

Chapter 1 INTRODUCTION

1.0 Definition

An **Operating System** is a program/software that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.

Every computer system must have at least one operating system to run other programs. Applications like Browsers, MS Office, Notepad Games, etc., need some environment to run and perform its tasks.

1.1 Objectives of Operating System

The objectives of the operating system are –

- **Program Creation** --- editors, debuggers, ... etc.. These are in the forms of utility programs that are not actually part of the O.S. but are accessible through the O.S.
- **Program Execution** --- to execute a program, instructions and data must be loaded into the main memory, I/O devices and files must be initialized.
- **Access to I/O devices** --- as if simple read and write to the programmers
- **Controlled Access to Files** --- not only the control of I/O devices, but file format on the storage medium.
- **System Access** --- shared and public resources, protection of resources and data, resolve conflicts in the contention for resources.
- **Error Detection and Response**
 - Internal/external hardware errors (memory error, device failures and mal-functions)
 - Software errors (arithmetic overflows, attempt to access forbidden memory locations, inability of the O.S. to grant the request of an application)
 - Ending a program, retrying , and reporting errors.
- **Accounting** --- collect usage statistics for various resources, billing, and monitoring performance.
- **Convenient handling**
- **Efficiency**
- **Ability to evolve**

1.1.1 Functions of Operating System

Below are the main functions of Operating System:

In an operating system software performs each of the function:

1. **Process management:-** Process management helps OS to create and delete processes. It also provides mechanisms for synchronization and communication among processes.
2. **Memory management:-** Memory management module performs the task of allocation and de-allocation of memory space to programs in need of this resources.
3. **File management:-** It manages all the file-related activities such as organization storage, retrieval, naming, sharing, and protection of files.
4. **Device Management:** Device management keeps tracks of all devices. This module also responsible for this task is known as the I/O controller. It also performs the task of allocation and de-allocation of the devices.
5. **I/O System Management:** One of the main objects of any OS is to hide the peculiarities of that hardware devices from the user.
6. **Secondary-Storage Management:** Systems have several levels of storage which includes primary storage, secondary storage, and cache storage. Instructions and data must be stored in primary storage or cache so that a running program can reference it.
7. **Security:-** Security module protects the data and information of a computer system against malware threat and authorized access.
8. **Command interpretation:** This module is interpreting commands given by the and acting system resources to process that commands.
9. **Networking:** A distributed system is a group of processors which do not share memory, hardware devices, or a clock. The processors communicate with one another through the network.
10. **Job accounting:** Keeping track of time & resource used by various job and users.
11. **Communication management:** Coordination and assignment of compilers, interpreters, and another software resource of the various users of the computer systems.

1.2 Evolution/(History)Of OS

- Operating systems were first developed in the late 1950s to manage tape storage
- The General Motors Research Lab implemented the first OS in the early 1950s for their **IBM 701**
- In the mid-1960s, operating systems started to use disks.
- In the late 1960s, the first version of the Unix OS was developed
- The first OS built by Microsoft was DOS. It was built in 1981 by purchasing the 86-DOS software from a Seattle company
- The present-day popular OS Windows first came to existence in 1985 when a GUI was created and paired with MS-DOS.

1.2.1 Types of Operating System (OS)

Following are the popular types of Operating System:

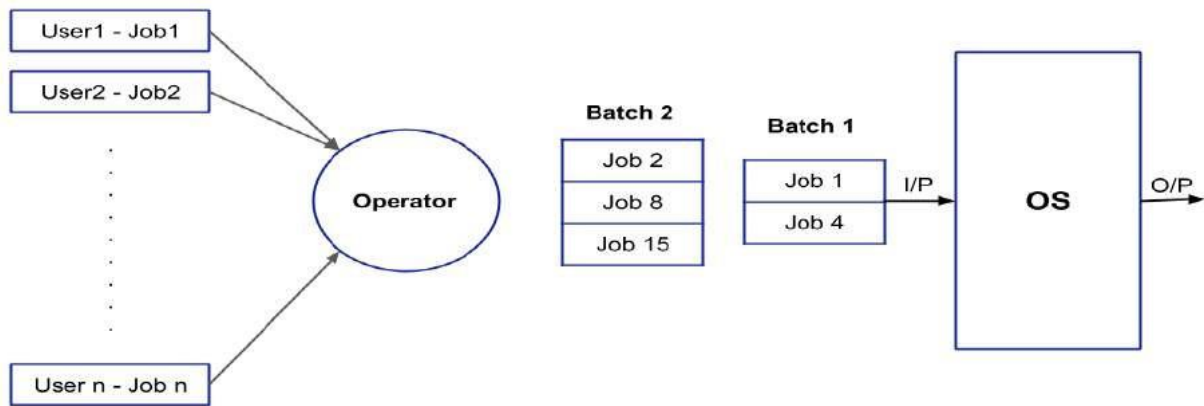
- Batch Operating System
- Multitasking/Time Sharing OS
- Distributed OS
- Embedded OS
- Real Time OS
- Network OS
- Mobile OS

1. Batch Operating System(1950s)

In a Batch Operating System, the similar jobs are grouped together into batches with the help of some operator and these batches are executed one by one.

Some computer processes are very lengthy and time-consuming. To speed the same process, a job with a similar type of needs are batched together and run as a group.

For example, let us assume that we have 10 programs that need to be executed. Some programs are written in C++, some in C and rest in Java. Now, every time when we run these programmes individually then we will have to load the compiler of that particular language and then execute the code. But what if we make a batch of these 10 programmes. The benefit with this approach is that, for the C++ batch, you need to load the compiler only once. Similarly, for Java and C, the compiler needs to be loaded only once and the whole batch gets executed. The following image describes the working of a Batch Operating System.



Advantages:

1. The overall time taken by the system to execute all the programmes will be reduced.
2. The Batch Operating System can be shared between multiple users.

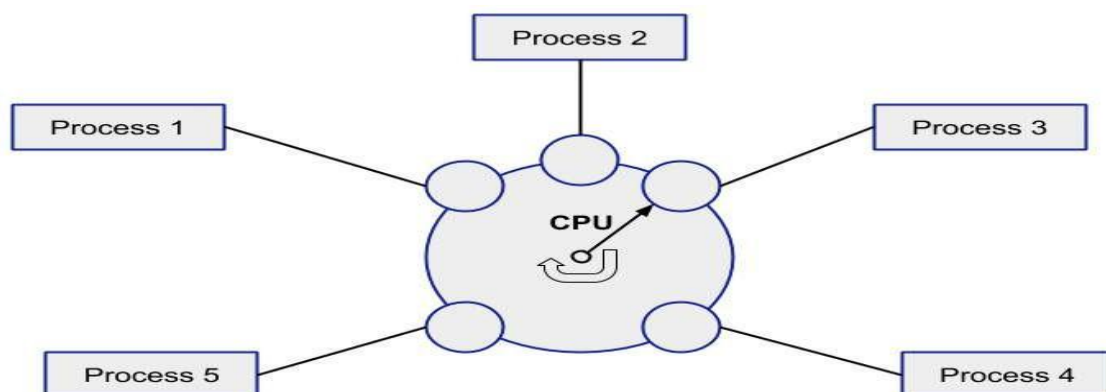
Disadvantages:

1. Manual interventions are required between two batches.
2. The CPU utilization is low because the time taken in loading and unloading of batches is very high as compared to execution time.

2. Time-Sharing(Multi-Tasking) Operating System(1960s)

In a Multi-tasking Operating System, more than one processes are being executed at a particular time with the help of the time-sharing concept. (or) Time-sharing operating system enables people located at a different terminal(shell) to use a single computer system at the same time. The processor time (CPU) which is shared among multiple users is termed as time sharing.

So, in the time-sharing environment, we decide a time that is called time quantum and when the process starts its execution then the execution continues for only that amount of time and after that, other processes will be given chance for that amount of time only. In the next cycle, the first process will again come for its execution and it will be executed for that time quantum only and again next process will come. This process will continue. The following image describes the working of a Time-Sharing Operating System.



Advantages:

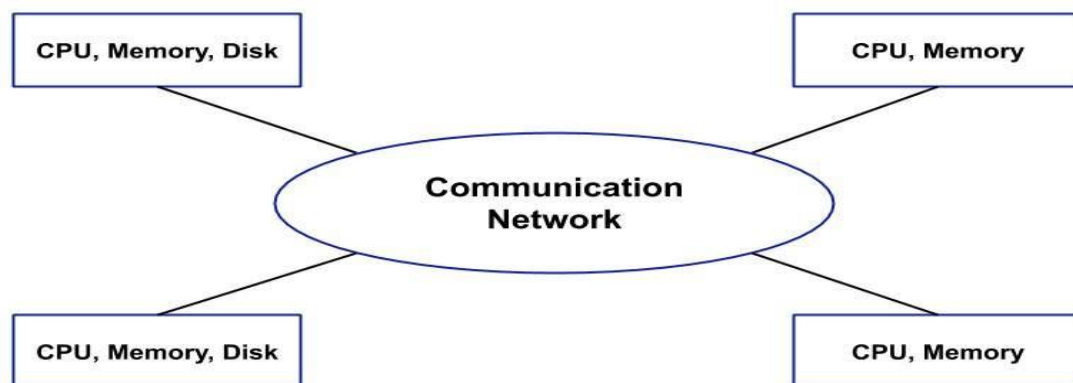
1. Since equal time quantum is given to each process, so each process gets equal opportunity to execute.
2. The CPU will be busy in most of the cases and this is good to have case.

Disadvantages:

1. Process having higher priority will not get the chance to be executed first because the equal opportunity is given to each process.

3. Distributed Operating System(1970s)

In a Distributed Operating System, we have various systems and all these systems have their own CPU, main memory, secondary memory, and resources. These systems are connected to each other using a shared communication network. Here, each system can perform its task individually. The best part about these Distributed Operating System is remote access i.e. one user can access the data of the other system and can work accordingly. So, remote access is possible in these distributed Operating Systems. The following image shows the working of a Distributed Operating System.

**Advantages:**

1. Since the systems are connected with each other so, the failure of one system can't stop the execution of processes because other systems can do the execution.
2. Resources are shared between each other.
3. The load on the host computer gets distributed and this, in turn, increases the efficiency.

Disadvantages:

4. Since the data is shared among all the computers, so to make the data secure and accessible to few computers, you need to put some extra efforts.
5. If there is a problem in the communication network then the whole communication will be broken.

4. Embedded Operating System(1970s)

An Embedded Operating System is designed to perform a specific task for a particular device which is not a computer. For example, the software used in elevators is dedicated to

the working of elevators only and nothing else. So, this can be an example of Embedded Operating System. The Embedded Operating System allows the access of device hardware to the software that is running on the top of the Operating System.

Advantages:

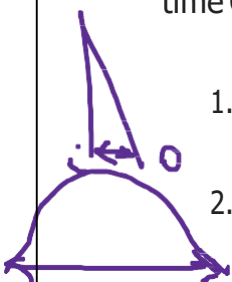
1. Since it is dedicated to a particular job, so it is fast.
2. Low cost.
3. These consume less memory and other resources.

Disadvantages:

1. Only one job can be performed.
2. It is difficult to upgrade or is nearly scalable.

5. Real-time Operating System(1980s)

The Real-time Operating Systems are used in the situation where we are dealing with some real-time data. So, as soon as the data comes, the execution of the process should be done and there should be no delay i.e. no buffer delays should be there. Real-time OS is a time-sharing system that is based on the concept of clock interrupt. So, whenever you want to process a large number of request in a very short period of time, then you should use Real-time Operating System. For example, the details of the temperature of the petroleum industry are very crucial and this should be done in real-time and in a very short period of time. A small delay can result in a life-death situation. So, this is done with the help of Real-time Operating System. There are two types of Real-time Operating System:

- 
1. **Hard Real-time:** In this type, a small delay can lead to drastic change. So, when the time constraint is very important then we use the Hard Real-time.
 2. **Soft Real-time:** Here, the time constraint is not that important but here also we are dealing with some real-time data.

Advantages:

3. There is maximum utilization of devices and resources.
4. These systems are almost error-free.

Disadvantages:

1. The algorithms used in Real-time Operating System is very complex.
2. Specific device drivers are used for responding to the interrupts as soon as possible.

6. Network Operating System(1980s)

Network Operating System runs on a server. It provides the capability to serve to manage data, user, groups, security, application, and other networking functions.

7. Mobile OS(1990s)

Mobile operating systems are those OS which is especially that are designed to power smartphones, tablets, and wearables devices. Some most famous mobile operating systems are Android and iOS, but others include BlackBerry, Web, and watchOS.

Advantage of using Operating System

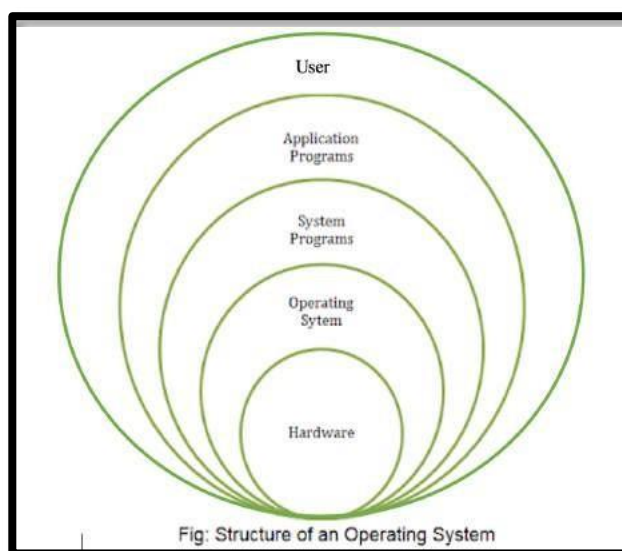
- Allows you to hide details of hardware by creating an abstraction
- Easy to use with a GUI
- Offers an environment in which a user may execute programs/applications
- The operating system must make sure that the computer system convenient to use
- Operating System acts as an intermediary among applications and the hardware components
- It provides the computer system resources with easy to use format
- Acts as an intermediary between all hardware's and software's of the system

Disadvantages of using Operating System

- If any issue occurs in OS, you may lose all the contents which have been stored in your system
- Operating system's software is quite expensive for small size organization which adds burden on them. Example Windows
- It is never entirely secure as a threat can occur at any time.

1.3 Structure of Operating System

The structure of operating system consists of four layers. These are hardware, software, system programs and application programs. The following figure depicts the structure of OS.



The hardware part consists of CPU, main memory, IO devices, secondary storage etc. The software includes routines for process management, memory management, IO control, file management. The system program layer consists of compilers, assemblers, linker, loader etc. The application programs depend on the users.

1.3.1 Simple Structure:

Simple structure OS consist of small, simple and limited systems. Their structure is not well defined. For ex. MS - DOS is a simple structure OS.

It was initially written to provide the most functionality in the least space. The system started as a small and grew beyond its original scope. Levels were not well separated and as such programs could access IO devices directly. There was no dual user or kernel mode.

Following figure shows MS-DOS simple structure OS:

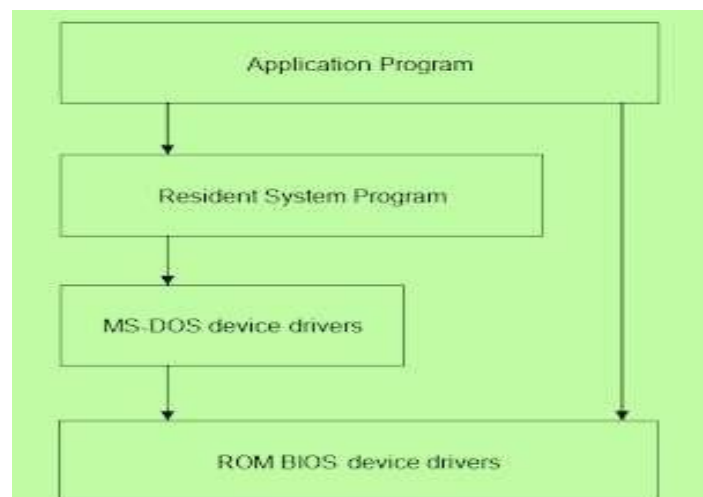


Fig: MS-DOS simple structure OS

1.3.2 Layered Approach:

At the bottom layer is the hardware layer having a layer number 0. The topmost layer (layer N) is the user interface. An OS could be structured to contain many layers.

Layered structure provides good modularity. Each layer of the OS forms a module with a clearly defined functionality and interface with the rest of the OS.

It simplifies debugging, system verification and provides facility of information hiding.

All the data and program are hidden from other module. Each layer hides the existence of certain data structures, operations and hardware from higher level layers. For eg. Windows XP, 7 .

The following figure shows a layered kernel approach:

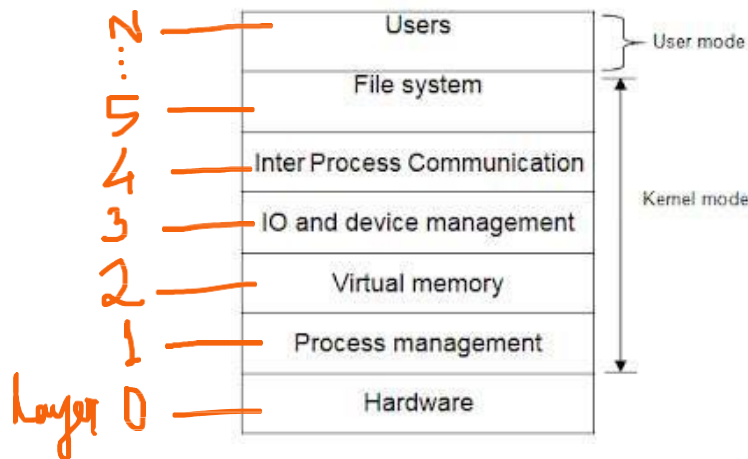
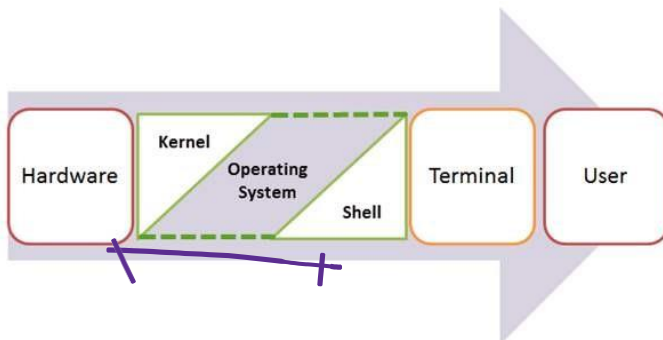


Fig: Layered kernel approach

What is a Kernel?

The kernel is the central component of a computer operating systems. The only job performed by the kernel is to manage the communication between the software and the hardware. A Kernel is at the nucleus of a computer. It makes the communication between the hardware and software possible. While the Kernel is the innermost part of an operating system, a shell is the outermost one.



