status information**.

Some programs simply ask the system for the date, time, amount of available memory or disk space, number of users, or similar status information. Others are more complex, providing detailed performance, logging, and debugging information. Typically, these programs format and print the output to the terminal or other output devices or files or display it in a window

of the GUI. Some systems also support a **registry**, which is used to store and retrieve configuration information.

- **File modification:** .

Several text editors may be available to create and modify the content of files stored on disk or other storage devices. There may also be special commands to search contents of files or perform transformations of the text.

- **Programming-language support**:

Compilers, assemblers, debuggers, and interpreters for common programming languages (such as C, C++, Java, and Python) are often provided with the operating system or available as a separate download.

- **Program loading and execution**:

Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either higher-level languages or machine language are needed as well.

- **Communications**:

These programs provide the mechanism for creating virtual connections among processes, users, and computer systems. They allow users to send messages to one another's screens, to browse web pages, to send e-mail messages, to log in remotely, or to transfer files from one machine to another.

- **Background services**:

All general-purpose systems have methods for launching certain system-program processes at boot time. Some of these processes terminate after completing their tasks, while others continue to run until the system is halted. Constantly running system-program processes are known as services, subsystems, or daemons

OPERATING SYSTEM DESIGN AND IMPLEMENTATION

DESIGN GOALS

The first problem in designing a system is to define goals and specifications. At the highest level, the design of the system will be affected by the choice of hardware and the type of system: traditional desktop/laptop, mobile, distributed, or real time. Beyond this highest design level, the requirements may be much harder to specify.

The requirements can, however, be divided into two basic groups:

User goals and **system goals**.

- User goals

Convenience and efficiency , Easy to learn , Reliable , Safe and Fast

- System goals

Easy to design, implement, and maintain , Flexible, reliable, error-free, and efficient

Mechanisms and Policies

Mechanisms determine ***how*** to do something; policies determine ***what*** will be done.

For example, the timer construct is a mechanism for ensuring CPU protection, but deciding how long the timer is to be set for a particular user is a policy decision.

- Early OS in assembly language, Now C, C++
- Using emulators of the target hardware, particularly if the real hardware is unavailable (e.g. not built yet), or not a suitable platform for development, (e.g. smart phones, game consoles, or other similar devices.)
- Android

Library : C, C++

Application Frameworks : JAVA

OPERATING SYSTEM STRUCTURE

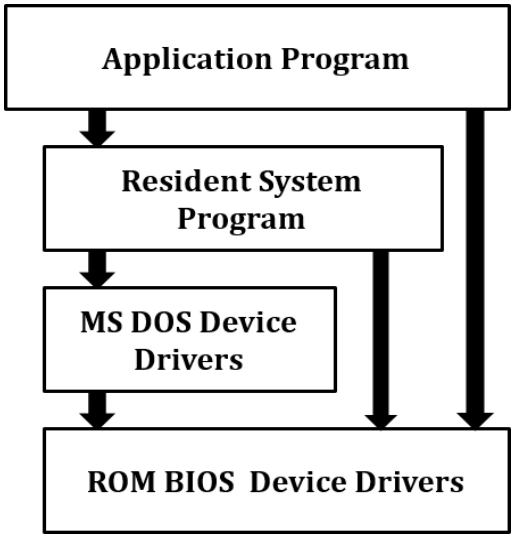
Operating system can be implemented with the help of various structures. The structure of the OS depends mainly on how the various common components of the operating system are interconnected and melded into the kernel.

Depending on this we have following structures of the operating system:

- Monolithic Structure
- Layered Approach
- Microkernels
- Modules
- Hybrid Systems
  - macOS and iOS
  - Android

❖ **Monolithic structure:**

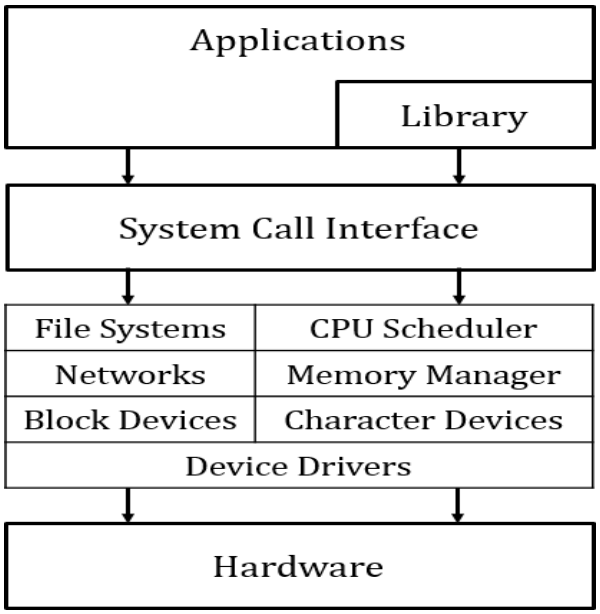
Such operating systems do not have well defined structure and are small, simple and limited systems. The interfaces and levels of functionality are not well separated. MS-DOS is an example of such operating system. In MS-DOS application programs are able to access the basic I/O routines. These types of operating system cause the entire system to crash if one of the user programs fails. Diagram of the structure of MS-DOS is shown below.



