Unit I: Introduction to Operating System – Operating System

Introduction of Operating System:

Computer Software can roughly be divided into two types:

a. **Application Software:** Which perform the actual work the user wants.
b. **System Software:** Which manage the operation of the computer itself.

The most fundamental system program is the operating system, whose job is to control all the computer's resources and provide a base upon which the application program can be written.

Operating system acts as an intermediary between a user of a computer and the computer hardware. A computer system can be divided roughly into four components: *the hardware, the operating system, the application program, and the users* as shown in the figure below:

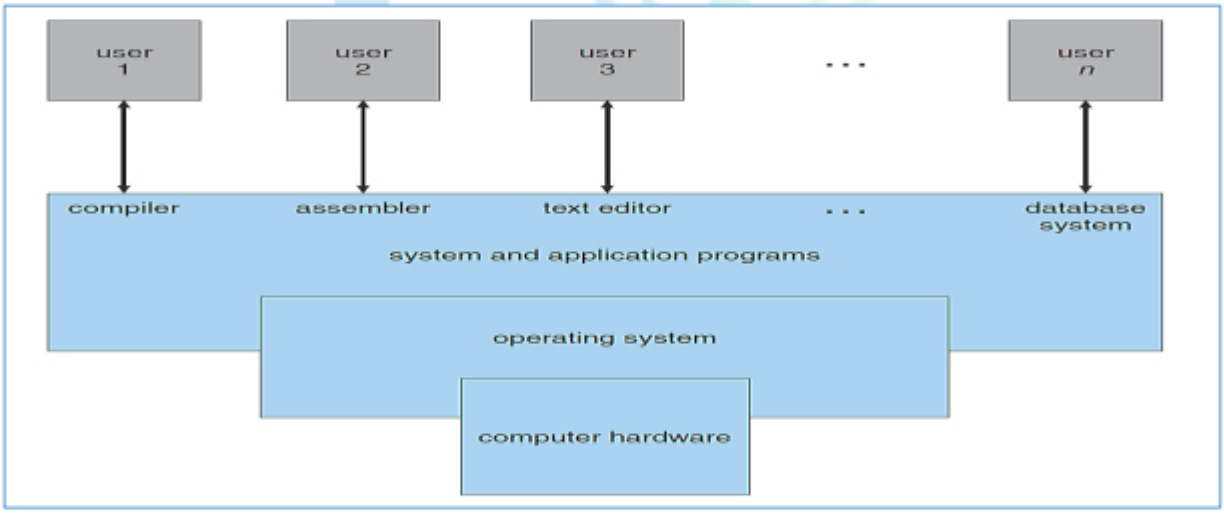


Figure: Abstract View of the Components of a Computer System.

An operating system is similar to a **government.** Like a government it performs no useful function by itself. It simply provides an environment within which other programs can do useful work.

History of Operating System:

Historically operating systems have been tightly related to the computer architecture, it is good idea to study the history of operating systems from the architecture of the computers on which they run.

Operating systems have evolved through a number of distinct phases or generations which corresponds roughly to the decades.

The 1940's - First Generation:

The earliest electronic digital computers had no operating systems. Machines of this 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 1950's - Second Generation:

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 1960's - Third Generation:

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.

Note that spooling technique is much like thread being spun to a spool so that it may be later be unwound as needed.

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 multi-program large number of simultaneous interactive users.

The 1970's-Fourth Generation:

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 become 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.

Objectives of Operating System:

1. Operating System as an Extended Machine or Virtual:

The operating system masks or hides the details of the Hardware form the programmers and general users and provides a convenient interface for using the system.

The program that hides the truth about the hardware from the user and presents a nice simple view of named files that can be read and written is of course the operating system.

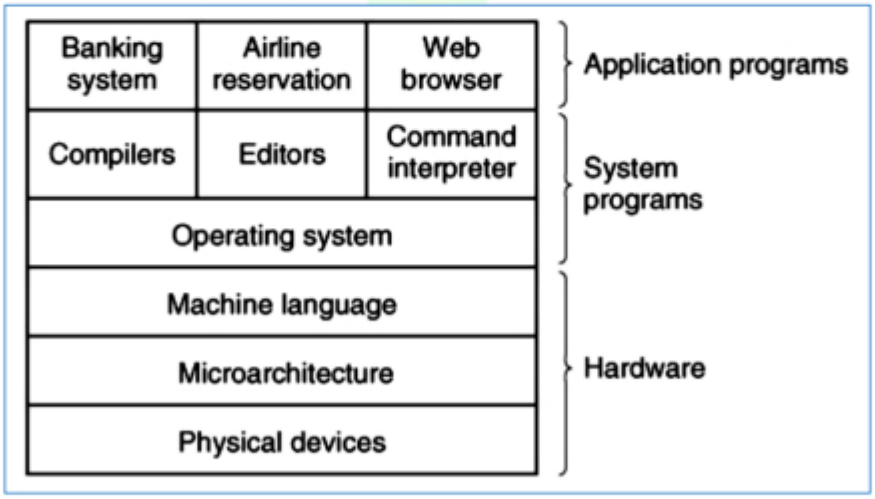


Figure: Computer System Consists Of Hardware, System Program and Application Program

In this view the function of OS is to present the user with the equivalent of an extended machine or virtual machine that is easier to program than underlying hardware.

Just as the operating system shields the user from the disk hardware and presents a simple file-oriented interface, it also conceals a lot of unpleasant business concerning interrupts, timers, memory management and other low level features.

The placement of OS is as shown in figure below. A major function of OS is to hide all the complexity presented by the underlying hardware and gives the programmer a more convenient set of instructions to work with.

2. Operating System as a Resource Manager:

A computer system has many resources. Modern computers consist of processors, memories, timers, disks, mice, network interfaces, printers, and a wide variety of other devices.

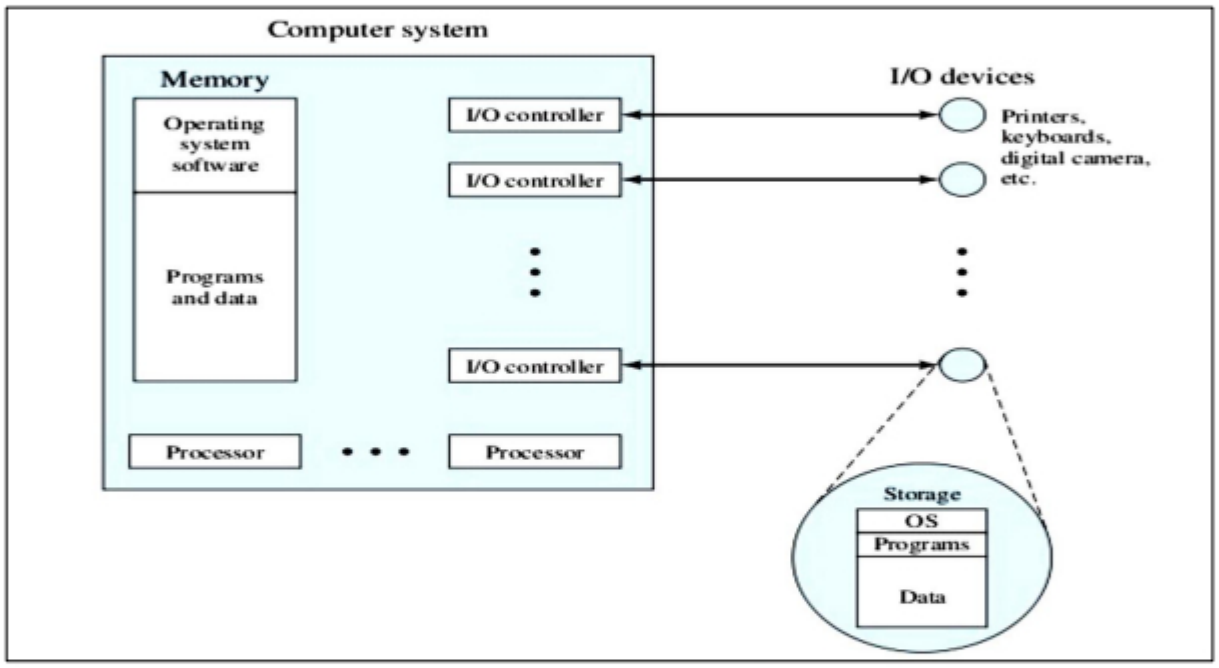


Figure: The Operating system as Resource manger

In the alternative view, the job of the operating system is to provide for an orderly and controlled allocation of the processors, memories, and I/O devices among the various programs competing for them.

Imagine what would happen if three programs running on some computer all tried to print their output simultaneously on the same printer. The first few lines of printout might be from program 1, the next few from program 2, then some from program 3, and so forth. The result would be chaos.

The operating system can bring order to the potential chaos by buffering all the output destined for the printer on the disk.

When one program is finished, the operating system can then copy its output from the disk file where it has been stored to the printer, while at the same time the other program can continue generating more output, oblivious to the fact that the output is not really going to the printer (yet).

Above figure suggests the main resources that are managed by the OS. A portion of the OS is in main memory. This includes Kernel or nucleus. The remainder of main memory contains user programs and data. The allocation of this resource (main memory) is controlled jointly by the OS and memory management hardware in the processor

Types of Operating Systems:

1. Serial Processing:

The Serial Processing Operating Systems are those which Performs all the instructions into a Sequence Manner or the Instructions those are given by the user will be executed by using the FIFO Manner means First in First Out.

All the Instructions those are Entered First in the System will be Executed First and the Instructions those are Entered Later Will be Executed Later. For Running the Instructions the Program Counter is used which is used for Executing all the Instructions.

In this the Program Counter will determines which instruction is going to Execute and the which instruction will be Execute after this. Mainly the Punch Cards are used for this.

In this all the Jobs are firstly Prepared and Stored on the Card and after that card will be entered in the System and after that all the Instructions will be executed one by One.

These System Presented Two Major Problems:

a. Scheduling:

Used sign-up sheet to reserve machine time. A user may sign up for an hour but finishes his job in 45 minutes. This would result in wasted computer idle time, also the user might run into the problem not finish his job in allotted time.

b. Set Up Time:

A single program involves:

- Loading compiler and source program in memory
- Saving the compiled program (object code)
- Loading and linking together object program and common function

Each of these steps involves the mounting or dismounting tapes on setting up punch cards. If an error occur user had to go the beginning of the set up sequence. Thus, a considerable amount of time is spent in setting up the program to run.

This mode of operation is turned as serial processing, reflecting the fact that users access the computer in series.

2. Batch Processing:

The Batch Processing is same as the Serial Processing Technique. But in the Batch Processing Similar Types of jobs are Firstly Prepared and they are Stored on the Card. and that card will be Submit to the System for the Processing.

The System then Perform all the Operations on the Instructions one by one. And a user can't be Able to specify any input. And Operating System wills increments his Program Counter for Executing the Next Instruction.

The Main Problem is that the Jobs those are prepared for Execution must be the Same Type and if a job requires for any type of Input then this will not be Possible for the user. And Many Time will be wasted for Preparing the Batch.

The Batch Contains the Jobs and all those jobs will be executed without the user Intervention. And Operating System will use the LOAD and RUN Operation. This will first LOAD the Job from the Card and after that he will execute the instructions. By using the RUN Command.

The Speed of the Processing the Job will be Depend on the Jobs and the Results those are produced by the System in difference of Time which is used for giving or submit the Job and the Time which is used for Displaying the Results on the Screen.

3. Multitasking or Time Sharing System:

Multiprogramming didn't provide the user interaction with the computer system. Time sharing or Multitasking is a logical extension of Multiprogramming that provides user interaction.

There are more than one user interacting the system at the same time. The switching of CPU between two users is so fast that it gives the impression to user that he is only working on the system but actually it is shared among different users.

CPU bound is divided into different time slots depending upon the number of users using the system. Just as multiprogramming allows the processor to handle multiple batch jobs at a time, multiprogramming can also be used to handle multiple interactive jobs.

In this latter case, the technique is referred to as time sharing, because processor time is shared among multiple users.

A multitasking system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time shared computer. Each user has at least one separate program in memory.

Multitasking are more complex than multiprogramming and must provide a mechanism for jobs synchronization and communication and it may ensure that system does not go in deadlock.

Although batch processing is still in use but most of the system today available uses the concept of multitasking and Multiprogramming.

4. Multiprocessing:

Generally a Computer has a Single Processor means a Computer have a just one CPU for Processing the instructions. But if we are Running multiple jobs, then this will decrease the Speed of CPU.

For Increasing the Speed of Processing then we uses the Multiprocessing, in the Multi Processing there are two or More CPU in a Single Operating System if one CPU will fail, then other CPU is used for providing backup to the first CPU. With the help of Multi-processing, we can Execute Many Jobs at a Time.

All the Operations are divided into the Number of CPU's. if first CPU Completed his Work before the Second CPU, then the Work of Second CPU will be divided into the First and Second.

5. Multi-Programming:

As we know that in the Batch Processing System there are multiple jobs Execute by the System. The System first prepare a batch and after that he will Execute all the jobs those are Stored into the Batch.

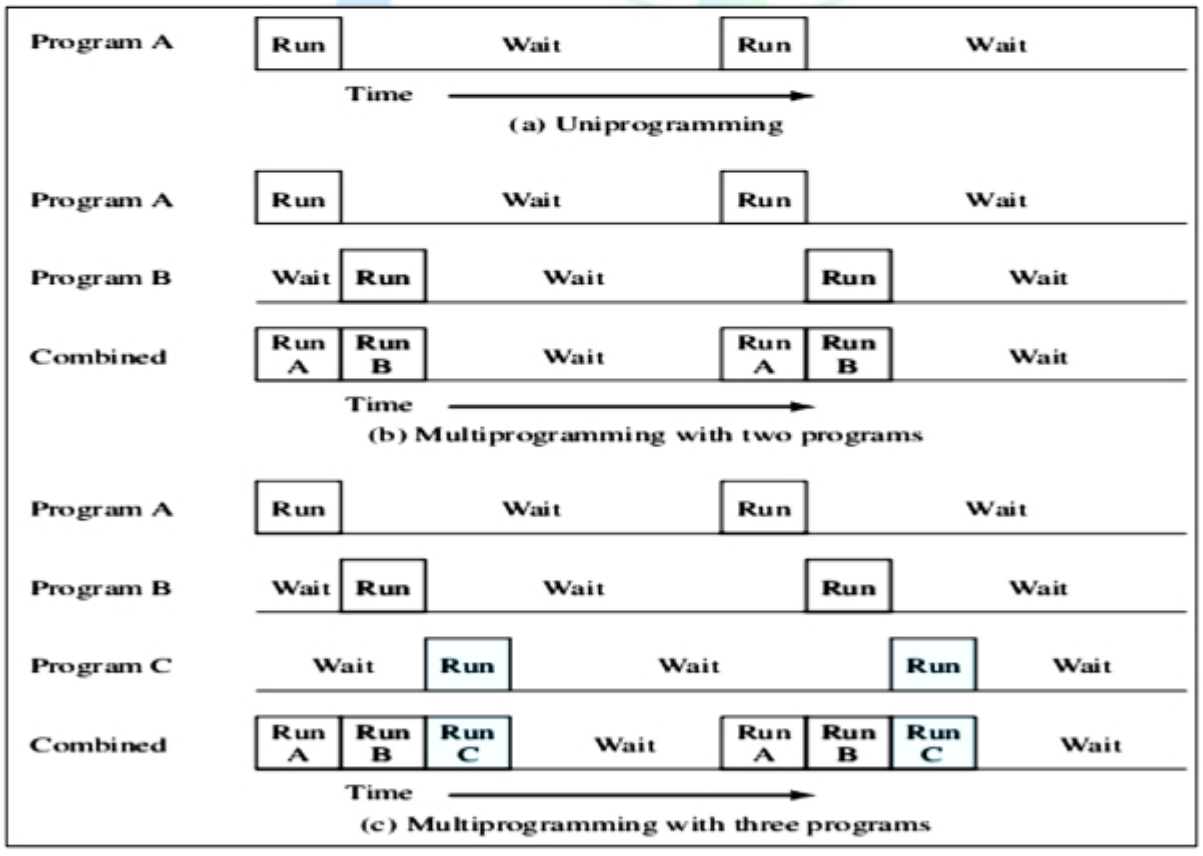


Figure: Multiprogramming

But the Main Problem is that if a process or job requires an Input and Output Operation, then it is not possible and second there will be the wastage of the Time when we are preparing the batch and the CPU will remain idle at that Time.

But With the help of Multi programming we can Execute Multiple Programs on the System at a Time. In the Multi-programming the CPU will never get idle, because with the help of Multi-Programming we can Execute Many Programs on the System.

When we are working with the Program then we can also submit the Second or Another Program for Running and the CPU will then execute the Second Program after the completion of the First Program. And in this we can also specify our Input means a user can also interact with the System.

The Multi-programming Operating Systems never use any cards because the Process is entered on the Spot by the user.

But the Operating System also uses the Process of Allocation and De-allocation of the Memory Means he will provide the Memory Space to all the Running and all the Waiting Processes. There must be the Proper Management of all the Running Jobs.

6. Distributed System:

Distributed Means Data is Stored and Processed on Multiple Locations. When a Data is stored on to the Multiple Computers, those are placed in Different Locations. Distributed means In the Network, Network Collections of Computers are connected with Each other.

Then if we want to Take Some Data from other Computer, Then we uses the Distributed Processing System. And we can also Insert and Remove the Data from out Location to another Location.

In this Data is shared between many users. And we can also Access all the Input and Output Devices are also accessed by Multiple Users.

7. Distributed Operating System:

A recent trend in computer system is to distribute computation among several processors. In contrasts to the tightly coupled system the processors do not share memory or a clock.

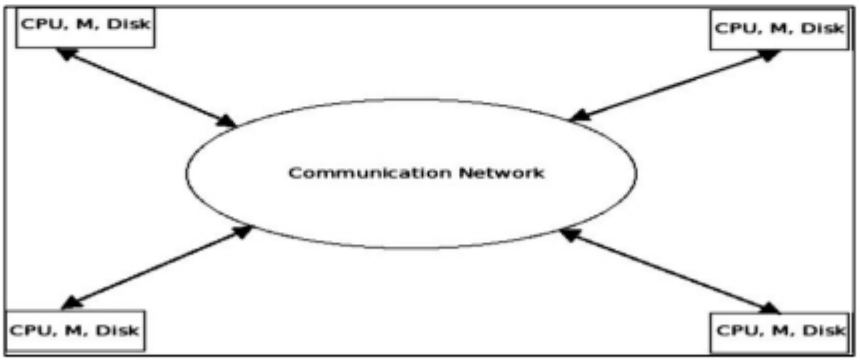


Figure: Architecture of a Distributed system.

Instead, each processor has its own local memory. The processors communicate with one another through various communication lines such as computer network.

Distributed operating system are the operating system for a distributed system (a network of autonomous computers connected by a communication network through a message passing mechanisms).

A distributed operating system controls and manages the hardware and software resources of a distributed system. When a program is executed on a distributed system, user is not aware of where the program is executed or the location of the resources accessed.

Example of Distributed OS: Amoeba, Chorus, Alpha Kernel.

8. Real Time Operating System:

Primary objective of Real Time Operating System is to provide quick response time and thus to meet a scheduling deadline. User convenience and resource utilization are secondary concern to these systems.

Real time systems has many events that must be accepted and processed in a short time or within certain deadline. Such applications include: Rocket launching, flight control, robotics, real time simulation, telephone switching equipment etc.

Real Time Systems Are Classified Into Two Categories:

a. Hard Real Time System:

In the Hard Real Time System, Time is fixed and we can't Change any Moments of the Time of Processing. Means CPU will Process the data as we Enters the Data.

b. Soft Real Time System:

In the Soft Real Time System, some Moments can be Change. Means after giving the Command to the CPU, CPU Performs the Operation after a **Microsecond**.

Functions of Operating System:

1. Memory Management:

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address. Main memory provides a fast storage that can be accessed directly by the CPU.

For a program to be executed, it must in the main memory. An Operating System does the following activities for memory management:

- Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part are not in use.
- In multiprogramming, the OS decides which process will get memory when and how much.

- Allocates the memory when a process requests it to do so.
- De-allocates the memory when a process no longer needs it or has been terminated.

2. Processor Management:

In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called **process scheduling**. An Operating System does the following activities for processor management:

- Keeps tracks of processor and status of process. The program responsible for this task is known as traffic controller.
- Allocates the processor (CPU) to a process.
- De-allocates processor when a process is no longer required.

3. Device Management:

An Operating System manages device communication via their respective drivers. It does the following activities for device management:

- Keeps tracks of all devices. Program responsible for this task is known as the I/O controller.
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way.
- De-allocates devices.

4. File Management:

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

An Operating System does the following activities for file management:

- Keeps track of information, location, uses, status etc. The collective facilities are often known as file system.
- Decides who gets the resources.
- Allocates the resources.
- De-allocates the resources.

5. Other Important Activities:

Following are some of the important activities that an Operating System performs –

- **Security**: By means of password and similar other techniques, it prevents unauthorized access to programs and data.

- **Control over system performance**: Recording delays between request for a service and response from the system.
- **Job accounting**: Keeping track of time and resources used by various jobs and users.
- **Error detecting aids**: Production of dumps, traces, error messages, and other debugging and error detecting aids.
- **Coordination between other software and users**: Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

Different Views of Operating System:

Operating System is designed both by taking user view and system view into consideration. Below is what the users and system thinks about Operating System.

User View:

1. The goal of the Operating System is to maximize the work and minimize the effort of the user.
2. Most of the systems are designed to be operated by single user, however in some systems multiple users can share resources, memory. In these cases Operating System is designed to handle available resources among multiple users and CPU efficiently.
3. Operating System must be designed by taking both usability and efficient resource utilization into view.
4. In embedded systems (Automated systems) user view is not present.
5. Operating System gives an effect to the user as if the processor is dealing only with the current task, but in background processor is dealing with several processes.

System View:

1. From the system point of view Operating System is a program involved with the hardware.
2. Operating System is allocator, which allocate memory, resources among various processes. It controls the sharing of resources among programs.
3. It prevents improper usage, error and handle deadlock conditions.
4. It is a program that runs all the time in the system in the form of Kernel.
5. It controls application programs that are not part of Kernel.

Unit II: Operating System Structure – Operating System

Introduction of Operating System Structure:

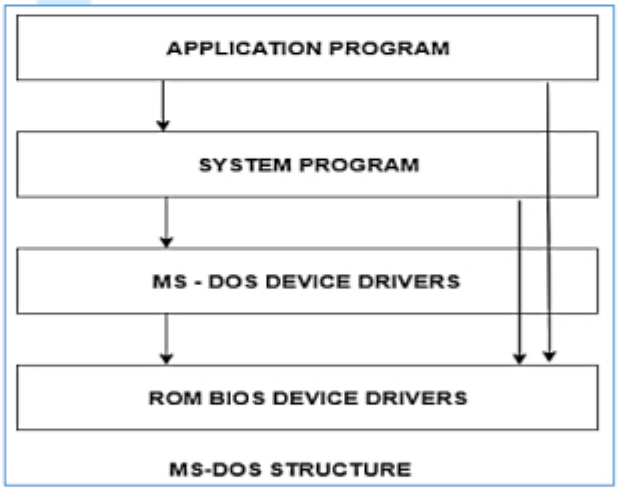
An operating system is a construct that allows the user application programs to interact with the system hardware. Since the operating system is such a complex structure, it should be created with utmost care so it can be used and modified easily.

An easy way to do this is to create the operating system in parts. Each of these parts should be well defined with clear inputs, outputs and functions.

1. Simple Structure:

There are many operating systems that have a rather simple structure. These started as small systems and rapidly expanded much further than their scope. A common example of this is MS-DOS.

It was designed simply for a niche amount for people. There was no indication that it would become so popular. An image to illustrate the structure of MS-DOS is as follows:



MS-DOS STRUCTURE

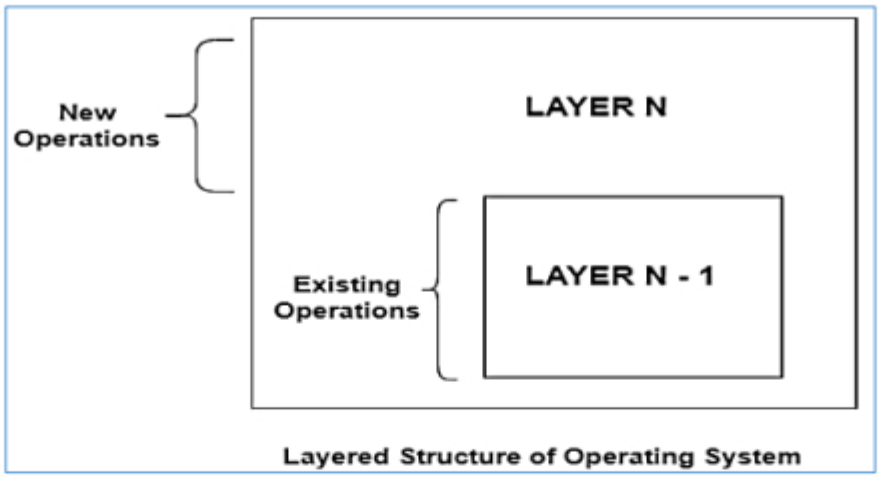
It is better that operating systems have a modular structure, unlike MS-DOS. That would lead to greater control over the computer system and its various applications.

The modular structure would also allow the programmers to hide information as required and implement internal routines as they see fit without changing the outer specifications.

2. Layered Structure:

One way to achieve modularity in the operating system is the layered approach. In this, the bottom layer is the hardware and the topmost layer is the user interface.

An image demonstrating the layered approach is as follows:

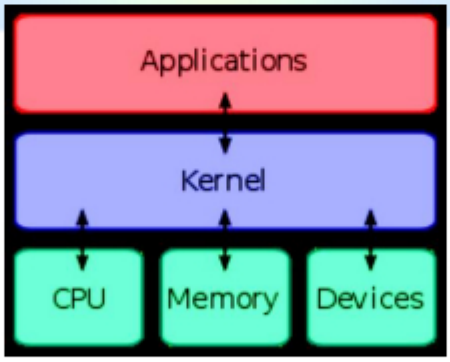


As seen from the image, each upper layer is built on the bottom layer. All the layers hide some structures, operations etc. from their upper layers.

One problem with the layered structure is that each layer needs to be carefully defined. This is necessary because the upper layers can only use the functionalities of the layers below them.

Kernel of Operating System:

A kernel is a central component of an operating system. It acts as an interface between the user applications and the hardware. The sole aim of the kernel is to manage the communication between the software (user level applications) and the hardware (CPU, disk memory etc.).



The Kernel is responsible for low-level tasks such as disk management, memory management, task management, etc. It provides an interface between the user and the hardware components of the system. When a process makes a request to the Kernel, then it is called System Call.

A Kernel is provided with a protected Kernel Space which is a separate area of memory and this area is not accessible by other application programs. So, the code of the Kernel is loaded into this protected Kernel Space.

Apart from this, the memory used by other applications is called the User Space. As these are two different spaces in the memory, so communication between them is a bit slower.

Is LINUX A Kernel Or An Operating System?

Well, there is a difference between kernel and OS. Kernel as described above is the heart of OS which manages the core features of an OS while if some useful applications and utilities are added over the kernel, then the complete package becomes an OS.

So, it can easily be said that an operating system consists of a kernel space and a user space. We can say that Linux is a kernel as it does not include applications like file-system utilities, windowing systems and graphical desktops, system administrator commands, text editors, compilers etc.

So, various companies add these kind of applications over Linux kernel and provide their operating system like Ubuntu, suse, centOS, redHat etc.

Types of Kernel:

Kernels may be classified mainly in two categories:

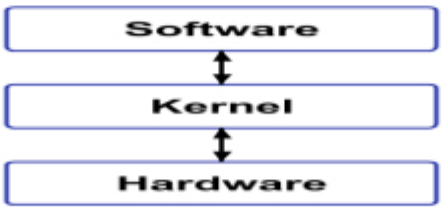
1. Monolithic Kernels:

Monolithic Kernels are those Kernels where the user services and the kernel services are implemented in the same memory space i.e. different memory for user services and kernel services are not used in this case.

By doing so, the size of the Kernel is increased and this, in turn, increases the size of the Operating System. As there is no separate User Space and Kernel Space, so the execution of the process will be faster in Monolithic Kernels.

This type of architecture led to some serious drawbacks like:

a. Size of kernel, which was huge
b. Poor maintainability, which means bug fixing or addition of new features resulted in recompilation of the whole kernel which could consume hours.



In a modern day approach to monolithic architecture, the kernel consists of different modules which can be dynamically loaded and un-loaded. This modular approach allows easy extension of OS's capabilities.

With this approach, maintainability of kernel became very easy as only the concerned module needs to be loaded and unloaded every time there is a change or bug fix in a particular module.

So, there is no need to bring down and recompile the whole kernel for a smallest bit of change. Also, stripping of kernel for various platforms (say for embedded devices etc.) became very easy as we can easily unload the module that we do not want. Linux follows the monolithic modular approach.

Advantages of Monolithic Kernels:

- It provides CPU scheduling, memory scheduling, file management through System calls only.
- Execution of the process is fast because there is no separate memory space for user and kernel.

Disadvantages of Monolithic Kernels:

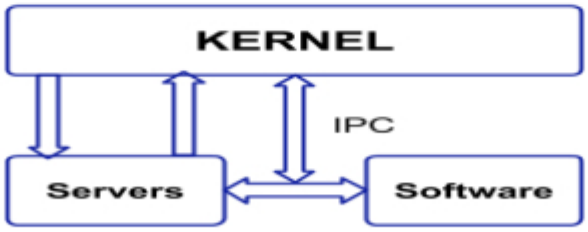
- If any service fails, then it leads to system failure.
- If new services are to be added then the entire Operating System needs to be modified.

2. Microkernels:

This architecture majorly caters to the problem of ever growing size of kernel code which we could not control in the monolithic approach. This architecture allows some basic services like device driver management, protocol stack, file system etc. to run in user space.

This reduces the kernel code size and also increases the security and stability of OS as we have the bare minimum code running in kernel.

So, if suppose a basic service like network service crashes due to buffer overflow, then only the networking service's memory would be corrupted, leaving the rest of the system still functional.



In this architecture, all the basic OS services which are made part of user space are made to run as servers which are used by other programs in the system through inter process communication (IPC).

Hybrid kernels are micro kernels that have some "non-essential" code in kernel-space in order for the code to run more quickly than it would be in user-space.

So, some services such as network stack or file system are run in Kernel space to reduce the performance overhead, but still, it runs kernel code as servers in the user-space.

4. Nanokernel

In a Nanokrnel, as the name suggests, the whole code of the kernel is very small i.e. the code executing in the privileged mode of the hardware is very small. The term nanokernel is used to describe a kernel that supports a nanosecond clock resolution.

5. Exokernel:

Exokernel is an Operating System kernel that is developed by the MIT parallel and the Distributed Operating Systems group. Here in this type of kernel, the resource protection is separated from the management and this, in turn, results in allowing us to perform application-specific customization.

In the Exokernel, the idea is not to implement all the abstractions. But the idea is to impose as few abstractions as possible and by doing so the abstraction should be used only when needed.

So, no force abstraction will be there in Exokernel and this is the feature that makes it different from a Monolithic Kernel and Microkernel. But the drawback of this is the complex design.

The design of the Exokernel is very complex. This is done by moving abstractions into untrusted userspace libraries called "library operating systems" (libOSes), which are linked to applications and call the operating system on their behalf.

Function of Kernel:

1. The Central Processing Unit:

This central component of a computer system is responsible for running or executing programs. The kernel takes responsibility for deciding at any time which of the many running programs should be allocated to the processor or processors (each of which can usually run only one program at a time).

2. Random-Access Memory:

Random-access memory is used to store both program instructions and data. Typically, both need to be present in memory in order for a program to execute.

Often multiple programs will want access to memory, frequently demanding more memory than the computer has available.

The kernel is responsible for deciding which memory each process can use, and determining what to do when not enough memory is available.

3. Input/output (I/O) Devices:

I/O devices include such peripherals as keyboards, mice, disk drives, printers, network adapters, and display devices.

The kernel allocates requests from applications to perform I/O to an appropriate device and provides convenient methods for using the device (typically abstracted to the point where the application does not need to know implementation details of the device).

4. Memory Management:

The kernel has full access to the system's memory and must allow processes to safely access this memory as they require it. Often the first step in doing this is virtual addressing, usually achieved by paging and/or segmentation.

Virtual addressing allows the kernel to make a given physical address appear to be another address, the virtual address.

Virtual address spaces may be different for different processes; the memory that one process accesses at a particular (virtual) address may be different memory from what another process accesses at the same address.

This allows every program to behave as if it is the only one (apart from the kernel) running and thus prevents applications from crashing each other.

5. Device Management:

A kernel must maintain a list of available devices. This list may be known in advance (e.g. on an embedded system where the kernel will be rewritten if the available hardware changes), configured by the user (typical on older PCs and on systems that are not designed for personal use) or detected by the operating system at run time (normally called plug and play).

In a plug and play system, a device manager first performs a scan on different hardware buses, such as Peripheral Component Interconnect (PCI) or Universal Serial Bus (USB), to detect installed devices, then searches for the appropriate drivers.

As device management is a very OS-specific topic, these drivers are handled differently by each kind of kernel design, but in every case, the kernel has to provide the I/O to allow drivers to physically access their devices through some port or memory location.

Very important decisions have to be made when designing the device management system, as in some designs accesses may involve context switches, making the operation very CPU-intensive and easily causing a significant performance overhead.

Architectures of Operating System:

An operating system might have many structure. According to the structure of the operating system; operating systems can be classified into many categories. Some of the main structures used in operating systems are:

1. Monolithic Architecture Of Operating System:

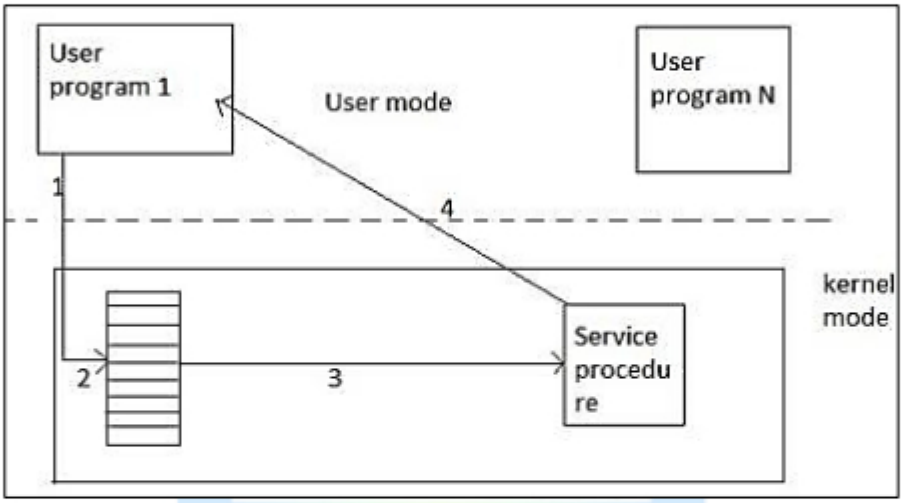


Fig: Monolithic Architecture

It is the oldest architecture used for developing operating system. Operating system resides on kernel for anyone to execute. System call is involved i.e. Switching from user mode to kernel mode and transfer control to operating system shown as event 1.

Many CPU has two modes, kernel mode, for the operating system in which all instruction are allowed and user mode for user program in which I/O devices and certain other instruction are not allowed.

Two operating system then examines the parameter of the call to determine which system call is to be carried out shown in event 2. Next, the operating system index's into a table that contains procedure that carries out system call.

This operation is shown in events. Finally, it is called when the work has been completed and the system call is finished, control is given back to the user mode as shown in event 4.

This approach might well be subtitled "The Big Mess." The structure is that there is no structure. The operating system is written as a collection of procedures, each of which can call any of the other ones whenever it needs to.

When this technique is used, each procedure in the system has a well-defined interface in terms of parameters and results, and each one is free to call any other one, if the latter provides some useful computation that the former needs. ***Example Systems: CP/M and MS-DOS***

2. Layered Architecture Of Operating System:

The layered Architecture of operating system was developed in 60's in this approach; the operating system is broken up into number of layers. The bottom layer (layer 0) is the hardware layer and the highest layer (layer n) is the user interface layer as shown in the figure.

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

The layered approach consists of breaking the operating system into the number of layers (level), each built on the top of lower layers. The bottom layer (layer 0) is the hardware layer; the highest layer is the user interface.

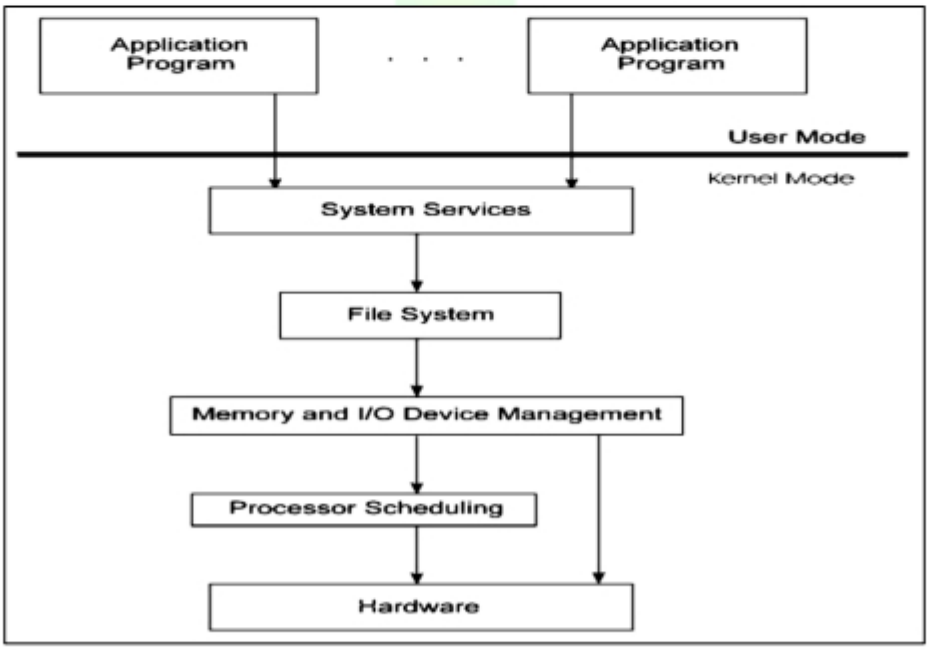


Fig: Layered Architecture

The main advantages of the layered approach is modularity. The layers are selected such that each uses functions (operations) and services of only lower-level layers. This approach simplifies debugging and system verifications.

That is in this approach, the Nth layer can access services provided by the (N-1)th layer and provide services to the (N+1)th layer.

This structure also allows the operating system to be debugged starting at the lowest layer, adding one layer at a time until the whole system works correctly. Layering also makes it easier to enhance the operating system; one entire layer can be replaced without affecting other parts of the system. ***Example Systems: VAX/VMS, Multics, UNIX***

3. Virtual Memory Architecture Of Operating System:

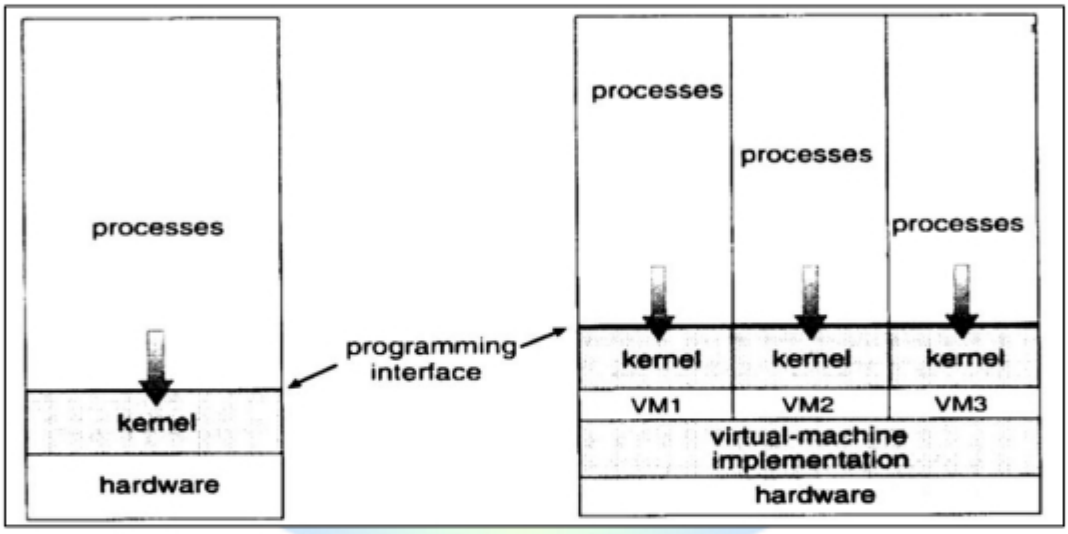


Fig: Virtual Memory Architecture

Virtual machine is an illusion of a real machine. It is created by a real machine operating system, which make a single real machine appears to be several real machine. The architecture of virtual machine is shown above.

The best example of virtual machine architecture is IBM 370 computer. In this system each user can choose a different operating system. Actually, virtual machine can run several operating systems at once, each of them on its virtual machine.

Its multiprogramming shares the resource of a single machine in different manner. **Example: Java JVM**

Concepts of Virtual Machine:

a. **Control Program (Cp):** Control Program creates the environment in which virtual machine can execute. It gives to each user facilities of real machine such as processor, storage I/0 devices.

b. **Conversation Monitor System (Cons):** Conversion Monitor System is a system application having features of developing program. It contains editor, language translator, and various application packages.

c. **Remote Spooling Communication System (RSCS):** It provides virtual machine with the ability to transmit and receive file in distributed system.

d. **IPCS (Interactive Problem Control System):** It is used to fix the virtual machine software problems.

4. Client/Server Architecture Of Operating System:

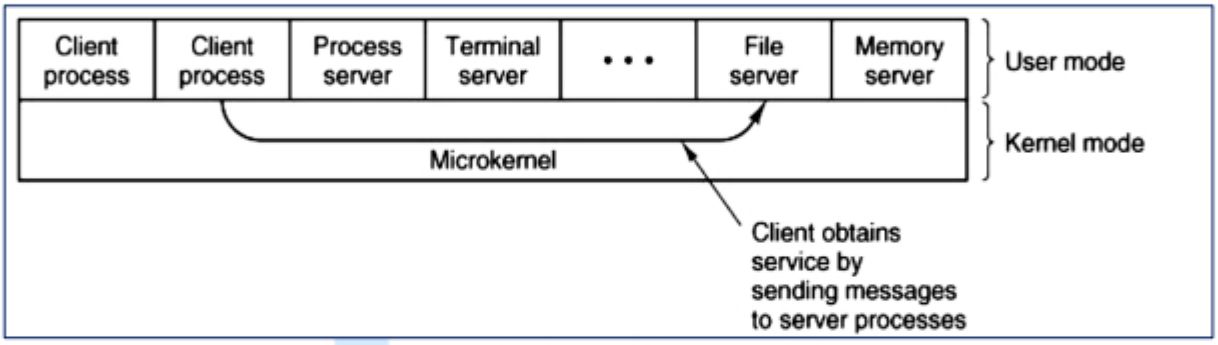


Fig: The Client Server Model

The advent of new concepts in operating system design, microkernel, is aimed at migrating traditional services of an operating system out of the monolithic kernel into the user-level process.

The idea is to divide the operating system into several processes, each of which implements a single set of services - for example, I/O servers, memory server, process server, threads interface system.

Each server runs in user mode, provides services to the requested client. The client, which can be either another operating system component or application program, requests a service by sending a message to the server.

An OS kernel (or microkernel) running in kernel mode delivers the message to the appropriate server; the server performs the operation; and microkernel delivers the results to the client in another message, as illustrated in Figure.

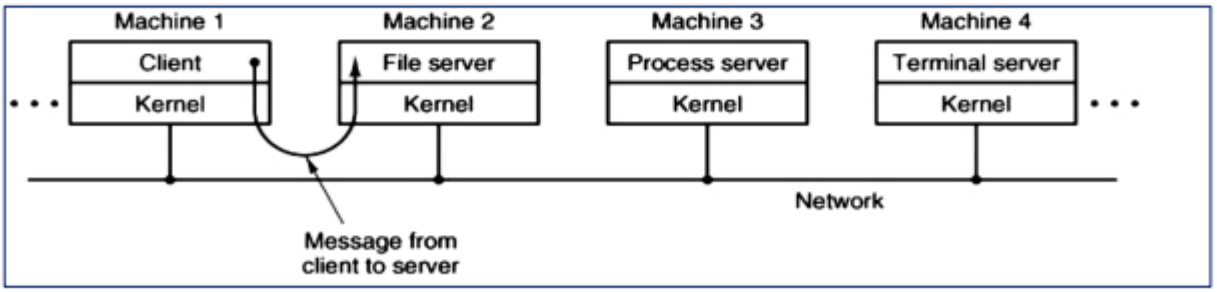


Fig: The Client-Server Model in Distributed System

In this model, the main task of the kernel is to handle all the communication between the client and the server by splitting the operating system into number of ports, each of which only handle some specific task i.e. file server, process server, terminal server and memory service.

Another advantage of the client-server model is it's adaptability to user in distributed system. If the client communicates with the server by sending it the message, the client need not know whether it was send.

Is the network to a server on a remote machine? As in case of client, same thing happen and occurs in client side that is a request was send and a reply come back.

Shell:

A shell is a program that provides the traditional text only user interface for Linux and other Unix operating system. Its primary function is to read commands typed into a console or terminal window and then execute it.

The term shell derives its name from the fact that it is an outer layer of the OS. A shell is an interface between the user and the internal part of the operating system. A user is in shell (i.e. interacting with the shell) as soon as the user has logged into the system.

A shell is the most fundamental way that user can interact with the system and the shell hides the detail of the underlying system from the user.

Types of Shell:

1. The C Shell:

Denoted as **csh**

Bill Joy created it at the University of California at Berkeley. It incorporated features such as aliases and command history. It includes helpful programming features like built-in arithmetic and C-like expression syntax.

In C shell:

```
Command full-path name is /bin/csh,
Non-root user default prompt is hostname %,
Root user default prompt is hostname #.
```

2. The Bourne Shell:

Denoted as **sh**

It was written by Steve Bourne at AT&T Bell Labs. It is the original UNIX shell. It is faster and more preferred. It lacks features for interactive use like the ability to recall previous

commands. It also lacks built-in arithmetic and logical expression handling. It is default shell for Solaris OS.

For the Bourne shell the:

```
Command full-path name is /bin/sh and /sbin/sh,
Non-root user default prompt is $,
Root user default prompt is #.
```

3. The Korn Shell:

It is denoted as **ksh**

It Was written by David Korn at AT&T Bell LabsIt is a superset of the Bourne shell.So it supports everything in the Bourne shell.It has interactive features.

It includes features like built-in arithmetic and C-like arrays, functions, and string-manipulation facilities.It is faster than C shell. It is compatible with script written for C shell.

For the Korn shell the:

```
Command full-path name is /bin/ksh,
Non-root user default prompt is $,
Root user default prompt is #.
```

4. GNU Bourne-Again Shell:

Denoted as **bash**

It is compatible to the Bourne shell. It includes features from Korn and Bourne shell.

For the GNU Bourne-Again shell the:

```
Command full-path name is /bin/bash,
Default prompt for a non-root user is bash-g.gg$
(g.ggindicates the shell version number like bash-3.50$),
Root user default prompt is bash-g.gg#.
```

Unit III: Process Management – Operating System

Process Concepts:

A process is a program in execution. Process is not as same as program code but a lot more than it. A process is an 'active' entity as opposed to program which is considered to be a 'passive' entity. Attributes held by process include hardware state, memory, CPU etc.