Features of Kernel

- Low-level scheduling of processes
- Inter-process communication
- Process synchronization
- Context switching

Types of Kernels

There are many types of kernels that exists, but among them, the two most popular kernels are:

1. Monolithic Kernel

Monolithic kernel can be defined as single large process or we can say that everything is in one single space. The whole stuff resides in one single mode that is kernel mode. This increases size of kernel, further increases the size of OS. All the file systems, memory

management systems, device drivers all are included in kernel address space. Eg. of monolithic kernel operating systems are Unix, Linux.

Advantage of monolithic kernel

1. The execution of process is very fast because all required resources are at one address space so the access time is very less and hence the execution is faster.
2. It makes a communication link between the application and the hardware using system call.

Disadvantage of monolithic kernel

1. If one of the process or service fails the entire system will crashed, or if a new service is to be added that the entire system has to be modified.

2. Microkernels

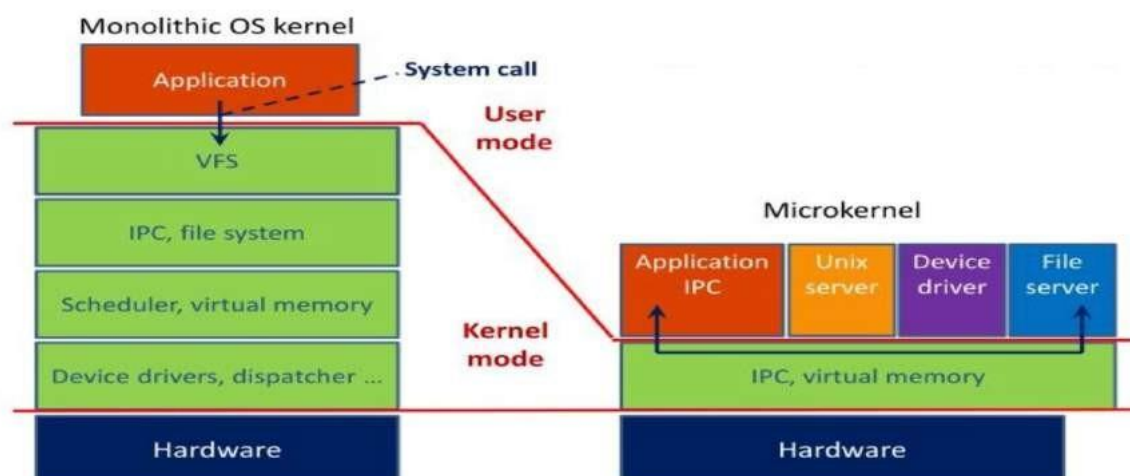
On the other hand the In microkernel the kernel is broke down into separate processes, some of them run in kernel mode or space while some of them run in user space. Functionalities like IPC(inter process communication), process scheduling, memory management resides in kernel space while device drivers, file system reside in user space. As it is cleared from the above figure that Microkernel has two separate areas while Monolithic have one large space which includes all processes. In Microkernel, concept through which processes communicate is Inter process Communication(IPC). It is the mechanism through which processes send message to each other in microkernel OS and this mechanism includes sharing of resources as well as information between the processes. For eg. of microkernel OS are MacOS, Windows NT, L4Linux.

Advantage of microkernel

1. Any process failure will not affect the whole system since the processes are running in different modes. Hence it is secure and reliable as compared to monolithic kernel.
2. One more advantage of microkernel is since the user services and kernel services are implemented in different address space, this reduces the size of kernel and further reduces the size of OS.

Disadvantage of microkernel

1. The execution of process is slow because the device drivers reside in a separate mode and any process have to invoke for a resource by sending message via IPC (Inter Process Communication).



Difference between Firmware and Operating System

Firmware	Operating System
Firmware is one kind of programming that is embedded on a chip in the device which controls that specific device.	OS provides functionality over and above that which is provided by the firmware.
Firmware is programs that been encoded by the manufacture of the IC or something and cannot be changed.	OS is a program that can be installed by the user and can be changed.
It is stored on non-volatile memory.	OS is stored on the hard drive.

Difference between 32-Bit vs. 64 Bit Operating System

Parameters	32-Bit OS	64-Bit OS
Architecture and Software	Allow 32 bit of data processing simultaneously	Allow 64 bit of data processing simultaneously
Compatibility	32-bit applications require 32-bit OS and CPUs.	64-bit applications require a 64-bit OS and CPU.
Systems Available	All versions of Windows 8, Windows 7, Windows Vista, and Windows XP, Linux, etc.	Windows XP Professional, Vista, 7, Mac OS X and Linux.
Memory Limits	32-bit systems are limited to 3.2 GB of RAM.	64-bit systems allow a maximum 17 Billion GB of RAM.

Summary

- Define Operating System: An operating system is a software which acts as an interface between the end user and computer hardware
- Operating systems were first developed in the late 1950s to manage tape storage
- The kernel is the central component of a computer operating systems. The only job performed by the kernel is to the manage the communication between the software and the hardwareTwo most popular kernels are Monolithic and MicroKernels
- Process, Device, File, I/O, Secondary-Storage, Memory management are various functions of an OS.

Chapter 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.

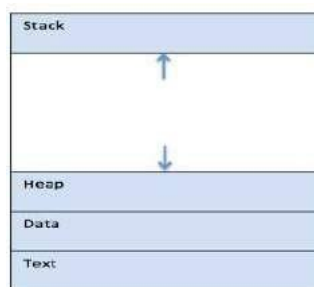
2.1 Process Concept

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

A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections — stack, heap, text and data. The following image shows a simplified **layout of a process inside main memory** –



S.N.	Component & Description
1	Stack The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2	Heap This is dynamically allocated memory to a process during its run time.
3	Text This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4	Data This section contains the global and static variables.

2.1.1 Program

A computer program is a collection of instructions that performs a specific task when executed by a computer. For example, here is a simple program written in C programming language –

```
#include <stdio.h>

int main() {
    printf("Hello, World! \n");
    return 0;
}
```

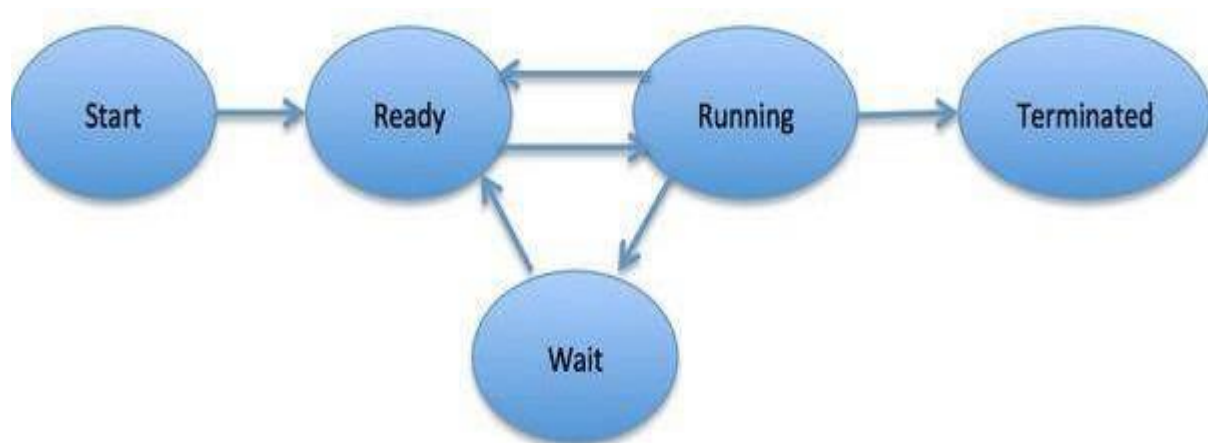
When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.

A part of a computer program that performs a well-defined task is known as an **algorithm**.

A collection of computer programs, libraries and related data are referred to as a **software**.

2.1.2 Process Life Cycle

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.



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.

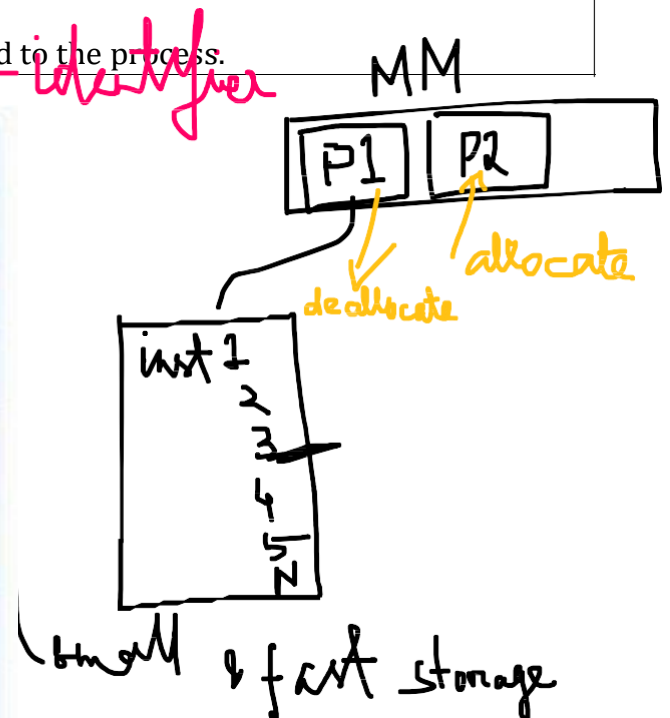
2.1.3 Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for 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 below in the table –

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-

S.N.	Information & Description
1	Process State The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	Process privileges This is required to allow/disallow access to system resources.
3	Process ID Unique identification for each of the process in the operating system.
4	Pointer A pointer to parent process.
5	Program Counter

	Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	CPU registers Various CPU registers where process need to be stored for execution for running state.
7	CPU Scheduling Information Process priority and other scheduling information which is required to schedule the process.
8	Memory management information This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	Accounting information This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	IO status information This includes a list of I/O devices allocated to the process.



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

2.1.4 Interacting processes

Interactive processing means that the person needs to provide the computer with instructions whilst it is doing the processing.

For example, imagine that a computer is running a program that takes a set of files from one directory and does some work on each one. As each file is processed the computer sends a screen message to the operator "Where do you want this file to be stored". i.e. the user 'interacts' with the computer to complete the processing.

2.1.5 Inter process communication IPC

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

Why IPC?

Here, are the reasons for using the interprocess communication protocol for information sharing:

- It helps to speedup modularity
- Computational
- Convenience
- Helps operating system to communicate with each other and synchronize their actions.

Approaches for Inter-Process Communication

A process can be of two types:

- Independent process.
- Co-operating process.

An independent process is not affected by the execution of other processes while a co-operating 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 utilized for increasing computational speed, convenience, and modularity.

Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions.

IPC mechanism provides two operations: Send (message) and Received (message).

Here, are few important methods for interprocess communication:

1. **Pipes:** Pipe is widely used for communication between two related processes. This is a half-duplex method, so the first process communicates with the second process. However, in order to achieve a full-duplex, another pipe is needed.
2. **Message Passing:** It is a mechanism for a process to communicate and synchronize. Using message passing, the process communicates with each other without resorting to shared variables.
3. **Message Queues:** A message queue is a linked list of messages stored within the kernel. It is identified by a message queue identifier. This method offers communication between single or multiple processes with full-duplex capacity.
4. **Direct Communication:** In this type of inter-process communication process, should name each other explicitly. In this method, a link is established between one pair of communicating processes, and between each pair, only one link exists.
5. **Indirect Communication:** Indirect communication establishes like only when processes share a common mailbox each pair of processes sharing several communication links. A link can communicate with many processes. The link may be bi-directional or unidirectional.
6. **Shared Memory:** Shared memory is a memory shared between two or more processes that are established using shared memory between all the processes. This type of memory requires to protected from each other by synchronizing access across all the processes.
7. **FIFO:** Communication between two unrelated processes. It is a full-duplex method, which means that the first process can communicate with the second process, and the opposite can also happen.



2.2 Implementation issues of Processes

- It is required to protect multiple processes from one another.
- It is required to coordinate multiple processes through additional mechanisms.
- Additional performance overheads and complexities in operating systems are required for switching among processes.
- Sometimes running too many processes concurrently leads to severely degraded performance.

2.3 Process scheduling, Job scheduling

The act of determining which process is in the **ready** state, and should be moved to the **running** state is known as **Process Scheduling**.

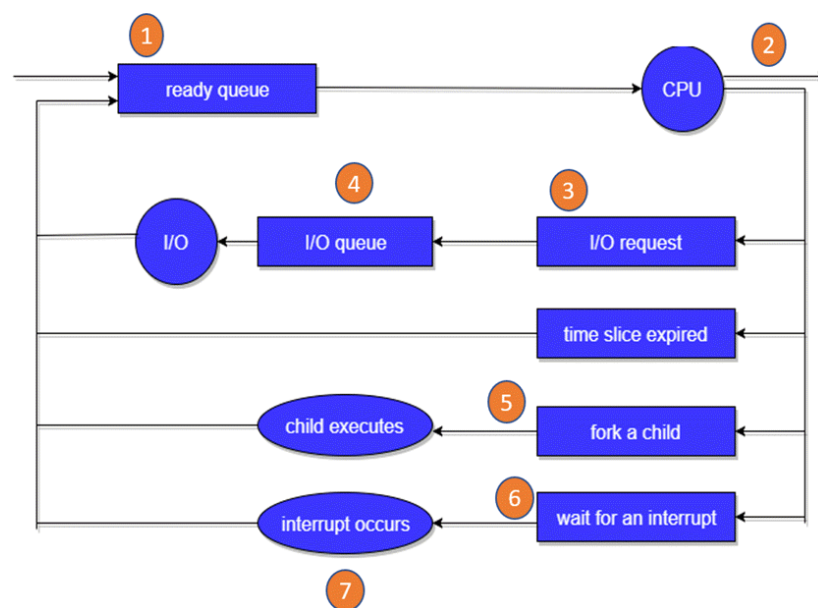
The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs. For achieving this, the scheduler must apply appropriate rules for swapping processes **IN** and **OUT** of CPU.

Scheduling fell into one of the two general categories:

- **Non Pre-emptive Scheduling:** When the currently executing process gives up the CPU voluntarily.
- **Pre-emptive Scheduling:** When the operating system decides to favour another process, pre-empting the currently executing process.

Three types of operating system queues are:

1. **Job queue** – It helps you to store all the processes in the system.
2. **Ready queue** – This type of queue helps you to set every process residing in the main memory, which is ready and waiting to execute.
3. **Device queues** – It is a process that is blocked because of the absence of an I/O device.



In the above-given Diagram,

- Rectangle represents a queue.
 - Circle denotes the resource
 - Arrow indicates the flow of the process.
1. Every new process first put in the Ready queue .It waits in the ready queue until it is finally processed for execution. Here, the new process is put in the ready queue and wait until it is selected for execution or it is dispatched.
 2. One of the processes is allocated the CPU and it is executing

3. The process should issue an I/O request
4. Then, it should be placed in the I/O queue.
5. The process should create a new subprocess
6. The process should be waiting for its termination.
7. It should remove forcefully from the CPU, as a result interrupt. Once interrupt is completed, it should be sent back to ready queue.

2.3.1 Scheduling Objectives

Here, are important objectives of Process scheduling

- Maximize the number of interactive users within acceptable response times.
- Achieve a balance between response and utilization.
- Avoid indefinite postponement and enforce priorities.
- It also should give reference to the processes holding the key resources.

2.3.2 Type of Process Schedulers

A scheduler is a type of system software that allows you to handle process scheduling. There are mainly three types of Process Schedulers:

1. Long Term
2. Short Term
3. Medium Term

1. Long Term Scheduler

Long term scheduler is also known as a **job scheduler**. This scheduler regulates the program and select process from the queue and loads them into memory for execution. It also regulates the degree of multi-programming.

However, the main goal of this type of scheduler is to offer a balanced mix of jobs, like Processor, I/O jobs., that allows managing multiprogramming.

2. Medium Term Scheduler

Medium-term scheduling is an important part of **swapping**. It enables you to handle the swapped out-processes. In this scheduler, a running process can become suspended, which makes an I/O request.

A running process can become suspended if it makes an I/O request. A suspended processes can't make any progress towards completion. In order to remove the process from memory and make space for other processes, the suspended process should be moved to secondary storage.

3. Short Term Scheduler

Short term scheduling is also known as **CPU scheduler**. The main goal of this scheduler is to boost the system performance according to set criteria. This helps you to select from a group of processes that are ready to execute and allocates CPU to one of them. The dispatcher gives control of the CPU to the process selected by the short term scheduler.

2.3.3 Difference between Schedulers

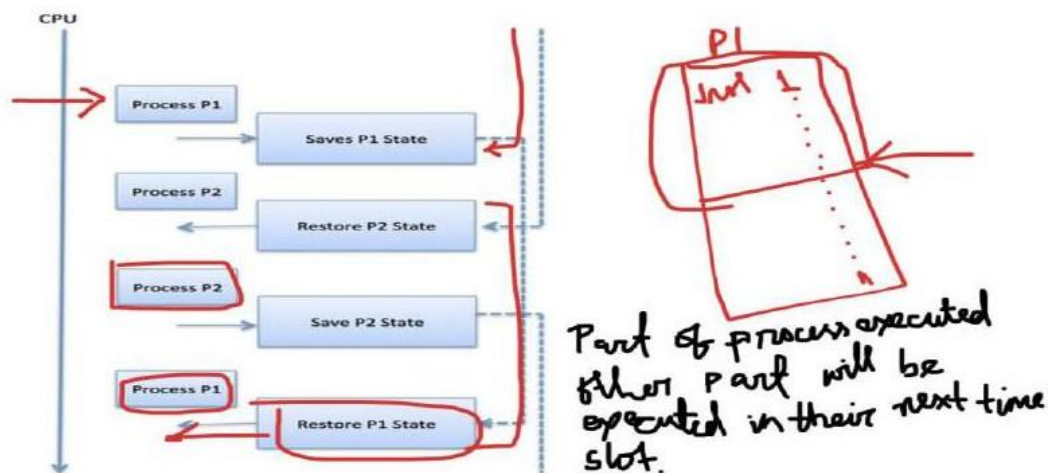
Long-Term Vs. Short Term Vs. Medium-Term

Long-Term	Short-Term	Medium-Term
Long term is also known as a <i>job scheduler</i>	Short term is also known as <i>CPU scheduler</i>	Medium-term is also called <i>swapping scheduler</i> .
It is either absent or minimal in a time-sharing system.	It is insignificant in the time-sharing order.	This scheduler is an element of Time-sharing systems.
Speed is less compared to the short term scheduler.	Speed is the fastest compared to the long-term and medium-term scheduler.	It offers medium speed.
Allow you to select processes from the system and pool back into the memory	It only selects processes that is in a ready state of the execution.	It helps you to send process back to memory.
Offers full control but provides balanced multiprogramming	Offers less control and high level multiprogramming	Reduce the level of multiprogramming.