An example of such limited structuring is the original UNIX operating system, which consists of two separable parts: the kernel and the system programs. The kernel is further separated into a series of interfaces and device drivers, which have been added and expanded over the years as UNIX has evolved. Everything below the system-call interface and above the physical hardware is the kernel. The kernel provides the file system, CPU scheduling, memory management, and other operating system functions through system calls. Taken in sum, that is an enormous amount of functionality to be combined into one single address space. UNIX Structure is shown below

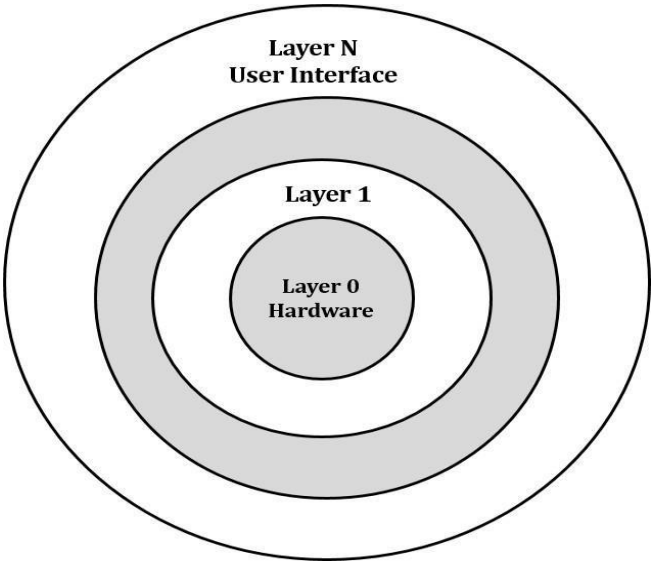
Users		
Shells and Commands Compilers and Interpreters System Libraries		
System Call Interface to the Kernel		
Signals handling	File Management	Memory Management
Kernel Interface to Hardware		
Terminal Controllers	Device Controllers	Memory Controllers

The Linux operating system is based on UNIX shown in the figure below. Applications typically use the glibc standard C library when communicating with the system call interface to the kernel. The Linux kernel is monolithic in that it runs entirely in kernel mode in a single address space, it does have a modular design that allows the kernel to be modified during run time.



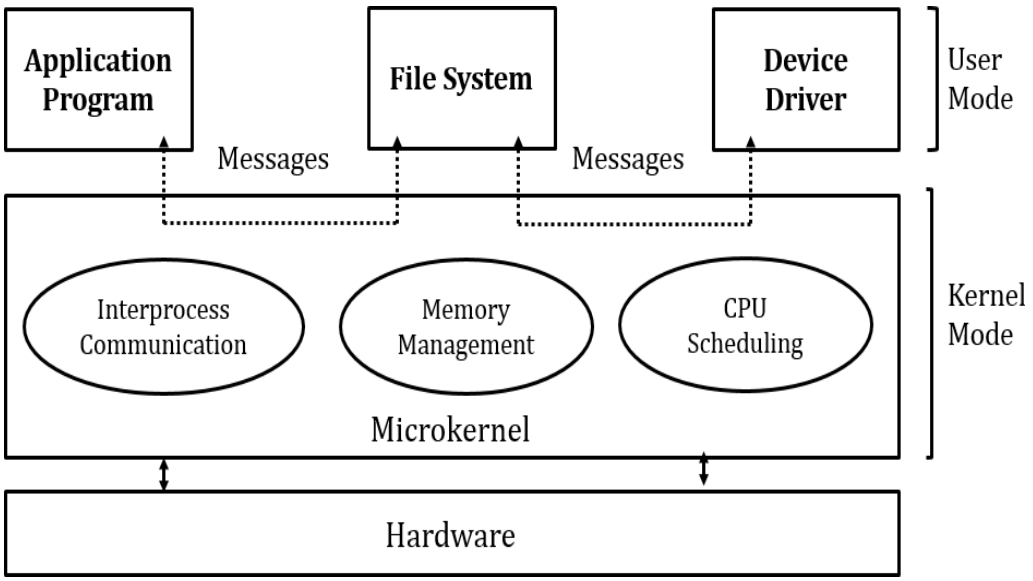
❖ Layered Approach

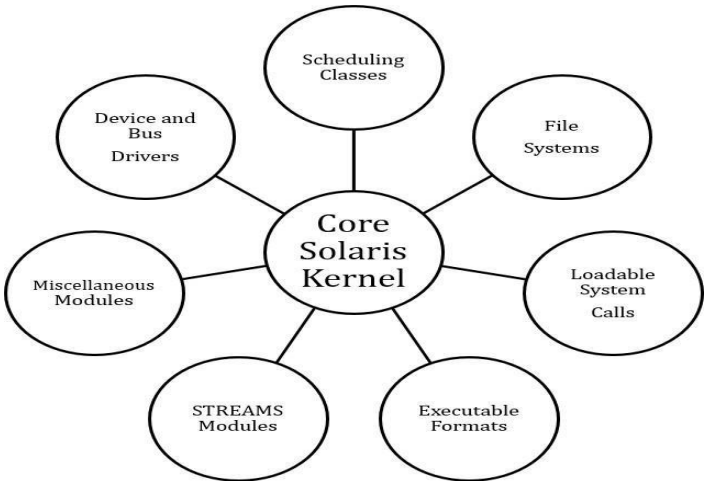
An OS can be broken into pieces and retain much more control on system. In this structure the OS is broken into number of layers (levels). The bottom layer (layer 0) is the hardware and the topmost layer (layer N) is the user interface. These layers are so designed that each layer uses the functions of the lower level layers only. This simplifies the debugging process as if lower level layers are debugged and an error occurs during debugging then the error must be on that layer only as the lower level layers have already been debugged. The main disadvantage of this structure is that at each layer, the data needs to be modified and passed on which adds overhead to the system. Moreover careful planning of the layers is necessary as a layer can use only lower level layers. UNIX is an example of this structure.



❖ **Micro-kernel:**

This structure designs the operating system by removing all non-essential components from the kernel and implementing them as system and user programs. This result in a smaller kernel called the micro-kernel.





Advantages of this structure are that all new services need to be added to user space and does not require the kernel to be modified. Thus it is more secure and reliable as if a service fails then rest of the operating system remains untouched. Mac OS is an example of this type of OS.