Process Memory Is Divided Into Four Sections For Efficient Working:

1. The **Text section** is made up of the compiled program code, read in from non-volatile storage when the program is launched.
2. The **Data section** is made up the global and static variables, allocated and initialized prior to executing the main.
3. The **Heap** is used for the dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
4. The **Stack** is used for local variables. Space on the stack is reserved for local variables when they are declared.

Process are:

- Dynamic
- Part of a program in execution
- Live entity, it can be created, executed and terminated.
- It goes through different states (wait, running, Ready etc.)
- Requires resources to be allocated by the OS
- One or more processes may be executing the same code.

The Process Model:

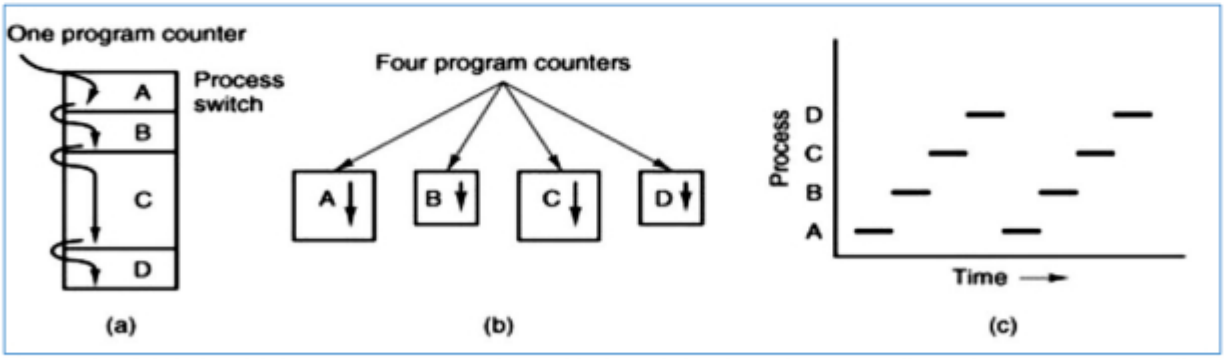


Fig: (a) Multiprogramming of four programs. (b) Conceptual model of four independent, sequential processes. (c) Only one program is active at any instant

In process model, all the runnable software on the computer sometimes including the operating system is organized into number of sequential process. Conceptually, each

process has its own CPU, but in the reality, the real CPU switches back and forth from process to process i.e. processes are running in quasi-parallel and the CPU switches from program to program. This rapid switching back and forth is called multi-programming.

Process States:

Although each process is an independent entity, with its own program counter and internal state, process often need to interact with other processes, one process may generate output and other process uses it as input. Process can be in one of the following states:

1. Running (using the CPU at that instant)
2. Ready (runnable; temporarily stopped to let another process run)
3. Blocked (unable to run until some external event happens)

Logically first two states are similar. Only in second case temporarily CPU is not available. The third state is different from the first two in that the process cannot run, even if CPU has nothing else to do i.e. idle.

Process States Transitions:

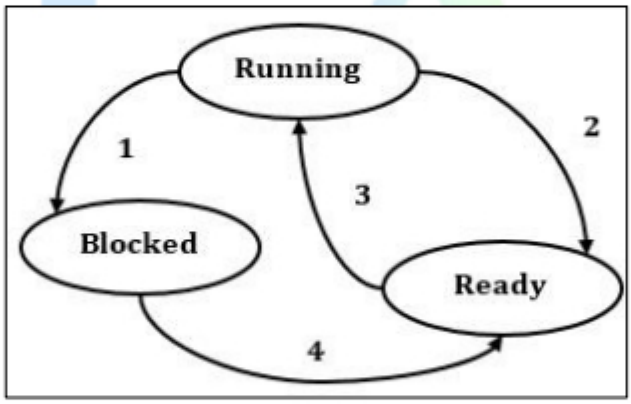


Fig: Process States Transitions

A process can be in running, ready and blocked state. Transitions 1, 2, 3, 4 are possible between these states:

1. Process blocks for input
2. Scheduler picks another job (process)
3. Scheduler picks this (current) process
4. Input becomes available

Transition '1' occurs when the operating system discovers that a process cannot continue right now.

Transition '2' and '3' are caused by process scheduler. (Here we deal with scheduling)

- Transition '2' occurs when scheduler decides that the running process has run long enough and it is time to let another process to have some CPU time.

- Transition '3' occurs when all the other processes have had their fair share and it is time for the first process to get the CPU again.

Transition '4' occurs, when the external event for which a process was waiting (such as arrival of some i/p) happens.

If no other process is running at that instant transition '3' will be triggered (process will start again), otherwise it may have to wait in ready state for a little while until the CPU is available and it turns comes.

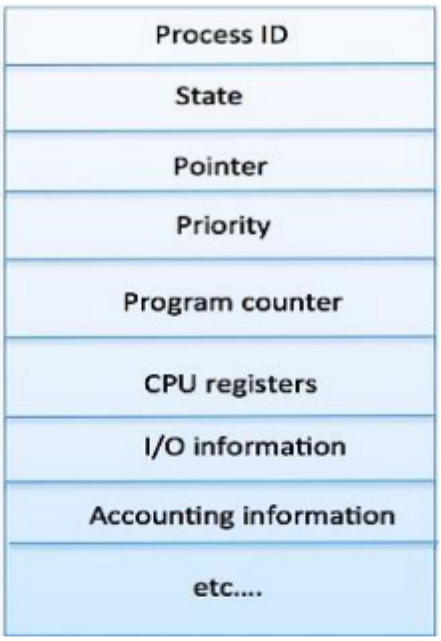
Process Control Block (PCB):

Process Control block is used for storing the collection of information about the Processes and this is also called as the Data Structure which Stores the information about the process. The information of the Process is used by the CPU at the Run time.

Process Control Block is maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). Process Control Block keeps all the information needed to keep track of a process as listed below in the table:

S.N.	Information & Description
1.	**Process State:** The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2.	**Process Privileges:** This is required to allow/disallow access to system resources.
3.	**Process ID:** Unique identification for each of the process in the operating system.
4.	**Pointer:** A pointer to parent process.
5.	**Program Counter:** Program Counter is a pointer to the address of the next instruction to be executed for this process.
6.	**CPU registers:** Various CPU registers where process need to be stored for execution for running state.
7.	**CPU Scheduling Information:** Process priority and other scheduling information which is required to schedule the process.
8.	**Memory Management Information:** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9.	**Accounting Information:** This includes the amount of CPU used for process execution, time limits, execution ID etc.
10.	**IO Status Information:** This includes a list of I/O devices allocated to the process.

The architecture of Process Control Block is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a Process Control Block:



The Process Control Block is maintained for a process throughout its lifetime, and is deleted once the process terminates.

Operations on Process:

1. Process Creation:

There are four principal events that cause processes to be created:

1. System initialization.
2. Execution of a process creation system call by a running process.
3. A user request to create a new process.
4. Initiation of a batch job.

Parent process create children processes, which, in turn create other processes, forming a tree of processes. Generally, process identified and managed via a process identifier. When an operating system is booted, often several processes are created.

Some of these are foreground processes, that is, processes that interact with (human) users and perform work for them. Others are background processes, which are not associated with particular users, but instead have some specific function.

For example, a background process may be designed to accept incoming requests for web pages hosted on that machine, waking up when a request arrives to service the request. Processes that stay in the background to handle some activity such as web pages, printing, and so on are called daemons.

In addition to the processes created at boot time, new processes can be created afterward as well. Often a running process will issue system calls to create one or more new processes to help it do its job.

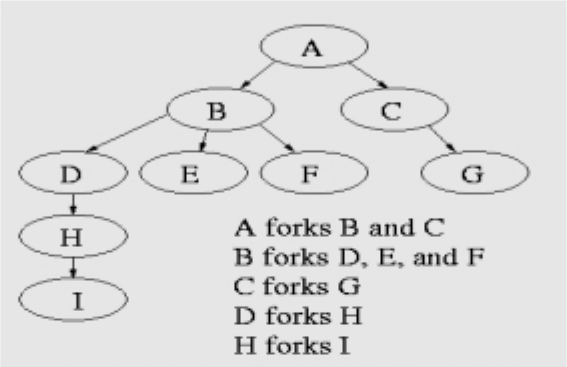
2. Process Termination:

After a process has been created, it starts running and does whatever its job is: After some time it will terminate due to one of the following conditions.

- Normal exit (voluntary)
- Error exit (voluntary)
- Fatal error (involuntary)
- Killed by another process (involuntary).

3. Process Hierarchies:

Modern general purpose operating systems permit a user to create and destroy processes.



- In UNIX this is done by the **fork** system call, which creates a **child** process, and the **exit** system call, which terminates the current process.
- After a fork both parent and child keep running (indeed they have the *same* program text) and each can fork off other processes.
- A process tree results. The root of the tree is a special process created by the OS during startup.
- A process can *choose* to wait for children to terminate. For example, if C issued a wait () system call it would block until G finished.

Old or primitive operating system like MS-DOS are not multi-programmed so when one process starts another, the first process is *automatically* blocked and waits until the second is finished.

4. Process Implementation:

Operating system maintains a table (an array of structure) known as process table with one entry per process to implement the process.

The entry contains detail about the process such as, *process state, program counter, stack pointer, memory allocation, the status of its open files, its accounting information, scheduling information and everything else* about the process that must be saved when the

process is switched from running to ready or blocked state, so that it can be restarted later as if it had never been stopped.

Process management	Memory management	File management
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Cooperating Process:

A process is independent if it cannot affect other process or be affected by it. Any process that does not share data with others is independent. Otherwise the process is cooperating.

Cooperation is done to provide information sharing, computational speedups, modularity and convenience. To allow cooperation there should be some mechanism for communication (called IPC: Inter-Process Comm.) and to synchronize their actions.

Advantages of Cooperating Processes:

1. Information Sharing:

Several users may which to share the same information e.g. a shared file. The O/S needs to provide a way of allowing concurrent access.

2. Computation Speedup:

Some problems can be solved quicker by sub-dividing it into smaller tasks that can be executed in parallel on several processors.

3. Modularity:

The solution of a problem is structured into parts with well-defined interfaces, and where the parts run in parallel.

4. Convenience:

A user may be running multiple processes to achieve a single goal, or where a utility may invoke multiple components, which interconnect via a pipe structure that attaches the stdout of one stage to stdin of the next etc.

If we allow processes to execute concurrently and share data, then we must either provide some mechanisms to handle conflicts e.g. writing and reading the same piece of data. We must also be prepared to handle inconsistent or corrupted data.

Example:

Below is a very simple example of two cooperating processes. The problem is called the Producer Consumer problem and it uses two processes, the producer and the consumer.

Producer Process: It produces information that will be consumed by the consumer.

Consumer Process: It consumes information produced by the producer.

Both processes run concurrently. If the consumer has nothing to consume, it waits.

There are two versions of the producer. In version one, the producer can produce an infinite amount of items. This is called the Unbounded Buffer Producer Consumer Problem. In the other version, there is a fixed limit to the buffer size.

When the buffer is full, the producer must wait until there is some space in the buffer before it can produce a new item. (Buffer could be provided by OS using IPC)

System Call:

In computing, a system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.

This may include hardware-related services (for example, accessing a hard disk drive), creation and execution of new processes, and communication with integral kernel services such as process scheduling. System calls provide an essential interface between a process and the operating system.

In most systems, system calls are possible to be made only from userspace processes, while in some systems, OS/360 and successors for example, privileged system code also issues system calls.

Types of System Call

1. Process Management:

A running program needs to be able to stop execution either normally or abnormally. When execution is stopped abnormally, often a dump of memory is taken and can be examined with a debugger. It include following items:

- Load
- Execute

- Create Process
- Terminate Process
- Allocate
- Get/Set Process
- Wait

2. File Management:

Some common system calls are **create, delete, read, write, reposition, open, and close**. Also, there is a need to determine the file attributes – **get and set file attribute**. Many times the OS provides an API to make these system calls.

3. Device Management:

Process usually require several resources to execute, if these resources are available, they will be granted and control returned to the user process. These resources are also thought of as devices. Some are physical, such as a video card, and others are abstract, such as a file.

User programs **request the device**, and when finished they **release the device**. Similar to files, we can **read, write, and reposition the device**. It is also responsible for **get/set attribute** and **attach/detach devices**.

4. Information Management:

Some system calls exist purely for transferring information between the user program and the operating system. An example of this is **get/set time or date**.

The OS also keeps information about all its processes and provides system calls to report this information. It also manage information of **get/set system data, processes, files, devices.**

5. Communication:

There are two models of communication, the message-passing model and the shared memory model.

- Message-passing uses a common mailbox to pass messages between processes.
- Shared memory use certain system calls to create and gain access to create and gain access to regions of memory owned by other processes. The two processes exchange information by reading and writing in the shared data.

Threads:

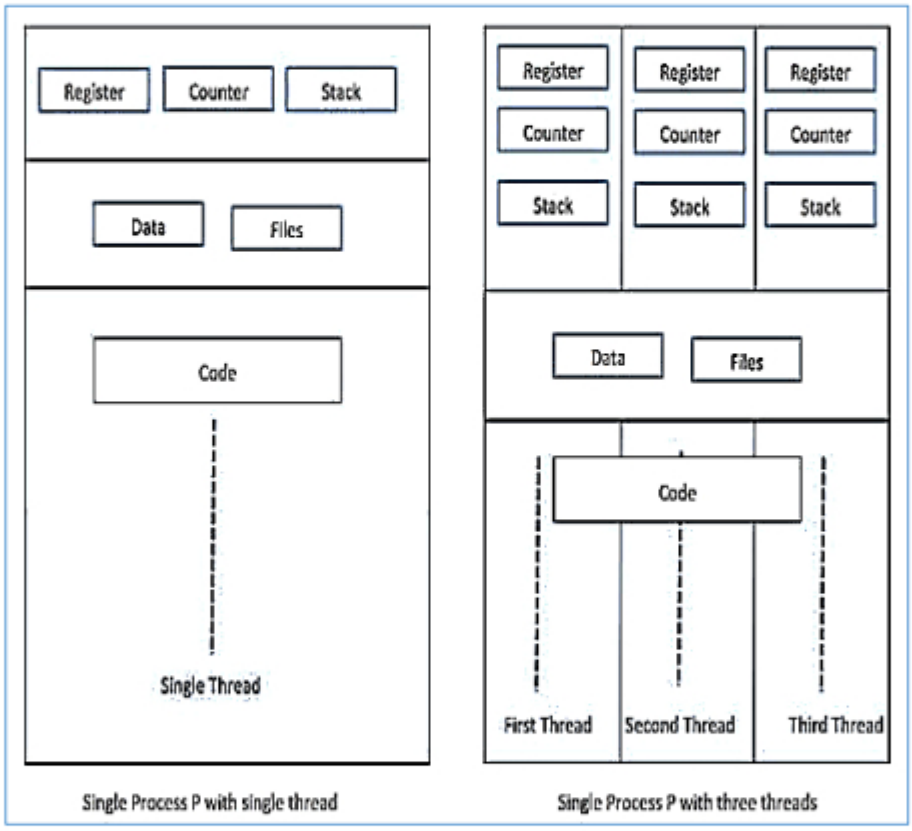
A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server.

They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.



Difference between Process and Thread:

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.

3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Advantages of Thread:

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

Types of Threads:

1. User Level Threads:

- User managed threads

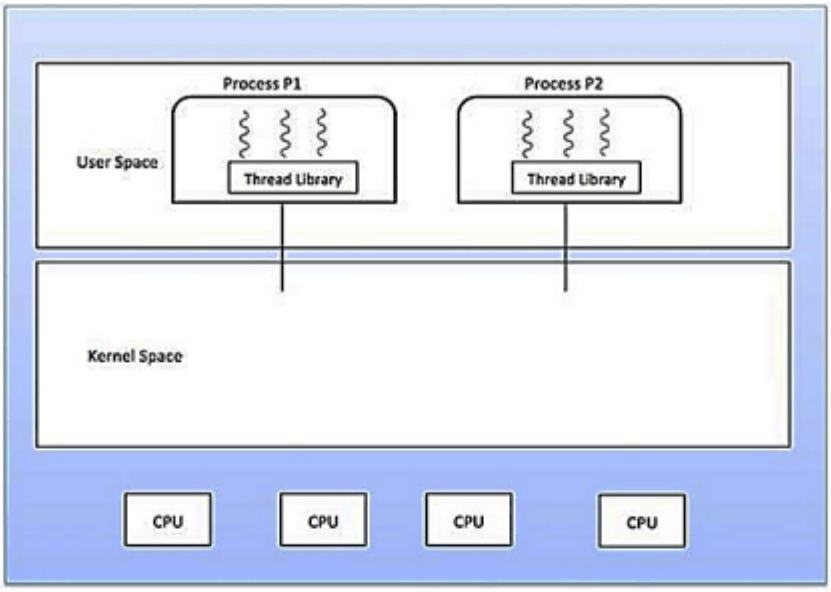
In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.

Advantages:

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages:

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.



2. Kernel Level Threads:

- Operating System managed threads acting on kernel

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system.

Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals' threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages:

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Disadvantages:

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Difference between User-Level & Kernel-Level Thread:

User-Level Threads	Kernel-Level Thread
User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

Multithreading Models:

Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach.

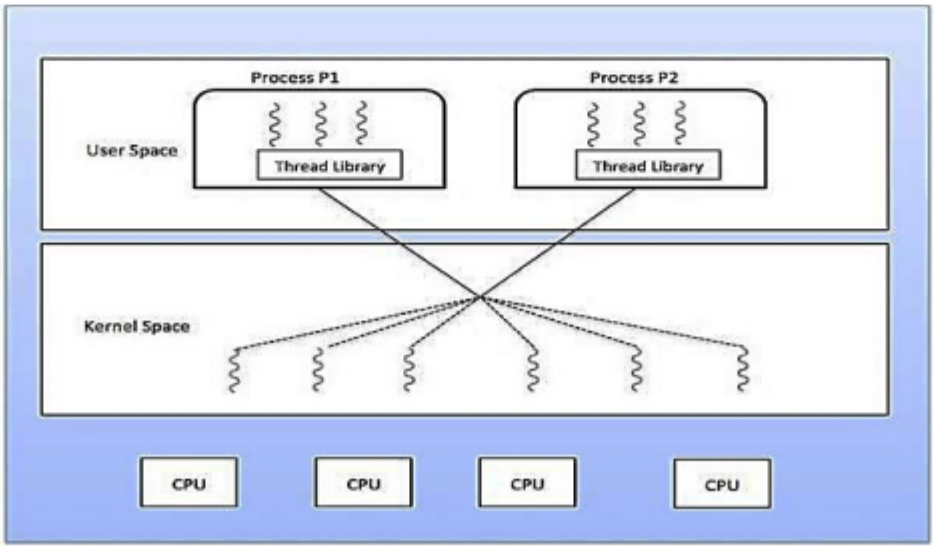
In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types:

1. Many to Many Model:

The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.

The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads.

In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine. This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.

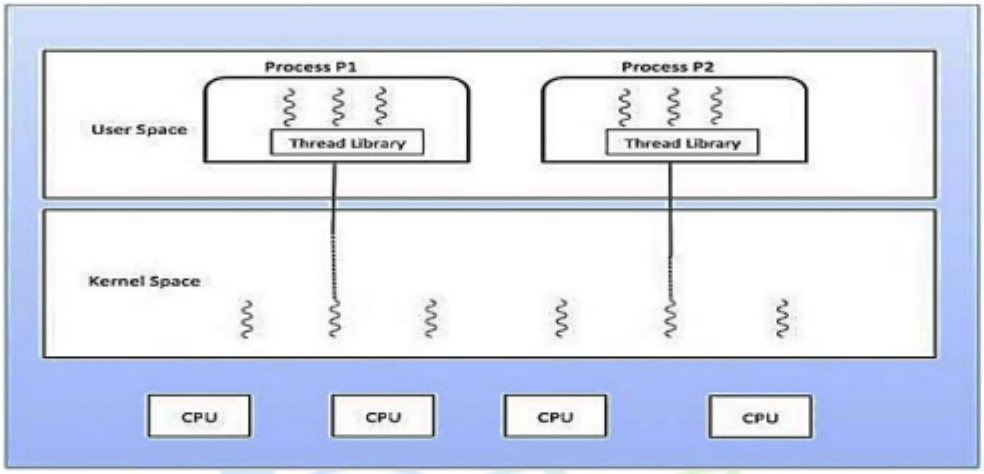


2. Many to One Model:

Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library.

When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.

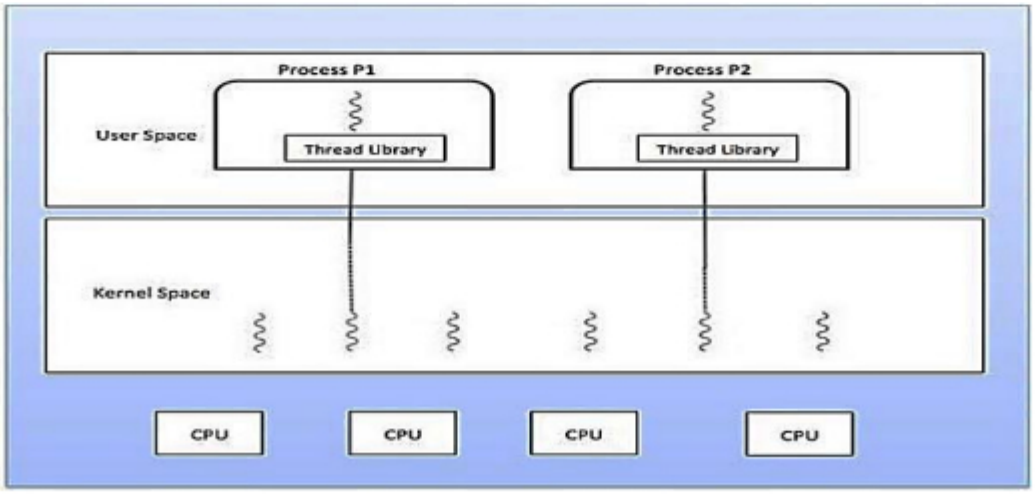


3. One to One Model:

There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model.

It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, Windows NT and windows 2000 use one to one relationship model.



Inter-Process Communication and Synchronization:

Introduction:

Inter process communication (IPC) is used for exchanging data between multiple threads in one or more processes or programs. The Processes may be running on single or multiple computers connected by a network. The full form of IPC is Inter-process communication.

IPC is a concept which allows the communication and synchronization between processes. Through a set of programming interfaces, it helps a programmer to organize and exchange the information between different processes.

"In computing, **Inter-process Communication** (**IPC**) is a set of methods for the exchange of data among multiple threads in one or more processes. Processes may be running on one or more computers connected by a network.

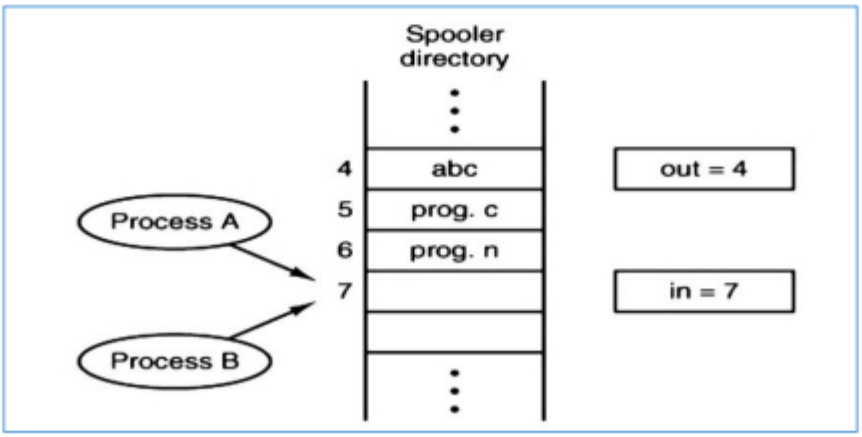
IPC methods are divided into methods for message passing, synchronization, shared memory, and remote procedure calls (RPC).

Race Condition:

A race condition occurs when two threads access a shared variable at the same time. The first thread reads the variable, and the second thread reads the same value from the variable.

Then the first thread and second thread perform their operations on the value, and they race to see which thread can write the value last to the shared variable. The value of the thread that writes its value last is preserved, because the thread is writing over the value that the previous thread wrote.

Example: A Printer Spooler



When a process wants to print a file, it enters the file name in a special spooler directory. Another process, the printer daemon, periodically checks to see if there are any files to be printed, and if there are, it prints them and removes their names from the directory.

Imagine that our spooler directory has a large number of slots, numbered 0, 1, 2, ..., each one capable of holding a file name. Also imagine that there are two shared variables,

- Out: which points to the next file to be printed
- In: which points to the next free slot in the directory.

At a certain instant, slots 0 to 3 are empty (the files have already been printed) and slots 4 to 6 are full (with the names of files to be printed). More or less simultaneously, processes A and B decide they want to queue a file for printing as shown in the figure above.

Process A reads in and stores the value, 7, in a local variable called **NextFreeSlot**. Just then a clock interrupt occurs and the CPU decides that process A has run long enough, so it switches to process B. Process B also reads in, and also gets a 7, so it stores the name of its file in slot 7 and updates in to be an 8. Then it goes off and does other things.

Eventually, process A runs again, starting from the place it left off last time. It looks at **NextFreeSlot**, finds a 7 there, and writes its file name in slot 7, erasing the name that process B just put there.

Then it computes **NextFreeSlot + 1,** which is 8, and sets in to 8. The spooler directory is now internally consistent, so the printer daemon will not notice anything wrong, but process B will never receive any output.

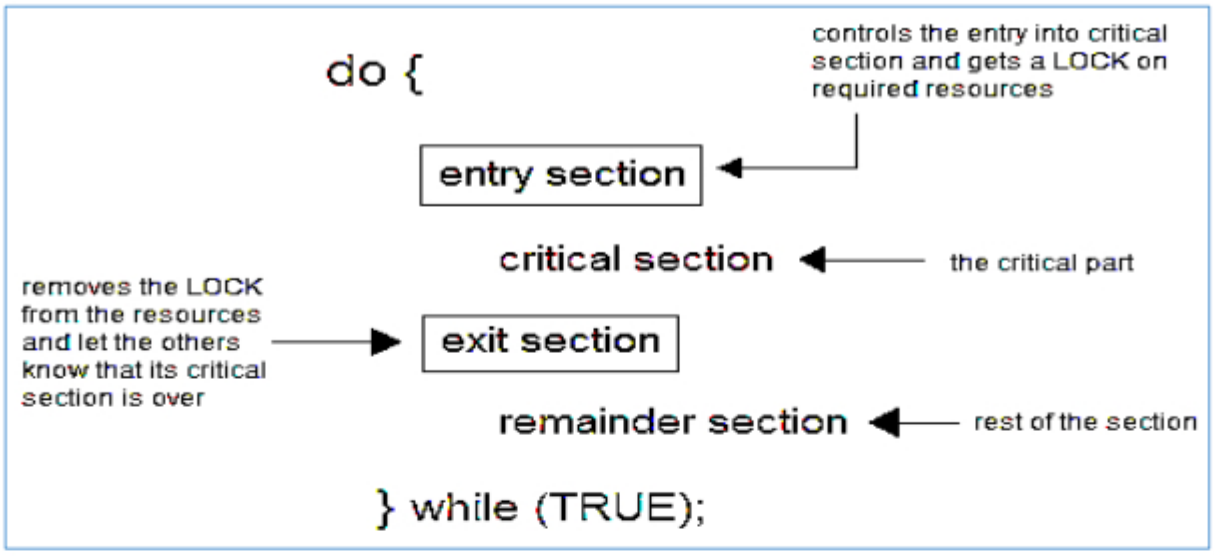
(Rules for Avoiding Race Condition) Solution to Critical Section Problem:

1. No two processes may be simultaneously inside their critical regions. (Mutual Exclusion)
2. No assumptions may be made about speeds or the number of CPUs.
3. No process running outside its critical region may block other processes.
4. No process should have to wait forever to enter its critical region.

Critical Region:

A Critical Section or Region is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section.

If any other process also wants to execute its critical section, it must wait until the first one finishes.



The critical section must satisfy the following three conditions:

1. Mutual Exclusion:

Out of a group of cooperating processes, only one process can be in its critical section at a given point of time.

2. Progress:

If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.

3. Bounded Waiting:

After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, system must grant the process permission to get into its critical section.

Methods for Avoiding Critical Region:

1. Mutual Exclusion Condition:

Here process **A** enters its critical region at time T_1. A little later, at time T_2 process **B** attempts to enter its critical region, but fails because another process is already in its

critical region and we allow only one at a time. Temporarily, process **B** is suspended until time T_3.

When process **A** leaves its critical region (at time T_3) then process **B** allow to enter its critical region immediately. Eventually process **B** leaves its critical region at time T_4 and we are back to the original process with no processes in its critical region.

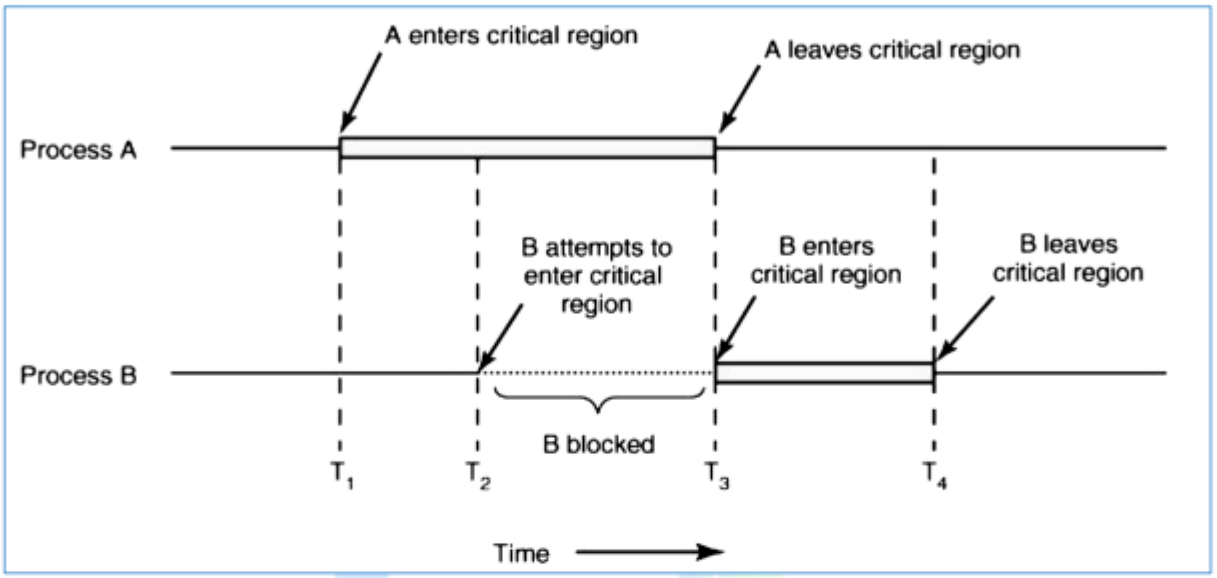


Fig: Mutual Exclusion Using Critical Region/Critical Section

2. Proposals for Achieving Mutual Exclusion:

Here, we study various proposals for achieving mutual exclusion, so that while one process is busy updating shared memory in its critical region, no other process will enter its critical region and cause trouble.

a. Disabling Interrupts:

On a single processor system, the simplest solution is to have each process disable all interrupts just after entering its critical region and re-enable them just before leaving its critical region.

With interrupt disabled, no clock interrupts can occur. Thus, once a process has disabled interrupts it can use critical region without fear that any process will intervene (disturb).

Disadvantages:

- Suppose that one of the process disable interrupts and never turned on again, that could be the end of the system.
- Not suitable for system having multiple CPUs.

b. Lock Variables:

It is a software solution. Consider having a single shared variable initially 0*(no process in its critical region)*. When a process wants to enter its critical region, it first test the lock.

If the lock is 0, the processor set it to 1*(some process in its critical region)* and enter its critical region. If the lock is already 1 the process just waits until it becomes 0.

Disadvantages:

It is just like as we saw in spooler directory. Suppose that one process reads the lock and sees that is 0. Before it can set the lock to 1 another process is scheduled, runs and sets the clock to 1.

When the first process run again, it will also set the lock to 1 and two process will be in their critical region at the same time.

c. Strict Alternation:

Process 0	Process 1

While (TRUE) {
while (turn != 0); //wait
critical_section();
turn = 1;
noncritical_section();
}
	While (TRUE) {
	while (turn != 1); // wait
	critical_section();
	turn = 0;
	noncritical_section();
	}

These code fragments offer a solution to the mutual exclusion problem.
Assume the variable ***turn*** is initially set to zero. Process 0 is allowed to run. It finds that ***turn*** is zero and is allowed to enter its critical region.

If process 1 tries to run, it will also find that ***turn*** is zero and will have to wait (the while statement) until ***turn*** becomes equal to 1. When process 0 exits its critical region it sets ***turn*** to 1, which allows process 1 to enter its critical region.

If process 0 tries to enter its critical region again it will be blocked as turn is no longer zero.

However, there is one major flaw in this approach. Consider this sequence of events.

1. Process 0 runs, enters its critical section and exits; setting turn to 1. Process 0 is now in its non-critical section. Assume this non-critical procedure takes a long time.
2. Process 1, which is a much faster process, now runs and once it has left its critical section turn is set to zero.
3. Process 1 executes its non-critical section very quickly and returns to the top of the procedure.

4. The situation is now that process 0 is in its non-critical section and process 1 is waiting for turn to be set to zero. In fact, there is no reason why process 1 cannot enter its critical region as process 0 is not in its critical region.

What we can see here is violation of one of the conditions that we listed above (number iii). That is, a process, not in its critical section, is blocking another process.

If you work through a few iterations of this solution you will see that the processes must enter their critical sections in turn; thus this solution is called strict alternation.

d. Peterson's Solution:

A solution to the mutual exclusion problem that does not require strict alternation, but still uses the idea of lock (and warning) variables together with the concept of taking turns is described in (Dijkstra, 1965).

In fact the original idea came from a Dutch mathematician (T. Dekker). This was the first time the mutual exclusion problem had been solved using a software solution. (Peterson, 1981), came up with a much simpler solution.

The solution consists of two procedures, shown here in a C style syntax.

```
#define FALSE 0
#define TRUE 1
int No_Of_Processes; // Number of processes
int turn; // Whose turn is it?
int interested[No_Of_Processes]; // All values initially FALSE
void EnterRegion(int process) {
int other; // number of the other process
other = 1 - process; // the opposite process
interested[process] = TRUE; // this process is interested
turn = process; // set flag
while(turn == process && interested[other] == TRUE); // wait
}
void LeaveRegion (int process) {
interested[process] = FALSE; // process leaves critical region
}
```

A process that is about to enter its critical region has to call ***EnterRegion***. At the end of its critical region it calls ***LeaveRegion***.

Initially, both processes are not in their critical region and the array *interested* has all (both in the above example) its elements set to false.

Assume that process 0 calls ***EnterRegion***. The variable *other* is set to one (the other process number) and it indicates its interest by setting the relevant element of *interested*. Next it sets the *turn* variable, before coming across the while loop. In this instance, the process will be allowed to enter its critical region, as process 1 is not interested in running.

Now process 1 could call ***EnterRegion***. It will be forced to wait as the other process (0) is still interested. Process 1 will only be allowed to continue when *interested* [0] is set to false which can only come about from process 0 calling ***LeaveRegion***.

If we ever arrive at the situation where both processes call enter region at the same time, one of the processes will set the turn variable, but it will be immediately overwritten.

Assume that process 0 sets turn to zero and then process 1 immediately sets it to 1. Under these conditions process 0 will be allowed to enter its critical region and process 1 will be forced to wait.

e. Test and Set Lock:

If we are given assistance by the instruction set of the processor we can implement a solution to the mutual exclusion problem. The instruction we require is called test and set lock (TSL).

This instructions reads the contents of a memory location, stores it in a register and then stores a non-zero value at the address. This operation is guaranteed to be indivisible. That is, no other process can access that memory location until the TSL instruction has finished.

This assembly (like) code shows how we can make use of the TSL instruction to solve the mutual exclusion problem.

```
EnterRegion:
tsl register, flag ; copy flag to register and set flag to 1
cmp register, #0 ;was flag zero?
jnz EnterRegion ;if flag was non zero, lock was set , so loop ret ;return
(and enter critical region)
LeaveRegion:
mov flag, #0 ; store zero in flag
ret ;return
```

Assume, again, two processes.

Process 0 calls ***EnterRegion***. The tsl instruction copies the flag to a register and sets it to a non-zero value. The flag is now compared to zero (cmp - compare) and if found to be non-zero (jnz - jump if non-zero) the routine loops back to the top.

Only when process 1 has set the flag to zero (or under initial conditions), by calling ***LeaveRegion***, will process 0 be allowed to continue.

f. Sleep and Weak up:

As its name suggests, it uses two system calls: Sleep and Wakeup. If a process is unable to enter a critical section, it is put to sleep/temporarily suspend.

Afterwards, when the critical section is no longer occupied, the process is woken up. As you can already see, this solution puts much less stress on the CPU compared to the busy waiting solutions.

Note that sleep and wakeup calls are library functions/methods in languages like Java. Therefore, processes/threads can easily be put to sleep or woken up.

To understand how this concept works, we can take the example of a shared buffer with *n* slots in it. The producer sits on the left and inputs data bits and the consumer sits on the right and takes out the data bits.

In a way, it is like an airport luggage belt, with bags being put on one end and then being taken off on the other. There is a shared variable *count* that stores the number of bits currently in the buffer.

Assuming the buffer is 8 bits long, when count is 8 the consumer must put the producer to sleep. And when count is 0 the producer must put the consumer to sleep. This makes sure that the producer can't input anything when the buffer is full and that the consumer can't take out what's not there.

3. Producer-Consumer Problem:

Two process share a common fixed size buffer. One of them, the producer, puts information in the buffer and other one, the consumer takes it out.

Trouble arises when the producer wants to put a new item in the buffer but it is already full. The solution for the producer is to go to sleep, to be awakened when the consumer has removed one or more items.

Similarly, if the consumer wants to remove an item from the buffer and sees that the buffer is empty it goes to sleep until the producer puts something in the buffer and wakes it up.

To keep track of number of items in the buffer, we will need a variable “count”. If the maximum number of items the buffer can holds is “N”, the producer code will first test to see if count is “N”, if it is so, the producer will go to sleep. If it is not the producer will add an item and increment “count”.

The consumer’s code is also same as producer. It first test “count” to see if it is 0. When it found “count” is 0, it goes to sleep. Otherwise it starts to remove items and decrement the count.

```
#define N 100
Int count = 0;
Void producer (void) {
      Int item;
      While(TRUE){
      Item = produce_item();
      If(count==N){
      Sleep();
```

```
        Insert_item();
        Count=count+1;
        }
        If(count==1){
        Wakeup(consumer)
        }
        }
        }
Void consumer(void){
        Int item;
        While(TRUE){
        If(count==0){
        Sleep();
        Item=remove-item();
        Count=count-1;
        }
        If(count==N-1){
        Wakeup(producer)
        Consumer_item(item)
        }
        }
        }
```

Race condition can occur because access to the count is unconstrained. The following situation could be occur.

The buffer is empty and the consumer has just reads the count to see if it is 0. At that instant scheduler decides to stop running the consumer temporarily and starts running the producer.

The producer inserts the item in the buffer increments the count and notice that is in 1. Producer thinks that consumer must be sleeping (because it reads 0), the produce calls wakeup to wake up the consumer.

Unfortunately the consumer is not yet asleep, so the wakeup signal is lost. When the consumer next runs it and goes to sleep. Sooner or later the producer will fill the buffer and go to sleep. At last both will sleep forever.

A quick fix is to modify the rules to add a **wakeup waiting bit**. When a wakeup is sent to a process that is still awake, this bit is set. Later, when the process tries to go to sleep, if the wakeup waiting bit is on, it will be turned off, but the process will stay awake.

4. Types of Mutual Exclusion:

a. Semaphore:

E.W. Dijkstra (1965), suggested using an integer variable to count the number of wakeups saved for future use. In his proposal a new variable type which he called a semaphore was introduced.

A semaphore could have the value 0 indicating no wakeups were saved or some positive value if one or more wakeups are pending. Operations involved in semaphore are down as sleep and up as wake up.

❖ Down Operation:

The down operation on semaphore checks to see if the value is greater than 0. If so, it decrements the value (i.e. uses one stored wakeup) and just continues. If the value is 0, the process is put to sleep without completing the down for the moment.

Checking the value, changing it and possibly going to sleep are all atomic operations. It is guaranteed that once a semaphore operation has started, no other process can access the semaphore until the operation has completed or blocked (for synchronization and avoiding race condition).

❖ Up Operation:

The up operation increments the value of the semaphore. If one or more process were sleeping on that semaphore, unable to complete an earlier down operation one of them is chosen by the system (at random) and is allowed to complete its down operation.

Thus after an up operation on a semaphore with processes sleeping on it, the semaphore will still be 0, but there will be one fewer process sleeping on it.

The operation of incrementing the semaphore and waking up one process is also indivisible (atomic). No, process ever blocks doing up, just as no process ever blocks doing a wakeup.

Atomic: Group of related operations are either not performed or all performed at all.

Advantages of semaphores:

- ❖ Processes do not busy wait while waiting for resources. While waiting, they are in a "suspended'' state, allowing the CPU to perform other chores.
- ❖ Works on (shared memory) multiprocessor systems.
- ❖ User controls synchronization.

Disadvantages of semaphores:

- ❖ Can only be invoked by processes--not interrupt service routines because interrupt routines cannot block
- ❖ User controls synchronization could mess up.

Semaphore Implementation without Busy Waiting:

❖ Implementation Of Wait:

```
wait (S){
value - -;
if (value < 0) {
add this process to waiting queue
block();
}
}
```

❖ Implementation Of Signal:

```
Signal (S){
value++;
if (value <= 0) {
remove a process P from the waiting queue
wakeup(P);
}
}
```

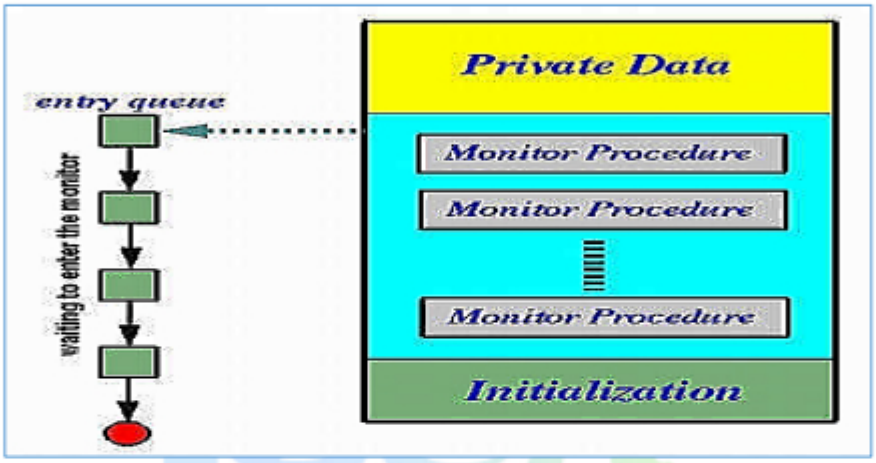
```
#define N 100 /* number of slots in the buffer */
typedef int semaphore; /* semaphores are a special kind of int */
semaphore mutex = 1; /* controls access to critical region */
semaphore empty = N; /* counts empty buffer slots */
semaphore full = 0; /* counts full buffer slots */
void producer(void){
int item;
while (TRUE){ /* TRUE is the constant 1 */
item = produce_item(); /* generate something to put in buffer */
down(&empty); /* decrement empty count */
down(&mutex); /* enter critical region */
insert_item(item); /* put new item in buffer */
up(&mutex); /* leave critical region */
up(&full); /* increment count of full slots */
}
}

void consumer(void){
int item;
while (TRUE){ /* infinite loop */
down(&full); /* decrement full count */
down(&mutex); /* enter critical region */
item = remove_item(); /* take item from buffer */
up(&mutex); /* leave critical region */
up(&empty); /* increment count of empty slots */
consume_item(item); /* do something with the item */
}
}
```

b. Monitors:

A monitor has four components as shown below: initialization, private data, monitor procedures, and monitor entry queue. The initialization component contains the code that is used exactly once when the monitor is created.

The private data section contains all private data, including private procedures that can only be used within the monitor. Thus, these private items are not visible from outside of the monitor.

The monitor procedures are procedures that can be called from outside of the monitor. The monitor entry queue contains all threads that called monitor procedures but have not been granted permissions.



A monitor looks like a class with the initialization, private data and monitor procedures corresponding to constructors, private data and methods of that class. The only major difference is that classes do not have entry queues.

Monitors are supposed to be used in a multithreaded or multi-process environment in which multiple threads/processes may call the monitor procedures at the same time asking for service.

Thus, a monitor guarantees that at any moment at most one thread can be executing in a monitor! What does this mean? When a thread calls a monitor procedure, we can view the called monitor procedure as an extension to the calling thread.

If the called monitor procedure is in execution, we will say the calling thread is in the monitor executing the called monitor procedure. Now, if two threads are in the monitor (i.e., they are executing two, possibly the same, monitor procedures), some private data may be modified by both threads at the same time causing race conditions to occur.

Therefore, to guarantee the integrality of the private data, a monitor enforces mutual exclusion implicitly. More precisely, if a thread calls a monitor procedure, this thread will be blocked if there is another thread executing in the monitor.

Those threads that were not granted the entering permission will be queued to a monitor entry queue outside of the monitor. When the monitor becomes empty (i.e., no thread is

executing in it), one of the threads in the entry queue will be released and granted the permission to execute the called monitor procedure.

Although we say "entry queue," you should not view it literally. More precisely, when a thread must be released from the entry queue, you should not assume any policy for which thread will be released.

In summary, monitors ensure mutual exclusion automatically so that there is no more than one thread can be executing in a monitor at any time. This is a very usably and handy capability.

c. Mutex:

Mutex is the short form for 'Mutual Exclusion object'. A mutex allows multiple threads for sharing the same resource. The resource can be file. A mutex with a unique name is created at the time of starting a program.

A mutex must be locked from other threads, when any thread that needs the resource. When the data is no longer used / needed, the mutex is set to unlock. A mutex can be unlocked only by the thread that locked it. Thus a mutex has an owner concept.

A mutex and the binary semaphore are essentially the same. Both can take values: 0 as unlock and 1 as lock. However, there is a significant difference between them that makes mutexes more efficient than binary semaphores.

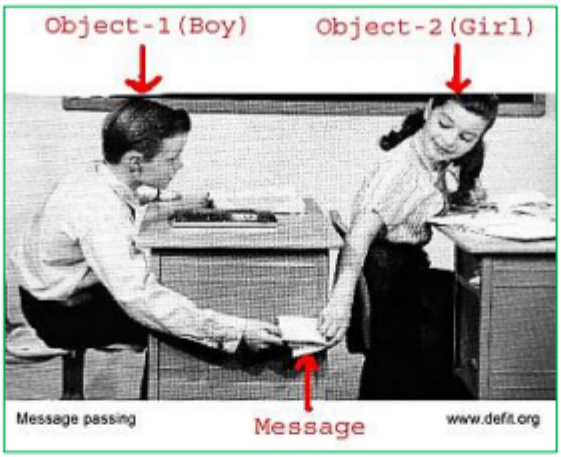
❖ Working Mechanism:

Two procedures are used with mutex. When a process thread needs access to critical region it calls MutexLock. If the mutex is currently unlocked the call succeeds and the calling process is free to enter its critical region.

On the other hand if the mutex is already locked the calling process is blocked until the process is finished and calls the MutexUnlock. If multiple process are blocked on the mutex one of them is chosen at random and allowed to acquire the lock. No busy waiting only tests for "lock"

d. Message Passing:

Message passing is a type of communication between processes or objects in computer science. In this model, processes or objects can send and receive messages (signals, functions, complex data structures, or data packets) to other processes or objects.



In other word, Message passing is a form of communication between objects, processes or other resources used in object-oriented programming, inter-process communication and parallel computing.