2.3.4 What is Context switch?

It is a method to store/restore the state or of a CPU in PCB. So that process execution can be resumed from the same point at a later time. The context switching method is

important for multitasking OS.



2.3.5 Job Scheduling

- Job scheduling is the process of allocating system resources to many different tasks by an operating system (OS).
- The system handles prioritized job queues that are awaiting CPU time and it should determine which job to be taken from which queue and the amount of time to be allocated for the job.
- This type of scheduling makes sure that all jobs are carried out fairly and on time.
- Job scheduling is performed using job schedulers. Job schedulers are programs that enable scheduling and, at times, track computer "batch" jobs, or units of work like the operation of a payroll program.
- Job schedulers have the ability to start and control jobs automatically by running prepared job-control-language statements or by means of similar communication with a human operator. Generally, the present-day job schedulers include a graphical user interface (GUI) along with a single point of control.
- In scheduling, many different schemes are used to determine which specific job to run.
- Some parameters that may be considered are as follows:
 - Job priority
 - Availability of computing resource
 - License key if the job is utilizing a licensed software
 - Execution time assigned to the user
 - Number of parallel jobs permitted for a user
 - Projected execution time
 - Elapsed execution time
 - Presence of peripheral devices
 - Number of cases of prescribed events

2.4 Process synchronization, semaphore

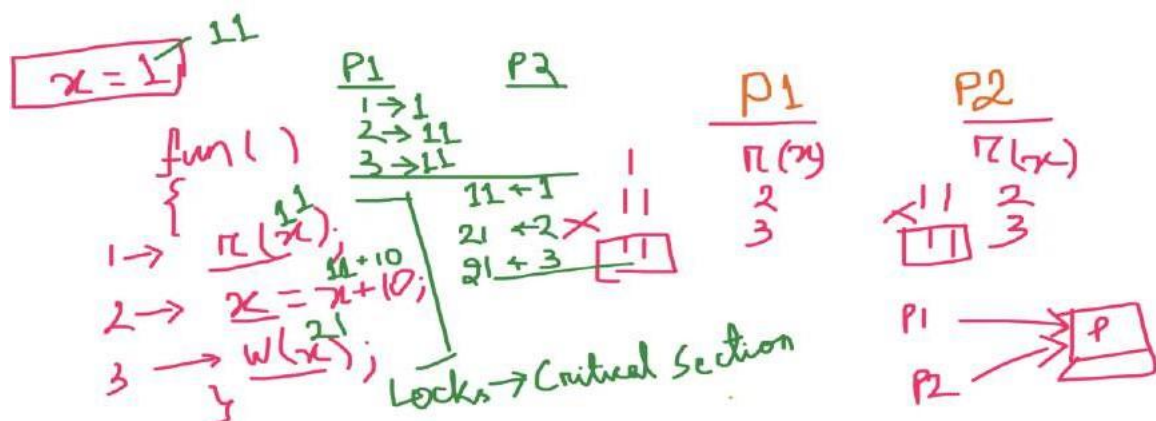
2.4.1 Process Synchronization: It is the task phenomenon of coordinating the execution of processes in such a way that no two processes can have access to the same shared data and resources.

- 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.

1. Race Condition

A race condition is a situation that may occur inside a critical section. A race condition is a condition when there are many processes and every process shares the data with each other and accessing the data concurrently, and the output of execution depends on a particular sequence in which they share the data and access.

Also, proper process(thread) synchronization using locks or atomic variables can prevent race conditions.



2. Critical Section

The critical section in a code segment where the shared variables can be accessed. Atomic action is required in a critical section i.e. only one process can execute in its critical section at a time. All the other processes have to wait to execute in their critical sections.

The critical section is given as follows:

```
do{
    Entry Section  $S = x - 1$ 
    Critical Section
} while(1)
```

Handwritten notes in green ink: $P1 \dots$, $\{$, $\}$, entry sec

```
Exit Section
Remainder Section
} while (TRUE);
```

CS
} exit sec;

Here, are four essential elements of the critical section:

- **Entry Section:** It is part of the process which decides the entry of a particular process. It acquires the resources needed for execution by the process.
- **Critical Section:** This part allows one process to enter and modify the shared variable.
- **Exit Section:** It checks that a process that finished its execution should be removed through this Section. It releases the resources and also informs the other processes that critical section is free.
- **Remainder Section:** All other parts of the Code, which is not in Critical, Entry, and Exit Section, are known as the Remainder Section.

The critical section problem needs a solution to synchronise the different processes.

The solution to the critical section problem must satisfy the following conditions –

- **Mutual Exclusion**

Mutual exclusion implies that only one process can be inside the critical section at any time. If any other processes require the critical section, they must wait until it is free.

- **Progress**

Progress means that if a process is not using the critical section, then it should not stop any other process from accessing it. In other words, any process can enter a critical section if it is free.

- **Bounded Waiting**

Bounded waiting means that each process must have a limited waiting time. It should not wait endlessly to access the critical section.

2.4.2 Semaphores

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.

The definitions of wait and signal are as follows –

- **Wait**

The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

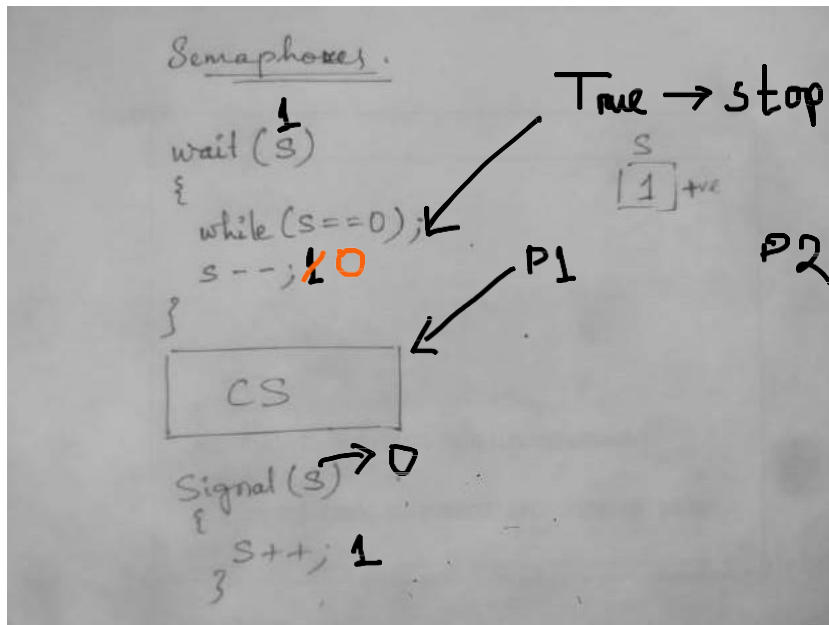
```
wait(S) ← +ve value
{
    while (S==0);

    S--;
}
```

- **Signal**

The signal operation increments the value of its argument S.

```
signal(S)
{
    S++;
}
```



Types of Semaphores

There are two main types of semaphores i.e. counting semaphores and binary semaphores. Details about these are given as follows –

- **Counting Semaphores**

These are integer value semaphores and have an unrestricted value domain. These semaphores are used to coordinate the resource access where the semaphore count is the number of available resources. If the resources are added, semaphore count automatically incremented and if the resources are removed, the count is decremented.

- **Binary Semaphores**

The binary semaphores are like counting semaphores but their value is restricted to 0 and 1. The wait operation only works when the semaphore is 1

and the signal operation succeeds when semaphore is 0. It is sometimes easier to implement binary semaphores than counting semaphores.

Advantages of Semaphores

Some of the advantages of semaphores are as follows –

- Semaphores allow only one process into the critical section. They follow the mutual exclusion principle strictly and are much more efficient than some other methods of synchronization.
- There is no resource wastage because of busy waiting in semaphores as processor time is not wasted unnecessarily to check if a condition is fulfilled to allow a process to access the critical section.
- Semaphores are implemented in the machine independent code of the microkernel. So they are machine independent.

Disadvantages of Semaphores

Some of the disadvantages of semaphores are as follows –

- Semaphores are complicated so the wait and signal operations must be implemented in the correct order to prevent deadlocks.
- Semaphores are impractical for last scale use as their use leads to loss of modularity. This happens because the wait and signal operations prevent the creation of a structured layout for the system.
- Semaphores may lead to a priority inversion where low priority processes may access the critical section first and high priority processes later.

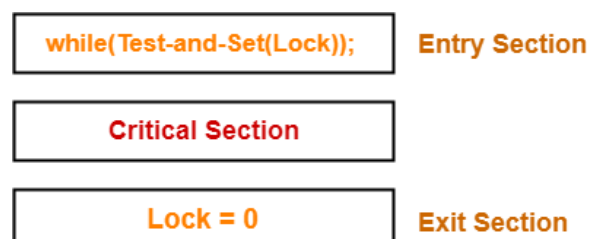
2.4.3 Test and Set Instruction (Lock)–

- Test and Set Lock (TSL) is a synchronization mechanism.
- It uses a test and set instruction to provide the synchronization among the processes executing concurrently.

- **Test-and-Set Instruction**

- It is an instruction that returns the old value of a memory location and sets the memory location value to 1 as a single atomic operation.
- If one process is currently executing a test-and-set, no other process is allowed to begin another test-and-set until the first process test-and-set is finished.

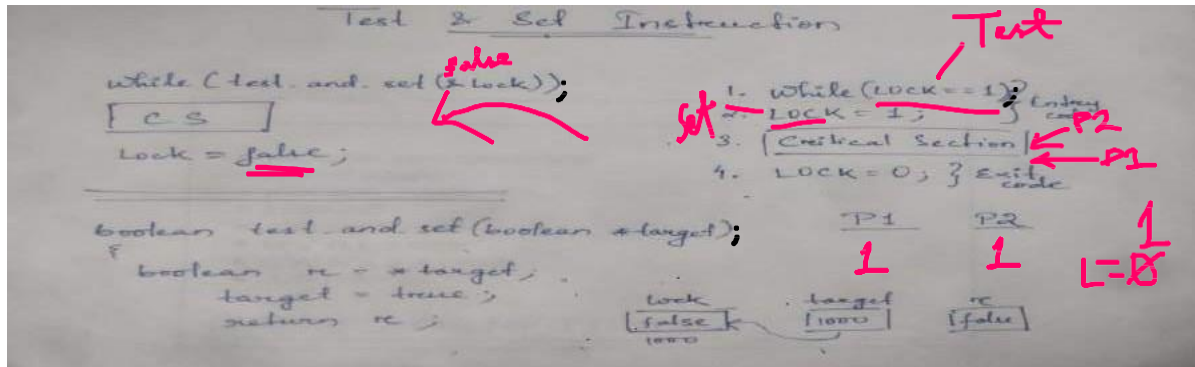
It is implemented as-



Initially, lock value is set to 0.

- Lock value = 0 means the critical section is currently vacant and no process is present inside it.
- Lock value = 1 means the critical section is currently occupied and a process is present inside it.

Working-



Scene-01:

- Process P_0 arrives.
- It executes the test-and-set(Lock) instruction.
- Since lock value is set to 0, so it returns value 0 to the while loop and sets the lock value to 1.
- The returned value 0 breaks the while loop condition.
- Process P_0 enters the critical section and executes.
- Now, even if process P_0 gets preempted in the middle, no other process can enter the critical section.
- Any other process can enter only after process P_0 completes and sets the lock value to 0.

Scene-02:

- Another process P_1 arrives.
- It executes the test-and-set(Lock) instruction.
- Since lock value is now 1, so it returns value 1 to the while loop and sets the lock value to 1.
- The returned value 1 does not break the while loop condition.
- The process P_1 is trapped inside an infinite while loop.
- The while loop keeps the process P_1 busy until the lock value becomes 0 and its condition breaks.

Scene-03:

- Process P_0 comes out of the critical section and sets the lock value to 0.
- The while loop condition breaks.
- Now, process P_1 waiting for the critical section enters the critical section.

- Now, even if process P_1 gets preempted in the middle, no other process can enter the critical section.
- Any other process can enter only after process P_1 completes and sets the lock value to 0.

Characteristics-

The characteristics of this synchronization mechanism are-

- It ensures mutual exclusion.
- It is deadlock free.
- It does not guarantee bounded waiting and may cause starvation.
- It suffers from spin lock.
- It is not architectural neutral since it requires the operating system to support test-and-set instruction.
- It is a busy waiting solution which keeps the CPU busy when the process is actually waiting.

Point-01:

This synchronization mechanism guarantees mutual exclusion.

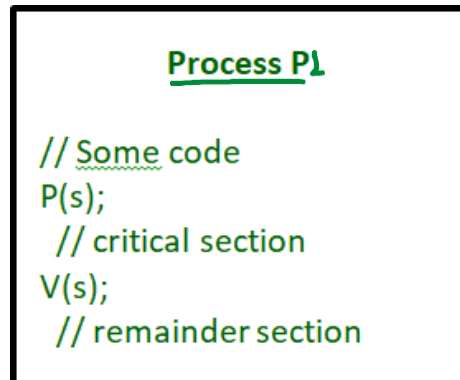
Explanation-

- The success of the mechanism in providing mutual exclusion lies in the test-and-set instruction.
- Test-and-set instruction returns the old value of memory location (lock) and updates its value to 1 simultaneously.
- The fact that these two operations are performed as a single atomic operation ensures mutual exclusion.
- Preemption after reading the lock value was a major cause of failure of lock variable synchronization mechanism.
- Now, no preemption can occur immediately after reading the lock value.

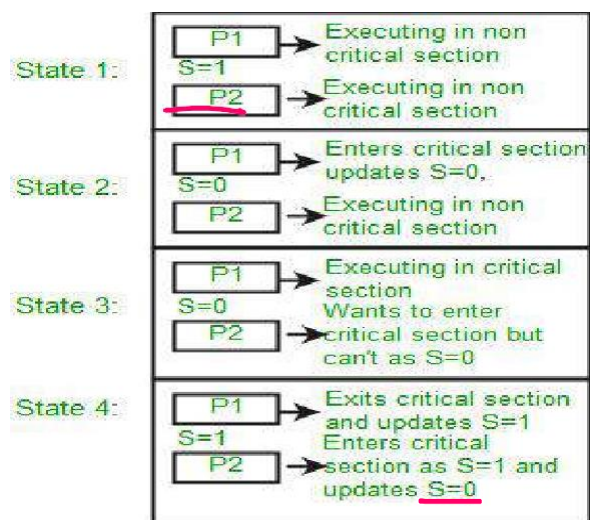
2.4.4 P() and V() operation-

1. P operation is also called wait, sleep, or down operation, and V operation is also called signal, wake-up, or up operation.
2. The general form of lock and unlock mechanism is called as P() and V() operation. Semaphores support P() and V() operation.
3. Both operations are atomic and semaphore(s) is always initialized to one. Here atomic means that variable on which read, modify and update happens at the same time/moment with no pre-emption i.e. in-between read, modify and update no other operation is performed that may change the variable.

4. A critical section is surrounded by both operations to implement process synchronization. See the below image. The critical section of Process P1 is in between P and V operation.



Now, let us see how it implements mutual exclusion. Let there be two processes P1 and P2 and a semaphore S is initialized as 1. Now if suppose P1 enters in its critical section then the value of semaphore s becomes 0. Now if P2 wants to enter its critical section then it will wait until $s > 0$, this can only happen when P1 finishes its critical section and calls V operation on semaphore S. This way mutual exclusion is achieved. Look at the below image for details which is Binary semaphore.



2.5 Principle of concurrency, types of scheduling

2.5.1 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.

2.5.2 CPU Scheduling: Scheduling Criteria

There are many different criteria to check when considering the "**best**" scheduling algorithm, they are:

- i. **CPU Utilization:** To make out the best use of the CPU and not to waste any CPU cycle, the CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)
- ii. **Throughput** It is the total number of processes completed per unit of time or rather says *the total amount of work done in a unit of time*. This may range from 10/second to 1/hour depending on the specific processes.
- iii. **Turnaround Time(TAT):** It is the amount of time taken to execute a particular process, i.e. The interval from the time of submission of the process to the time of completion of the process(Wall clock time).
- iv. **Waiting Time:** The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.
- v. **TAT Average:** It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.
- vi. **Response Time:** Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

Note: In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

2.5.3 Scheduling Algorithms

To decide which process to execute first and which process to execute last to achieve maximum CPU utilization, computer scientists have defined some algorithms, they are:

1. First Come First Serve(FCFS)
2. Shortest Job First(SJF)
3. Priority Scheduling
4. Round Robin Scheduling

1. First Come First Serve(FCFS)

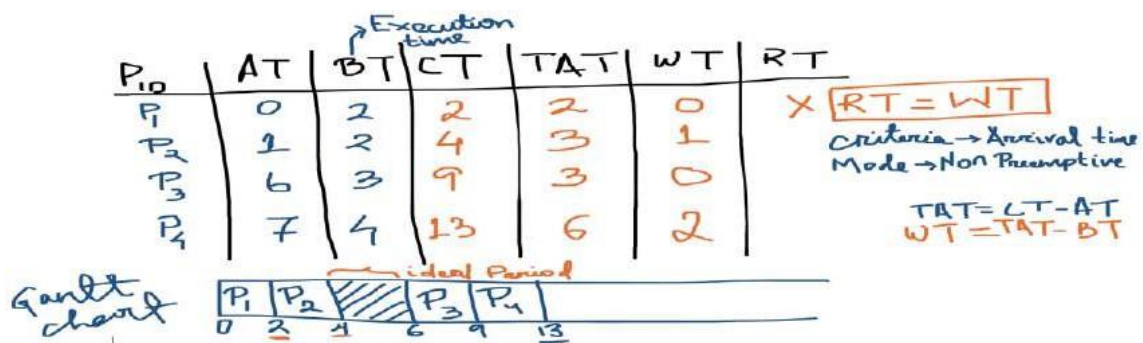
First Come First Serve is the full form of FCFS. It is the easiest and most simple CPU scheduling algorithm. In this type of algorithm, the process which requests the CPU

gets the CPU allocation first. This scheduling method can be managed with a FIFO queue.

As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue. So, when CPU becomes free, it should be assigned to the process at the beginning of the queue.

Characteristics of FCFS method:

- It offers non pre-emptive scheduling algorithm.
- Jobs are always executed on a first-come, first-serve basis
- It is easy to implement and use.
- However, this method is poor in performance, and the general wait time is quite high.



2. Shortest Job First Scheduling(SJF)

SJF is a full form of (Shortest job first) is a scheduling algorithm in which the process with the shortest execution time should be selected for execution next. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.