❖ **Modular structure or approach:**

The best approach for an OS. It involves designing of a modular kernel. The kernel has only set of core components and other services are added as dynamically loadable modules to the kernel either during run time or boot time. It resembles layered structure due to the fact that each kernel has defined and protected interfaces but it is more flexible than the layered structure as a module can call any other module. For example Solaris OS is organized as shown in the figure.

❖ **Hybrid Systems**

The Apple macOS operating system and the two mobile operating systems—iOS and Android.

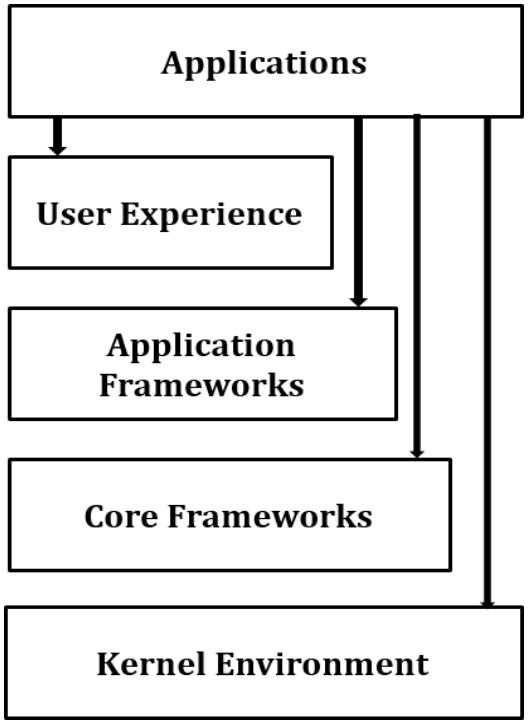
macOS and iOS

Apple's macOS operating system is designed to run primarily on desktop and laptop computer systems, whereas iOS is a mobile operating system designed for the iPhone smartphone and iPad tablet computer. Highlights of the various layers include the following:

- **User experience layer**. This layer defines the software interface that allows users to interact with the computing devices. macOS uses the ***Aqua*** user interface, which is designed for a mouse or trackpad, whereas iOS uses the ***Springboard*** user interface, which is designed for touch devices.

- **Application frameworks layer**. This layer includes the ***Cocoa*** and ***Cocoa Touch*** frameworks, which provide an API for the Objective-C and Swift programming languages. The primary difference between Cocoa and Cocoa Touch is that the former is used for developing macOS applications, and the latter by iOS to provide support for hardware features unique to mobile devices, such as touch screens.
- **Core frameworks**. This layer defines frameworks that support graphics and media including, Quicktime and OpenGL.

- **Kernel environment**. This environment, also known as **Darwin**, includes the Mach microkernel and the BSD UNIX kernel.



Applications can be designed to take advantage of user-experience features or to bypass them and interact directly with either the application framework or the core framework. Additionally, an application can forego frameworks entirely and communicate directly with the kernel environment.

Some significant distinctions between macOS and iOS include the following:

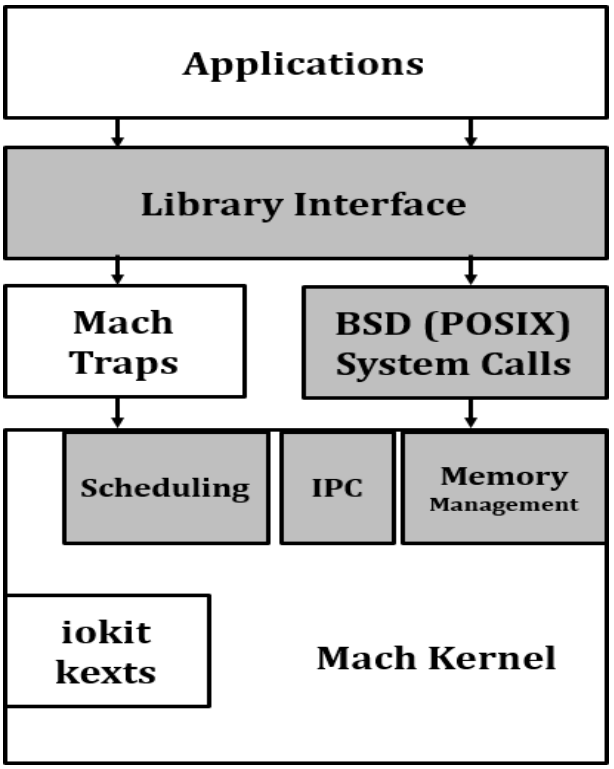
- Because macOS is intended for desktop and laptop computer systems, it is compiled to run on Intel architectures. iOS is designed for mobile devices and thus is compiled for ARM-based architectures. Similarly, the iOS kernel has been modified somewhat to address specific features and needs of mobile systems, such as power management and aggressive memory management. Additionally, iOS has more stringent security settings than macOS.
- The iOS operating system is generally much more restricted to developers than macOS and may even be closed to developers. For example, iOS restricts access to POSIX and BSD APIs on iOS, whereas they are openly available to developers on macOS.

Darwin OS

Darwin OS is a layered system that consists primarily of the Mach microkernel and the BSD UNIX kernel. Darwin's structure is shown below

Darwin provides ***two*** system-call interfaces: Mach system calls (known as **traps**) and BSD system calls (which provide POSIX functionality). The interface to these system calls is a

rich set of libraries that includes not only the standard C library but also libraries that provide networking, security, and programming language support.



Beneath the system-call interface, Mach provides fundamental operating system services, including memory management, CPU scheduling, and inter process communication (IPC) facilities such as message passing and remote procedure calls (RPCs). Much of the functionality provided by Mach is available through **kernel abstractions**, which include tasks (a Mach process), threads, memory objects, and ports (used for IPC). As an example, an application may create a new process using the BSD POSIX fork() system call. Mach will, in turn, use a task kernel abstraction to represent the process in the kernel.

