S.K.D.A.V. GOVERNMENT POLYTECHNIC, ROURKELA

BRANCH-INFORMATION TECHNOLOGY

SUBJECT-OPERATING SYSTEM

SEMESTER-4TH

PREPARED BY-DIPTI REKHA MISHRA

INTRODUCTION

UNIT-1

1. INTRODUCTION

1.1 Objectives and Explain functions of operating system.

1.2 Evolution of Operating system

1.3 Structure of operating system.

DEFINITION OF OS:-

In the Computer System (comprises of Hardware and software), Hardware can only understand machine code (in the form of 0 and 1) which doesn't make any sense to a naive user.

We need a system which can act as an intermediary and manage all the processes and resources present in the system.

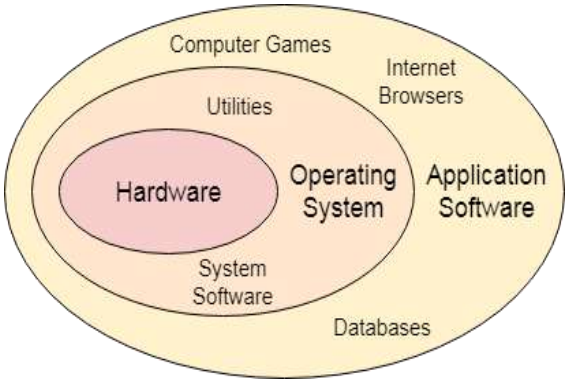
An **Operating System** can be defined as an **interface between user and hardware**. It is responsible for the execution of all the processes, Resource Allocation, CPU management, File Management and many other tasks.

The purpose of an operating system is to provide an environment in which a user can execute programs in convenient and efficient manner.

STRUCTURE OF A COMPUTER SYSTEM

A Computer System consists of:

- Users (people who are using the computer)
- Application Programs (Compilers, Databases, Games, Video player, Browsers, etc.)
- System Programs (Shells, Editors, Compilers, etc.)
- Operating System (A special program which acts as an interface between user and hardware)
- Hardware (CPU, Disks, Memory, etc)



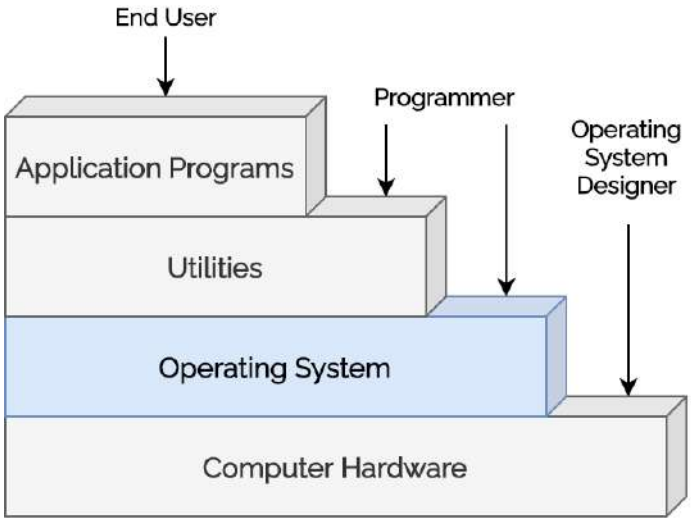
1.1 OBJECTIVES AND EXPLAIN FUNCTIONS OF OPERATING SYSTEM.

OBJECTIVES

The main objectives of an Operating System are,

- **Convenience** – An operating system makes a computer more convenient to use.
- **Efficiency** – An operating system permits the resources to be used efficiently.
- **Ability to evolve** – An operating system should be constructed in such a way that, it should allow the effective development, testing and introduction of new features without any interference with service.

Th hardware and software in a computer system can be viewed in a layered fashion as shown in the figure.



The **end-user** views the computer system as a set of applications. Application users are not concerned with the details of computer hardware. The operating system masks the details of the hardware from the **programmer** and acts as a mediator, making it easier to access the services.

FUNCTIONS OF OPERATING SYSTEM

1. Memory Management

It is the management of the main or primary memory. Whatever program is executed, it has to be present in the main memory. Main memory is a quick storage area that may be accessed directly by the CPU. When the program is completed, the memory region is released and can be used by other programs. Therefore, there can be more than one program present at a time. Hence, it is required to manage the memory.

The operating system:

- Allocates and deallocates the memory.
- Keeps a record of which part of primary memory is used by whom and how much.
- Distributes the memory while multiprocessing.
- In multiprogramming, the operating system selects which processes acquire memory when and how much memory they get.

2. Processor Management/Scheduling

Every software that runs on a computer, whether in the background or in the frontend, is a process. Processor management is an execution unit in which a program operates. The operating system determines the status of the processor and processes, selects a job and its processor, allocates the processor to the process, and de-allocates the processor after the process is completed.

When more than one process runs on the system the OS decides how and when a process will use the CPU. Hence, the name is also **CPU Scheduling**. The OS:

- Allocates and deallocates processor to the processes.
- Keeps record of CPU status.

Certain algorithms used for CPU scheduling are as follows:

- First Come First Serve (FCFS)
- Shortest Job First (SJF)
- Round-Robin Scheduling
- Priority-based scheduling etc.

Purpose of CPU scheduling

The purpose of CPU scheduling is as follows:

- Proper utilization of CPU. Since the proper utilization of the CPU is necessary. Therefore, the OS makes sure that the CPU should be as busy as possible.

- Since every device should get a chance to use the processor. Hence, the OS makes sure that the devices get fair processor time.
- Increasing the efficiency of the system.

3. Device Management

An operating system regulates device connection using drivers. The processes may require devices for their use. This management is done by the OS. The OS:

- Allocates and deallocates devices to different processes.
- Keeps records of the devices.
- Decides which process can use which device for how much time.

4. File Management

The operating system manages resource allocation and de-allocation. It specifies which process receives the file and for how long. It also keeps track of information, location, uses, status, and so on. These groupings of resources are referred to as file systems. The files on a system are stored in different directories. The OS:

- Keeps records of the status and locations of files.
- Allocates and deallocates resources.
- Decides who gets the resources.

5. Storage Management

Storage management is a procedure that allows users to maximize the utilization of storage devices while also protecting data integrity on whatever media on which it lives. Network virtualization, replication, mirroring, security, compression, deduplication, traffic analysis, process automation, storage provisioning, and memory management are some of the features that may be included. The operating system is in charge of storing and accessing files. The creation of files, the creation of directories, the reading and writing of data from files and directories, as well as the copying of the contents of files and directories from one location to another are all included in storage management.

The OS uses storage management for:

- Improving the performance of the data storage resources.
- It optimizes the use of various storage devices.
- Assists businesses in storing more data on existing hardware, speeding up the data retrieval process, preventing data loss, meeting data retention regulations, and lowering IT costs

The Operating System provides certain services to the users which can be listed **in the following manner:**

- **Program Execution**:

The Operating System is responsible for execution of all types of programs whether it be user programs or system programs. The Operating System utilises various resources available for the efficient running of all types of functionalities.

- **Handling Input/Output Operations**:

The Operating System is responsible for handling all sort of inputs, i.e, from keyboard, mouse, desktop, etc. The Operating System does all interfacing in the most appropriate manner regrading all kind of Inputs and Outputs.
For example, there is difference in nature of all types of peripheral devices such as mouse or keyboard, then Operating System is responsible for handling data between them.

- **Manipulation of File System**:

The Operating System is responsible for making of decisions regarding the storage of all types of data or files, i.e, floppy disk/hard disk/pen drive, etc. The Operating System decides as how should the data should be manipulated and stored.

- **Error Detection and Handling**:

The Operating System is responsible for detection of any types of error or bugs that can occur while any task. The well secured OS sometimes also acts as countermeasure for preventing any sort of breach to the Computer System from any external source and probably handling them.

- **Resource Allocation:**

The Operating System ensures the proper use of all the resources available by deciding which resource to be used by whom for how much time. All the decisions are taken by the Operating System.

- **Accounting:** The Operating System tracks an account of all the functionalities taking place in the computer system at a time. All the details such as the types of errors occurred are recorded by the Operating System.

- **Information and Resource Protection:** The Operating System is responsible for using all the information and resources available on the machine in the most protected way. The Operating System must foil an attempt from any external resource to hamper any sort of data or information.
- All these services are ensured by the Operating System for the convenience of the users to make the programming task easier. All different kinds of Operating System more or less provide the same services

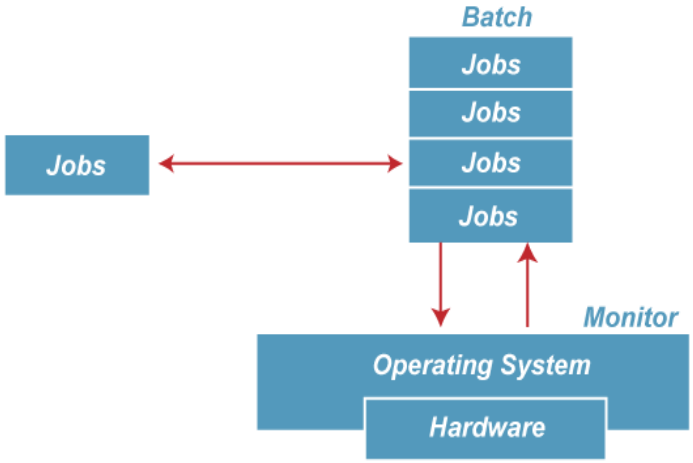
1.2 EVOLUTION OF OPERATING SYSTEM

BATCH OPERATING SYSTEM

In the 1970s, Batch processing was very popular. In this technique, similar types of jobs were batched together and executed in time. People were used to having a single computer which was called a mainframe.

In Batch operating system, access is given to more than one person; they submit their respective jobs to the system for the execution.

The system put all of the jobs in a queue on the basis of first come first serve and then executes the jobs one by one. The users collect their respective output when all the jobs get executed.



The purpose of this operating system was mainly to transfer control from one job to another as soon as the job was completed. It contained a small set of programs called the resident monitor that always resided in one part of the main memory. The remaining part is used for servicing jobs.

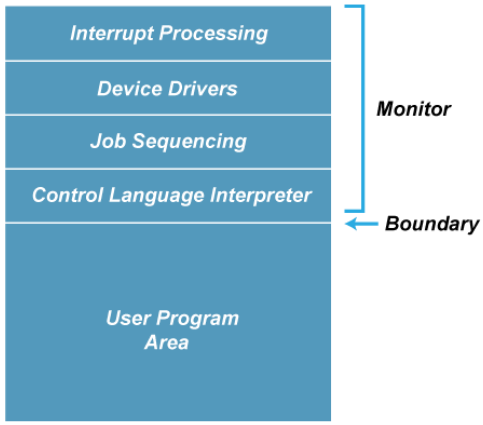


Figure: Memory Layout of the resident monitor

Advantages of Batch OS

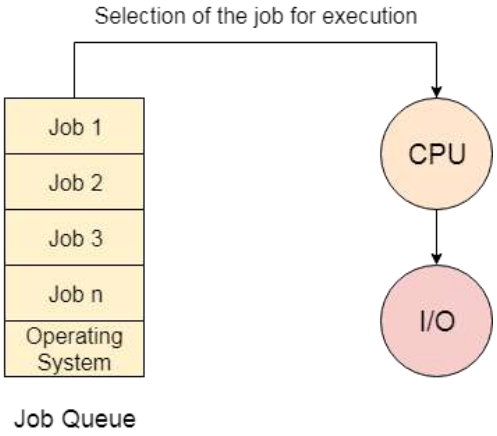
- The use of a resident monitor improves computer efficiency as it eliminates CPU time between two jobs.

Disadvantages of Batch OS

1. Starvation

Batch processing suffers from starvation.

For Example:



There are five jobs J1, J2, J3, J4, and J5, present in the batch. If the execution time of J1 is very high, then the other four jobs will never be executed, or they will have to wait for a very long time. Hence the other processes get starved.

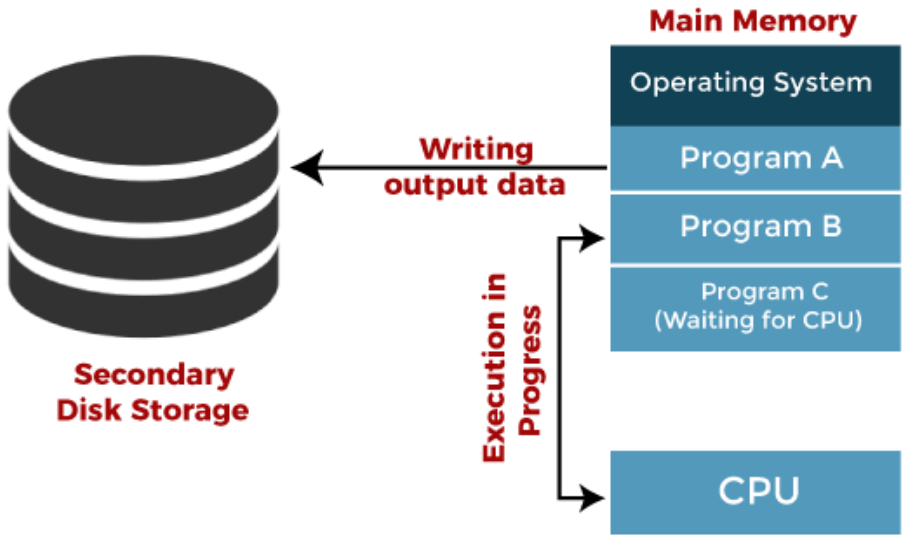
2. Not Interactive

Batch Processing is not suitable for jobs that are dependent on the user's input. If a job requires the input of two numbers from the console, then it will never get it in the batch processing scenario since the user is not present at the time of execution.

MULTIPROGRAMMING OPERATING SYSTEM

Multiprogramming is an extension to batch processing where the CPU is always kept busy. Each process needs two types of system time: CPU time and IO time.

In a multiprogramming environment, when a process does its I/O, The CPU can start the execution of other processes. Therefore, multiprogramming improves the efficiency of the system.



Jobs in multiprogramming system

Advantages of Multiprogramming OS

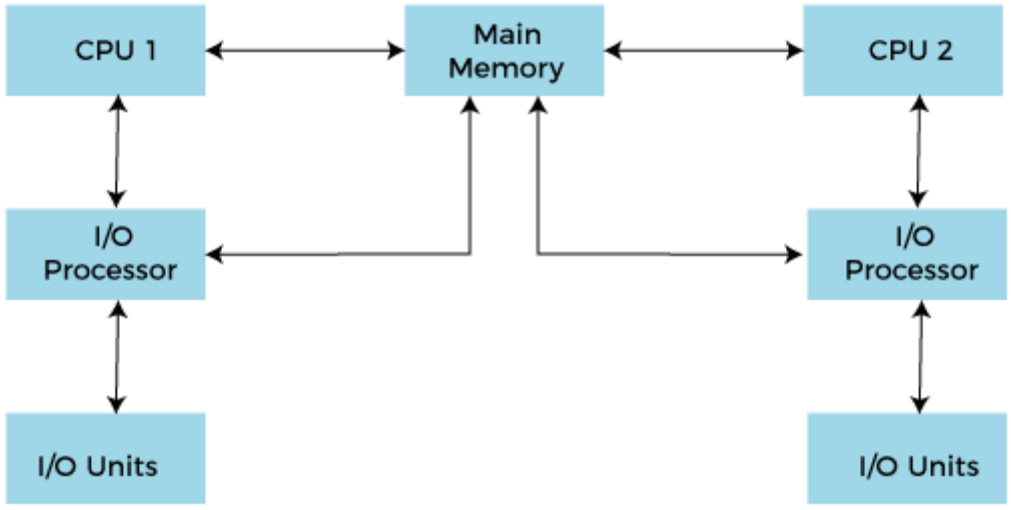
- Throughout the system, it increased as the CPU always had one program to execute.
- Response time can also be reduced.

Disadvantages of Multiprogramming OS

- Multiprogramming systems provide an environment in which various systems resources are used efficiently, but they do not provide any user interaction with the computer system.

MULTIPROCESSING OPERATING SYSTEM

In Multiprocessing, Parallel computing is achieved. There are more than one processors present in the system which can execute more than one process at the same time. This will increase the throughput of the system.



Working of Multiprocessor System

In Multiprocessing, Parallel computing is achieved. More than one processor present in the system can execute more than one process simultaneously, which will increase the throughput of the system.

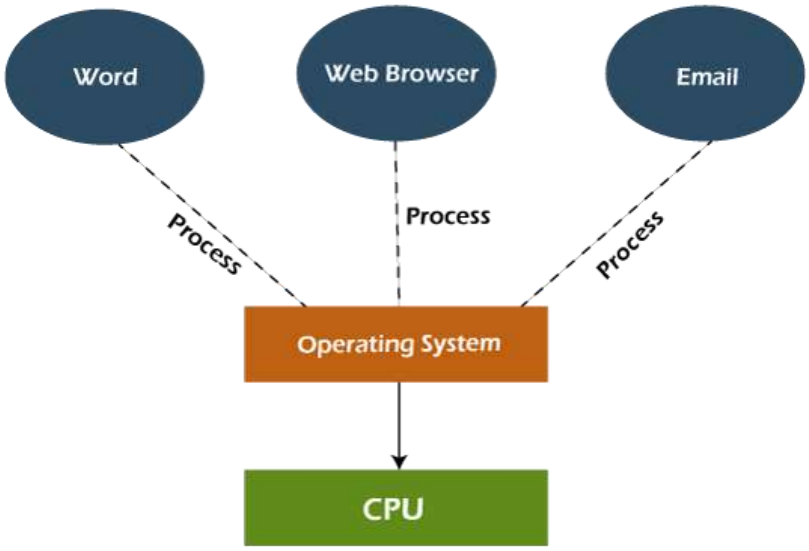
Advantages of Multiprocessing operating system:

- **Increased reliability:** Due to the multiprocessing system, processing tasks can be distributed among several processors. This increases reliability as if one processor fails, the task can be given to another processor for completion.
- **Increased throughout:** As several processors increase, more work can be done in less.

Disadvantages of Multiprocessing operating System

- Multiprocessing operating system is more complex and sophisticated as it takes care of multiple CPUs simultaneously.

Multitasking Operating System



The multitasking operating system is a logical extension of a multiprogramming system that enables **multiple** programs simultaneously. It allows a user to perform more than one computer task at the same time.

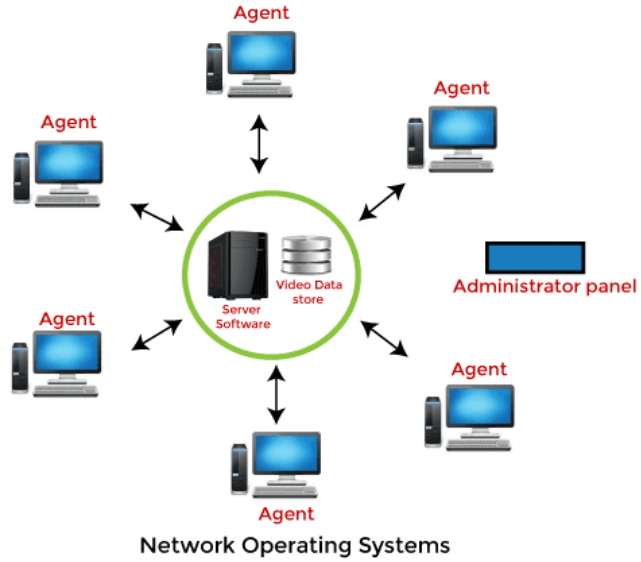
Advantages of Multitasking operating system

- This operating system is more suited to supporting multiple users simultaneously.
- The multitasking operating systems have well-defined memory management.

Disadvantages of Multitasking operating system

- The multiple processors are busier at the same time to complete any task in a multitasking environment, so the CPU generates more heat.

NETWORK OPERATING SYSTEM



Network Operating Systems

An Operating system, which includes software and associated protocols to communicate with other computers via a network conveniently and cost-effectively, is called Network Operating System.

Advantages of Network Operating System

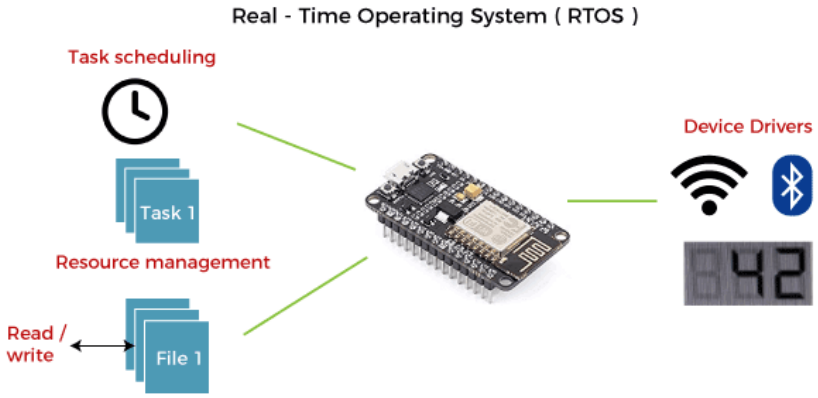
- In this type of operating system, network traffic reduces due to the division between clients and the server.
- This type of system is less expensive to set up and maintain.

Disadvantages of Network Operating System

- In this type of operating system, the failure of any node in a system affects the whole system.
- Security and performance are important issues. So trained network administrators are required for network administration.

Real Time Operating System

In Real-Time Systems, each job carries a certain deadline within which the job is supposed to be completed, otherwise, the huge loss will be there, or even if the result is produced, it will be completely useless.



The Application of a Real-Time system exists in the case of military applications, if you want to drop a missile, then the missile is supposed to be dropped with a certain precision.

Advantages of Real-time operating system:

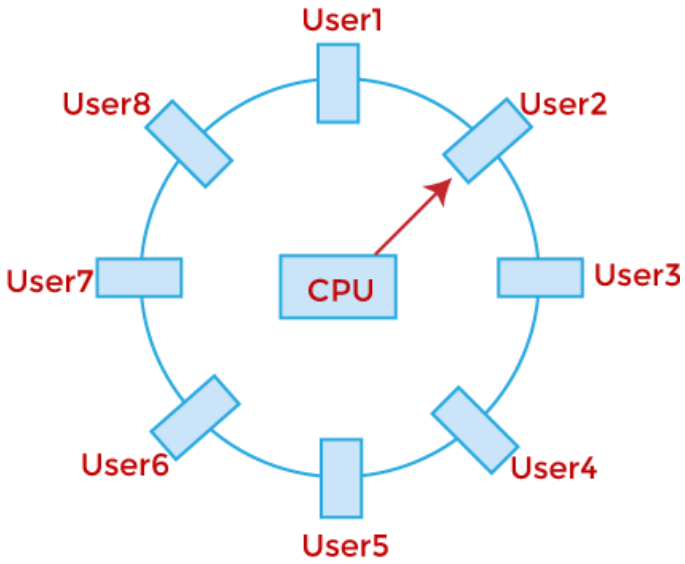
- Easy to layout, develop and execute real-time applications under the real-time operating system.
- In a Real-time operating system, the maximum utilization of devices and systems.

Disadvantages of Real-time operating system:

- Real-time operating systems are very costly to develop.
- Real-time operating systems are very complex and can consume critical CPU cycles.

TIME-SHARING OPERATING SYSTEM

In the Time Sharing operating system, computer resources are allocated in a time-dependent fashion to several programs simultaneously. Thus it helps to provide a large number of user's direct access to the main computer. It is a logical extension of multiprogramming. In time-sharing, the CPU is switched among multiple programs given by different users on a scheduled basis.



Timesharing in case of 8 users

A time-sharing operating system allows many users to be served simultaneously, so sophisticated CPU scheduling schemes and Input/output management are required.

Time-sharing operating systems are very difficult and expensive to build.

Advantages of Time Sharing Operating System

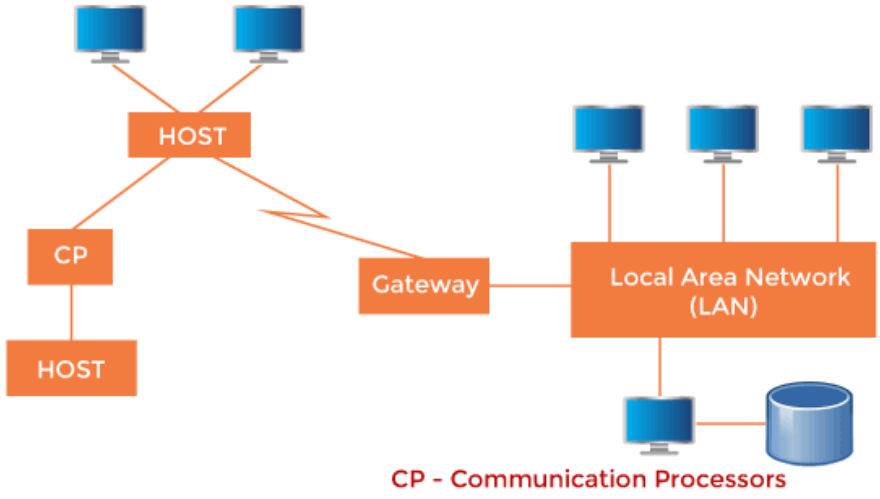
- The time-sharing operating system provides effective utilization and sharing of resources.
- This system reduces CPU idle and response time.

Disadvantages of Time Sharing Operating System

- Data transmission rates are very high in comparison to other methods.
- Security and integrity of user programs loaded in memory and data need to be maintained as many users access the system at the same time.

DISTRIBUTED OPERATING SYSTEM

The Distributed Operating system is not installed on a single machine, it is divided into parts, and these parts are loaded on different machines. A part of the distributed Operating system is installed on each machine to make their communication possible. Distributed Operating systems are much more complex, large, and sophisticated than Network operating systems because they also have to take care of varying networking protocols.



A Typical View of a Distributed System

Advantages of Distributed Operating System

- The distributed operating system provides sharing of resources.
- This type of system is fault-tolerant.

Disadvantages of Distributed Operating System

- Protocol overhead can dominate computation cost.

1.3 STRUCTURE OF OPERATING SYSTEM.

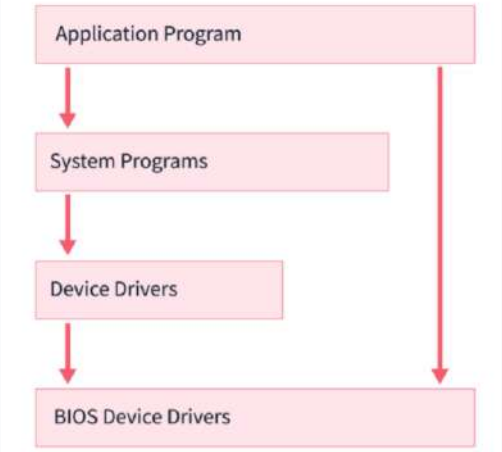
- **What is the Operating System Structure?**
- An Operating System has a complex structure, So to implement such a complex thing, we need a good predefined structure that will help us to implement Operating systems for our specific tasks.
- Just like we break down the complex problem into subproblems that are easier to solve, an easier way to design an OS is by making it in parts.
- And each part is a component of the Operating System.
- We can describe operating system structure as the method of interconnecting and integrating various components of the operating system into the kernel. There are different types of structures on which operating system is implemented as discussed below:

STRUCTURE OF OPERATING SYSTEM

It is the simplest Operating System Structure and is not well defined, It can only be used for small and limited systems. In this structure, the interfaces and levels of functionality are well separated, hence programs can access I/O routines which can cause unauthorized access to I/O routines.

This structure is implemented in MS-DOS operating system:

- The MS-DOS operating system is made up of various layers, each with its own set of functions.
- These layers are:
    - Application Program
    - System Program
    - MS-DOS device drivers
    - ROM BIOS device drivers
- Layering has an advantage in the MS-DOS operating system since all the levels can be defined separately and can interact with each other when needed.
- It is easier to design, maintain, and update the system if it is made in layers. So that's why limited systems with less complexity can be constructed easily using Simple Structure.
- If one user program fails, the entire operating system gets crashed.
- The abstraction level in MS-DOS systems is low, so programs and I/O routines are visible to the end-user, so the user can have unauthorized access.



Advantages of Simple Structure

- It is easy to develop because of the limited number of interfaces and layers.
- Offers good performance due to lesser layers between hardware and applications.

Disadvantages of Simple Structure

- If one user program fails, the entire operating system crashes.
- Abstraction or data hiding is not present as layers are connected and communicate with each other.
- Layers can access the processes going in the Operating System which can lead to data modification and can cause Operating System to crash.