Characteristics of SJF Scheduling:

- It is associated with each job as a unit of time to complete.
- In this method, when the CPU is available, the next process or job with the shortest completion time will be executed first.
- It is Implemented with non-preemptive policy.
- This algorithm method is useful for batch-type processing, where waiting for jobs to complete is not critical.
- It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

P_{ID}	AT	<u>BT</u>	CT	TAT	WT
P_1	1	3	6	5	2
P_2	2	<u>4</u>	10	8	4
P_3	1	2	3	2	0
P_4	4	<u>4</u>	14	10	6

SJF

criteria
BT

$TAT = CT - AT$
 $WT = TAT - BT$

Graph chart

P_3	P_1	P_2	P_4
1	3	6	10
14			

P_1 P_3 P_4

3. Round-Robin Scheduling(RR)

Round robin is the oldest, simplest scheduling algorithm. The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turn. It is mostly used for scheduling algorithms in multitasking. This algorithm method helps for starvation free execution of processes.

Characteristics of Round-Robin Scheduling:

- Round robin is a hybrid model which is clock-driven
- Time slice should be minimum, which is assigned for a specific task to be processed. However, it may vary for different processes.
- It is a real time system which responds to the event within a specific time limit.

Priority	P ₁₀	AT	DT	CT	TAT	WT	RT
10	P ₁	0	5	12	12	12-5=7	0
20	P ₂	1	4	8	7	7-4=3	0
30	P ₃	2	2	4	2	2-2=0	0
40	P ₄	4	1	5	1	1-1=0	0

Grant Chart	P ₁	P ₂	P ₃	P ₄	P ₃	P ₂	P ₁
	0	1	2	3	4	5	6

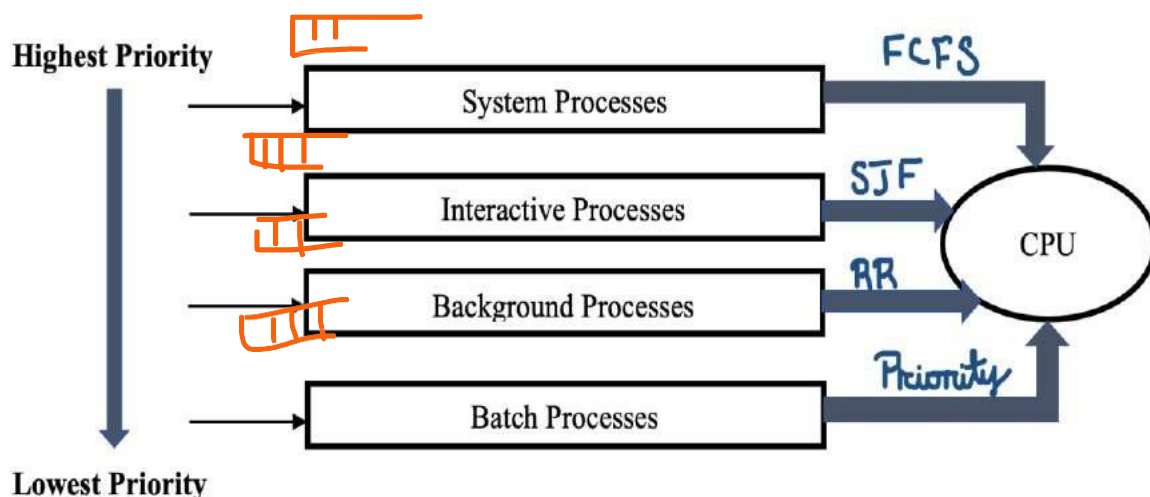
Higher Value, Higher Priority
 Avg TAT = $\frac{22}{4} = 5.5$
 Avg WT = $\frac{10}{4} = 2.5$

Mode "Pre-emptive"
 Criteria - Priority

5. Multiple-Level Queues Scheduling

This algorithm separates the ready queue into various separate queues. In this method, processes are assigned to a queue based on a specific property of the process, like the process priority, size of the memory, etc.

However, this is not an independent scheduling OS algorithm as it needs to use other types of algorithms in order to schedule the jobs.



Characteristic of Multiple-Level Queues Scheduling:

- Multiple queues should be maintained for processes with some characteristics.
- Every queue may have its separate scheduling algorithms.
- Priorities are given for each queue.

The Purpose of a Scheduling algorithm

Here are the reasons for using a scheduling algorithm:

- The CPU uses scheduling to improve its efficiency.
- It helps you to allocate resources among competing processes.
- The maximum utilization of CPU can be obtained with multi-programming.
- The processes which are to be executed are in ready queue.

2.5.4 Concurrency , Principles of Concurrency:

Concurrency is the execution of the multiple instruction sequences at the same time. It happens in the operating system when there are several process threads running in parallel. The running process threads always communicate with each other through shared memory or message passing. Concurrency results in sharing of resources result in problems like deadlocks and resources starvation.

It helps in techniques like coordinating execution of processes, memory allocation and execution scheduling for maximizing throughput.

Principles of Concurrency:

Both interleaved and overlapped processes can be viewed as examples of concurrent processes, they both present the same problems.

Multiprogramming processes- allow processing time to be dynamically shared among a number of active applications.

Structured applications: As an extension of the principles of modular design and structured programming, some applications can be effectively programmed as a set of concurrent processes.

Operating system structure: The same structuring advantages apply to systems programs, and we have seen that operating systems are themselves often implemented as a set of processes or threads.

The relative speed of execution cannot be predicted. It depends on the following:

- The activities of other processes
- The way operating system handles interrupts
- The scheduling policies of the operating system

Problems in Concurrency :

- **Sharing global resources -**
Sharing of global resources safely is difficult. If two processes both make use of a global variable and both perform read and write on that variable, then the order in which various read and write are executed is critical.
- **Optimal allocation of resources -**
It is difficult for the operating system to manage the allocation of resources optimally.
- **Locating programming errors -**
It is very difficult to locate a programming error because reports are usually not reproducible.
- **Locking the channel -**
It may be inefficient for the operating system to simply lock the channel and prevents its use by other processes.

Advantages of Concurrency :

- **Running of multiple applications -**
It enable to run multiple applications at the same time.
- **Better resource utilization -**
It enables that the resources that are unused by one application can be used for other applications.

- **Better average response time –**
Without concurrency, each application has to be run to completion before the next one can be run.
- **Better performance –**
It enables the better performance by the operating system. When one application uses only the processor and another application uses only the disk drive then the time to run both applications concurrently to completion will be shorter than the time to run each application consecutively.

Some more Issues of Concurrency :

- **Non-atomic**
Operations that are non-atomic but interruptible by multiple processes can cause problems.
- **Race-conditions**
A race condition occurs if the outcome depends on which of several processes gets to a point first.
- **Blocking**
Processes can block waiting for resources. A process could be blocked for long period of time waiting for input from a terminal. If the process is required to periodically update some data, this would be very undesirable.
- **Starvation**
It occurs when a process does not obtain service to progress.
- **Deadlock**
It occurs when two processes are blocked and hence neither can proceed to execute.

Chapter 3

Memory Management

3.1 Memory allocation Techniques

1. Contiguous memory allocation
2. Non-contiguous memory allocation

3.2 Swapping

3.3 Paging, Segmentation, virtual memory using paging,

3.4 Demand paging, page fault handling.

3.0. Definition

Memory Management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution.

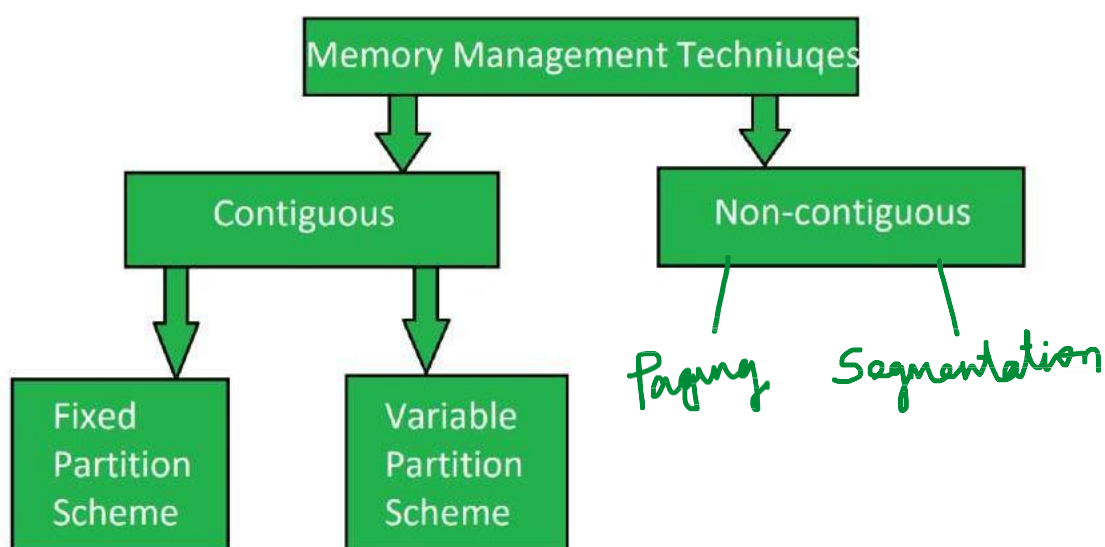
The main functions of memory management are as follows:

- It's most important function is to manages primary memory.
- It helps processes to move back and forward between the main memory and execution disk.
- It helps OS to keep track of every memory location, irrespective of whether it is allocated to some process or it remains free and correspondingly updates its status.
- It checks how much memory is to be allocated to processes.
- It decides which process will get memory at what time.

Goals of Memory Management:

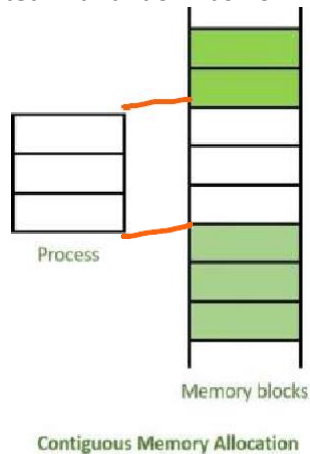
1. **Space Utilization:** By using **Fragmentation**, this tries to keep memory wastage as low as possible.
2. **Runs larger program in smaller memory area-** By using the concept of **Virtual memory**.

3.1 Memory allocation Techniques

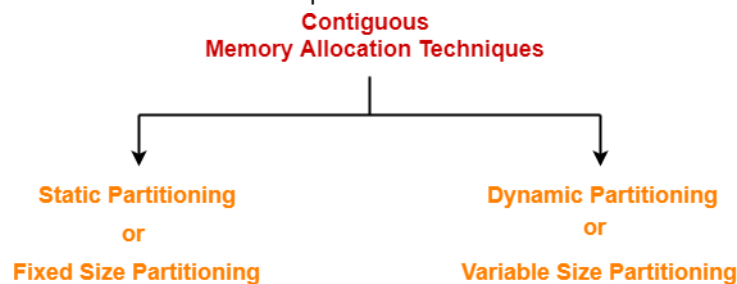


3.1.1. Contiguous Memory Allocation :

Contiguous memory allocation is basically a method in which a single contiguous section/part of memory is allocated to a process or file needing it. Because of this all the available memory space resides at the same place together, which means that the freely/unused available memory partitions are not distributed in a random fashion here and there across the whole memory space .

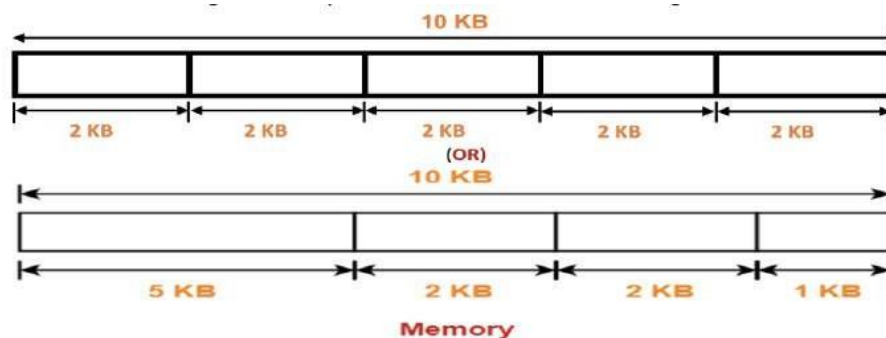


- The main memory is a combination of two main portions- one for the operating system and other for the user program.
- We can implement/achieve contiguous memory allocation by dividing the memory partitions into fixed size partitions and variable size partitions .



(a) Fixed-size Partition Scheme:

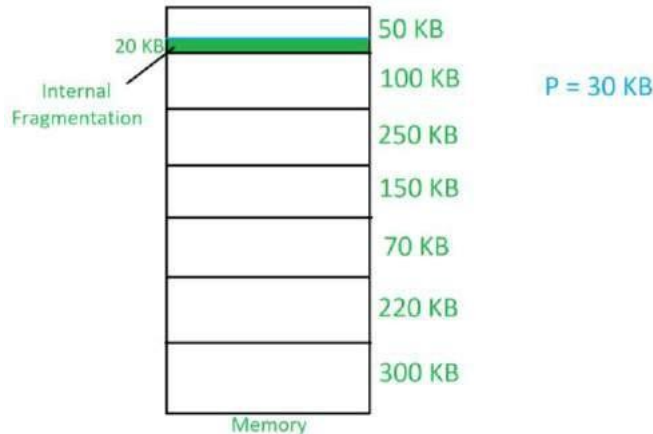
- This technique is also known as **Static partitioning**.
- In this technique, main memory is pre-divided into fixed size partitions.
- The partitions may or may not be the same size.
- The size of each partition is fixed and cannot be changed.
- Each partition is allowed to store only one process.



In this partition scheme, there is a problem that this technique will limit the degree of multiprogramming because the number of partitions will basically decide the number of processes.

Whenever any process terminates and get deallocated then the partition becomes available for another process.

If there is some wastage inside the partition then it is termed **Internal Fragmentation**.

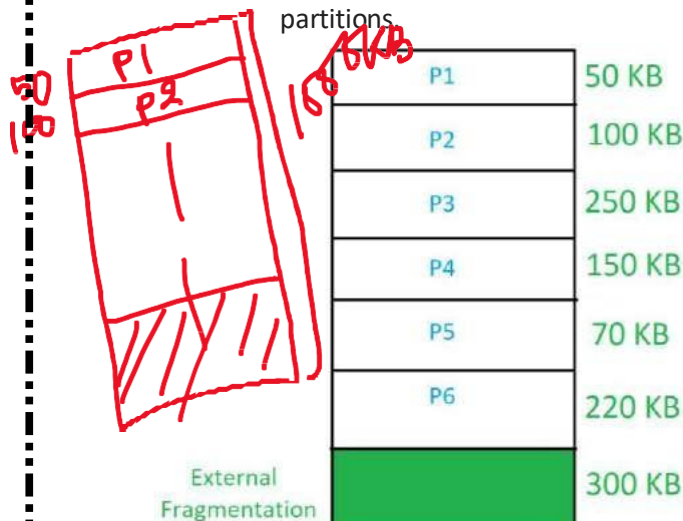


Internal Fragmentation is found in fixed partition scheme.

To overcome the problem of internal fragmentation, instead of fixed partition scheme, variable partition scheme is used.

(b) Variable size partition Scheme:

- This technique is also known as **Dynamic partitioning**.
- It performs the allocation dynamically.
- When a process arrives, a partition of size equal to the size of process is created.
- Then, that partition is allocated to the process.
- If smaller processes keep on coming then the larger partitions will be made into smaller partitions.



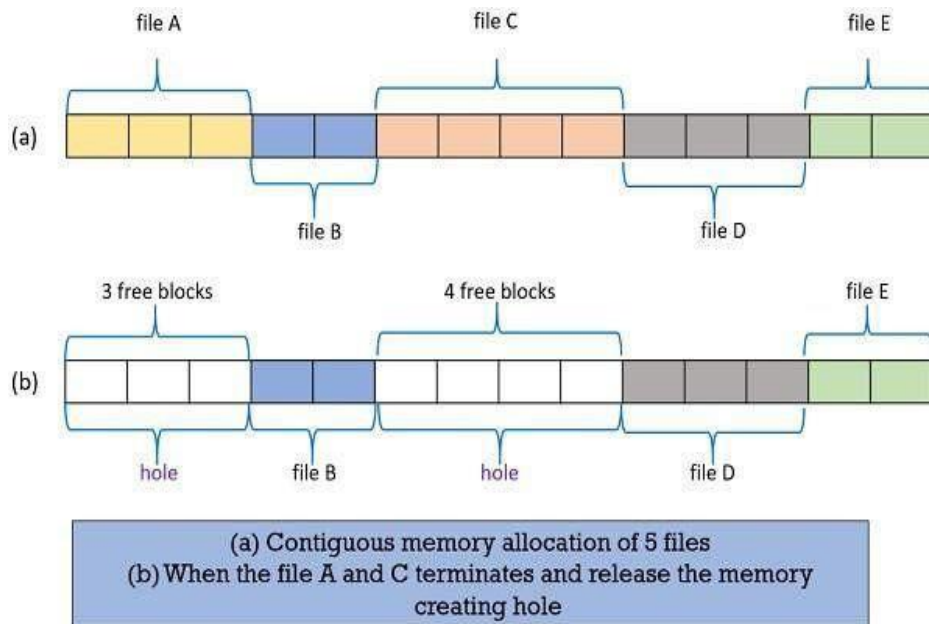
External Fragmentation is found in variable partition scheme.

To overcome the problem of external fragmentation, compaction technique is used or non-contiguous memory management techniques are used.

(c) Partition Allocation Algorithms:

- The processes arrive and leave the main memory.
- As a result, holes of different size are created in the main memory.
- These holes are allocated to the processes that arrive in future.

*holes- free/available space in memory



Partition allocation algorithms are used to decide which hole should be allocated to the arrived process.



1. First Fit Algorithm

First Fit algorithm scans the linked list and whenever it finds the first big enough hole to store a process, it stops scanning and load the process into that hole. This procedure produces two partitions. Out of them, one partition will be a hole while the other partition will store the process.

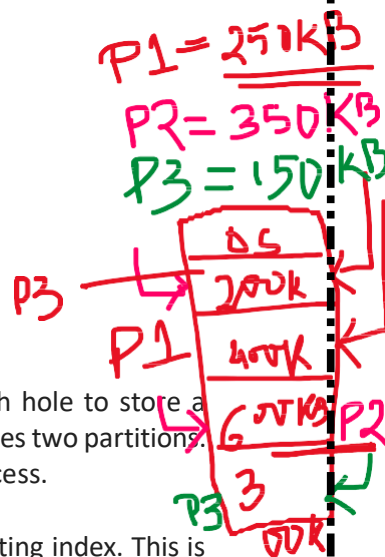
First Fit algorithm maintains the linked list according to the increasing order of starting index. This is the simplest to implement among all the algorithms and produces bigger holes as compare to the other algorithms.

2. Best Fit Algorithm

The Best Fit algorithm tries to find out the smallest hole possible in the list that can accommodate the size requirement of the process.