Message passing can be synchronous or asynchronous. Synchronous message passing systems require the sender and receiver to wait for each other while transferring the message. In asynchronous communication the sender and receiver do not wait for each other and can carry on their own computations while transfer of messages is being done.

The concept of message passing makes it easier to build systems that model or simulate real-world problems.

5. Serializability:

It identifies data transactions as occurring serially, independent of one another, even though they may have occurred concurrently. A schedule or list of transactions is deemed to be correct if they are serialized, otherwise, they may contain errors that can lead to duplication or overlap.

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions.

We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. Concurrency control protocols can be broadly divided into two categories:

1. Lock-based Protocols:

Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds:

❖ Binary Locks:

A lock on a data item can be in two states; it is either locked or unlocked.

❖ Shared/exclusive:

This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

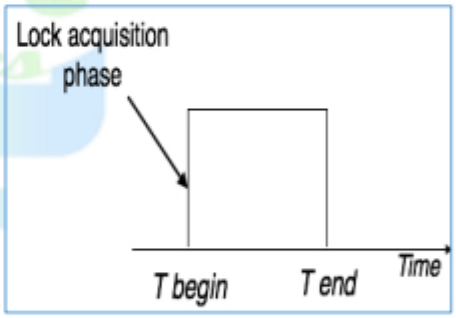
There Are Four Types Of Lock Protocols Available:

a. Simplistic Lock Protocol:

Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the 'write' operation.

b. Pre-claiming Lock Protocol:

Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand.

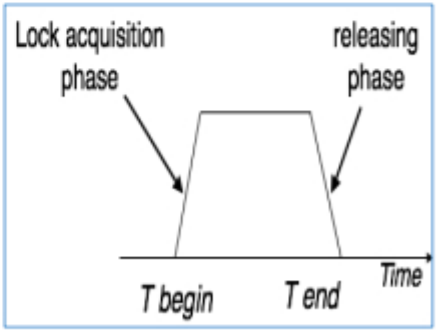


If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.

c. Two-Phase Locking 2PL:

This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires.

The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts.



In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.

Two-phase locking has two phases, one is **growing**, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released.

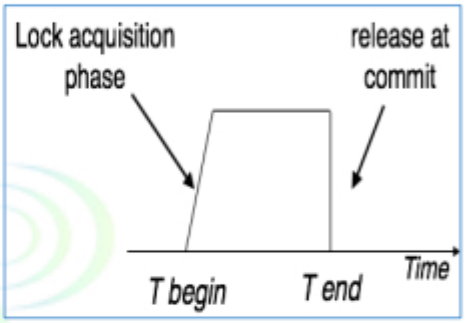
To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.

d. Strict Two-Phase Locking:

The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally.

But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.

Strict-2PL does not have cascading abort as 2PL does.



2. Timestamp-based Protocols:

The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.

Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it.

For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one.

In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last 'read and write' operation was performed on the data item.

Timestamp Ordering Protocol:

The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

- The timestamp of transaction T_i is denoted as TS(T_i).
- Read time-stamp of data-item X is denoted by R-timestamp(X).
- Write time-stamp of data-item X is denoted by W-timestamp(X).

Timestamp Ordering Protocol Works As Follows:

1. **If a transaction Ti issues a read(X) operation:**
   - If TS(Ti) < W-timestamp(X)
     - Operation rejected.
   - If TS(Ti) >= W-timestamp(X)
     - Operation executed.
   - All data-item timestamps updated.

2. **If a transaction Ti issues a write(X) operation:**
   - If TS(Ti) < R-timestamp(X)
     - Operation rejected.
   - If TS(Ti) < W-timestamp(X)
     - Operation rejected and Ti rolled back.
   - Otherwise, operation executed.

Thomas' Write Rule:

This rule states if TS(Ti) < W-timestamp(X), then the operation is rejected and T_i is rolled back. Time-stamp ordering rules can be modified to make the schedule view serializable. Instead of making T_i rolled back, the 'write' operation itself is ignored.

Classical IPC Problems:

1. Dining Philosophers Problems:

There are N philosophers sitting around a circular table eating spaghetti and discussing philosophy. The problem is that each philosopher needs 2 forks to eat, and there are only N forks, one between each 2 philosophers.

Design an algorithm that the philosophers can follow that insures that none starves as long as each philosopher eventually stops eating, and such that the maximum number of philosophers can eat at once.

- Philosophers eat/think
- Eating needs 2 forks
- Pick one fork at a time
- How to prevent deadlock

The problem was designed to illustrate the problem of avoiding deadlock, a system state in which no progress is possible.

One idea is to instruct each philosopher to behave as follows:

- Think until the left fork is available; when it is, pick it up
- Think until the right fork is available; when it is, pick it up
- Eat
- Put the left fork down
- Put the right fork down
- Repeat from the start

This solution is incorrect: it allows the system to reach deadlock. Suppose that all five philosophers take their left forks simultaneously. None will be able to take their right forks, and there will be a deadlock.

We could modify the program so that after taking the left fork, the program checks to see if the right fork is available. If it is not, the philosopher puts down the left one, waits for some time, and then repeats the whole process.

This proposal too, fails, although for a different reason.

With a little bit of bad luck, all the philosophers could start the algorithm simultaneously, picking up their left forks, seeing that their right forks were not available, putting down their left forks, waiting, and picking up their left forks again simultaneously, and so on, forever.

A situation like this, in which all the programs continue to run indefinitely but fail to make any progress is called starvation.

The solution presented below is deadlock-free and allows the maximum parallelism for an arbitrary number of philosophers. It uses an array, state, to keep track of whether a philosopher is eating, thinking, or hungry (trying to acquire forks).

A philosopher may move into eating state only if neither neighbor is eating. Philosopher i's neighbors are defined by the macros LEFT and RIGHT. In other words, if i is 2, LEFT is 1 and RIGHT is 3.

Solution:

```
#define N 5 /* number of philosophers */
#define LEFT (i+N-1)%N /* number of i's left neighbor */
#define RIGHT (i+1)%N /* number of i's right neighbor */
#define THINKING 0 /* philosopher is thinking */
#define HUNGRY 1 /* philosopher is trying to get forks */
```

```
#define EATING 2 /* philosopher is eating */
typedef int semaphore; /* semaphores are a special kind of int */
int state[N]; /* array to keep track of everyone's state */
semaphore mutex = 1; /* mutual exclusion for critical regions */
semaphore s[N]; /* one semaphore per philosopher */
void philosopher(int i) /* i: philosopher number, from 0 to N1 */
{
while (TRUE){ /* repeat forever */
think(); /* philosopher is thinking */
take_forks(i); /* acquire two forks or block */
eat(); /* yum-yum, spaghetti */
put_forks(i); /* put both forks back on table */
}
}
void take_forks(int i) /* i: philosopher number, from 0 to N1 */
{
down(&mutex); /* enter critical region */
state[i] = HUNGRY; /* record fact that philosopher i is hungry */
test(i); /* try to acquire 2 forks */
up(&mutex); /* exit critical region */
down(&s[i]); /* block if forks were not acquired */
}
void put_forks(i) /* i: philosopher number, from 0 to N1 */
{
down(&mutex); /* enter critical region */
state[i] = THINKING; /* philosopher has finished eating */
test(LEFT); /* see if left neighbor can now eat */
test(RIGHT); /* see if right neighbor can now eat */
up(&mutex); /* exit critical region */
}
void test(i) /* i: philosopher number, from 0 to N1* /
{
if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
{
state[i] = EATING;
up(&s[i]);
}
}
```

2. Readers Writer problems:

The dining philosophers problem is useful for modeling processes that are competing for exclusive access to a limited number of resources, such as I/O devices. Another famous problem is the readers and writers problem which models access to a database (Courtesies et al., 1971).

Imagine, for example, an airline reservation system, with many competing processes wishing to read and write it.

It is acceptable to have multiple processes reading the database at the same time, but if one process is updating (writing) the database, no other process may have access to the database, not even a reader.

The question is how do we program the readers and the writers? One solution is shown below.

Solution:

```
typedef int semaphore; /* use your imagination */
semaphore mutex = 1; /* controls access to 'rc' */
semaphore db = 1; /* controls access to the database */
int rc = 0; /* # of processes reading or wanting to */
void reader(void)
{
while (TRUE){ /* repeat forever */
down(&mutex); /* get exclusive access to 'rc' */
rc = rc + 1; /* one reader more now */
if (rc == 1) down(&db); /* if this is the first reader ... */
up(&mutex); /* release exclusive access to 'rc' */
read_data_base(); /* access the data */
down(&mutex); /* get exclusive access to 'rc' */
rc = rc 1; /* one reader fewer now */
if (rc == 0) up(&db); /* if this is the last reader ... */
up(&mutex); /* release exclusive access to 'rc' */
use_data_read(); /* noncritical region */
}
}
void writer(void)
{
while (TRUE){ /* repeat forever */
think_up_data(); /* noncritical region */
down(&db); /* get exclusive access */
write_data_base(); /* update the data */
up(&db); /* release exclusive access */
}
}
```

In this solution, the first reader to get access to the database does a down on the semaphore database. Subsequent readers merely have to increment a counter. As readers leave, they decrement the counter and the last one out does an up on the semaphore, allowing a blocked writer, if there is one, to get in.

3. Sleeping Barber Problem:

Customers arrive to a barber, if there are no customers the barber sleeps in his chair. If the barber is asleep then the customers must wake him up. The analogy is based upon a hypothetical barber shop with one barber.

The barber has one barber chair and a waiting room with a number of chairs in it. When the barber finishes cutting a customer's hair, he dismisses the customer and then goes to the waiting room to see if there are other customers waiting.

If there are, he brings one of them back to the chair and cuts his hair. If there are no other customers waiting, he returns to his chair and sleeps in it.

Each customer, when he arrives, looks to see what the barber is doing. If the barber is sleeping, then the customer wakes him up and sits in the chair. If the barber is cutting hair, then the customer goes to the waiting room.

If there is a free chair in the waiting room, the customer sits in it and waits his turn. If there is no free chair, then the customer leaves.

Based on a naive analysis, the above description should ensure that the shop functions correctly, with the barber cutting the hair of anyone who arrives until there are no more customers, and then sleeping until the next customer arrives.

In practice, there are a number of problems that can occur that are illustrative of general scheduling problems.

The problems are all related to the fact that the actions by both the barber and the customer (checking the waiting room, entering the shop, taking a waiting room chair, etc.) all take an unknown amount of time.

For example, a customer may arrive and observe that the barber is cutting hair, so he goes to the waiting room. While he is on his way, the barber finishes the haircut he is doing and goes to check the waiting room.

Since there is no one there (the customer not having arrived yet), he goes back to his chair and sleeps. The barber is now waiting for a customer and the customer is waiting for the barber.

In another example, two customers may arrive at the same time when there happens to be a single seat in the waiting room. They observe that the barber is cutting hair, go to the waiting room, and both attempt to occupy the single chair.

Solution:

Many possible solutions are available. The key element of each is a mutex, which ensures that only one of the participants can change state at once.

The barber must acquire this mutex exclusion before checking for customers and release it when he begins either to sleep or cut hair.

A customer must acquire it before entering the shop and release it once he is sitting in either a waiting room chair or the barber chair. This eliminates both of the problems mentioned in the previous section.

A number of semaphores are also required to indicate the state of the system. For example, one might store the number of people in the waiting room.

A multiple sleeping barber's problem has the additional complexity of coordinating several barbers among the waiting customers.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<errno.h>
#include<sys/ipc.h>
#include<semaphore.h>
#define N 5

time_t end_time;/*end time*/
sem_t mutex,customers,barbers;/*Three semaphors*/
int count=0;/*The number of customers waiting for haircuts*/
void barber(void *arg);
void customer(void *arg);
int main(int argc,char *argv[])
{
      pthread_t id1,id2;
      int status=0;
      end_time=time(NULL)+20;/*Barber Shop Hours is 20s*/
      /*Semaphore initialization*/
      sem_init(&mutex,0,1);
      sem_init(&customers,0,0);
      sem_init(&barbers,0,1);
      /*Barber_thread initialization*/
      status=pthread_create(&id1,NULL,(void *)barber,NULL);
      if(status!=0)
            perror("create barbers is failure!\n");
      /*Customer_thread initialization*/
      status=pthread_create(&id2,NULL,(void *)customer,NULL);
      if(status!=0)
            perror("create customers is failure!\n");
      /*Customer_thread first blocked*/
      pthread_join(id2,NULL);
      pthread_join(id1,NULL);
      exit(0);
}
void barber(void *arg)/*Barber Process*/
{
      while(time(NULL)<end_time || count>0)
      {
            sem_wait(&customers);/*P(customers)*/
            sem_wait(&mutex);/*P(mutex)*/
            count--;
            printf("Barber:cut hair,count is:%d.\n",count);
            sem_post(&mutex);/*V(mutex)*/
            sem_post(&barbers);/*V(barbers)*/
            sleep(3);
      }
}
void customer(void *arg)/*Customers Process*/
{
      while(time(NULL)<end_time)
      {
            sem_wait(&mutex);/*P(mutex)*/
            if(count<N)
            {
```