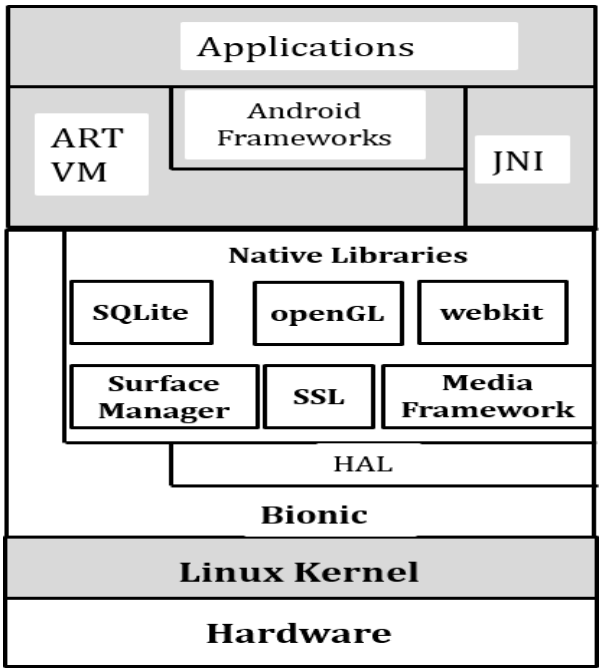
In addition to Mach and BSD, the kernel environment provides an I/O kit for development of device drivers and dynamically loadable modules (which macOS refers to as **kernel extensions**, or **kexts**).

Android

Developed for Android smartphones and tablet computers. Whereas iOS is designed to run on Apple mobile devices and is close-sourced, Android runs on a variety of mobile platforms and is open sourced, partly explaining its rapid rise in popularity. Android is similar to iOS in that it is a layered stack of software that provides a rich set of frameworks supporting graphics, audio, and hardware features. These features, in turn, provide a platform for developing mobile applications that run on a multitude of Android-enabled devices.

Software designers for Android devices develop applications in the Java language, but they do not generally use the standard Java API. Google has designed a separate Android API for Java development. Java applications are compiled into a form that can execute on the Android RunTime ART, a virtual machine designed for Android and optimized for mobile devices with limited memory and CPU processing capabilities. Java programs are first compiled to a Java bytecode .class file and then translated into an executable .dex file. Whereas many Java virtual machines perform just-in-time (JIT) compilation to improve application efficiency, ART performs **ahead-of- time** (**AOT**) compilation

The structure of Android appears is shown below



.dex files are compiled into native machine code when they are installed on a device, from which they can execute on the ART. AOT compilation allows more efficient application execution as well as reduced power consumption, features that are crucial for mobile systems.

Programs written using Java native interface JNI are generally not portable from one hardware device to another. The set of native libraries available for Android applications includes frameworks for developing web browsers (webkit), database support (SQLite), and network support, such as secure sockets (SSLs). Android can run on an almost unlimited number of hardware devices, Google has chosen to abstract the physical hardware through the hardware abstraction layer, or HAL. By abstracting all hardware, such as the camera, GPS chip, and other sensors, the HAL provides applications with a consistent view independent of specific hardware. This feature, of course, allows developers to write programs that are portable across different hardware platforms.

The standard C library used by Linux systems is the GNU C library (glibc). Google instead developed the **Bionic** standard C library for Android. Not only does Bionic have a smaller memory footprint than glibc, but it also has been designed for the slower CPUs that characterize mobile devices. At the bottom of Android's software stack is the Linux kernel. Google has modified the Linux kernel used in Android in a variety of areas to support the special needs of mobile systems, such as power management. It has also made changes in memory management and allocation.

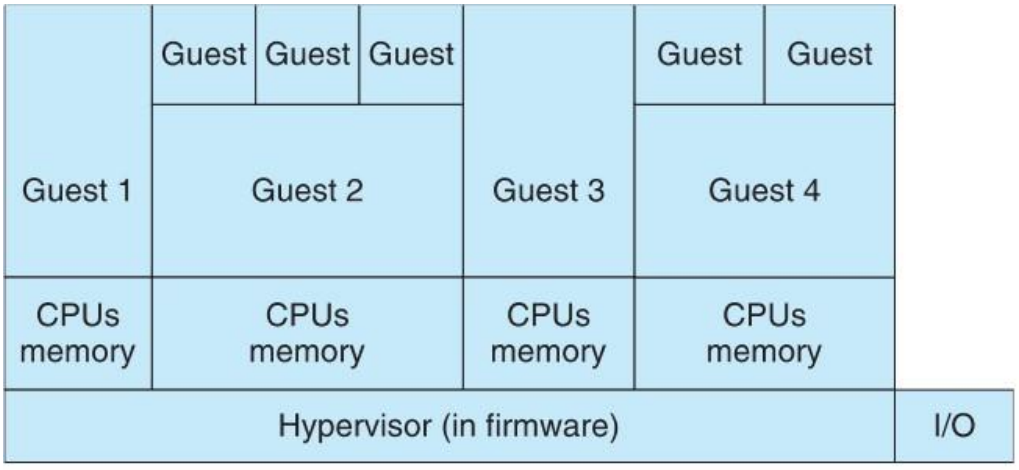
Need of virtual machines - OS design consideration for multiprocessor and multi core

1.1. Definition & Basic Concept

Virtual Machine (VM): A Virtual Machine is a software-based emulation of a physical computer. It allows a guest operating system (like Windows or Linux) to run in an isolated environment on top of the host's actual hardware. VMs use a hypervisor to manage and allocate system resources like CPU, memory, and storage. This enables multiple VMs to run on a single physical machine, each functioning as if it were a separate computer, with its own OS and applications, offering flexibility, isolation, and efficient resource utilization.