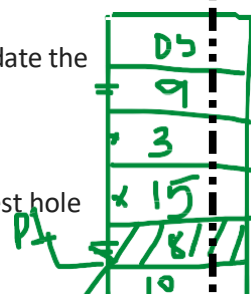
Using Best Fit has some disadvantages.

1. It is slower because it scans the entire list every time and tries to find out the smallest hole which can satisfy the requirement the process.



$$P_1 = 7KB$$

$$P_2 = 12KB$$



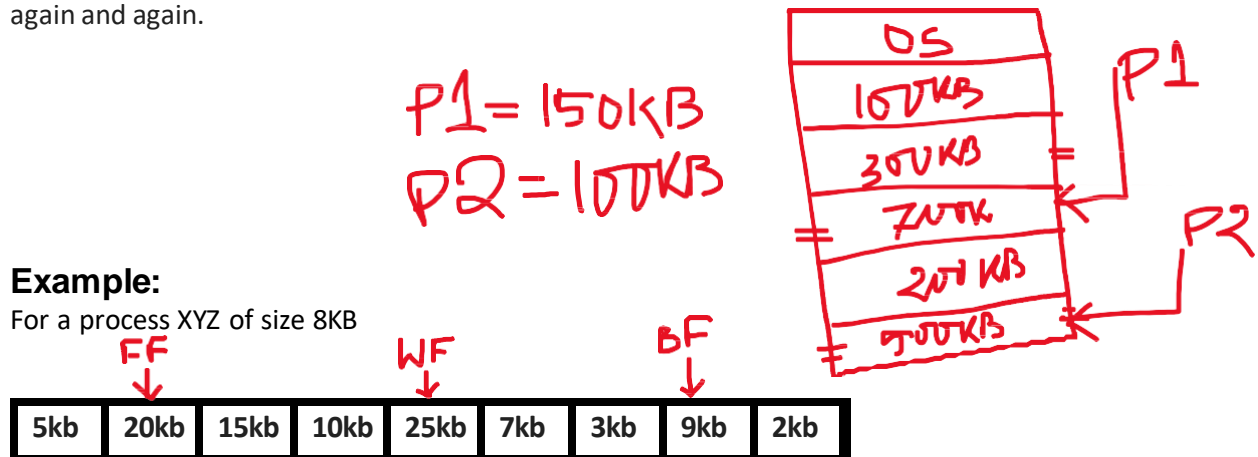
2. Due to the fact that the difference between the whole size and the process size is very small, the holes produced will be as small as it cannot be used to load any process and therefore it remains useless.

Despite of the fact that the name of the algorithm is best fit, It is not the best algorithm among all.

3. Worst Fit Algorithm

The worst fit algorithm scans the entire list every time and tries to find out the biggest hole in the list which can fulfill the requirement of the process.

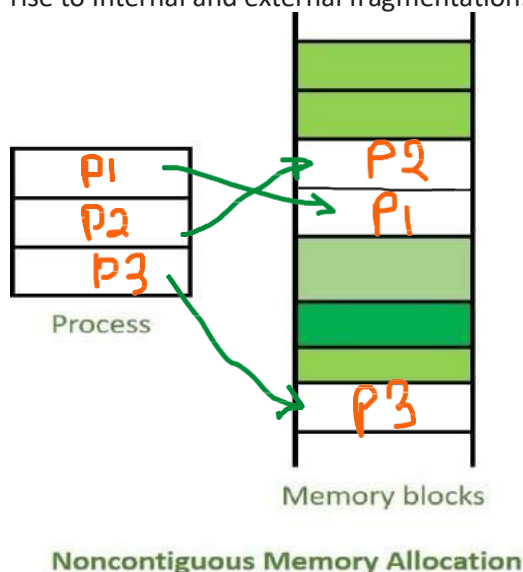
Despite of the fact that this algorithm produces the larger holes to load the other processes, this is not the better approach due to the fact that it is slower because it searches the entire list every time again and again.



3.1.2 Non-Contiguous Memory Allocation :

Non-Contiguous memory allocation is basically a method contrary to contiguous memory allocation method, allocates the memory space present in different locations to the process as per it's requirements.

This technique of memory allocation helps to reduce the wastage of memory, which eventually gives rise to Internal and external fragmentation.



Difference between Contiguous and Non-contiguous Memory Allocation :

Sl. No.	Contiguous Memory Allocation	Non-Contiguous Memory Allocation
1.	Contiguous memory allocation allocates consecutive blocks of memory to a file/process.	Non-Contiguous memory allocation allocates separate blocks of memory to a file/process.
2.	Faster in Execution.	Slower in Execution.
3.	It is easier for the OS to control.	It is difficult for the OS to control.
4.	Overhead is minimum as not much address translations are there while executing a process.	More Overheads are there as there are more address translations.
5.	Both Internal fragmentation and external fragmentation occurs in Contiguous memory allocation method.	External fragmentation occurs in Non-Contiguous memory allocation method.
6.	It includes single partition allocation and multi-partition allocation.	It includes paging and segmentation.
7.	Wastage of memory is there.	No memory wastage is there.
8.	In contiguous memory allocation, swapped-in processes are arranged in the originally allocated space.	In non-contiguous memory allocation, swapped-in processes can be arranged in any place in the memory.

3.1.3 Fragmentation

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types –

1. External fragmentation

Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.

External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block.

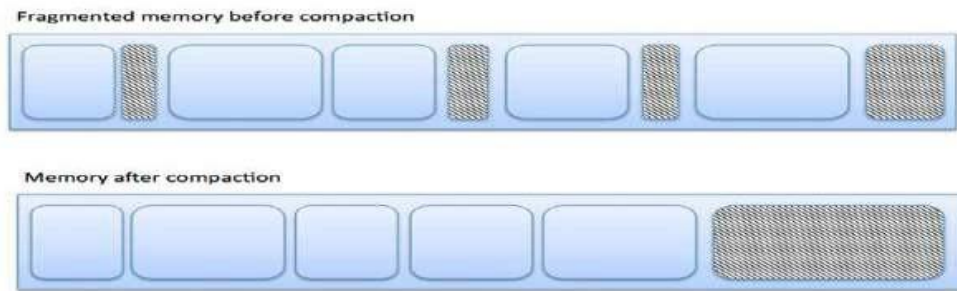
2. Internal fragmentation

Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

(a) 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 –

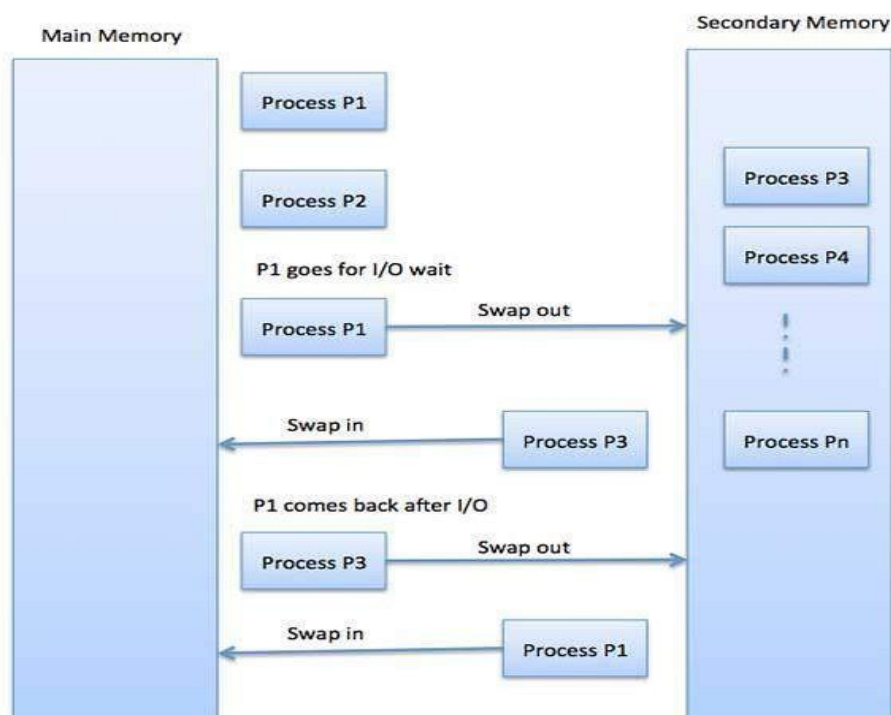


- Compaction is a process in which the free space is collected in a large memory chunk to make some space available for processes.
- In memory management, swapping creates multiple fragments in the memory because of the processes moving in and out.
- Compaction refers to combining all the empty spaces 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.

3.2 SWAPPING

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason **Swapping is also known as a technique for memory compaction.**



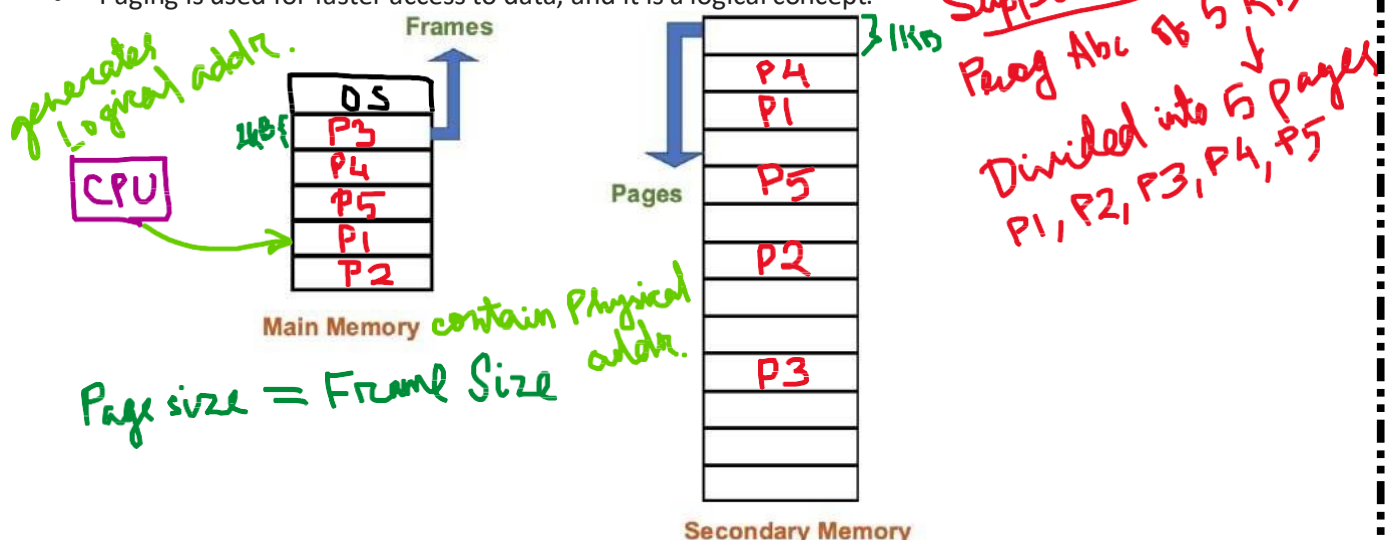
Variation in Swapping Method:

- Swapping of the processes also depends on the **priority-based pre-emptive scheduling**.
- Whenever a process with higher priority arrives the memory manager **swaps out** the process with the lowest priority to the disk and **swaps in** the process with the highest priority in the main memory for execution. When the highest priority process is finished, the lower priority process is swapped back in memory and continues to execute. This Variant of swapping is termed as **roll-out, roll-in**.

3.3 Paging, Segmentation, virtual memory using paging

3.3.1 PAGING

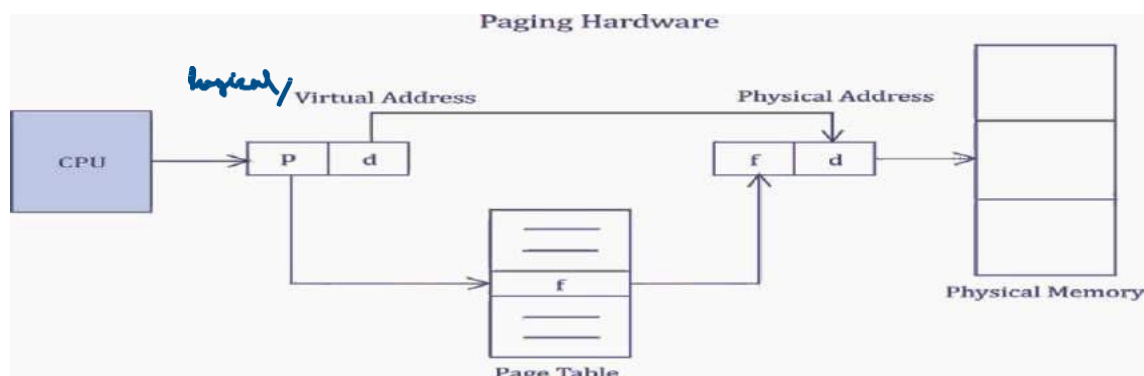
- **Paging** is a storage mechanism that allows OS to retrieve processes from the secondary storage into the main memory in the form of pages.
- In the Paging method, secondary memory is divided into small fixed-size blocks of logical memory, which is called **pages** and the main memory is divided into small fixed-size blocks of physical memory, which is called **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.
- Paging is used for faster access to data, and it is a logical concept.



(a) Translation of Logical Address into Physical Address

Some important points to note:

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

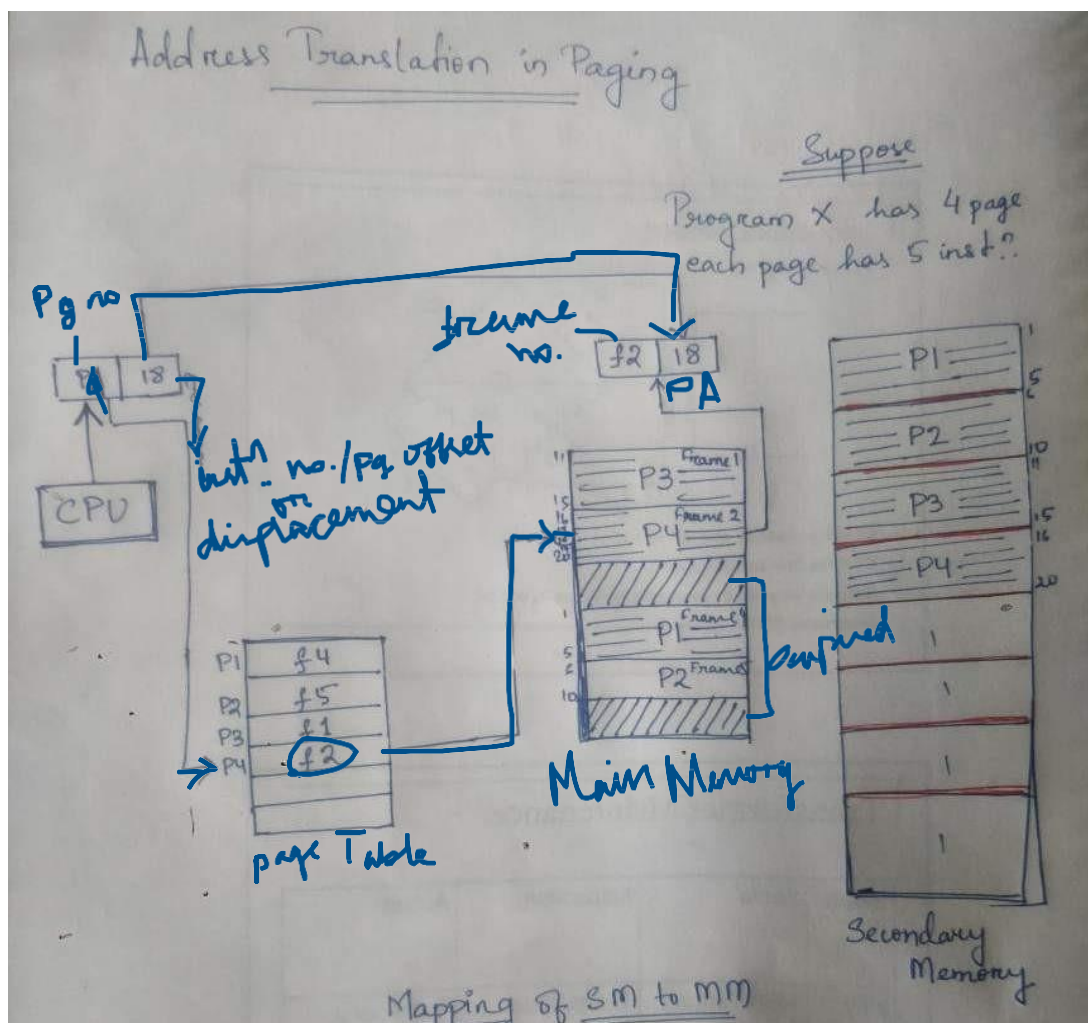


- The **logical address** generated by CPU always consists of two parts:
 - Page Number(p)
 - Page Offset (d)
 where, **Page Number** is used to specify the specific page of the process from which the CPU wants to read the data and it is also used as an index to the page table and **Page offset** is mainly used to specify the specific word(instruction or line of code) on the page that the CPU wants to read.
- The **physical address** consists of two parts:
 - Frame Number(f)
 - Page offset(d)
 where, The **Frame number** is used to indicate the specific frame where the required page is stored and **Page Offset** indicates the specific word that has to be read from that page.
- Page address is called logical address and represented by page number and the offset.

$$\text{Logical Address} = \text{Page number} + \text{page offset}$$
- Frame address is called physical address and represented by a frame number and the offset.

$$\text{Physical Address} = \text{Frame number} + \text{page offset}$$
- 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.

Paging Example



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.

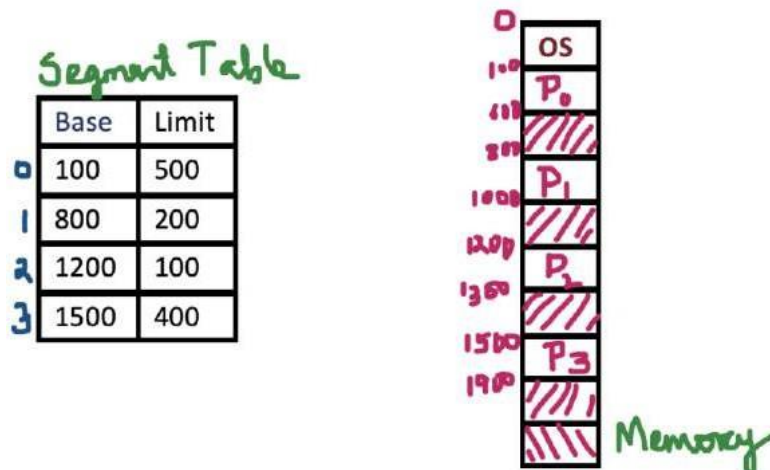
3.3.2 SEGMENTATION

In Operating Systems, Segmentation is a memory management technique in which, the memory is divided into the variable size parts. Each part is known as segment which can be allocated to a process.