MONOLITHIC STRUCTURE

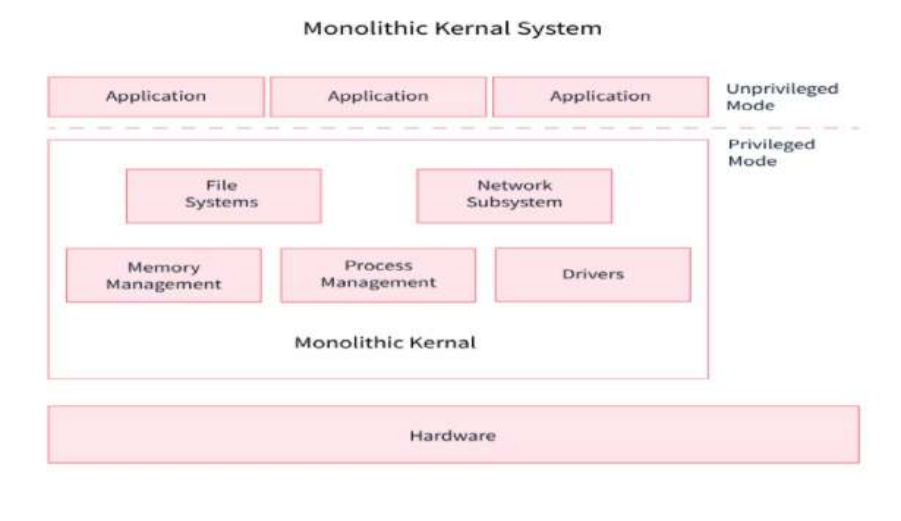
The monolithic operating system in which kernel acts as a manager by managing all things like file management, memory management, device management, and operational processes of the Operating system.

The kernel is the heart of a computer operating system (OS). Kernel delivers basic services to all other elements of the system. It serves as the primary interface between the operating system and the hardware.

In monolithic systems, kernels can directly access all the resources of the operating system like physical hardware exp Keyboard, Mouse, etc.

The monolithic kernel is another name for the monolithic operating system. Batch processing and time-sharing maximize the usability of a processor by multiprogramming. The monolithic kernel functions as a virtual machine by working on the top of the Operating system and controlling all hardware components. This is an outdated operating system that was used in banks to accomplish minor activities such as batch processing and time-sharing which enables many people, at various terminals to access the Operating system.

A Diagram of the Monolithic structure is shown below:



Advantages of Monolithic structure:

- It is simple to design and implement because all operations are managed by kernel only and layering is not needed.
- As services such as memory management, file management, process scheduling, etc are implemented in the same address space, the execution of the monolithic kernel is relatively fast as compared to normal systems. Using the same address saves time for address allocation for new processes and makes it faster.

Disadvantages of Monolithic structure:

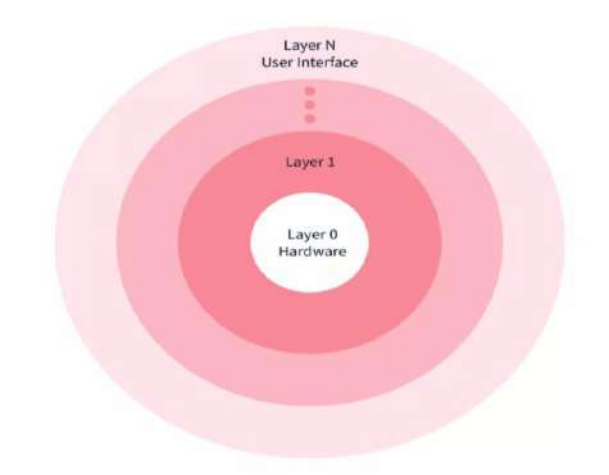
- If any service in the monolithic kernel fails, the entire system fails because, in address space, the services are connected to each other and affect each other.
- It is not flexible and to introduce a new service

LAYERED APPROACH

In this type of structure, OS is divided into layers or levels. The hardware is on the bottom layer (layer 0), while the user interface is on the top layer (layer N). These layers are arranged in a hierarchical way in which the top-level layers use the functionalities of their lower-level levels.

In this approach functionalities of each layer are isolated and abstraction is also available. In layered structure, debugging is easier as it is a hierarchical model, so all lower-level layered are debugged and then the upper layer is checked. So all the lower layers are already checked and the current layer is to be checked only.

Below is the Image illustrating the Layered structure in OS:



Advantages of Layered Structure

- Each layer has its functionalities so work tasks are isolated and abstraction is present up to some level.
- Debugging is easier as lower layers are debugged and then upper layers are checked.

Disadvantages of Layered Structure

- In Layered Structure, layering causes degradation in performance.
- It takes careful planning to construct the layers since higher layers only utilize the functions of lower layers.

MICRO-KERNEL

Micro-Kernel structure designs the Operating System by removing all non-essential components of the kernel. These non-essential components of kernels are implemented as systems and user programs. Hence these implemented systems are called as Micro-Kernels.

Each Micro-Kernel is made independently and is isolated from other Micro-Kernels. So this makes the system more secure and reliable. If any Micro-Kernel fails then the remaining operating System remains untouched and works fine.

Advantages of Micro-kernel structure:

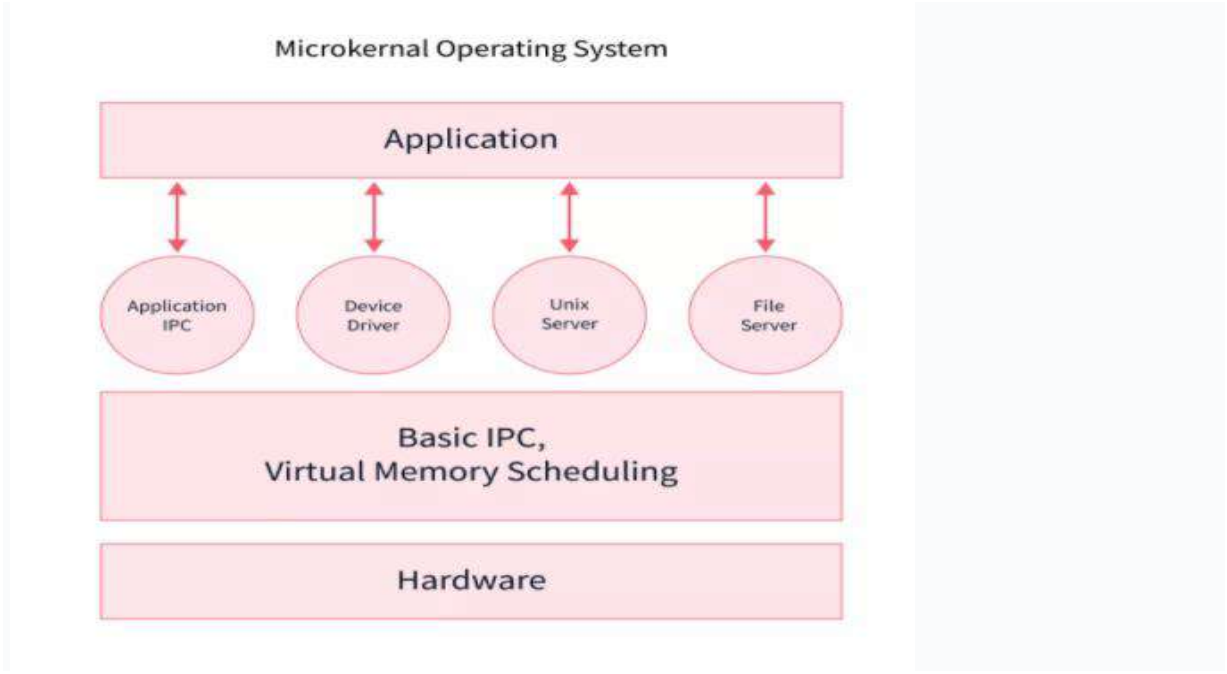
- It allows the operating system to be portable between platforms.
- As each Micro-Kernel is isolated, it is safe and trustworthy.
- Because Micro-Kernels are smaller, they can be successfully tested.
- If any component or Micro-Kernel fails, the remaining operating system is unaffected and continues to function normally.

Disadvantages of Micro-kernel structure:

- Increased inter-module communication reduces system performance.
- System is complex to be constructed.

Conclusion

- The operating system enables the end-user to interact with the computer hardware. System software is installed and utilized on top of the operating system.
- We can define operating system structure as to how different components of the operating system are interconnected.
- There are many structures of the operating system:
    - Simple Structure
    - Monolithic Approach
    - Layered Approach
    - Microkernels
- All these approaches or structures evolved from time to time making the OS more and more improved than before.
-
-


Microkernal Operating System
Application
Application IPC
Device Driver
Unix Server
File Server
Basic IPC,
Virtual Memory Scheduling
Hardware

PROCESS MANAGEMENT

UNIT II

2. PROCESS MANAGEMENT

2.1 Process concept, process control, interacting processes, inter process messages.

2.2 Implementation issues of Processes.

2.3 Process scheduling, job scheduling.

2.4 Process synchronization, semaphore.

2.5 Principle of concurrency, types of scheduling

2. PROCESS MANAGEMENT

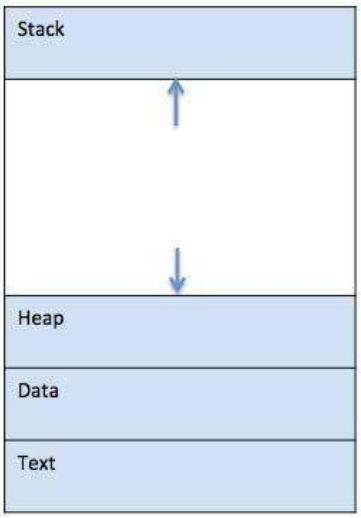
Definition of process:

A process is basically a program in execution or instance of the program execution. The execution of a process must progress in a sequential fashion.

- 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 :

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

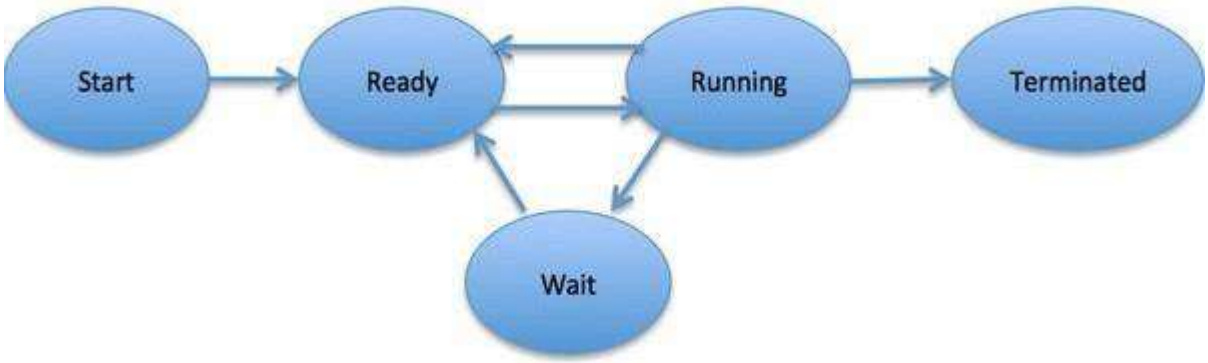


Process States:

When a process executes, it passes through different states. These stages may differ in different operating systems.

In general, a process can have one of the following five states at a time.

S.N.	State & Description
1	**Start:** This is the initial state when a process is first started/created.
2	**Ready:** The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process.
3	**Running:** Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.
4	**Waiting:** Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
5	**Terminated or Exit:** Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.



Process State Diagram

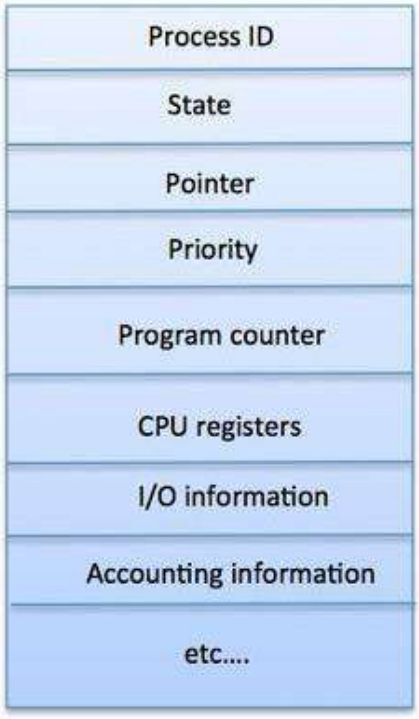
Process Control Block (PCB):

- A Process Control Block is a data structure maintained by the Operating Systemfor every process.

- The PCB is identified by an integer process ID (PID).
- A PCB keeps all the information needed to keep track of a process as listed belowin 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: P**rogram 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 a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB –



Process Control Block (PCB) Diagram

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

Process SchedulingDefinition

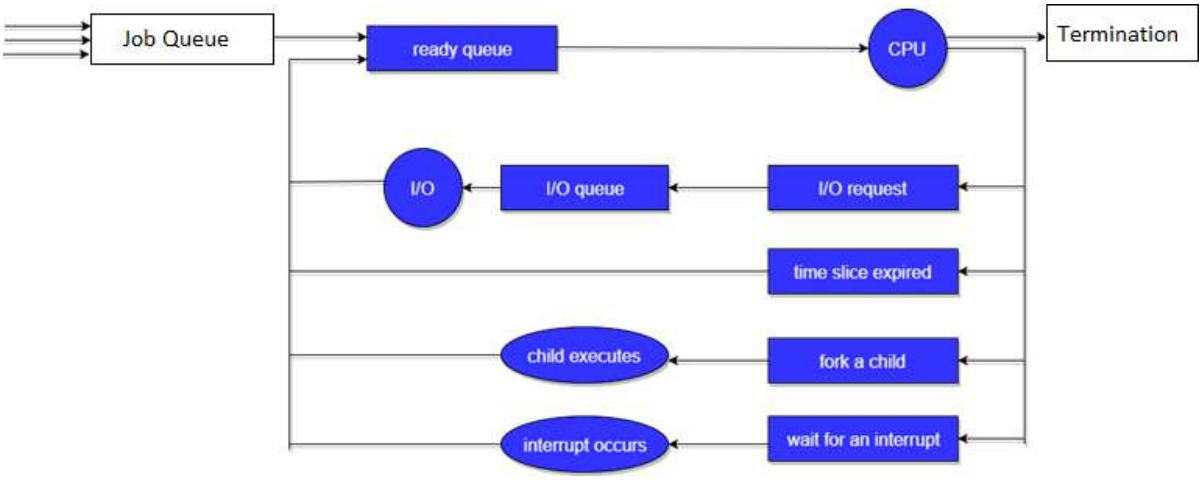
- The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
- Process scheduling is an essential part of a Multiprogramming operating systems.
- Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

What are Scheduling Queues?

- All processes, upon entering into the system, are stored in the **Job Queue**.
- Processes in the Ready state are placed in the **Ready Queue**.
- Processes waiting for a device to become available are placed in **Device Queues**. There are unique device queues available for each I/O device.

A new process is initially put in the **Ready queue**. It waits in the ready queue until it is selected for execution (or dispatched). Once the process is assigned to the CPU and is executing, one of the following several events can occur:

- The process could issue an I/O request, and then be placed in the **I/O queue**.
- The process could create a new sub-process and wait for its termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.



In the first two cases, the process eventually switches from the waiting state to the ready state, and is then put back in the ready queue. A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.

Schedulers:

- Schedulers are special system software which handle process scheduling in various ways.
- Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

- It is also called a job scheduler.
- A long-term scheduler determines which programs are admitted to the system for processing.
- It selects processes from the queue and loads them into memory for execution.
- Process loads into the memory for CPU scheduling.
- The primary objective of the job scheduler is to provide a balanced mix of jobs, suchas I/O bound and processor bound.

- It also controls the degree of multiprogramming.
- If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.
- On some systems, the long-term scheduler may not be available or minimal.
- Time-sharing operating systems have no long term scheduler.
- When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler:

- It is also called as CPU scheduler.
- Its main objective is to increase system performance in accordance with the chosenset of criteria.
- It is the change of ready state to running state of the process.
- CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.
- Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

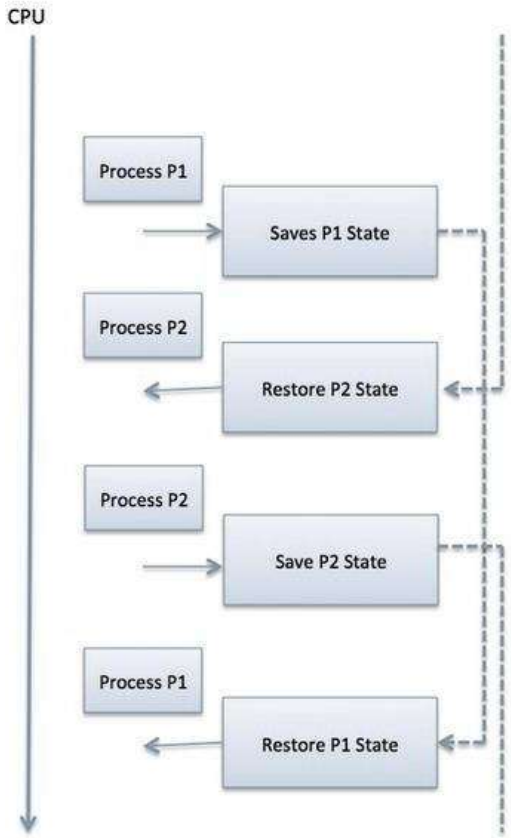
- Medium-term scheduling is a part of swapping.
- It removes the processes from the memory.
- It reduces the degree of multiprogramming.
- The medium-term scheduler is in-charge of handling the swapped out-processes.
- A running process may become suspended if it makes an I/O request.
- A suspended process cannot make any progress towards completion.
- In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage.
- This process is called swapping, and the process is said to be swapped out or rolled out.
- Swapping may be necessary to improve the process mix.
-

Comparison among Scheduler

S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

CONTEXT SWITCH:

- A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time.
- Using this technique, a context switcher enables multiple processes to share a single CPU.
- Context switching is an essential part of a multitasking operating system features.
- When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block.
- After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc.
- At that point, the second process can start executing.



- Context switches are computationally intensive since register and memory state mustbe saved and restored.
- To avoid the amount of context switching time, some hardware systems employ twoor more sets of processor registers.
- When the process is switched, the following information is stored for later use.
  - Program Counter
  - Scheduling information
  - Base and limit register value
  - Currently used register
  - Changed State
  - I/O State information
  - Accounting information

Inter-process Communication

Inter process communication is the mechanism provided by the operating system that allows processes to communicate with each other. This communication could involve a process letting another process know that some event has occurred or the transferring of data from one process to another.

A process can be of two types:

1. **Independent process.**
2. **Co-operating process.**

An independent process is not affected by the execution of other processes

cooperating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently, in reality, there are many situations when co-operative nature can be utilised for increasing computational speed, convenience and modularity.

ADVANTAGES OF CO-OPERATING PROCESS:-

Following are the advantages.
⇒ Information sharing
⇒ Computation speedup
⇒ Modularity
⇒ Convenience

INFORMATION SHARING:-⇒ In multi programming system several users may need the same piece of information.
⇒ Co-operating environment allow concurrent access to these type of resources.

COMPUTATION SPEEDUP:-⇒ To execute a task faster, it must break into subtasks. Each sub tasks executes parallel with each other, such a speed can be achieved only if the computer has multiprocessing elements.

MODULARITY:-⇒ To construct a system in modular approach divide the system functions into no. of separate processes.

CONVENIENCE:-⇒ A user can perform many task at one time.

Ex. A user may be editing, printing and compiling in parallel

INTER PROCESS COMMUNICATION (IPC)

Inter process communication (IPC) is a mechanism which allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of cooperation between them.

Processes can communicate with each other through both:

1. **Shared Memory**
2. **Message passing**

1. Shared Memory

In shared memory system a region of memory is shared by the co-operating processes.

⇒ Then the processes can share the information by reading and writing data to the shared region.

⇒ Shared m.s. allows maximum speed and convenience of communication.

⇒ In shared memory system the shared region has been created through system calls.

⇒ A shared memory region resides in the address space of the process creating the share region.

⇒ In shared memory region all access are treated as routine memory access.

⇒ Other processes must attached there address space with the shared memory segment for communication

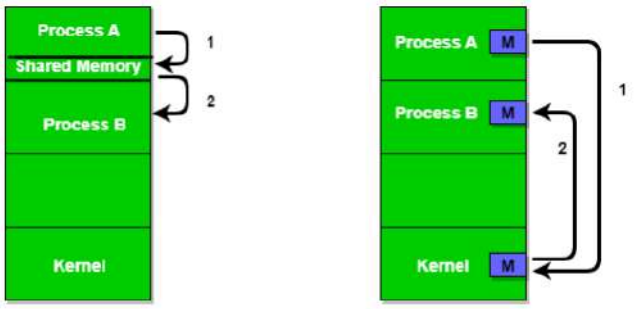


Figure 1 - Shared Memory and Message Passing

THIS PROBLEM ARE OF TWO CATEGORIES.

i) Bounded buffer problem
ii) Unbounded buffer problem .

i) Bounded buffer problem:-

⇒ In this problem the buffer size is fixed.

⇒ If the producer is faster than the consumer, then after some time the buffer is full and producer has to wait till the buffer becomes empty.

⇒ When the consuming rate is high than producing, then after some time buffer becomes empty and the consumer has to wait.

ii) Unbounded buffer problem:-

⇒ Here the buffer size can be vary.

⇒ In this case if the producer is faster than consumer, then the buffer is overloaded with data and if the consumer is faster than the producer then the consumer has to wait till the buffer becomes empty.

2. Message passing

⇒ Message passing system is a mechanism to allow processes to communicate and synchronies their activities without sharing the address space.

⇒ Message passing mechanism/ concept is generally used in distributed environment.

⇒ 2 primitives or operations are there in message passing system.

i) Send () :- To send a message.
ii) Receive () :- To receive message.

⇒ Messages used for communication are of 2 types i.e.

⇒ Fixed sized msg

⇒ Variable sized msg

⇒ In message passing system communication occurs in 3 ways.

i) Direct / indirect comm..
ii) Synchronous/ asynchronous comm..
iii) Explicit/ implicit (automatic buffering)

i) **Direct Comm:-**

⇒ In direct comm.. the processes that want to communicate with each other must know each others name.

⇒ In direct comm. There are 2 types of addressing or naming .

a) Symmetric addressing
b) Asymmetric

a) Symmetric:-

⇒ In symmetric addressing send () operation will occur as send (receiving/ msg process name):

⇒ The receive operation occur as receive (sender process, msg name);

send(P, message) -Send a message to process P.
receive (Q, message)-Receive a message from process Q.

Characteristics of communication link in direct communication:-

⇒ It is established automatically between every pair of process that wants to communicate.
⇒ The link must be established with exactly two processes.
⇒ Between each pair of processes, there should be exactly one link.
⇒ The link may be unidirectional or bidirectional.

b) Asymmetric:-

⇒ In this addressing only sender names the receiver.
⇒ The receiver will identify the sender process by his process id

send(P, message) -Send a message to process P.
receive (id, message) -Receive a message from any process; the variable id is set to the name of the process with which communication has taken place.

2) Indirect communication:-

⇒ In indirect communication the messages are send and received by mailbox or port.
⇒ Mailbox is an abstract object in which the messages can be placed and removed.
⇒ Two processes can communicate only if they share the same mailbox.

⇒ 2 operation will occur as
Send (mailbox name,msg)
Receive (mailbox name, msg)

send (A, message) -Send a message to mailbox A.
receive (A, message)-Receive a message from mailbox A.

⇒ Indirect communication has the following properties

i) A link is established between every pair of process if they share a common mail box.

ii) A link may be unidirectional or bidirectional

iii) A link may be associated basically with two process (more than two process also possible)

Synchronization:-

Communication between processes takes place through calls to send() and receive () primitives. There are different design options for implementing each primitive. Message passing may be either **blocking or nonblocking**also known as **synchronous and asynchronous.**

- **Blocking send**. -The sending process is blocked until the message is received by the receiving process or by the mailbox.
- **Nonblocking send**- The sending process sends the message and resumes operation.

- **Blocking receive.**-The receiver blocks until a message is available.
- **Nonblocking receive**.-The receiver retrieves either a valid message or a null.

Buffering:-

⇒ A link capacity determines a no. of msgs can reside in temporary queue.

⇒ A link can be assume as a queue like structure.

⇒ This link can be implemented in 3 ways

.i> Zero capacity (no buffing)

ii> Bounded capacity

iii> Unbounded capacity.

⇒ In zero capacity the queue maximum length is zero, the link can't hold nay msgs waiting in it.

⇒ In bounded capacity, the queue has finite length 'n'. hence at most 'n' msgs can reside the link.

⇒ In unbounded capacity any no. of msgs can be reside.

IMPLEMENTATION OF PROCESSES :

- Process Model is implemented by Process Table and Process Control Block which keep track all information of process.
- At the time of creation of a new process, operating system allocates a memory for it loads a process code in the allocated memory and setup data space for it .
- The state of process is stored as ' new ' in its PCB and when this process move to ready state its state is also changes in PCB.
- When a running process needs to wait for an input output devices, its state is changed

to 'blocked'. The various queues used for this which is implemented as linked list.

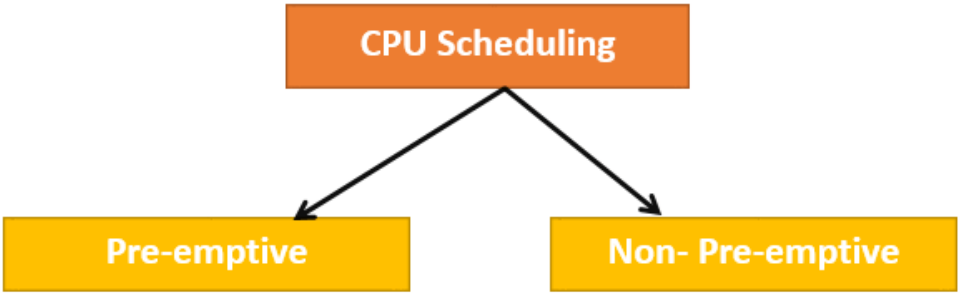
CPU SCHEDULING IN OPERATING SYSTEM

CPU scheduling is a process that allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast, and fair.

Whenever the CPU becomes idle, the operating system must select one of the processes in the **ready queue** to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute and allocates the CPU to one of them.

Types of CPU Scheduling

Here are two kinds of Scheduling methods:



Preemptive Scheduling

In Preemptive Scheduling, the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running. The lower priority task holds for some time and resumes when the higher priority task finishes its execution.

Non-Preemptive Scheduling

In this type of scheduling method, the CPU has been allocated to a specific process. The process that keeps the CPU busy will release the CPU either by switching context or terminating. It is the only method that can be used for various hardware platforms. That's because it doesn't need special hardware (for example, a timer) like preemptive scheduling.

When scheduling is Preemptive or Non-Preemptive?

To determine if scheduling is preemptive or non-preemptive, consider these four parameters:

1. A process switches from the running to the waiting state.
2. Specific process switches from the running state to the ready state.
3. Specific process switches from the waiting state to the ready state.
4. Process finished its execution and terminated.

Only conditions 1 and 4 apply, the scheduling is called non- preemptive.

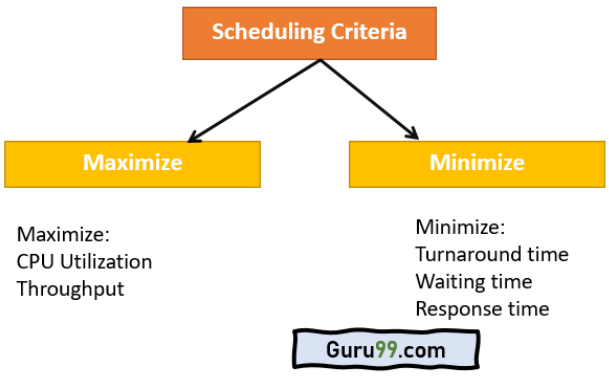
All other scheduling are preemptiv

Important CPU scheduling Terminologies

- **Burst Time/Execution Time:** It is a time required by the process to complete execution. It is also called running time.
- **Arrival Time:** when a process enters in a ready state
- **Finish Time:** when process complete and exit from a system
- **Multiprogramming:** A number of programs which can be present in memory at the same time.
- **Jobs:** It is a type of program without any kind of user interaction.
- **User:** It is a kind of program having user interaction.
- **Process:** It is the reference that is used for both job and user.
- **CPU/IO burst cycle:** Characterizes process execution, which alternates between CPU and I/O activity. CPU times are usually shorter than the time of I/O.