```
                    count++;
                    printf("Customer:add count,count is:%d\n",count);
                    sem_post(&mutex);/*V(mutex)*/
                    sem_post(&customer);/*V(customers)*/
                    sem_wait(&barbers);/*P(barbers)*/
              }
              else
                    /*V(mutex)*/
                    /*If the number is full of customers,just put the mutex
lock let go*/
                    sem_post(&mutex);
              sleep(1);
        }
}
```

Process Scheduling:

When a computer is multi-programmed, it frequently has multiple processes competing for the CPU at the same time. This situation occurs whenever two or more processes are simultaneously in the ready state.

If only one CPU is available, a choice has to be made which process to run next. The part of operating system that makes the choice is called scheduler and the algorithm it uses is called scheduling algorithm.

Types of Scheduling:

In different environment different scheduling algorithms are needed. The situation arises because different application area (different kinds of Operating System) have different goals. The different categories of scheduling algorithms are:

1. Preemptive Scheduling:

A preemptive scheduling algorithm picks a job/process and lets it run for a maximum of some fixed time. If it is still running at the end of the time interval, it is suspended and the scheduler picks another process to run (if one is available).

Such scheduling requires a clock interrupt occur at the end of the time interval to give control of the CPU back to the scheduler. If no clock available, non-preemptive scheduling is only the option.

Example: Round Robin, Preemptive SJF (Shortest Job First), SRTN (Shortest Remaining Time Next), etc.

2. Non-preemptive Scheduling:

Non-preemptive scheduling algorithm picks a process/job to run and then let it run until it blocks (either for input/output or waiting for another process) or until it voluntarily release the CPU.

Even if it runs for hours, it will not be forcefully suspended. After clock interrupt processing has been completed, the process that was running before the interrupt is always resumed.

Example: FCFS (First Come First Serve), SJF (Shortest Job First), etc.

3. Batch Scheduling:

In a batch processing system, there are users waiting at their terminals for a quick response. So, non-preemptive or preemptive algorithms with long time periods are acceptable. It reduces process switch and thus improves performance.

Example: FCFS (First Come First Serve), SJF (Shortest Job First), etc.

4. Interactive Scheduling:

Interactive scheduling policies assign a time-slice to each process. Once the time-slice is over, the process is swapped even if not yet terminated. It can also be said that scheduling of this kind are preemptive.

Batch scheduling policies, instead, are non-preemptive. Once a Process is in the Running-status, it will not change status until it terminates.

Example: Round Robin, Priority, etc.

5. Real-time Scheduling:

A real-time scheduling System is composed of the scheduler, clock and the processing hardware elements.

In a real-time system, a process or task has schedulability; tasks are accepted by a real-time system and completed as specified by the task deadline depending on the characteristic of the scheduling algorithm.

Modeling and evaluation of a real-time scheduling system concern is on the analysis of the algorithm capability to meet a process deadline.

A deadline is defined as the time required for a task to be processed. A task in a real-time system must be completed "neither too early nor too late."

Scheduling Criteria or Performance Analysis:

Goals depends on the environment (batch, interactive or real-time).

1. All System:

Fairness: Giving each process a fair share of CPU.

Balance: Keeping all parts of the system busy.

2. Batch System:

Throughput: Maximize jobs per hour.

Turnaround Time: Minimizing time between submission and termination

CPU Utilization: Keeps the CPU busy all the time.

3. Interactive System:

Response Time: Respond to request quickly

Proportionality: Meets user expectations/how long things should take.

4. Real-time System:

Meeting Deadline: Needs to maintain regular rate

Predictability: Avoid quality degradation in multimedia system.

Scheduling Algorithm:

1. Round Robin:

One of the oldest, simplest, fairest and most widely used algorithm is round robin (RR). In the round robin scheduling, processes are dispatched in a FIFO manner but are given a limited amount of CPU time called a time-slice or a quantum.

If a process does not complete before its CPU-time expires, the CPU is preempted and given to the next process waiting in a queue. The preempted process is then placed at the back of the ready list.

If the process has blocked or finished before the quantum has elapsed the CPU switching is done. Round Robin Scheduling is preemptive (at the end of time-slice) therefore it is effective in timesharing environments in which the system needs to guarantee reasonable response times for interactive users.

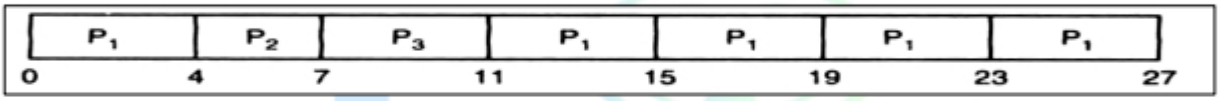
The only interesting issue with round robin scheme is the length of the quantum. Setting the quantum too short causes too many context switches and lower the CPU efficiency.

On the other hand, setting the quantum too long may cause poor response time and approximates FCFS. In any event, the average waiting time under round robin scheduling is on quite long.

Consider the following set of processes that arrives at time 0 ms.

Burst Time	Burst Time
P1	20
P2	3
P3	4

If we use time quantum of 4ms then calculate the average waiting time using R-R scheduling.



According to R-R scheduling processes are executed in FCFS order. So, firstly P1 (burst time=20ms) is executed but after 4ms it is preempted and new process P2 (Burst time = 3ms) starts its execution whose execution is completed before the time quantum.

Then next process P3 (Burst time=4ms) starts its execution and finally remaining part of P1 gets executed with time quantum of 4ms.

- Waiting time of Process P1: 0ms + (11 – 4) ms = 7ms
- Waiting time of Process P2: 4ms
- Waiting time of Process P3: 7ms
- Average Waiting time: (7+4+7)/3=6ms

2. First Come First Serve (FCFS):

FCFS is the simplest non-preemptive algorithm. Processes are assigned the CPU in the order they request it. That is the process that requests the CPU first is allocated the CPU first.

The implementation of FCFS is policy is managed with a FIFO (First in first out) queue. When the first job enters the system from the outside in the morning, it is started immediately and allowed to run as long as it wants to.

As other jobs come in, they are put onto the end of the queue. When the running process blocks, the first process on the queue is run next. When a blocked process becomes ready, like a newly arrived job, it is put on the end of the queue.

Advantages:

a. Easy to understand and program. With this algorithm a single linked list keeps track of all ready processes.
b. Equally fair.
c. Suitable especially for Batch Operating system.

Disadvantages:

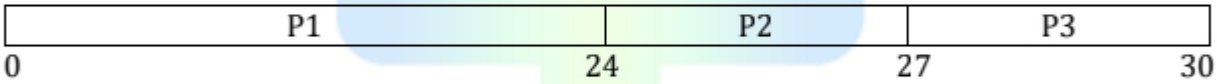
a. FCFS is not suitable for time-sharing systems where it is important that each user should get the CPU for an equal amount of arrival time.

Consider the following set of processes having their burst time mentioned in milliseconds. CPU burst time indicates that for how much time, the process needs the CPU.

Process	Burst Time
P1	24
P2	3
P3	3

Calculate The Average Waiting Time If The Processes Arrive In The Order Of:

The processes arrive the order P1, P2, P3. Let us assume they arrive in the same time at 0 ms in the system. We get the following gantt chart.



Waiting time for P1= 0ms, for P2 = 24 ms for P3 = 27ms

Avg waiting time: (0+24+27)/3= 17

3. Shortest Job First (SJF):

When several equally important jobs are sitting in the input queue waiting to be started, the scheduler picks the shortest jobs first. The disadvantages of this algorithm is the problem to know the length of time for which CPU is needed by a process. The SJF is optimal when all the jobs are available simultaneously.

The SJF is either preemptive or non-preemptive. Preemptive SJF scheduling is sometimes called **Shortest Remaining Time First** scheduling. With this scheduling algorithms the scheduler always chooses the process whose remaining run time is shortest.

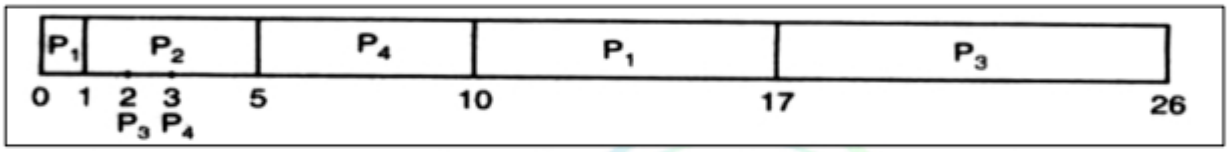
When a new job arrives its total time is compared to the current process remaining time. If the new job needs less time to finish than the current process, the current process is

suspended and the new job is started. This scheme allows new short jobs to get good service. Note: SJF Default: (Non Preemptive).

Calculate The Average Waiting Time In:

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

a. Preemptive SJF (Shortest Remaining Time First):



At t=0ms only one process P1 is in the system, whose burst time is 8ms; starts its execution. After 1ms i.e., at t=1, new process P2 (Burst time= 4ms) arrives in the ready queue.

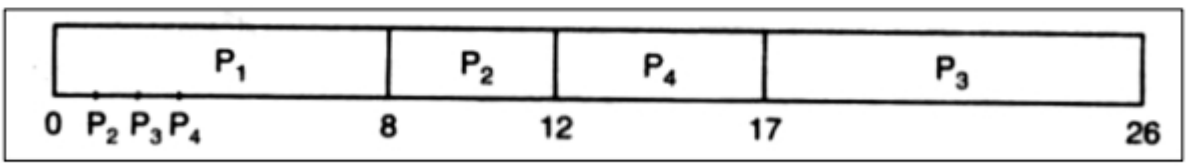
Since its burst time is less than the remaining burst time of P1 (7ms) P1 is preempted and execution of P2 is started.

Again at t=2, a new process P3 arrive in the ready queue but its burst time (9ms) is larger than remaining burst time of currently running process (P2 3ms). So P2 is not preempted and continues its execution.

Again at t=3, new process P4 (burst time 5ms) arrives. Again for same reason P2 is not preempted until its execution is completed.

- Waiting time of P1: 0ms + (10 – 1) ms = 9ms
- Waiting time of P2: 1ms – 1ms = 0ms
- Waiting time of P3: 17ms – 2ms = 15ms
- Waiting time of P4: 5ms – 3ms = 2ms
- Avg waiting time: (9+0+15+2)/4 = 6.5ms

b. Non-Preemptive SJF:



Since its non-preemptive process is not preempted until it finish its execution.

- Waiting time for P1: 0ms
- Waiting time for P2: (8-1)ms = 7ms
- Waiting time for P3: (17 – 2) ms = 15ms

- Waiting time for P4: (12 – 3)ms = 9ms
- Average waiting time: (0+7+15+9)/4 = 7.75ms

4. Shortest Remaining Time Next (SRTN):

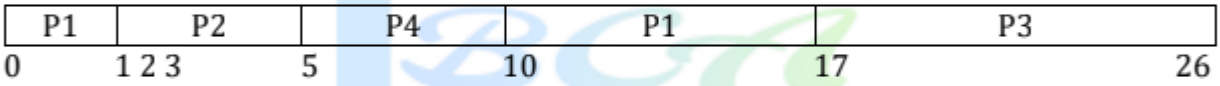
It is preemptive version of SJF. The scheduler always chooses the process whose remaining run time is the shortest.

Here, alos the runtime need to be known in advance. When a new job arrives, its total time is compared to the current process remaining time. If the new job needs less time to finish than the current process, the current process is suspended and the new job started.

This scheme allows new short jobs to get good service.

Calculate The Waiting And Turnover Time:

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5



Waiting time for: P1 = 9 ms, P2 = 0 ms, P3 = 15 ms, P4 = 2 ms

Turnover time for: P1 = 17ms, P2 = 4 ms, P3 = 24ms, P4 = 7 ms

5. Priority Scheduling:

Round Robin algorithm makes the implicit assumption that all processes are equally important. But, sometimes multi-users computers has to take external factor into account that lead to priority.

Idea is: "Each process is assigned a priority and the runnable process with the highest priority is allowed to run"

6. Fair-share Scheduling:

Fair-share scheduling is a scheduling algorithm for computer operating systems in which the CPU usage is equally distributed among system users or groups, as opposed to equal distribution among processes.

One common method of logically implementing the fair-share scheduling strategy is to recursively apply the round-robin scheduling strategy at each level of abstraction

(processes, users, groups, etc.) The time quantum required by round-robin is arbitrary, as any equal division of time will produce the same results.

For example, if four users (A,B,C,D) are concurrently executing one process each, the scheduler will logically divide the available CPU cycles such that each user gets 25% of the whole (100% / 4 = 25%).

If user B starts a second process, each user will still receive 25% of the total cycles, but each of user B's processes will now be attributed 12.5% of the total CPU cycles each, totaling user B's fair share of 25%.

On the other hand, if a new user starts a process on the system, the scheduler will reapportion the available CPU cycles such that each user gets 20% of the whole (100% / 5 = 20%).

7. Shortest Process Next:

Shortest job next (SJN), also known as shortest job first (SJF) or shortest process next (SPN), is a scheduling policy that selects for execution the waiting process with the smallest execution time. SJN is a non-preemptive algorithm. Shortest remaining time is a preemptive variant of SJN.

Shortest job next is advantageous because of its simplicity and because it minimizes the average amount of time each process has to wait until its execution is complete. However, it has the potential for process starvation for processes which will require a long time to complete if short processes are continually added. Highest response ratio next is similar but provides a solution to this problem using a technique called aging.

Another disadvantage of using shortest job next is that the total execution time of a job must be known before execution. While it is impossible to predict execution time perfectly, several methods can be used to estimate it, such as a weighted average of previous execution times.

Shortest job next can be effectively used with interactive processes which generally follow a pattern of alternating between waiting for a command and executing it. If the execution burst of a process is regarded as a separate "job", past behavior can indicate which process to run next, based on an estimate of its running time.

Shortest job next is used in specialized environments where accurate estimates of running time are available.

8. Lottery Scheduling:

Lottery Scheduling is a probabilistic scheduling algorithm for processes in an operating system. Processes are each assigned some number of lottery tickets, and the scheduler draws a random ticket to select the next process.

The distribution of tickets need not be uniform; granting a process more tickets provides it a relative higher chance of selection. This technique can be used to approximate other scheduling algorithms, such as shortest job next and Fair-share scheduling.

Lottery scheduling solves the problem of starvation. Giving each process at least one lottery ticket guarantees that it has non-zero probability of being selected at each scheduling operation.

Implementations of lottery scheduling should take into consideration that there could be billions of tickets distributed among a large pool of threads.

To have an array where each index represents a ticket, and each location contains the thread corresponding to that ticket, may be highly inefficient. Lottery scheduling can be preemptive or non-preemptive.

9. Guaranteed Scheduling:

Guaranteed scheduling makes real promises to the users about performance. If there are n users logged in while you are working, you will receive about 1 /n of the CPU power.

Similarly, on a single-user system with n processes running, all things being equal, each one should get 1 /n of the CPU cycles. To make good on this promise, the system must keep track of how much CPU each process has had since its creation.

CPU Time entitled = (Time Since Creation)/n

Then compute the ratio of Actual CPU time consumed to the CPU time entitled. A ratio of 0.5 means that a process has only had half of what it should have had, and a ratio of 2.0 means that a process has had twice as much as it was entitled to.

The algorithm is then to run the process with the lowest ratio until its ratio has moved above its closest competitor.

10. Scheduling in Real Time System:

Time is crucial and plays an essential role. Example: The computer in a compact disc player gets the bits as they come off the drive and must convert them into music with a very tight time interval.

If the calculation takes too long the music sounds peculiar. Other example includes:

- Auto pilot in Aircraft
- Robot control in automated factory.
- Patient monitoringin Factory. (ICU)

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

Types:

1. ***Hard Real Time system:*** There are absolute deadline that must be met.
2. ***Soft Real Time system***: Missing an occasional deadline is undesirable but nevertheless tolerable.

In both cases real time behavior is achieved by dividing the program into a no. of processes, each of whose behavior is predictable & known in advance. These processes

are short lived and can run to completion. It's the job of schedulers to schedule the process in such a way that all deadlines are met.

If there are m periodic events and event i occurs with period Pi and requires Ci seconds of CPU time to handle each event, then the load can only be handled if A real-time system that meets this criteria is said to be schedulable.

11. Multilevel Queuing:

Multi-level queue scheduling algorithm is used in scenarios where the processes can be classified into groups based on property like process type, CPU time, I/O access, memory size, etc.

One general classification of the processes is foreground processes and background processes. In a multi-level queue scheduling algorithm, there will be 'n' number of queues, where 'n' is the number of groups the processes are classified into.

Each queue will be assigned a priority and will have its own scheduling algorithm like Round-robin scheduling or FCFS.

For the process in a queue to execute, all the queues of priority higher than it should be empty, meaning the process in those high priority queues should have completed its execution.

In this scheduling algorithm, once assigned to a queue, the process will not move to any other queues.

12. Multilevel Feedback Queue:

Unlike multilevel queue scheduling algorithm where processes are permanently assigned to a queue, multilevel feedback queue scheduling allows a process to move between queues.

This movement is facilitated by the characteristic of the CPU burst of the process. If a process uses too much CPU time, it will be moved to a lower-priority queue. This scheme leaves I/O-bound and interactive processes in the higher priority queues.

In addition, a process that waits too long in a lower-priority queue may be moved to a higher priority queue. This form of aging also helps to prevent starvation of certain lower priority processes.

13. Highest Response Ration Next (HRN):

Highest response ratio next (HRRN) scheduling is a non-preemptive discipline, similar to shortest job next (SJN), in which the priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting.

Jobs gain higher priority the longer they wait, which prevents indefinite postponement (process starvation). In fact, the jobs that have spent a long time waiting compete against those estimated to have short run times.

$$\text{Priority} = \frac{waiting\ time + estimated\ run\ time}{estimated\ run\ time} = 1 + \frac{waiting\ time}{estimated\ run\ time}$$

Developed by Brinch Hansen to correct certain weaknesses in Shortest job next including the difficulty in estimating run time.

Unit IV: Deadlock – Operating System

System Model:

We know that computer systems are full of resources that can only be used by one process at a time. The resources includes printers, drivers, and slots in system's internal table.

Having two processes simultaneously reading/writing to the resources creates a problem. For many applications, a process needs exclusive access to not one resources but several. For Example:

Two processes each want to record scanned document on a CD. Process "A" request permission to use scanner and is granted. Process "B" is programmed and requests the CD recorder first and is also granted.

Now, process "A" asks for the recorder but the request is denied until process "B" release it. Unfortunately, instead of releasing the CD recorder process "B" asks for the scanner. At this point both processes are blocked and will remain so forever. This situation is called as deadlock.

System Resources:

A computer typically has many different resources. In some cases there may be many instances of a resource of a given type (e.g. buffers), all of which are equivalent. A process needing one of these resources can use any one of them.

In other cases there may be only one instance of a resource (e.g. CD-ROM drive with a particular CD). There are two types of system resources:

1. Preemptable:

A preemptable resource is one which can be allocated to a given process for a period of time, then be allocated to another process and then be reallocated to the first process without any ill effects. Examples of preemptable resources include:

- Memory
- Buffers
- CPU
- Array processor

2. Non-preemptable:

A **nonpreemptable resource** cannot be taken from one process and given to another without side effects. One obvious example is a printer: certainly we would not want to take the printer away from one process and give it to another in the middle of a print job.

(Actually, the operating system treated printers as preemptable resources, the operators would have to sort the printed output to reassemble print jobs.)

As we shall see, deadlocks usually involve nonpreemptable resources. The usual sequence of events that occur as a resource is used is:

1. **Request the resource**. One of two things can happen when a resource is requested: the request can be granted immediately (if the resource is available) or it can be postponed (or blocked) until a later time.
2. **Use the resource**. Once the resource has been acquired, it can be used.
3. **Release the resource**. When the process no longer needs the resource it releases it. Usually it is released as soon as possible but in most systems there is nothing to enforce this policy.

Conditions for Resources Deadlock:

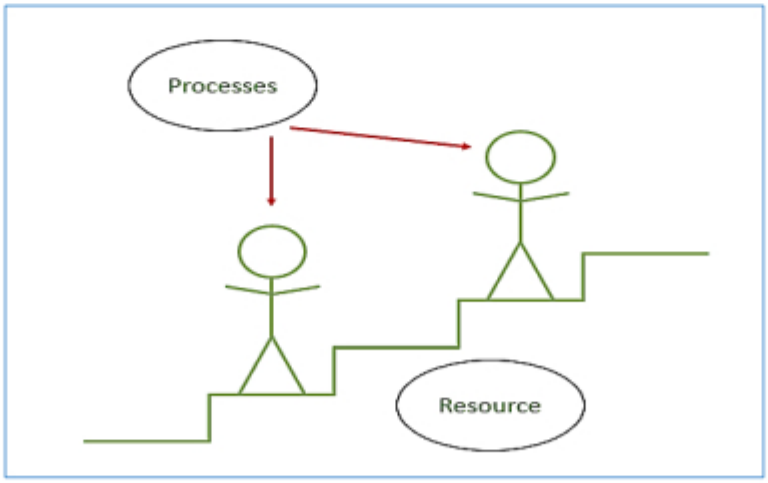
Deadlock is a situation which involves the interaction of more than one resources and processes with each other.

We can visualize the occurrence of deadlock as a situation where there are two people on a staircase. One is ascending the staircase while the other is descending. The staircase is so narrow that it can only fit one person at a time.

As a result, one has to retreat while the others moves on and uses the staircase. Once that person is finished, the other one can use that staircase. But here, none of the people is willing to retreat and waits for the each other to retreat.

None of them is able to use the staircase. The people here is the process and the staircase is the resource.

When a process requests for the resource that is been held another process which needs another resource to continue, but is been held by the first process, then it is called a **deadlock**.



There are 4 conditions necessary for the occurrence of a deadlock. They can be understood with the help of the above illustrated example of staircase:

1. Mutual Exclusion:

When two people meet in the landings, they can't just walk through because there is space only for one person. This condition to allow only one person (or process) to use the step between them (or the resource) is the first condition necessary for the occurrence of the deadlock.

2. Hold and Wait:

When the 2 people refuses to retreat and hold their grounds, it is called holding. This is the next necessary condition for the deadlock.

3. No Preemption:

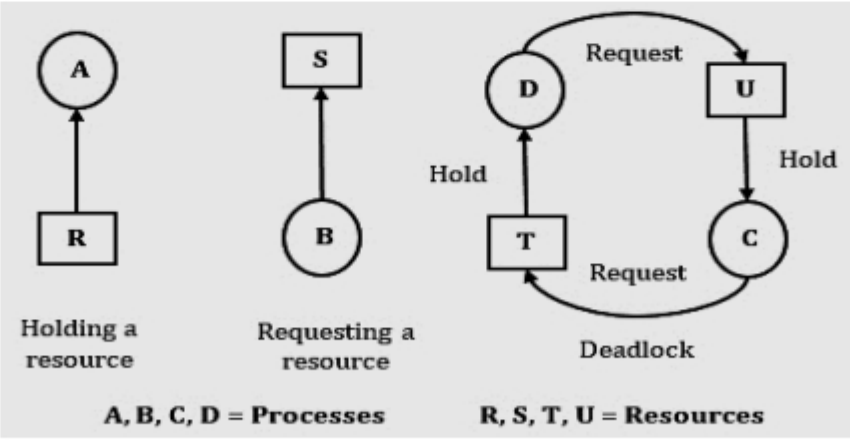
For resolving the deadlock one can simply cancel one of the processes for other to continue. But Operating System doesn't do so. It allocates the resources to the processors for as much time needed until the task is completed. Hence, there is no temporary reallocation of the resources. It is third condition for deadlock.

4. Circular Wait:

When the two people refuses to retreat and wait for each other to retreat, so that they can complete their task, it is called circular wait. It is the last condition for the deadlock to occur.

Note: All the 4 conditions are necessary for the deadlock to occur. If anyone is prevented or resolved, the deadlock is resolved.

Deadlock Modeling:



Hold (1972), showed how the above four conditions can be modeled using directed graphs. The graph has two kinds of nodes:

a. Processes (denoted by circles)
b. Resources (denoted by squares)

A directed arc from resource node (square) to a process node (circle) means that the request has previously been requested by granted to and is currently hold by that process.

A directed arc from process node (circle) to resource node (square) means that the process is requesting for resource.

Ostrich Algorithm:

In computer science, the ostrich algorithm is a strategy of ignoring potential problems on the basis that they may be exceedingly rare. It is named for the ostrich effect which is defined as "to stick one's head in the sand and pretend there is no problem."

It is used when it is more cost-effective to allow the problem to occur than to attempt its prevention.

This approach may be used in dealing with deadlocks in concurrent programming if they are believed to be very rare and the cost of detection or prevention is high. Different people react to this strategy in different ways.

Mathematicians find it totally unacceptable and say that deadlocks must be prevented at all costs. Engineers ask how often the problem is expected, how often the system crashes for other reasons, and how serious a deadlock is.

The Ostrich algorithm pretends there is no problem and is reasonable to use if deadlocks occur very rarely and the cost of their prevention would be high. The UNIX and Windows operating systems take this approach.

Although using the Ostrich algorithm is one of the methods of dealing with deadlocks, other effective methods exist such as dynamic avoidance, banker's algorithm, detection and recovery, and prevention.

Methods of Handling Deadlocks:

Generally speaking there are three ways of handling deadlocks:

1. **Deadlock Prevention or Avoidance:** Do not allow the system to get into a deadlocked state.
2. **Deadlock Detection and Recovery:** Abort a process or preempt some resources when deadlocks are detected.
3. **Ignore The Problem All Together:** If deadlocks only occur once a year or so, it may be better to simply let them happen and reboot as necessary than to incur the constant overhead and system performance penalties associated with deadlock prevention or detection. This is the approach that both Windows and UNIX take.

In order to avoid deadlocks, the system must have additional information about all processes. In particular, the system must know what resources a process will or may request in the future.

(Ranging from a simple worst-case maximum to a complete resource request and release plan for each process, depending on the particular algorithm.)

Deadlock detection is fairly straightforward, but deadlock recovery requires either aborting processes or preempting resources, neither of which is an attractive alternative.

If deadlocks are neither prevented nor detected, then when a deadlock occurs the system will gradually slow down, as more and more processes become stuck waiting for resources currently held by the deadlock and by other waiting processes.

Unfortunately this slowdown can be indistinguishable from a general system slowdown when a real-time process has heavy computing needs.

Deadlock Prevention:

Havender in his pioneering work showed that since all four of the conditions are necessary for deadlock to occur, it follows that deadlock might be prevented by denying any one of the conditions.

1. Elimination of "Mutual Exclusion" Condition:

The mutual exclusion condition must hold for non-sharable resources. That is, several processes cannot simultaneously share a single resource.

This condition is difficult to eliminate because some resources, such as the tap drive and printer, are inherently non-shareable. Note that shareable resources like read-only-file do not require mutually exclusive access and thus cannot be involved in deadlock.

2. Elimination of "Hold and Wait" Condition:

There are two possibilities for elimination of the second condition.

The first alternative is that a process request be granted all of the resources it needs at once, prior to execution. The second alternative is to disallow a process from requesting resources whenever it has previously allocated resources.

This strategy requires that all of the resources a process will need must be requested at once. The system must grant resources on "all or none" basis. If the complete set of resources needed by a process is not currently available, then the process must wait until the complete set is available.

While the process waits, however, it may not hold any resources. Thus the "wait for" condition is denied and deadlocks simply cannot occur. This strategy can lead to serious waste of resources.

For example, a program requiring ten tap drives must request and receive all ten derives before it begins executing. If the program needs only one tap drive to begin execution and then does not need the remaining tap drives for several hours.

Then substantial computer resources (9 tape drives) will sit idle for several hours. This strategy can cause indefinite postponement (starvation). Since not all the required resources may become available at once.

3. Elimination of "No-preemption" Condition:

The non-preemption condition can be alleviated by forcing a process waiting for a resource that cannot immediately be allocated to relinquish all of its currently held resources, so that other processes may use them to finish.

Suppose a system does allow processes to hold resources while requesting additional resources. Consider what happens when a request cannot be satisfied.

A process holds resources a second process may need in order to proceed while second process may hold the resources needed by the first process. This is a deadlock. This strategy require that when a process that is holding some resources is denied a request for additional resources.

The process must release its held resources and, if necessary, request them again together with additional resources. Implementation of this strategy denies the "no-preemptive" condition effectively.

High Cost When a process release resources the process may lose all its work to that point. One serious consequence of this strategy is the possibility of indefinite postponement (starvation). A process might be held off indefinitely as it repeatedly requests and releases the same resources.

4. Elimination of "Circular Wait" Condition:

The last condition, the circular wait, can be denied by imposing a total ordering on all of the resource types and then forcing, all processes to request the resources in order (increasing or decreasing).

This strategy impose a total ordering of all resources types, and to require that each process requests resources in a numerical order (increasing or decreasing) of enumeration. With this rule, the resource allocation graph can never have a cycle.

For example, provide a global numbering of all the resources, as shown

1 ≡ Card reader
2 ≡ Printer
3 ≡ Plotter
4 ≡ Tape drive
5 ≡ Card punch

Now the rule is this: processes can request resources whenever they want to, but all requests must be made in numerical order.

A process may request first printer and then a tape drive (order: 2, 4), but it may not request first a plotter and then a printer (order: 3, 2). The problem with this strategy is that it may be impossible to find an ordering that satisfies everyone.

Deadlock Avoidance:

The main algorithm for doing deadlock avoidance are based on the concept of safe states. A state is said to be safe, if there is some scheduling order in which every process can run to completion even if all of them suddenly requests maximum number of resources immediately.

Example:

Let "A", "B" and "C" are processes. "Has" means the number of resources process in holding and "Max" is maximum number of resources that process needs. Let the available resources are 10.

Process	Has	Max
A	3	9
B	2	4
C	2	7
Free: 3		

Process	Has	Max
A	3	9
B	4	4
C	2	7
Free: 1		

Process	Has	Max
A	3	9
B	0	4
C	2	7
Free: 5		

Process	Has	Max
A	3	9
B	-	-
C	7	7
Free: 0		

Process	Has	Max
A	3	9
B	-	4
C	-	-
Free: 7		

Process	Has	Max
A	9	9
B	-	-
C	-	-
Free: 1		

Process	Has	Max
A	-	-
B	-	-
C	-	-
Free: 10		

Banker's Algorithm (Single Resource):

The most famous deadlock avoidance algorithm, due to Dijkstra [1965], is the Banker's algorithm. It is modeled on the way a small town banker might deal with a group of customers to whom he has granted lines of credit.

The algorithm checks to see if granting the resources leads to an unsafe state, the request is denied. If granting the request leads to safe state, then it is carried out.

Here,

- Customers = Processes
- Units (lines of credit) = Resources
- Banker/teller = Operating system

Process	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7
Free: 10 (Safe)		

Process	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7
Free: 2 (Safe)		

Process	Has	Max
A	1	6
B	2	5
C	2	4
	4	7
Free: 1 (Unsafe)		

Banker's Algorithm (Multiple Resources):

The banker algorithm can be generalized to handle multiple resources:

1. Look for a row "R", whose unmet resources (current allocation resources) are smaller than or equal to "A" (available resources). If no such rows exist, the system will eventually deadlock since no process can run to completion.
2. Assume the process of the row chosen request all the resources it needs (which is granted to be possible) and finishes. Mark that process as terminated and add all its resources to "A" vector.
3. Repeat step 1 and step 2 until all processes are marked as terminated, in which case the initial state was safe or until a deadlock occurs in which case it was not.

Example:

Process	Tape Driver	Plotter	Scanner	CD-ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0
Current Allocation Matrix				

Process	Tape Driver	Plotter	Scanner	CD-ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0
Request Matrix				

In the table above:

- Existing Resource = 6, 3, 4, 2
- Processed Resource = 5, 3, 2, 2
- Available Resource = 1, 0, 2, 0

Solution:

- Process "A", "B", "C" cannot run to completion. Since for each process request is greater than available resources. Now, check for process "D", it can complete so the available resources are:
- (1, 0, 2, 0) + (1, 1, 0, 1) = Available resources (2, 1, 2, 1)
- Again, process "E" can run to completion. So, resources available (2, 1, 2, 1)
- Again, Process "A" can run to completion. So,
- (2, 1, 2, 1)+(3, 0, 1, 1) = Available resources (5, 1, 3, 2)
- Again, Process "B" can run to completion. So,
- (5, 1, 3, 2)+(0, 1, 0, 0) = Available resources (5, 2, 3, 2)
- Again, Process C can run to completion. So,
- (5, 2, 3, 2) + (1, 1, 1, 0) = Available resource (6, 3, 4, 2)
- Lastly the schedule is: "D"-"E"-"A"-"B"-"C"

Deadlock Detection:

Deadlock detection is the process of actually determining that a deadlock exists and identifying the processes and resources involved in the deadlock. Once a deadlock is detected, there needs to be a way to recover several alternatives exists:

1. Temporarily prevent resources from deadlocked processes.
2. Back off a process to some check point allowing preemption of a needed resource and restarting the process at the checkpoint later.
3. Successively kill processes until the system is deadlock free.

These methods are expensive in the sense that each iteration calls the detection algorithm until the system proves to be deadlock free. The complexity of algorithm is $O(N^2)$ where N is the number of proceeds. Another potential problem is starvation; same process killed repeatedly.

In order to detect deadlock, we can use resource allocation graph. Imagine, we have three processes "A", "B", "C" and three resources "R", "S", "T". The request and release of three processes are given as:

Process "A"	Process "B"	Process "C"
Request R	Request S	Request T
Request S	Request T	Request R

Release R	Release S	Release T
Release S	Release T	Release R
No Deadlock		

Let's make a graph for the following:

Process "A"	**Process "B"**	**Process "C"**
Request R	Request S	Request T
Request S	Request T	Request R
Deadlock		

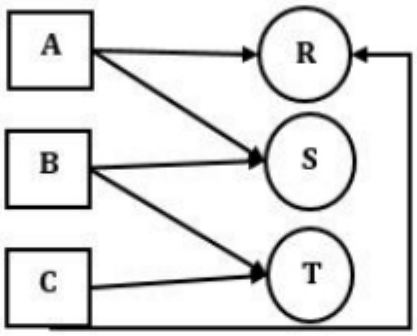


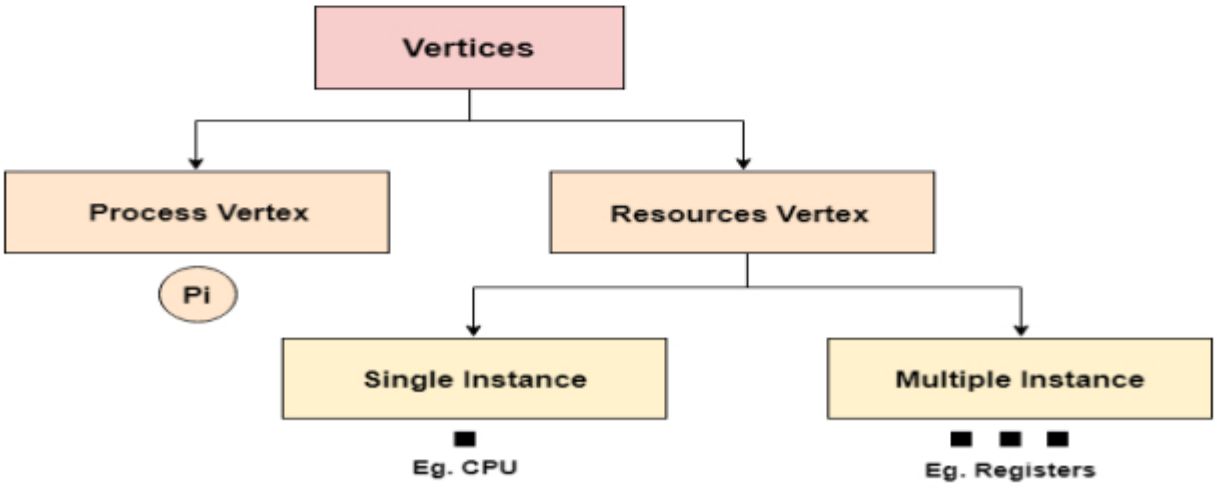
Fig: Deadlock

Resource Allocation Graph:

The resource allocation graph is the pictorial representation of the state of a system. As its name suggests, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources.

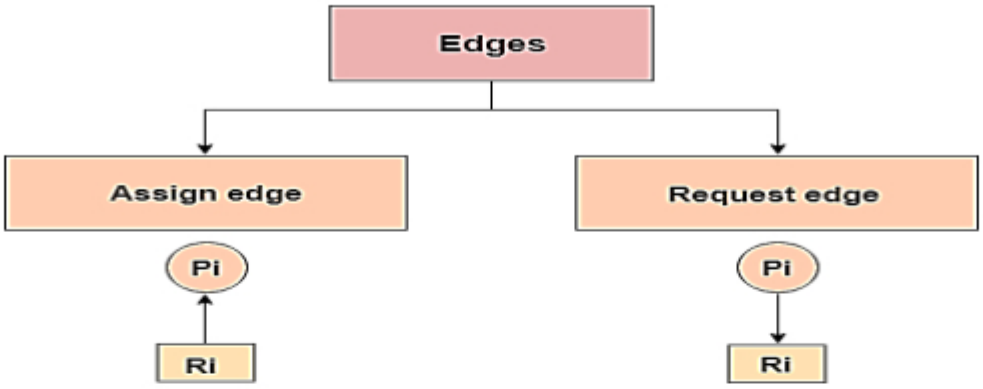
It also contains the information about all the instances of all the resources whether they are available or being used by the processes.

In Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle. Let's see the types of vertices and edges in detail.



Vertices are mainly of two types, Resource and process. Each of them will be represented by a different shape. Circle represents process while rectangle represents resource.

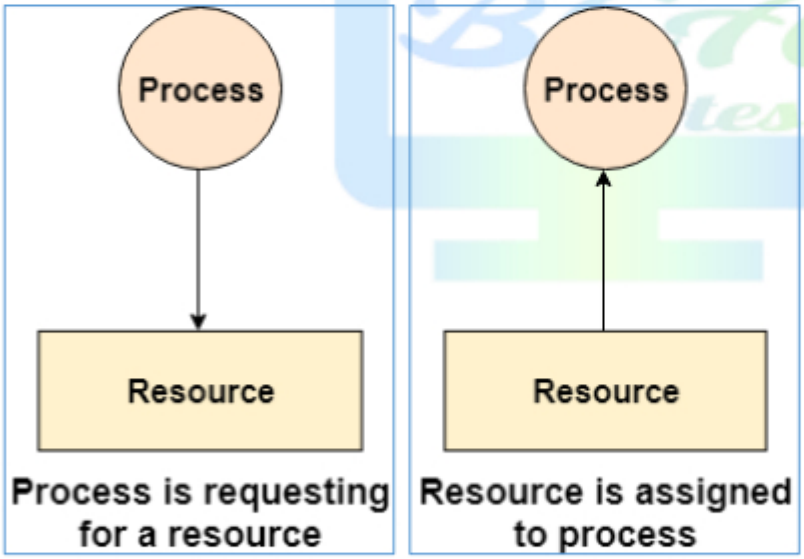
A resource can have more than one instance. Each instance will be represented by a dot inside the rectangle.



Edges in RAG are also of two types, one represents assignment and other represents the wait of a process for a resource. The above image shows each of them.

A resource is shown as assigned to a process if the tail of the arrow is attached to an instance to the resource and the head is attached to a process.

A process is shown as waiting for a resource if the tail of an arrow is attached to the process while the head is pointing towards the resource.

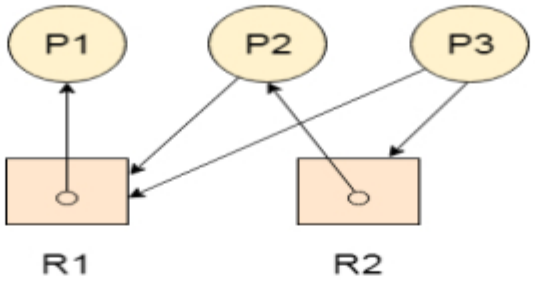


Example:

Let's consider 3 processes P1, P2 and P3, and two types of resources R1 and R2. The resources are having 1 instance each.

According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 is waiting for R1 as well as R2.

The graph is deadlock free since no cycle is being formed in the graph.



Recovery from Deadlock:

Suppose that our deadlock detection algorithm has succeeded and detected a deadlock. Some way is needed to recover and get the system going again. So, here we have some ways to recover from deadlock and they are:

1. Recovery Through Preemption:

In some cases, it may be possible to temporarily take a resource away from its current owner and give it to other process. For example:

To take a laser printer away from its owner, the operator can collect all the sheets already printed and put them in a pile. Then the process can be suspended (marked as not runnable).

At this point the printer can be assigned to another process. When the process finishes, the pile of sheets can be put back in the printer output tray and the process restarted. It depends upon the nature of resources.

2. Recovery Through Rollback:

In this approach processes are check pointed periodically (i.e. its state is written in file so that it can be restarted later). The checkpoint contains not only memory image but also the resource state i.e. which resources are currently assigned to which process.

When deadlock occurs, it is easy to see which resources are needed. To do the recovery, a process that owns a needed resource is rolled back to a point in time before it acquired the resource.

3. Recovery Through Killing Processes:

It is the simplest way to break a deadlock. In this approach we kill one or more processes. By killing a process in a cycle if other process get chance to continue then the deadlock is

ended but, if the processes did not continue, we again start to kill processes until the cycle get broken.

Starvation:

Starvation is closely related to deadlock. At certain condition, some processes never get resources even though they are not deadlocked, these processes goes on starvation due to which they get lost or dead.

For example: Allocation of a printer through a file (i.e. shortest file to print). If new files are continually being added, the huge file will starve to death (postponed indefinitely even though it is not deadlocked).

Some of the common causes of starvation are as follows:

- If a process is never provided the resources it requires for execution because of faulty resource allocation decisions, then starvation can occur.
- A lower priority process may wait forever if higher priority processes constantly monopolize the processor.
- Starvation may occur if there are not enough resources to provide to every process as required.
- If random selection of processes is used then a process may wait for a long time because of non-selection.

Some solutions that can be implemented in a system to handle starvation are as follows:

- An independent manager can be used for allocation of resources. This resource manager distributes resources fairly and tries to avoid starvation.
- Random selection of processes for resource allocation or processor allocation should be avoided as they encourage starvation.
- The priority scheme of resource allocation should include concepts such as aging, where the priority of a process is increased the longer it waits. This avoids starvation.

Difference between Deadlock and Starvation:

Deadlock	Starvation
All processes keep waiting for each other to complete and none get executed	High priority processes keep executing and low priority processes are blocked
Resources are blocked by the processes	Resources are continuously utilized by high priority processes

Necessary conditions Mutual Exclusion, Hold and Wait, No preemption, Circular Wait	Priorities are assigned to the processes
Also known as Circular wait	Also known as lived lock
It can be prevented by avoiding the necessary conditions for deadlock	It can be prevented by Aging

Unit V: Memory Management – Operating System

Basic Memory Management:

Introduction:

We know that main memory (RAM) is an important resource that must be carefully managed. Nowadays size of memory is heavily increased and programs are getting faster and bigger than memory.

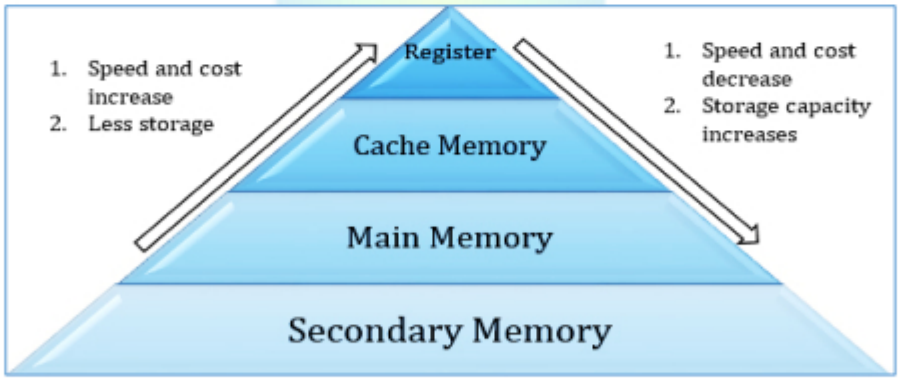
As memory size increases, programs also expands to fill them. Memory management deals with how operating system manage them. Once a program is loaded into memory, it remains there until it finishes.

Some operating system supports only one process in memory while other supports multi-programming.

Memory Hierarchy:

Every programmer like private, infinitely large and fast memory that is also non-volatile and inexpensive. Unfortunately, technology does not provide such memories at present. The choice is over the years people discovered the concept of memory hierarchy.

In memory hierarchy, computers have a few megabytes of very fast, expensive cache memory, a few gigabytes of medium speed, medium priced, volatile main memory and a few terabytes of slow, cheap, non-volatile disk storage and many removable storages such as DVDs, USB, etc.



It is the job of operating system to abstract this hierarchy into a useful model and then manage the abstraction. The part of operating system that manage the memory hierarchy is called the memory manager. Its job is to efficiently manage memory such as:

a. Keeps tracks of which parts of memory are in use.
b. Allocate memory to processes when they need it and deadlock it when they are done.

Logical Address VS Physical Address:

The address generated by the CPU is a ***logical address***, whereas the address actually seen by the memory hardware is a ***physical address***. Addresses bound at compile time or load time have identical logical and physical addresses.

Addresses created at execution time, however, have different logical and physical addresses. In this case the logical address is also known as a ***virtual address***, and the two terms are used interchangeably by our text.