Hypervisor / Virtual Machine Monitor (VMM): A hypervisor, also known as a Virtual Machine Monitor (VMM), is a software layer that enables the creation, management, and operation of virtual machines (VMs). It allows multiple operating systems to run simultaneously on a single physical machine by sharing hardware resources like CPU, memory, and storage. Hypervisors ensure isolation between VMs and efficiently allocate resources to each one.

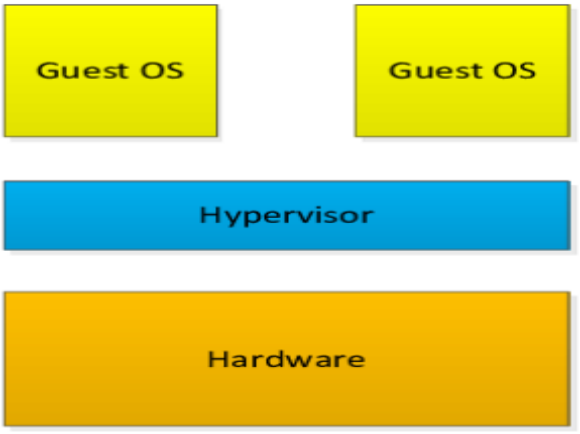
1. **Type 0 Hypervisors (Hardware-Based Hypervisors):•** Type 0 hypervisors are firmware-based solutions that provide virtualization support directly at the hardware level, eliminating the need for a software-based hypervisor.



Type 0 Hypervisor

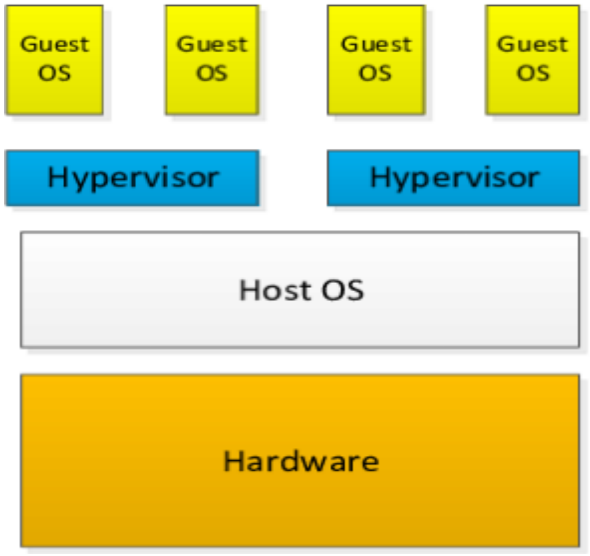
2. **Type 1 (Bare-metal)**: Type 1 hypervisors run directly on the physical hardware without relying on a host operating system. They manage the hardware and allocate resources directly to the virtual machines. Because there is no intermediate OS layer, they offer

better performance, stability, and are often used in enterprise environments. Examples include VMware ESXi, Xen, and Microsoft Hyper-V.



Type 1 (bare-metal)

3. **Type 2 Hypervisors (Hosted Hypervisors)**Type 2 hypervisors operate as software applications on top of an existing operating system. They are easier to install and use, making them suitable for personal or development use. However, they depend on the host OS for hardware access, which can lead to reduced performance compared to Type 1. Examples include VirtualBox and VMware Workstation.



1.2. Types of Virtualization

- **Full Virtualization**: Host emulates all hardware; guest OS runs unmodified; uses traps/emulation or hardware extensions.
- **Paravirtualization**: Guest OS is aware of running under a hypervisor; uses optimized interfaces (e.g., virtio drivers) to reduce overhead.

- **Hardware-Assisted Virtualization**: CPU features (Intel VT-x, AMD-V) facilitate trapping privileged instructions and managing nested page tables, improving performance.
- **Container vs. VM** (for context):
  - Containers share the host kernel; lightweight isolation.
  - VMs include a full OS kernel; stronger isolation at cost of more overhead.

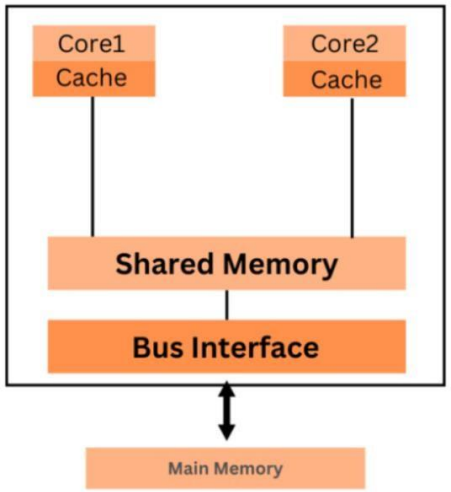
1.3. Advantages of VMs.

- **Resource Consolidation**: Run multiple OS instances on one physical machine—better CPU/memory/disk utilization.
- **Isolation & Security**: Faults, crashes, or compromises in one VM do not affect others or the host; useful for sandboxing, multi-tenant environments.
- **Portability & Encapsulation**: VM image packages OS + applications + configuration; easy to migrate, clone, or restore.
- **Testing / Development / Legacy Support**: Quickly spin up various OS versions; run legacy OS/software without dedicating old hardware.
- **Cost Efficiency**: Fewer physical servers, reduced energy and maintenance costs; cloud providers leverage virtualization to offer on-demand resources.
- **Snapshots & Rollback**: Save VM states, experiment, then revert if needed—valuable in development, testing, and security analysis.
- **High Availability & Live Migration**: Move running VMs between hosts with minimal downtime; enables maintenance and load balancing in data centers.
- **Overcommitment & Elasticity**: Dynamically adjust allocated CPU/memory; supports variable workloads in cloud settings.