CPU Scheduling Criteria

A CPU scheduling algorithm tries to maximize and minimize the following:



Maximize:

CPU utilization: CPU utilization is the main task in which the operating system needs to make sure that CPU remains as busy as possible. It can range from 0 to 100 percent. However, for the RTOS, it can be range from 40 percent for low-level and 90 percent for the high-level system.

Throughput: The number of processes that finish their execution per unit time is known Throughput. So, when the CPU is busy executing the process, at that time, work is being done, and the work completed per unit time is called Throughput.

Minimize:

Waiting time: Waiting time is an amount that specific process needs to wait in the ready queue.

Response time: It is an amount to time in which the request was submitted until the first response is produced.

Turnaround Time: Turnaround time is an amount of time to execute a specific process. It is the calculation of the total time spent waiting to get into the memory, waiting in the queue and, executing on the CPU. The period between the time of process submission to the completion time is the turnaround time.

Interval Timer

Timer interruption is a method that is closely related to preemption. When a certain process gets the CPU allocation, a timer may be set to a specified interval. Both timer interruption and preemption force a process to return the CPU before its CPU burst is complete.

Most of the multi-programmed operating system uses some form of a timer to prevent a process from tying up the system forever.

What is Dispatcher?

It is a module that provides control of the CPU to the process. The Dispatcher should be fast so that it can run on every context switch. Dispatch latency is the amount of time needed by the CPU scheduler to stop one process and start another.

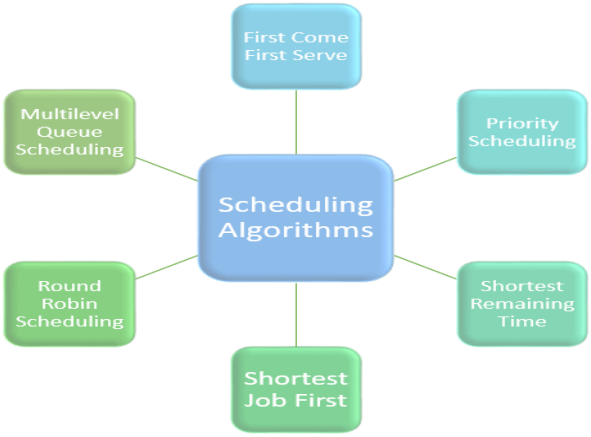
Functions performed by Dispatcher:

- Context Switching
- Switching to user mode
- Moving to the correct location in the newly loaded program.

Types of CPU scheduling Algorithm

There are mainly six types of process scheduling algorithms

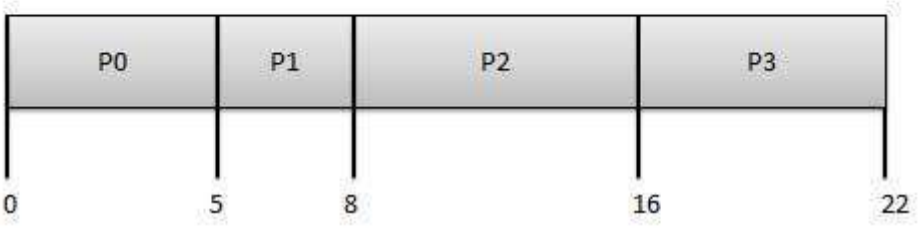
1. First Come First Serve (FCFS)
2. Shortest-Job-First (SJF) Scheduling
3. Shortest Remaining Time
4. Priority Scheduling
5. Round Robin Scheduling
6. Multilevel Queue Scheduling



FIRST COME FIRST SERVE (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0

P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.75

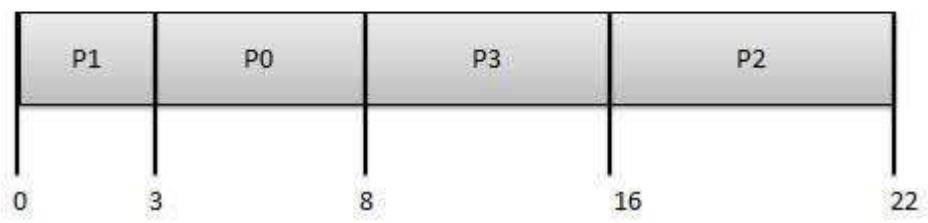
SHORTEST JOB NEXT (SJN)

- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Given: Table of processes, and their Arrival time, Execution time

Process	Arrival Time	Execution Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	14
P3	3	6	8

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8



Waiting time of each process is as follows −

Process	Waiting Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	14 - 2 = 12
P3	8 - 3 = 5

Average Wait Time: (0 + 4 + 12 + 5)/4 = 21 / 4 = 5.25

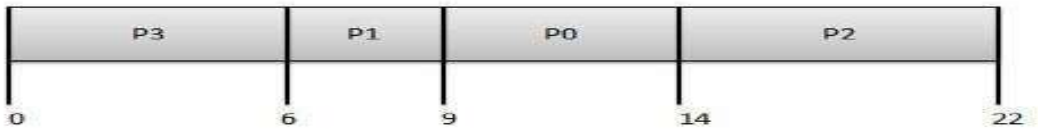
PRIORITY BASED SCHEDULING

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Given: Table of processes, and their Arrival time, Execution time, and priority. Here we are considering 1 is the lowest priority.

Process	Arrival Time	Execution Time	Priority	Service Time
P0	0	5	1	0
P1	1	3	2	11
P2	2	8	1	14
P3	3	6	3	5

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Waiting time of each process is as follows −

Process	Waiting Time
P0	0 - 0 = 0
P1	11 - 1 = 10
P2	14 - 2 = 12
P3	5 - 3 = 2

Average Wait Time: (0 + 10 + 12 + 2)/4 = 24 / 4 = 6

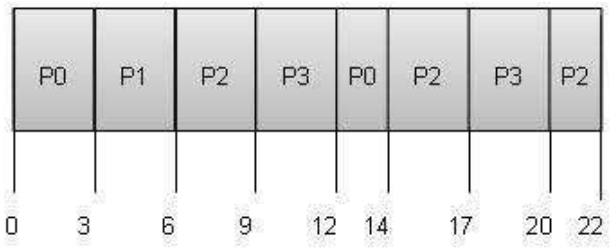
SHORTEST REMAINING TIME

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

ROUND ROBIN SCHEDULING

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.

Quantum = 3



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2

P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

MULTIPLE-LEVEL QUEUES SCHEDULING

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

Concurrency in Operating System

In this article, you will learn the concurrency in the operating system with its principles, issues, advantages and disadvantages.

What is Concurrency?

It refers to the execution of multiple instruction sequences at the same time. It occurs in an operating system when multiple process threads are executing concurrently. These threads can interact with one another via shared memory or message passing. Concurrency results in resource sharing, which causes issues like deadlocks and resource scarcity. It aids with techniques such as process coordination, memory allocation, and execution schedule to maximize throughput.

Principles of Concurrency

Today's technology, like multi-core processors and parallel processing, allows multiple processes and threads to be executed simultaneously. Multiple processes and threads can access the same memory space, the same declared variable in code, or even read or write to the same file.

The amount of time it takes a process to execute cannot be simply estimated, and you cannot predict which process will complete first, enabling you to build techniques to deal with the problems that concurrency creates.

Interleaved and overlapping processes are two types of concurrent processes with the same problems. It is impossible to predict the relative speed of execution, and the following factors determine it:

1. The way operating system handles interrupts

2. Other processes' activities
3. The operating system's scheduling policies

Problems in Concurrency

There are various problems in concurrency. Some of them are as follows:

1. Locating the programming errors

It's difficult to spot a programming error because reports are usually repeatable due to the varying states of shared components each time the code is executed.

2. Sharing Global Resources

Sharing global resources is difficult. If two processes utilize a global variable and both alter the variable's value, the order in which the many changes are executed is critical.

3. Locking the channel

It could be inefficient for the OS to lock the resource and prevent other processes from using it.

4. Optimal Allocation of Resources

It is challenging for the OS to handle resource allocation properly.

Issues of Concurrency

Various issues of concurrency are as follows:

1. Non-atomic

Operations that are non-atomic but interruptible by several processes may happen issues. A non-atomic operation depends on other processes, and an atomic operation runs independently of other processes.

2. Deadlock

In concurrent computing, it occurs when one group member waits for another member, including itself, to send a message and release a lock. Software and hardware locks are commonly used to arbitrate shared resources and implement process synchronization in parallel computing, distributed systems, and multiprocessing.

3. Blocking

A blocked process is waiting for some event, like the availability of a resource or completing an I/O operation. Processes may block waiting for resources, and a process may be blocked for a long time waiting for terminal input. If the process is needed to update some data periodically, it will be very undesirable.

4. Race Conditions

A race problem occurs when the output of a software application is determined by the timing or sequencing of other uncontrollable events. Race situations can also happen in multithreaded software, runs in a distributed environment, or is interdependent on shared resources.

5. Starvation

A problem in concurrent computing is where a process is continuously denied the resources it needs to complete its work. It could be caused by errors in scheduling or mutual exclusion algorithm, but resource leaks may also cause it.

Concurrent system design frequently requires developing dependable strategies for coordinating their execution, data interchange, memory allocation, and execution schedule to decrease response time and maximize throughput.

Advantages and Disadvantages of Concurrency in Operating System

Various advantages and disadvantages of Concurrency in Operating systems are as follows:

Advantages

1. Better Performance

It improves the operating system's performance. When one application only utilizes the processor, and another only uses the disk drive, the time it takes to perform both apps simultaneously is less than the time it takes to run them sequentially.

2. Better Resource Utilization

It enables resources that are not being used by one application to be used by another.

3. Running Multiple Applications

It enables you to execute multiple applications simultaneously.

Disadvantages

1. It is necessary to protect multiple applications from each other.
2. It is necessary to use extra techniques to coordinate several applications.
3. Additional performance overheads and complexities in OS are needed for switching between applications.

PROCESS SYNCHRONIZATION

In this tutorial, we will be covering the concept of Process synchronization in an Operating System.

Process Synchronization was introduced to handle problems that arose while multiple process executions.

Process is categorized into two types on the basis of synchronization and these are given below:

- Independent Process
- Cooperative Process

Independent Processes

Two processes are said to be independent if the execution of one process does not affect the execution of another process.

Cooperative Processes

Two processes are said to be cooperative if the execution of one process affects the execution of another process. These processes need to be synchronized so that the order of execution can be guaranteed.

Process Synchronization

It is the task phenomenon of coordinating the execution of processes in such a way that no two processes can have access to the same shared data and resources.

- It is a procedure that is involved in order to preserve the appropriate order of execution of cooperative processes.
- In order to synchronize the processes, there are various synchronization mechanisms.
- Process Synchronization is mainly needed in a multi-process system when multiple processes are running together, and more than one processes try to gain access to the same shared resource or any data at the same time.

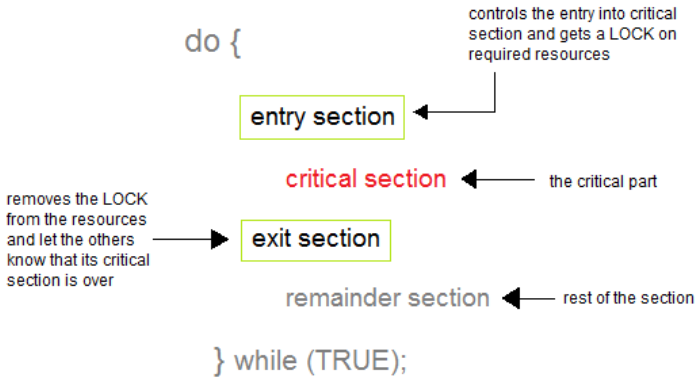
Race Condition

At the time when more than one process is either executing the same code or accessing the same memory or any shared variable; In that condition, there is a possibility that the output or the value of the shared variable is wrong so for that purpose all the processes are doing the race to say that my output is correct. This condition is commonly known as **a race condition.** As several processes access and process the manipulations on the same data in a concurrent manner and due to which the outcome depends on the particular order in which the access of data takes place.

Mainly this condition is a situation that may occur inside the **critical section**. Race condition in the critical section happens when the result of multiple thread execution differs according to the order in which the threads execute. But this condition is critical sections can be avoided if the critical section is treated as an atomic instruction. Proper thread synchronization using locks or atomic variables can also prevent race conditions.

Critical Section Problem

A Critical Section 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 entry to the critical section is mainly handled by `wait()` function while the exit from the critical section is controlled by the `signal()` function.



Entry Section

In this section mainly the process requests for its entry in the critical section.

Exit Section

This section is followed by the critical section.

The solution to the Critical Section Problem

A solution to the critical section problem 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, the system must grant the process permission to get into its critical section.

Solutions for the Critical Section

The critical section plays an important role in Process Synchronization so that the problem must be solved.

Some widely used method to solve the critical section problem are as follows:

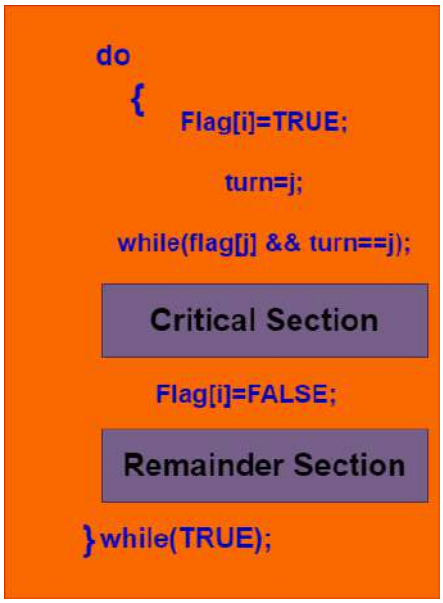
1.Peterson's Solution

This is widely used and software-based solution to critical section problems. Peterson's solution was developed by a computer scientist Peterson that's why it is named so.

With the help of this solution whenever a process is executing in any critical state, then the other process only executes the rest of the code, and vice-versa can happen. This method also helps to make sure of the thing that only a single process can run in the critical section at a specific time.

This solution preserves all three conditions:

- Mutual Exclusion is comforted as at any time only one process can access the critical section.
- Progress is also comforted, as a process that is outside the critical section is unable to block other processes from entering into the critical section.
- Bounded Waiting is assured as every process gets a fair chance to enter the Critical section.



The above shows the structure of process **Pi in Peterson's solution.**

- Suppose there are **N processes (P1, P2, ... PN)** and as at some point of time every process requires to enter in the **Critical Section**
- A **FLAG[]** array of size N is maintained here which is by default false. Whenever a process requires to enter in the critical section, it has to set its flag as true. Example: If Pi wants to enter it will set **FLAG[i]=TRUE.**
- Another variable is called **TURN** and is used to indicate the process number that is currently waiting to enter into the critical section.
- The process that enters into the critical section while exiting would change the **TURN** to another number from the list of processes that are ready.
- Example: If the turn is 3 then P3 enters the Critical section and while exiting turn=4 and therefore P4 breaks out of the wait loop.

Synchronization Hardware

Many systems provide hardware support for critical section code. The critical section problem could be solved easily in a single-processor environment if we could disallow interrupts to occur while a shared variable or resource is being modified.

In this manner, we could be sure that the current sequence of instructions would be allowed to execute in order without pre-emption. Unfortunately, this solution is not feasible in a multiprocessor environment.

Disabling interrupt on a multiprocessor environment can be time-consuming as the message is passed to all the processors.

This message transmission lag delays the entry of threads into the critical section, and the system efficiency decreases.

Mutex Locks

As the synchronization hardware solution is not easy to implement for everyone, a strict software approach called Mutex Locks was introduced. In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside the critical section, and in the exit section that LOCK is released.

As the resource is locked while a process executes its critical section hence no other process can access it.

SEMAPHORES

In 1965, Dijkstra proposed a new and very significant technique for managing concurrent processes by using the value of a simple integer variable to synchronize the progress of interacting processes. This integer variable is called a **semaphore**. So it is basically a synchronizing tool and is accessed only through two low standard atomic operations, **wait** and **signal** designated by P(S) and V(S) respectively.

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

In very simple words, the **semaphore** is a variable that can hold only a non-negative Integer value, shared between all the threads, with operations **wait** and **signal**, which work as follow:

```
P(S): if S >= 1 then S := S - 1
      else <block and enqueue the process>;


V(S): if <some process is blocked on the queue>
        then <unblock a process>
      else S := S + 1;
```

Copy

The classical definitions of **wait** and **signal** are:

- **Wait**: This operation decrements the value of its argument S, as soon as it would become non-negative(greater than or equal to 1). This Operation mainly helps you to control the entry of a task into the critical section. In the case of the negative or zero value, no operation is executed. wait() operation was originally termed as P; so it is also known as **P(S) operation**. The definition of wait operation is as follows:

```
wait(S)

{

   while (S<=0);//no operation

   S--;

}
```

Note:

When one process modifies the value of a semaphore then, no other process can simultaneously modify that same semaphore's value. In the above case the integer value of S(S<=0) as well as the possible modification that is S-- must be executed without any interruption.

- **Signal**: Increments the value of its argument S, as there is no more process blocked on the queue. This Operation is mainly used to control the exit of a task from the critical section.signal() operation was originally termed as V; so it is also known as **V(S) operation**. The definition of signal operation is as follows:

```
signal(S)

{

S++;

}
```

Also, note that all the modifications to the integer value of semaphore in the wait() and signal() operations must be executed indivisibly.

Properties of Semaphores

1. It's simple and always have a non-negative integer value.
2. Works with many processes.
3. Can have many different critical sections with different semaphores.
4. Each critical section has unique access semaphores.
5. Can permit multiple processes into the critical section at once, if desirable.

We will now cover the types of semaphores in the Operating system;

Types of Semaphores

Semaphores are mainly of two types in Operating system:

1. **Binary Semaphore:**

   It is a special form of semaphore used for implementing mutual exclusion, hence it is often called a **Mutex**. A binary semaphore is initialized to 1 and only takes the values 0 and 1 during the execution of a program. In Binary Semaphore, the wait operation works only if the value of semaphore = 1, and the signal operation succeeds when the semaphore= 0. Binary Semaphores are easier to implement than counting semaphores.

2. **Counting Semaphores:**

   These are used to implement **bounded concurrency**. The Counting semaphores can range over an **unrestricted domain**. These can be used to control access to a given resource that consists of a finite number of Instances. Here the semaphore count is used to indicate the number of available resources. If the resources are added then the semaphore count automatically gets incremented and if the resources are removed, the count is decremented. Counting Semaphore has no mutual exclusion.

Example of Use

Here is a simple step-wise implementation involving declaration and usage of semaphore.

```
Shared var mutex: semaphore = 1;

Process i

   begin

   .

   .

   P(mutex);

   execute CS;

   V(mutex);

   .

   .

   End;
```

Advantages of Semaphores

Benefits of using Semaphores are as given below:

- With the help of semaphores, there is a flexible management of resources.

- Semaphores are machine-independent and they should be run in the machine-independent code of the microkernel.
- Semaphores do not allow multiple processes to enter in the critical section.
- They allow more than one thread to access the critical section.
- As semaphores follow the mutual exclusion principle strictly and these are much more efficient than some other methods of synchronization.
- No wastage of resources in semaphores because of busy waiting in semaphores as processor time is not wasted unnecessarily to check if any condition is fulfilled in order to allow a process to access the critical section.

Disadvantages of Semaphores

- One of the biggest limitations is that semaphores may lead to priority inversion; where low priority processes may access the critical section first and high priority processes may access the critical section later.
- To avoid deadlocks in the semaphore, the Wait and Signal operations are required to be executed in the correct order.
- Using semaphores at a large scale is impractical; as their use leads to loss of modularity and this happens because the wait() and signal() operations prevent the creation of the structured layout for the system.
- Their use is not enforced but is by convention only.
- With improper use, a process may block indefinitely. Such a situation is called **Deadlock**. We will be studying deadlocks in detail in coming lessons.

UNIT- 3

MEMORY MANAGEMENT

Memory allocation Techniques

1. **Contiguous memory allocation**

2. **Non-contiguous memory allocation**

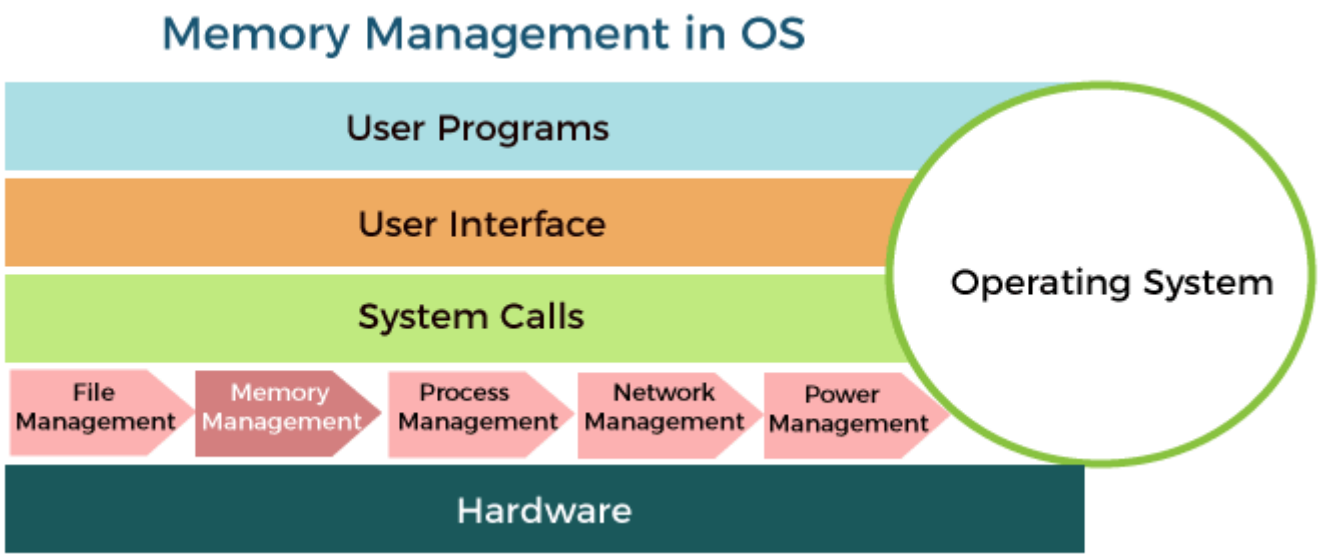
Swapping

Paging, Segmentation, virtual memory using paging,

Demand paging, page fault handling.

MEMORY MANAGEMENT

Memory is the important part of the computer that is used to store the data. Its management is critical to the computer system because the amount of main memory available in a computer system is very limited. At any time, many processes are competing for it. Moreover, to increase performance, several processes are executed simultaneously. For this, we must keep several processes in the main memory, so it is even more important to manage them effectively.



in a uni-programming system, main memory is divided into two parts: one part for the operating system (resident monitor, kernel) and one part for the user program currently being executed.

In a multiprogramming system, the “user” part of memory must be further subdivided to accommodate multiple processes. The task of subdivision is carried out dynamically by the operating system and is known as **memory management**

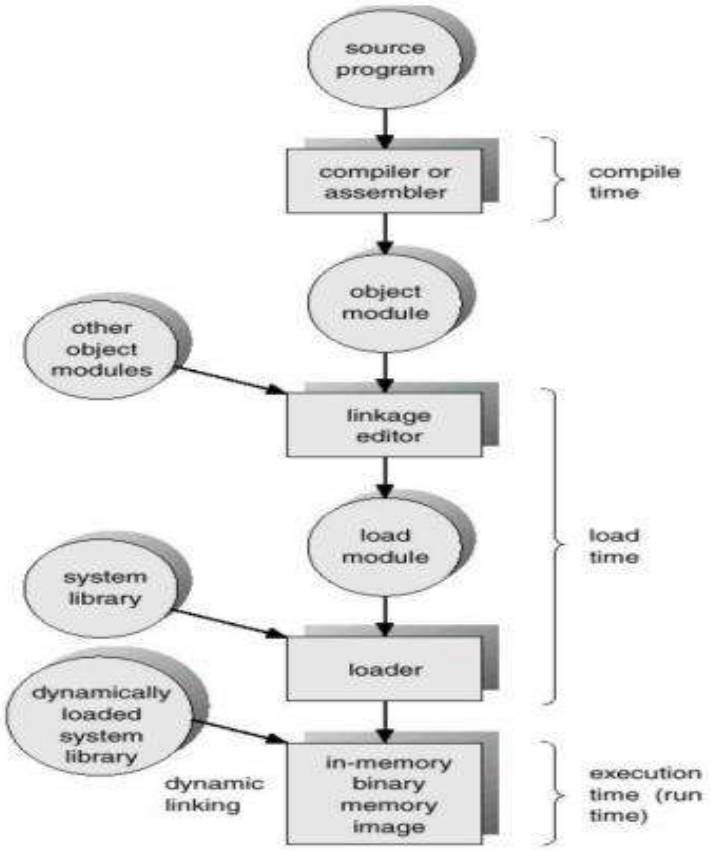
BINDING OF INSTRUCTIONS AND DATA TO MEMORY

Address binding of instructions and data to memory addresses can happen at three different stages.

1. **Compile time:** The compile time is the time taken to compile the program or source code. During compilation, if memory location known a priori, then it generates absolute codes.

2. **Load time**: It is the time taken to link all related program file and load into the main memory. It must generate relocatable code if memory location is not known at compile time.

3. **Execution time**: It is the time taken to execute the program in main memory by processor. Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., base and limit registers).



(Multistep processing of a user program.)

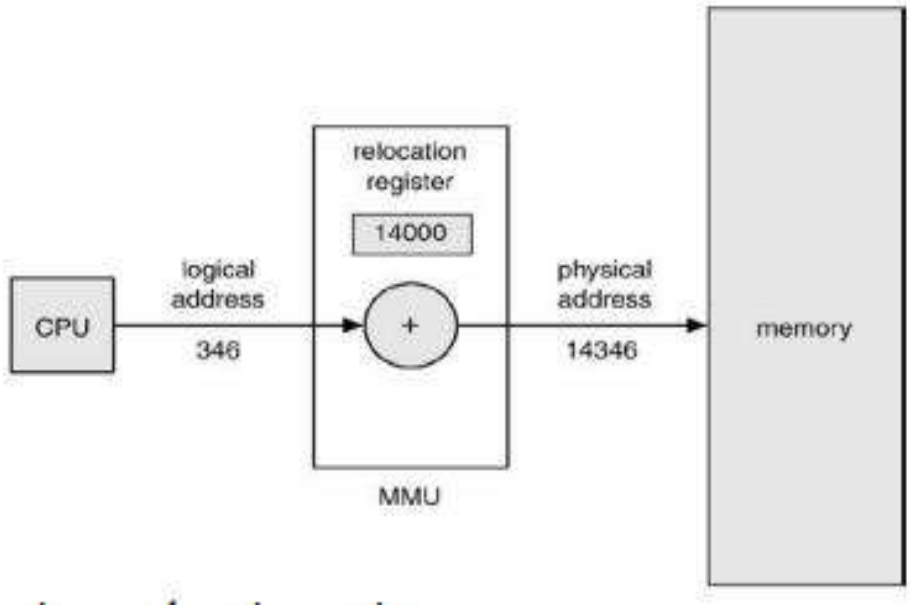
LOGICAL- VERSUS PHYSICAL-ADDRESS SPACE

⇒ An address generated by the CPU is commonly referred to as a logical address or a virtual address whereas an address seen by the main memory unit is commonly referred to as a physical address.

⇒ The set of all logical addresses generated by a program is a logical-address space whereas the set of all physical addresses corresponding to these logical addresses is a physicaladdress space.

⇒ Logical and physical addresses are the same in compile-time and load-time addressbinding schemes; logical (virtual) and physical addresses differ in execution-time addressbinding scheme.

⇒ The Memory Management Unit is a hardware device that maps virtual to physical address. In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory as follows:



(Dynamic relocation using a relocation register)

DYNAMIC LOADING

⇒ It loads the program and data dynamically into physical memory to obtain better memory-space utilization.
⇒ With dynamic loading, a routine is not loaded until it is called.
⇒ The advantage of dynamic loading is that an unused routine is never loaded.
⇒ This method is useful when large amounts of code are needed to handle infrequently occurring cases, such as error routines.
⇒ Dynamic loading does not require special support from the operating system.