The set of all logical addresses used by a program composes the ***logical address space***, and the set of all corresponding physical addresses composes the ***physical address space.***

The run time mapping of logical to physical addresses is handled by the ***memory-management unit, MMU***. The MMU can take on many forms. One of the simplest is a modification of the base-register scheme described earlier.

The base register is now termed a ***relocation register***, whose value is added to every memory request at the hardware level.

Note that user programs never see physical addresses. User programs work entirely in logical address space, and any memory references or manipulations are done using purely logical addresses. Only when the address gets sent to the physical memory chips is the physical memory address generated.

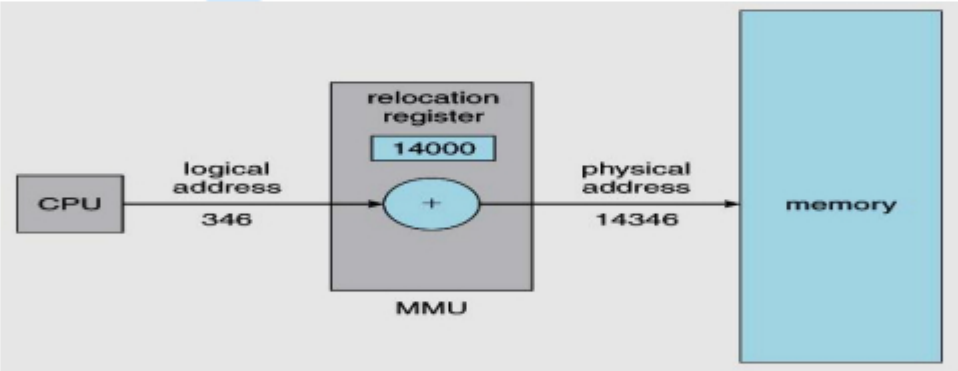


Fig: Dynamic Relocation Using a Relocation Register

Memory Management with Swapping:

Memory management can be divided into two classes these that more processes back and forth between main memory and disk during execution. More process back and forth between main memory and disk is known as swapping and paging.

1. Memory Management with Bitmaps:

When memory is assigned dynamically, the operating system must manage it. With a bitmap, memory is divided up into allocation units, perhaps as small as a few words and perhaps as large as several kilobytes.

Corresponding to each allocation unit is a bit in the bitmap, which is 0 if the unit is free and 1 if it is occupied (or vice versa). Figure below shows part of memory and the corresponding bitmap.

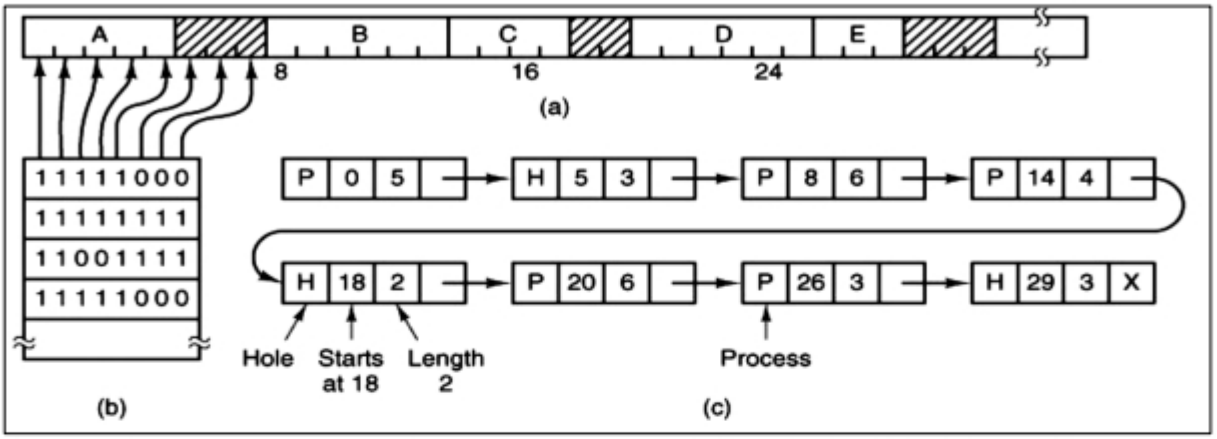


Fig: (a) A part of memory with five processes and three holes. The tick marks show the memory allocation units. The shaded regions (0 in the bitmap) are free. (b) The corresponding bitmap. (c) The same information as a list.

The smaller the allocation unit, the larger the bitmap. However, even with an allocation unit as smaller as 4 bytes, 32-bits of memory will require only 1-bit of the map.

If the allocation unit will be chosen large, the bitmap will be smaller but appreciable memory may be wasted in the last unit of process if the process size is not an exact multiple of allocation unit.

2. Memory Management with Linked List:

Another way to keep track of memory is to maintain a linked list of allocated and free memory segments as shown in figure (c), where a segment is either a Process (P) or a Hole (H) between two processes.

Each entry in the list specifies a Process (P) or a Hole (H), the address at which it starts, the length and a pointer to the next entry.

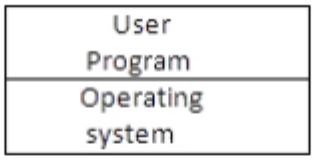
Memory Management without Swapping:

It allows programs to run when they partially in main memory. It uses concepts of virtual memory. Virtual memory addresses the program larger than memory.

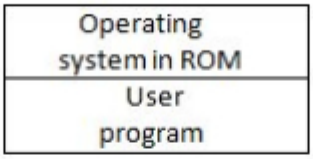
1. Mono Programming Without Swapping Or Paging:

This is the simplest memory management in which just one program is run at a time, sharing the memory between the program and the operating system. Three variables are used in this schemes:

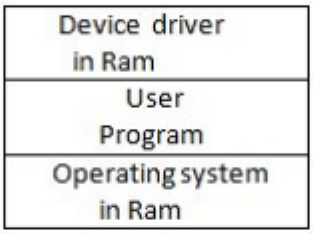
a. The Operating System May Be At The Bottom Of Memory In Ram.



b. The Operating System May Be In ROM At Top Of Memory.



c. Device Driver May Be At The Top Of The RAM.



This model is used in MS-DOS system. In this type of system only one process at a time can be operated or executed.

As soon as user type command the operating system copies the request program from the disk to the memory and execute it when the processes finishes, the operating system displays the prompt character and wait for a new command when it receive the command, it load a new program in memory and overload the first one.

2. Multiprogramming With Fix Partition:

It is often useful to have more than one process in memory at a time. The partitioning can be done manually by the operator. When the system to started up when the job carries, it can be put into the input queue for the small partition large enough to hold it.

Since the partition are fixed any space in two partition not used by a job is lost. The difference between fixed memory partition with separate input queues and single input queue can be shown as:

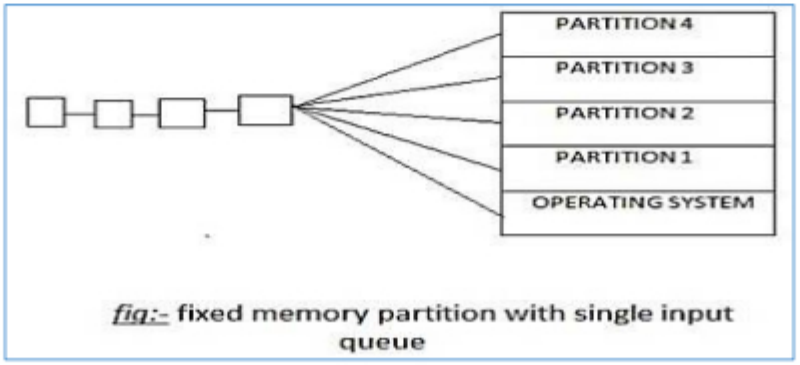


fig:- fixed memory partition with single input queue

The disadvantage of sorting the incoming job into separate queues becomes apparent when the queues for large partition is empty out the queue for small partition is full. But in care of single queues, whenever partitioning becomes free, the job closest to front of the queue that fits in it could be loaded into empty partition and run.

Contiguous Memory Allocation:

In contiguous memory allocation each process is contained in a single contiguous block of memory. Memory is divided into several fixed size partitions. Each partition contains exactly one process.

When a partition is free, a process is selected from the input queue and loaded into it. The free blocks of memory are known as holes. The set of holes is searched to determine which hole is best to allocate.

1. Memory Protection:

Memory protection is a phenomenon by which we control memory access rights on a computer. The main aim of it is to prevent a process from accessing memory that has not been allocated to it.

Hence prevents a bug within a process from affecting other processes, or the operating system itself, and instead results in a segmentation fault or storage violation exception being sent to the disturbing process, generally killing of process.

2. Memory Allocation:

Memory allocation is a process by which computer programs are assigned memory or space. It is of three types:

- **First Fit:** The first hole that is big enough is allocated to program.
- **Best Fit:** The smallest hole that is big enough is allocated to program.
- **Worst Fit:** The largest hole that is big enough is allocated to program.

3. Fragmentation:

Fragmentation occurs in a dynamic memory allocation system when most of the free blocks are too small to satisfy any request. It is generally termed as inability to use the available memory.

In such situation processes are loaded and removed from the memory. As a result of this, free holes exists to satisfy a request but is non-contiguous i.e. the memory is fragmented into large no. Of small holes. This phenomenon is known as **External Fragmentation**.

Also, at times the physical memory is broken into fixed size blocks and memory is allocated in unit of block sizes. The memory allocated to a space may be slightly larger than the requested memory.

The difference between allocated and required memory is known as **internal fragmentation** i.e. the memory that is internal to a partition but is of no use.

Non-Contiguous Memory Allocation:

Fragmentation is a main problem is contiguous memory allocation. We have seen a method called compaction to resolve this problem. Since it's an I/O operation system efficiency gets reduced.

So, a better method to overcome the fragmentation problem is to make our logical address space non-contiguous.

Consider a system in which before applying compaction, there are holes of size 1K and 2K. If a new process of size 3K wants to be executed then its execution is not possible without compaction.

An alternative approach is divide the size of new process P into two chunks of 1K and 2K to be able to load them into two holes at different places.

1. If the chunks have to be of same size for all processes ready for the execution then the memory management scheme is called PAGING.
2. If the chunks have to be of different size in which process image is divided into logical segments of different sizes then this method is called SEGMENTATION.
3. If the method can work with only some chunks in the main memory and the remaining on the disk which can be brought into main memory only when it's required, then the system is called VIRTUAL MEORY MANAGEMENT SYSTEM.

Memory Partitioning:

1. Fixed Partitioning:

Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size. Memory Manager will allocate a region to a process that best fits it. Unused memory within an allocated partition called internal fragmentation.

Advantages:

- Simple to implement
- Little OS overhead

Disadvantages:

Inefficient use of Memory due to internal fragmentation. Main memory utilization is extremely inefficient. Any program, no matter how small, occupies an entire partition. This phenomenon, in which there is wasted space internal to a partition due to the fact

that the block of data loaded is smaller than the partition, is referred to as internal fragmentation.

Possibilities:

❖ Equal-Size Partitions:

If there is an available partition, a process can be loaded into that partition because all partitions are of equal size, it does not matter which partition is used. If all partitions are occupied by blocked processes, choose one process to swap out to make room for the new process.

❖ Unequal-Size Partitions: Use Of Multiple Queues:

It assigns each process to the smallest partition within which it will fit. It allocates a queue for each partition size. It tries to minimize internal fragmentation. In this partition some queues will be empty if no processes within a size range is present.

❖ Unequal-Size Partitions: Use Of A Single Queue:

When it's time to load a process into main memory the smallest available partition that will hold the process is selected. It increases the level of multiprogramming at the expense of internal fragmentation.

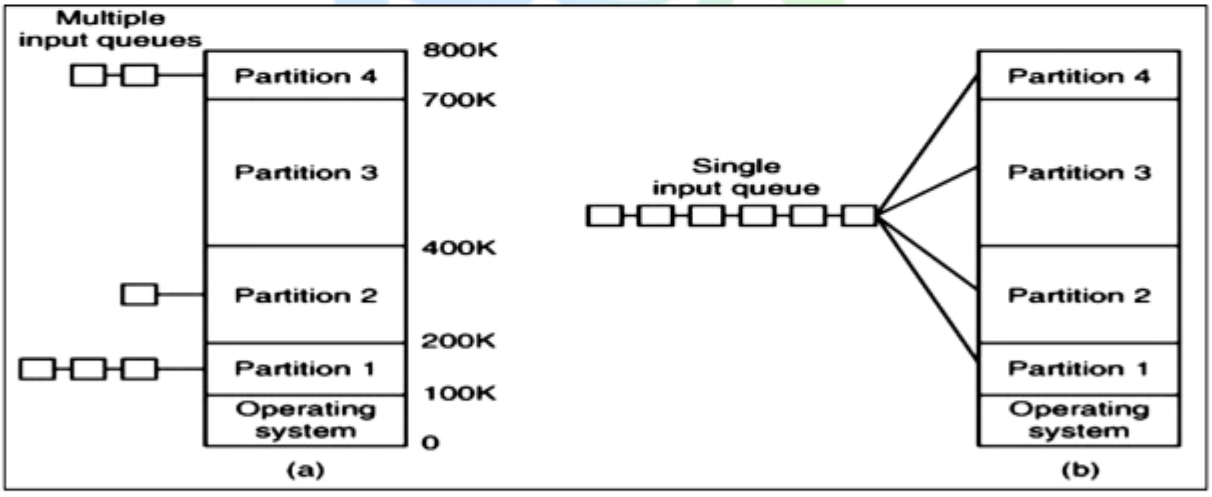


Fig: (a) Fixed memory partitions with separate input queues for each partition. (b) Fixed memory partitions with a single input queue.

When the queue for a large partition is empty but the queue for a small partition is full, as in the case for partitions 1 and 3. Here small jobs have to wait to get into memory, even though plenty of memory is free.

An alternative organization is to maintain a single queue as in Fig. (b) Whenever a partition becomes free, the job closest to the front of the queue that fits in it could be loaded into the empty partition and run.

2. Dynamic/Variable Partitioning:

To overcome some of the difficulties with fixed partitioning, an approach known as dynamic partitioning was developed. The partitions are of variable length and number.

When a process is brought into main memory, it is allocated exactly as much memory as it requires and no more. An example, using 64 Mbytes of main memory, is shown in Figure below:

Eventually it leads to a situation in which there are a lot of small holes in memory. As time goes on, memory becomes more and more fragmented, and memory utilization declines. This phenomenon is referred to as external fragmentation, indicating that the memory that is external to all partitions becomes increasingly fragmented.

One technique for overcoming external fragmentation is compaction: From time to time, the operating system shifts the processes so that they are contiguous and so that all of the free memory is together in one block.

For example, in Figure h, compaction will result in a block of free memory of length 16M. This may well be sufficient to load in an additional process. The difficulty with compaction is that it is a time consuming procedure and wasteful of processor time.

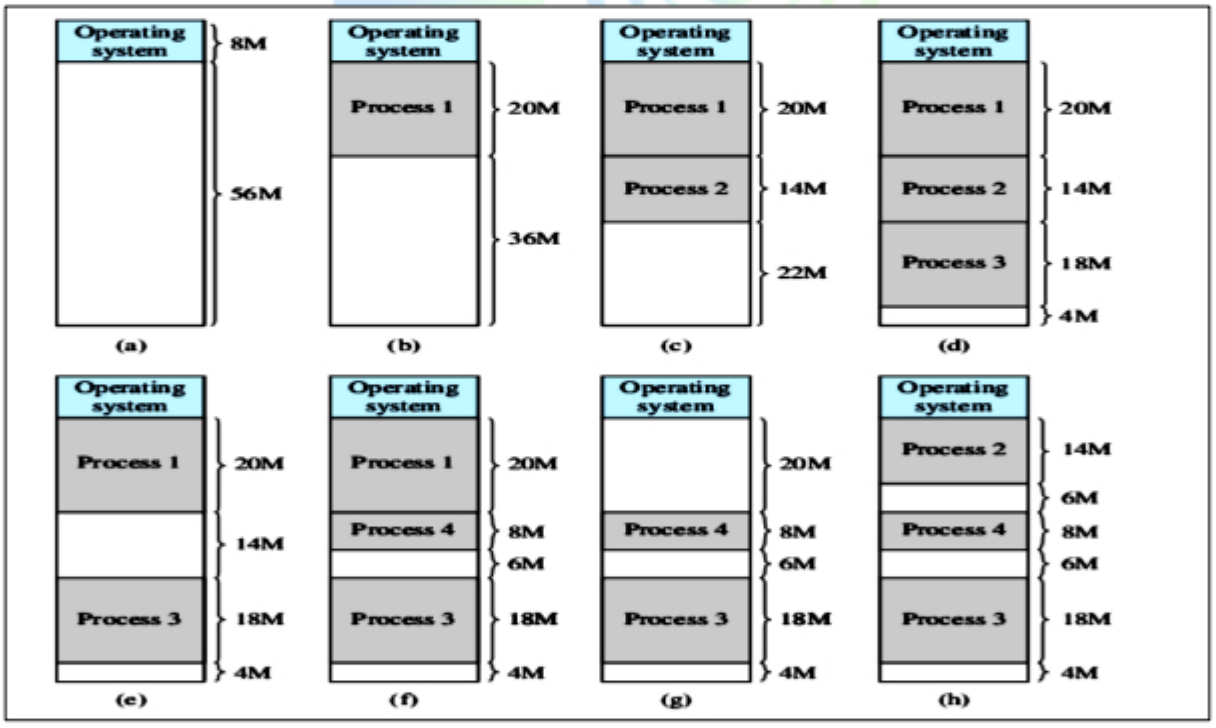


Fig. The Effect of dynamic partitioning

Memory Management Requirements:

1. Relocation:

- Programmer cannot know where the program will be placed in memory when it is executed

- Process may be (often) relocated in main memory due to swapping
- Swapping enables the OS to have a larger pool of ready-to-execute processes
- Memory references in code (for both instructions and data) must be translated to actual physical memory address

2. Protection:

- Processes should not be able to reference memory locations in another process without permission
- Impossible to check addresses at compile time in programs since the program could be relocated
- Address references must be checked at run time by hardware

3. Sharing:

- Must allow several processes to access a common portion of main memory without compromising protection
- Cooperating processes may need to share access to the same data structure
- Better to allow each process to access the same copy of the program rather than have their own separate copy

4. Logical Organization:

- Users write programs in modules with different characteristics
- Instruction modules are execute-only
- Data modules are either read-only or read/write
- Some modules are private others are public
- To effectively deal with user programs, the OS and hardware should support a basic form of module to provide the required protection and sharing

5. Physical Organization:

- Secondary memory is the long term store for programs and data while main memory holds program and data currently in use
- Moving information between these two levels of memory is a major concern of memory management (OS)
- It is highly inefficient to leave this responsibility to the application programmer

Coalescing and Compaction:

1. Coalescing:

In computer science, coalescing is the act of merging two adjacent free blocks of memory. When an application frees memory, gaps can fall in the memory segment that the

application uses. Among other techniques, coalescing is used to reduce external fragmentation, but is not totally effective.

Coalescing can be done as soon as blocks are freed, or it can be deferred until sometime later (known as deferred coalescing), or it might not be done at all. Coalescence and related techniques like heap compaction, can be used in garbage collection.

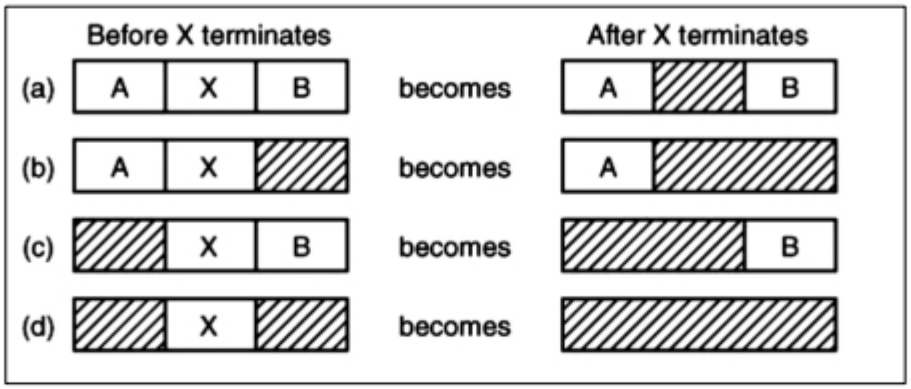


Fig: Coalescing Process

2. Compaction:

When swapping creates a multiple holes in memory it is possible to combine them all into one big one by moving all the processes downwards as far as possible. This technique is known as compaction.

It is usually not done because it requires a lot of CPU time. Compaction is used in implementation of segments.

Virtual Memory:

Background:

Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory.

This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.

In real scenarios, most processes never need all their pages at once, for following reasons:

1. Error handling code is not needed unless that specific error occurs, some of which are quite rare.
2. Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
3. Certain features of certain programs are rarely used.

Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Benefits of Having Virtual Memory:

- Large programs can be written, as virtual space available is huge compared to physical memory.
- Less I/O required, leads to faster and easy swapping of processes.
- More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory

Paging:

A solution to fragmentation problem is paging. Paging is a memory management mechanism that allows the physical address space of a process to be non-contagious.

Here physical memory is divided into blocks of equal size called Pages (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

The pages belonging to a certain process are loaded into available memory frames.

Advantages and Disadvantages of Paging

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

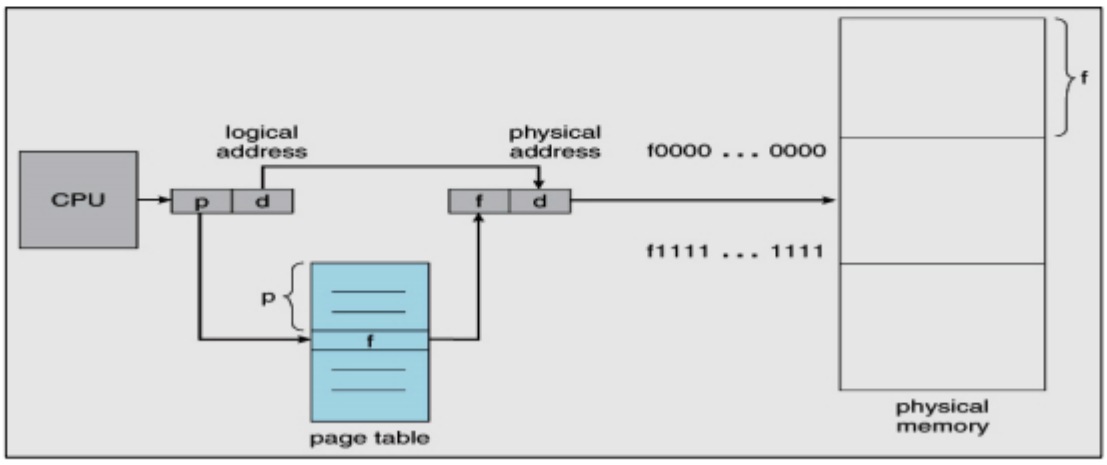


Fig: Paging Hardware

Fig: Paging Model

Page Table:

A Page Table is the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual address and physical addresses. Virtual address is also known as logical address and is generated by the CPU. While Physical address is the address that actually exists on memory.

Address Translation:

Page address is called **logical address** and represented by **page number** and the **offset**.

- Logical Address = Page number + page offset

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

- Physical Address = Frame number + page offset

A data structure called **page map table** is used to keep track of the relation between pages of a process to a frame in physical memory.

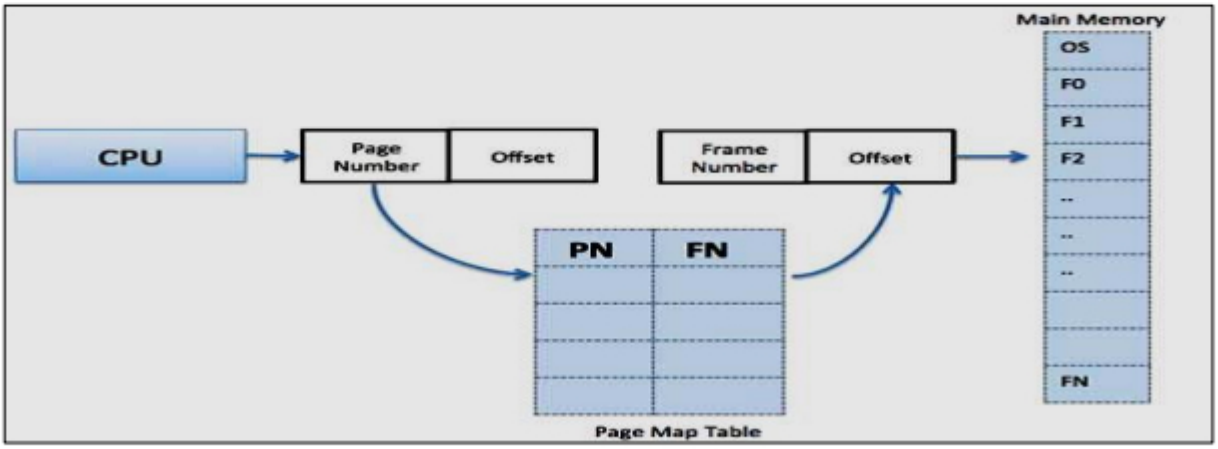


Fig: Page Table

When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture.

When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Structure of Page Table:

1. Hierarchical Paging:

Most modern computer systems support logical address spaces of 2^32 to 2^64. With a 2^32 address space and 4K (2^12) page sizes, this leave 2^20 entries in the page table.

At 4 bytes per entry, this amounts to a 4 MB page table, which is too large to reasonably keep in contiguous memory. (And to swap in and out of memory with each process switch.) Note that with 4K pages, this would take 1024 pages just to hold the page table!

One option is to use a two-tier paging system, i.e. to page the page table. For example, the 20 bits described above could be broken down into two 10-bit page numbers. The first identifies an entry in the outer page table, which identifies where in memory to find one page of an inner page table.

The second 10 bits finds a specific entry in that inner page table, which in turn identifies a particular frame in physical memory. (The remaining 12 bits of the 32 bit logical address are the offset within the 4K frame.)

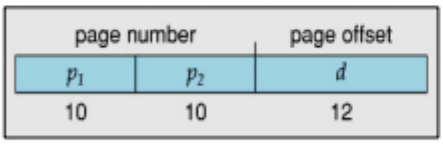


Fig: Two Level Page Number Offset

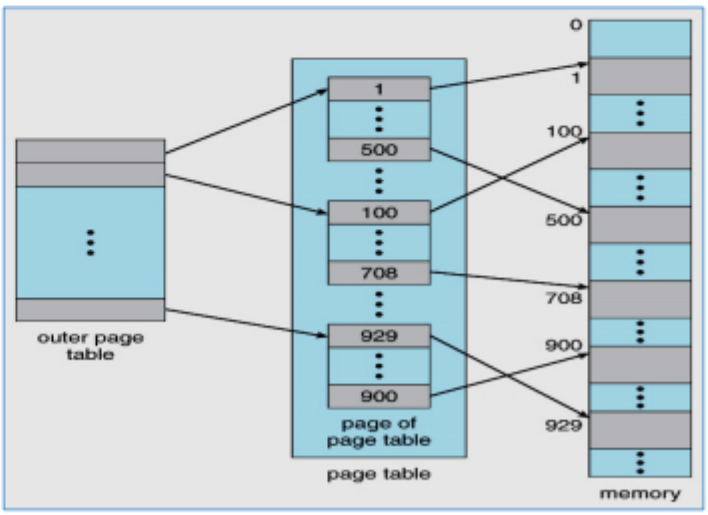


Fig: Two Level Page Table Scheme

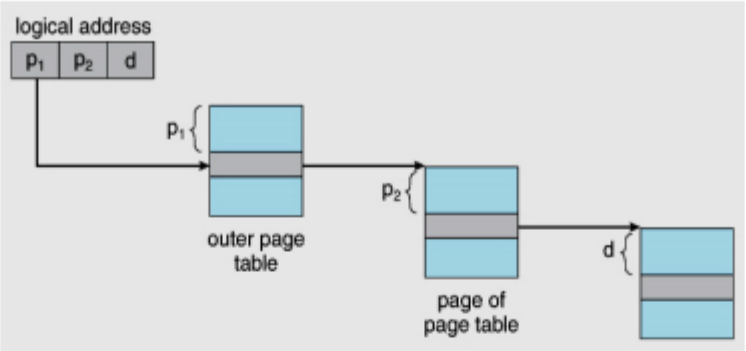


Fig: Address Translation for a Two-Level 32-Bit Paging Architecture

VAX Architecture divides 32-bit addresses into 4 equal sized sections, and each page is 512 bytes, yielding an address form of:

section	page	offset
s	*p*	*d*
2	21	9

Fig: VAX Address

With a 64-bit logical address space and 4K pages, there are 52 bits worth of page numbers, which is still too many even for two-level paging. One could increase the paging level, but with 10-bit page tables it would take 7 levels of indirection, which would be prohibitively slow memory access. So some other approach must be used.

outer page	inner page	offset
p_1	p_2	d
42	10	12

Fig: 64-Bits Two-Tiered Leaves 42 Bits in Outer Table

2nd outer page	outer page	inner page	offset
p_1	p_2	p_3	d
32	10	10	12

Fig: Going To A Fourth Level Still Leaves 32 Bits in the Outer Table.

2. Hashed Page Tables:

One common data structure for accessing data that is sparsely distributed over a broad range of possible values is with hash tables. Figure below illustrates a hashed page table using chain-and-bucket hashing.

Virtual page numbers are compared in this chain searching for a match. If a match is found, the corresponding physical frame is extracted.

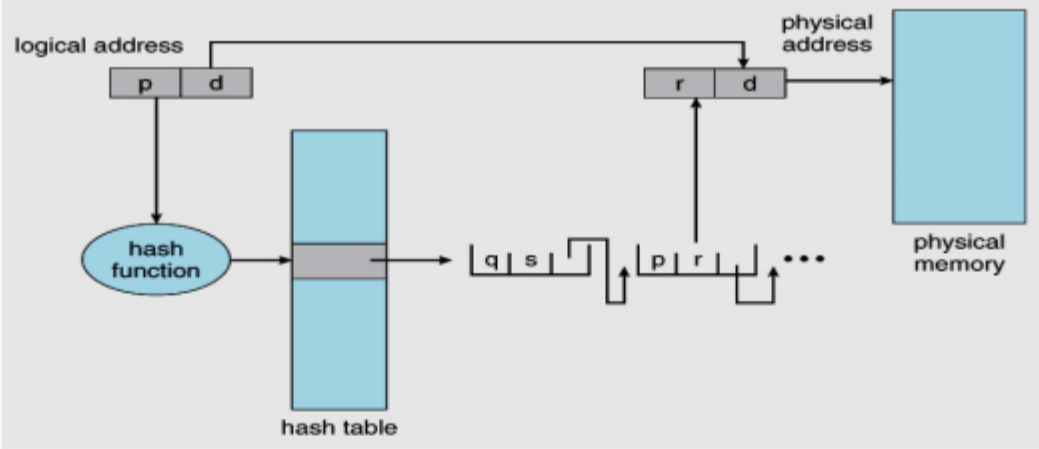


Fig: Hashed Page Table

3. Inverted Page Tables:

Another approach is to use an inverted page table. Instead of a table listing all of the pages for a particular process, an inverted page table lists all of the pages currently loaded in memory, for all processes. (I.e. there is one entry per frame instead of one entry per page.)

Access to an inverted page table can be slow, as it may be necessary to search the entire table in order to find the desired page (or to discover that it is not there.) Hashing the table can help speed up the search process.

Inverted page tables prohibit the normal method of implementing shared memory, which is to map multiple logical pages to a common physical frame. (Because each frame is now mapped to one and only one process.)

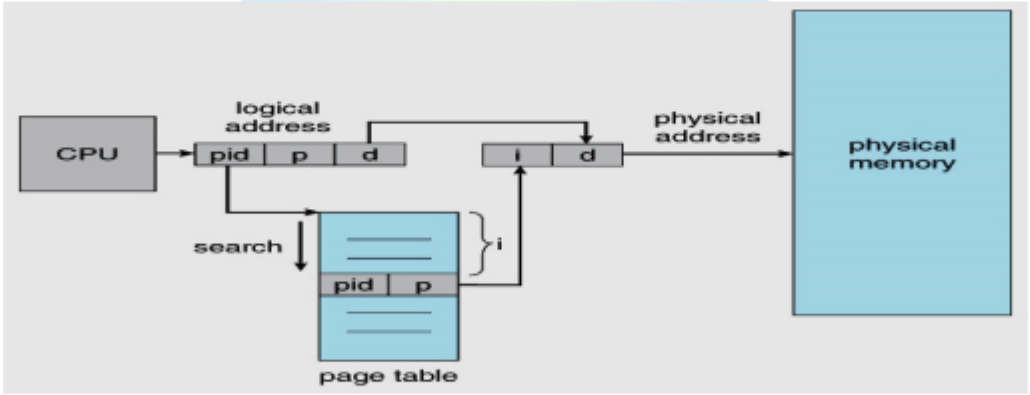


Fig: Inverted Page Table

4. Shared Pages:

Paging systems can make it very easy to share blocks of memory, by simply duplicating page numbers in multiple page frames. This may be done with either code or data.

If code is reentrant, that means that it does not write to or change the code in any way (it is none self-modifying), and it is therefore safe to re-enter it. More importantly, it means

the code can be shared by multiple processes, so long as each has their own copy of the data and registers, including the instruction register.

In the example given below, three different users are running the editor simultaneously, but the code is only loaded into memory (in the page frames) one time.

Some systems also implement shared memory in this fashion.

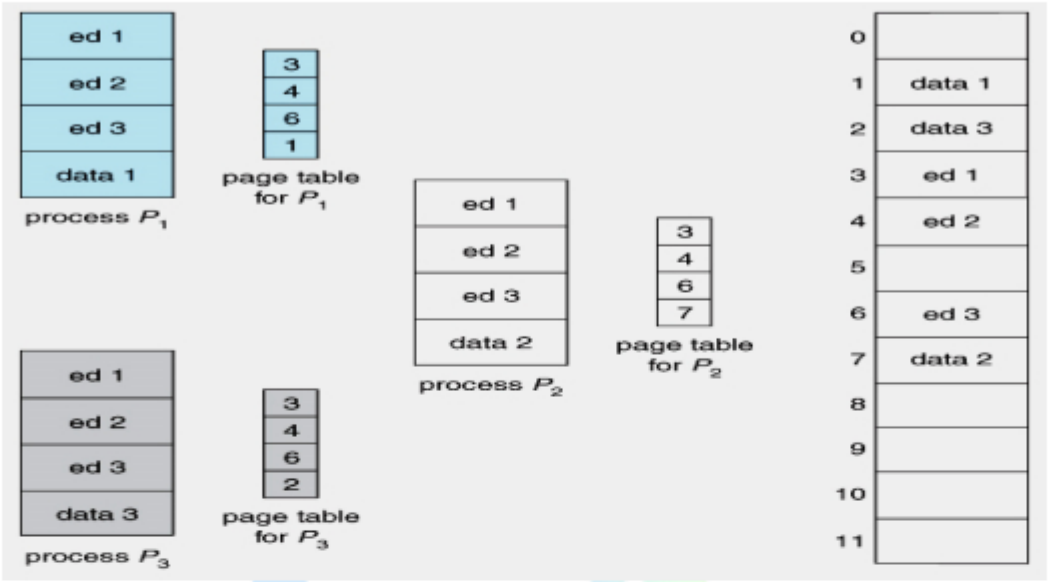


Fig: Sharing Of Code in a Paging Environment

Demand Paging:

The basic idea behind demand paging is that when a process is swapped in, its pages are not swapped in all at once. Rather they are swapped in only when the process needs them (On demand). This is termed as lazy swapper, although a pager is a more accurate term.

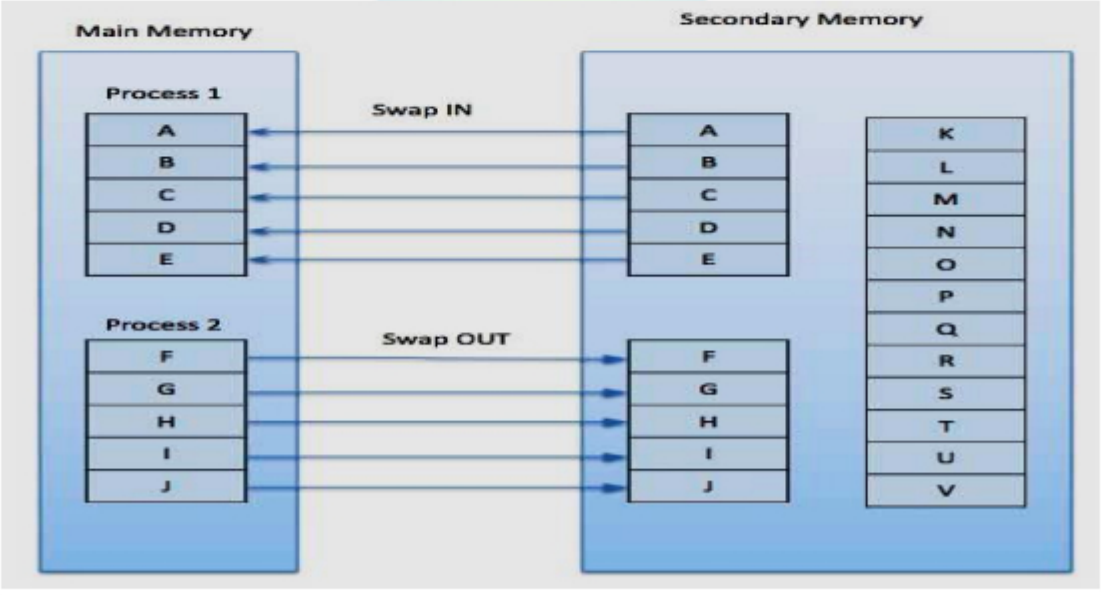


Fig: Demand Paging

Initially only those pages are loaded which will be required the process immediately.

The pages that are not moved into the memory, are marked as invalid in the page table. For an invalid entry the rest of the table is empty. In case of pages that are loaded in the memory, they are marked as valid along with the information about where to find the swapped out page.

When the process requires any of the page that is not loaded into the memory, a page fault trap is triggered and following steps are followed:

1. The memory address which is requested by the process is first checked, to verify the request made by the process.
2. If it found to be invalid, the process is terminated.
3. In case the request by the process is valid, a free frame is located, possibly from a free-frame list, where the required page will be moved.
4. A new operation is scheduled to move the necessary page from disk to the specified memory location. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)
5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to valid.
6. The instruction that caused the page fault must now be restarted from the beginning.

There are cases when no pages are loaded into the memory initially, pages are only loaded when demanded by the process by generating page faults. This is called **Pure Demand Paging**.

The only major issue with Demand Paging is, after a new page is loaded, the process starts execution from the beginning. It's not a big issue for small programs, but for larger programs it affects performance drastically.

Advantages:

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages:

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

Page Fault:

If the program is of five pages 1, 2, 3, 4, 5 and out of these pages only page 1, 4, and 2 are loaded into main memory at the time of execution.

If the program tries to address the address of page number 3 then, there will be an error, as this page is not swapped into main memory at the time of loading the program. This error is called as page fault error.

Page Replacement:

As studied in Demand Paging, only certain pages of a process are loaded initially into the memory. This allows us to get more number of processes into the memory at the same time.

But what happens when a process requests for more pages and no free memory is available to bring them in. Following steps can be taken to deal with this problem:

1. Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.
2. Or, remove some other process completely from the memory to free frames.
3. Or, find some pages that are not being used right now, move them to the disk to get free frames. This technique is called **Page replacement** and is most commonly used. We have some great algorithms to carry on page replacement efficiently.

Page Replacement Algorithm:

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated.

Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself.

There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults.

1. First In First Out (FIFO):

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.
- Performance of this algorithm is not always good because, although the page being the oldest one might be heavily used page and may be needed immediately.

Reference String:	7	0	1	2	0	3	0	4	2	3	0	3

Calculation of Page Fault:

7	0	1	2	2	3	0	4	2	3	0	0
	7	0	1	1	2	3	0	4	2	3	3
		7	0	0	1	2	3	0	4	2	2
×	×	×	×	✓	×	×	×	×	×	×	✓

Page Fault = 10; Page Fault Rate = 10/12 = 0.833

2. Optimal Page Replacement (OPR) Algorithm:

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Reference String:	7	0	1	2	0	3	0	4	2	3	0	3

Calculation of Page Fault:

7	0	1	2	2	2	2	2	2	2	0	0
	7	0	1	1	3	3	3	3	3	3	3
		7	0	0	0	0	4	4	4	4	4
×	×	×	×	✓	×	✓	×	✓	✓	×	✓

Page Fault = 7; Page Fault Rate = 7/12 = 0.5833

3. Least Recently Used (LRU):

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Reference String:	7	0	1	2	0	3	0	4	2	3	0	3

Calculation of Page Fault:

7	0	1	2	2	2	2	4	4	4	0	0
	7	0	1	1	3	3	3	2	2	2	2
		7	0	0	0	0	0	0	3	3	3
×	×	×	×	✓	×	✓	×	×	×	×	✓

Page Fault = 9; Page Fault Rate = 9/12 = 0.75

4. Second Chance Page (SCP) Replacement Algorithm:

A simple modification to FIFO that avoids the problem of throwing out a heavily used page is to inspect R-bit (Reference bit) of the oldest page. If it is 0, the page is both old and unused, so it is replaced immediately.

If the R-bit is 1, the bit is cleared, the page is put onto the end of the list of pages and its load time is updated as though it had just arrived in memory. Then the search continues.

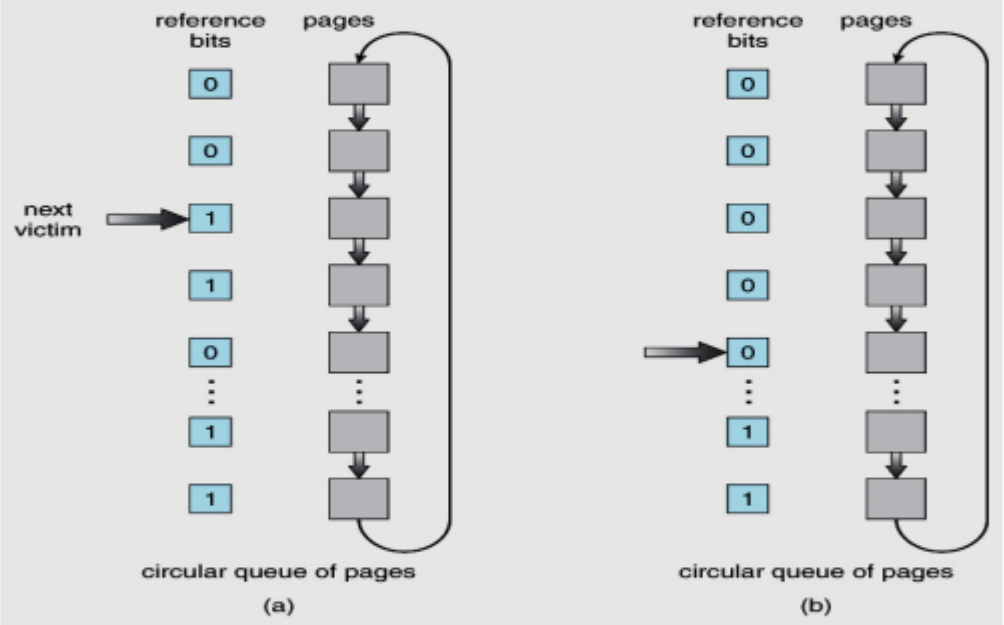


Fig: Second Chance Page (SCP) Replacement Algorithm

5. Page Buffering Algorithm:

- To get a process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.
- Write the new page in the frame of free pool, mark the page table and restart the process.
- Now write the dirty page out of disk and place the frame holding replaced page in free pool.

6. Least Frequently Used (LFU) Algorithm:

- The page with the smallest count is the one which will be selected for replacement.
- This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

7. Most Frequently Used(MFU) Algorithm:

- This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Trashing:

The set of pages that a process is currently using is called its working set. If the entire working set is in memory, the process will run without causing many page fault until it moves into another execution phase.

But, if the available memory is too small to hold the entire working set, the process will cause many page faults and runs slowly. Since, execution and instruction take a few milliseconds and reading in page from disk will take long time just like 10 milliseconds.

A program causing page faults every few instructions is said to be trashing. To prevent thrashing we must provide processes with as many frames as they really need "right now".

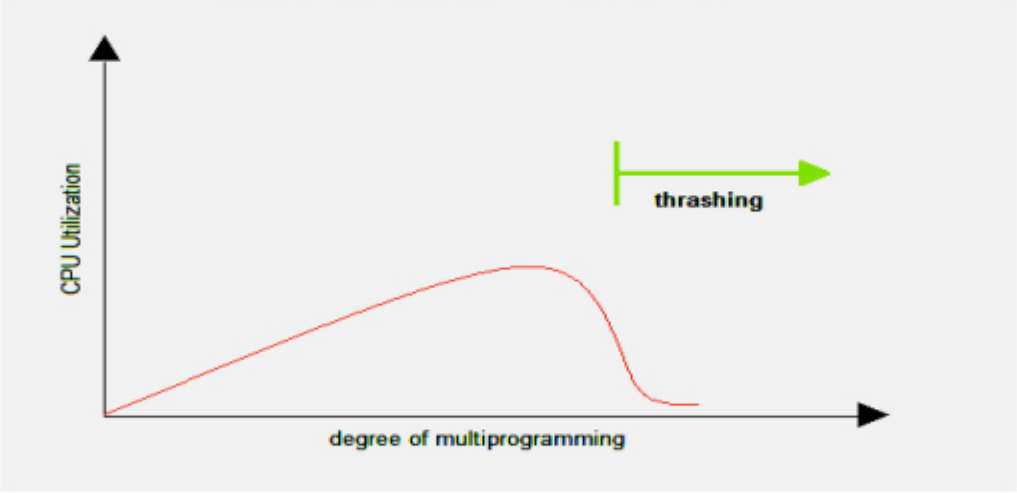


Fig: Thrashing

Segmentation:

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a segment map table for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory.

For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.

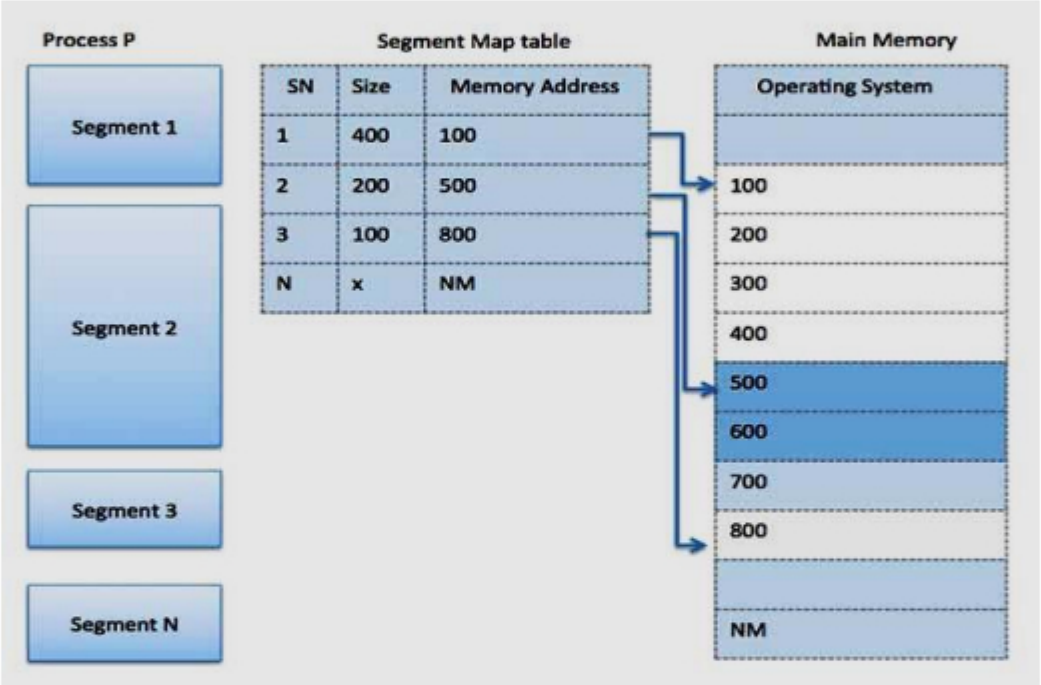


Fig: Segment Map Table

Segmentation with Paging:

Both paging and segmentation have their advantages and disadvantages, it is better to combine these two schemes to improve on each. The combined scheme is known as 'Page the Elements'.

Each segment in this scheme is divided into pages and each segment is maintained in a page table. So the logical address is divided into following 3 parts:

1. Segment numbers(S)
2. Page number (P)
3. The displacement or offset number (D)

Difference between Paging and Segmentation:

Paging	Segmentation
A page is a contiguous range of memory addresses which is mapped to physical memory.	A segment is an independent address space. Each segment has addresses in a range from 0 to maximum value.
It has only one linear address space.	It has many address spaces.
Programmer does not know that it is implemented	Programmer knows that it is implemented.
Procedures and data cannot be separated	Procedures and data can be separated
Procedures cannot be shared between users	Procedures can be shared between users
Procedures and data cannot be protected separately	Procedures and data can be protected separately
Compilation cannot be done separately	Compilation can be done separately
A page is a physical unit	A segment is a logical unit
A page is of fixed size	A segment is of arbitrary size.

Unit VI: Input/output Device Management – Operating System

Principle of Input/output Hardware:

One of the important jobs of an Operating System is to manage various Input/output devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio Input/output, printers etc.

Input/output Devices:

An input/output system is required to take an application input/output request and send it to the physical device, then take whatever response comes back from the device and send it to the application. Input/output devices can be divided into two categories:

1. **Block devices** – A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.
2. **Character devices** – A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards etc.

Device Controllers:

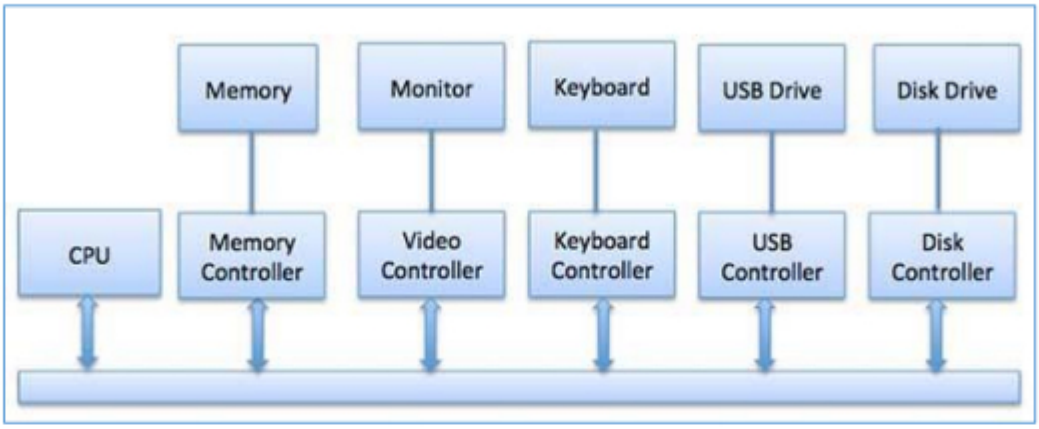
Device drivers are software modules that can be plugged into an Operating System to handle a particular device. Operating System takes help from device drivers to handle all input/output devices.

The Device Controller works like an interface between a device and a device driver. Input/output units (Keyboard, mouse, printer, etc.) typically consist of a mechanical component and an electronic component where electronic component is called the device controller.

There is always a device controller and a device driver for each device to communicate with the Operating Systems. A device controller may be able to handle multiple devices. As an interface its main task is to convert serial bit stream to block of bytes, perform error correction as necessary.

Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller. Following is a model for connecting the CPU, memory,

controllers, and input/output devices where CPU and device controllers all use a common bus for communication.



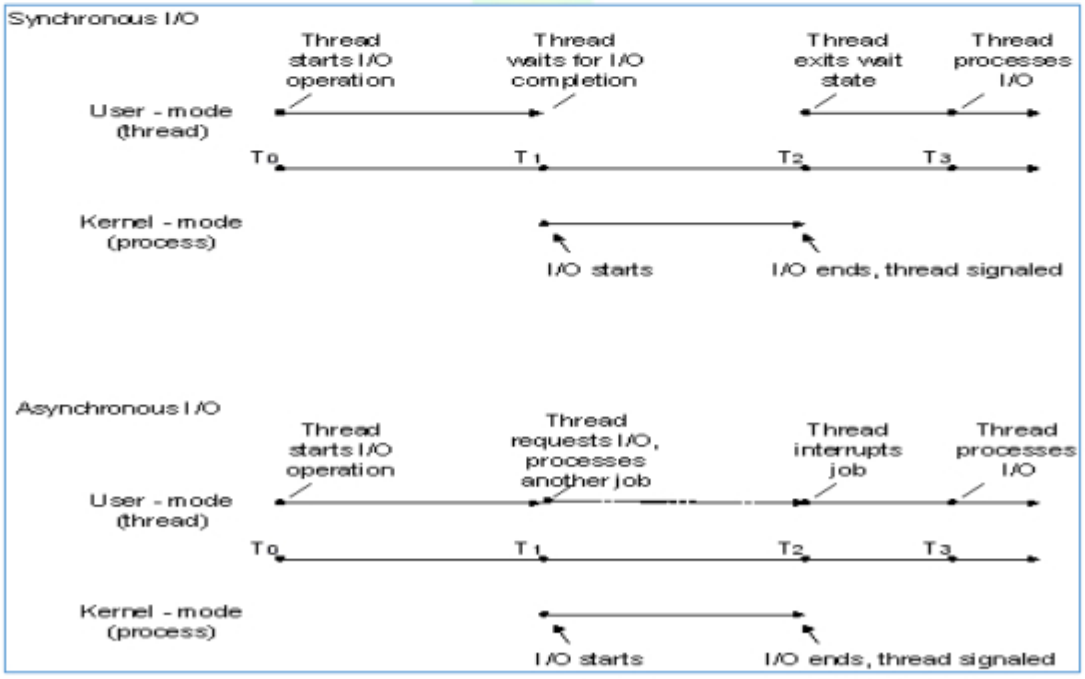
Synchronous V/S Asynchronous Input/output:

There are two types of input/output (I/O) synchronization: synchronous I/O and asynchronous I/O. Asynchronous I/O is also referred to as overlapped I/O.

In *synchronous file I/O*, a thread starts an I/O operation and immediately enters a wait state until the I/O request has completed. A thread performing *asynchronous file I/O* sends an I/O request to the kernel by calling an appropriate function.

If the request is accepted by the kernel, the calling thread continues processing another job until the kernel signals to the thread that the I/O operation is complete. It then interrupts its current job and processes the data from the I/O operation as necessary.

The two synchronization types are illustrated in the following figure.



In situations where an I/O request is expected to take a large amount of time, such as a refresh or backup of a large database or a slow communications link, asynchronous I/O is generally a good way to optimize processing efficiency.

However, for relatively fast I/O operations, the overhead of processing kernel I/O requests and kernel signals may make asynchronous I/O less beneficial, particularly if many fast I/O operations need to be made.

In this case, synchronous I/O would be better. The mechanisms and implementation details of how to accomplish these tasks vary depending on the type of device handle that is used and the particular needs of the application. In other words, there are usually multiple ways to solve the problem.

Communication to Input/output Devices:

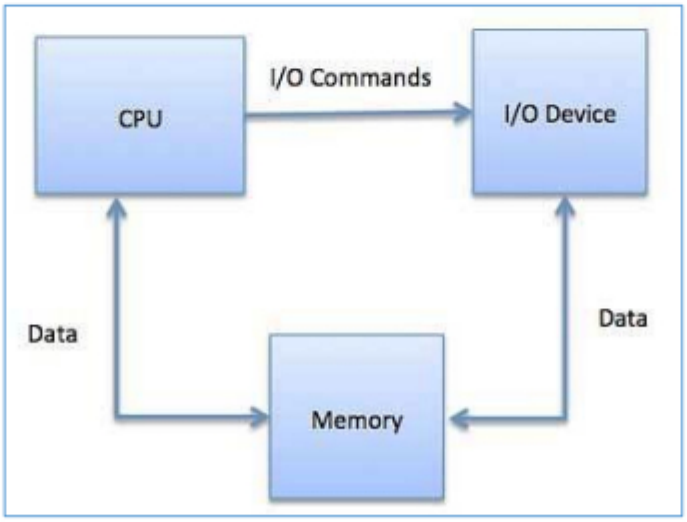
The CPU must have a way to pass information to and from an input/output device. There are three approaches available to communicate with the CPU and Device.

1. Special Instruction Input/Output:

This uses CPU instructions that are specifically made for controlling input/output devices. These instructions typically allow data to be sent to an input/output device or read from an input/output device.

2. Memory-Mapped Input/Output:

When using memory-mapped input/output, the same address space is shared by memory and input/output devices. The device is connected directly to certain main memory locations so that input/output device can transfer block of data to/from memory without going through CPU.



While using memory mapped input/output, operating system allocates buffer in memory and informs input/output device to use that buffer to send data to the CPU. Input/output device operates asynchronously with CPU, interrupts CPU when finished.

The advantage to this method is that every instruction which can access memory can be used to manipulate an input/output device. Memory mapped input/output is used for most high-speed input/output devices like disks, communication interfaces.

3. Direct Memory Access (DMA):

Slow devices like keyboards will generate an interrupt to the main CPU after each byte is transferred.

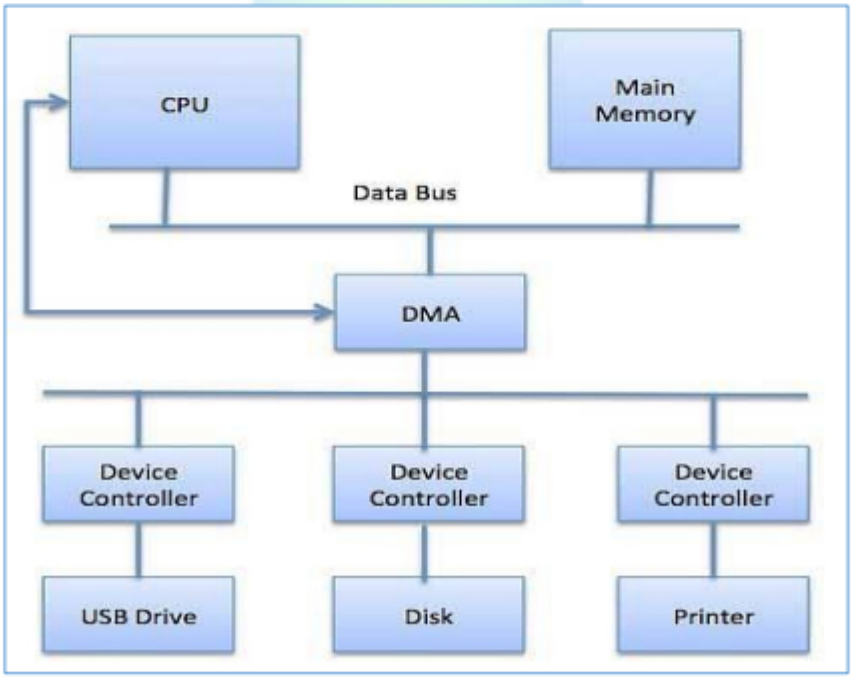
If a fast device such as a disk generated an interrupt for each byte, the operating system would spend most of its time handling these interrupts. So a typical computer uses direct memory access (DMA) hardware to reduce this overhead.

Direct Memory Access (DMA) means CPU grants input/output module authority to read from or write to memory without involvement.

DMA module itself controls exchange of data between main memory and the input/output device. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus.

The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes input/output and memory types, interrupts and states for the CPU cycles.



The operating system uses the DMA hardware as follows:

Step	Description
1	Device driver is instructed to transfer disk data to a buffer address X.
2	Device driver then instruct disk controller to transfer data to buffer.
3	Disk controller starts DMA transfer.
4	Disk controller sends each byte to DMA controller.
5	DMA controller transfers bytes to buffer, increases the memory address, decreases the counter C until C becomes zero.
6	When C becomes zero, DMA interrupts CPU to signal transfer completion.

Polling V/s interrupts Input/output:

A computer must have a way of detecting the arrival of any type of input. There are two ways that this can happen, known as **polling** and **interrupts**. Both of these techniques allow the processor to deal with events that can happen at any time and that are not related to the process it is currently running.

❖ Polling Input/Output:

Polling is the simplest way for an input/output device to communicate with the processor the processor. The process of periodically checking status of the device to see if it is time for the next input/output operation, is called polling.

The INPUT/OUTPUT device simply puts the information in a Status register, and the processor must come and get the information.

Most of the time, devices will not require attention and when one does it will have to wait until it is next interrogated by the polling program. This is an inefficient method and much of the processors time is wasted on unnecessary polls.

Compare this method to a teacher continually asking every student in a class, one after another, if they need help. Obviously the more efficient method would be for a student to inform the teacher whenever they require assistance.

❖ Interrupts input/output:

An alternative scheme for dealing with input/output is the interrupt-driven method. An interrupt is a signal to the microprocessor from a device that requires attention.

A device controller puts an interrupt signal on the bus when it needs CPU's attention when CPU receives an interrupt, it saves its current state and invokes the appropriate interrupt handler using the interrupt vector (addresses of OS routines to handle various events).

When the interrupting device has been dealt with, the CPU continues with its original task as if it had never been interrupted.

Principle of input/output Software:

The basic idea of input/output software is to organize the software as a series of layer. The lower layer contain the information of the hardware and the upper one concerned with presenting a nice, clean and regular interface to the user.

The main concept of input/output software is called as device independence which means that the program can be written in such a way that it can be read on the floppy, hard disk or CD-ROM without modifying the program for each different device.

Goals of input/output Software:

The main goal of input/output software has been broadly classified into four layers and they are:

1. Device Drivers:

Device drivers are software modules that can be plugged into an OS to handle a particular device.

Operating System takes help from device drivers to handle all I/O devices. Device drivers encapsulate device-dependent code and implement a standard interface in such a way that code contains device-specific register reads/writes.

Device driver, is generally written by the device's manufacturer and delivered along with the device on a CD-ROM. A device driver performs the following jobs:

- To accept request from the device independent software above to it.
- Interact with the device controller to take and give I/O and perform required error handling
- Making sure that the request is executed successfully

How a device driver handles a request is as follows: Suppose a request comes to read a block N.

If the driver is idle at the time a request arrives, it starts carrying out the request immediately. Otherwise, if the driver is already busy with some other request, it places the new request in the queue of pending requests.

2. Interrupt Handlers:

An interrupt handler, also known as an interrupt service routine or ISR, is a piece of software or more specifically a callback function in an operating system or more specifically in a device driver, whose execution is triggered by the reception of an interrupt.

When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt, updates data structures and wakes up process that was waiting for an interrupt to happen.

The interrupt mechanism accepts an address, a number that selects a specific interrupt handling routine/function from a small set. In most architectures, this address is an offset stored in a table called the interrupt vector table. This vector contains the memory addresses of specialized interrupt handlers.

3. Device-Independent I/O Software:

The basic function of the device-independent software is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user-level software.

Though it is difficult to write completely device independent software but we can write some modules which are common among all the devices. Following is a list of functions of device-independent I/O Software:

- Uniform interfacing for device drivers
- Device naming - Mnemonic names mapped to Major and Minor device numbers
- Device protection
- Providing a device-independent block size
- Buffering because data coming off a device cannot be stored in final destination.
- Storage allocation on block devices
- Allocation and releasing dedicated devices
- Error Reporting

4. User-Space I/O Software:

These are the libraries which provide richer and simplified interface to access the functionality of the kernel or ultimately interactive with the device drivers.

Most of the user-level I/O software consists of library procedures with some exception like spooling system which is a way of dealing with dedicated I/O devices in a multiprogramming system.

I/O Libraries (e.g., stdio) are in user-space to provide an interface to the OS resident device-independent I/O SW. For example putchar(), getchar(), printf() and scanf() are example of user level I/O library stdio available in C programming.

5. Kernel I/O Subsystem:

Kernel I/O Subsystem is responsible to provide many services related to I/O. Following are some of the services provided.

a. Scheduling:

Kernel schedules a set of I/O requests to determine a good order in which to execute them. When an application issues a blocking I/O system call, the request is placed on the queue for that device.

The Kernel I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by the applications.

b. Buffering:

Kernel I/O Subsystem maintains a memory area known as **buffer** that stores data while they are transferred between two devices or between a devices with an application operation.

Buffering is done to cope with a speed mismatch between the producer and consumer of a data stream or to adapt between devices that have different data transfer sizes.

c. Caching:

Kernel maintains cache memory which is region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original.

d. Spooling and Device Reservation:

A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. The spooling system copies the queued spool files to the printer one at a time.

In some operating systems, spooling is managed by a system daemon process. In other operating systems, it is handled by an in kernel thread.

e. Error Handling:

An operating system that uses protected memory can guard against many kinds of hardware and application errors.

Programmed Input/output:

The processor issues an input/output command, on behalf of a process, to an input/output module; that process then busy waits for the operation to be completed before proceeding.

When the processor is executing a program and encounters an instruction relating to input/output, it executes that instruction by issuing a command to the appropriate input/output module.

With the programmed input/output, the input/output module will perform the required action and then set the appropriate bits in the input/output status register. The input/output module takes no further action to alert the processor.

In particular it doesn't interrupt the processor. Thus, it is the responsibility of the processor to check the status of the input/output module periodically, until it finds that the operation is complete.

It is simplest to illustrate programmed input/output by means of an example. Consider a process that wants to print the eight character string ABCDEFGH.

a. It first assemble the string in a buffer in user space as shown in figure.
b. The user process then acquires the printer for writing by making system call to open it.

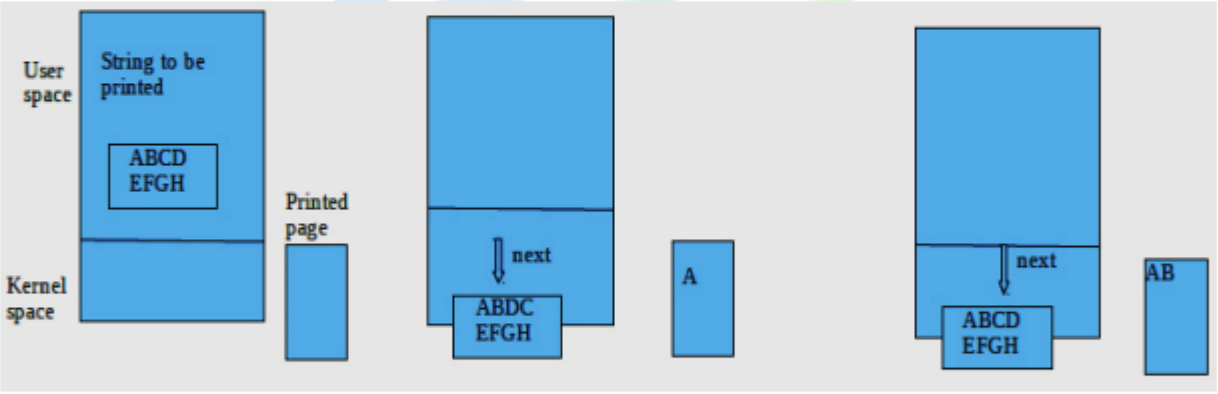


Fig. Steps in printing a string

c. If printer is in use by other the call will fail and enter an error code or will block until printer is available, depending on OS and the parameters of the call.
d. Once it has printer the user process makes a system call to print it.
e. OS then usually copies the buffer with the string to an array, say P in the kernel space where it is more easily accessed since the kernel may have to change the memory map to get to user space.
f. As the printer is available the OS copies the first character to the printer data register, in this example using memory mapped input/output. This action activates the printer. The character may not appear yet because some printers buffer a line or a page before printing.

g. As soon as it has copied the first character to the printer the operating system checks to see if the printer is ready to accept another one.
h. Generally printer has a second register which gives its status

First data are copied to the kernel, then the operating system enters a tight loop outputting the characters one at a time. The essentials aspects of programmed input/output is after outputting a character, the CPU continuously polls the device to see if it is ready to accept one. This behavior is often called polling or busy waiting.