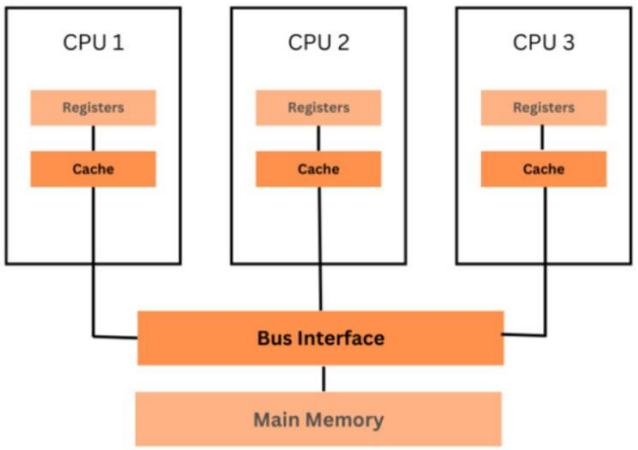
OS Design Consideration for Multiprocessor and Multi-core Systems

Multiprocessor and multicore systems refer to computer architectures that contain **more than one processing unit**, which allows for improved performance, multitasking, and parallel execution of programs.

- A **Multiprocessor system** consists of **multiple physical CPUs** (Central Processing Units) connected to a shared memory and working together. Each CPU operates independently but can also coordinate tasks with other CPUs. These systems are commonly found in servers, high-performance computing environments, and enterprise-grade machines, offering increased reliability and speed for demanding tasks.



- A **Multicore system**, on the other hand, has a **single physical CPU chip** that contains **multiple processing cores**. Each core can execute its own thread or process, allowing true parallelism even on personal computers and mobile devices. Since the cores are integrated on a single chip, communication between them is faster and more power-efficient than in multiprocessor systems.



Challenges and Considerations

1. **Concurrency:**
   - The OS must manage **multiple threads and processes** running simultaneously.
   - Requires **advanced scheduling algorithms** for efficient CPU utilization.
2. **Synchronization:**
   - Shared memory access must be synchronized to prevent **race conditions**.
   - Mechanisms like **semaphores, mutexes, spinlocks** are necessary.
3. **Scalability:**
   - OS should scale efficiently as the number of processors or cores increases.
   - Algorithms need to avoid bottlenecks.
4. **Load Balancing:**

- The OS must **distribute workload evenly** among all processors.
- Avoids overloading one processor while others remain idle.

5. **Cache Coherence:**
   - Each processor/core may have its own cache; changes in one cache must be visible to others.
   - OS must support or work with hardware to maintain **cache coherence**.
6. **Thread and Process Scheduling:**
   - Scheduling policies must be aware of **processor affinity** (binding threads to specific cores for performance).
   - OS needs to choose between **symmetric multiprocessing (SMP)** or **asymmetric multiprocessing (AMP)**.
7. **Memory Management:**
   - Must handle shared memory access, possibly with **NUMA (Non-Uniform Memory Access)** support.
   - Needs to ensure efficient and conflict-free access.
8. **Interrupt Handling:**
   - OS must route interrupts to the appropriate processor/core.
   - Requires **Interrupt Steering/Redirection** mechanisms.
9. **Power Management:**
   - Multicore systems consume more power; OS must manage **dynamic voltage and frequency scaling (DVFS)**.
10. **Virtualization Support:**

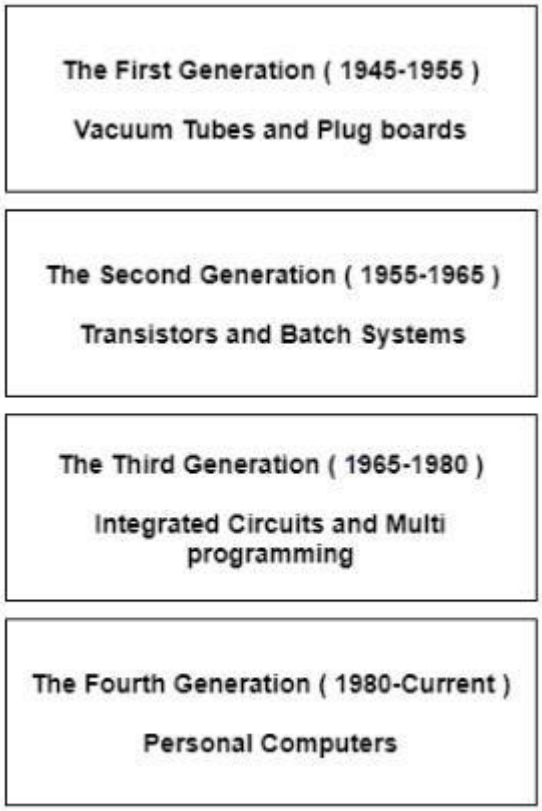
- Modern OSes must be designed to support **hardware-assisted virtualization** (e.g., Intel VT, AMD-V).
- Facilitates efficient execution of VMs.

MultiCore	**MultiProcessor**
A single CPU or processor with two or more independent processing units called cores that are capable of reading and executing program instructions.	A system with two or more CPU's that allows simultaneous processing of programs.
It executes single program faster.	It executes multiple programs Faster.

MultiCore	MultiProcessor
Not as reliable as multiprocessor.	More reliable since failure in one CPU will not affect other.
It has less traffic.	It has more traffic.
It does not need to be configured.	It needs little complex configuration.
It's very cheaper (single CPU that does not require multiple CPU support system).	It is Expensive (Multiple separate CPU's that require system that supports multiple processors) as compared to MultiCore.

OPERATING SYSTEM GENERATIONS

Operating Systems have evolved over the years. So, their evolution through the years can be mapped using generations of operating systems. There are four generations of operating systems.



OPERATING SYSTEM GENERATIONS

The First Generation (1945 - 1955): Vacuum Tubes and Plugboards

Digital computers were not constructed until the second world war. Calculating engines with mechanical relays were built at that time. However, the mechanical relays were very slow and were later replaced with vacuum tubes. These machines were enormous but were still very slow.