DYNAMIC LINKING

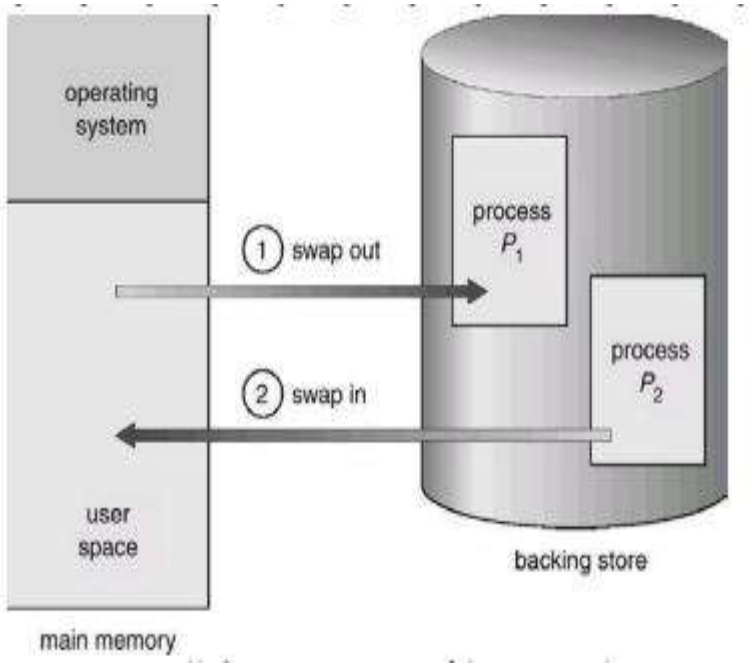
⇒ Linking postponed until execution time.
⇒ Small piece of code (stub) used to locate the appropriate memory-resident library routine.
⇒ Stub replaces itself with the address of the routine and executes the routine.
⇒ Operating system needed to check if routine is in processes memory address.
⇒ Dynamic linking is particularly useful for libraries.

OVERLAYS

⇒ Keep in memory only those instructions and data that are needed at any given time.
⇒ Needed when process is larger than amount of memory allocated to it.
⇒ Implemented by user, no special support needed from operating system, programming design of overlay structure is complex.

SWAPPING

⇒ A process can be swapped temporarily out of memory to a backing store (large disc), and then brought back into memory for continued execution.
⇒ **Roll out, roll in**: A variant of this swapping policy is used for priority-based scheduling Algorithms. If a higher-priority process arrives and wants service, the memory manager can swap out the lower-priority process so that it can load and execute the higher-priority process. When the higher-priority process finishes, the lower-priority process can be swapped back in and continued. This variant of swapping is called roll out, roll in.
⇒ Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped.
⇒ Modified versions of swapping are found on many systems (UNIX, Linux, and Windows).

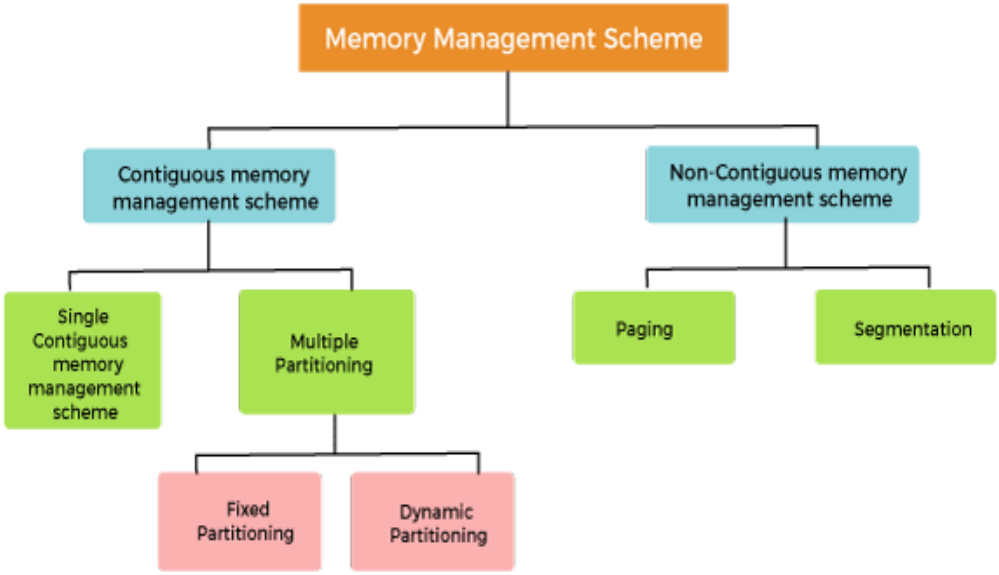


(Schematic View of Swapping

MEMORY MANAGEMENT TECHNIQUES:

The Memory management Techniques can be classified into following main categories:

- Contiguous memory management schemes
- Non-Contiguous memory management schemes



Classification of memory management schemes

CONTIGUOUS MEMORY MANAGEMENT SCHEMES:

In a Contiguous memory management scheme, each program occupies a single contiguous block of storage locations, i.e., a set of memory locations with consecutive addresses.

SINGLE CONTIGUOUS MEMORY MANAGEMENT SCHEMES:

The Single contiguous memory management scheme is the simplest memory management scheme used in the earliest generation of computer systems. In this scheme, the main memory is divided into two contiguous areas or partitions. The operating systems reside permanently in one partition, generally at the lower memory, and the user process is loaded into the other partition.

Advantages of Single contiguous memory management schemes:

- Simple to implement.
- Easy to manage and design.
- In a Single contiguous memory management scheme, once a process is loaded, it is given full processor's time, and no other processor will interrupt it.

Disadvantages of Single contiguous memory management schemes:

- Wastage of memory space due to unused memory as the process is unlikely to use all the available memory space.
- The CPU remains idle, waiting for the disk to load the binary image into the main memory.

- It can not be executed if the program is too large to fit the entire available main memory space.
- It does not support multiprogramming, i.e., it cannot handle multiple programs simultaneously.

MULTIPLE PARTITIONING:

The single Contiguous memory management scheme is inefficient as it limits computers to execute only one program at a time resulting in wastage in memory space and CPU time. The problem of inefficient CPU use can be overcome using multiprogramming that allows more than one program to run concurrently. To switch between two processes, the operating systems need to load both processes into the main memory. The operating system needs to divide the available main memory into multiple parts to load multiple processes into the main memory. Thus multiple processes can reside in the main memory simultaneously.

The multiple partitioning schemes can be of two types:

- Fixed Partitioning
- Dynamic Partitioning

FIXED PARTITIONING

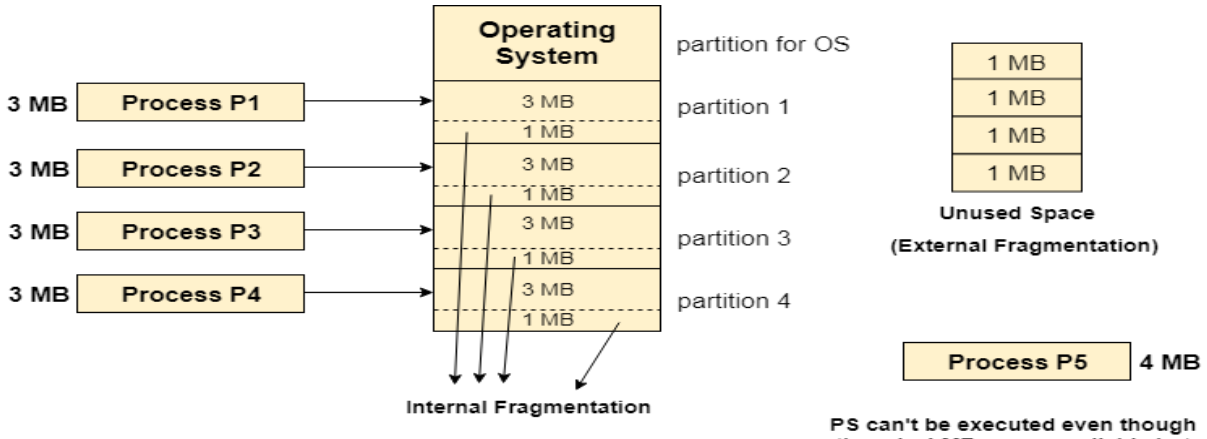
The main memory is divided into several fixed-sized partitions in a fixed partition memory management scheme or static partitioning. These partitions can be of the same size or different sizes. Each partition can hold a single process. The number of partitions determines the degree of multiprogramming, i.e., the maximum number of processes in memory. These partitions are made at the time of system generation and remain fixed after that.

Advantages of Fixed Partitioning memory management schemes:

- Simple to implement.
- Easy to manage and design.

Disadvantages of Fixed Partitioning memory management schemes:

- This scheme suffers from internal fragmentation.
- The number of partitions is specified at the time of system generation.

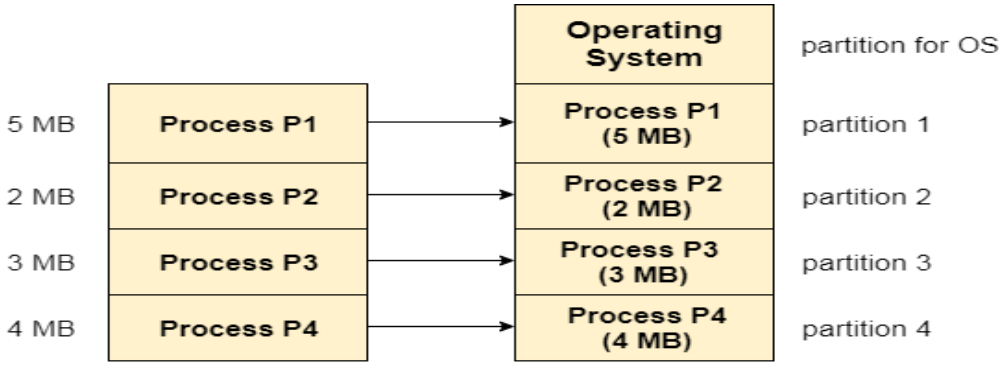


Fixed Partitioning

(Contiguous memory allocation)

DYNAMIC PARTITIONING

The dynamic partitioning was designed to overcome the problems of a fixed partitioning scheme. In a dynamic partitioning scheme, each process occupies only as much memory as they require when loaded for processing. Requested processes are allocated memory until the entire physical memory is exhausted or the remaining space is insufficient to hold the requesting process. In this scheme the partitions used are of variable size, and the number of partitions is not defined at the system generation time.



Dynamic Partitioning

(Process Size = Partition Size)

PARTITION SELECTION POLICY:

When the multiple memory holes (partitions) are large enough to contain a process, the operating system must use an algorithm to select in which hole the process will be loaded.

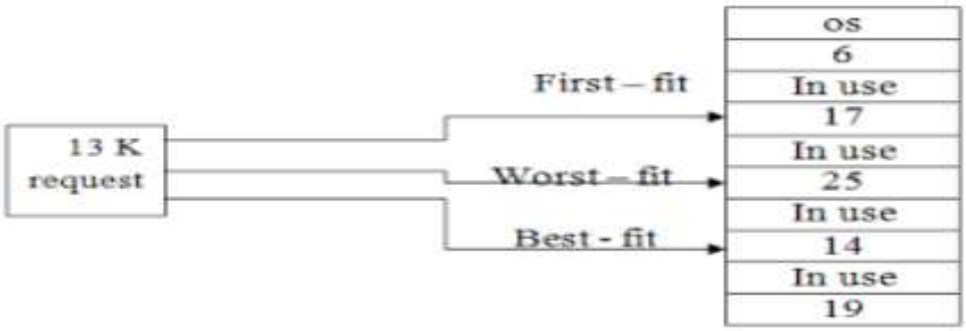
The partition selection algorithm are as follows:

⇒ **First-fit:** The OS looks at all sections of free memory. The process is allocated to the firsthole found that is big enough size than the size of process.

⇒ **Next Fit:** The next fit search starts at the last hole allocated and The process is allocated tothe next hole found that is big enough size than the size of process.

⇒ **Best-fit:** The Best Fit searches the entire list of holes to find the smallest hole that is big enough size than the size of process.

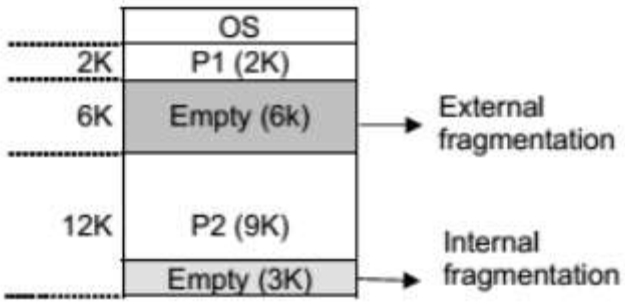
⇒ **Worst-fit**: The Worst Fit searches the entire list of holes to find the largest hole that is big enough size than the size of process.



Fragmentation: The wasting of memory space is called fragmentation. There are two types of fragmentation as follows:

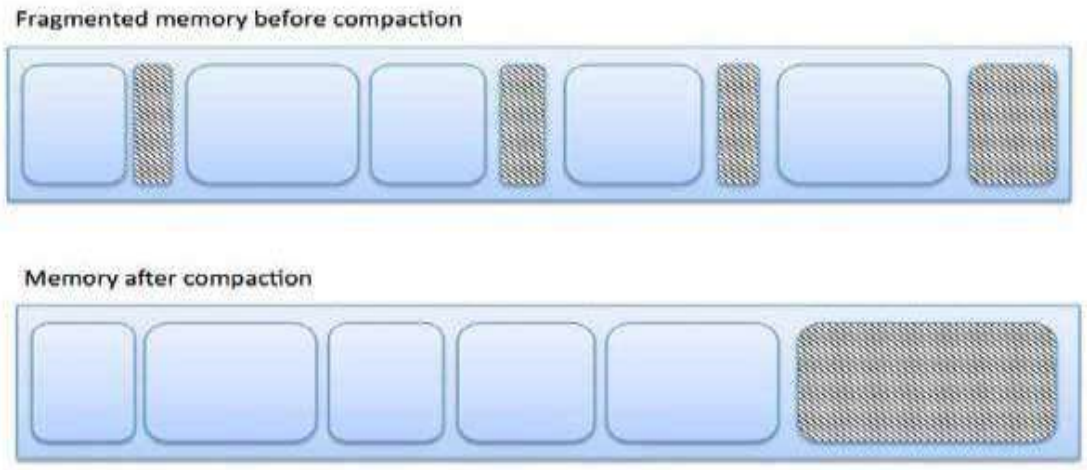
1. **External Fragmentation**: The total memory space exists to satisfy a request, but it is not contiguous. This wasted space not allocated to any partition is called external fragmentation. The external fragmentation can be reduce by compaction. The goal is to shuffle the memory contents to place all free memory together in one large block. Compaction is possible only if relocation is dynamic, and is done at execution time.

2. **Internal Fragmentation:** The allocated memory may be slightly larger than requested memory. The wasted space within a partition is called internal fragmentation. One method to reduce internal fragmentation is to use partitions of different size



RE-ALLOCATION OF MEMORY (OR) COMPACTION:

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory −



- Compaction is a process in which the free space is collected in a large memory chunk to make some space available for processes.
- In memory management, swapping creates multiple fragments in the memory because of the processes moving in and out.
- Compaction refers to combining all the empty spaces together and processes.
- Compaction helps to solve the problem of fragmentation, but it requires too much of CPU time. It moves all the occupied areas of store to one end and leaves one large free space for incoming jobs, instead of numerous small ones.
- In compaction, the system also maintains relocation information and it must be performed on each new allocation of job to the memory or completion of job from memory.
- Compaction is not always easy. If relocation is static and is done at assembly or load time, compaction cannot be done. Compaction is possible only if relocation is dynamic and performed at execution time.

NON-CONTIGUOUS MEMORY MANAGEMENT SCHEMES:

In a Non-Contiguous memory management scheme, the program is divided into different blocks and loaded at different portions of the memory that need not necessarily be adjacent to one another. This scheme can be classified depending upon the size of blocks and whether the blocks reside in the main memory or not.

What is paging?

Paging is a technique that eliminates the requirements of contiguous allocation of main memory. In this, the main memory is divided into fixed-size blocks of physical memory called frames. The size of a frame should be kept the same as that of a page to maximize the main memory and avoid external fragmentation.

Advantages of paging:

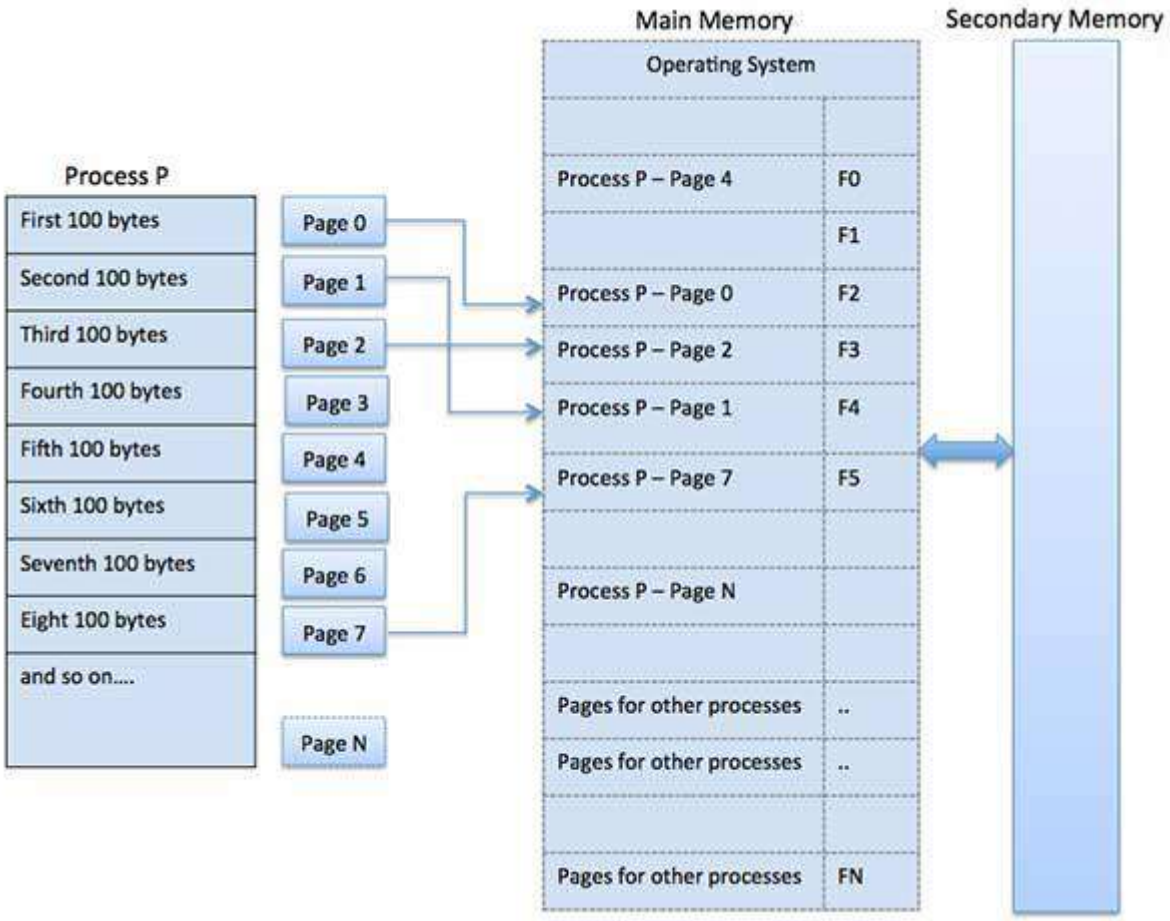
- Pages reduce external fragmentation.
- Simple to implement.
- Memory efficient.
- Due to the equal size of frames, swapping becomes very easy.
- It is used for faster access of data.

What is Segmentation?

Segmentation is a technique that eliminates the requirements of contiguous allocation of main memory. In this, the main memory is divided into variable-size blocks of physical memory called segments. It is based on the way the programmer follows to structure their programs. With segmented memory allocation, each job is divided into several segments of different sizes, one for each module. Functions, subroutines, stack, array, etc., are examples of such modules.

PAGING

- In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.
- The main idea behind the paging is to divide each process in the form of **pages**. The main memory will also be divided in the form of **frames**.
- The size of a frame should be **kept same** as that of a page to have maximum utilization of the main memory and to avoid external fragmentation



ADDRESS TRANSLATION

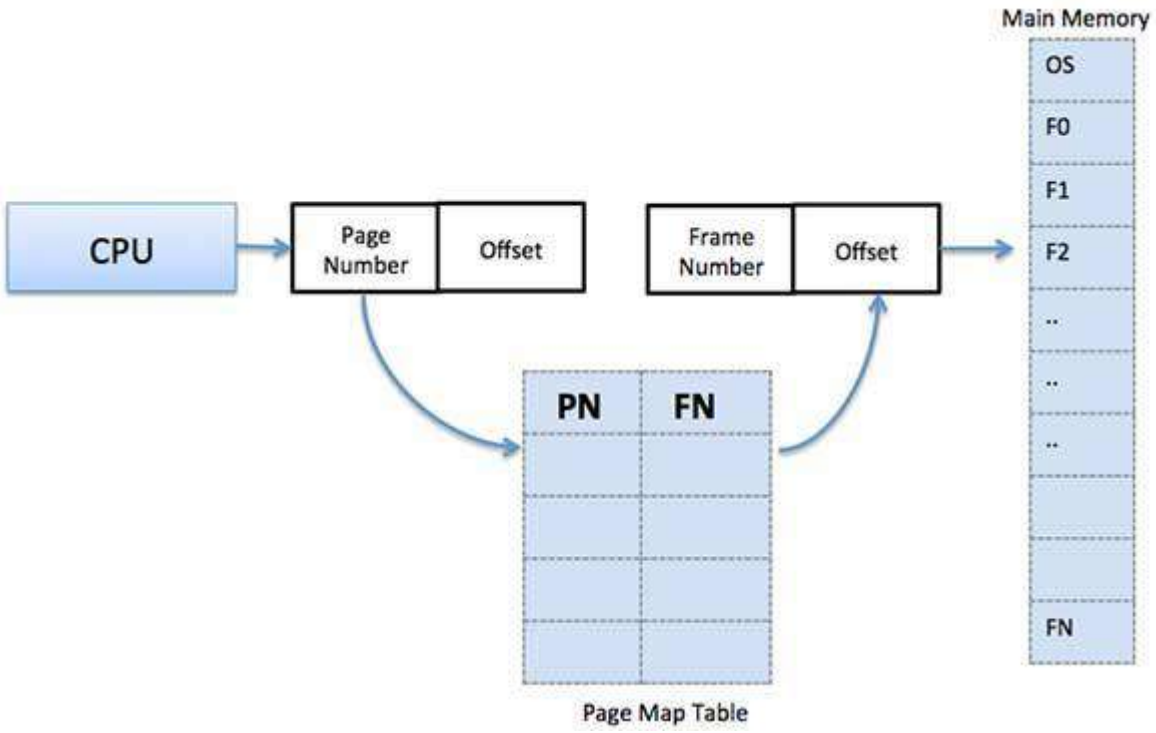
Page address is called **logical address** and represented by **page number** and the **offset**.

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

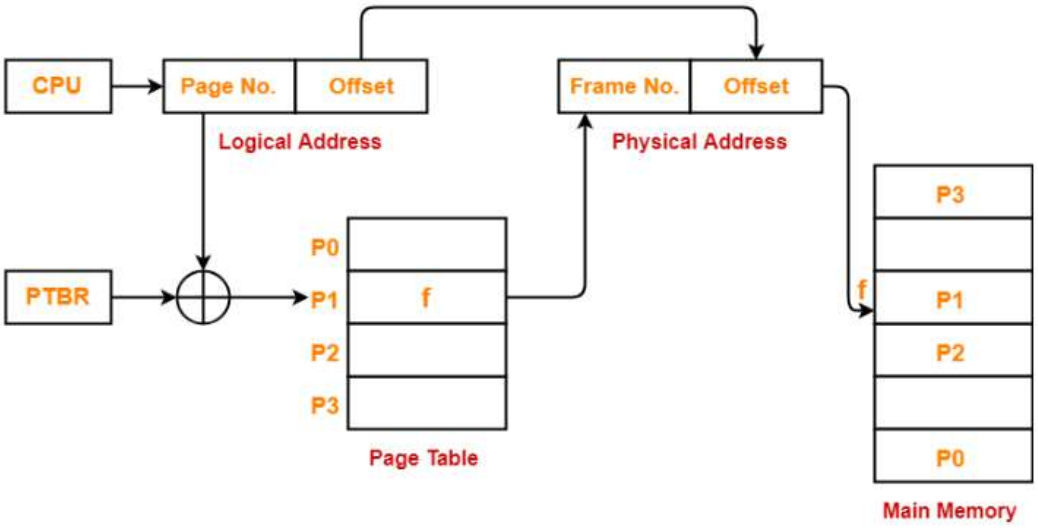
A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.



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.

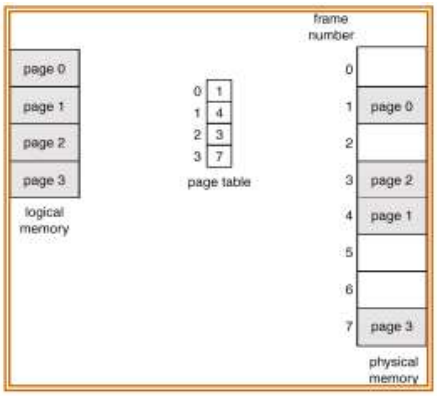
When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system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



Translating Logical Address into Physical Address

- The paging model of memory is shown in below figure. The page size is defined by the hardware. The size of a page is typically of a power of 2, varying between 512 bytes and 16 MBper page, depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy.
- If the size of logical address is 2m , and a page size is 2n addressing units, then the high order m-n bits of a logical address designate the page number, and the n low order bits designate the page offset.



- Keep track of all free frames.

- To run a program of size n pages, need to find n free frames and load program.
- Set up a page table to translate logical to physical addresses.
- infernal fragmentation may occur.

- **Let us take an example**.

Suppose a program of size 4B.

Page size =2b

Then total no of page =2

Suppose the memory size =16 b

Frame size =page size=2B

Then frame =8

PAGE TABLE

F2	p0
F4	p1

p0

P1

0	1
2	3

F0
F1
F2
F3
F4
F5
F6
F7

0	1	
2	3	
4	5	P0
6	7	
8	9	P1
10	11	
12	13	
14	15	

CPU will generate logical address

– lets it want byte 3 of the process which is page no 1

Cpu will generate 2 b address 1 b for page no + 1b for page offset

So byte 3 can be represent as 1 1 –i.e page no 1 and offset 1

In page table page 1 is in frame 4

formula.

- Physical address = page size of logical memory X frame number + offset
- Physical address=2 x 4+1=9
- So byte no 9

ADVANTAGES OF PAGING

• Paging reduces external 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.

Disadvantages of Paging

• Page table requires extra memory space, so may not be good for a system having small RAM.

• Suffers from internal fragmentation.

• There is an increase in time taken to fetch the instruction since now two memory accesses are required.

SEGMENTATION

It is another scheme of memory management and it generally supports the user view of memory.

The Logical address space is basically the collection of segments. Each segment has a name and a length.

Basically, a process is divided into segments. Like paging, segmentation divides or segments the memory.

But there is a difference and that is while the **paging** divides the memory into a **fixed size** and on the other hand,

segmentation divides the **memory into variable segments** these are then loaded into logical memory space.

A program is a collection of segments A segment is a logical unit such as:

main program

Procedure

function method

object

local variables,

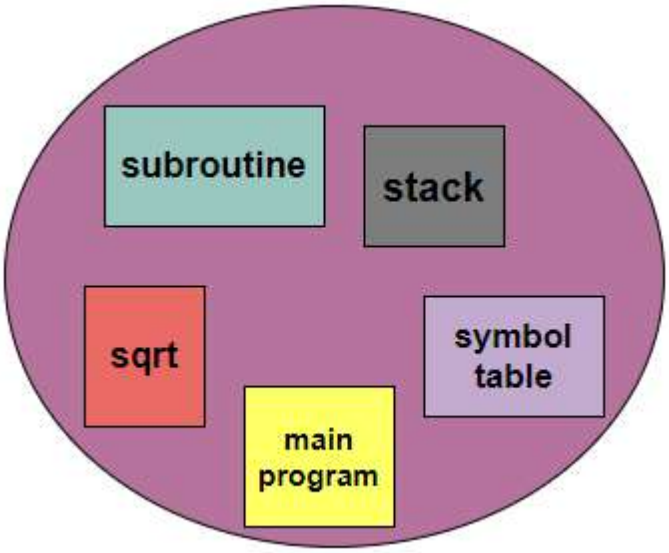
global variables

common block

stack

symbol table

arrays



Segment Table-

- Segment table is a table that stores the information about each segment of the process.
- It has two columns.
- First column stores the size or length of the segment.
- Second column stores the base address or starting address of the segment in the main memory.
- Segment table is stored as a separate segment in the main memory.
- Segment table base register (STBR) stores the base address of the segment table.