- The details about each segment are stored in a table called as segment table. Segment table is stored in one (or many) of the segments.
- Segment table contains mainly two information about segment:

Base: It is the base address of the segment

Limit: It is the length of the segment.



Why Segmentation is required?

- Paging is more close to Operating system rather than the User. Operating system doesn't care about the User's view of the process. It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory. It decreases the efficiency of the system.
- It is better to have segmentation which divides the process into the segments. Each segment contain same type of functions such as main function can be included in one segment and the library functions can be included in the other segment,

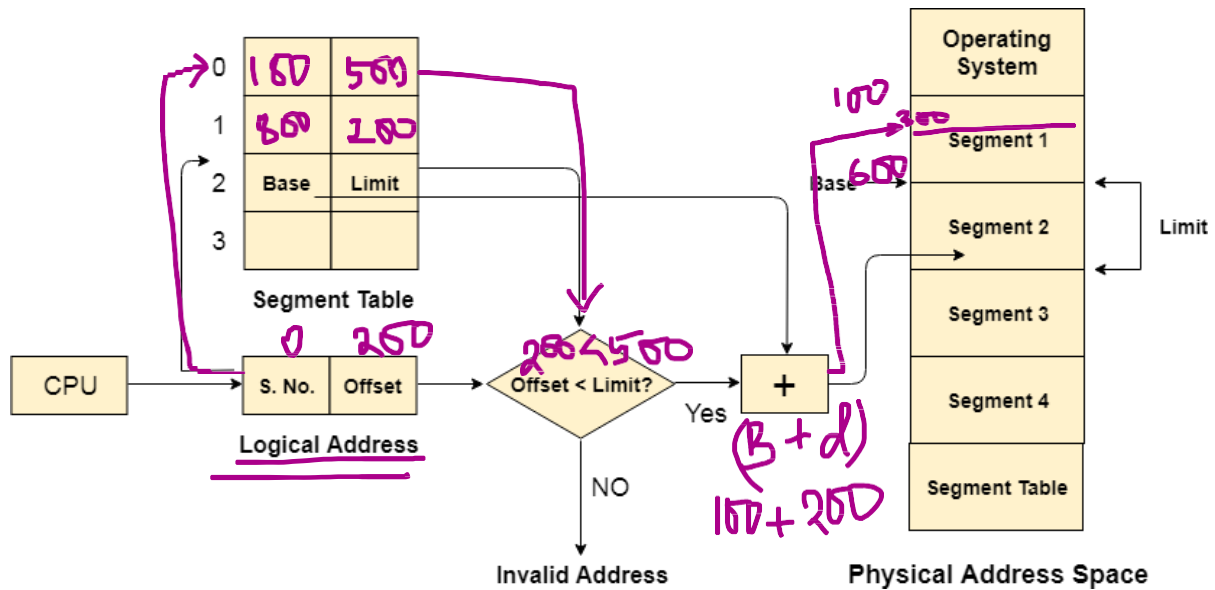
(a) Translation of Logical address into physical address by segment table

CPU generates a logical address which contains two parts:

- Segment Number
- Offset

The Segment number is mapped to the segment table. The limit of the respective segment is compared with the offset. If the offset is less than the limit then the address is valid otherwise it throws an error as the address is invalid.

In the case of valid address, the base address of the segment is added to the offset to get the physical address of actual word in the main memory.



Advantages of Segmentation

- No internal fragmentation
- Average Segment Size is larger than the actual page size.
- Less overhead
- It is easier to relocate segments than entire address space.
- The segment table is of lesser size as compare to the page table in paging.

Disadvantages of Segmentation

- It can have external fragmentation.
- it is difficult to allocate contiguous memory to variable sized partition.
- Costly memory management algorithms.

Difference between Paging and Segmentation:

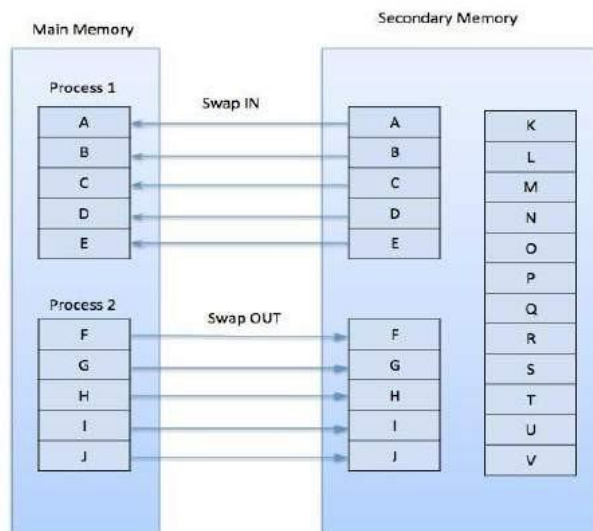
S.NO	Paging	Segmentation
1.	In paging, program is divided into fixed or mounted size pages.	In segmentation, program is divided into variable size sections.
2.	For paging, operating system is accountable.	For segmentation, compiler is accountable.
3.	Page size is determined by hardware.	Here, the section size is given by the user.
4.	It is faster in the comparison of segmentation.	Segmentation is slow.
5.	Paging could result in internal fragmentation.	Segmentation could result in external fragmentation.
6.	In paging, logical address is split into page number and page offset.	Here, logical address is split into section number and section offset.
7.	Paging comprises a page table which encloses the base address of every page.	While segmentation also comprises the segment table which encloses segment number and segment offset.

8.	Page table is employed to keep up the page data.	Section Table maintains the section data.
9.	In paging, operating system must maintain a free frame list.	In segmentation, operating system maintain a list of holes in main memory.
10.	Paging is invisible to the user.	Segmentation is visible to the user.
11.	In paging, processor needs page number, offset to calculate absolute address.	In segmentation, processor uses segment number, offset to calculate full address.

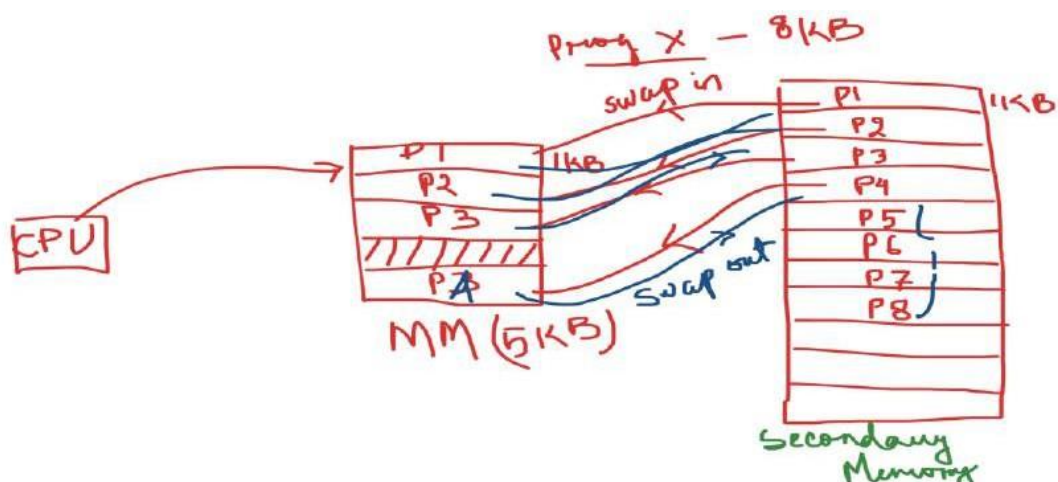
3.3.3 Virtual Memory using paging

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.



Example:



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.

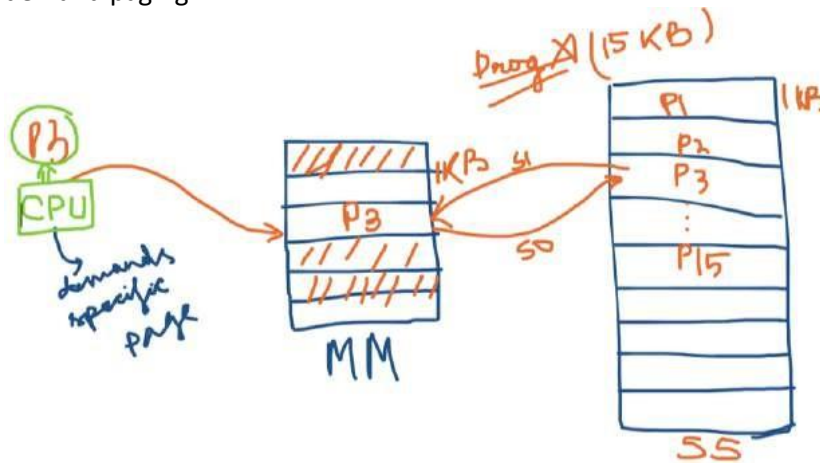
Disadvantages of Virtual Memory

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.

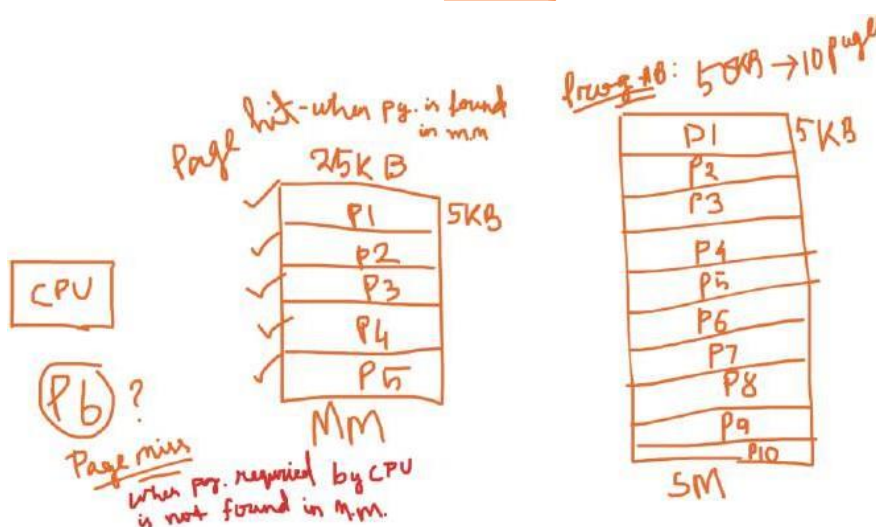
3.4 Demand paging, page fault handling

3.4.1 DEMAND PAGING :

The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging.

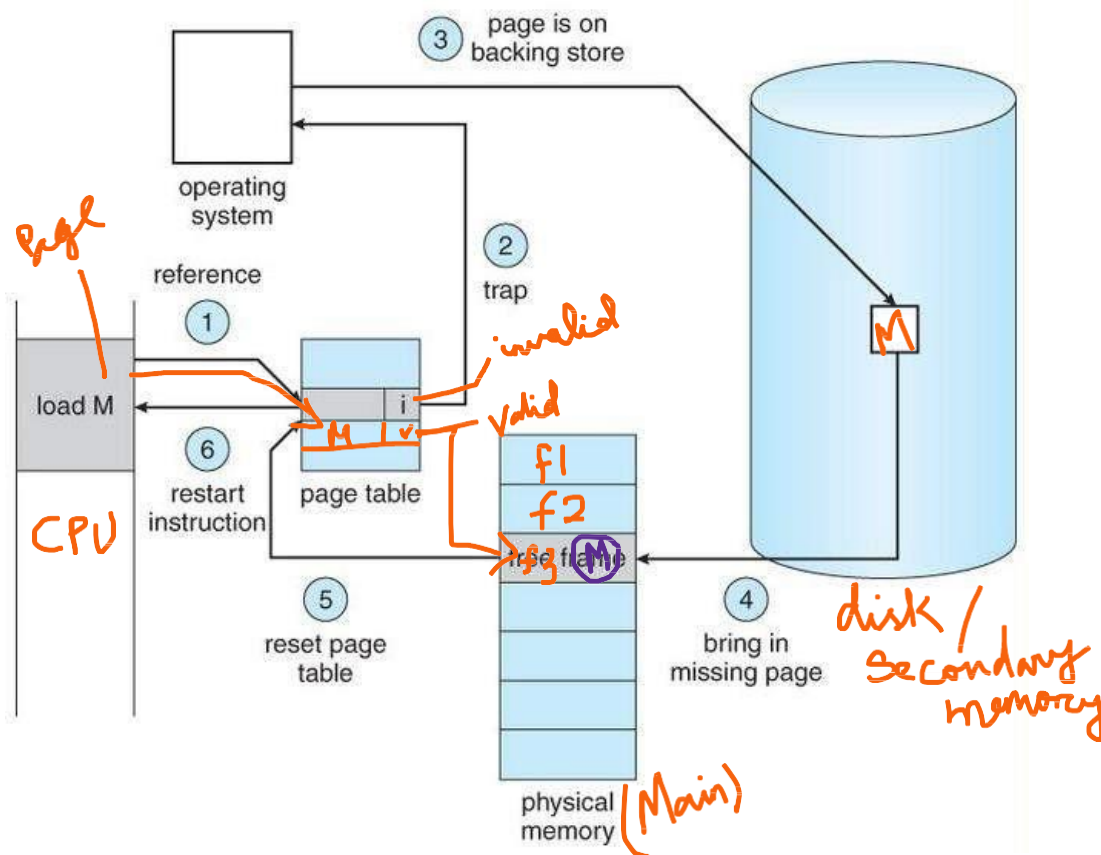


- A **page fault** occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM.



3.4.2 PAGE FAULT HANDLING:

So when page fault occurs then following sequence of events happens :



- 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 is scheduled, operating system returns to routine that called it.
- Assembly Routine reloads register and other state information, returns to user space to continue execution.

Chapter 4

DEVICE MANAGEMENT

4.1 Techniques for Device Management

- Dedicated,
- shared and
- virtual.

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

4.3 SPOOLING.

4.0 Introduction

Device management in operating system implies the management of the I/O devices such as a keyboard, magnetic tape, disk, printer, microphone, USB ports, scanner, camcorder etc.as well as the supporting units like control channels.

The main functions of device management in the operating system:

An operating system or the OS manages communication with the devices through their respective drivers. The operating system component provides a uniform interface to access devices of varied physical attributes. For device management in operating system:

- Keep tracks of all devices and the program which is responsible to perform this is called **I/O controller**.
- Monitoring the status of each device such as storage drivers, printers and other peripheral devices.
- Enforcing pre-set policies and taking a decision which process gets the device when and for how long.
- Allocates and Deallocates the device in an efficient way. De-allocating them at two levels: at the **process level** when I/O command has been executed and the device is temporarily released, and at the **job level**, when the job is finished and the device is permanently released.
- Optimizes the performance of individual devices.

Types of devices:

The OS peripheral devices can be categorized into three : **Dedicated, Shared, and Virtual**.

The differences among them are *the functions of the characteristics* of the devices as well as *how they are managed by the Device Manager*.

i. **Dedicated devices-**

Such type of devices in the device management in operating system are dedicated or assigned to only one job at a time until that job releases them. Devices like printers, tape drivers, plotters etc. demand such allocation scheme since it would be awkward if several users share them at the same point of time. The disadvantages of such kind of devices is the inefficiency resulting from the allocation of the device to a single user for the entire duration of job execution even though the device is not put to use 100% of the time.



ii. Shared devices-

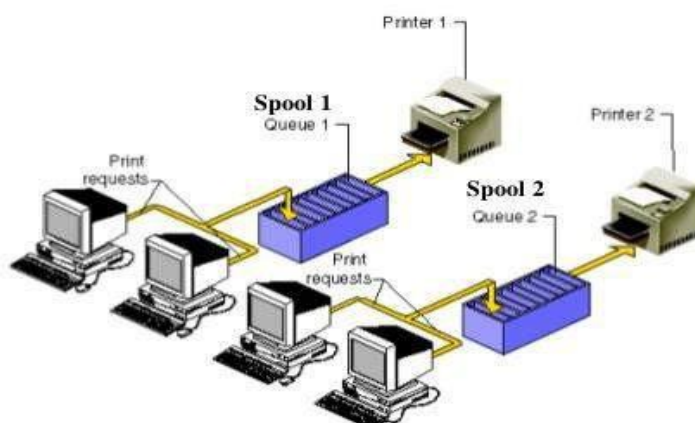
These devices can be allocated of several processes. Disk-DASD can be shared among several processes at the same time by interleaving their requests. The interleaving is carefully controlled by the Device Manager and all issues must be resolved on the basis of pre-determined policies.



iii. Virtual Devices-

These devices are the combination of the first two types and they are dedicated devices which are transformed into shared devices.

For example, a printer converted into a shareable device via spooling program which re-routes all the print requests to a disk. A print job is not sent straight to the printer, instead, it goes to the disk(spool)until it is fully prepared with all the necessary sequences and formatting, then it goes to the printers. This technique can transform one printer into several virtual printers which leads to better performance and use.



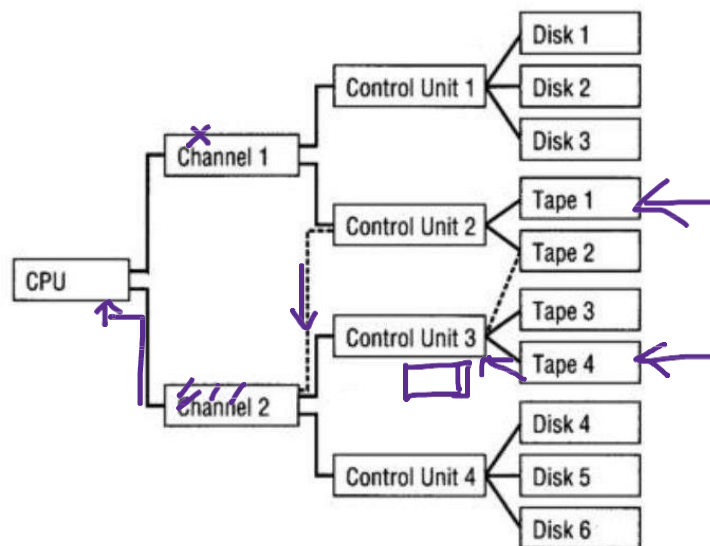
4.2 Device allocation considerations, I/O traffic control & I/O Schedule, I/O Device handlers

a. Speed Issues

- CPU operates at speeds much faster than the fastest I/O device
- Devices operate at different speeds
- Bursts of data
- Block data transfer required for some devices

b. Coordination

- Several devices perform I/O simultaneously
- Unexpected input
- Various input formats
- Status information needed for each device



Therefore the CPU makes a lot of **I/O requests** and the **I/O channels** work to coordinate the devices to synchronize the fast speed of the CPU with the slower speeds of **the I/O devices**. The **I/O control units** are attached to several similar devices and control these devices based on instructions from the channel.