These early computers were designed, built and maintained by a single group of people. Programming languages were unknown and there were no operating systems so all the programming was done in machine language. All the problems were simple numerical calculations.

By the 1950's punch cards were introduced and this improved the computer system. Instead of using plugboards, programs were written on cards and read into the system.

Second Generation (1955 - 1965): Transistors and Batch Systems

Transistors led to the development of the computer systems that could be manufactured and sold to paying customers. These machines were known as mainframes and were locked in air-conditioned computer rooms with staff to operate them.

The **Batch System** was introduced to reduce the wasted time in the computer. A tray full of jobs was collected in the input room and read into the **magnetic tape**. After that, the tape was rewound and mounted on a tape drive. Then the batch operating system was loaded in which read the first job from the tape and ran it. The output was written on the second tape. After the whole batch was done, the input and output tapes were removed and the output tape was printed.

The Third Generation (1965 - 1980): Integrated Circuits and Multiprogramming

Until the 1960's, there were two types of computer systems i.e., the scientific and the commercial computers. These were combined by IBM in the System/360. This used integrated circuits and provided a major price and performance advantage over the second generation systems.

The third generation operating systems also introduced **multiprogramming**. This meant that the processor was not idle while a job was completing its I/O operation. Another job was scheduled on the processor so that its time would not be wasted.

The Fourth Generation (1980 - Present): Personal Computers

Personal Computers were easy to create with the development of large-scale integrated circuits. These were chips containing thousands of transistors on a square centimeter of silicon. Because of these, microcomputers were much cheaper than minicomputers and that made it possible for a single individual to own one of them.

The advent of personal computers also led to the growth of networks. This created network operating systems and distributed operating systems. The users were aware of a network while using a network operating system and could log in to remote machines and copy files from one machine to another.

SYSTEM BOOT

When a computer or any other computing device is in a powerless state, its operating system remains stored in secondary storage like a hard disk or SSD. But, when the computer is started, the operating system must be present in the main memory or RAM of the system.

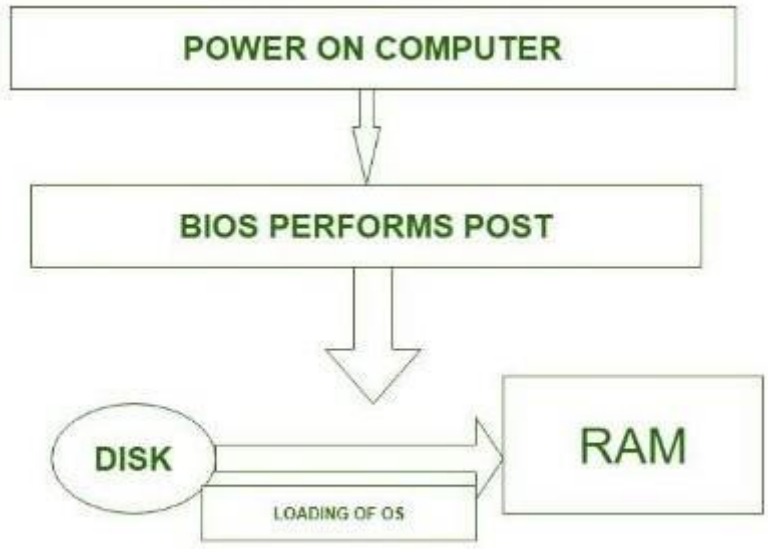
When a computer system is started, there is a mechanism in the system that loads the operating system from the secondary storage into the main memory, or RAM, of the system. This is called the **booting process** of the system.

Process of Booting

After an operating system is generated, it must be available for use by the hardware. But how does the hardware know where the kernel is or how to load that kernel? The procedure of starting a computer by loading the kernel is known as **booting** the system. Hence, it needs a special program, stored in the ROM to do this job known as the Bootstrap loader. Example: BIOS (boot input-output system). A modern PC BIOS (Basic Input/Output System) supports booting from various devices. Typically, the BIOS will allow the user to configure a boot order. If the boot order is set to:

- CD Drive
- Hard Disk Drive
- Network

Then the BIOS will try to boot from the CD drive first, and if that fails, it will try to boot from the hard disk drive, and if that fails then it will try to boot from the network, and if that fails, it won't boot at all. Booting is a startup sequence that starts the operating system of a computer when it is turned on. A boot sequence is the initial set of operations that the computer performs when it is switched on. Every computer has a boot sequence. The Bootstrap loader locates the kernel, loads it into main memory, and starts its execution. In some systems, a simple bootstrap loader fetches a more complex boot program from disk, which in turn loads the kernel.



Process of Booting

Types of Booting

There are two **types of booting** depending on the number of operating systems installed on the machine/computer, i.e.

1. Cold or Hard Booting

A state in which a computer is switched on from being switched off is referred to as cold booting. Powering on a computer that has been turned off completely is usually called a cold boot. In this procedure, the system undergoes a complete power-on self-test (POST) that initializes hardware devices and loads operating systems from a storage medium into random-access memory (RAM).