For the above illustration, consider the segment table is-

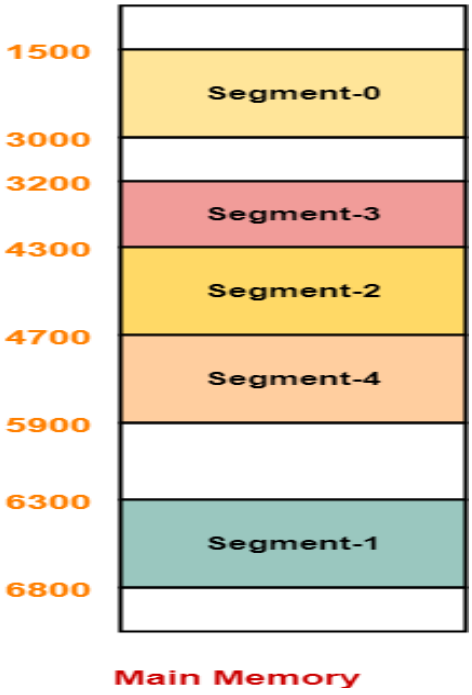
	Limit	Base
Seg-0	1500	1500
Seg-1	500	6300
Seg-2	400	4300
Seg-3	1100	3200
Seg-4	1200	4700

Segment Table

Here,

- Limit indicates the length or size of the segment.
- Base indicates the base address or starting address of the segment in the main memory.

In accordance to the above segment table, the segments are stored in the main memory as-



Translating Logical Address into Physical Address-

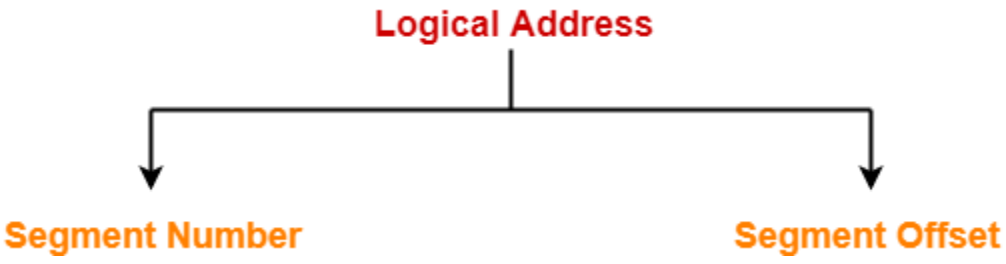
- CPU always generates a logical address.
- A physical address is needed to access the main memory.

Following steps are followed to translate logical address into physical address-

Step-01:

CPU generates a logical address consisting of two parts-

1. Segment Number
2. Segment Offset



- Segment Number specifies the specific segment of the process from which CPU wants to read the data.
- Segment Offset specifies the specific word in the segment that CPU wants to read.

Step-02:

- For the generated segment number, corresponding entry is located in the segment table.
- Then, segment offset is compared with the limit (size) of the segment.

Now, two cases are possible-

Case-01: Segment Offset >= Limit

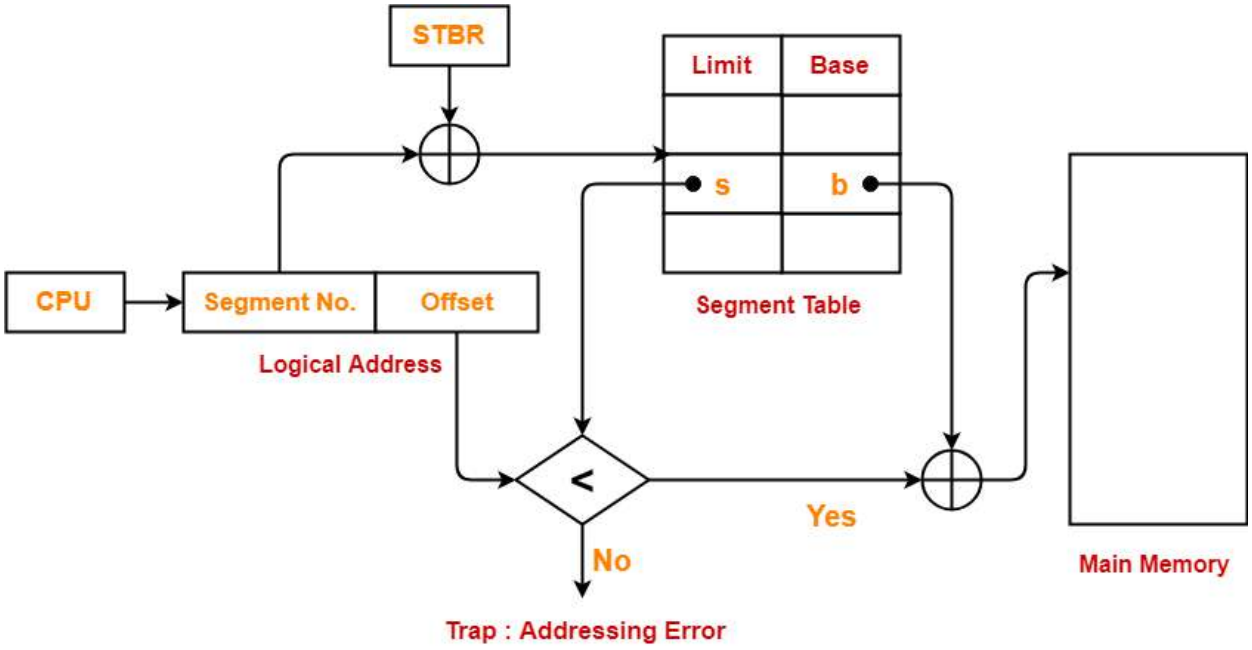
- If segment offset is found to be greater than or equal to the limit, a trap is generated.

Case-02: Segment Offset < Limit

- If segment offset is found to be smaller than the limit, then request is treated as a valid request.
- The segment offset must always lie in the range [0, limit-1],
- Then, segment offset is added with the base address of the segment.
- The result obtained after addition is the address of the memory location storing the required word.

Diagram-

The following diagram illustrates the above steps of translating logical address into physical address-



Translating Logical Address into Physical Address

Advantages-

The advantages of segmentation are-

- It allows to divide the program into modules which provides better visualization.
- Segment table consumes less space as compared to **Page Table** in paging.
- It solves the problem of internal fragmentation.

Disadvantages-

The disadvantages of segmentation are-

- There is an overhead of maintaining a segment table for each process.
- The time taken to fetch the instruction increases since now two memory accesses are required.
- Segments of unequal size are not suited for swapping.
- It suffers from external fragmentation as the free space gets broken down into smaller pieces with the processes being loaded and removed from the main memory.

PAGING VS SEGMENTATION

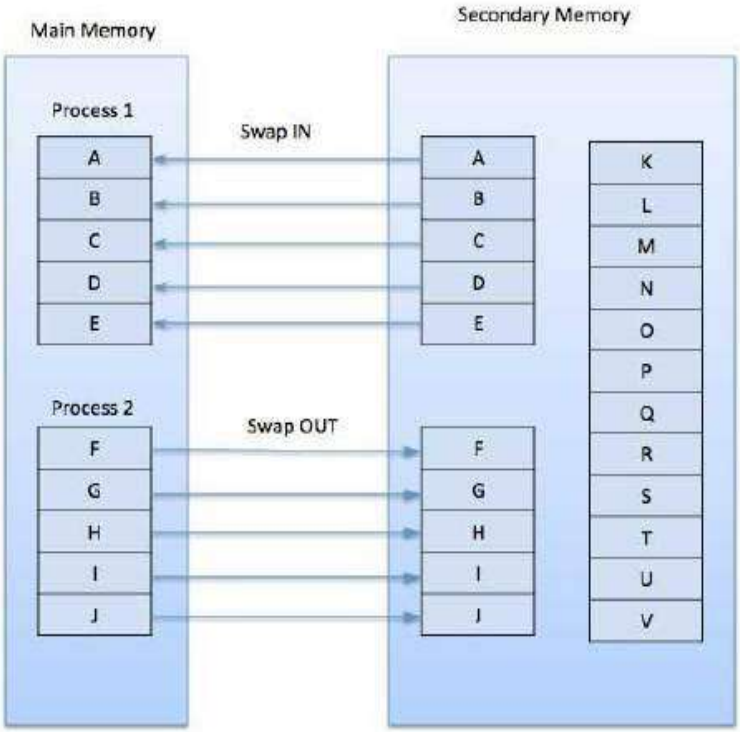
Sr No.	Paging	Segmentation

1	Non-Contiguous memory allocation	Non-contiguous memory allocation
2	Paging divides program into fixed size pages.	Segmentation divides program into variable size segments.
3	OS is responsible	Compiler is responsible.
4	Paging is faster than segmentation	Segmentation is slower than paging
5	Paging is closer to Operating System	Segmentation is closer to User
6	It suffers from internal fragmentation	It suffers from external fragmentation
7	There is no external fragmentation	There is no external fragmentation
8	Logical address is divided into page number and page offset	Logical address is divided into segment number and segment offset
9	Page table is used to maintain the page information.	Segment Table maintains the segment information
10	Page table entry has the frame number and some flag bits to represent details about pages.	Segment table entry has the base address of the segment and some protection bits for the segments.

Virtual Memory using paging

Virtual Memory is a storage mechanism which offers user an illusion of having a very big main memory. It is done by treating a part of secondary memory as the main memory.

- In Virtual memory, the user can store processes with a bigger size than the available main memory.
- Instead of loading one long process in the main memory, the OS loads the various parts of more than one process in the main memory.
- Virtual memory is needed whenever your computer doesn't have space in the physical memory.
- Virtual memory is mostly implemented with demand paging and demand segmentation.

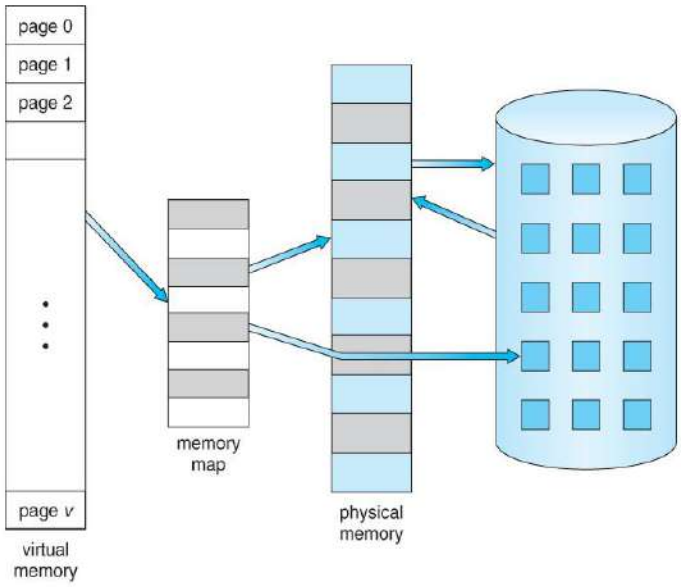


Advantages of Virtual Memory

1. The degree of Multiprogramming will be increased.
2. User can run large application with less real RAM.
3. There is no need to buy more memory RAMs.

Disadvantages of Virtual Memory

1. The system becomes slower since swapping takes time.
2. It takes more time in switching between applications.
3. The user will have the lesser hard disk space for its use.

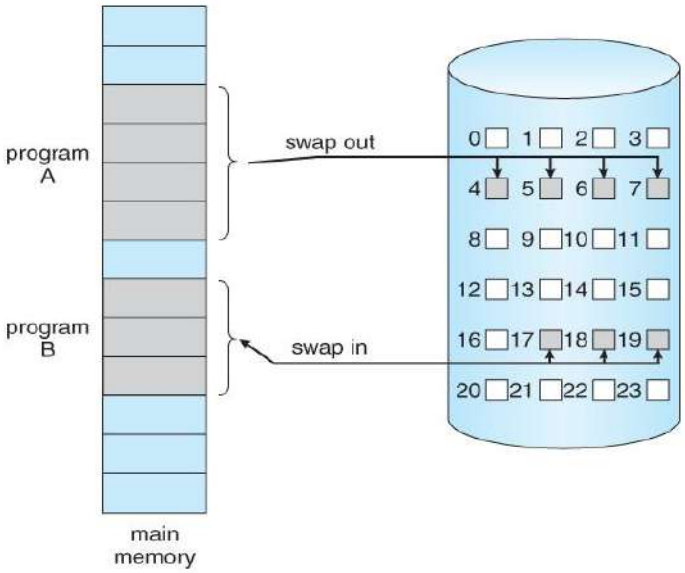


DEMAND PAGING

A demand paging system is similar to the paging system with swapping feature. When we want to execute a process we swap it into the memory.

A swapper manipulates entire process where as a pager is concerned with the individual pages of a process. The demand paging concept is using pager rather than swapper. When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory. The transfer of a paged memory to contiguous disk space

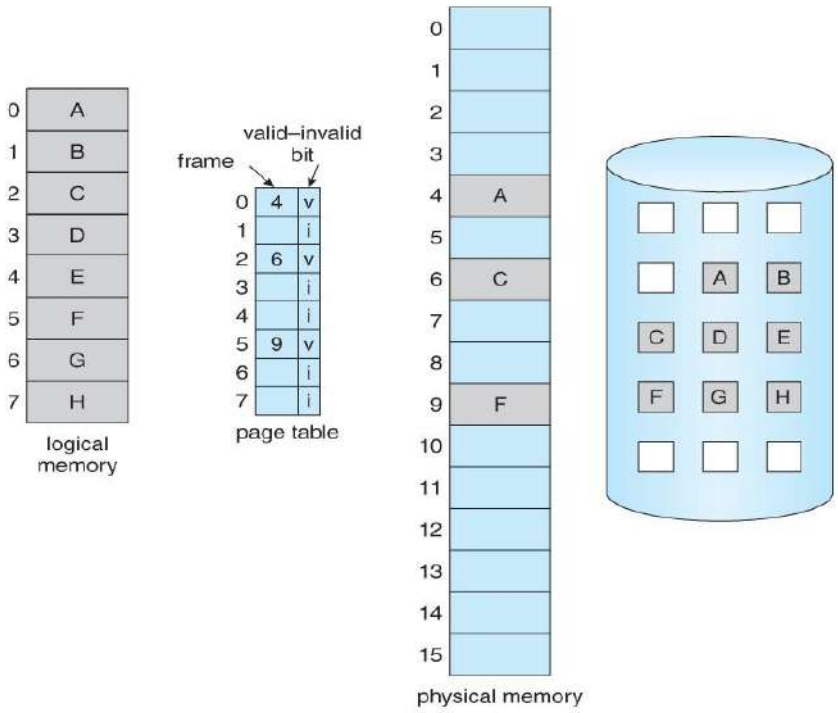
is shown in below figure.



Thus it avoids reading into memory pages that will not used any way decreasing the swap time and the amount of physical memory needed.

In this technique we need some hardware support to distinct between the pages that are in memory and those that are on the disk.

A valid and invalid bit is used for this purpose. When this bit is set to valid it indicates that the associate page is in memory. If the bit is set to invalid it indicates that the page is either not valid or is valid but currently not in the disk.



Marking a page invalid will have no effect if the process never attempts to access that page. So while a process executes and access pages that are memory resident, execution proceeds normally.

Access to a page marked invalid causes a page fault trap. It is the result of the OS's failure to bring the desired page into memory.

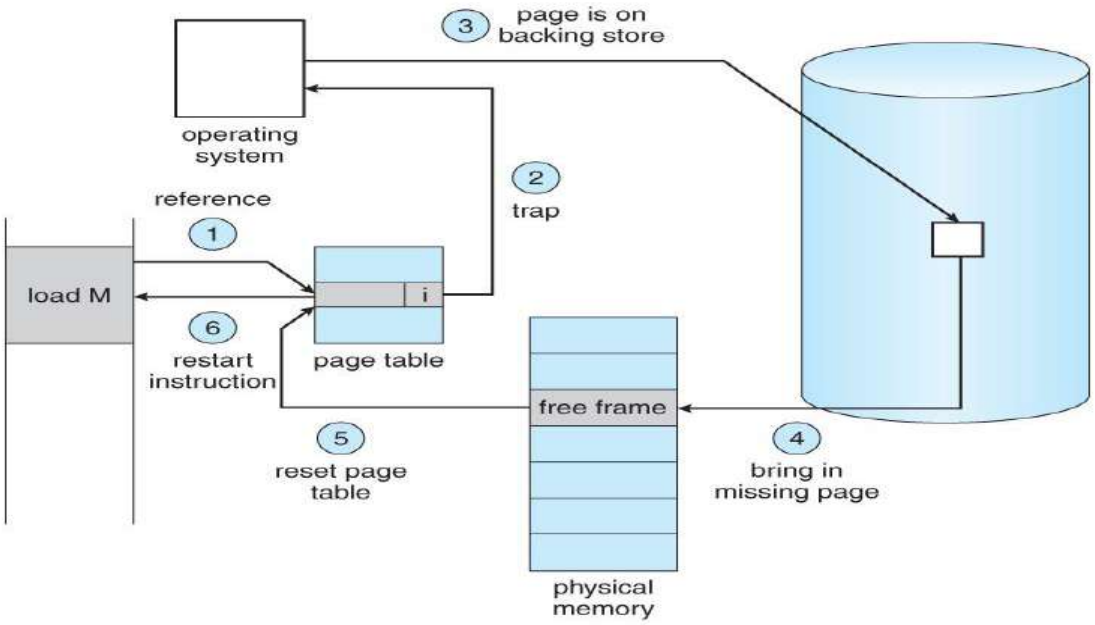
ROCEDURE TO HANDLE PAGE FAULT

- The computer hardware traps to the kernel and program counter (PC) is saved on the stack. Current instruction state information is saved in CPU registers.
- An assembly program is started to save the general registers and other volatile information to keep the OS from destroying it.
- Operating system finds that a page fault has occurred and tries to find out which virtual page is needed. Sometimes, hardware register contains this required information. If not, the operating system must retrieve PC, fetch instruction and find out what it was doing when the fault occurred.
- Once virtual address caused page fault is known, system checks to see if address is valid and checks if there is no protection access problem.
- If the virtual address is valid, the system checks to see if a page frame is free. If no frames are free, the page replacement algorithm is run to remove a page.
- If frame selected is dirty, page is scheduled for transfer to disk, context switch

takes place, fault process is suspended and another process is made to run until disk transfer is completed.

- As soon as page frame is clean, operating system looks up disk address where needed page is, schedules disk operation to bring it in.
- When disk interrupt indicates page has arrived, page tables are updated to reflect its position, and frame marked as being in normal state.
- Faulting instruction is backed up to state it had when it began and PC is reset. Faulting isscheduled, operating system returns to routine that called it.

Assembly Routine reloads register and other state information, returns to user space tocontinue execution.



DEVICE MANAGEMENT

Techniques for Device Management

- **Dedicated,**
- **shared and**
- **virtual.**

Device allocation considerations I/O traffic control & I/O Schedule, I/O Devicehandlers.

SPOOLING.

Device management in an operating system means controlling the Input/Output devices like disk, microphone, keyboard, printer, magnetic tape, USB ports, camcorder, scanner, other accessories, and supporting units like supporting units control channels.

The system has multiple devices, and in order to handle these physical or virtual devices, the operating system requires a separate program known as an ad device controller. It also determines whether the requested device is available.

The fundamentals of I/O devices may be divided into three categories:

1. **Boot Device**
2. **Character Device**
3. **Network Device**

- Boot Device

It stores data in fixed-size blocks, each with its unique address. For example- Disks.

- Character Device

It transmits or accepts a stream of characters, none of which can be addressed individually. For instance, keyboards, printers, etc.

- Network Device

It is used for transmitting the data packets.

Functions of the device management in the operating system

The operating system (OS) handles communication with the devices via their drivers. The OS component gives a uniform interface for accessing devices with various physical

features. There are various functions of device management in the operating system. Some of them are as follows:

1. It keeps track of data, status, location, uses, etc. The file system is a term used to define a group of facilities.
2. It enforces the pre-determined policies and decides which process receives the device when and for how long.
3. It improves the performance of specific devices.
4. It monitors the status of every device, including printers, storage drivers, and other devices.
5. It allocates and effectively deallocates the device. De-allocating differentiates the devices at two levels: first, when an I/O command is issued and temporarily freed. Second, when the job is completed, and the device is permanently release

Types of devices

There are three types of Operating system peripheral devices: dedicated, shared, and virtual. These are as follows:

1. Dedicated Device

In device management, some devices are allocated or assigned to only one task at a time until that job releases them. Devices such as plotters, printers, tape drivers, and other similar devices necessitate such an allocation mechanism because it will be inconvenient if multiple people share them simultaneously. The disadvantage of such devices is the inefficiency caused by allocating the device to a single user for the whole duration of task execution, even if the device is not used 100% of the time.

2. Shared Devices

These devices could be assigned to a variety of processes. By interleaving their requests, disk-DASD could be shared by multiple processes simultaneously. The Device Manager carefully controls the interleaving, and pre-determined policies must resolve all difficulties.

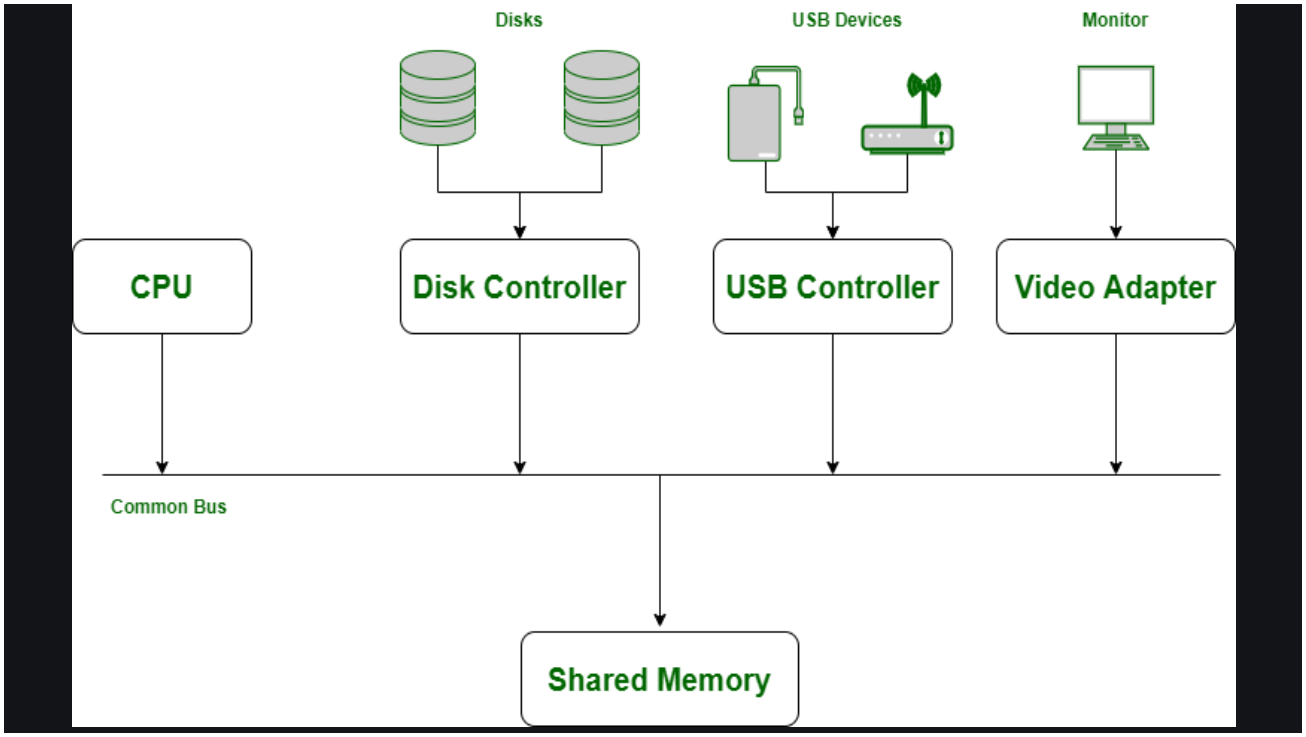
3. Virtual Devices

Virtual devices are a hybrid of the two devices, and they are dedicated devices that have been transformed into shared devices. For example, a printer can be transformed into a shareable device by using a spooling program that redirects all print requests to a disk. A print job is not sent directly to the printer; however, it is routed to the disk until it is fully prepared with all of the required sequences and formatting, at which point it is transmitted to the printers. The approach can transform a single printer into numerous virtual printers, improving performance and ease of use.

I/O scheduling in Operating Systems

Before understanding the I/O scheduling, It's important to get an overview of I/O operations.

- Operating System has a certain portion of code that is dedicated to managing Input/Output in order to improve the reliability and the performance of the system.
- A computer system contains CPUs and more than one device controllers connected to a common bus channel, generally referred to as the device driver.
- These device drivers provide an interface to I/O devices for communicating with the system hardware promoting ease of communication providing access to shared memory.



I/O Requests in operating systems :

I/O Requests are managed by Device Drivers in collaboration with some system programs inside the I/O device. The requests are served by OS using three simple segments :

- **I/O TRAFFIC CONTROLLER :** Keeps track of the status of all devices, control units, and communication channels.
- **I/O SCHEDULER :** Executes the policies used by OS to allocate and access the device, control units, and communication channels.
- **I/O DEVICE HANDLER :** Serves the device interrupts and heads the transfer of data.

I/O SCHEDULING IN OPERATING SYSTEMS :

Scheduling is used for efficient usage of computer resources avoiding deadlock and serving all processes waiting in the queue.

I/O Traffic Controller has 3 main tasks :

The primary task is to check if there's at least one path available.

If there exists more than one path, it must decide which one to select.

If all paths are occupied, its task is to analyze which path will be available at the earliest.

I/O Scheduler functions similar to Process scheduler, it allocates the devices, control units, and communication channels. However, under heavy load of I/O requests, Scheduler must decide what request should be served first and for that we multiple queues to be managed by OS.

The major difference between **Process scheduler**< and **I/O scheduler** is that I/O requests are not preempted: Once the channel program has started, it's allowed to continue to completion. Although it is feasible because programs are relatively short (50 to 100 ms). Some modern OS allows I/O Scheduler to serve and higher priority requests. In simpler words, If an I/O request has higher priority then they are served before other I/O requests with lower priority. I/O scheduler works in coordination

with the I/O traffic controller to keep track of which path is being served for the current I/O request.

I/O Device Handler manages the I/O interrupts (if any) and scheduling algorithms. Few I/O handling algorithms are :

- FCFS [First come first server].
- SSTF [Shortest seek time first].
- SCAN
- Look
- N-Step Scan
- C-SCAN
- C-LOOK

Every scheduling algorithm aims to minimize arm movement, mean response time, variance in response time. An overview of all I/O scheduling algorithms are described below :

First Come First Serve [FCFS]

It is one of the simplest device-scheduling algorithm since it is easy to program and essentially fair to users (I/O devices). The only barrier could be the high seek time, so any other algorithm that can surpass the minimum seek time is suitable for scheduling.

Shortest Seek Time First [SSTF] **:**

It uses the same ideology as the Shortest Job First in process scheduling, where shortest processes are served first and longer processes have to wait for their turn. Comparing the SJF concept in I/O scheduling, the request with track closest to one being served (The one with the shortest distance to travel on disk) is next to be satisfied. The main advantage over FCFS is that it minimizes overall seek time. It favors easy-to-reach requests and postpone traveling to those that are out of the way.

SCAN Algorithm :

SCAn uses a status flag which tells the direction of the arm, it tells whether the arm is moving towards the center of the disk or to the other side. This algorithm moves arm from the end of the disk to the center track servicing every request in its way. When it reaches innermost track, it reverses the direction and moves towards outer tracks on the disk, again servicing every request in its path.

LOOK [Elevator Algorithm] :
It's a variation of SCAN algorithm, here arm doesn't necessarily go all the way to either side on disk unless there are requests pending. It looks ahead for a request before servicing it. A big question arises is "Why should we use LOOK over SCAN?". The major advantage of LOOK over SCAN is that it discards the indefinite delay of I/O requests.

Other variations of SCAN :

N-Step Scan: It holds all the pending requests until the arm starts its way back. New requests are grouped for the next cycle of rotation.

C-SCAN [Circular SCAN] : It provides a uniform wait time as the arm serves requests on its way during the inward cycle. To know more, refer Difference between SCAN and C-SCAN.