I/O traffic control & I/O Scheduler, I/O Device handlers

Device Manager divides task into three parts, each handled by specific software component of I/O subsystem:

- **I/O traffic controller** watches status of all devices, control units, and channels
- **I/O scheduler** implements policies that determine allocation of, and access to, devices, control units, and channels
- **I/O device handler** performs actual transfer of data and processes the device interrupts

i. I/O traffic control

Three main tasks of I/O traffic controller:

- Determine if there's at least one path available
- If more than one path available, it must determine which to select
- If the paths are all busy, it must determine when one will become available

- Maintains a database containing status and connections for each unit

<u>Channel Control Block</u>	<u>Control Unit Control Block</u>	<u>Device Control Block</u>
<ul style="list-style-type: none">• Channel identification	<ul style="list-style-type: none">• Control Unit identification	<ul style="list-style-type: none">• Device identification
<ul style="list-style-type: none">• Status	<ul style="list-style-type: none">• Status	<ul style="list-style-type: none">• Status
<ul style="list-style-type: none">• List of control units connected to it	<ul style="list-style-type: none">• List of channels connected to it	<ul style="list-style-type: none">• List of control units connected to it
<ul style="list-style-type: none">• List of processes waiting for it	<ul style="list-style-type: none">• List of devices connected to it	<ul style="list-style-type: none">• List of processes waiting for it
	<ul style="list-style-type: none">• List of processes waiting for it	

ii. I/O Scheduler:

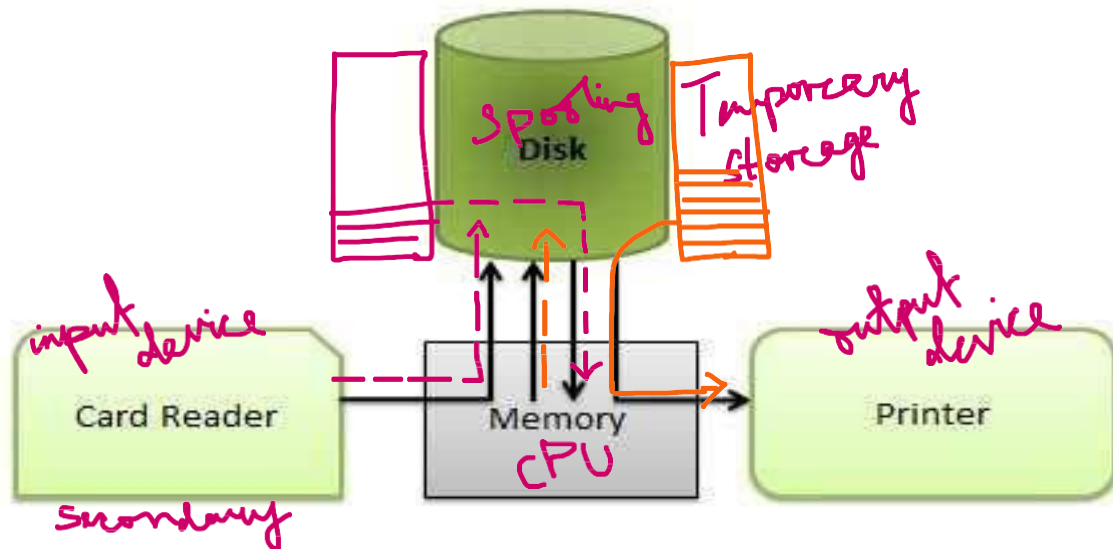
- When number of requests is greater than number of available paths, I/O scheduler must decide which request will be satisfied first based on different criteria
- I/O requests are not pre-empted

iii. I/O device handler:

- Provides detailed scheduling algorithms, which are extremely device dependent
- Each type of I/O device has its own device handler algorithm

4.3 SPOOLING

- SPOOL is an acronym for **Simultaneous Peripheral Operations On-Line**.
- It is a kind of buffering mechanism or a process in which data is temporarily held to be used and executed by a device, program or the system.
- Data is sent to and stored in memory or other volatile storage until the program or computer requests it for execution.
- In a computer system peripheral equipment, such as printers and punch card readers etc. (batch processing), are very slow relative to the performance of the rest of the system. Getting input and output from the system was quickly seen to be a bottleneck. Here comes the need for spool.
- Spooling works like a typical request queue where data, instructions and processes from multiple sources are accumulated for execution later on.
- Generally, it is maintained on computer's physical memory, buffers or the I/O device-specific interrupts.
- The spool is processed in FIFO manner i.e. whatever first instruction is there in the queue will be popped and executed.



Applications/Implementations of Spool:

- 1) The most common can be found in I/O devices like keyboard printers and mouse. For example, in printer, the documents/files that are sent to the printer are first stored in the memory or the printer spooler. Once the printer is ready, it fetches the data from the spool and prints it.

Even experienced a situation when suddenly for some seconds your mouse or keyboard stops working? Meanwhile, we usually click again and again here and there on the screen to check if its working or not. When it actually starts working, what and wherever we pressed during its hang state gets executed very fast because all the instructions got stored in the respective device's spool.

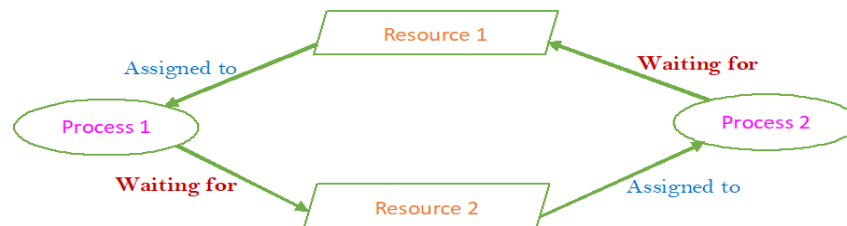
- 2) A batch processing system uses spooling to maintain a queue of ready-to-run jobs which can be started as soon as the system has the resources to process them.
- 3) Spooling is capable of overlapping I/O operation for one job with processor operations for another job. i.e. multiple processes can write documents to a print queue without waiting and resume with their work.
- 4) E-mail: an email is delivered by a MTA (Mail Transfer Agent) to a temporary storage area where it waits to be picked up by the MA (Mail User Agent).
- 5) Can also be used for generating Banner pages (e.g. the originator of the print requests by username, an account number. Such pages are used in office environments where many people share the small number of available resources).

Difference between Spooling and Buffering:

Definition	Spooling	Buffering
Basic Difference	It overlap the input/output of one job with the execution of another job.	It overlaps the input/output of one job with the execution of the same job.
Efficiency	Spooling is more efficient than buffering.	Buffering is less efficient than spooling.
Consider Size	It considers disk as a huge spool or buffer.	Buffer is a limited area in main memory.

CHAPTER 5: DEADLOCK

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



EXPLANATION: 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.

5.1.1 NECESSARY CONDITIONS FOR DEADLOCK (i.e. 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 –

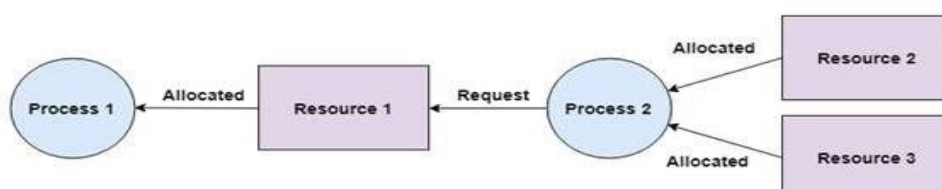
- **Mutual Exclusion**

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.



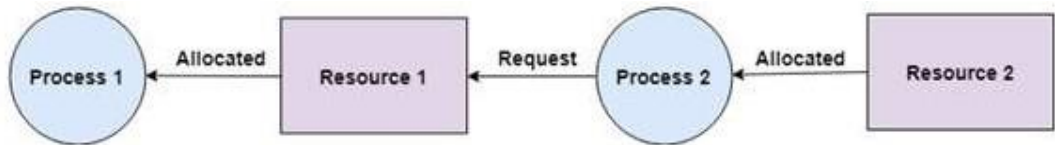
- **Hold and Wait**

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



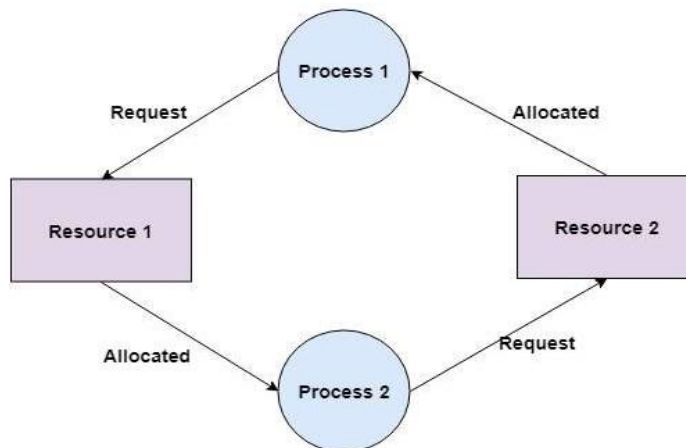
- **No Pre-emption**

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



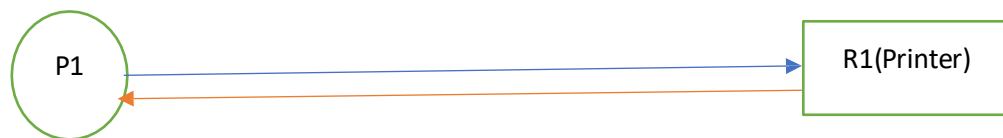
- **Circular Wait**

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

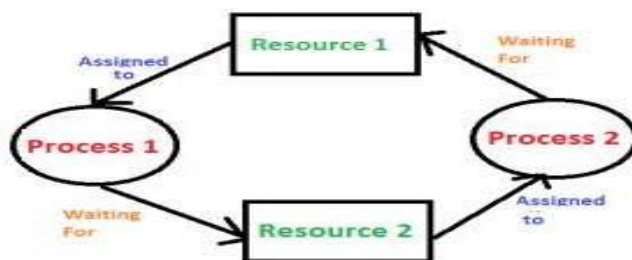
- For the purposes of deadlock discussion, a system can be modelled as a collection of limited resources($r_1, r_2, r_3, \dots, r_n$) which can be partitioned into different categories, to be allocated to a number of processes($P_1, P_2, P_3, \dots, P_n$) each having different needs.
- Resource categories 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 the file*.
 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 the deadlock is resolved.
3. **If resources have single instance:**
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 condition for deadlock.



In the above diagram, resource 1 and resource 2 have single instances. There is a cycle $R1 \rightarrow P1 \rightarrow R2 \rightarrow P2$. So, Deadlock is Confirmed.

4. 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 (RAG)

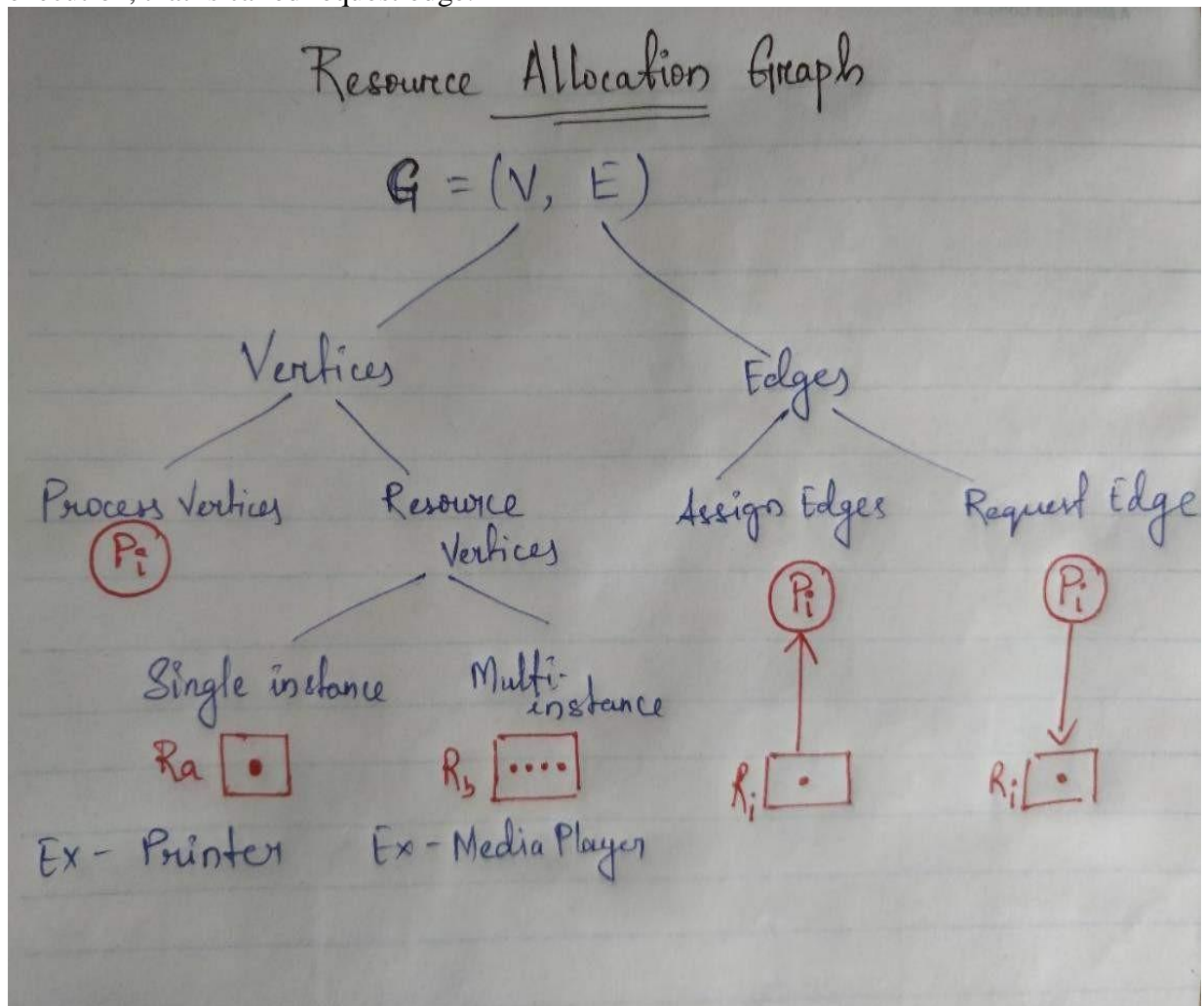
Resource allocation graph is explained to us to see what is the state of the system in terms of **processes and resources**.

We know that any graph contains vertices and edges i.e. $G = (v, e)$. So, RAG also contains vertices and edges. In RAG vertices are two type –

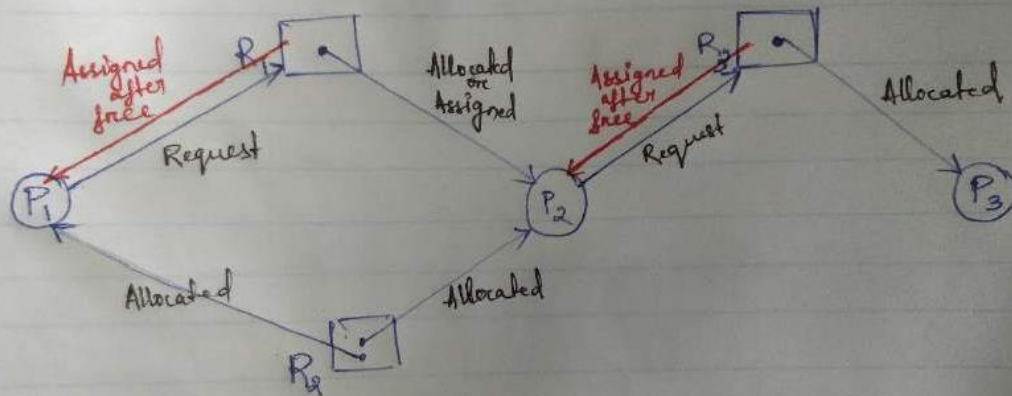
1. **Process vertex** – Every process will be represented as a process vertex. Generally, the process will be represented with a circle.
2. **Resource vertex** – Every resource will be represented as a resource vertex. It is also two type –
 - *Single instance type resource* – It represents as a box, inside the box, there will be one dot. So, the number of dots indicate how many instances are present of each resource type.
 - *Multiple instance type resource* – It also represents as a box, inside the box, there will be many dots present.

Now coming to the edges of RAG. There are two types of edges in RAG –

1. **Assign Edge** – If you already assigned a resource to a process then it is called *Assign edge*.
2. **Request Edge** – It means in future the process might want some resource to complete the execution, that is called request edge.

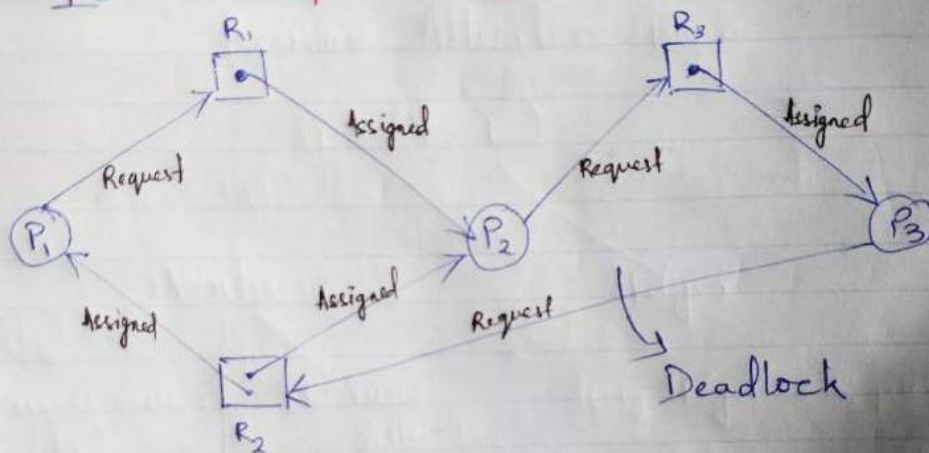


Example 1: A graph without Deadlock



- After completion of P_3 , R_3 will be released/free
- Now R_3 can be assigned to P_2
- After completion of P_2 , P_1 , P_2 & P_3 will be free
- So, now P_1 can complete. Hence, no deadlock.

Example 2: A Graph with Deadlock



- P_3 requires R_2 to complete execution
- P_2 has R_1 & R_2 but requires R_3 being used by P_3 so P_2 is dependent on P_3 to complete execution.
- P_1 has R_2 but requires R_3 being used by P_2 so P_1 is dependent on P_2 to complete execution. Hence it's a DEADLOCK.