```
copy_from_user(buffer,p,count); /*P is the kernel buffer*/
for(i=0;i<count;i++) { /* loop on every characters*/
while(*printer_status_reg!=READY); /*loop until ready*/
printer_data_register=P[i]; /*output one character */
}
return_to_user();
```

Programmed input/output is simple but has disadvantages of tying up the CPU full time until all the input/output is done.

In an embedded system where the CPU has nothing else to do, busy waiting is reasonable. However in more complex system where the CPU has to do other things, busy waiting is inefficient. A better input/output method is needed.

Interrupt-driven Input/output:

The problem with the programmed input/output is that the processor has to wait a long time for the input/output module of concern to be ready for either reception or transmission of more data.

The processor, while waiting, must repeatedly interrogate the status of the Input/ Output module. As a result the level of performance of entire system is degraded.

An alternative approach for this is interrupt driven Input / Output. The processor issue an Input/output command to a module and then go on to do some other useful work. The input/output module will then interrupt the processor to request service, when it is ready to exchange data with the processor.

The processor then executes the data transfer as before and then resumes its former processing. Interrupt-driven input/output still consumes a lot of time because every data has to pass with processor.

Input/output using DMA:

The previous ways of input/output suffer from two inherent drawbacks.

a. The input/output transfer rate is limited by the speed with which the processor can test and service a device.

b. The processor is tied up in managing an input/output transfer. A number of instructions must be executed for each input/output transfer.

When large volumes of data are to be moved, a more efficient technique is required i.e. direct memory access. The DMA function can be performed by a separate module on the system bus, or it can be incorporated into an input/output module. In either case, the technique works as follow.

When the processor wishes to read or write a block of data, it issues a command to the DMA module by sending the following information.

- Whether a read or write is requested.
- The address of the I/O devices.
- Starting location in memory to read from or write to.
- The number of words to be read or written.

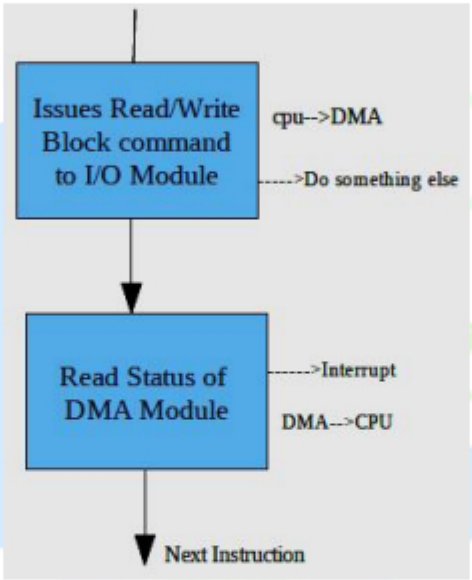


Fig: DMA

The processor then continues with other work. It has delegated this input/output operation to the DMA module, and that module will take care of it. The DMA module transfers the entire block of data, one word at time, directly to or from memory, without going through the processor.

When the transfer is complete, the DMA module sends an interrupt signal to the processor. Thus the processor is involved only at the beginning and at the end of the transfer.

In programmed input/output CPU takes care of whether the device is ready or not. Data may be lost. Whereas in Interrupt-driven input/output, device itself inform the CPU by generating an interrupt signal.

If the data rate of the input/output is too fast. Data may be lost. In this case CPU most be cut off, since CPU is too slow for the particular device. The initial state is too fast.

It is meaningful to allow the device to put the data directly to the memory. This is called DMA. DMA controller will take over the task of CPU. CPU is general purpose but the DMA controller is specific purpose.

A DMA module controls the exchange of data between main memory and an input/output module. The processor sends a request for the transfer of a block of data to the DMA module and is interrupted only after the entire block has been transferred.

I/O Software Layer:

I/O software is often organized in the following layers:

- **User Level Libraries**: This provides simple interface to the user program to perform input and output. For example, **stdio** is a library provided by C and C++ programming languages.
- **Kernel Level Modules**: This provides device driver to interact with the device controller and device independent I/O modules used by the device drivers.
- **Hardware**: This layer includes actual hardware and hardware controller which interact with the device drivers and makes hardware alive.

A key concept in the design of I/O software is that it should be device independent where it should be possible to write programs that can access any I/O device without having to specify the device in advance.

For example, a program that reads a file as input should be able to read a file on a floppy disk, on a hard disk, or on a CD-ROM, without having to modify the program for each different device.

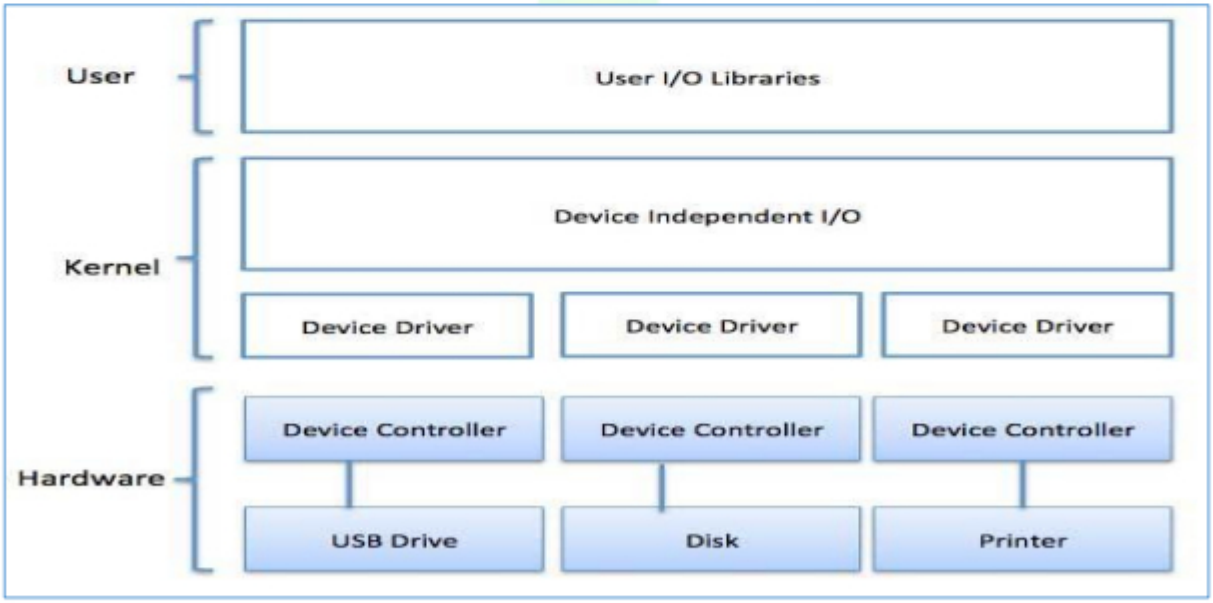


Fig: Input/output Software Layer

Disks:

All real disks are organized into cylinders, each one containing as many tracks as there are heads stacked vertically.

The tracks are divided into sectors, with the number of sectors around the circumference typically being 8 to 32 on floppy disks, and up to several hundred on some hard disks. The simplest designs have the same number of sectors on each track.

Disk Hardware:

1. Magnetic Disks:

Magnetic disks are organized into cylinders where each one containing as many tracks as there are heads stacked vertically. And the tracks are again divided into sectors.

2. RAID:

RAID stands for Redundant Array of Inexpensive Disks.

The basic idea behind Redundant Array of Inexpensive Disks is just to install a box full of disks next to the computer, typically a large server, replace the disk controller card with a RAID controller, copy the data over to the RAID and then continue the normal operation.

In other word, you can say that RAID looks like a SLED (Single Large Expensive Disk) to the OS. But RAID have better performance and reliability than SLED.

3. CD-ROM:

In the year of 1980, major two company namely Philips and Sony, together developed the Compact Disk (CD) which rapidly replaced 33 1/3-rpm vinyl record for music. All the compact disks are 120mm across and 1.2mm thick, with a 15mm hole in the middle of the CD.

Now, after almost 4 years, that is, in the year of 1984, these two company now realized the power and potential of using the compact disks to store the computer data, therefore they developed CD-ROM, that stands for Compact Disk-Read Only Memory.

4. CD-Recordable:

Physically, CD-Recordable (CD-R) stars with 120mm polycarbonate blanks that are like CD-ROMs, except that they contain a 0.6mm wide groove to guide the laser for writing.

5. CD-Rewritable:

After the people's demand for a rewritable CD-ROM, a technology was developed named CD-Rewritable (CD-RW) that uses the same size media as CD-Recordable (CD-R). But, CD-RW uses an alloy of silver, antimony, indium, and tellurium for the recording layer.

6. DVD:

DVD stands for Digital Video Disk or Digital Versatile Disk. DVD uses the same general design as of CD with 120mm injection-molded polycarbonate disks containing pits and lands that are illuminated by a laser diode and read by a photodetector.

Disk Scheduling:

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

Disk scheduling is important because:

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
- Two or more request may be far from each other so can result in greater disk arm movement.
- Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

There are many Disk Scheduling Algorithms but before discussing them let's have a quick look at some of the important terms:

a. Seek Time:

Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.

b. Rotational Latency:

Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.

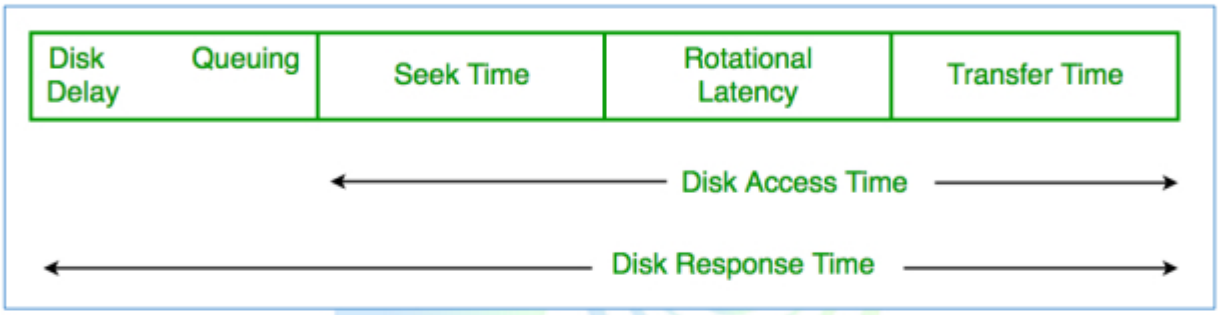
c. Transfer Time:

Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.

d. Disk Access Time:

Disk Access Time is:

Disk Access Time = Seek Time + Rotational Latency + Transfer Time



e. Disk Response Time:

Response Time is the average of time spent by a request waiting to perform its I/O operation. *Average Response time* is the response time of the all requests.

Variance Response Time is measure of how individual request are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better.

Disk Scheduling Algorithms:

Although there are other algorithms that reduce the seek time of all requests, I will only concentrate on the following disk scheduling algorithms:

Given the following queue 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the track 50 and the tail track being at 199 let us now discuss the different algorithms.

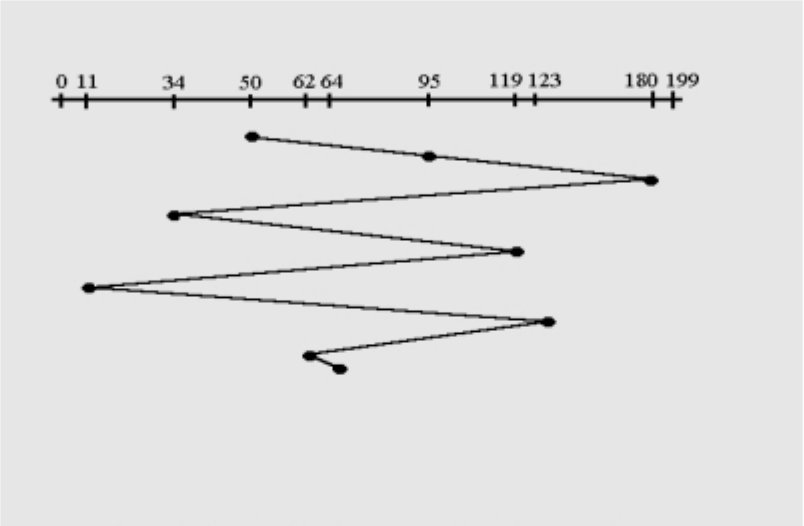
1. First Come -First Serve (FCFS):

All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results.

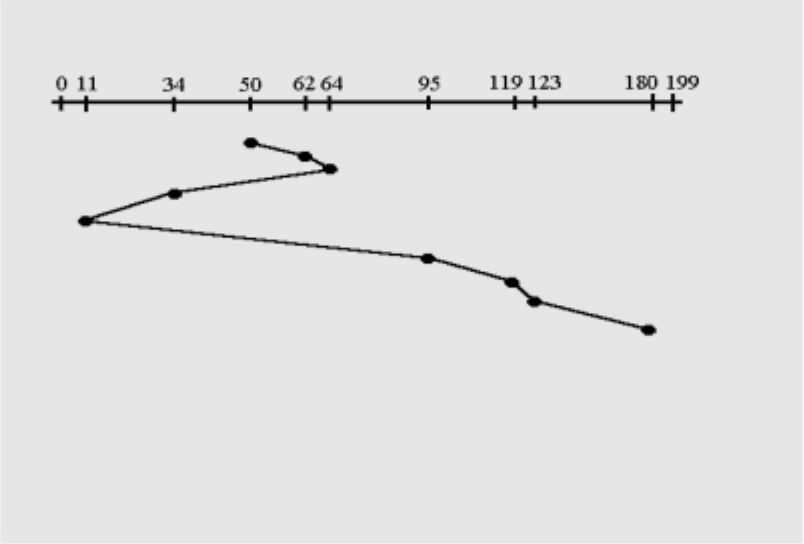
To determine the number of head movements you would simply find the number of tracks it took to move from one request to the next. For this case it went from 50 to 95 to 180 and so on. From 50 to 95 it moved 45 tracks.

If you tally up the total number of tracks you will find how many tracks it had to go through before finishing the entire request. In this example, it had a total head movement of 640 tracks.

The disadvantage of this algorithm is noted by the oscillation from track 50 to track 180 and then back to track 11 to 123 then to 64. As you will soon see, this is the worse algorithm that one can use.



2. Shortest Seek Time First (SSTF):

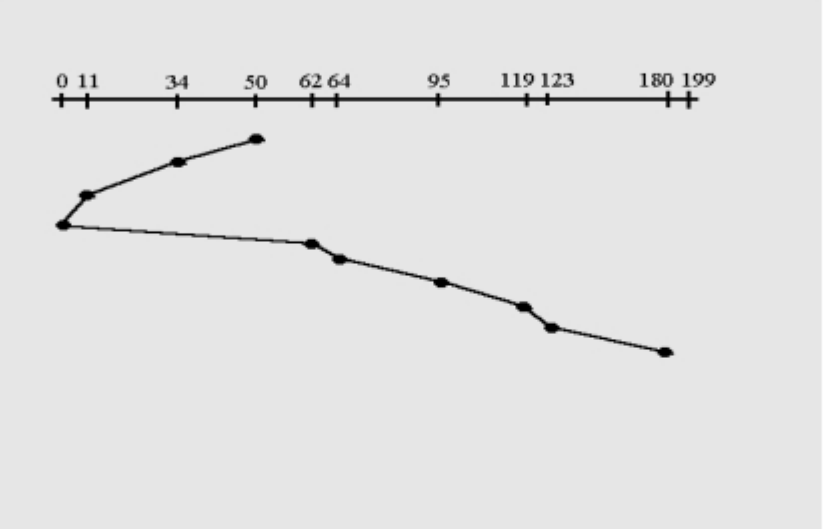


In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34.

The process would continue until all the process are taken care of. For example the next case would be to move from 62 to 64 instead of 34 since there are only 2 tracks between them and not 18 if it were to go the other way.

Although this seems to be a better service being that it moved a total of 236 tracks, this is not an optimal one. There is a great chance that starvation would take place. The reason for this is if there were a lot of requests close to each other the other requests will never be handled since the distance will always be greater.

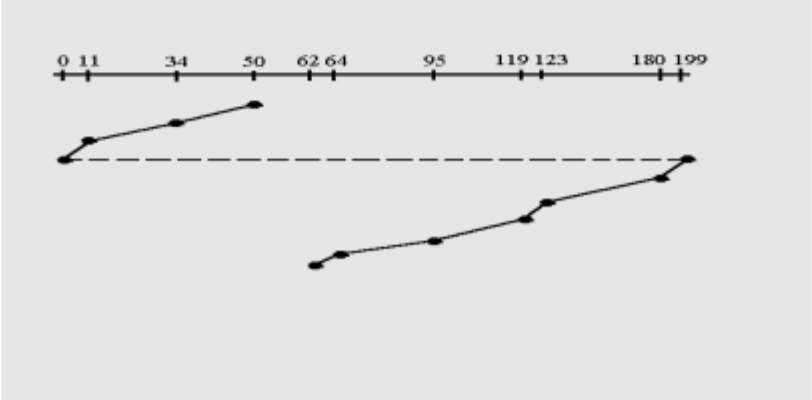
3. Elevator (SCAN):



This approach works like an elevator does. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down.

If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up. This process moved a total of 230 tracks. Once again this is more optimal than the previous algorithm, but it is not the best.

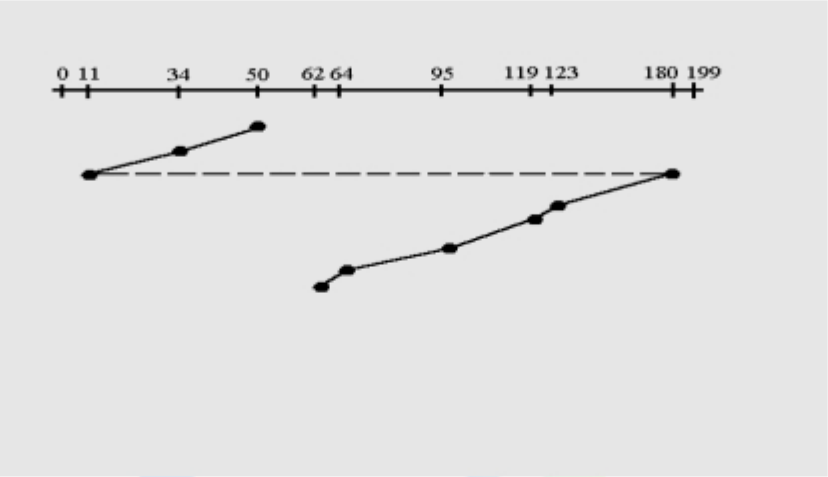
4. Circular Scan (C-SCAN):



Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works its way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction.

Keep in mind that the huge jump doesn't count as a head movement. The total head movement for this algorithm is only 187 track, but still this isn't the most sufficient.

5. C-LOOK:



This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request. C-SCAN had a total movement of 187 but this scan (C-LOOK) reduced it down to 157 tracks.

From this you were able to see a scan change from 644 total head movements to just 157. You should now have an understanding as to why your operating system truly relies on the type of algorithm it needs when it is dealing with multiple processes.

NOTE: It is important that you draw out the sequence when handling algorithms like this one. One would have a hard time trying to determine which algorithm is best by just reading the definition. There is a good chance that without the drawings there could be miscalculations.

RAID:

RAID (**redundant array of independent disks**, originally **redundant array of inexpensive disks** is a storage technology that combines multiple disk drive components into a logical unit.

Data is distributed across the drives in one of several ways called "RAID levels", depending on what level of redundancy and performance (via parallel communication) is required.

RAID Levels:

RAID 0:

RAID 0 (block-level striping without parity or mirroring) has no (or zero) redundancy. It provides improved performance and additional storage but no fault tolerance. Hence simple stripe sets are normally referred to as RAID 0.

Any drive failure destroys the array, and the likelihood of failure increases with more drives in the array (at a minimum, catastrophic data loss is almost twice as likely compared to single drives without RAID).

A single drive failure destroys the entire array because when data is written to a RAID 0 volume, the data is broken into fragments called blocks. The number of blocks is dictated by the stripe size, which is a configuration parameter of the array.

The blocks are written to their respective drives simultaneously on the same sector. This allows smaller sections of the entire chunk of data to be read off each drive in parallel, increasing bandwidth.

RAID 0 does not implement error checking, so any error is uncorrectable. More drives in the array means higher bandwidth, but greater risk of data loss.

RAID 1:

In ***RAID 1*** (mirroring without parity or striping), data is written identically to multiple drives, thereby producing a "mirrored set"; at least 2 drives are required to constitute such an array.

While more constituent drives may be employed, many implementations deal with a maximum of only 2; of course, it might be possible to use such a limited level 1 RAID itself as a constituent of a level 1 RAID, effectively masking the limitation.

The array continues to operate as long as at least one drive is functioning. With appropriate operating system support, there can be increased read performance, and only a minimal write performance reduction; implementing RAID 1 with a separate controller for each drive in order to perform simultaneous reads (and writes) is sometimes called *multiplexing* (or *duplexing* when there are only 2 drives).

RAID 2:

In ***RAID 2*** (bit-level striping with dedicated Hamming-code parity), all disk spindle rotation is synchronized, and data is striped such that each sequential bit is on a different drive. Hamming-code parity is calculated across corresponding bits and stored on at least one parity drive.

RAID 3:

In ***RAID 3*** (byte-level striping with dedicated parity), all disk spindle rotation is synchronized, and data is striped so each sequential byte is on a different drive. Parity is calculated across corresponding bytes and stored on a dedicated parity drive.

RAID 4:

RAID 4 (block-level striping with dedicated parity) is identical to RAID 5 (see below), but confines all parity data to a single drive. In this setup, files may be distributed between multiple drives. Each drive operates independently, allowing I/O requests to be performed in parallel.

However, the use of a dedicated parity drive could create a performance bottleneck; because the parity data must be written to a single, dedicated parity drive for each block of non-parity data, the overall write performance may depend a great deal on the performance of this parity drive.

RAID 5:

RAID 5 (block-level striping with distributed parity) distributes parity along with the data and requires all drives but one to be present to operate; the array is not destroyed by a single drive failure.

Upon drive failure, any subsequent reads can be calculated from the distributed parity such that the drive failure is masked from the end user.

However, a single drive failure results in reduced performance of the entire array until the failed drive has been replaced and the associated data rebuilt. Additionally, there is the potentially disastrous RAID 5 write hole. RAID 5 requires at least 3 disks.

RAID 6:

RAID 6 (block-level striping with double distributed parity) provides fault tolerance of two drive failures; the array continues to operate with up to two failed drives. This makes larger RAID groups more practical, especially for high-availability systems.

This becomes increasingly important as large capacity drives lengthen the time needed to recover from the failure of a single drive.

Single-parity RAID levels are as vulnerable to data loss as a RAID 0 array until the failed drive is replaced and its data rebuilt; the larger the drive, the longer the rebuild takes. Double parity gives additional time to rebuild the array without the data being at risk if a single additional drive fails before the rebuild is complete.

Unit VII: File System Interface Management – Operating System

File Concept:

Files are logical units of information created by processes, we model disk instead of RAM. Processes can read existing files and create new ones if needed. Information stored in files must be persistent that is not affected by process creation and termination.

Files should only be disappear if we remove it. How files are structured, named, accessed, used, protected, implemented and managed are major topics in operating system design. As a whole the part of operating system dealing with file is called files system.

File Naming:

The most important characteristics of any abstraction mechanism is the way the objects being named and managed. When a process creates a file, it gives the file a name. When the process terminates, the file continues to exist and can be accessed by other processes using its name.

File Structure:

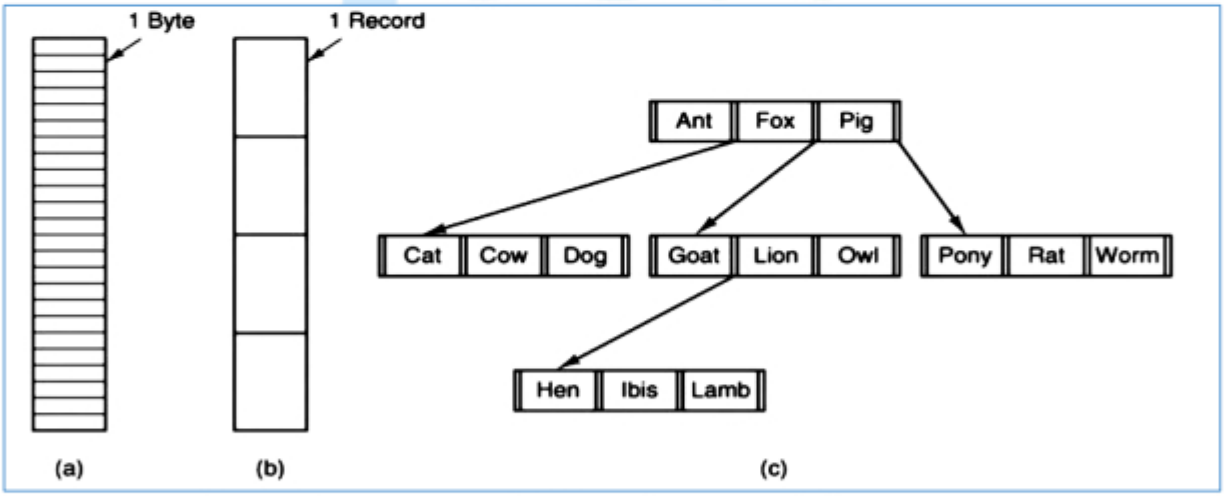


Fig: (a) Byte sequence (b) Record sequence (c) Tree Sequence

a. Byte Sequence:

The file in Fig: (a) just an unstructured sequence of bytes. In effect, the operating system does not know or care what is in the file. All it sees are bytes. Any meaning must be imposed by user-level programs. Both UNIX and Windows 98 use this approach.

b. Record Sequence:

In this model, a file is a sequence of fixed-length records, each with some internal structure. Central to the idea of a file being a sequence of records is the idea that the read operation returns one record and the write operation overwrites or appends one record.

As a historical note, when the 80-column punched card was king many (mainframe) operating systems based their file systems on files consisting of 80-character records, in effect, card images

c. Record Sequence:

In this organization, a file consists of a tree of records, not necessarily all the same length, each containing a key field in a fixed position in the record. The tree is sorted on the key field, to allow rapid searching for a particular key.

File Type:

File type refers to the ability of the operating system to distinguish different types of file such as text files, source files and binary files etc. Many operating systems support many types of files.

Operating system like MS-DOS and UNIX have the following types of files:

1. Ordinary files:

- These are the files that contain user information.
- These may have text, databases or executable program.
- The user can apply various operations on such files like add, modify, delete or even remove the entire file.

2. Directory files:

- These files contain list of file names and other information related to these files.

3. Special files:

- These files are also known as device files.
- These files represent physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types:

a. **Character special files**: data is handled character by character as in case of terminals or printers.
b. **Block special files**: data is handled in blocks as in the case of disks and tapes.

File Attributes:

Every file has a name and its data. In addition, all operating system associate other information with each file such as: date and time the file was created and the file size. We call these extra information file attributes or metadata.

The list of attributes varies considerably from system to system.

Attributes	Meanings
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random Access
Temporary flag	0 for normal; 1 for delete file on process Exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

File Operations:

Files exist to store information and allow it to be retrieved later. Different systems provide different operations to allow storage and retrieval. Below is a discussion of the most common system calls relating to files:

1. **Create:** The file is created with no data. The purpose of the call is to announce that the file is coming and to set some of the attributes.

2. **Delete:** When the file is no longer needed, it has to be deleted to free up disk space. A system call for this purpose is always provided.

3. **Open:** Before using a file, a process must open it. The purpose of the open call is to allow the system to fetch the attributes and list of disk addresses into main memory for rapid access on later calls.

4. **Close:** When all the accesses are finished, the attributes and disk addresses are no longer needed, so the file should be closed to free up some internal table space. Many systems encourage this by imposing a maximum number of open files on processes. A disk is written in blocks, and closing a file forces writing of the file's last block, even though that block may not be entirely full yet.

5. **Read:** Data are read from file. Usually, the bytes come from the current position. The caller must specify how much data are needed and must also provide a buffer to put them in.

6. **Write:** Data are written to the file, again, usually at the current position. If the current position is the end of the file, the file's size increases. If the current position is in the middle of the file, existing data are overwritten and lost forever.

7. **Append:** This call is a restricted form of write. It can only add data to the end of the file. Systems that provide a minimal set of system calls do not generally have append, but many systems provide multiple ways of doing the same thing, and these systems sometimes have append.

8. **Seek:** For random access files, a method is needed to specify from where to take the data. One common approach is a system call, seek, that repositions the file pointer to a specific place in the file. After this call has completed, data can be read from, or written to, that position.

9. **Get Attributes:** Processes often need to read file attributes to do their work. For example, the UNIX make program is commonly used to manage software development projects consisting of many source files. When make is called, it examines the modification times of all the source and object files and arranges for the minimum number of compilations required to bring everything up to date. To do its job, it must look at the attributes, namely, the modification times.

10. **Set Attributes:** Some of the attributes are user settable and can be changed after the file has been created. This system call makes that possible. The protection mode information is an obvious example. Most of the flags also fall in this category.

11. **Rename:** It frequently happens that a user needs to change the name of an existing file. This system call makes that possible. It is not always strictly necessary, because the file can usually be copied to a new file with the new name, and the old file then deleted.

12. **Lock:** Locking a file or a part of a file prevents multiple simultaneous access by different process. For an airline reservation system, for instance, locking the database while making a reservation prevents reservation of a seat for two different travelers.

File Descriptor:

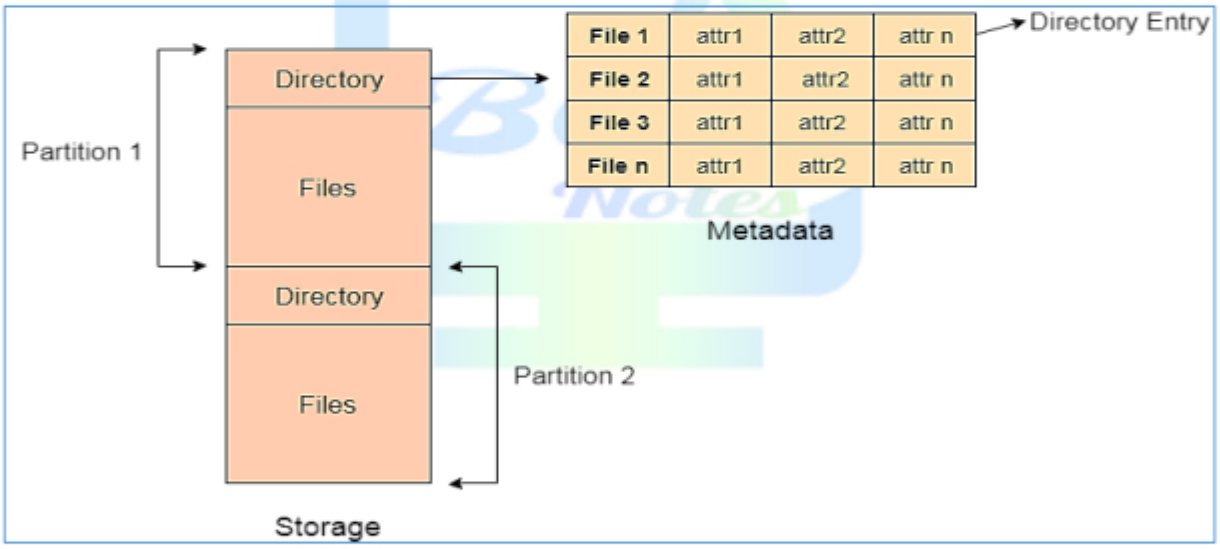
In UNIX and related computer operating systems, a file descriptor (FD, less frequently fildes) is an abstract indicator (handle) used to access a file or other input/output resource, such as a pipe or network socket.

File descriptors form part of the POSIX application programming interface. A file descriptor is a non-negative integer, generally represented in the C programming language as the type int (negative values being reserved to indicate "no value" or an error condition).

Directories:

Directory can be defined as the listing of the related files on the disk. The directory may store some or the entire file attributes. To get the benefit of different file systems on the different operating systems, a hard disk can be divided into the number of partitions of different sizes. The partitions are also called volumes or mini disks.

Each partition must have at least one directory in which, all the files of the partition can be listed. A directory entry is maintained for each file in the directory which stores all the information related to that file.



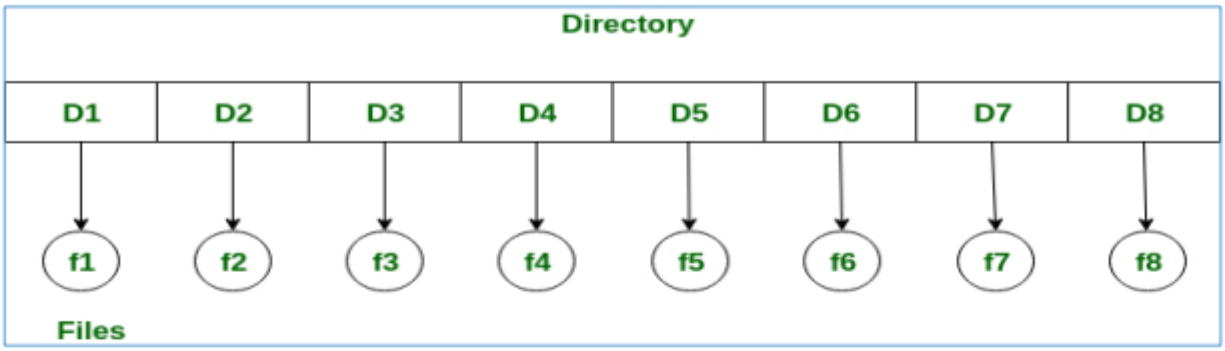
A directory can be viewed as a file which contains the Meta data of the bunch of files. There are several logical structures of a directory, these are given below:

1.Single-Level Directory:

The single-level directory is the simplest directory structure. In it, all files are contained in the same directory which makes it easy to support and understand.

A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user. Since all the files are in the same directory, they must have a unique name.

If two users call their dataset test, then the unique name rule violated.



Advantages:

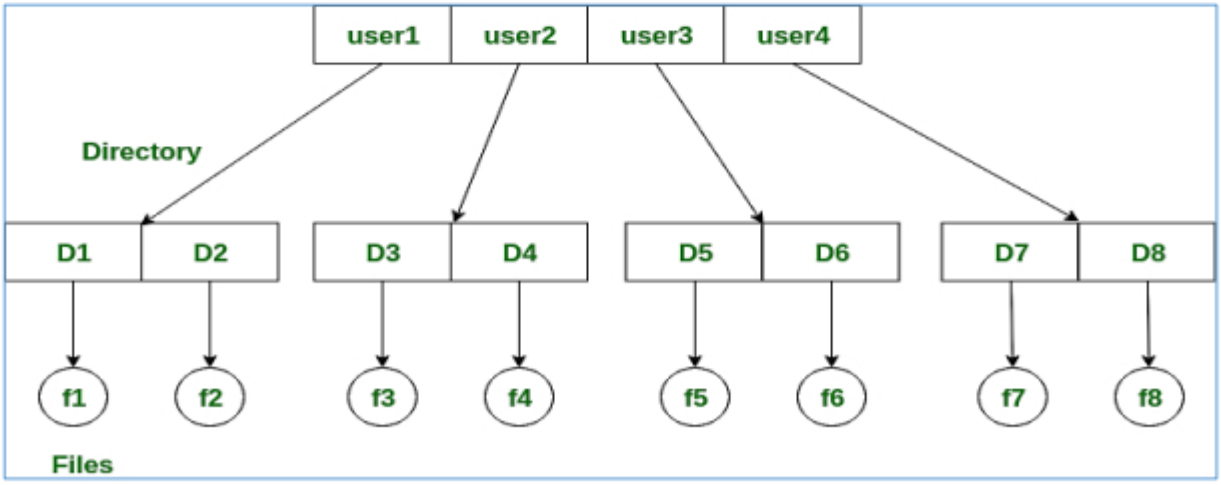
- Since it is a single directory, so its implementation is very easy.
- If the files are smaller in size, searching will become faster.
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

Disadvantages:

- There may chance of name collision because two files cannot have the same name.
- Searching will become time taking if the directory is large.
- This cannot group the same type of files together.

2. Two-Level Directory:

As we have seen, a single level directory often leads to confusion of files names among different users. The solution to this problem is to create a separate directory for each user.



In the two-level directory structure, each user has their own *user files directory (UFD)*. The UFDs have similar structures, but each lists only the files of a single user.

System's *master file directory (MFD)* is searches whenever a new user id=s logged in. The MFD is indexed by username or account number, and each entry points to the UFD for that user.

Advantages:

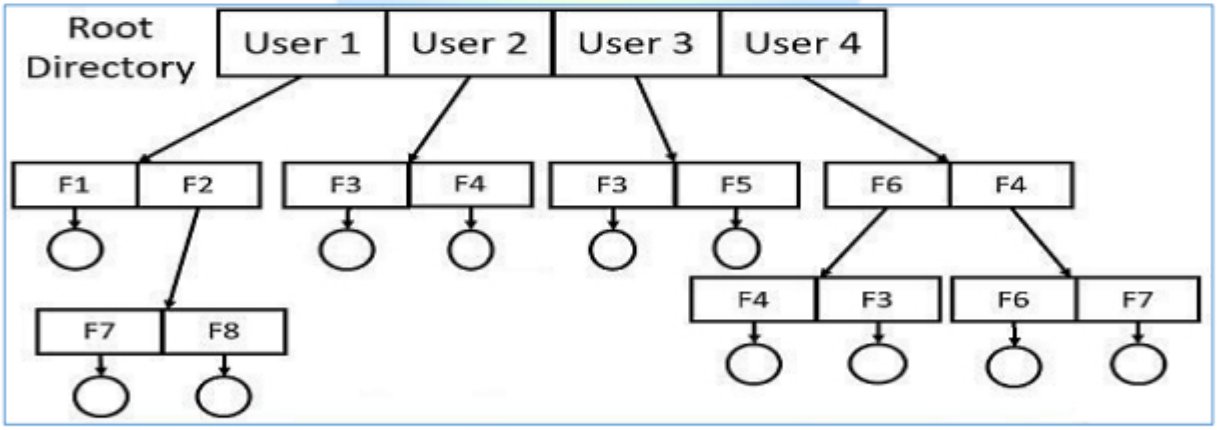
- We can give full path like /User-name/directory-name/.
- Different users can have the same directory as well as the file name.
- Searching of files becomes easier due to pathname and user-grouping.

Disadvantages:

- A user is not allowed to share files with other users.
- Still, it not very scalable, two files of the same type cannot be grouped together in the same user.

3. Hierarchical Directory Structure:

In Hierarchical directory structure, the users can create directories under the root directory and can also create sub-directories under this structure. As the user is free to create many sub-directories, it can create different sub-directories for different file types.

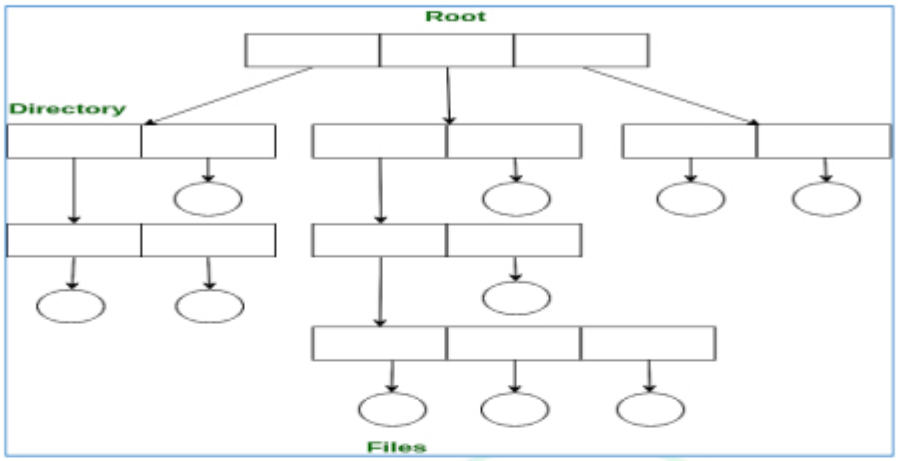


Here, the files are accessed by their location using the path. There are two types of paths to locate the file in this directory structure:

- **Absolute Path:** Here, the path for the desired file is described by considering the root directory as the base directory.
- **Relative Path:** Here, either the user's directory is considered as the base directory or the desired file directory is considered as the base directory.

4. Tree-Structured Directory:

Once we have seen a two-level directory as a tree of height 2, the natural generalization is to extend the directory structure to a tree of arbitrary height. This generalization allows the user to create their own subdirectories and to organize their files accordingly.



A tree structure is the most common directory structure. The tree has a root directory, and every file in the system has a unique path.

Advantages:

- Very general, since full pathname can be given.
- Very scalable, the probability of name collision is less.
- Searching becomes very easy, we can use both absolute paths as well as relative.

Disadvantages:

- Every file does not fit into the hierarchical model, files may be saved into multiple directories.
- We cannot share files.
- It is inefficient, because accessing a file may go under multiple directories.

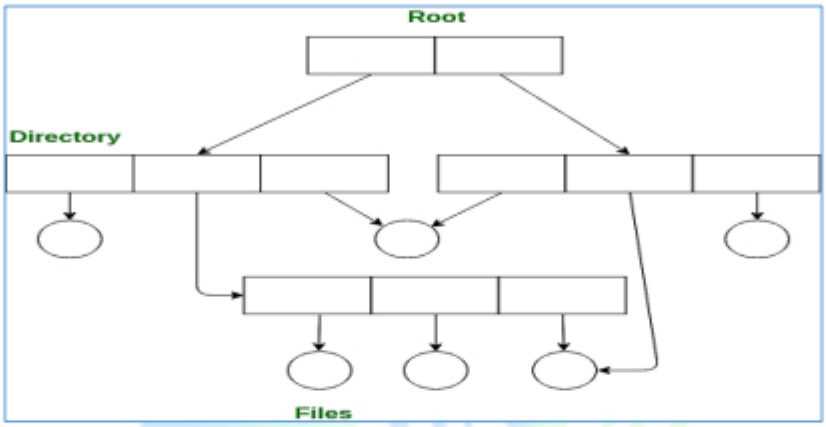
5. Acyclic Graph Directory:

An acyclic graph is a graph with no cycle and allows us to share subdirectories and files. The same file or subdirectories may be in two different directories. It is a natural generalization of the tree-structured directory.

It is used in the situation like when two programmers are working on a joint project and they need to access files.

The associated files are stored in a subdirectory, separating them from other projects and files of other programmers since they are working on a joint project so they want the subdirectories to be into their own directories. The common subdirectories should be shared. So here we use acyclic directories.

It is the point to note that the shared file is not the same as the copy file. If any programmer makes some changes in the subdirectory it will reflect in both subdirectories.



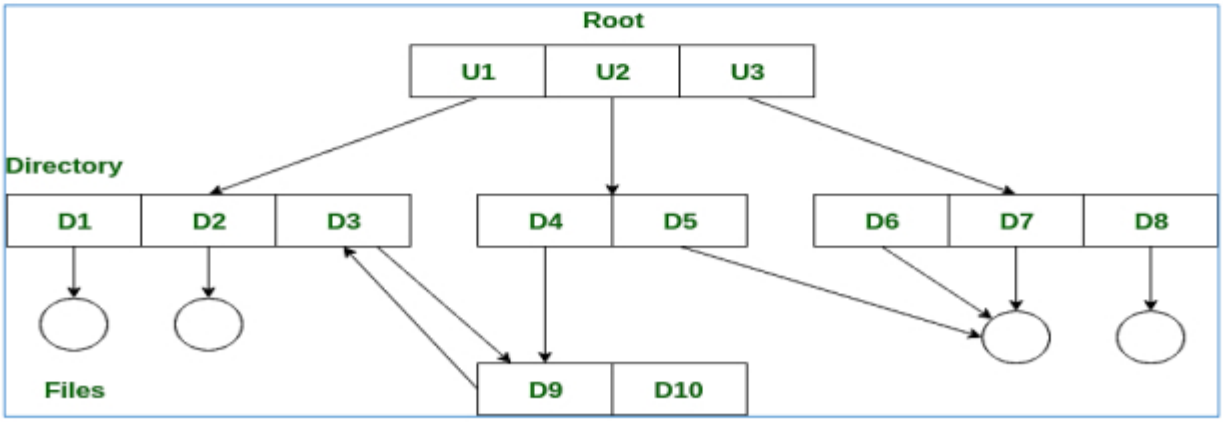
Advantages:

- We can share files.
- Searching is easy due to different-different paths.

Disadvantages:

- We share the files via linking, in case deleting it may create the problem,
- If the link is a soft link then after deleting the file we left with a dangling pointer.
- In the case of a hard link, to delete a file we have to delete all the references associated with it.

6. General Graph Directory Structure:



In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory.

The main problem with this kind of directory structure is to calculate the total size or space that has been taken by the files and directories.

Advantages:

- It allows cycles.
- It is more flexible than other directories structure.

Disadvantages:

- It is more costly than others.
- It needs garbage collection.

Path Names:

A sequence of symbols and names that identifies a file. Every file has a name, called a filename, so the simplest type of pathname is just a filename. If we specify a filename as the pathname, the operating system looks for that file in our current working directory.

However, if the file resides in a different directory, we must tell the operating system how to find that directory. We perform this by specifying a path that the operating system must follow.

The pathname always starts from our working directory or from the root directory. Each operating system has its own rules for specifying paths. In DOS systems, the root directory is named \ and each subdirectory is separated by an additional backslash.

In UNIX, the root directory is named /, and each subdirectory is followed by a slash. In Macintosh environments, directories are separated by a colon.

Operations on Directory:

A directory contains the entries of all the related files. For organizing the directory in the better way the user must be able to insert, delete, search, and list the entries in the directory. The operation that can be performed on the directory are:

1. **Searching:** A directory can be **searched** for a particular file or for another directory. It can also be searched to list all the files with the same name.

2. **Creating**: A new file can be created and inserted to the directory or new directory can be created keeping in mind that its name must be unique under that particular directory.

3. **Deleting**: If a file is no longer needed by the user, it can be deleted from the directory. The entire directory can also be deleted if it is not needed. An empty directory can also be deleted. When a directory is empty it is resembled by dot and dot dot.

4. **List a directory**: List of all the files in the directory can be retrieved and also the contents of the directory entry, for each file in a list. To read the list of all the files in the directory, it must be opened and after reading the directory must be closed to free up the internal tablespace.

5. **Renaming**: The name of the file or a directory represents the content it holds and its use. The file or directory can be renamed in case, the content inside or the use of file get changed. Renaming the file or directory also changes its position inside the directory.

6. **Link**: The file can be allowed to appear in more than one directory. Here, the system call creates a link between the file and the name specified by the path where the file is to appear.

7. **Unlink**: If the file is unlinked and is only present in one directory its directory entry is removed. If the file appears in multiple directories, only the link is removed.

Access Method/Mechanism:

When a file is used, information is read and accessed into computer memory and there are several ways to access this information of the file. Some systems provide only one access method for files.

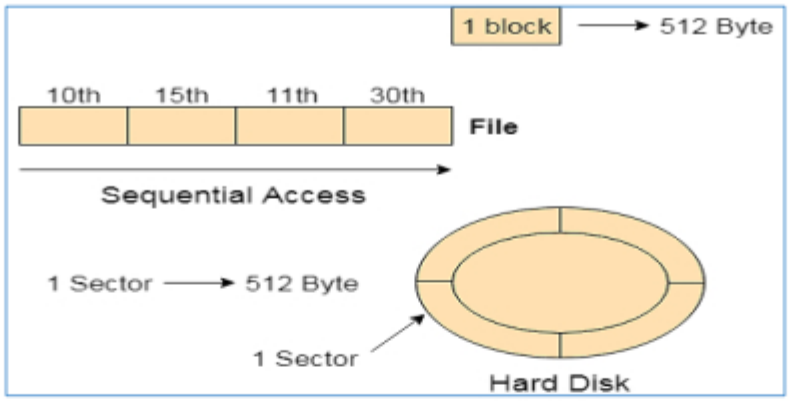
Other systems, such as those of IBM, support many access methods, and choosing the right one for a particular application is a major design problem. There are three ways to access a file into a computer system:

1.Sequential Access Method:

Sequential Access Method is one of the simplest file access methods. Most of the OS (operating system) uses a sequential access method to access the file. In this method, word by word, the operating system read the file, and we have a pointer that points the file's base address.

If we need to read the file's first word, then there is a pointer that offers the word which we want to read and increment the value of the word by 1, and till the end of the file, this process will continue.

The Modern world system offers the facility of index access and direct access to file. But the sequential access method is one of the most used methods because more files like text files, audio files, and the video files require to be sequentially accessed.

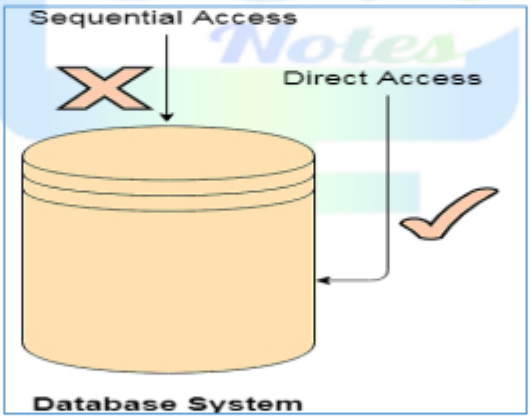


Key Points:

- Sequential Access Method is reasonable for tape.
- In the sequential access method, if we use read command, then the pointer is moved ahead by 1.
- If the write command is used, then the memory will be allocated, and at the end of the file, a pointer will be moved.

2. Direct Access Method:

Direct Access Method is also called the Relative access method. In the Database System, we mostly use the Direct Access Method.



In most of the situations, there is a need for information in the filtered form from the database. And in that case, the speed of sequential access may be low and not efficient.

Let us assume that each storage block stores 4 records, and we already know that the block which we require is stored in the 10th block. In this situation, the sequential access method is not suitable. Since if we use this, then to access the record which is needed, this method will traverse all the blocks.

So, in this situation, the Direct access method gives a more satisfying result, else the OS (operating system) needs to perform a few complex jobs like defining the block number, which is required. It is mostly implemented in the application of the database.

3.Index Access Method:

Index Access Method is another essential method of file, accessing. In this method, for a file an index is created, and the index is just like an index which is at the back of the book. The index includes the pointer to the different blocks.

If we want to find a record in the file, then first, we search in the index, and after that, with the help of the pointer, we can directly access the file.

In the Index Access method, we can search fast in the large database, and also easy. But in this, the method need some additional space to store the value of the index in the memory.

Key Points:

- On top of the sequential access method, the index access method is built.
- The Index access method can control the pointer with the help of the index.

Protection:

File system often contain information that is highly valuable to their users. Protecting these information against unauthorized usage is the major concern of all file system.

We generally use the term security to refer to the overall problem, and the term protection mechanism to refer to the specific operating system mechanism used to safe guard information in the computers.

Defense Mechanism:

To protect the system, we must take security at four levels:

a. **Physical:** Site containing the computer system must be physically secured.
b. **Human:** User authentication by secure password.
c. **Network:** Setting firewall
d. **Operating System:** Checking its file system regular interval of time.

Types of Access:

Access control is a security technique that can be used to regulate who or what can view or use resources in a computing environment. There are two main types of access control: physical and logical.

Physical access control limits access to campuses, buildings, rooms and physical IT assets. Logical access limits connections to computer networks, system files and data.

Access controls are necessary to protect the *confidentiality, integrity,* and *availability* of objects (and by extension, their information and data). The term access control is used to

describe a broad range of controls, from forcing a user to provide a valid username and password to log on to preventing users from gaining access to a resource outside of their sphere of access.

Access controls can be divided into the following seven categories of function or purpose. We should notice that some security mechanisms can be labeled with multiple function or purpose categories.

1. Preventative Access Control:

A preventative access control is deployed to stop unwanted or unauthorized activity from occurring.

Examples of preventative access controls include fences, locks, biometrics, mantraps, lighting, alarm systems, separation of duties, job rotation, data classification, penetration testing, access control methods, encryption, auditing, presence of security cameras or closed circuit television (CCTV), smart cards, callback, security policies, security awareness training, and antivirus software.

2. Deterrent Access Control:

A deterrent access control is deployed to discourage the violation of security policies. A deterrent control picks up where prevention leaves off. The deterrent doesn't stop with trying to prevent an action; instead, it goes further to exact consequences in the event of an attempted or successful violation.

Examples of deterrent access controls include locks, fences, security badges, security guards, mantraps, security cameras, trespass or intrusion alarms, separation of duties, work task procedures, awareness training, encryption, auditing, and firewalls.

3. Detective Access Control:

A detective access control is deployed to discover unwanted or unauthorized activity. Often detective controls are after-the-fact controls rather than real-time controls.

Examples of detective access controls include security guards, guard dogs, motion detectors, recording and reviewing of events seen by security cameras or CCTV, job rotation, mandatory vacations, audit trails, intrusion detection systems, violation reports, honey pots, supervision and reviews of users, incident investigations, and intrusion detection systems.

4. Corrective Access Control:

A corrective access control is deployed to restore systems to normal after an unwanted or unauthorized activity has occurred. Usually corrective controls have only a minimal capability to respond to access violations.

Examples of corrective access controls include intrusion detection systems, antivirus solutions, alarms, mantraps, business continuity planning, and security policies.

5. Recovery Access Control:

A recovery access control is deployed to repair or restore resources, functions, and capabilities after a violation of security policies. Recovery controls have more advanced or complex capability to respond to access violations than a corrective access control.

For example, a recovery access control can repair damage as well as stop further damage. Examples of recovery access controls include backups and restores, fault tolerant drive systems, server clustering, antivirus software, and database shadowing.

6. Compensation Access Control:

A compensation access control is deployed to provide various options to other existing controls to aid in the enforcement and support of a security policy.

Examples of compensation access controls include security policy, personnel supervision, monitoring, and work task procedures. Compensation controls can also be considered to be controls used in place of or instead of more desirable or damaging controls.

For example, if a guard dog cannot be used because of the proximity of a residential area, a motion detector with a spotlight and a barking sound playback device can be used.

7. Directive Access Control:

A directive access control is deployed to direct, confine, or control the actions of subject to force or encourage compliance with security policies.

Examples of Directive access controls include security guards, guard dogs, security policy, posted notifications, escape route exit signs, monitoring, supervising, work task procedures, and awareness training.

Access controls can be further categorized by how they are implemented. In this case, the categories are administrative, logical/technical, or physical:

❖ Administrative Access Controls:

Administrative access controls are the policies and procedures defined by an organizations security policy to implement and enforce overall access control. Administrative access controls focus on two areas: personnel and business practices (e.g., people and policies).

Examples of administrative access controls include policies, procedures, hiring practices, background checks, data classification, security training, vacation history, reviews, work supervision, personnel controls, and testing.

❖ Logical/Technical Access Controls:

Logical access controls and technical access controls are the hardware or software mechanisms used to manage access to resources and systems and provide protection for those resources and systems.

Examples of logical or technical access controls include encryption, smart cards, passwords, biometrics, constrained interfaces, access control lists (ACLs), protocols, firewalls, routers, intrusion detection systems, and clipping levels.

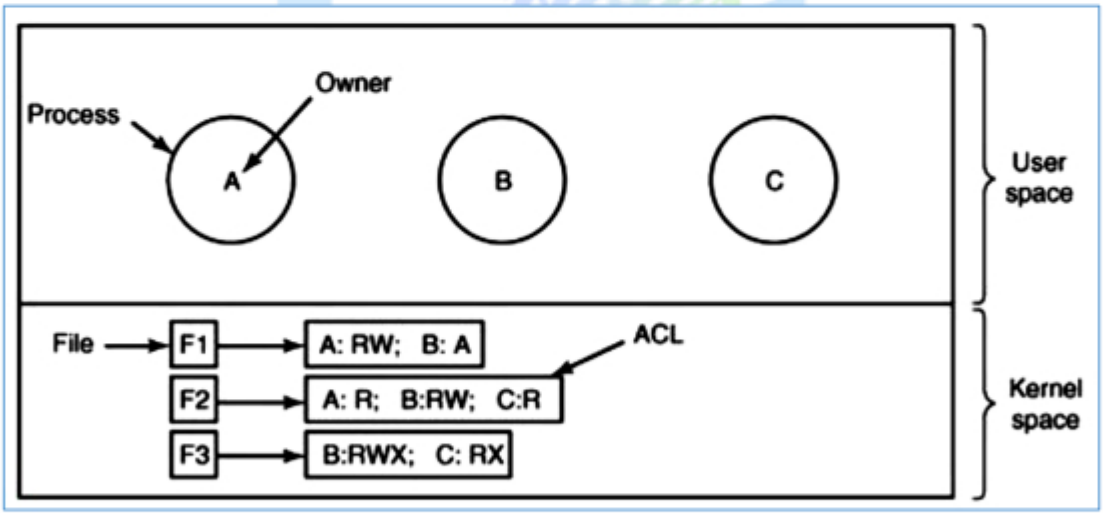
❖ Physical Access Controls:

Physical access controls are the physical barriers deployed to prevent direct contact with systems or portions of a facility. Examples of physical access controls include guards, fences, motion detectors, locked doors, sealed windows, lights, cable protections, laptop locks, swipe cards, guard dogs, video cameras, mantraps, and alarms.

Access Control List (ACL):

Access control list, a set of data that informs a computer's operating system which permissions, or access rights, that each user or group has to a specific system object, such as a directory or file.

Each object has a unique security attribute that identifies which users have access to it, and the ACL is a list of each object and user access privileges such as read, write or execute.



Each file has an ACL associated with it. File F1 has two entries in its ACL (separated by a semicolon). The first entry says that any process owned by user A may read and write the file. The second entry says that any process owned by user B may read the file.

All other accesses by these users and all accesses by other users are forbidden. Note that the rights are granted by user, not by process. As far as the protection system goes, any process owned by user A can read and write file F1.

It does not matter if there is one such process or 100 of them. It is the owner, not the process ID that matters. File F2 has three entries in its ACL: A, B, and C can all read the file, and in addition B can also write it. No other accesses are allowed. File F3 is apparently an executable program, since B and C can both read and execute it. B can also write it.

Access Control Matrix:

The access controls provided with an operating system typically authenticate principals using some mechanism such as passwords or Kerberos, then mediate their access to files, communications ports, and other system resources.

Their effect can often be modelled by a matrix of access permissions, with columns for files and rows for users. We'll write r for permission to read, w for permission to write, x for permission to execute a program, and (–) for no access at all, as shown in Figure below.

	Operating System	Accounts Program	Accounting Data	Audit Trail
Sam	rwx	rwx	rw	r
Alice	x	x	rw	–
Bob	rx	r	r	r

In this simplified example, Sam is the system administrator, and has universal access (except to the audit trail, which even he should only be able to read).

Alice, the manager, needs to execute the operating system and application, but only through the approved interfaces. She mustn't have the ability to tamper with them. She also needs to read and write the data. Bob, the auditor, can read everything.

Access control matrices (whether in two or three dimensions) can be used to implement protection mechanisms, as well as just model them. But they do not scale well. For instance, a bank with 50,000 staff and 300 applications would have an access control matrix of 15 million entries.

This is inconveniently large. It might not only impose a performance problem but also be vulnerable to administrators' mistakes. We will usually need a more compact way of storing and managing this information.

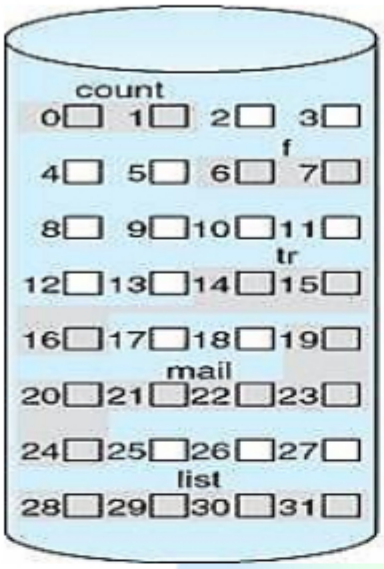
The two main ways of doing this are to use groups or roles to manage the privileges of large sets of users simultaneously, or to store the access control matrix either by columns (access control lists) or rows (capabilities, sometimes known as "tickets") or certificates.

File System Implementation:

Contiguous File Allocation:

With contiguous allocation, each file has to occupy contiguous blocks on the disk. The location of a file is defined by the disk address of the first block and its length. Both sequential access and direct/Random access are supported by the contiguous allocation.

The disadvantage of contiguous allocation is that it is often difficult to find free space for a new file. Moreover, one is often not sure of the space required while creating a new file. The various methods adopted to find space for a new file suffer from external fragmentation.

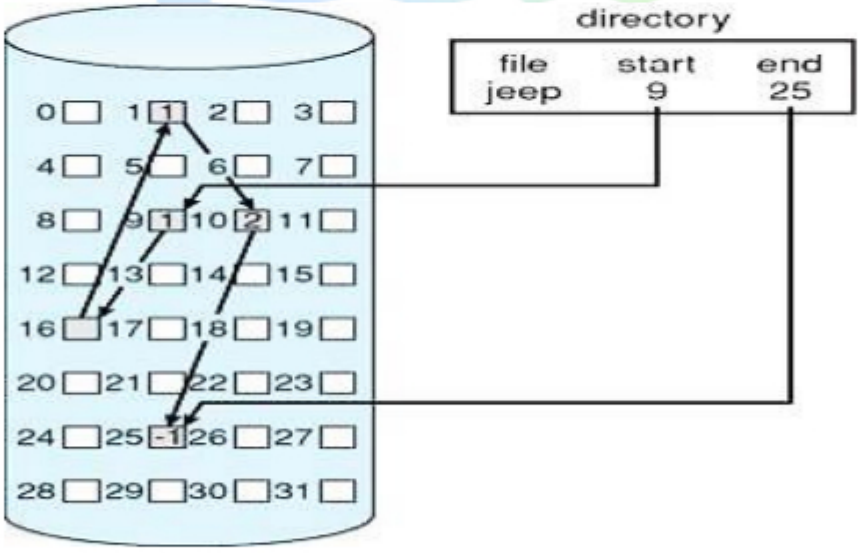


directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Linked List File Allocation:

In linked allocation, each file is a linked list of disk blocks. The directory contains a pointer to the first and (optionally the last) block of the file.



For example, a file of 5 blocks which starts at block 4, might continue at block 7, then block 16, block 10, and finally block 27. Each block contains a pointer to the next block and the last block contains a NIL pointer. The value -1 may be used for NIL to differentiate it from block 0.

There is no external fragmentation with linked allocation. Any free block can be used to satisfy a request. Notice also that there is no need to declare the size of a file when that file is created. A file can continue to grow as long as there are free blocks.

Linked allocation, does have disadvantages, however. The major problem is that it is inefficient to support direct-access; it is effective only for sequential-access files. To find the ith block of a file, it must start at the beginning of that file and follow the pointers until the ith block is reached. Note that each access to a pointer requires a disk read.

Linked List File Allocation with Table:

Disadvantage of linked list file allocation can be eliminated by taking the pointer word from each disk block and putting it in a table in memory.

In this method of storage allocation, instead of a pointer, a table in memory called file allocation table is used. Advantage of this method is that random access of files is faster.

The only disadvantage of this method is that the entire table should always be in the memory. FAT file system is of 3 types: FAT-12, FAT-16, and FAT-32. Here the numbers 12, 16 are the number of bits of disk addresses.

However, in FAT-32, 32 is a misnomer. Here only 28 number of bits of disk addresses are used.

Index	FAT	Comment
0		
1		
2	10	
3	11	
4	7	File A starts here
5		
6	3	File B starts here
7	2	
8		
9		
10	12	
11	14	
12	-1	File A terminate
13		
14	-1	File B terminate
15		

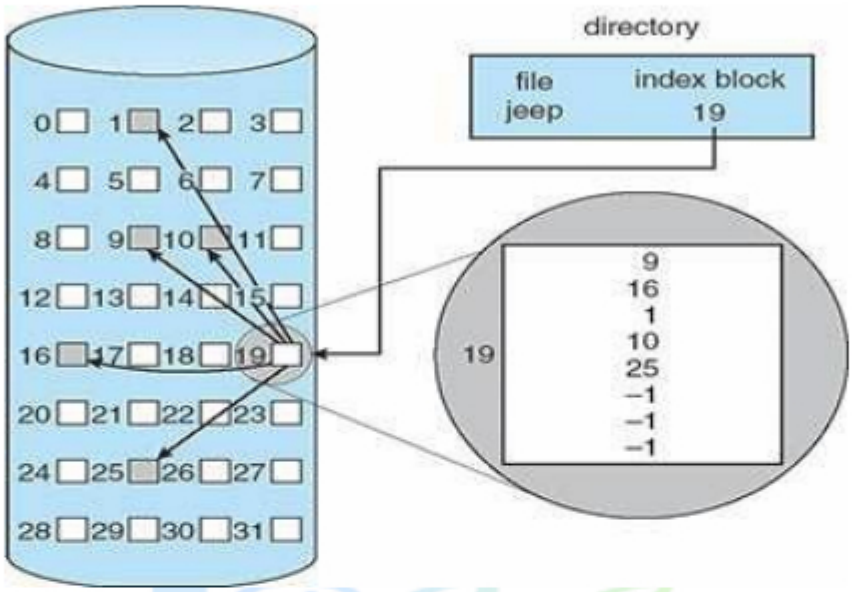
In the above figure, File 'A' uses disk blocks 4, 7, 2, 10 and 12 in that order and File 'B' uses disk blocks 6, 3, 11, and 14 in that order. Using table we can start with block 4 and follow the chain all the way to end.

The same can be done starting with block 6. Both chains are terminated with a special mark (-1) that is not valid block number. Such a table in memory is called a FAT (File Allocation Table).

Indexed File Allocation (I-NODE):

Linked allocation does not support random access of files, since each block can only be found from the previous. Indexed allocation solves this problem by bringing all the pointers together into an index block.

This type of allocation will have a pointer which has the address of all the blocks of a file. This method solves the problem of fragmentation as the blocks can be stored in any location.



File Security:

In computer systems, a lot of user's information is stored, the objective of the operating system is to keep safe the data of the user from the improper access to the system. Protection can be provided in number of ways.