C-LOOK [Optimized version of C-SCAN] : Arm doesn't necessarily return to the lowest-numbered track, it returns from the lowest request to be served. It optimized the C-SCAN as the arm doesn't move to the end of the disk if not required. To know more, refer to the Difference between C-LOOK and C-SCAN.

What is Spooling

Spooling is a process in which data is temporarily held to be used and executed by a device, program, or system. Data is sent to and stored in memory or other volatile storage until the program or computer requests it for execution.

SPOOL is an acronym for ***simultaneous peripheral operations online***. Generally, the spool is maintained on the computer's physical memory, buffers, or the I/O device-specific interrupts. The spool is processed in ascending order, working based on a FIFO (first-in, first-out) algorithm.

Spooling refers to putting data of various I/O jobs in a buffer. This buffer is a special area in memory or hard disk which is accessible to I/O devices. An operating system does the following activities related to the distributed environment:

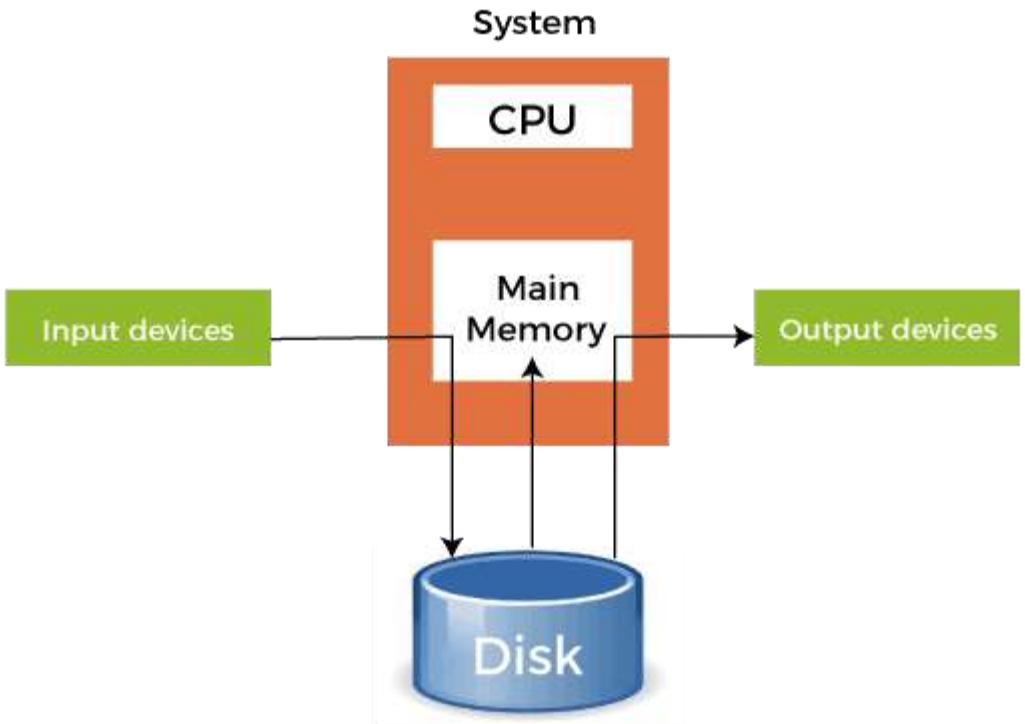
- Handles I/O device data spooling as devices have different data access rates.
- Maintains the spooling buffer, which provides a waiting station where data can rest while the slower device catches up.

- Maintains parallel computation because of the spooling process as a computer can perform I/O in parallel order. It becomes possible to have the computer read data from a tape, write data to disk, and write out to a tape printer while it is doing its computing task.

How Spooling Works in Operating System

In an operating system, spooling works in the following steps, such as:

1. Spooling involves creating a buffer called SPOOL, which is used to hold off jobs and data till the device in which the SPOOL is created is ready to make use and execute that job or operate on the data.
2. When a faster device sends data to a slower device to perform some operation, it uses any secondary memory attached as a SPOOL buffer. This data is kept in the SPOOL until the slower device is ready to operate on this data. When the slower device is ready, then the data in the SPOOL is loaded onto the main memory for the required operations.

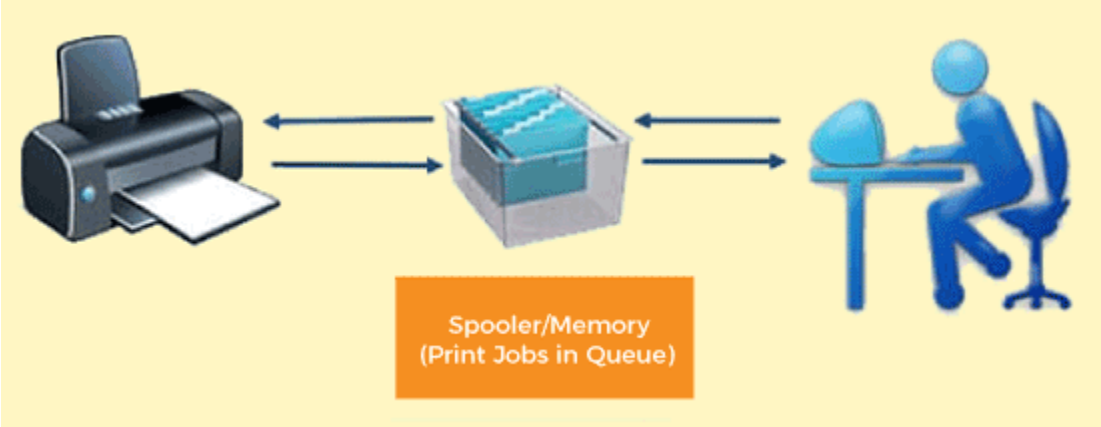


3. Spooling considers the entire secondary memory as a huge buffer that can store many jobs and data for many operations. The advantage of Spooling is that it can create a queue of jobs that execute in FIFO order to execute the jobs one by one.

4. A device can connect to many input devices, which may require some operation on their data. So, all of these input devices may put their data onto the secondary memory (SPOOL), which can then be executed one by one by the device. This will make sure that the CPU is not idle at any time. So, we can say that Spooling is a combination of buffering and queuing.
5. After the CPU generates some output, this output is first saved in the main memory. This output is transferred to the secondary memory from the main memory, and from there, the output is sent to the respective output devices.

Example of Spooling

The biggest example of Spooling is ***printing***. The documents which are to be printed are stored in the SPOOL and then added to the queue for printing. During this time, many processes can perform their operations and use the CPU without waiting while the printer executes the printing process on the documents one-by-one.



Many features can also be added to the Spooling printing process, like setting priorities or notification when the printing process has been completed or selecting the different types of paper to print on according to the user's choice.

Advantages of Spooling

Here are the following advantages of spooling in an operating system, such as:

- The number of I/O devices or operations does not matter. Many I/O devices can work together simultaneously without any interference or disruption to each other.
- In spooling, there is no interaction between the I/O devices and the CPU. That means there is no need for the CPU to wait for the I/O operations to take place.

Such operations take a long time to finish executing, so the CPU will not wait for them to finish.

- CPU in the idle state is not considered very efficient. Most protocols are created to utilize the CPU efficiently in the minimum amount of time. In spooling, the CPU is kept busy most of the time and only goes to the idle state when the queue is exhausted. So, all the tasks are added to the queue, and the CPU will finish all those tasks and then go into the idle state.
- It allows applications to run at the speed of the CPU while operating the I/O devices at their respective full speeds.

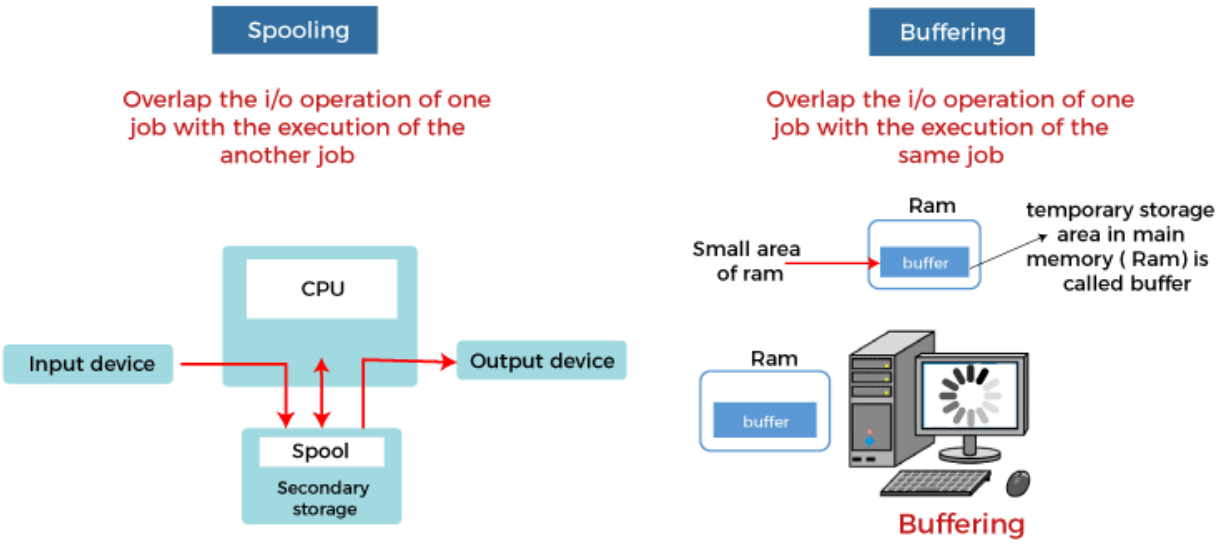
Disadvantages of Spooling

In an operating system, spooling has the following disadvantages, such as:

- Spooling requires a large amount of storage depending on the number of requests made by the input and the number of input devices connected.
- Because the SPOOL is created in the secondary storage, having many input devices working simultaneously may take up a lot of space on the secondary storage and thus increase disk traffic. This results in the disk getting slower and slower as the traffic increases more and more.
- Spooling is used for copying and executing data from a slower device to a faster device. The slower device creates a SPOOL to store the data to be operated upon in a queue, and the CPU works on it. This process in itself makes Spooling futile to use in real-time environments where we need real-time results from the CPU. This is because the input device is slower and thus produces its data at a slower pace while the CPU can operate faster, so it moves on to the next process in the queue. This is why the final result or output is produced at a later time instead of in real-time.

Difference between Spooling and Buffering

Spooling and buffering are the two ways by which I/O subsystems improve the performance and efficiency of the computer by using a storage space in the main memory or on the disk.



The basic difference between Spooling and Buffering is that Spooling overlaps the I/O of one job with the execution of another job. In comparison, the buffering overlaps the I/O of one job with the execution of the same job. Below are some more differences between Spooling and Buffering, such as:

Terms	Spooling	Buffering
Definition	Spooling, an acronym of Simultaneous Peripheral Operation Online (SPOOL), puts data into a temporary working area to be accessed and processed by another program or resource.	Buffering is an act of storing data temporarily in the buffer. It helps in matching the speed of the data stream between the sender and receiver.
Resource requirement	Spooling requires less resource management as different resources manage the process for specific jobs.	Buffering requires more resource management as the same resource manages the process of the same divided job.

Internal implementation	Spooling overlaps the input and output of one job with the computation of another job.	Buffering overlaps the input and output of one job with the computation of the same job.
Efficient	Spooling is more efficient than buffering.	Buffering is less efficient than spooling.
Processor	Spooling can also process data at remote sites. The spooler only has to notify when a process gets completed at the remote site to spool the next process to the remote side device.	Buffering does not support remote processing.
Size on memory	It considers the disk as a huge spool or buffer.	Buffer is a limited area in the main memory.

DEAD LOCKS

UNIT-5

5. DEAD LOCKS

5.1 Concept of deadlock.

5.2 System Model

5.3 Dead Lock Detection

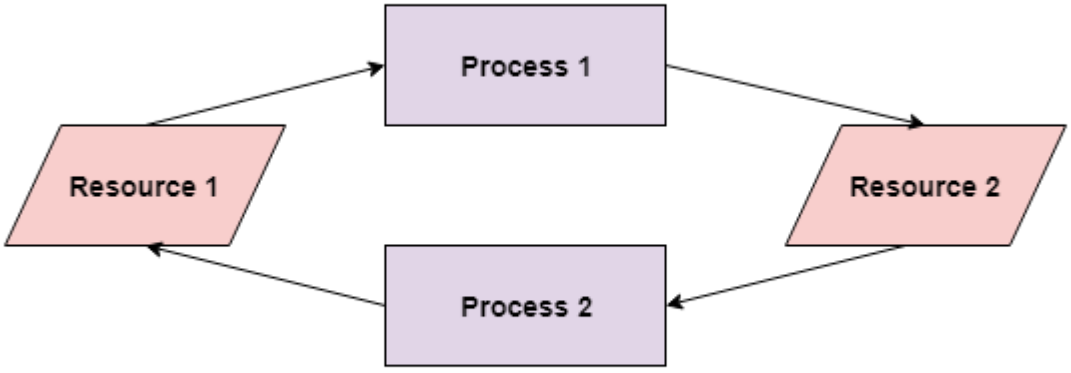
5.4 Resources allocation Graph

5.5 Methods of Deadlock handling

5.6 Recovery &Prevention, Explain Bankers Algorithm & Safety Algorithm

5.1 Concept of deadlock.

Deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.



Deadlock in Operating System

In the above diagram, the process 1 has resource 1 and needs to acquire resource 2. Similarly process 2 has resource 2 and needs to acquire resource 1. Process 1 and process 2 are in deadlock as each of them needs the other's resource to complete their execution but neither of them is willing to relinquish their resources.

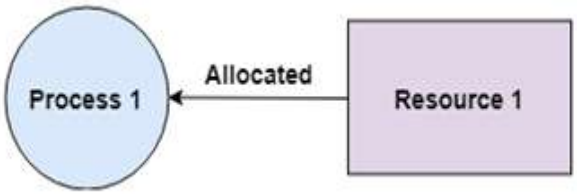
Coffman Conditions

A deadlock occurs if the four Coffman conditions hold true. But these conditions are not mutually exclusive.

The Coffman conditions are given as follows −

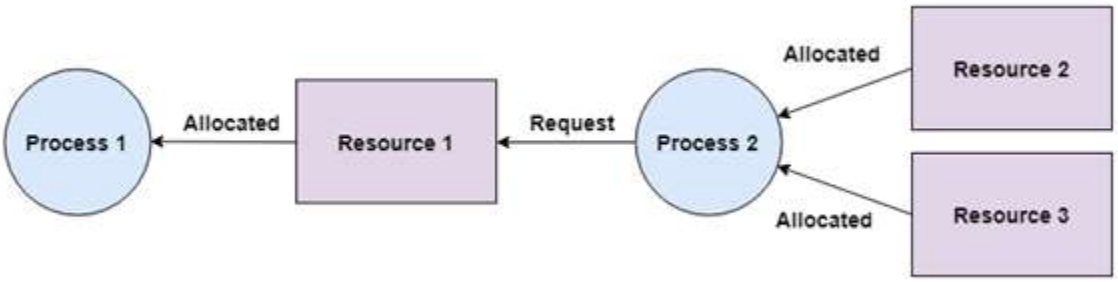
- **Mutual Exclusion**

  There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



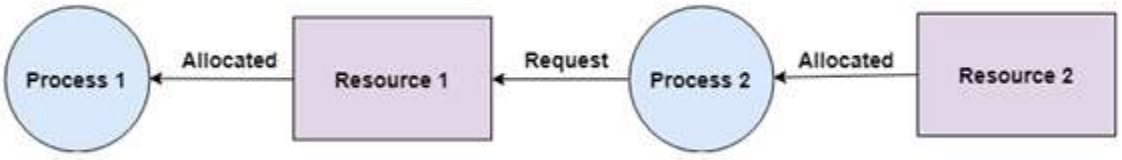
- **Hold and Wait**

  A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



- **No Preemption**

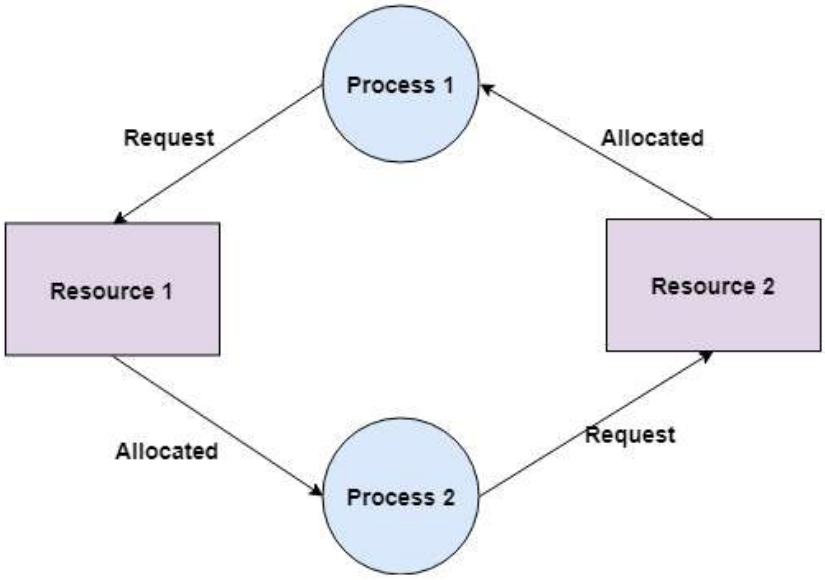
  A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



- **Circular Wait**

  A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly,

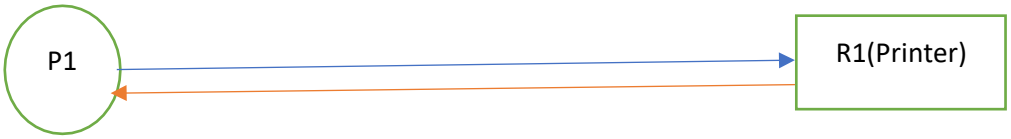
Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



- **5.2 SYSTEM MODEL**

- For the purposes of deadlock discussion, a system can be modeled as a collection of limited resources(r1, r2,r3,….rn) which can be partitioned into different categories, to be allocated to a number of processes(P1, P2 ,P3 ,...Pn) each having different needs.
- Resource categories may include memory, printers, CPUs, open files, tape drives, CD- ROMS, etc.
- By definition, all the resources within a category are equivalent, and a request of this category can be equally satisfied by any one of the resources in that category. If this is not the case (i.e. if there is some difference between the resources within a category), then that category needs to be further divided into separate categories. For example, "printers" may need to be separated into "laser printers" and "colour inkjet printers".
- Some categories may have a single resource.
- In normal operation a process must request a resource before using it, and release it when it is done, in the following sequence:
  1. ***Request*** - If the request cannot be immediately granted, then the process must wait until the resource(s) it needs become available. For example the system calls open( ) , malloc( ), new( ), and request( ).

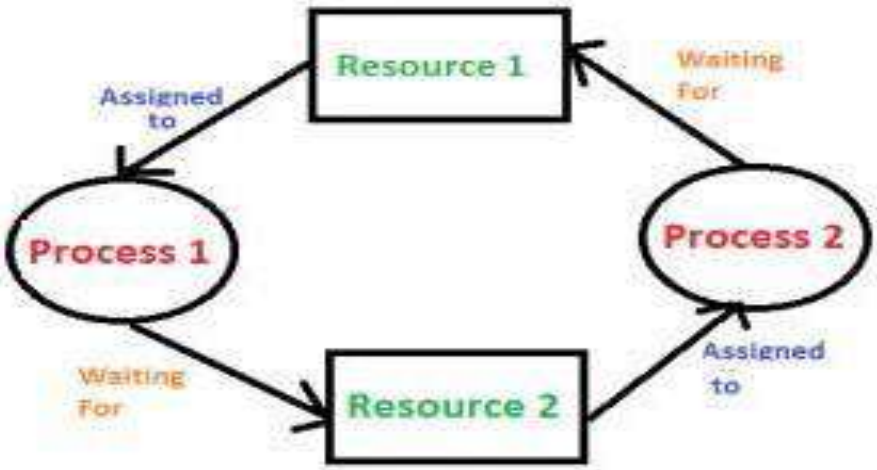
2. ***Use*** - The process uses the resource, e.g. *prints to the printer* or *reads from thefile.*
3. ***Release*** - The process relinquishes the resource. so that it becomes available for other processes. For example, close( ), free( ), delete( ), and release( ).



- ## 5.3 DEADLOCK DETECTION

A deadlock can be detected by a resource scheduler as it keeps track of all the resources that are allocated to different processes. After a deadlock is detected, it can be resolved using the following methods:

1. All the processes that are involved in the deadlock are terminated. This is not a good approach as all the progress made by the processes is destroyed.
2. Resources can be pre-empted from some processes and given to others till thedeadlock is resolved.
3. **If resources have single instance:**
4. In this case for Deadlock detection we can run an algorithm to check for cycle in the Resource Allocation Graph. Presence of cycle in the graph is the sufficient conditionfor deadlock.



In the above diagram, resource 1 and resource 2 have single instances. There is a cycle

R1 → P1 → R2 → P2. So, Deadlock is Confirmed.

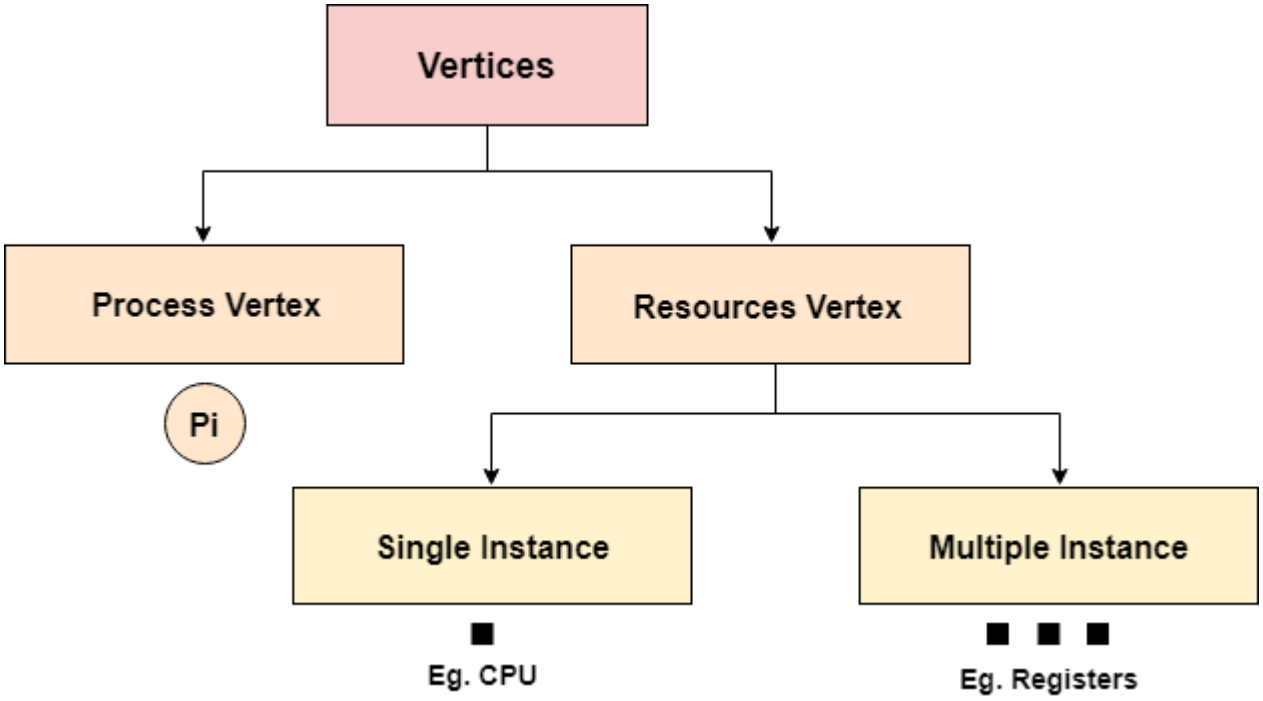
1. **If there are multiple instances of resources**:
   Detection of the cycle is necessary but not sufficient condition for deadlock detection, in this case, the system may or may not be in deadlock varies according to different situations.

• 5.4 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.

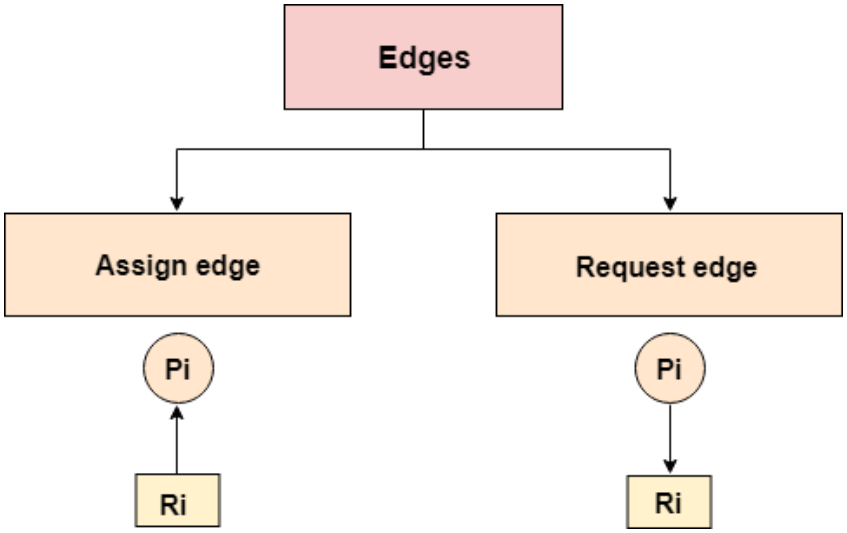
It also contains the information about all the instances of all the resources whether they are available or being used by the processes.

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.

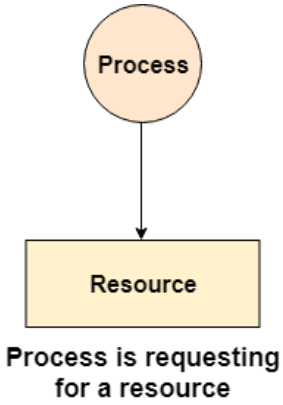
A resource can have more than one instance. Each instance will be represented by a dot inside the rectangle.



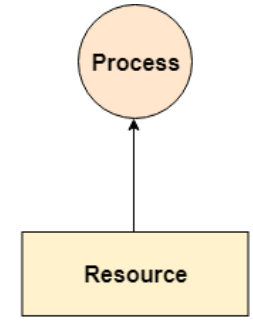
Edges in RAG are also of two types, one represents assignment and other represents the wait of a process for a resource. The above image shows each of them.

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.



Process is requesting for a resource



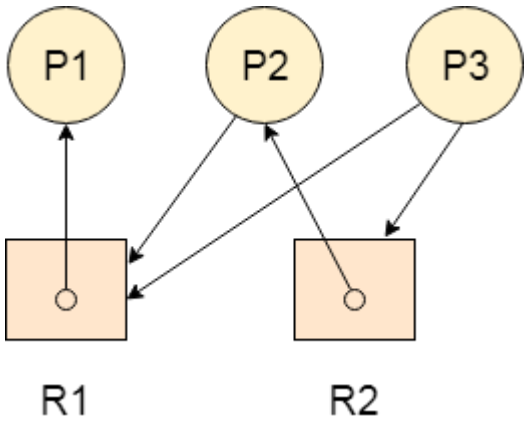
Resource is assigned to process

Example

Let'sconsider 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.



5.5 METHODS FOR 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 resourceswhen 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.

5.6 DEADLOCK PREVENTION

We can prevent Deadlock by eliminating any of the above four conditions.

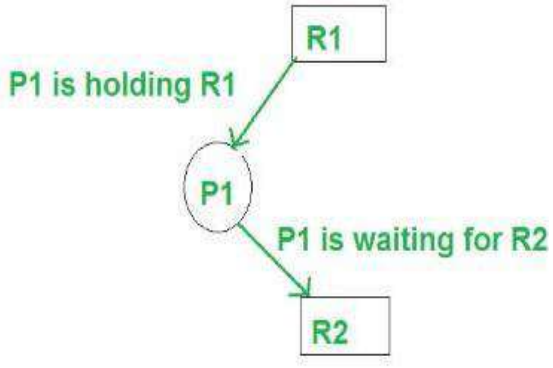
1. Eliminate Mutual Exclusion

It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.

2. Eliminate Hold and wait

Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization. for example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remain blocked till it has completed its execution.

The process will make a new request for resources after releasing the current set of resources.This solution may lead to starvation.



HOLD AND WAIT

3. **Eliminate No Pre-emption**

Pre-empt resources from the process when resources required by other high priorityprocesses.