2. Soft or Warm Booting

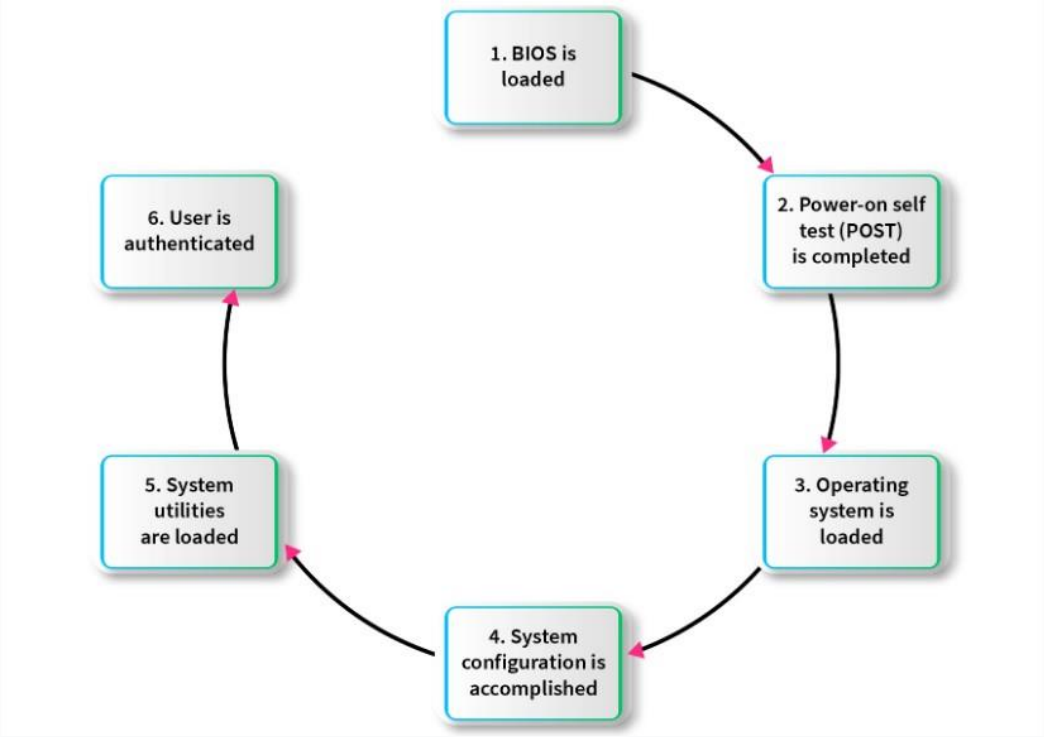
Soft boot or restart method Warm Booting, also called soft boots or restarts, reboots a computer system without shutting it down entirely. This technique is usually started by an operating system restart command or by pressing an appropriate key combination. Warm reboots do skip some of the hardware initialization processes that are done on cold booting since the hardware components have been on power and have been initialized earlier. In operation of a computer system, both cold boot and warm boot processes are absolutely necessary, where the cold boot yields total system initialization whereas the warm boot allows a quicker restart choice that does not really involve the entire start up sequence.

Booting in Operating System

"Booting" might sound like a peculiar term if you're new to computing, but it's a foundational process that takes place every time you switch on your computer. The simple question "what is booting?" can be explained as the process by which a computer starts up, initializing its hardware components and launching its operating system (OS) from the computer's storage into its working memory.

Steps In the Booting Process

When the computer is powered on, all the hardware components receive the power and they get initialized. After that, the computer system goes through 6 steps booting process as follows:



1. **Loading of BIOS:** The small set of instructions present in the ROM is loaded into the computer memory and the CPU executes those instructions.
2. **Power-On Self Test (POST):** In order to check the operability of all the hardware connected to our computer system, BIOS carries out POST which will check the hardware components and if any problem is found user is alerted with POST beeps and POST screen messaged.
3. **Loading of Operating System:**
    - After the successful completion of POST, the bootable sequence present in CMOS (Common Metal Oxide Semiconductor) is read by BIOS.
    - Based on the bootable sequence it will search for Master Boot Record (MBR) in bootable devices like floppy disk, CD-ROM, and hard disk.
    - If MBR is not found in any of them, the system will halt by displaying "No Boot Device Found".
    - if MBR is found, the BIOS will load the special application program called Boot Loader, which will eventually load the Operating system.
4. **System Configuration is Accomplished:** After the OS is loaded, device drivers are loaded into the memory so that our devices can function correctly.
5. **System Utilities are Loaded:** System utilities like antivirus, volume control, etc. are loaded into the memory in this step
6. **User Authentication:** If any user authentication is being set, the system will ask the user to enter the credentials, and on receiving the correct credentials the computer system will run GUI shell (in most cases) or CLI shell.

As BIOS is lightweight, it will just load the Boot Loader which can load the Complex set of libraries required for loading the Operating System. BIOS can't directly load the heavily weighted set of instructions responsible for loading the Operating System.

Dual Booting

When two operating systems are installed on a computer system, it is called dual booting. In fact, multiple operating systems can be installed on such a system. But how does the system know which operating system to boot? A boot loader that understands multiple file systems and multiple operating systems can occupy the boot space. Once loaded, it can boot one of the operating systems available on the disk. The disk can have multiple partitions, each containing a different type of operating system. When a computer system turns on, a boot manager program displays a menu, allowing the user to choose the operating system to use.

Power on Self Test (POST) Booting

Power on Self Test booting is a part of the booting cycle in a computer system. The POST is the very first diagnostic routine that the installed hardware components undergo every time you power up your computer to assure the presence and functionality of the devices. The POST tests the status of many hardware components, including the CPU, memory, storage devices, and other peripherals. It watches for problems that may prevent booting. In case of a malfunction, the POST usually displays some error message or beeps in a pattern that indicates where the problem lies. If it succeeds, the computer starts loading the operating system and other necessary software for normal running.

Master Boot Record (MBR)

A piece that is very important in the boot process of a computer is called the Master Boot Record (MBR). This thing is located at the very beginning on the hard disk, and it has

critical details for starting up. It is composed by division tables among other parts for different types of partitions used on disks with their respective filesystems being identified here too. During system startup sequence or POST (Power On Self Test), firmware like BIOS (Basic Input Output System) /UEFI (Unified Extensible Firmware Interface) tries looking for MBR from storage device used during boot up process (boot device) before running its contents. The bootloader is loaded by this code, and subsequently, the operating system is loaded by the bootloader. MBR is an essential cog in booting procedure wherein it starts off steps that culminate in the system booting appropriately.