For a single laptop system, we might provide protection by locking the computer in a desk drawer or file cabinet. For multi-user systems, different mechanisms are used for the protection.

Types of Access:

The files which have direct access of the any user have the need of protection. The files which are not accessible to other users doesn't require any kind of protection. The mechanism of the protection provide the facility of the controlled access by just limiting the types of access to the file.

Access can be given or not given to any user depends on several factors, one of which is the type of access required. Several different types of operations can be controlled:

- **Read:** Reading from a file.
- **Write:** Writing or rewriting the file.

- **Execute:** Loading the file and after loading the execution process starts.
- **Append:** Writing the new information to the already existing file, editing must be end at the end of the existing file.
- **Delete:** Deleting the file which is of no use and using its space for the data.
- **List:** List the name and attributes of the file.

Operations like renaming, editing the existing file, copying; these can also be controlled. There are many protection mechanism. Each of them mechanism have different advantages and disadvantages and must be appropriate for the intended application.

Access Control:

There are different methods used by different users to access any file. The general way of protection is to associate *identity-dependent access* with all the files and directories and list called access-control list (ACL) which specify the names of the users and the types of access associate with each of the user.

The main problem with the access list is their length. If we want to allow everyone to read a file, we must list all the users with the read access. This technique has two undesirable consequences: Constructing such a list may be tedious and unrewarding task, especially if we do not know in advance the list of the users in the system.

Previously, the entry of the any directory is of the fixed size but now it changes to the variable size which results in complicates space management. These problems can be resolved by use of a condensed version of the access list.

To condense the length of the access-control list, many systems recognize three classification of users in connection with each file:

- **Owner:** Owner is the user who has created the file.
- **Group:** A group is a set of members who has similar needs and they are sharing the same file.
- **Universe:** In the system, all other users are under the category called universe.

The most common recent approach is to combine access-control lists with the normal general owner, group, and universe access control scheme.

For example: Solaris uses the three categories of access by default but allows access-control lists to be added to specific files and directories when more fine-grained access control is desired.

Other Protection Approaches:

The access to any system is also controlled by the password. If the use of password are is random and it is changed often, this may be result in limit the effective access to a file. The use of passwords has a few disadvantages:

- The number of passwords are very large so it is difficult to remember the large passwords.
- If one password is used for all the files, then once it is discovered, all files are accessible; protection is on all-or-none basis.

Multimedia Files:

We can say multimedia are digital movies, videos, clips, music, etc. Multimedia literally means more than one medium or multiple medium. Now-a-day, video clips, digital movies and music are common way to present the information and entertainment using our computer system.

We can store our desired audio and video files and able to play it later on demand. Audio and video files can be stored on the disk. The characteristics of the audio and video files are very different from the traditional text files that the current file system were designed for.

Therefore, a new type of file system must be needed just to handle the audio and video files. Storing the audio and video files and playing back on demand puts new demands on the scheduler and other parts of the OS as well.

Therefore, there is a need of designing an OS, that are also able to play the multimedia files and those operating systems are also called as multimedia operating systems. Now-a-day, almost all the operating system can be called as multimedia operating system.

In most computer systems, an ordinary text file consists of a linear sequence of bytes without any structure that the OS knows about or cares about. Now, with multimedia files, the situation becomes more complicated.

Therefore, to start with multimedia files, audio and video are completely different. They are captured by some distinct devices (CCD chip versus microphone), have a different internal structure (audio has 44100 samples/second and video has 25-30 frames/second), and they are played back by different devices (monitor versus loudspeakers).

Unit VIII: Security Management – Operating System

Introduction:

File system often contain information that is highly valuable to their users. Protecting these information against unauthorized usage is the major concern of all file system. We generally use the term security to refer to the specific operating system mechanism used to safe guard information in the computer.

Security Problems:

Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system.

If a computer program is run by an unauthorized user, then he/she may cause severe damage to computer or data stored in it. So a computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc.

Causes of Security Problems:

1. Hardware Error:

A hardware error is a malfunction of a hardware component in a computer system. The hardware components contain error detection mechanisms that can detect when a hardware error condition exists.

Hardware errors can be classified as either **corrected errors, or uncorrected errors.**

a. Corrected Error:

A corrected error is a hardware error condition that has been corrected by the hardware or the firmware by the time that the operating system is notified about the presence of the error condition.

b. Uncorrected Error:

An uncorrected error is a hardware error condition that cannot be corrected by the hardware or the firmware. Uncorrected errors are classified as either fatal or nonfatal.

- ***A fatal hardware error*** is an uncorrected or uncontained error condition that is determined to be unrecoverable by the hardware. When a fatal uncorrected error occurs, the operating system generates a bug check to contain the error.

- ***A nonfatal hardware error*** is an uncorrected error condition from which the operating system can attempt recovery by trying to correct the error. If the operating system cannot correct the error, it generates a bug check to contain the error.

2. Software Error:

A software bug is an error, flaw, failure or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

Most bugs arise from mistakes and errors made in either a program's source code or its design, or in components and operating systems used by such programs. A few are caused by compilers producing incorrect code.

A program that contains a large number of bugs, and/or bugs that seriously interfere with its functionality, is said to be buggy (defective).

Bugs trigger errors that may have ripple effects. Bugs may have subtle effects or cause the program to crash or freeze the computer. Others qualify as security bugs and might, for example, enable a malicious user to bypass access controls in order to obtain unauthorized privileges.

3. Human Error:

Errors are the result of actions that fail to generate the intended outcomes. They are categorized according to the cognitive processes involved towards the goal of the action and according to whether they are related to planning or execution of the activity.

Actions by human operators can fail to achieve their goal in three different ways according to **Dr. J. Rasmussen.**

a. Lapse Error: Memory Lapse

- It is a run-time error caused by an oversight.
- The action has a result different from the one expected due to the memory.
- The error is not directly observable.
- The action plan designed by the person is correct but one or more of the actions scheduled are skipped and have not been performed.

b. Slip Error: Forgetfulness Or Involuntary Mistake

- It is a runtime error that concerns the level of skill.
- The actions are executed in a different way than planned, the person should know how to perform the task but it doesn't and executes it incorrectly.
- The error is directly observable.
- In this case, the action plan designed by the person is correct but one or more of the actions envisaged were conducted incorrectly.

c. Mistake: Perform The Wrong Things

- The error is not committed during the practical execution of the action plan, in fact the actions are carried out as planned, but there are other problems that lead to error.
- The actions have been carried out successfully, but all the plan prepared by the person is wrong.
- For this type of error we can have:
  - **Rule-Based Mistakes:** errors due to the choice of the wrong rule due to an erroneous perception of the situation, or omissions in the application of a rule.
  - **Knowledge-Based Mistakes:** mistakes due to lack of knowledge or incorrect application.

4. Intruders:

An Intruder is a person who attempts to gain unauthorized access to a system, to damage that system, or to disturb data on that system. In summary, this person attempts to violate Security by interfering with system Availability, data Integrity or data Confidentiality.

Types of Intruders:

a. Passive Attack:

A passive attack is a network attack in which a system is monitored and sometimes scanned for open ports and vulnerabilities. The purpose is solely to gain information about the target and no data is changed on the target.

Passive attacks include active reconnaissance and passive reconnaissance. In passive reconnaissance, an intruder monitors systems for vulnerabilities without interaction, through methods like session capture.

In active reconnaissance, the intruder engages with the target system through methods like port scans.

Methods of Passive Attacks:

War driving detects vulnerable Wi-Fi networks by scanning them from nearby locations with a portable antenna.

The attack is typically carried out from a moving vehicle, sometimes with GPS systems that hackers use to plot out areas with vulnerabilities on a map. War driving can be done just to steal an Internet connection or as a preliminary activity for a future attack.

In dumpster diving, intruders look for information stored on discarded computers and other devices or even passwords in trash bins. The intruders can then use this information to facilitate covert entry to a network or system.

An intruder might masquerade as an authorized network user and spy without interaction. With that access, an intruder might monitor network traffic by setting the network adapter to promiscuous mode.

A passive attack contrasts with an active attack, in which an intruder attempts to alter data on the target system or data en route for the target system.

b. Active Attack:

An active attack is a network exploit in which a hacker attempts to make changes to data on the target or data en route to the target.

Types of Active Attacks:

In a **masquerade attack**, the intruder pretends to be a particular user of a system to gain access or to gain greater privileges than they are authorized for.

A masquerade may be attempted through the use of stolen login IDs and passwords, through finding security gaps in programs or through bypassing the authentication mechanism.

In a **session replay attack**, a hacker steals an authorized user's log in information by stealing the session ID. The intruder gains access and the ability to do anything the authorized user can do on the website.

In a **message modification attack**, an intruder alters packet header addresses to direct a message to a different destination or modify the data on a target machine.

In a **denial of service (DoS) attack**, users are deprived of access to a network or web resource. This is generally accomplished by overwhelming the target with more traffic than it can handle.

In a **distributed denial-of-service (DDoS) exploit**, large numbers of compromised systems (sometimes called a botnet or zombie army) attack a single target.

Active attacks contrast with passive attacks, in which an unauthorized party monitors networks and sometimes scans for open ports and vulnerabilities.

The purpose is to gain information about the target and no data is changed. However, passive attacks are often preparatory activities for active attacks.

User Authentication:

The primary goal of authentication is to allow access to the legal system users and deny access to unauthorized users.

It is the responsibility of the Operating System to create a protection system which ensures that a user who is running a particular program is authentic. Operating Systems generally identifies/authenticates users using following three ways:

1. **Username / Password:** User need to enter a registered username and password with Operating system to login into the system.

2. **User card/key:** User need to punch card in card slot, or enter key generated by key generator in option provided by operating system to login into the system.

3. **User attribute (bio-metric) fingerprint/ eye retina pattern/ signature:** User need to pass his/her attribute via designated input device used by operating system to login into the system.

Password:

The most widely used method to prevent unauthorized access is to use passwords. A **password** is a string of characters used to authenticate a user to access a system. The password needs to be kept secret and is only intended for the specific user.

In computer systems, each password is associated with a specific username since many individuals may be accessing the same system.

Good passwords are essential to keeping computer systems secure. Unfortunately, many computer users don't use very secure passwords, such as the name of a family member or important dates - things that would be relatively easy to guess by a hacker.

One of the most widely used passwords - you guessed it - 'password.' Definitely not a good password to use. So what makes for a strong password?

1. **Longer Is Better:** A long password is much harder to break. The minimum length should be 8 characters, but many security experts have started recommending 12 characters or more.

2. **Avoid The Obvious:** A string like '0123456789' is too easy for a hacker, and so is 'LaDyGaGa'. We should also avoid all words from the dictionary.
3. **Mix It Up:** Use a combination of upper and lowercase and add special characters to make a password much stronger. A password like 'hybq4' is not very strong, but 'Hy%Bq&4$' is very strong.

Remembering strong passwords can be challenging. One tip from security experts is to come up with a sentence that is easy to remember and to turn that into a password by using abbreviations and substitutions. For example, 'My favorite hobby is to play tennis' could become something like Mf#Hi$2Pt%.

Password Vulnerability:

1. Password Vulnerability Due To Phishing:

This type of attack causes victims to believe they are accessing legitimate content, usually e-mail or websites, when in fact they are accessing fake content produced by the attackers.

This type of content usually leads victims to fill in existing login and password data from other legitimate sites or services, such as Google and Facebook, which when filled in, allows the attacker to store the passwords before redirecting the victims to a legitimate site.

How to Avoid:

Attackers often copy the image of sites almost perfectly, that are looking to steal passwords. But there are a few important items which cannot be copied, such as the site addresses and the links within it. Always check the links to make sure they belong to the desired location.

2. Brute Force Attack Puts Password At Risk:

A brute force attack is the name of the action performed on a website to test it with thousands of software, check against millions of passwords until you find the right one. It is a robot that randomly tries passwords to connect to the website.

How to Avoid:

No one can really prevent a robot from doing these actions, but it is possible to reduce and discourage such hackers. One of the first solutions is to increase the security of the website by forcing its members to create more complex passwords.

For example, a minimum of 8 characters, containing a combination of numbers and letters. This will make the task of the robot much more complex.

3. Dictionary or Wordlist Attack:

The dictionary-based attack or wordlist attack is also considered a brute-force attack. The attacker uses files containing thousands or even millions of words of the most varied types and languages and software that allows this list to be tested quickly until the victim's password is found or until the dictionary finishes.

How to Avoid:

Usually, the passwords present in dictionaries are not very extensive, that is, they have less than ten characters. To avoid becoming a victim of dictionary attacks use passwords that have more than 12 characters.

Like most attacks, the above attacks can be prevented by adopting some simple behavioral changes, and there are security solutions that can make this task even simpler.

4. Social Engineering:

Social engineering is somewhat similar to phishing attacks and is a widespread spying method aimed at gaining access to confidential data.

To extract confidential information, scammers very often exploit good faith, helpfulness, but also the insecurity of people. Whether over the phone, pretending to be someone else or the Internet, they are ready to do anything to get access to personal data.

How to Avoid:

Reveal as little personal information as possible; social networks are real mines of information. Be suspicious when asked for an email ID. Even emails from known senders can be falsified.

5. Malware Attack On Passwords Increasing By The Day

Malware is the most obvious and efficient tactic to steal passwords at the moment. Unlike most powerful viruses, they are not so apparent because their goal is to steal your data without you knowing or introduce a remote access Trojan horse to steal your credentials.

How to Avoid:

To prevent this from happening to you, keep your antivirus up to date, scan frequently, and avoid suspicious sites that are full of pop-up ads.

Encrypted Password:

Encryption is the conversion of electronic data into another form, called cipher-text, which cannot be easily understood by anyone except authorized parties.

The purpose of encryption is to protect the digital data stored on certain computer, system, or server such as email and digital storage sites using the algorithm. It alters the underlying character string of a file so that it cannot be (easily) comprehended.

Encrypting data requires a mechanism to reconstruct the character string that is typically done using a password. Only the authorized user can decrypt the file using various forms of combinations such as fingerprint, retina scan, or patterns.

There are two main types of encryption: asymmetric encryption (also called public-key encryption) and symmetric encryption.

Encryption has been fervently used by militaries to facilitate secret communication. With the advent of the internet for public use, it is now commonly used in protecting information.

The Computer Security Institute reported that in 2007, 71% of companies surveyed utilized encryption for some of their data in transit, and 53% utilized encryption for some of their data in storage

One Time passwords:

One-time passwords provide additional security along with normal authentication. In One-Time Password system, a unique password is required every time user tries to login into the system.

Once a one-time password is used, then it cannot be used again. One-time password are implemented in various ways.

1. **Random Numbers:** Users are provided cards having numbers printed along with corresponding alphabets. System asks for numbers corresponding to few alphabets randomly chosen.

2. **Secret Key:** User are provided a hardware device which can create a secret id mapped with user id. System asks for such secret id which is to be generated every time prior to login.

3. **Network Password:** Some commercial applications send one-time passwords to user on registered mobile/ email which is required to be entered prior to login.

Biometrics Password:

Biometrics are physical or behavioral human characteristics to that can be used to digitally identify a person to grant access to systems, devices or data.

Examples of these biometric identifiers are fingerprints, facial patterns, voice or typing cadence. Each of these identifiers is considered unique to the individual, and they may be used in combination to ensure greater accuracy of identification.

Because biometrics can provide a reasonable level of confidence in authenticating a person with less friction for the user, it has the potential to dramatically improve enterprise security.

Computers and devices can unlock automatically when they detect the fingerprints of an approved user. Server room doors can swing open when they recognize the faces of trusted system administrators.

Help desk systems might automatically pull up all relevant information when they recognize an employee's voice on the support line.

According to a recent Ping Identity survey, 92 percent of enterprises rank biometric authentication as an "effective" or "very effective" to secure identity data stored on premises, and 86 percent say it is effective for protecting data stored in a public cloud.

Another survey, released last year by Spiceworks, reports that 62 percent of companies are already using biometric authentication, and another 24 percent plan to deploy it within the next two years.

However, companies need to be careful about how they roll out their biometric authentication systems to avoid infringing on employee or customer privacy or improperly exposing sensitive information. After all, while it's easy to issue a new password when the old one has been compromised, we can't issue someone a new eyeball.

According to the Spiceworks survey, 48 percent cite the risks of stolen biometric data as a top security risk with the technology. Other barriers to adoption include costs, cited by 67 percent of respondents, followed by reliability concerns at 59 percent.

For companies specifically using biometrics to secure IT infrastructure in cloud, SaaS, on-prem and hybrid environments, adoption rates are even lower, according to the Ping Identity survey. Only 28 percent of companies use biometrics on premises, and even fewer, 22 percent, use it for cloud applications.

User Authorization:

Authorization is the process of giving someone permission to do or have something. In multi-user computer systems, a system administrator defines for the system which users are allowed access to the system and what privileges of use (such as access to which file directories, hours of access, amount of allocated storage space, and so forth).

Assuming that someone has logged in to a computer operating system or application, the system or application may want to identify what resources the user can be given during this session.

Thus, authorization is sometimes seen as both the preliminary setting up of permissions by a system administrator and the actual checking of the permission values that have been set up when a user is getting access.

We may assign a user several form of authorization:

1. **Read Authorization:** Allows reading but not modifications.
2. **Insert Authorization:** Allows insertion of new data but not modifications.
3. **Update Authorization:** Allows update or modifications but not deletion.
4. **Delete Authorization:** Allows deletion.

We may assign the user or process, all, none or combination of these types of authorization. Authorization deals with the access rights on the file.

Difference between Authentication and Authorization:

Authentication	Authorization
It is the process of verifying the identity of a user.	It is the process of checking whether the user has the access rights to the system.
It always proceeds to authorization.	It is the process of allowing an authenticated user access to resources.
It has two separate levels because all the requests coming through the IIS before it is handled.	It allows two ways to authorize the access to a given resources.
They have additional schemes like windows authentication, forms authentication and passport authentication.	The two ways are URL authorization and File authorization.

Program Threats:

Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks, then it is known as Program Threats. One of the common example of program threat is a program installed in a computer which

can store and send user credentials via network to some hacker. Following is the list of some well-known program threats.

1.Trojan Horse:

It is a written malware which can be installed to millions of computers. For this, a common practice requires to write some genuinely useful program and embed the malware inside it.

Games, Music players, Promo Viewer and anything with splashy graphics are likely candidates. People will then voluntarily download and install the application and as a free bonus they get the malware installed too. This approach is called a Trojan horse attack.

2.Trap Door:

If a program which is designed to work as required, have a security hole in its code and perform illegal action without knowledge of user then it is called to have a trap door. Such program traps user login credentials and stores them to send to malicious user who can later on login to computer and can access system resources.

3.Logic Bomb:

Logic bomb is a situation when a program misbehaves only when certain conditions met otherwise it works as a genuine program. It is harder to detect.

4.Stack Buffer Overflow:

In software, a stack buffer overflow or stack buffer overrun occurs when a program writes to a memory address on the program's call stack outside of the intended data structure, which is usually a fixed-length buffer.

Stack buffer overflow bugs are caused when a program writes more data to a buffer located on the stack than what is actually allocated for that buffer.

This almost always results in corruption of adjacent data on the stack, and in cases where the overflow was triggered by mistake, will often cause the program to crash or operate incorrectly.

Stack buffer overflow is a type of the more general programming malfunction known as buffer overflow (or buffer overrun). Overfilling a buffer on the stack is more likely to derail program execution than overfilling a buffer on the heap because the stack contains the return addresses for all active function calls.

Stack buffer overflow can be caused deliberately as part of an attack known as stack smashing. If the affected program is running with special privileges, or accepts data from untrusted network hosts (e.g. a webserver) then the bug is a potential security vulnerability.

If the stack buffer is filled with data supplied from an untrusted user then that user can corrupt the stack in such a way as to inject executable code into the running program and take control of the process.

This is one of the oldest and more reliable methods for attackers to gain unauthorized access to a computer.

System Threats:

System threats refers to misuse of system services and network connections to put user in trouble. System threats can be used to launch program threats on a complete network called as program attack.

System threats creates such an environment that operating system resources/ user files are misused. Following is the list of some well-known system threats.

1. Worm:

Worm is a process which can choked down a system performance by using system resources to extreme levels. A Worm process generates its multiple copies where each copy uses system resources, prevents all other processes to get required resources. Worm's processes can even shut down an entire network.

2. Port Scanning:

Port scanning is a mechanism or means by which a hacker can detects system vulnerabilities to make an attack on the system.

3. Denial of Service:

Denial of service attacks normally prevents user to make legitimate use of the system. For example, a user may not be able to use internet if denial of service attacks browser's content settings.

4. Virus:

Virus as name suggest can replicate themselves on computer system. They are highly dangerous and can modify/delete user files, crash systems. A virus is generally a small

code embedded in a program. As user accesses the program, the virus starts getting embedded in other files/ programs and can make system unusable for user.

Types of Virus:

a. Companion Viruses
b. Executable Program Viruses
c. Memory Resident Viruses
d. Boot Sector Viruses
e. Device Driver Viruses
f. Macro Viruses
g. Source Code Viruses

Cryptography:

Cryptography is a method to transform (encrypt) information in such a way that it cannot be understood by anyone except the intended recipient, who possesses the means to reverse the transformation (decrypt).

Encryption means modifying the data such a way that it become useless or unidentifiable or unreadable to anyone or intruder but only the desired receiver can view the actual data.

The original unencrypted text is called plain text or clear text. It can be encrypted using some encryption method parameterized by encryption key. The result is called cipher text.

The cipher text may be stored or transmitted over the insecure communication medium. Now, the plain text (i.e. original text) can be obtained by decrypting the cipher text message using the decrypting key. So, use of key to reverse this encryption process and return the data to its original form is called decryption.

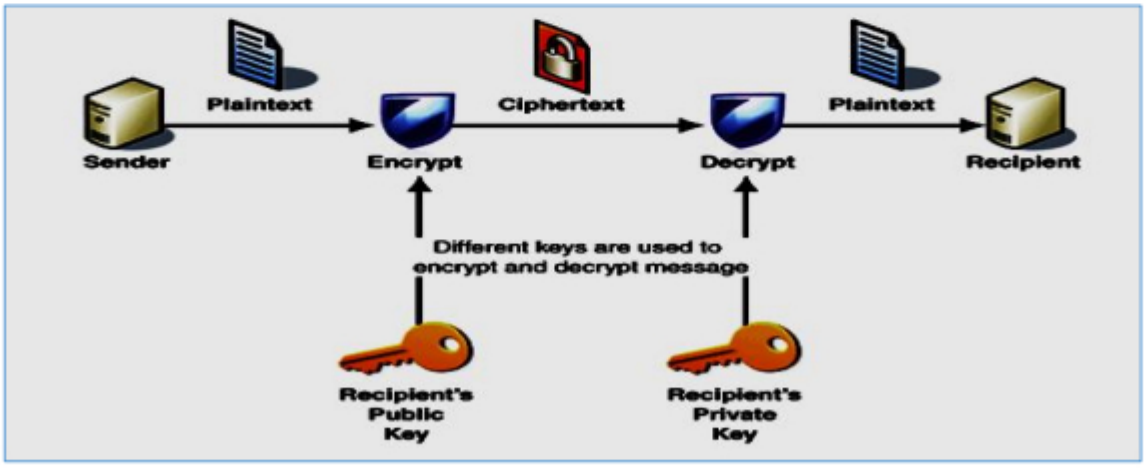


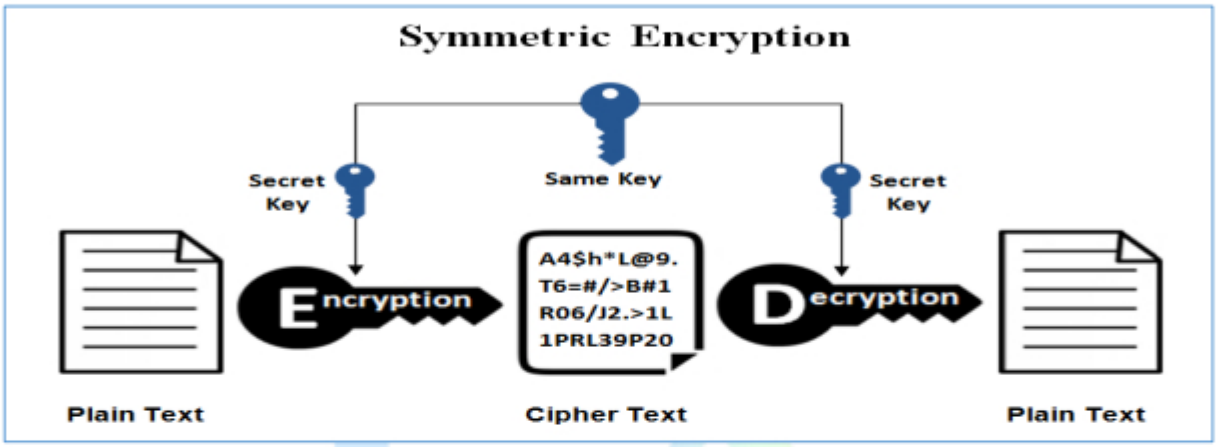
Fig: Cryptography System

Types of Cryptography:

1. Symmetric Key Cryptography:

Symmetric key cryptography is a type of cryptography in which the single common key is used by both sender and receiver for the purpose of encryption and decryption of a message.

This system is also called private or secret key cryptography and AES (Advanced Encryption System) is the most widely uses symmetric key cryptography.

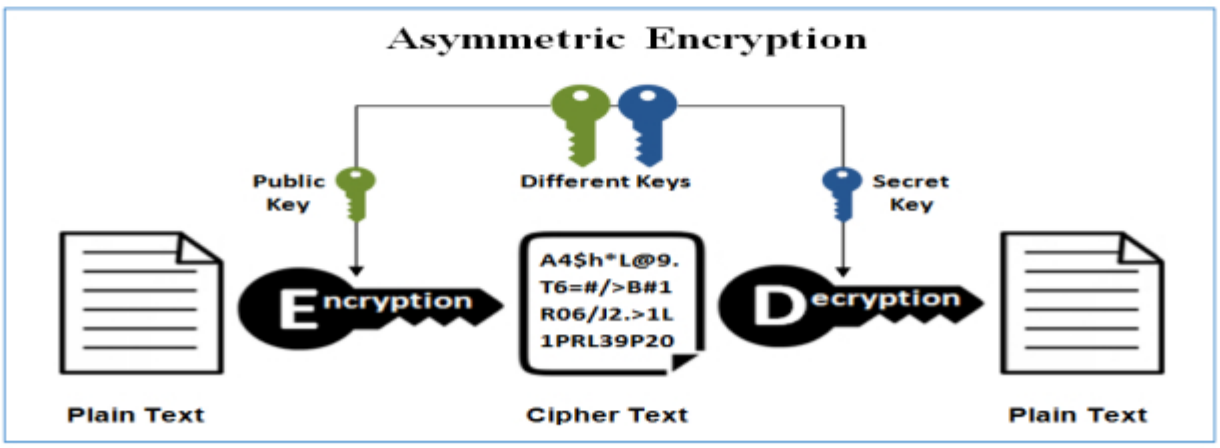


The symmetric key system has one major drawback that the two parties must somehow exchange the key in a secure way as there is only one single key for encryption as well as decryption process.

Types: AES (Advanced Encryption Standard), DES, Triple DES, RC2, RC4, RC5, IDEA, Blowfish, Stream cipher, Block cipher, etc. are the types of symmetric key cryptography.

2. Asymmetric Key Cryptography:

Asymmetric Key Cryptography is completely different and a more secure approach than symmetric key cryptography. In this system, every user uses two keys or a pair of keys (private key and public key) for encryption and decryption process.



Private Key is kept as a secret with every user and public key is distributed over the network so if anyone wants to send message to any user can use those public keys.

Either of the key can be used to encrypt the message and the one left is used for decryption purpose. Asymmetric key cryptography is also known as public key cryptography and is more secure than symmetric key. RSA is the most popular and widely used asymmetric algorithm.

Types: RSA, DSA, PKCs, Elliptic Curve techniques, etc. are the common types of asymmetric key cryptography.

Protection Mechanism:

Protection refers to a mechanism which controls the access of programs, processes, or users to the resources defined by a computer system. We can take protection as a helper to multi programming operating system, so that many users might safely share a common logical name space such as directory or files.

Protection Domain:

A protection domain is a grouping of a code source and permissions that is, a protection domain represents all the permissions that are granted to a particular code source. In the default implementation of the Policy class, a protection domain is one grant entry in the file.

When associated with a class, a protection domain means that the given class was loaded from the site specified in the code source, was signed by the public keys specified in the code source, and should have permission to perform the set of operations represented in the permission collection object.

Each class in the virtual machine may belong to one and only one protection domain, which is set by the class loader when the class is defined.

However, not all class loaders have a specific protection domain associated with them: classes that are loaded by the primordial class loader have no protection domain. In particular, this means that classes that exist as part of the system class path (that is, the Java API classes) have no explicit protection domain.

Access Control List (ACL):

Access control list (ACL) refers to the permissions attached to an object that specify which users are granted access to that object and the operations it is allowed to perform. Each entry in an access control list specifies the subject and an associated operation that is permitted.

File system ACL is a data structure that holds entries that specify individual user or group rights to system objects such as processes, files and programs. These entries are referred to as access control entities. Each system object is associated with a security attribute that identifies its access control list.

The ACL has an entry for each system user that defines the user's privileges, such as reading a file, writing to a file or executing a file. The operating systems that use ACL include Novell's Netware, Microsoft Windows NT/2000, Digital's OpenVMS and UNIX-based systems.

When a subject requests an object in an ACL-based security model, the OS initially checks the ACL for an applicable entry to decide whether the requested operation is authorized. The ACL model is applicable to both individual entities and the collection of objects within the system hierarchy.

Capabilities:

Capabilities achieve their objective of improving system security by being used in place of forgeable references. A forgeable reference (for example, a path name) identifies an object, but does not specify which access rights are appropriate for that object and the user program which holds that reference.

Consequently, any attempt to access the referenced object must be validated by the operating system, based on the ambient authority of the requesting program, typically via the use of an access control list (ACL).

Instead, in a system with capabilities, the mere fact that a user program possesses that capability entitles it to use the referenced object in accordance with the rights that are specified by that capability.

In theory, a system with capabilities removes the need for any access control list or similar mechanism by giving all entities all and only the capabilities they will actually need.

A capability is typically implemented as a privileged data structure that consists of a section that specifies access rights, and a section that uniquely identifies the object to be accessed.

The user does not access the data structure or object directly, but instead via a handle. In practice, it is used much like a file descriptor in a traditional operating system (a traditional handle), but to access every object on the system.

Capabilities are typically stored by the operating system in a list, with some mechanism in place to prevent the program from directly modifying the contents of the capability (so as to forge access rights or change the object it points to).

Some systems have also been based on capability-based addressing (hardware support for capabilities), such as Plessey System 250.

Programs possessing capabilities can perform functions on them, such as passing them on to other programs, converting them to a less-privileged version, or deleting them. The operating system must ensure that only specific operations can occur to the capabilities in the system, in order to maintain the integrity of the security policy.

Trusted System:

In the security engineering subspecialty of computer science, a trusted system is a system that is relied upon to a specified extent to enforce a specified security policy. This is equivalent to saying that a trusted system is one whose failure would break a security policy (if a policy exists that the trusted system is trusted to enforce).

The meaning of the word "trust" is critical, as it does not carry the meaning that might be expected in everyday usage.

A system trusted by a user, is one that the user feels safe to use, and trusts to do tasks without secretly executing harmful or unauthorized programs; while trusted computing refers to whether programs can trust the platform to be unmodified from that expected, whether or not those programs are innocent, malicious or execute tasks that are undesired by the user.

Trusted system can also be seen as level base security system where protection is provided and handled according to different levels. This is commonly found in military, where information is categorized as unclassified(U), confidential(C), Secret(S), Top secret(TS) and beyond. These also enforces the policies of No-read up and No-write down.

Common Uses of Trusted System:

- A trusted system can protect malicious attacks from future bugs or viruses.
- The code of a trusted system is passed through rigorous analysis and development
- A trusted system and an untrusted system can share a similar foundation

Common Misuses of Trusted System:

- A trusted system cannot withstand attacks from future malware attack

Unit IX: Distributed Operating System – Operating System

Introduction of Distributed Operating System:

A computing system composed of large numbers of CPUs connected by a network is known as the distributed system.

In other word, a distributed system can be defined as a collection of autonomous computers linked by a network with software designed to produce an integrated computing facility (or equipped with distributed system software).

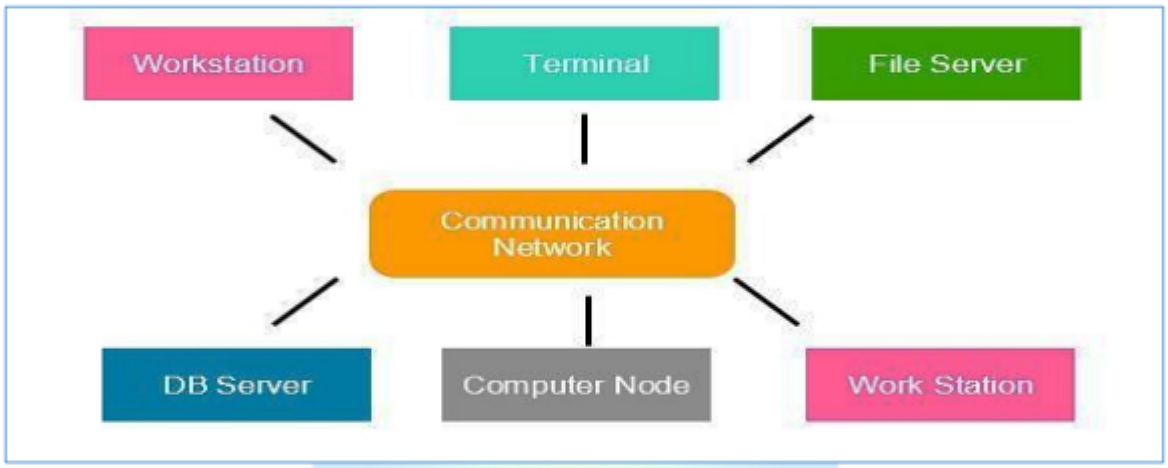


Fig: Distributed Operating System

Some of the Examples of Distributed Operating Systems:

1. IRIX operating system; is the implementation of UNIX System V, Release 3 for Silicon Graphics multiprocessor workstations.
2. DYNIX operating system running on Sequent Symmetry multiprocessor computers.
3. AIX operating system for IBM RS/6000 computers.
4. Solaris operating system for SUN multiprocessor workstations.
5. Mach/OS is a multi-threading and multitasking UNIX compatible operating system;
6. OSF/1 operating system developed by Open Foundation Software: UNIX compatible.

Advantages of Distributed System over Centralized System:

1. Speed:

When used to implement parallel processing where only goal is to achieve maximum speed on a single problem, distributed systems can achieve very high speed as compared to the centralized ones.

2. Inherent Distribution:

Another reason for building a distributed system is that some applications are inherently distributed. Banking, Airline reservation etc. are examples of the applications that are inherently distribute. When all the branches of a bank are connected, we have a commercial distributed system.

3. Reliability:

Another potential advantage of a distributed system over a centralized one is higher reliability. By distributing the workload over many machines, a single chip failure will bring down at most one machine, leaving the rest intact.

Ideally, if 5 percent of the machines are down at any moment, the system should be able to continue to work with a 5 percent loss in performance. For critical applications, such as control of nuclear reactors or aircraft, using a distributed system to achieve high reliability may be a dominant consideration.

4. Incremental Growth:

Incremental growth is also potentially a big plus point. Often a company will buy a mainframe with the intention of doing all its work on it. If the company prospers and the workload grows, at a certain point the mainframe will no longer be adequate.

The only solutions are to either replace the mainframe with a larger one (if it exists), or add a second mainframe. Both of these can cause management difficulties with the company's operations.

In contrast, with a distributed system, it may be possible to simply add more processors to the system, thus allowing it to expand gradually as the need arises.

5. Open System:

This is the most important point and the most characteristic point of a distributed system. Since it is an open system it is always ready to communicate with other systems. An open system that scales has an advantage over a perfectly closed and self-contained system.

Advantages of Distributed System over Independent PCs:

1. Resource Sharing:

- Data Sharing: Share common files or data
- Device Sharing: Printer sharing, etc.
- Software Sharing: Online Reservation, Banking, etc.

2. Flexibility/Scalability/Openness:

Distributed system can be extended or contracted i.e. addition or the removal of various hardware peripherals is possible as per the requirement or the operation of organization.

Similarly, operating system features, communication protocols and resource sharing services can also be customized (changed or modified). These all are possible without disruption of or duplication of an existing system.

3. Concurrency:

Many server processes run concurrently. Each responding to different requests from client processes or many users simultaneously invoke commands or interact with application programs.

4. Reliability:

If one computer of distributed system crashes, the system can still operate smoothly without interruption.

5. Fault Tolerance:

Ability to tolerate the fault that occurs when crashes. The system is easy to trace back after crash (i.e. recovery is easy without disruption of data and information).

6. Transparency (Feeling of a single system):

Transparency is defined as the concealment (to hide something carefully) from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole (i.e. single system) rather than as collection of independent PCs.

Disadvantage of Distributed System:

1. Complexity:

A distributed system that hides the distributed nature from the user and provides an acceptable level of performance, reliability, availability is inherently more complex than a centralized system.

The fact that data can be replicated also adds an extra level of complexity to the distributed system.

If the software does not handle data replication adequately, there will be degradation in availability, reliability and performance compared with the centralized system, and the advantages we cites above will become disadvantages.

2. Cost:

Increased complexity means that we can expect the procurement and maintenance costs *for* a distributed system to be higher than those for a centralized system.

Furthermore, a distributed system requires additional hardware to establish a network between sites. There are ongoing communication costs incurred with the use of this network.

There are also additional labor costs to manage and maintain the local systems and the underlying network.

3. Security:

In a centralized system, access to the data can be easily controlled. However, in a distributed system not only does access to replicated data have to be controlled in multiple locations but also the network itself has to be made secure.

In the past, networks were regarded as an insecure communication medium. Although this is still partially true, significant developments have been made to make networks more secure.

4. Integrity Control More Difficult:

System integrity refers to the validity and consistency of stored data. Integrity is usually expressed in terms of constraints, which are consistency rules that the database is not permitted to violate.

Enforcing integrity constraints generally requires access to a large amount of data that defines the constraints. In a distributed system, the communication and processing costs that are required to enforce integrity constraints are high as compared to centralized system.

5. Lack of Standards:

Although distributed systems depend on effective communication, we are only now starting to see the appearance of standard communication and data access protocols.

This lack of standards has significantly limited the potential of distributed systems. There are also no tools or methodologies to help users convert a centralized system into a distributed system.

6. Lack of Experience:

General-purpose distributed system have not been widely accepted, although many of the protocols and problems are well understood. Consequently, we do not yet have the same level of experience in industry as we have with centralized systems. For a prospective adopter of this technology, this may be a significant deterrent.

7. System Design More Complex:

Besides the normal difficulties of designing a centralized database, the design of a distributed database has to take account of fragmentation of data, allocation of fragmentation to specific sites, and data replication.

Hardware and Software Concepts for Distributed System:

1. Hardware Concepts:

Distributed system is different from other systems due to presence of multiple CPUs. There are several ways the hardware can be organized, especially in terms of how they are interconnected and how they communicate.

There exists various classification schemes but the most popular one is introduced by Flynn (1972) that is based on two characteristics:

a. **The Number Of Instruction Stream**
b. **The Number Of Data Streams**

A computer with a **single instruction stream and a single data stream** is called **SISD**. All traditional uniprocessor computers (i.e., those having only one CPU) fall in this category, from personal computers to large mainframes.

The next category is **SIMD, single instruction stream, multiple data stream**. This type refers to array processors with one instruction unit that fetches an instruction, and then commands many data units to carry it out in parallel, each with its own data.

These machines are useful for computations that repeat the same calculation on many sets of data, for example, adding up all the elements of 64 independent vectors. Some supercomputers are SIMD.

The next category is **MISD, multiple instruction stream, single data stream**. No known computers fit this model. Finally, **comes MIMD, multiple instruction stream, multiple data stream**, which essentially means a group of independent computers, each with its own program counter, program, and data.

All distributed systems are MIMD, so this classification system is not tremendously useful for our purposes.

Although Flynn stopped here, we will go further and divide all MIMD computers into two groups: those that have shared memory, usually called **multiprocessors**, and those that do not, sometimes called **multicomputer**.

The essential difference is this: in a multiprocessor, there is a single virtual address space that is shared by all CPUs. If any CPU writes, for example, the value 44 to address 1000, any other CPU subsequently reading from its address 1000 will get the value 44. All the machines share the same memory.

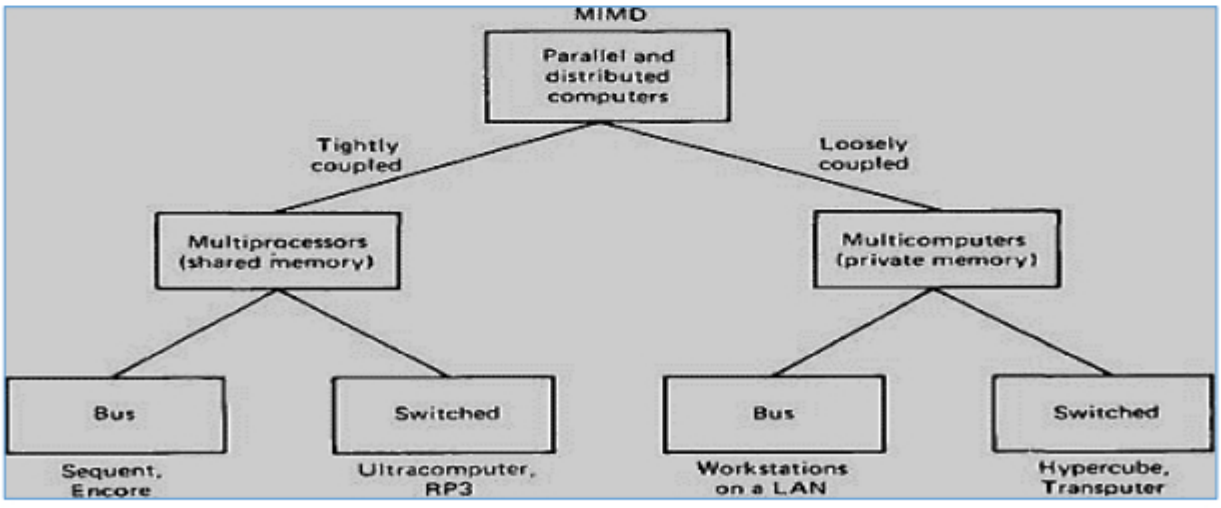
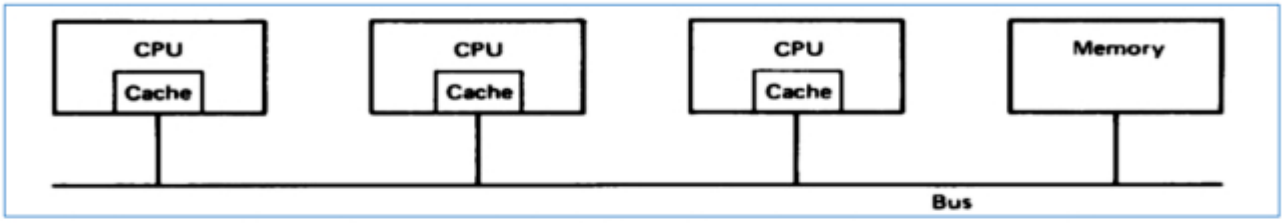


Fig: A Taxonomy of Parallel and Distributed Computer Systems

Each of these category can be further divided into bus and switched based on architecture of the interconnection network.

By bus we mean that there is single network, backbone, bus, cable, or other medium that connects all machine. Cable television uses a scheme like this, the cable company runs the cable down the street, and all the subscribers have taps running to it from their television sets.



Switched system do not have a single backbone like cable television, instead there are individual wires from machine to machine, with many different wiring patterns in use.

Messages move along the wires, with an explicit switching decisions made at each step to route the message along one of the outgoing wires. The world-wide public telephone system is organized in this way.

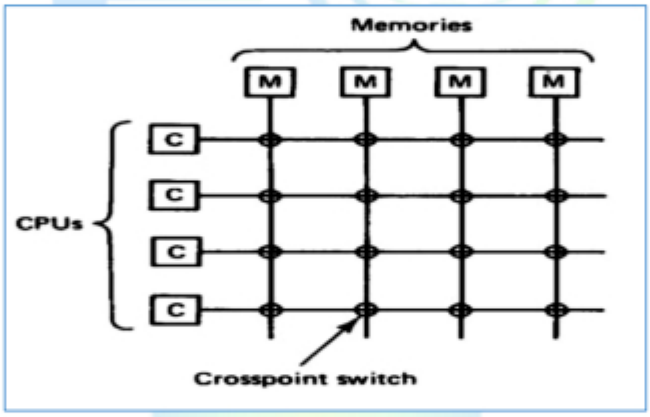


Fig: Crossbar Switch

2. Software Concepts:

In centralized computer systems, the operating system is the main system software layer. It manages all the basic resources in the systems and provides essential services to application programs and users. These include:

a. Basic resource management such as memory allocation and protection, process creation and processor scheduling, peripheral device handling, etc.
b. User and application services such as; user authentication and access control, file management, etc.

In distribute system, a part from those above ones other series are to be provided to application programs in a manner that enables new services to be conveniently added. Hence, the kernel's function is restricted to basic resource management:

- Memory allocation and protection.
- Process creation and processor scheduling.
- Peripheral device handling
- Inter-process communication

And a new class of software components are introduced called open services to provide all other shared resources and services.