4. **Eliminate Circular Wait**

Each resource will be assigned with a numerical number. A process can request the resources increasing/decreasing. order of numbering.

For Example, if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.

DEADLOCK AVOIDANCE

The general idea behind deadlock avoidance is to prevent deadlocks from ever happening, by preventing at least one of the aforementioned conditions.

- This requires more information about each process, and tends to lead to low device utilization. (i.e. it is a conservative approach.)
- In some algorithms the scheduler only needs to know the *maximum* number of each resource that a process might potentially use. In more complex algorithms the scheduler can also take advantage of the *schedule* of exactly what resources may be needed in what order.
- When a scheduler sees that starting a process or granting resource requests may lead to future deadlocks, then that process is just not started or the request is not granted.
- A resource allocation **state** is defined by the number of available and allocated resources, and the maximum requirements of all processes in the system.

Safe State- A state is **safe** if the system can allocate all resources requested by all processes (up to their stated maximums) without entering a deadlock state. If the system cannot fulfill the request of all processes then the state of the system is called **unsafe**.

More formally, a state is safe if there exists a **safe sequence(the order of process termination)** of processes { P0, P1, P2, ..., PN } such that all of the resource requests for Pi can be granted using the resources currently allocated to Pi and all processes Pj where j < i. (i.e. if all the processes prior to Pi finish and free up

their resources, then Pi will be able to finish also, using the resources that they have freed up.)

- If a safe sequence does not exist, then the system is in an unsafe state, which may lead to deadlock. (All safe states are deadlock free, but not all unsafe states lead to deadlocks.)

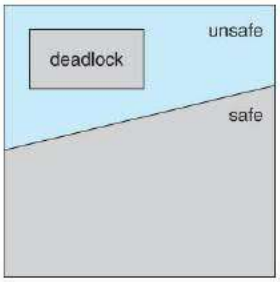


Figure - Safe, unsafe, and deadlocked state spaces.

NOTE: Deadlock prevention ensures that atleast one of the necessary conditions to cause deadlock will never occur while deadlock avoidance ensures that the system never enters anunsafe state.

BANKER'S ALGORITHM IN OPERATING SYSTEM

Banker's algorithm is a **deadlock avoidance algorithm**. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not.

Consider there are n account holders in a bank and the sum of the money in all of their accounts is S. Every time a loan has to be granted by the bank, it subtracts the **loan amount** from the **total money** the bank has. Then it checks if that difference is greater than S. It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once.

Banker's algorithm works in a similar way in computers.

Whenever a new process is created, it must specify the maximum instances of each resource type that it needs, exactly.

CHARACTERISTICS OF BANKER'S ALGORITHM

The characteristics of Banker's algorithm are as follows:

- If any process requests for a resource, then it has to wait.
- This algorithm consists of advanced features for maximum resource allocation.
- There are limited resources in the system we have.
- In this algorithm, if any process gets all the needed resources, then it is that it should return the resources in a restricted period.
- Various resources are maintained in this algorithm that can fulfill the needs of at least one client.

Let us assume that there are n processes and m resource types.

DATA STRUCTURES USED TO IMPLEMENT THE BANKER'S ALGORITHM

Some data structures that are used to implement the banker's algorithm are:

1. Available

It is an **array** of length m. It represents the number of available resources of each type. If Available[j] = k, then there are k instances available, of resource type Rj.

2. Max

It is an n x m matrix which represents the maximum number of instances of each resource that a process can request. If Max[i][j] = k, then the process Pi can request atmost k instances of resource type Rj.

3. Allocation

It is an n x m matrix which represents the number of resources of each type currently allocated to each process. If Allocation[i][j] = k, then process Pi is currently allocated k instances of resource type Rj.

4. Need

It is a two-dimensional array. It is an n x m matrix which indicates the remaining resource needs of each process. If Need[i][j] = k, then process Pi may need k more instances of resource type Rj to complete its task.

Need[i][j] = Max[i][j] - Allocation [i][j]

Banker's algorithm comprises of two algorithms:

1. Safety algorithm
2. Resource request algorithm

SAFETY ALGORITHM

A safety algorithm is an algorithm used to find whether or not a system is in its safe state. The algorithm is as follows:

1. Let `Work` and `Finish` be vectors of length **m** and **n**, respectively. Initially,
2. Work = Available
3. Finish[i] =false for i = 0, 1, ... , n - 1.

   This means, initially, no process has finished and the number of available resources is represented by the **Available** array.

4. Find an index **i** such that both
5. Finish[i] ==false
6. Needi <= Work

   If there is no such i present, then proceed to step 4.

   It means, we need to find an unfinished process whose needs can be satisfied by the available resources. If no such process exists, just go to step 4.

7. Perform the following:
8. Work = Work + Allocationi
9. Finish[i] = true

   Go to step 2.

When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other processes.

10. If `Finish[i] == true` for all i, then the system is in a safe state.

    That means if all processes are finished, then the system is in safe state.

This algorithm may require an order of **mxn² operations** in order to determine whether a state is safe or not.

RESOURCE REQUEST ALGORITHM

Now the next algorithm is a resource-request algorithm and it is mainly used to determine whether requests can be safely granted or not.

Let Requesti be the request vector for the process Pi. If Requesti[j]==k, then process Pi wants k instance of Resource type Rj.When a request for resources is made by the process Pi, the following are the actions that will be taken:

1. If Requesti <= Needi, then go to step 2;else raise an error condition, since the process has exceeded its maximum claim.

2.If Requesti <= Availablei then go to step 3; else Pi must have to wait as resources are not available.

3.Now we will assume that resources are assigned to process Pi and thus perform the following steps:

Available= Available-Requesti;

Allocationi=Allocationi +Requesti;

Needi =Needi - Requesti;

If the resulting resource allocation state comes out to be safe, then the transaction is completed and, process Pi is allocated its resources. But in this case, if the new state is unsafe, then Pi waits for Requesti, and the old resource-allocation state is restored.

Disadvantages of Banker's Algorithm

Some disadvantages of this algorithm are as follows:

1. During the time of Processing, this algorithm does not permit a process to change its maximum need.
2. Another disadvantage of this algorithm is that all the processes must know in advance about the maximum resource needs.
3. This algorithm permits the requests to be provided in constrained time, but for one year which is a fixed period.

DEADLOCK RECOVERY

In order to recover the system from deadlocks, either OS considers resources or processes.

FOR RESOURCE

Preempt the resource

We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner. Well, choosing a resource which will be snatched is going to be a bit difficult.

Rollback to a safe state

System passes through various states to get into the deadlock state. The operating system canrollback the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.

The moment, we get into deadlock, we will rollback all the allocations to get into the previous safe state.

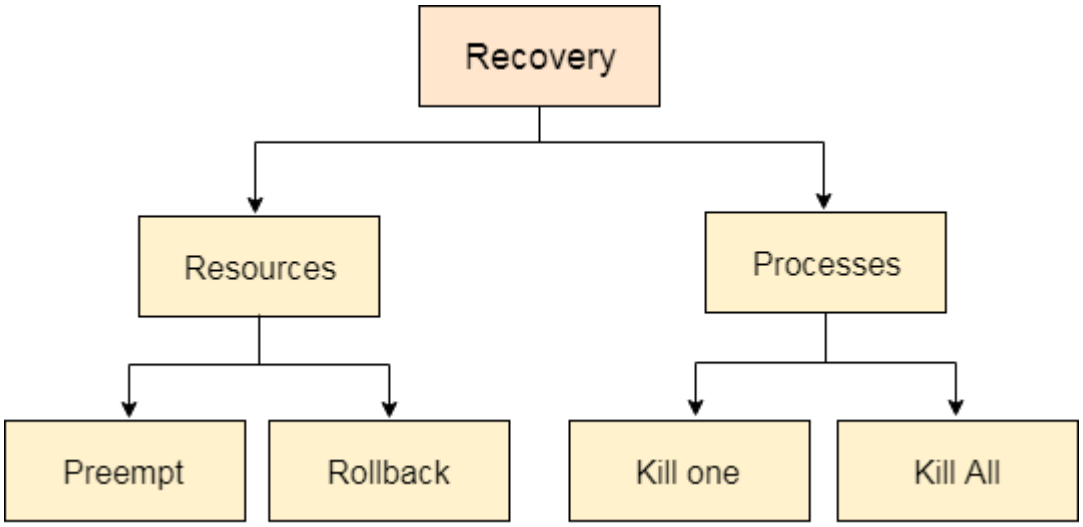
FOR PROCESS

Kill a process

Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.

Kill all process

This is not a suggestible approach but can be implemented if the problem becomes very serious. Killing all process will lead to inefficiency in the system because all the processes will execute again from starting.



FILE MANAGEMENT
UNIT-6

6. FILE MANAGEMENT
6.1 File organization, Directory & file structure, sharing of files
6.2 File access methods, file systems, reliability
6.3 Allocation of disk space
6.4 File protection, secondary storage management.

6. FILE MANAGEMENT

le management is one of the basic and important features of operating system. Operating system is used to manage files of computer system. All the files with different extensions are managed by operating system.

A file is collection of specific information stored in the memory of computer system. File management is defined as the process of manipulating files in computer system, it management includes the process of creating, modifying and deleting the files.

The following are some of the tasks performed by file management of operating system of any computer system:

1. It helps to create new files in computer system and placing them at the specific locations.
2. It helps in easily and quickly locating these files in computer system.
3. It makes the process of sharing of the files among different users very easy and user friendly.
4. It helps to stores the files in separate folders known as directories. These directories help users to search file quickly or to manage the files according to their types or uses.
5. It helps the user to modify the data of files or to modify the name of the file in the directories.

6.1 FILE ORGANIZATION, DIRECTORY & FILE STRUCTURE, SHARING OF FILES

FILE- A file is a collection of related information that is recorded on secondary storage. From user's point of view, a file is the smallest allotment of logical data in secondary storage.

Or file is a collection of logically related entities.

FILE ORGANIZATION refers to the way data is stored in a file. File organization is very important because it determines the methods of access, efficiency, flexibility

and storage devices to use. Or we can say, File organization is the "logical structuring'' as well as the access method(s) of files.

Common file organization schemes are:

- sequential,
- random,
- serial and
- indexed-sequential

FILE DIRECTORIES:

Collection of files is a file directory. The directory contains information about the files, including attributes, location and ownership. Much of this information, especially that is concerned with storage, is managed by the operating system. The directory is itself a file, accessible by various file management routines.

Information contained in a device directory are:

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed
- Date last updated
- Owner id
- Protection information

Operation performed on directory are:

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Advantages of maintaining directories are:

- **Efficiency:** A file can be located more quickly.
- **Naming:** It becomes convenient for users as two users can have same name for different files or may have different name for same file.

Grouping: Logical grouping of files can be done by properties e.g. all java programs, all

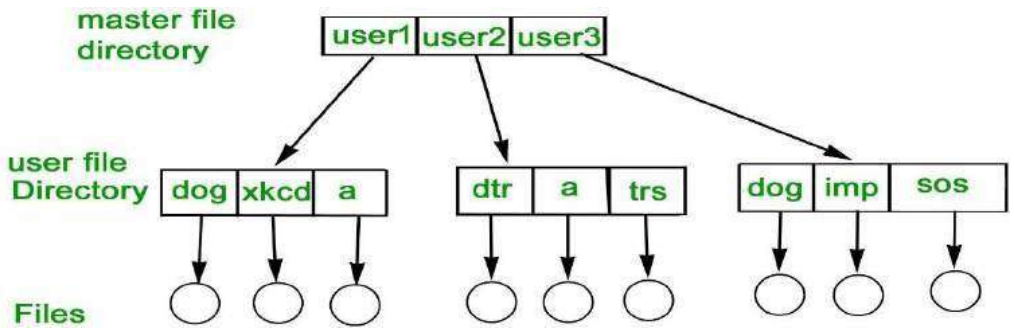
1. **SINGLE-LEVEL DIRECTORY**
   In this a single directory is maintained for all the users.

- **Naming problem:** Users cannot have same name for two files.
- **Grouping problem:** Users cannot group files according to their need.

2. TWO-LEVEL DIRECTORY

In this, separate directories for each user is maintained.

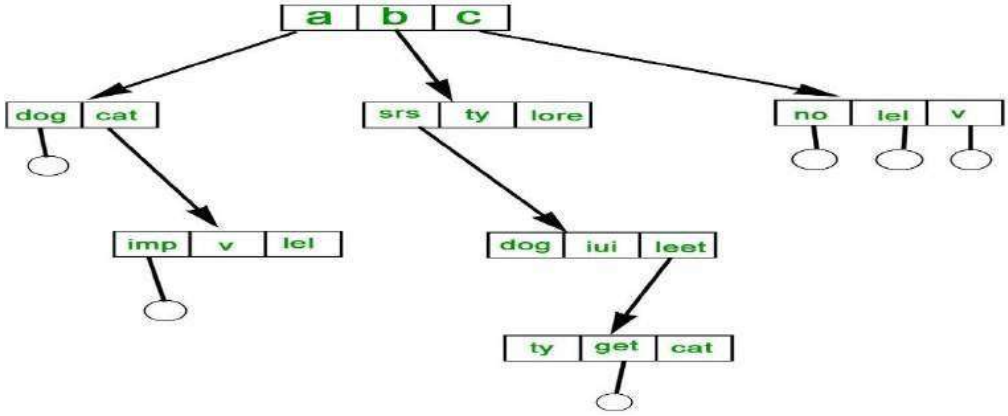
- **Path name:** Due to two levels there is a path name for every file to locate that file.
- Now, we can have same file name for different user.
- Searching is efficient in this method.



3. TREE-STRUCTURED DIRECTORY:

Directory is maintained in the form of a tree. Searching is efficient and also there is groupingcapability. We have absolute or relative path name for a file.

When operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.



FILE STRUCTURE

A File Structure should be according to a required format that the operating system canunderstand.

- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.
- An object file is a sequence of bytes organized into blocks that are understandable bythe machine.
- When operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.

FILE SHARING

Allowing users to share files raises a major issue: protection. A general approac h is to provide controlled access to files through a set of operations such as read, write, delete, list, and append. Then permit users to perform one or more operations.

One popular protection mechanism is a condensed version of access list, where the syst emrecognizes three classifications of users with each file and directory:

- user
- group
- other

6.2 FILE ACCESS METHODS, FILE SYSTEMS, RELIABILITY

FILE ACCESS METHODS

File access methods refers to the manner in which the records of a file may be accessed.

There are several ways to access files −

- Sequential access
- Direct/Random access
- Indexed sequential access

SEQUENTIAL ACCESS

- A sequential access is that in which the records are accessed in some sequence, i.e.,the information in the file is processed in order, one record after the other.
- This access method is the most primitive one.
- Example: **Compilers** usually access files in this fashion.

DIRECT/RANDOM ACCESS

- Records are stored randomly but accessed directly.
  Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing
- The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.
- **Magnetic** and **optical** disks allow data to be stored and accessed randomly.

INDEXED SEQUENTIAL ACCESS

- This mechanism is built up on base of sequential access.
- An index is created for each file which contains pointers to various blocks.
- Index is searched sequentially and its pointer is used to access the file directly.
- For example, on a **magnetic drum**, records are stored sequential on the tracks. However, each record is assigned an index that can be used to access it directly.

RELIABILITY

Regular file operations often involve changing several disk blocks. For example, Whatcan happen if disk loses power or machine software crashes?

- Some operations in progress may complete
- Some operations in progress may be lost
- Overwrite of a block may only partially complete

what if the computer crashes between two of these operations? Then the filesystem

could enter an *inconsistent state*, confusing the OS to the point that the filesystem can't be used at all. In order to make file system reliable we need to keep Backup/restore (disaster scenarios) and have a consistent check on File system (i.e., maintain consistency) (e.g., UNIX fsck).

File system wants durability

– Data previously stored can be retrieved (maybe after some recovery step), regardless offailure.

6.3 ALLOCATION OF DISK SPACE

File Allocation Methods

There are different kinds of methods that are used to allocate disk space. We must select the best method for the file allocation because it will directly affect the system performance and system efficiency. With the help of the allocation method, we can utilize the disk, and also files can be accessed.

There are various types of file allocations method:

1. Contiguous allocation
2. Extents
3. Linked allocation
4. Clustering
5. FAT
6. Indexed allocation
7. Linked Indexed allocation
8. Multilevel Indexed allocation
9. Inode

There are different types of file allocation methods, but we mainly use three types of file allocation methods:

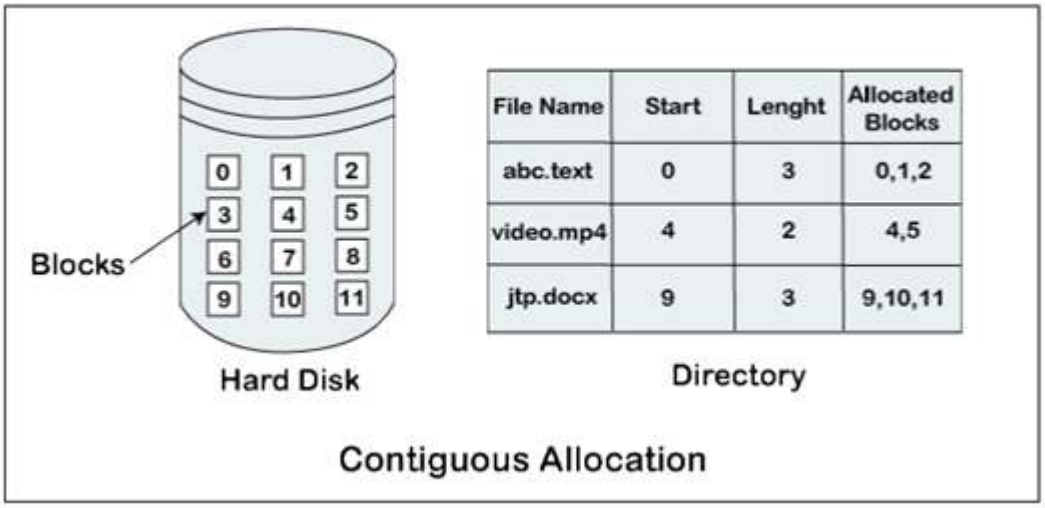
1. Contiguous allocation
2. Linked list allocation
3. Indexed allocation

These methods provide quick access to the file blocks and also the utilization of disk space in an efficient manner.

CONTIGUOUS ALLOCATION:

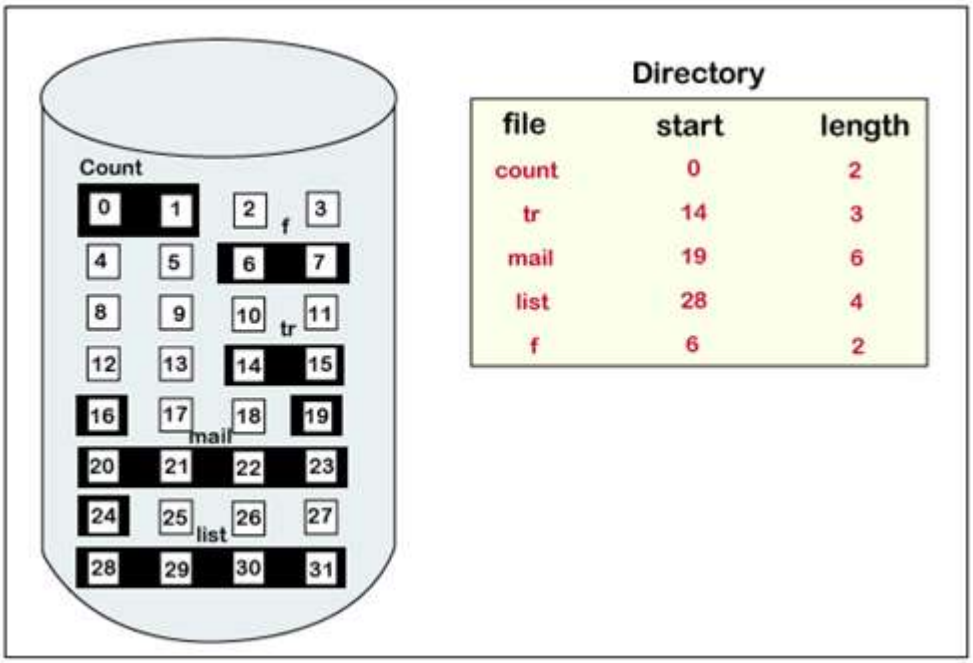
Contiguous allocation is one of the most used methods for allocation. Contiguous allocation means we allocate the block in such a manner, so that in the hard disk, all the blocks get the contiguous physical block.

We can see in the below figure that in the directory, we have three files. In the table, we have mentioned the starting block and the length of all the files. We can see in the table that for each file, we allocate a contiguous block.



Example of contiguous allocation

We can see in the given diagram, that there is a file. The name of the file is 'mail.' The file starts from the 19th block and the length of the file is 6. So, the file occupies 6 blocks in a contiguous manner. Thus, it will hold blocks 19, 20, 21, 22, 23, 24.



Advantages of Contiguous Allocation

The advantages of contiguous allocation are:

1. The contiguous allocation method gives excellent read performance.
2. Contiguous allocation is easy to implement.
3. The contiguous allocation method supports both types of file access methods that are sequential access and direct access.
4. The Contiguous allocation method is fast because, in this method number of seeks is less due to the contiguous allocation of file blocks.

Disadvantages of Contiguous allocation

The disadvantages of contiguous allocation method are:

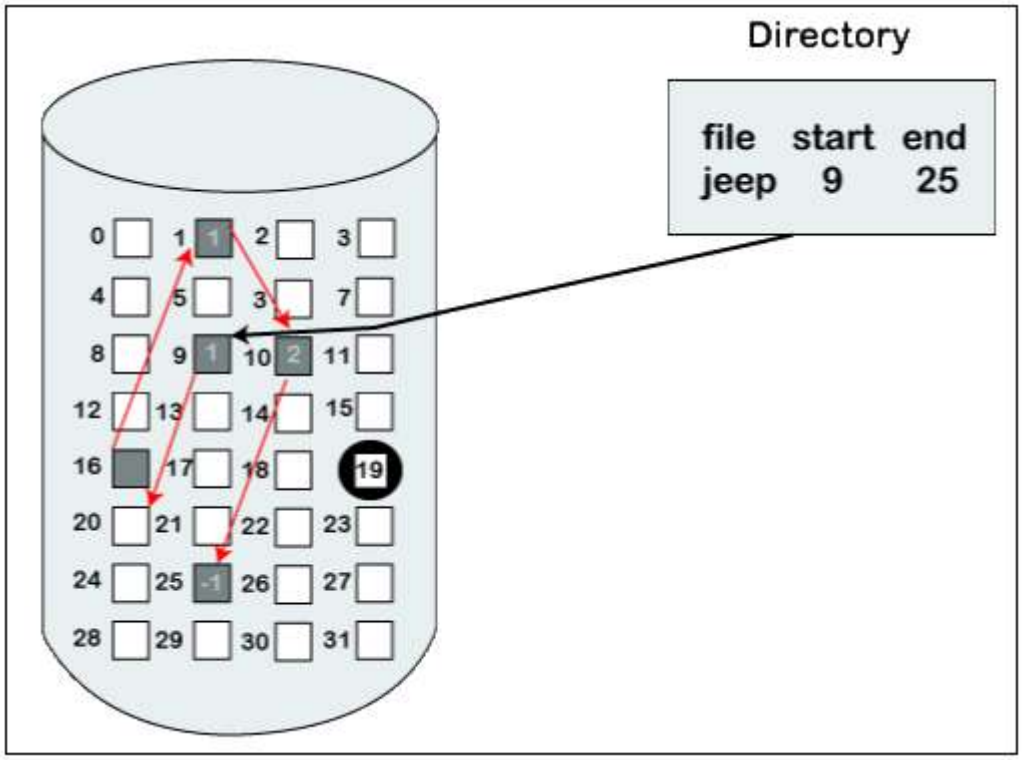
1. In the contiguous allocation method, sometimes disk can be fragmented.
2. In this method, it is difficult to increase the size of the file due to the availability of the contiguous memory block.

LINKED LIST ALLOCATION

The linked list allocation method overcomes the drawbacks of the contiguous allocation method. In this file allocation method, each file is treated as a linked list of disks blocks. In the linked list allocation method, it is not required that disk blocks assigned to a specific file are in the contiguous order on the disk. The directory entry comprises of a pointer for starting file block and also for the ending file block. Each disk block that is allocated or assigned to a file consists of a pointer, and that pointer point the next block of the disk, which is allocated to the same file.

Example of linked list allocation

We can see in the below figure that we have a file named 'jeep.' The value of the start is 9. So, we have to start the allocation from the 9^{th} block, and blocks are allocated in a random manner. The value of the end is 25. It means the allocation is finished on the 25^{th} block. We can see in the below figure that the block (25) comprised of -1, which means a null pointer, and it will not point to another block.



Advantages of Linked list allocation

There are various advantages of linked list allocation:

1. In liked list allocation, there is no external fragmentation. Due to this, we can utilize the memory better.
2. In linked list allocation, a directory entry only comprises of the starting block address.
3. The linked allocation method is flexible because we can quickly increase the size of the file because, in this to allocate a file, we do not require a chunk of memory in a contiguous form.

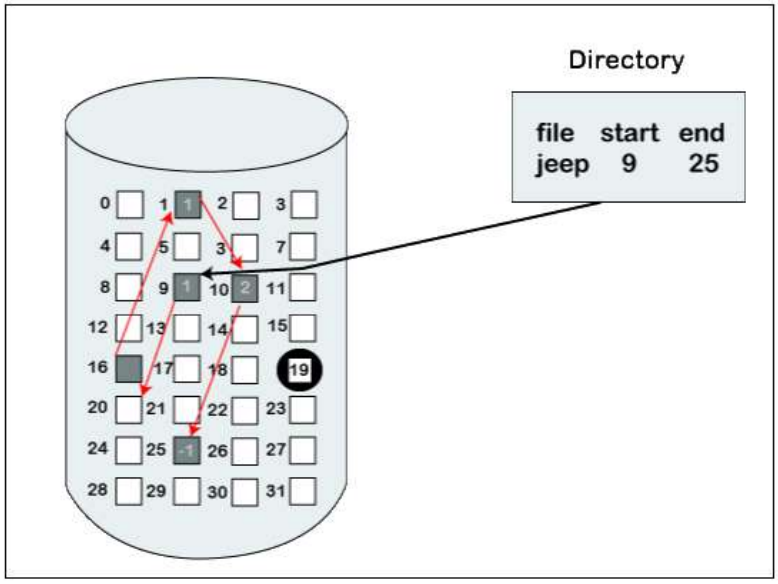
Disadvantages of Linked list Allocation

There are various disadvantages of linked list allocation:

1. Linked list allocation does not support direct access or random access.
2. In linked list allocation, we need to traverse each block.
3. If the pointer in the linked list break in linked list allocation, then the file gets corrupted.
4. In the disk block for the pointer, it needs some extra space.

INDEXED ALLOCATION

The Indexed allocation method is another method that is used for file allocation. In the index allocation method, we have an additional block, and that block is known as the index block. For each file, there is an individual index block. In the index block, the ith entry holds the disk address of the ith file block. We can see in the below figure that the directory entry comprises of the address of the index block.



Advantages of Index Allocation

The advantages of index allocation are:

1. The index allocation method solves the problem of external fragmentation.
2. Index allocation provides direct access.

Disadvantages of Index Allocation

The disadvantages of index allocation are:

1. In index allocation, pointer overhead is more.
2. We can lose the entire file if an index block is not correct.
3. It is totally a wastage to create an index for a small file.

6.4 FILE PROTECTION, SECONDARY STORAGE MANAGEMENT.

FILE PROTECTION

When information is kept in a computer system, a major concern is its protection from physical damage (reliability) as well as improper access.

Types of access: In case of systems that don't permit access to the files of other users. Protection is not needed. So, one extreme is to provide protection by prohibiting access. The other extreme is to provide free access with no protection. Both these approaches are too extreme for general use.

So, we need controlled access. It is provided by limiting the types of file access. Access is permitted depending on several factors. One major factor is type of access requested. The different type of operations that can be controlled are:

• Read

• Write

• Execute

• Append

• Delete

• List

Access lists and groups:

Various users may need different types of access to a file or directory. So, we can associate an access lists with each file and directory to implement identity dependent access. When a user access requests access to a particular file, the OS checks the access list associated with that file. If that user is granted the requested access, then the access is allowed. Otherwise, a protection violation occurs & the user is denied access to the file. But the main problem with access lists is their length. It isvery tedious to construct such a list. So, we use a condensed version of the access list by classifying

the users into 3 categories:

• Owners: The user who created the file.

• Group: A set of users who are sharing the files.

• Others: All other users in the system.

Here only 3 fields are required to define protection. Each field is a collection of bits each of which

either allows or prevents the access. E.g. The UNIX file system defines 3 fields of 3 bits each: rwx

• r(read access)

- w(write access)
- x(execute access)

Separate fields are kept for file owners, group & other users. So, a bit is needed to record protection information for each file.

NEED FOR PROTECTING FILES

- Need for protection is due to the ability to access files
- Two ways to tackle the problem
    1. Prohibit access providing complete protection
    2. Provide free access without protection
- Both approaches are too extreme for general use.

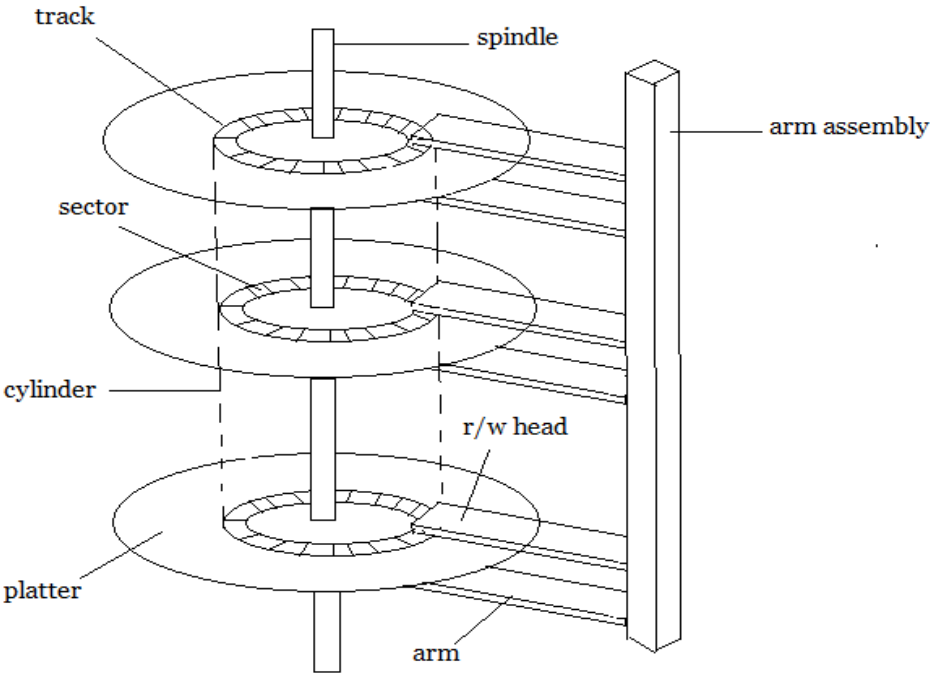
SECONDARY STORAGE AND DISK SCHEDULING ALGORITHMS

Secondary storage devices are those devices whose memory is non volatile, meaning, the stored data will be intact even if the system is turned off. Here are a few things worth noting about secondary storage.

- Secondary storage is also called auxiliary storage.
- Secondary storage is less expensive when compared to primary memory like RAMs.
- The speed of the secondary storage is also lesser than that of primary storage.
- Hence, the data which is less frequently accessed is kept in the secondary storage.
- A few examples are magnetic disks, magnetic tapes, removable thumb drives etc.

MAGNETIC DISK STRUCTURE

In modern computers, most of the secondary storage is in the form of magnetic disks. Hence, knowing the structure of a magnetic disk is necessary to understand how the data in the disk is accessed by the computer.



Structure of a magnetic disk

A magnetic disk contains several **platters**. Each platter is divided into circular shaped **tracks**. The length of the tracks near the centre is less than the length of the tracks farther from the centre. Each track is further divided into **sectors**, as shown in the figure.

Tracks of the same distance from centre form a cylinder. A read-write head is used to read data from a sector of the magnetic disk.

The speed of the disk is measured as two parts:

- **Transfer rate:** This is the rate at which the data moves from disk to the computer.
- **Random access time:** It is the sum of the seek time and rotational latency.

Seek time is the time taken by the arm to move to the required track. **Rotational latency** is defined as the time taken by the arm to reach the required sector in the track.

Even though the disk is arranged as sectors and tracks physically, the data is logically arranged and addressed as an array of blocks of fixed size. The size of a block can be **512** or **1024** bytes. Each logical block is mapped with a sector on the disk, sequentially. In this way, each sector in the disk will have a logical address.

SYSTEM PROGRAMMING

UNIT-7

7.0 system programming
7.1 concepts of system programming and show difference from application complier:
7.2 compiler , functions of compiler.
7.3 compare compiler and interpreter.
7.4 seven phases of compiler, brief description of each phase

7.1 CONCEPTS OF SYSTEM PROGRAMMING

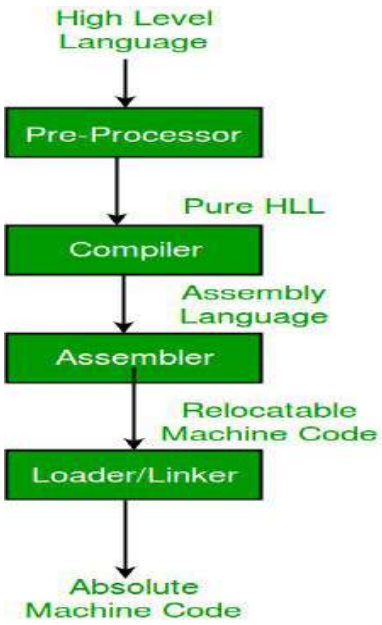
DEFINITION

System programming (or systems programming) is the activity of programming system software. The primary distinguishing characteristic of systems programming when compared to application programming is that application programming aims at producing software which provides services to the user (e.g. word processor, Spreadsheets, databases, Accounting packages)

Systems programming aims at producing software which provides services to the computer hardware (e.g. disk defragmenter, Operating Systems…). It also requires a greater degree of hardware awareness that is to say it is machine dependent and the programmer needs to know the hardware within which the software will operate.

Language processing systems (using Compiler) –

We know a computer is a logical assembly of Software and Hardware. The hardware knows a language, that is hard for us to grasp, consequently, we tend to write programs in a high-level language, that is much less complicated for us to comprehend and maintain in thoughts. Now, these programs go through a series of transformations so that they can readily be used by machines. This is where language procedure systems come in handy.



- **High-Level Language** – If a program contains #define or #include directives such as #include or #define it is called HLL. They are closer to humans but far from machines. These (#) tags are called preprocessor directives. They direct the pre-processor about what to do.
- **Pre-Processor** – The pre-processor removes all the #include directives by including the files called file inclusion and all the #define directives using macro expansion. It performs file inclusion, augmentation, macro-processing, etc.
- **Assembly Language** – It's neither in binary form nor high level. It is an intermediate state that is a combination of machine instructions and some other useful data needed for execution.
- **Assembler** – For every platform (Hardware + OS) we will have an assembler. They are not universal since for each platform we have one. The output of the assembler is called an object file. Its translates assembly language to machine code.
- **Interpreter** – An interpreter converts high-level language into low-level machine language, just like a compiler. But they are different in the way they read the input. The Compiler in one go reads the inputs, does the processing, and executes the source code whereas the interpreter does the same line by line. Compiler scans the entire program and translates it as a whole into machine code whereas an interpreter translates the program one statement at a time. Interpreted programs are usually slower with respect to compiled ones.

- **Relocatable Machine Code** – It can be loaded at any point and can be run. The address within the program will be in such a way that it will cooperate with the program movement.
- **Loader/Linker** – It converts the relocatable code into absolute code and tries to run the program resulting in a running program or an error message (or sometimes both can happen). Linker loads a variety of object files into a single file to make it executable. Then loader loads it in memory and executes it.

DIFFERENCES BETWEEN SYSTEM PROGRAMMING AND APPLICATION COMPILER:

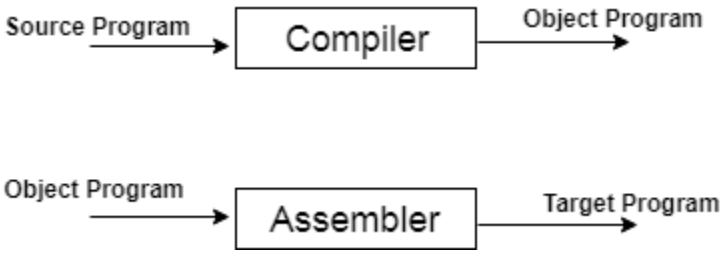
SYSTEM PROGRAMMING	APPLICATION COMPILER
1. System programming aims at producing software and software platforms(such as OS or web browsers or application programming interface) which provides services to others platform.	1. A compiler like all applications relies on the OS to load it into memory and start its execution.
2. System programming provides an environment where program can be developed and executed.	2. Application compiler is used to convert high level language to lower level language or machine code.
3. examples of system software includes os, computational sciences s/w, device driver, assembler, interpreter, compiler etc.	3. Examples of application compiler are C, C++, Java etc.

7.2 COMPILER
FUNCTIONS OF COMPILER.

INTRODUCTION TO COMPILER

- A compiler is a translator that converts the high-level language into the machine language.
- High-level language is written by a developer and machine language can be understood by the processor.
- Compiler is used to show errors to the programmer.

- The main purpose of compiler is to change the code written in one language without changing the meaning of the program.
- When you execute a program which is written in HLL programming language then it executes into two parts.
- In the first part, the source program compiled and translated into the object program (low level language).
- In the second part, object program translated into the target program through the assembler.



FUNCTIONS OF COMPILER:

i. Detects errors during translation and gives suggestions on how to correct them.

ii. Links the subroutines (or a function, procedure, and subprogram, is code that may be called and executed anywhere in a program) usedin program instructions.

iii. Converts a source code written using high level programming languageinto machine code.

iv. Generates executable (.exe) object code file.

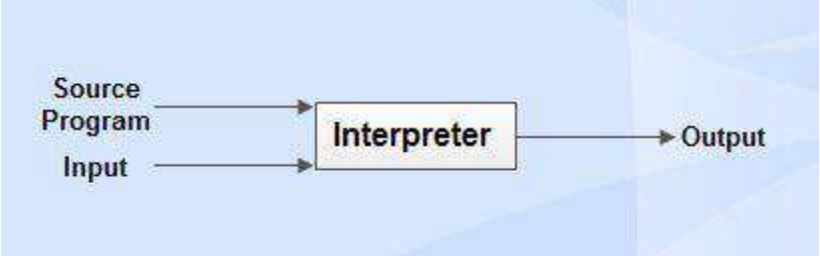
7.3 COMPARE COMPILER AND INTERPRETER.

Interpreter

Interpreter is a translator which is used to convert programs in high-level language to low-level language. Interpreter translates line by line and reports the error once it encountered during the translation process.

It directly executes the operations specified in the source program when the input is given by the user.

It gives better error diagnostics than a compiler.



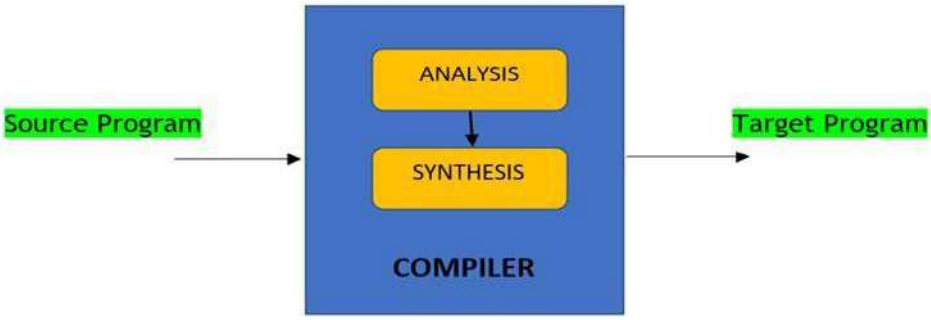
Compiler	Interpreter
A compiler translates the entire source code in a single run.	An interpreter translates the entire source code line by line.
It consumes less time i.e., it is faster than an interpreter.	It consumes much more time than the compiler i.e., it is slower than the compiler.
It is more efficient.	It is less efficient.
CPU utilization is more.	CPU utilization is less as compared to the compiler.
Both syntactic and semantic errors can be checked simultaneously.	Only syntactic errors are checked.
The compiler is larger.	Interpreters are often smaller than compilers.
It is not flexible.	It is flexible.
The localization of errors is difficult.	The localization of error is easier than the compiler.

Compiler	Interpreter
A presence of an error can cause the whole program to be re-organized.	A presence of an error causes only a part of the program to be re-organized.
The compiler is used by the language such as C, C++.	An interpreter is used by languages such as Java.

7.4 SEVEN PHASES OF COMPILER, BRIEF DESCRIPTION OF EACH PHASE

PHASES OF COMPILER:

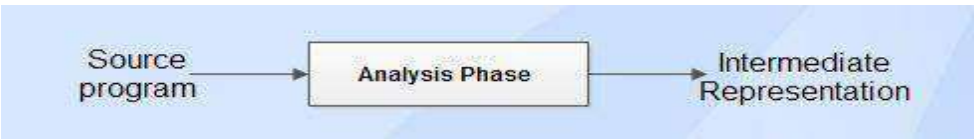
Compiler operates in various phases each phase transforms the source program from one representation to another. Every phase takes inputs from its previous stage and feeds its output to the next phase of the compiler.



The structure of compiler consists of two parts:

1. *Analysis phase*

- Analysis part breaks the source program into constituent pieces and imposes a grammatical structure on them which further uses this structure to create an intermediate representation of the source program.
- It is also termed as front end of compiler.
- Information about the source program is collected and stored in a data



- structure called symbol table.

Fig. 2.0. Diagram of Analysis phase

2. Synthesis phase

•Synthesis part takes the intermediate representation as input and transforms it tothe target program.

- It is also termed as back end of compiler.

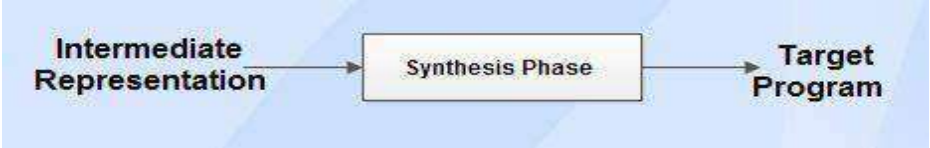


Fig. 3.0. Diagram of Synthesis phase

The design of compiler can be decomposed into several phases, each of which converts one form of source program into another

The different phases of compiler are as follows:

1. Lexical analysis
2. Syntax analysis
3. Semantic analysis
4. Intermediate code generation
5. Code optimization
6. Code generation

All of the aforementioned phases involve the following tasks:

- Symbol table management.
- Error handling.

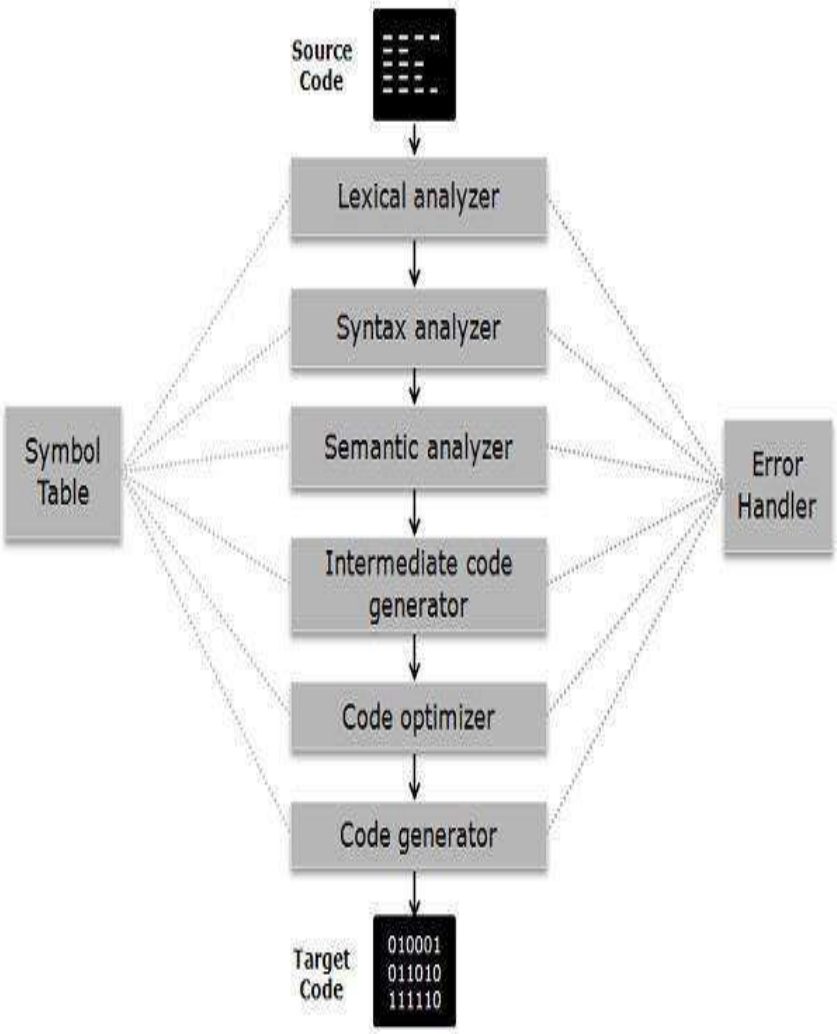


Fig. 4.0. Block diagram of phases of compiler

1. LEXICAL ANALYSIS

- Lexical analysis is the first phase of compiler which is also termed as scanning.

  •Source program is scanned to read the stream of characters and those characters are grouped to form a sequence called lexemes which produces token as output.

- **Token:** Token is a sequence of characters that represent lexical unit, which matches with the pattern, such as keywords, operators, identifiers etc.

- **Lexeme:** Lexeme is instance of a token i.e., group of characters forming a token.

,

- **Pattern:** Pattern describes the rule that the lexemes of a token takes. It is thestructure that must be matched by strings.

- Once a token is generated the corresponding entry is made in the symbol table.

Input: stream of charactersOutput: Token

Token Template: <token-name, attribute-value>

(eg.) c=a+b*5;

Lexemes and tokens

Lexemes	Tokens
c	Identifier
=	assignment symbol
a	Identifier
+	+ (addition symbol)
b	Identifier
*	* (multiplication symbol)
5	5 (number)

Hence, <id, 1><=>< id, 2>< +><id, 3 >< * >< 5>

2. SYNTAX ANALYSIS

- Syntax analysis is the second phase of compiler which is also called as parsing.

- Parser converts the tokens produced by lexical analyzer into a tree like representation called parse tree.

- A parse tree describes the syntactic structure of the input.

•Syntax tree is a compressed representation of the parse tree in which the operators appear as interior nodes and the operands of the operator are the children of the node for that operator.

Input: *Tokens*

Output: *Syntax tree*

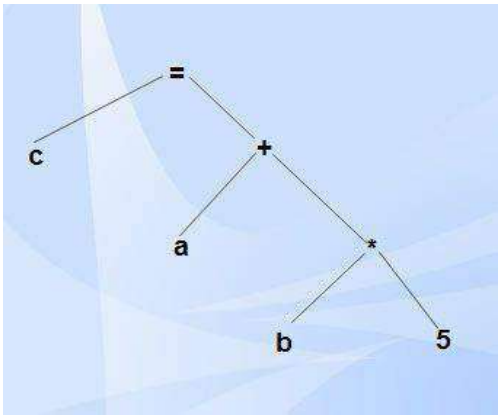


Fig. 5.0. Diagram of Syntax tree

3. SEMANTIC ANALYSIS

- Semantic analysis is the third phase of compiler.
- It checks for the semantic consistency.
- Type information is gathered and stored in symbol table or in syntax tree.
- Performs type checking.

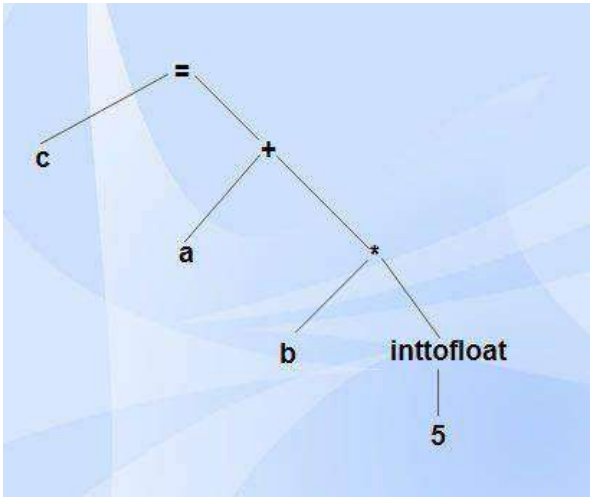


Fig. 6.0. Diagram of Semantic Tree

4. INTERMEDIATE CODE GENERATION

• Intermediate code generation produces intermediate representations for the source program which are of the following

forms:o Postfix notation

o ree address code

o Syntax tree

Most commonly used form is the three-address code.

tofloat (5)t2 = id3* tl

t3 = id2 + t2

id1 = t3

Properties of intermediate code

• It should be easy to produce.

• It should be easy to translate into target program.

5. CODE OPTIMIZATION

• Code optimization phase gets the intermediate code as input and produces optimized intermediate code as output.

• It results in faster running machine code.

• It can be done by reducing the number of lines of code for a program.

• This phase reduces the redundant code and attempts to improve the intermediate code so that faster-running machine code will result.

• During the code optimization, the result of the program is not affected.

• To improve the code generation, the optimization involves

o Deduction and removal of dead code (unreachable code).

o Calculation of constants in expressions and terms.

o Collapsing of repeated expression into temporary string.

o Loop unrolling.

o Moving code outside the loop.

o Removal of unwanted temporary variables.

t1 = id3* 5.0

id1 = id2 + t1

6. CODE GENERATION

• Code generation is the final phase of a compiler.

• It gets input from code optimization phase and produces the target code or object code as result.

• Intermediate instructions are translated into a sequence of machine instructions that perform the same task.

• The code generation involves

o Allocation of register and memory.

o Generation of correct references.

o Generation of correct data types.

o Generation of missing code.

LDF R2, id3

MULF R2, # 5.0

LDF R1, id2

ADDF R1, R2

STF id1, R1

7(a) SYMBOL TABLE MANAGEMENT

•Symbol table is used to store all the information about identifiers used in theprogram.

• It is a data structure containing a record for each identifier, with fields for the attributes of the identifier.

• It allows finding the record for each identifier quickly and to store or retrieve data from that record.

• Whenever an identifier is detected in any of the phases, it is stored in the symbol

table.

Example

int a, b; float c; char z;

Symbol name	Type	Address
A	Int	1000
B	Int	1002
C	Float	1004
Z	char	1008

Example

```
double test (double x); double sample (int count)
{
  double sum= 0.0;
      for (int i = 1; i < = count;
       i++) sum+= test((double) i);
       return sum;
  }
```

Symbol name	Type	Scope
Test	function, double	extern
X	double	function parameter
Sample	function, double	global
Count	int	function parameter
Sum	double	block local
I	int	for-loop statement

7(b) ERROR HANDLING

• Each phase can encounter errors. After detecting an error, a phase must handle the error so that compilation can proceed.

• In lexical analysis, errors occur in separation of tokens.

• In syntax analysis, errors occur during construction of syntax tree.

• In semantic analysis, errors may occur at the following cases:

(i) When the compiler detects constructs that have right syntactic structure but nomeaning

(ii) During type conversion.

• In code optimization, errors occur when the result is affected by the optimization. In code generation, it shows error when code is missing etc.

Error Encountered in Different Phases

Each phase can encounter errors. After detecting an error, a phase must some howdeal with the error, so that compilation can proceed.

A program may have the following kinds of errors at various stages:

Lexical Errors

It includes incorrect or misspelled name of some identifier i.e., identifiers typed incorrectly.

Syntactical Errors

It includes missing semicolon or unbalanced parenthesis. Syntactic errors arehandled by syntax analyzer (parser).

When an error is detected, it must be handled by parser to enable the parsing of the rest of the input. In general, errors may be expected at various stages of compilation but most of the errors are syntactic errors and hence the parser shouldbe able to detect and report those errors in the program.

The goals of error handler in parser are:

- Report the presence of errors clearly and accurately.
- Recover from each error quickly enough to detect subsequent errors.
- Add minimal overhead to the processing of correcting programs.

There are four common error-recovery strategies that can be implemented in the parser to deal with errors in the code.

- o Panic mode.
- o Statement level.
- o Error productions.
- o Global correction.

Semantical Errors

These errors are a result of incompatible value assignment. The semantic errors that the semantic analyzer is expected to recognize are:

- Type mismatch.
- Undeclared variable.
- Reserved identifier misuse.
- Multiple declaration of variable in a scope.
- Accessing an out of scope variable.
- Actual and formal parameter mismatch.

Logical errors

These errors occur due to not reachable code-infinite loop.

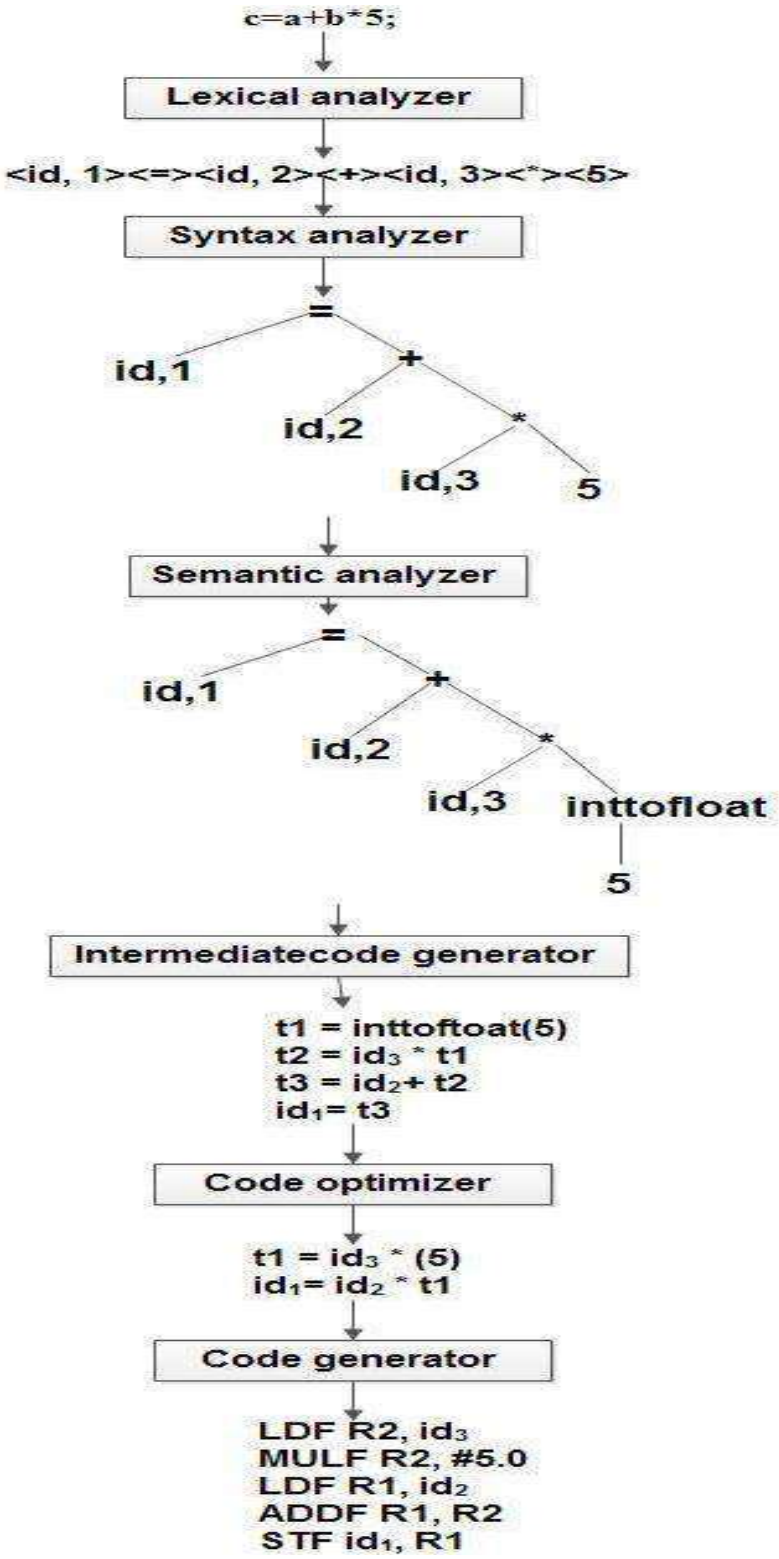


Fig.7.0 Flowchart of Phases of Compiler