Communication in Distributed System:

1. ATM (Asynchronous Transfer Mode) Network:

ATM has been designed to carry a wide variety of data including multimedia data such as voice and video. It is a fast packet switching network based on the method of packet routing known as cell relay, which can operate faster than conventional packet switching.

It achieves its speed by avoiding flow control and error checking at the intermediate nodes in a transmission. ATM operates in a connected mode, but a connection can only be set up if sufficient resources are available. Once a connection is established, its quality (i.e. bandwidth and latency) can be guaranteed.

The ATM service is structured in three layers as shown in figure below. The ATM layer provides a connection-oriented service that transmits fixed-length packet called cells. A connection consists of a sequence of virtual channels within virtual paths.

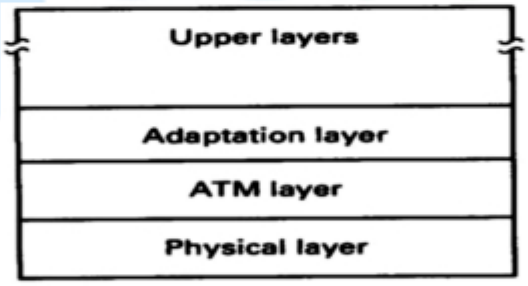


Fig: ATM Layers

A virtual path is bundle of virtual channels that are associated with the physical path between two switching nodes.

2. Layered Protocols:

All communication in distributed system is based on message passing. When a process A wants to communicate with a process B it first builds a message in its own address space. Then it executes a system call that causes the operating system to fetch the message and send it over the network to B.

To make it easier to deal with numerous levels and issues involved in communication, the international Standards Organization has developed a reference model, that clearly identifies the various level involved, gives them standard names and points out which level should do which job. This model is called OSI (Open System Interconnection Reference Model).

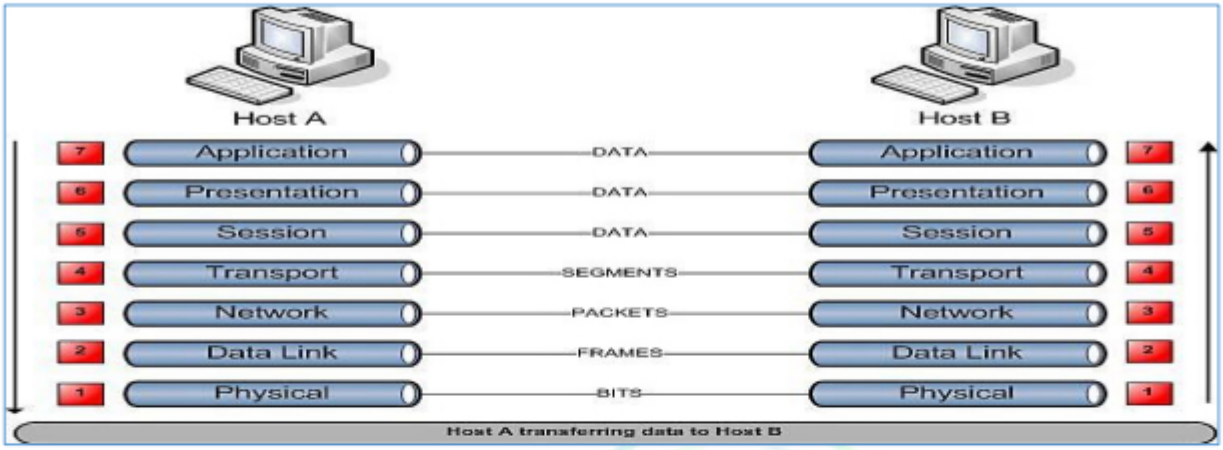


Fig: Network Protocol

3. Client Server Model:

For construction of distributed system, we introduce two patterns of communication that are most commonly used. They are:

a. Client Server communication for communication between pair of processes (client process and server process)
b. Group multicast communication model for communicating between groups of co-operating processes.

In client server model an exchange consists of:

- Transmission of request from a client process to a server process
- Execution of the request by the server (serving)
- Transmission of a reply to client

This pattern of communication involves the transmission of two messages and a specific form of synchronization of the client and server.

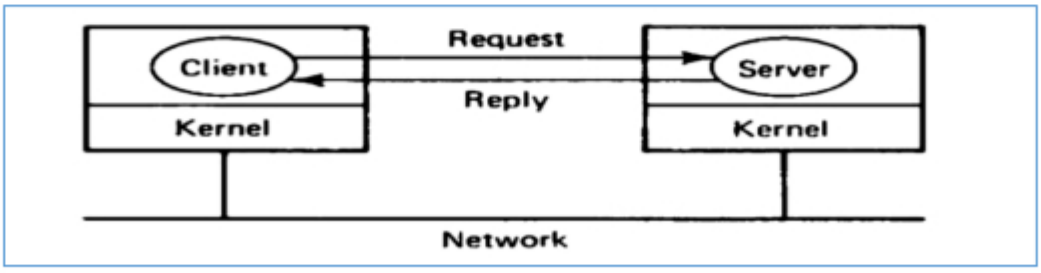


Fig: Client-Server Model

The server process must become aware of the request message sent in step 1 as soon as it arrive to activity issuing the request in the client process must be suspended (blocked), causing to wait after step 1 until the reply has been received in step 3.

4. Message Passing:

Distributed systems can be designed entirely in terms of message passing. Each message passing action involves the transmission by the sending process a set of data values (i.e. a message) through a specified communication mechanism (i.e. a channel or port) and the acceptance of the message by the receiving process.

Mechanism for Message Passing:

a. Synchronous (Blocking):

The sender waits after transmitting a message until the receiver has perform a receive operation.

b. Asynchronous (Non-Blocking):

Message is placed in queue of messages waiting for the receiver to accept them and the sending process can proceed immediately.

5. Remote Procedure Call (RPC):

A distributed program consists of a set of software components running in a number of computers in the network. The communication between the different computers are important (just like client-server model).

Another way of attacking the communication problem is RPC, suggested by Birrell and Nelson, which allow programs to call procedure located in other machines. When a process on machine 'A' calls a procedure in machine 'B', the calling process on 'A' is suspended (i.e. blocked) and the execution of the called procedure takes place on 'B'.

The information can be spent from sender to receiver in the parameters (just like in function in programming), and come back in the procedure result. This method is known as RPC.

6. Process in Distributed System:

Process in distributed system is same as process in operating system. Process refers to a job or a quantum of work dealt with as an entity. It is also known as task which has a life cycle which consists of creation, execution and termination.

It consists of address spaces as execution environment. The concept is thread is also same. A computer program that runs within a distributed system is called a distributed program (and distributed programming is the process of writing such programs).

There are many different types of implementations for the message passing mechanism, including pure HTTP, RPC-like connectors and message queues. Distributed computing also refers to the use of distributed systems to solve computational problems.

In distributed computing, a problem is divided into many tasks, each of which is solved by one or more computers, which communicate with each other via message passing.

7. Clock Synchronization:

Clock synchronization is necessary in distributed system in order to synchronize and co-ordinate timing between events because each computer in distributed system contains its own physical clock (timer).

The timer need to be read by the events of the process in order to operate (i.e. different events in distributed system need to use the same clock reference so, the physical clock in each system cannot be used).

In order to synchronize clock in operating system. Some mechanism is necessary:

- Network time protocol
- Happened before relationship
- Reference to time server, etc.

NTP defines architecture for a time service and a protocol to distribute time information over a wide variety of interconnected networks. Time server is used to provide the time to the device on request.

Time server acts as a reference for time in network. Happened before relationship deals with the ordering of events (i.e. what events occurs after completing which events).

Unit X: Case Study Issues – Operating System

DOS (Disk Operating System):

DOS was the first operating system used by IBM-compatible computers. It was originally available in two versions that were essentially the same, but marketed under two different names.

"PC-DOS" was the version developed by IBM and sold to the first IBM-compatible manufacturers. "MS-DOS" was the version that Microsoft bought the rights to, and was bundled with the first versions of Windows.

DOS uses a command line, or text-based interface, that allows the user to type commands. By typing simple instructions such as pwd (print working directory) and cd (change directory), the user can browse the files on the hard drive, open files, and run programs.

While the commands are simple to type, the user must know the basic commands in order to use DOS effectively (similar to UNIX). This made the operating system difficult for novices to use, which is why Microsoft later bundled the graphic-based Windows operating system with DOS.

The first versions of Windows (through Windows 95) actually ran on top of the DOS operating system. This is why so many DOS-related files (such as .INI, .DLL, and .COM files) are still used by Windows.

However, the Windows operating system was rewritten for Windows NT (New Technology), which enabled Windows to run on its own, without using DOS. Later versions of Windows, such as Windows 2000, XP, and Vista, also do not require DOS.

DOS is still included with Windows, but is run from the Windows operating system instead of the other way around. The DOS command prompt can be opened in Windows by selecting "Run..." from the Start Menu and typing cmd.

Advantages of MS-DOS:

1. It supports various computer languages.
2. It supports different disk like floppy, hard disk, CD, etc.
3. It is small sized operating system.
4. It initiates the concept of operating system during the time of booting.

Disadvantages of MS-DOS:

1. It has a command line user interface so it is totally command based operating system.
2. It has limited features to work with the modern computer system.
3. It is not so user friendly like windows system and cannot support advance computer peripheral devices even mouse.
4. It is a single user, single tasking operating system.

Types of DOS Commands:

1. Internal Commands:

Commands such as DEL, COPY, TYPE, etc. are the internal commands that remain stored in computer memory.

Internal commands are built in the COMMAND.COM files. It can be executed from any DOS prompt because each of the internal commands are memory resident. As long as the computer is running, we are ready to give internal commands.

2. External Commands:

Commands like FORMAT, DISKCOPY, etc. are the external commands and remain stored on the disk.

Commands that need external additional files with COMMAND.COM are external commands. We need additional corresponding files to run these commands. For example, we need tree.com file to run tree command.

Some Terms used in DOS:

1. Booting:

The process of loading system files into computer's memory from disk is called booting. It starts when the computer is turned on. It makes computer ready to work. In the booting process, the command interpreter and system files are loaded into computer's memory. There are two types of booting: cold booting and warm booting.

- **Cold Booting:** Booting process from off stage to on stage of the computer is performed by the switch on the computer.

- **Warm Booting:** Booting process during the time of running the computer system is warm booting. We have to perform this process when the computer hangs up. We can perform it by pressing the reset button or pressing ALT+CTRL+DEL.

2. File:

The systematic collection of related data or information or program instructions is known as file. A unique name is given for each file to identify. Such unique name is known as a file name.

A file name contains name and extension. The name helps to identify the file and extension helps to identify. The name helps to identify the file and extension helps to identify the type of file.

3. Directory:

Directory is the location to store files and sub-directories. It contains information about files stored on the disk like name, size, last date of modification, time of creation and disk volume label.

Examples of DOS Commands:

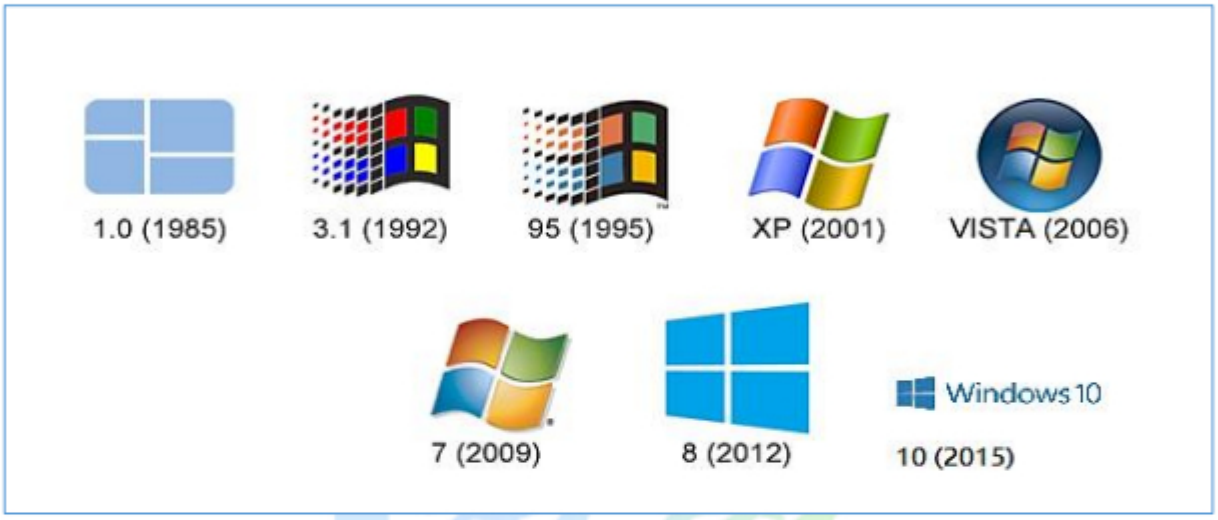
The DOS command-line on modern Windows systems can be accessed from the Start menu by typing cmd and pressing the Enter key.

Command	Function
CD and CHDIR	Changes the current working directory
DIR	Displays the contents of a directory
COPY	Makes a copy of selected file(s); can also be used in concatenation
DEL and ERASE	Deletes one or more files
EDIT	Launches the text editor
MOVE	Move one or more files to another directory
REN	Renames a file or directory

DELTREE	Deletes one or more files or directories
CLS	Clears the screen
DATE	Displays or changes the system date

Windows Operating System:

Windows is a **graphical operating system** developed by Microsoft. It allows users to view and store files, run the software, play games, watch videos, and provides a way to connect to the internet. It was released for both home computing and professional works.



Microsoft introduced the first version as 1.0. It was released for both home computing and professional functions of Windows on **10 November 1983**. Later, it was released on many versions of Windows as well as the current version, Windows 10.

In 1993, the first business-oriented version of Windows was released, which is known as **Windows NT 3.1**. Then it introduced the next versions, **Windows 3.5, 4/0**, and **Windows 2000**.

When the XP Windows was released by Microsoft in 2001, the company designed its various versions for a personal and business environment. It was designed based on standard x86 hardware, like **Intel** and **AMD processor**.

Accordingly, it can run on different brands of hardware, such as HP, Dell, and Sony computers, including home-built PCs.

Editions of Windows:

Microsoft has produced several editions of Windows, starting with Windows XP. These versions have the same core operating system, but some versions included advance features with an additional cost. There are two most common editions of Windows:

1. Windows Home:

Windows Home is basic edition of Windows. It offers all the fundamental functions of Windows, such as browsing the web, connecting to the Internet, playing video games, using office software, watching videos. Furthermore, it is less expensive and comes pre-installed with many new computers.

2. Windows Professional:

Windows Professional is also known as Window Pro or win Pro. It is an enhanced edition of Windows, which is beneficial for power users and small to medium-size businesses. It contains all features of Windows Home as well as the following:

a. Remote Desktop:

Windows Professional editions allow users to create a remote desktop connection. It provides users the option to connect with another computer remotely, including share the control of its mouse, keyboard, and view display.

It is mainly accessed with the help of port 3389. Additionally, we can also use the TeamViewer or VNC application to create a remote desktop connection.

b. Trusted Boot:

It provides security as encrypting to the boot loader and protects the computer from **rootkits** (Collection of software tools that allow users to enter another computer through an unauthorized way known as rootkits).

c. Bitlocker:

It allows users to encrypt a storage drive by using AES (Advanced Encryption Standard) algorithm. This feature is present in Windows 7, and Windows Vista (Only ultimate and Enterprise versions), including Windows Server 2008.

Business laptops or computers mainly use the Bitlocker feature to protect their data on the computer. As if our computer has been stolen, it is very difficult to break the Bitlocker password.

It can be unlocked by entering the correct password only. Furthermore, if we forget our Bitlocker password, it cannot be retrieved.

d. Windows Sandbox:

A sandbox is located on a computer, network, or an online service enables users to experiment or test computer security without interrupting the system.

e. Hyper-V:

It stands for a hypervisor, and developed by Microsoft Corporation on 26 June 2008. It is also called Windows Server Virtualization. Hyper-V is used for virtualization of x86-64 servers, running virtual machines and third party software like VirtualBox.

f. Group Policy Management:

An admin can specify group policies in an organization to manage different Windows users. It provides support for the systems that have more than 128 GB of RAM. Furthermore, it also offers more Windows update installation options as well as flexible scheduling and postponement around 34 days.

Features of Windows:

Microsoft Windows includes a lot of features to help users. Some of its excellent features are as follows:

1. Control Panel:

Windows provides a Control Panel feature that includes many tools to configure and manage the resources on their computer. For example, users can change settings for audio, video, printers, mouse, keyboard, network connections, date and time, power saving options, user accounts, installed applications, etc.

2. Cortana:

Windows 10 introduced a feature named Cortana, which is able to accept voice commands. It can perform various tasks such as it can answers our questions, search data on our computer, online purchases, set reminders, and appointments, etc.

Furthermore, it acts like other voice-activated services such as Google Assistant, Alexa, or Siri, including one more benefit of searching the information on our computer. To open the Cortana in Windows 10, press **Window key + S**.

3. File Explorer:

It is also known as Windows Explorer, which displays our files and folders on the computer. It allows users to browse the data on the hard drive, SSD and other inserted removable disks like pen drives and CDs, and we can manage the content according to the requirements such as delete, rename, search, and transfer the data.

4. Internet browser:

As the internet browser is very important to search for anything, view pages, online shopping, play games, watch videos, etc. Windows come with a pre-installed internet browser in Windows 10, the Edge internet browser is the default browser. Furthermore, Internet Explorer was the default browser in Microsoft Windows from the Windows edition 95 to 8.1 version.

5. Microsoft Paint:

Since November 1985, Microsoft Windows comes with pre-installed Microsoft Paint. It is a simple software to create, view, and edit an image. It offers several tools to draw an image, crop, resize, and save an image with a different file extension.

6. Taskbar:

Windows comes with a taskbar that displays currently opened programs, it also allows users to access any specific programs. Additionally, it includes the notification area on the right side that shows date and time, battery, network, volume, and other background running applications.

7. Start Menu:

Microsoft Windows contains a start menu to the left side of the taskbar. It displays programs and utilities that are installed on the computer. It can be simply opened by clicking on the Start menu button or pressing the start key on the keyboard.

8. Task Manager:

Windows includes the task manager feature that provides detail of the running applications or programs on the computer. We can also check how much of the system resources, such as RAM, CPU, disk I/O, are being used by each of the applications.

9. Disk Cleanup:

It is used to free up disk space with the help of deleting temporary or unnecessary files. It also helps to enhance the performance of the computer, and boost storage space to download the programs and documents. To open Disk Cleanup, follow the below steps:

- Open the File Explorer by pressing **Window + E.**
- Then, right-click on any disk drive and select Properties option from the drop-down list.
- Now, click on the **Disk Cleanup**.

Advantages of Windows Operating System:

1. Support For All Hardware:

As windows OS is used by 95% of users so most of the hardware vendors make drivers for windows.

2. Ease Of Use:

All versions of Microsoft Windows have something common in it which makes users easy to shift from one version to another. Windows 7 users have no difficulty in migrating to Windows 10 because most of the features of Windows 10 are the same as windows 7. The user interface of windows is also easy to use than UNIX and MAC.

3. Software Support:

Windows platform is best suited for game and software developers. Windows have large number audience so developers prefer to make utilities, games and software for windows OS. Linux users cannot make windows apps so it is better to use windows for developing apps.

4. Plug And Play Feature:

Most hardware can be detected automatically by plug and play feature. You do not need to manually install the hardware but it is ready to use when attached e.g. webcam, keyboard, mouse, mobile device etc.

5. Desktop And Touch Screen:

Windows 10 is made for both touch screen devices and desktop computers. The user interface of Windows 10 is made in such a way that it works better for any type of windows device.

Disadvantages of Windows Operating System

1. Virus Attacks:

Windows have a high amount of hacker attacks. The hackers can easily break windows security. So windows users are dependent on anti-virus software and have to pay monthly charges to companies to protect their data. Also, windows users have to update OS to keep up-to-date with security patches.

2. Most Of The Software Is Paid:

Most windows programs are paid e.g. games, graphics software (Photoshop), download manager (IDM) and other popular software are paid. You have to buy these software or pay a monthly fee to use them.

3. Rebooting A System:

If your system becomes slow in performance then you have to reboot it. If you load many programs at the same time then your system slows down and hangs up. The only solution for this is to reboot.

4. High Price:

Linux OS is open source and is free to use for everyone but windows OS has paid license and you cannot use windows OS legally free. The cost of buying a copy of windows OS is high as well. You also need to buy other Microsoft software e.g. MS Office to do regular office work on the computer.

5. High Computer Resources:

If you are installing windows OS then your computer should have high ram capacity, a lot of hard drive space and good graphics card. This is because of features that are used in windows. If you want to install graphics software i.e. Photoshop then 16 GB of ram is recommended.

6. Technical Support:

Windows support is not good for most users. Only some large organizations can get good support from the windows team. Common users have to search for forums to get their problem solved.

UNIX Operating System:

UNIX is a multi-user, multitasking operating system (OS) that was developed at AT&T Bell Labs in the late 1960s. It was designed to be used exclusively by programmers and became a leading operating system for workstations because of its portability, flexibility, and power.

Historically, it has been less popular in the personal computer market than professional settings, but numerous UNIX distributions like FreeBSD, Linux, and macOS brought UNIX principles into the mainstream.

UNIX systems are composed of several components that are bundled together to create a self-contained software application. These components include:

- A kernel, which is the source code that includes configuration, device drivers, file structures, memory management, system calls, etc.
- A development environment, which allows users to recreate the entire system from source code
- Documentation, including manual pages and larger files that detail major subsystems
- Commands, which allow users to navigate the operating system and perform specific actions, as well as maintenance and general utility applications

UNIX commands are numerous and case-sensitive. Some basic commands include:

- ls (lists files)
- mv (rename or move files)
- cd (changes directory)
- mkdir (creates new directory)
- history (shows history of previous commands)

Not all UNIX commands are universal, so some are specific to the device on which UNIX is being used. This list contains more examples of basic commands and how they are used.

UNIX Architecture:

The main concept that unites all the versions of UNIX is the following four basics:

1. Kernel:

The kernel is the heart of the operating system. It interacts with the hardware and most of the tasks like memory management, task scheduling and file management.

2. Shell:

The shell is the utility that processes our requests. When we type in a command at our terminal, the shell interprets the command and calls the program that we want. The shell uses standard syntax for all commands.

C Shell, Bourne Shell and Korn Shell are the most famous shells which are available with most of the Unix variants.

3. Commands and Utilities:

There are various commands and utilities which we can make use of in our day to day activities. **cp**, **mv**, **cat** and **grep**, etc. are few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various options.

4. Files and Directories:

All the data of UNIX is organized into files. All files are then organized into directories. These directories are further organized into a tree-like structure called the **filesystem**.

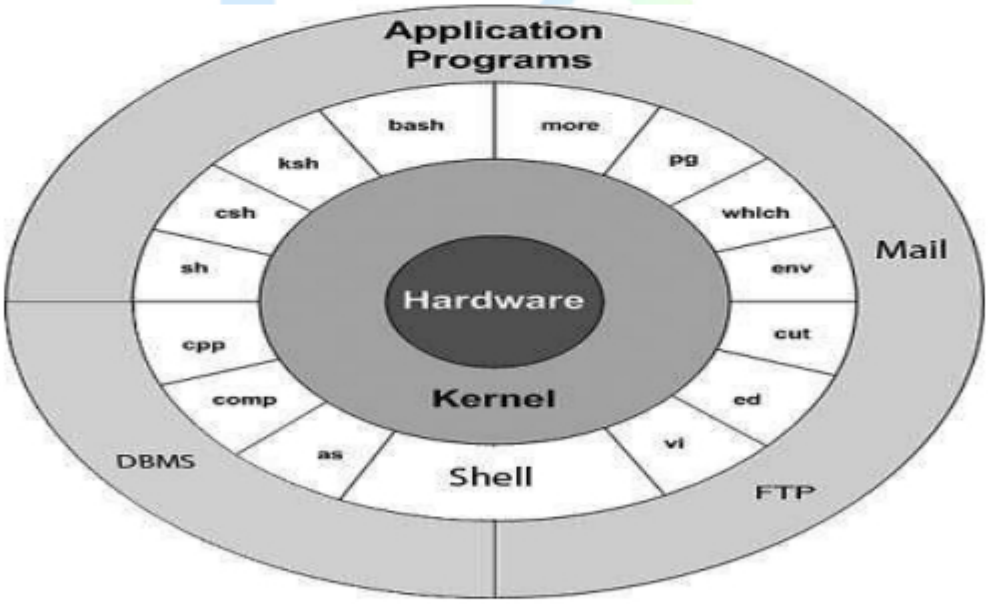


Fig: UNIX Architecture

Advantages of UNIX Operating System:

1. Full multitasking with protected memory. Multiple users can run multiple programs each at the same time without interfering with each other or crashing the system.

2. Very efficient virtual memory, so many programs can run with a modest amount of physical memory.
3. Access controls and security. All users must be authenticated by a valid account and password to use the system at all. All files are owned by particular accounts. The owner can decide whether others have read or write access to his files.
4. A rich set of small commands and utilities that do specific tasks well -- not cluttered up with lots of special options. Unix is a well-stocked toolbox, not a giant do-it-all Swiss Army Knife.
5. Ability to string commands and utilities together in unlimited ways to accomplish more complicated tasks -- not limited to preconfigured combinations or menus, as in personal computer systems.
6. A powerfully unified file system. Everything is a file: data, programs, and all physical devices. Entire file system appears as a single large tree of nested directories, regardless of how many different physical devices (disks) are included.
7. A lean kernel that does the basics for you but doesn't get in the way when you try to do the unusual.
8. Available on a wide variety of machines - the most truly portable operating system.
9. Optimized for program development, and thus for the unusual circumstances that are the rule in research.

Disadvantages of UNIX Operating System:

1. The traditional command line shell interface is user hostile -- designed for the programmer, not the casual user.
2. Commands often have cryptic names and give very little response to tell the user what they are doing. Much use of special keyboard characters - little typos have unexpected results.
3. To use UNIX well, you need to understand some of the main design features. Its power comes from knowing how to make commands and programs interact with each other, not just from treating each as a fixed black box.
4. Richness of utilities (over 400 standard ones) often overwhelms novices. Documentation is short on examples and tutorials to help you figure out how to *use* the many tools provided to accomplish various kinds of tasks.

Linux Operating System:

The Linux OS was developed by Linus Torvalds in 1991, which sprouted as an idea to improve the UNIX OS. He suggested improvements but was rejected by UNIX designers. Therefore, he thought of launching an OS, designed in a way that could be modified by its users.

Linux is an open-source operating system like other operating systems such as Microsoft Windows, Apple Mac OS, iOS, Google android, etc. An operating system is a software that enables the communication between computer hardware and software.

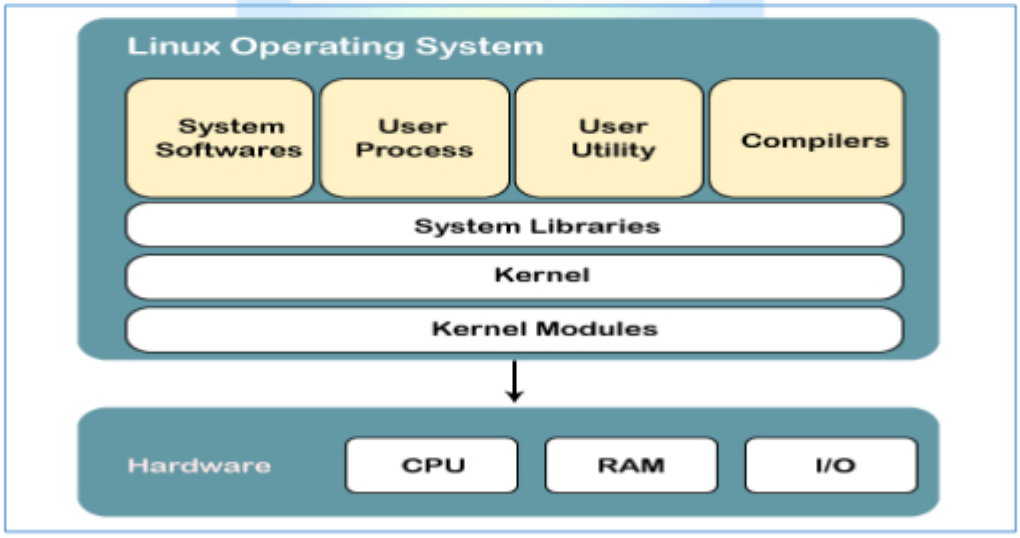
It conveys input to get processed by the processor and brings output to the hardware to display it. This is the basic function of an operating system. Although it performs many other important tasks, let's not talk about that.

Linux is around us since the mid-90s. It can be used from wristwatches to supercomputers. It is everywhere in our phones, laptops, PCs, cars and even in refrigerators. It is very much famous among developers and normal computer users.

Nowadays, Linux is the fastest-growing OS. It is used from phones to supercomputers by almost all major hardware devices.

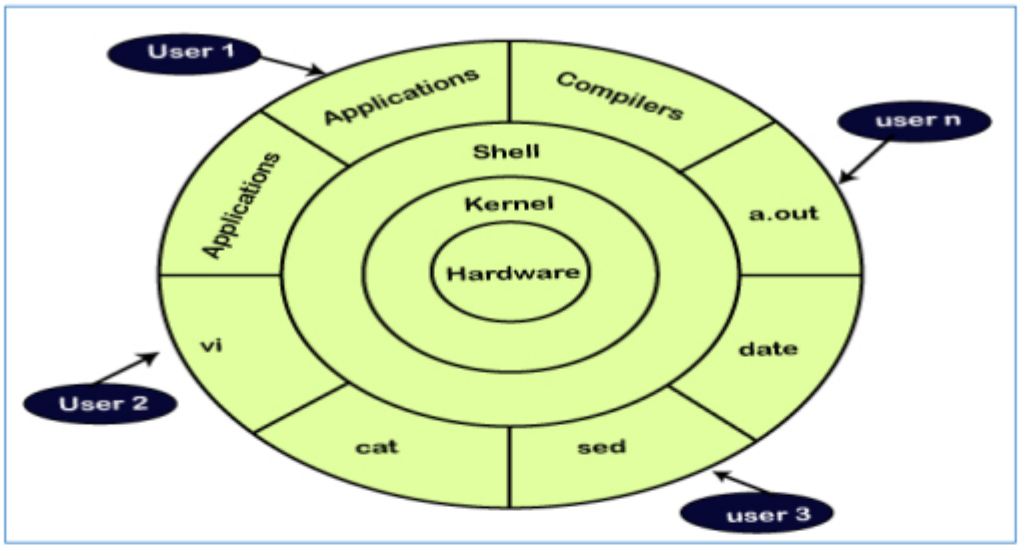
Structure of Linux Operating System:

An operating system is a collection of software, each designed for a specific function. Linux OS has following components:



1. Kernel:

Linux kernel is the core part of the operating system. It establishes communication between devices and software. Moreover, it manages system resources. It has four responsibilities:



a. Device Management:

A system has many devices connected to it like CPU, a memory device, sound cards, graphic cards, etc. A kernel stores all the data related to all the devices in the device driver (without this kernel won't be able to control the devices).

Thus kernel knows what a device can do and how to manipulate it to bring out the best performance. It also manages communication between all the devices. The kernel has certain rules that have to be followed by all the devices.

b. Memory management:

Another function that kernel has to manage is the memory management. The kernel keeps track of used and unused memory and makes sure that processes shouldn't manipulate data of each other using virtual memory addresses.

c. Process management:

In the process, management kernel assigns enough time and gives priorities to processes before handling CPU to other processes. It also deals with security and ownership information.

d. Handling System Calls:

Handling system calls means a programmer can write a query or ask the kernel to perform a task.

2. System Libraries:

System libraries are special programs that help in accessing the kernel's features. A kernel has to be triggered to perform a task, and this triggering is done by the

applications. But applications must know how to place a system call because each kernel has a different set of system calls.

Programmers have developed a standard library of procedures to communicate with the kernel. Each operating system supports these standards, and then these are transferred to system calls for that operating system.

The most well-known system library for Linux is Glibc (GNU C library).

3. System Tools:

Linux OS has a set of utility tools, which are usually simple commands. It is a software which GNU project has written and publish under their open source license so that software is freely available to everyone.

With the help of commands, we can access our files, edit and manipulate data in our directories or files, change the location of files, or anything.

4. Development Tools:

With the above three components, our OS is running and working. But to update our system, we have additional tools and libraries. These additional tools and libraries are written by the programmers and are called toolchain. A toolchain is a vital development tool used by the developers to produce a working application.

5. End User Tools:

These end tools make a system unique for a user. End tools are not required for the operating system but are necessary for a user.

Some examples of end tools are graphic design tools, office suites, browsers, multimedia players, etc.

Advantages of Linux Operating System:

1. Open Source:

Linux is an open-source OS that means anyone can see the source code and change it according to his needs. You can freely install Linux on many computers without getting paid license.

If we compare this with windows or mac then they are paid operating systems. You have to get license of windows and mac to use on your machine.

2. No Anti-Virus Software Needed:

In Linux, you do not need anti-virus software to be installed on your PC. Linux has fewer chances to be affected with virus. The reason for strong virus protection is that Linux has large number of open source developers which keeps an eye on virus-related stuff. If any source code needs to be updated then it is done in no time.

3. Text Editors:

Linux has a vast range of text editors available. If you are a programmer then you can pick any of free software packages like visual studio code, Vim, Atom etc. Most of text editors are freely available and you can use it without any issue.

4. Powerful Command Prompt:

Command prompt in Linux is very advanced and if you are developer then you can perform most of your work using the command-line interface. You can install different repositories and packages through the command-line interface.

5. No Reboot Needed:

If you are windows user then you have seen system reboot while you install/uninstall any software or rebooting when the system becomes slow. But in case of Linux, you do not need to reboot your system in such cases.

6. Low System Specifications:

If you have an old computer that has low specification then you can still run Linux. Linux has different distributions that are available for all types of computers e.g. large scale computers, servers, Pc etc.

7. Good At Multitasking:

If you want to do some batch works like printing a large file or downloading large file then you can concurrently perform other tasks like typing or coding any program. Linux is good in doing such multitasking and your system will not slow down.

8. Less Disk Space Needed:

If you have limited disk space then you can still run Linux. You do not need extra disk space for running Linux for a longer time.

9. File Formats:

Linux supports a large number of file formats. So you have to not worry if any file format does not run on Linux. You can install different software packages for specific file format and it will work fine.

Disadvantages of Linux Operating System:

1. Hardware Drivers:

One of the issues that most Linux users face is that some hardware drivers are not available for Linux. Hardware companies prefer to make drivers for windows or mac because they have more users as compared to Linux.

2. Learning Curve:

Getting started with windows is easy for beginners but learning Linux is difficult. You have to learn about the command-line interface and searching for new software is also little bit difficult. If you face any problem in the operating system then finding solution is problematic. There are fewer experts for Linux as compared to windows and mac.

3. Software Alternative:

Take an example of Photoshop which is a popular graphic editing tool. Photoshop is available for windows but is not present in Linux. There are other photo editing tools but Photoshop is a more powerful tool than others. MS office is another example which is not available for Linux users.

4. Games:

Most of the games are made for windows but not Linux. As windows platform is widely used so game developers have more interest in windows.

Difference between Linux and Windows OS:

Windows	Linux
Windows allows users to use the command line, but not as a Linux command line. To open the command line, click on the Run dialog box and type CMD in the run search bar and press Enter key.	Although the Linux command-line offers more features for administration and daily tasks, it does not offer much to end-users.

Windows has improved its reliability in the past few years, but still it is less reliable as compared to Linux.	Linux is more reliable and secure then Windows OS. It mainly focuses on system security, process management, and up-time.
Windows is easier to use as it provides a simple user interface. But its installation process can take more time.	Although Linux has the ability to perform complex tasks easier, its installation process is complicated.
Microsoft has enhanced the security features in Windows over recent years. As it has a huge user base, mostly for new computer users, it can be easily targeted for malicious coders. Furthermore, among all of the operating systems, Microsoft Windows can be part of developing malware and viruses.	Linux is a more secure operating system as compared to Microsoft Windows. Even attackers found difficulty in breaking the security with the help of Linux.
It provides users the online and integrated help systems, as well as a large number of informative books, are available to provide help for people at all skill levels.	A wide number of books are available to offer help about Linux, including online support.
Regular Windows update makes users frustrated by alerting the Windows update for inconvenient times. Additionally, it takes more time to get an update.	Linux provides users full control over updates. They can update it accordingly, and it takes less time to get an update as well as without any reboot the system.
Microsoft Windows with license does not allow to modify the software (don't have access to the source code). It can be installed only on the systems with a Windows license key.	Linux operating system with a license offers users the benefit to re-use the source code on any number of systems. It is also allowed the users to modify the software and sell its modified version.

Difference between LINUX and UNIX OS:

Linux	Unix
Linux is freely distributed, downloaded through magazines, Books, website, etc. There are paid versions also available for Linux.	Different flavors of Unix have different pricing depending upon the type of vendor.
Linux is Open Source, and thousands of programmer collaborate online and contribute to its development.	Unix systems have different versions. These versions are primarily developed by AT&T as well as other commercial vendors.

Everyone. From home users to developers and computer enthusiasts alike.	The UNIX can be used in internet servers, workstations, and PCs.
BASH is the Linux default shell. It offers support for multiple command interpreters.	Originally made to work in Bourne Shell. However, it is now compatible with many others software.
Linux provides two GUIs, viz., KDE and Gnome. Though there are many alternatives such as Mate, LXDE, etc.	Common Desktop Environment and also has Gnome.
Linux has had about 60-100 viruses listed to date which are currently not spreading.	There are between 80 to 120 viruses reported till date in Unix.
Threat detection and solution is very fast because Linux is mainly community driven. So, if any Linux user posts any kind of threat, a team of qualified developers starts working to resolve this threat.	Unix users require longer wait time, to get the proper bug fixing patch.
Initially developed for Intel's x86 hardware processors. It is available for over twenty different types of CPU which also includes an ARM.	It is available on PA-RISC and Itanium machines.
Linux OS can be installed on various types of devices like mobile, tablet computers.	The UNIX operating system is used for internet servers, workstations & PCs.
Kernel update without reboot	Feta ZFS - next generation filesystem DTrace - dynamic Kernel Tracing
Different Versions of Linux are Redhat, Ubuntu, OpenSuse, etc.	Different Versions of Unix are HP-UX, AIS, BSD, etc.
The Filesystems supported by file type like xfs, nfs, cramfsm ext 1 to 4, ufs, devpts, NTFS.	The Filesystems supported by file types are zfs, hfx, GPS, xfs, vxfs.
Linux is portable and is booted from a USB Stick	Unix is not portable
The source is available to the general public	The source code is not available to anyone.

Unit XI: Future Issues – Operating System

Future of Operating System:

Today's operating systems are conceptually upside-down. They developed the hard way, gradually struggling upwards from the machinery (processors, memory, disks and displays) toward the user. In the future, operating systems and information management tools will grow top-down.

Computing power should make life simpler, not weigh us down with fancy features. Computing power should unify our life online, help us pull threads together, not add more virtual shoe boxes for information to get lost in.

But operating systems have been traveling in the exact opposite direction, away from unity and simplicity. Today, most users' documents are distributed over many computers (at home, at work and a laptop).

Inside each computer, documents are scattered as if someone had dumped them out of a low-flying airplane: some in the file hierarchy or on the desktop, mail in the mailer, bookmarks in the browser, images, other multimedia types, calendar and address information in other boxes.

If we own a PDA, Internet-enabled cell phone or other digital gadgets, we have even more boxes to lose things in.

Today's information environment is in this sense, a huge step backward from the world of, say, 1946. In 1946, we could say "Pull the Marry file," and the whole Marry file would be there, letters, memos, reports, photos, jottings, resumes, publications, bills, contracts and receipts, the whole story.

Current operating systems have traditionally been built bottom-up: Start with the machine, then connect it somehow to the user. Their goal is to package the processor, memory, disk and other peripherals (which are unmitigated troubles to manipulate directly), so that we can manage them by remote control. Instead of moving bits around the disk, we drag file icons around the desktop.

The next-generation operating system starts with the user. It ignores the underlying hardware and as a result, such systems are inherently less efficient than today's primitive, machine-centered ones. Instead, it reflects the shape of our life. Its role is to track our life event by event, moment by moment, and thought by thought.

The user interface will keep going in the direction it's been heading. It was basically one-dimensional in the DOS age of the 1980s, we typed a command line, and the operating system typed lines back.

With the Mac of 1984 and Windows 3.0 of 1990, the user interface (UI) became fully two-dimensional. The next-generation UI will be three-dimensional, not literally (for now) but pictorially, in the sense that a printed tablecloth is a 2-D image. The computer screen will no longer be a glass bulletin board with windows and icons stuck to it; it will be a viewport with an "information landscape" in simulated 3-D on the other side.

Why? Because our hardest task in today's information storm is to master the big picture. The mind's picture-processing capacities are amazing, and we need to use them to see as many electronic documents at a glance as we can.

That's why people pack their desktops with icons. Suppose we were looking at a thousand people. If we lined them up side-by-side, we'd have to walk the length of the row or stand way back to see them all.

But if we put them in a column and stood right in front, slightly to one side or above, then we'd see the whole parade in one glance. Foreshortening does that for us.

Next-generation UIs will use foreshortening to show us a deep parade of digital documents instead of today's flat chaos. Simulated 3-D interfaces are a perfect match to the needs of a narrative stream, which just happens to be a parade of documents.

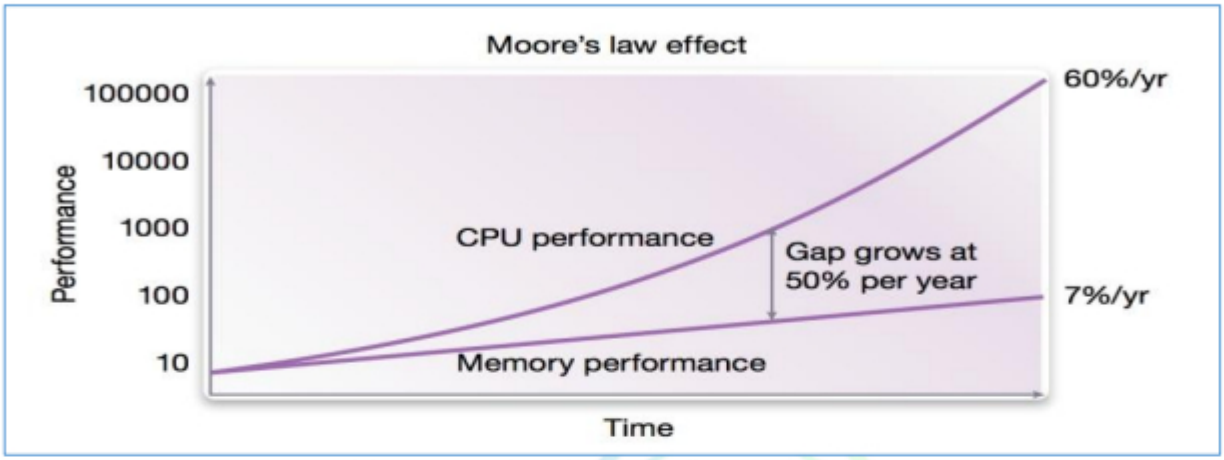
Software revolutions will keep happening in the way they've been happening, without requiring that we throw out our old software and data and start over. To install UNIX in, say, 1976, we had to rip out our old operating system first.

By 1990, we laid down Windows 3.0 like carpeting: DOS stayed put underneath. The coming info-management revolution will be the same sort of low-trauma affair. Our new software will sit on top of Windows and our Windows applications.

The approaching operating system and info-management revolution is no mere speculation. Microsoft's "Longhorn" is supposed to be ready in two years, and its goal is to solve the information management problem. The software exists today and some narrative stream or other is the future of operating systems.

Memory Wall:

The "memory wall" is the growing disparity of speed between CPU and memory outside the CPU chip. An important reason for this disparity is the limited communication bandwidth beyond chip boundaries, which is also referred to as bandwidth wall.



From 1986 to 2000, CPU speed improved at an annual rate of 55% while memory speed only improved at 10%. Given these trends, it was expected that memory latency would become an overwhelming bottleneck in computer performance.

CPU speed improvements slowed significantly partly due to major physical barriers and partly because current CPU designs have already hit the memory wall in some sense. Intel summarized these causes in a 2005 document.

First of all, as chip geometries shrink and clock frequencies rise, the transistor leakage current increases, leading to excess power consumption and heat.

Secondly, the advantages of higher clock speeds are in part negated by memory latency, since memory access times have not been able to keep pace with increasing clock frequencies.

Third, for certain applications, traditional serial architectures are becoming less efficient as processors get faster (due to the so-called Von Neumann bottleneck), further undercutting any gains that frequency increases might otherwise buy.

In addition, partly due to limitations in the means of producing inductance within solid state devices, resistance-capacitance (RC) delays in signal transmission are growing as feature sizes shrink, imposing an additional bottleneck that frequency increases don't address.

The RC delays in signal transmission were also noted in "Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures" which projected a maximum of 12.5% average annual CPU performance improvement between 2000 and 2014.

A different concept is the processor-memory performance gap, which can be addressed by 3D integrated circuits that reduce the distance between the logic and memory aspects that are further apart in a 2D chip.

Memory subsystem design requires a focus on the gap, which is widening over time. The main method of bridging the gap is the use of caches; small amounts of high-speed memory that houses recent operations and instructions nearby the processor, speeding up the execution of those operations or instructions in cases where they are called upon frequently.

Multiple levels of caching have been developed to deal with the widening gap, and the performance of high-speed modern computers relies on evolving caching techniques. There can be up to a 53% difference between the growth in speed of processor speeds and the lagging speed of main memory access.

Solid-state hard drives have continued to increase in speed, from ~400 Mbit/s via SATA3 in 2012 up to ~3 GB/s via NVMe/PCIe in 2018, closing the gap between RAM and hard disk speeds, although RAM continues to be an order of magnitude faster, with single-lane DDR4 3200 capable of 25 GB/s, and modern GDDR even faster.

Fast, cheap, non-volatile solid state drives have replaced some functions formerly performed by RAM, such as holding certain data for immediate availability in server farms - 1 terabyte of SSD storage can be had for $200, while 1 TiB of RAM would cost thousands of dollars.

Some Future of Operating System about Speed:

All current computer device technologies are indeed limited by the speed of electron motion.

This limitation is rather fundamental, because the fastest possible speed for information transmission is of course the speed of light, and the speed of an electron is already a substantial fraction of this.

Where we hope for future improvements is not so much in the speed of computer devices as in the speed of computation.

At first, these may sound like the same thing, until we realize that the number of computer device operations needed to perform a computation is determined by something else namely, an algorithm.

A very efficient algorithm can perform a computation much more quickly than can an inefficient algorithm, even if there is no change in the computer hardware.

So further improvement in algorithms offers a possible route to continuing to make computers faster; better exploitation of parallel operations, pre-computation of parts of a problem, and other similar tricks are all possible ways of increasing computing efficiency.

These ideas may sound like they have nothing to do with 'physical restrictions,' but in fact we have found that by taking into account some of the quantum-mechanical properties of future computer devices, we can devise new kinds of algorithms that are much, much more efficient for certain computations. We still know very little about the ultimate limitations of these 'quantum algorithms.'

Will the Web Browser Swallow the Operating System?

The idea that Web browsers could eventually replace computer operating systems caused Bill Gates to completely freak out, somewhere around 1994, leading to the "browser wars" with Netscape.

Now, a newly announced "Cloud" OS aims to replace netbook operating systems with a simple fast-booting browser-based environment.

The company is Good OS, or gOS, which announced its "Cloud" quick-boot environment for netbooks today at a Netbook expo in Paris. The product appears to be available now for licensing by netbook designers. It follows the release earlier this year of gOS 3.0, Good's widget-enabled distro for netbooks.

Good says gOS Cloud can boot in seconds, into a barebones browser-based environment not unlike the locked-down "kiosk" set-ups at libraries and cafes everywhere. Yet, here, the browser has taken on OS-like features, such as the familiar Mac OS X-like application dock shown in the screenshot below.

Browser swallows OS

In fairness, Cloud is not meant to actually replace operating systems. Rather, it is designed to be the environment most users will use most often. It would be installed in parallel with another OS, Linux or Windows XP typically on low-powered "netbook" or "nettop"

hardware. Users needing to run legacy applications could boot into "that other OS" when they really need to.

Really, it's a lot like having a second steering wheel used only to park a car. Windows and increasingly Linux (in its full-blown GNOME/KDE dress, anyway) both seem to be rapidly outrunning the computing resources available on modern hardware, especially the new breed of low-cost hardware being offered for these modern times.

Yet, quick-boot environments such as Cloud may result in Linux outshipping Windows next year, one industry watcher has suggested. Given that most people do most things in a browser these days, it only makes sense to feature the browser front-and-center. If nothing else, it will save users the "challenge" of having to learn Linux.

And who knows. Maybe Steve Ballmer will freak out and either fix Windows' ridiculously long boot-up time (five minutes plus for Vista on dual-core AMD x2 systems? Puh-leeze) or else maybe create a similar fast-boot environment based on Windows and IE.