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 resources when deadlocks are detected.
 3. **Ignore the problem all together** - If deadlocks only occur once a year or so, it may be better to simply let them happen and reboot as necessary than to incur the constant overhead and system performance penalties associated with deadlock prevention or detection. This is the approach that both Windows and UNIX take.
- In order to avoid deadlocks, the system must have additional information about all processes. In particular, the system must know what resources a process will or may request in the future. (Ranging from a simple worst-case maximum to a complete resource request and release plan for each process, depending on the particular algorithm.)
- Deadlock detection is fairly straightforward, but deadlock recovery requires either aborting processes or preempting resources, neither of which is an attractive alternative.
- If deadlocks are neither prevented nor detected, then when a deadlock occurs the system will gradually slow down, as more and more processes become stuck waiting for resources currently held by the deadlock and by other waiting processes. Unfortunately, this slowdown can be indistinguishable from a general system slowdown when a real-time process has heavy computing needs.

5.6 DEADLOCK PREVENTION

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

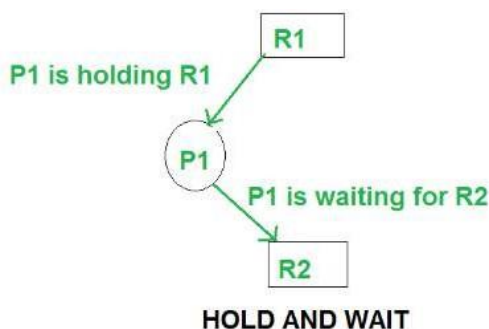
1. Eliminate Mutual Exclusion

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.



3. Eliminate No Pre-emption

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

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.

5.6.1 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.)

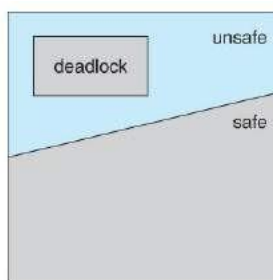


Figure 7.6 - Safe, unsafe, and deadlocked state spaces.

Cont....

NOTE: Deadlock prevention ensures that at least one of the necessary conditions to cause deadlock will never occur while deadlock avoidance ensures that the system never enters an unsafe state.

5.6.2 BANKER'S ALGORITHM

Deadlock avoidance can be done with Banker's Algorithm.

Banker's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it doesn't allow the request made by the process.

Banker's Algorithm Notations

Following **Data structures** are used to implement the Banker's Algorithm:

Let '**n**' be the number of processes in the system and '**m**' be the number of resources types.

Available :

- It is a 1-d array of size '**m**' indicating the number of available resources of each type.
- $\text{Available}[j] = k$ means there are '**k**' instances of resource type **R_j**

Max :

- It is a 2-d array of size '**n*m**' that defines the maximum demand of each process in a system.
- $\text{Max}[i, j] = k$ means process **P_i** may request at most '**k**' instances of resource type **R_j**.

Allocation :

- It is a 2-d array of size '**n*m**' that defines the number of resources of each type currently allocated to each process.
- $\text{Allocation}[i, j] = k$ means process **P_i** is currently allocated '**k**' instances of resource type **R_j**

Need :

- It is a 2-d array of size '**n*m**' that indicates the remaining resource need of each process.
- $\text{Need}[i, j] = k$ means process **P_i** currently need '**k**' instances of resource type **R_j**

for its execution.

- $\text{Need}[i, j] = \text{Max}[i, j] - \text{Allocation}[i, j]$

Allocation_i specifies the resources currently allocated to process **P_i** and Need_i specifies the additional resources that process **P_i** may still request to complete its task.

Banker's algorithm consists of Safety algorithm and Resource request algorithm.

5.6.3.1 RESOURCE REQUEST ALGORITHM

Resource request algorithm enables you to represent the system behaviour when a specific process makes a resource request.

Let understand this by the following steps:

Step 1) When a total requested instance of all resources is lesser than the process, move to step 2.

Step 2) When a requested instance of each and every resource type is lesser compared to the available resources of each type, it will be processed to the next step. Otherwise, the process requires to wait because of the unavailability of sufficient resources.

Step 3) Resource is allocated as shown in the below given Pseudocode.

Available = Available – Request (y)

Allocation(x) = Allocation(x) + Request(x)

Need(x) = Need(x) - Request(x)

This final step is performed because the system needs to assume that resources have been allocated. So that there should be less resources available after allocation.

Characteristics of Banker's Algorithm

Here are important characteristics of banker's algorithm:

- Keep many resources that satisfy the requirement of at least one client
- Whenever a process gets all its resources, it needs to return them in a restricted period.
- When a process requests a resource, it needs to wait
- The system has a limited number of resources
- Advance feature for max resource allocation

Disadvantage of Banker's algorithm

Here, are cons/drawbacks of using banker's algorithm

- Does not allow the process to change its Maximum need while processing
- It allows all requests to be granted in restricted time, but one year is a fixed period for that.
- All processes must know and state their maximum resource needs in advance.

Summary:

- Banker's algorithm is used majorly in the banking system to avoid deadlock. It helps you to identify whether a loan will be given or not.
- Notations used in banker's algorithms are 1) Available 2) Max 3) Allocation 4) Need
- Resource request algorithm enables you to represent the system behavior when a specific process makes a resource request.
- Banker's algorithm keeps many resources that satisfy the requirement of at least one client

- The biggest drawback of banker's algorithm is that it does not allow the process to change its Maximum need while processing.

Note: Deadlock prevention is more strict than Deadlock Avoidance.

5.6.3.2 SAFETY ALGORITHM

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

- 1) Let Work and Finish be vectors of length 'm' and 'n' respectively.

Initialize: Work = Available

Finish[i] = false; for i = 1, 2, 3, 4....n

- 2) Find an i such that both

a) Finish[i] = false

b) Need_i ≤ Work

if no such i exists goto step (4)

- 3) Work = Work + Allocation[i]

Finish[i] = true

goto step (2)

- 4) if Finish [i] = true for all i
then the system is in a safe state

5.6.4 DEADLOCK RECOVERY

A traditional operating system such as Windows doesn't deal with deadlock recovery as it is time and space consuming process. Real-time operating systems use Deadlock recovery.

Recovery method

- Raise an alarm- signals or tells the user and administrator.
- Roll back- Checkpoints states where the states are stored in the disk at regular interval and then roll back is performed from the recently checkpoint state when deadlock occurs.
- Killing the process: killing all the process involved in the deadlock. Killing process one by one. After killing each process check for deadlock again keep repeating the process till system recover from deadlock.
- Resource Pre-emption: Resources are pre-empted from the processes involved in the deadlock, pre-empted resources are allocated to other processes so that there is a possibility of recovering the system from deadlock. In this case, the system goes into starvation.

Deadlock Recovery

What should the OS do when it detects a deadlock?

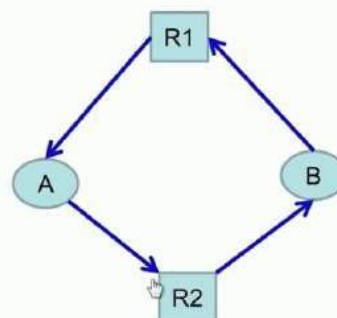
- **Raise an alarm**
 - Tell users and administrator
- **Preemption**
 - Take away a resource temporarily (frequently not possible)
- **Rollback**
 - Checkpoint states and then rollback
- **Kill process**
 - Keep killing processes until deadlock is broken



Deadlock Recovery

What should the OS do when it detects a deadlock?

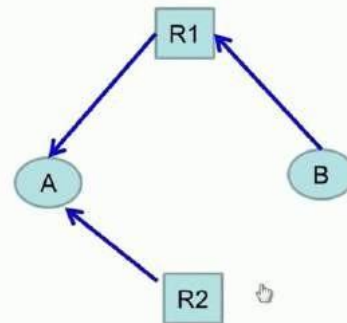
- **Raise an alarm**
 - Tell users and administrator
- **Preemption**
 - Take away a resource temporarily (frequently not possible)
- **Rollback**
 - Checkpoint states and then rollback
- **Kill process**
 - Keep killing processes until deadlock is broken



Deadlock Recovery

What should the OS do when it detects a deadlock?

- Raise an alarm
 - Tell users and administrator
- Preemption
 - Take away a resource temporarily (frequently not possible)
- Rollback
 - Checkpoint states and then rollback
- Kill process
 - Keep killing processes until deadlock is broken

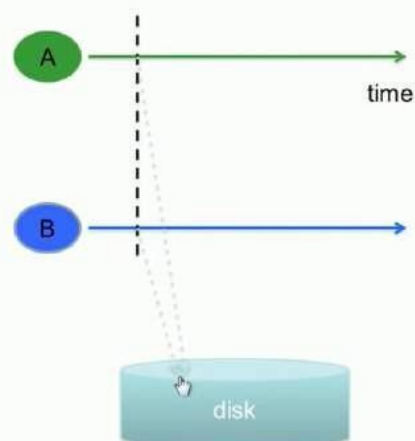


<https://www.youtube.com/watch?v=9j05ng2274>

Deadlock Recovery

What should the OS do when it detects a deadlock?

- Raise an alarm
 - Tell users and administrator
- Preemption
 - Take away a resource temporarily (frequently not possible)
- Rollback
 - Checkpoint states and then rollback
- Kill process
 - Keep killing processes until deadlock is broken

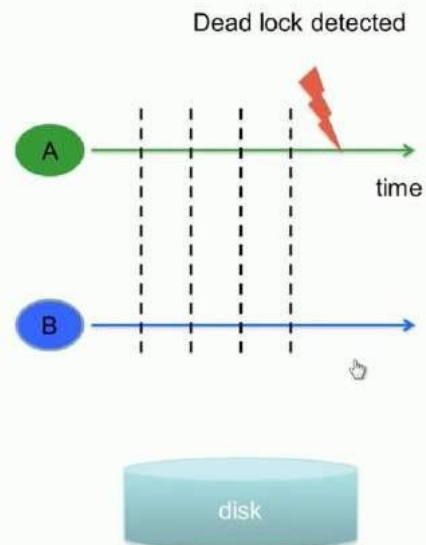


130

Deadlock Recovery

What should the OS do when it detects a deadlock?

- Raise an alarm
 - Tell users and administrator
- Preemption
 - Take away a resource temporarily (frequently not possible)
- Rollback
 - Checkpoint states and then rollback
- Kill process
 - Keep killing processes until deadlock is broken

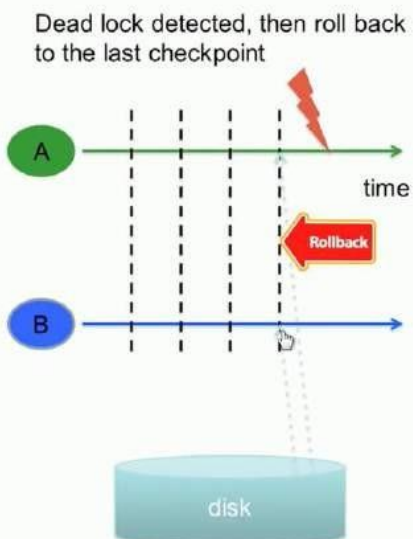


131

Deadlock Recovery

What should the OS do when it detects a deadlock?

- Raise an alarm
 - Tell users and administrator
- Preemption
 - Take away a resource temporarily (frequently not possible)
- Rollback
 - Checkpoint states and then rollback
- Kill process
 - Keep killing processes until deadlock is broken

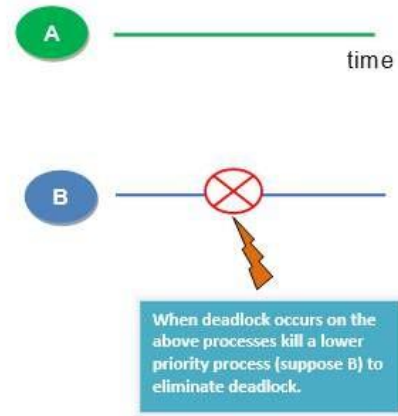


132

Deadlock Recovery

What should the OS do when it detects a deadlock?

- Raise an alarm
 - tells users and the administrators
- Preemption
 - take away a resource temporarily (frequently not possible)
- Rollback
 - Checkpoint states and then rollback
- Kill a process
 - keep killing processes till deadlock is broken



CHAPTER-6

FILE MANAGEMENT

6.0 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 secondary storage.

Or file is a collection of logically related entities.

6.1 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

6.1.1 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:

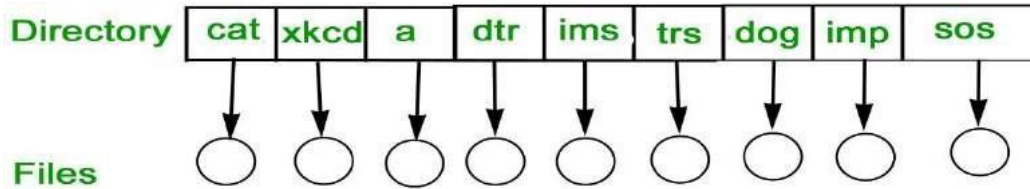
- **Efficiency:** A file can be located more quickly.
- **Naming:** It becomes convenient for users as two users can have same name for different files or may have different name for same file.
- **Grouping:** Logical grouping of files can be done by properties e.g. all java programs, all

games etc.

1. SINGLE-LEVEL DIRECTORY

In this a single directory is maintained for all the users.

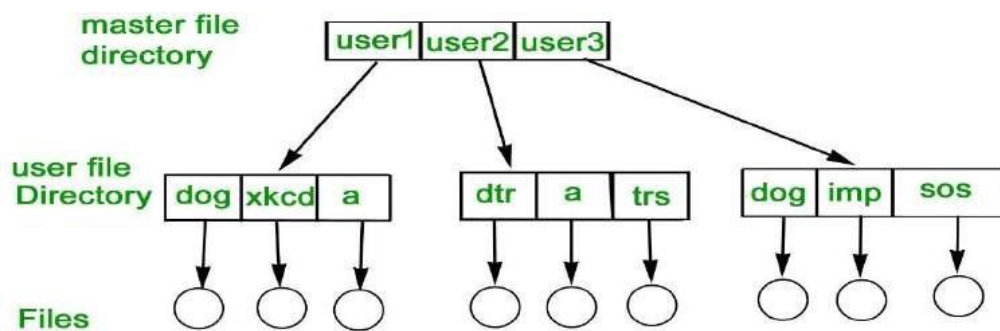
- **Naming problem:** Users cannot have same name for two files.
- **Grouping problem:** Users cannot group files according to their need.



2. TWO-LEVEL DIRECTORY

In this, separate directories for each user is maintained.

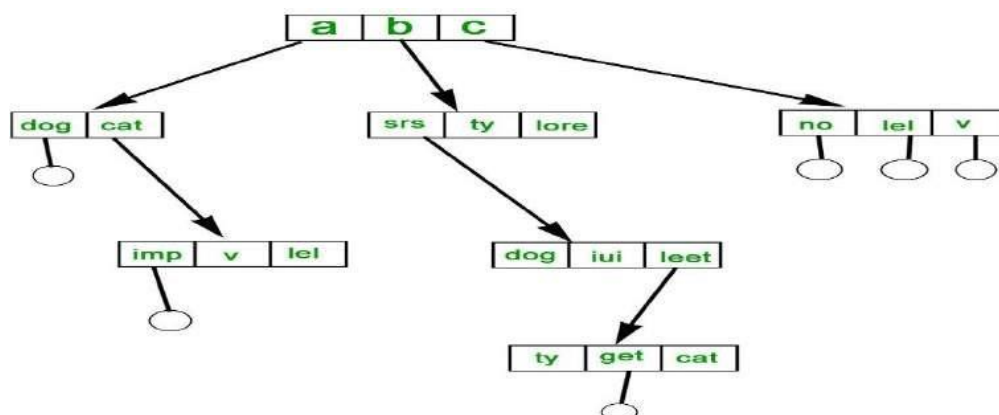
- **Path name:** Due to two levels there is a path name for every file to locate that file.
- Now, we can have same file name for different user.
- Searching is efficient in this method.



3. TREE-STRUCTURED DIRECTORY:

Directory is maintained in the form of a tree. Searching is efficient and also there is grouping capability. 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.



6.1.2 FILE STRUCTURE

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

- 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 by the 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.

6.1.3 FILE SHARING

Allowing users to share files raises a major issue: protection. A general approach 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 system recognizes three classifications of users with each file and directory:

- user
- group
- other

6.2.1 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 –

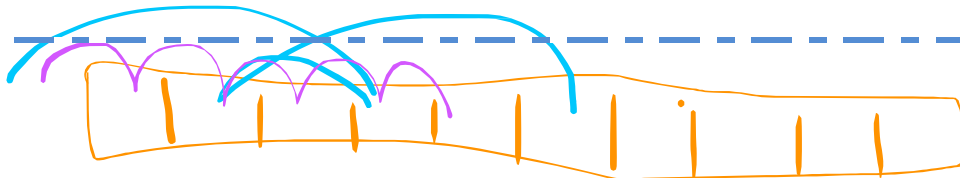
- Sequential access
- Direct/Random access
- Indexed sequential access

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.

6.2.2 FILE SYSTEMS

A file system provides a mapping between the logical and physical views of a file, through a set of services and an interface. Simply put, the file system hides all the device specific aspects of file manipulation from users.

The basic services of a file system include:

- keeping track of files (knowing location),
- I/O support, especially the transmission mechanism to and from main memory,
- management of secondary storage,
- sharing of I/O devices,
- providing protection mechanisms for information held on the system.

6.2.3 RELIABILITY

Regular file operations often involve changing several disk blocks. For example, What can 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 of failure.

6.3 FILE ALLOCATION METHODS

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

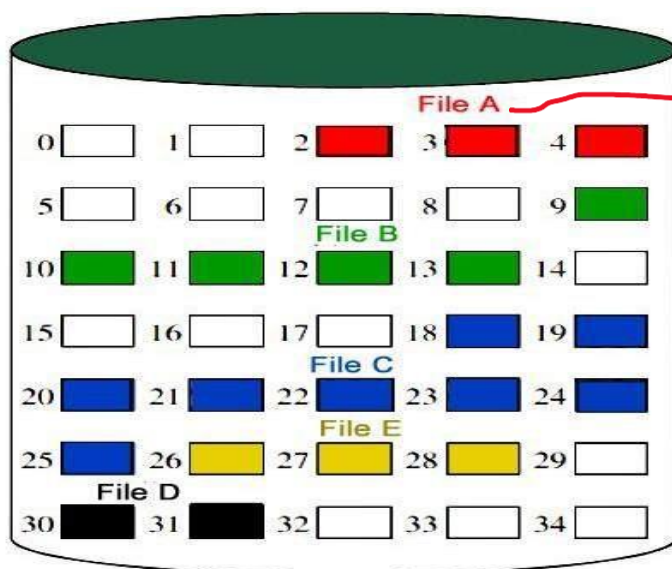
- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

The main idea behind these methods is to provide:

- Efficient disk space utilization.
- Fast access to the file blocks.

1. CONTINUOUS ALLOCATION:

- Each file occupies a contiguous address space on disk.
- Assigned disk address is in linear order.



File allocation table

File name	Start block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Advantages

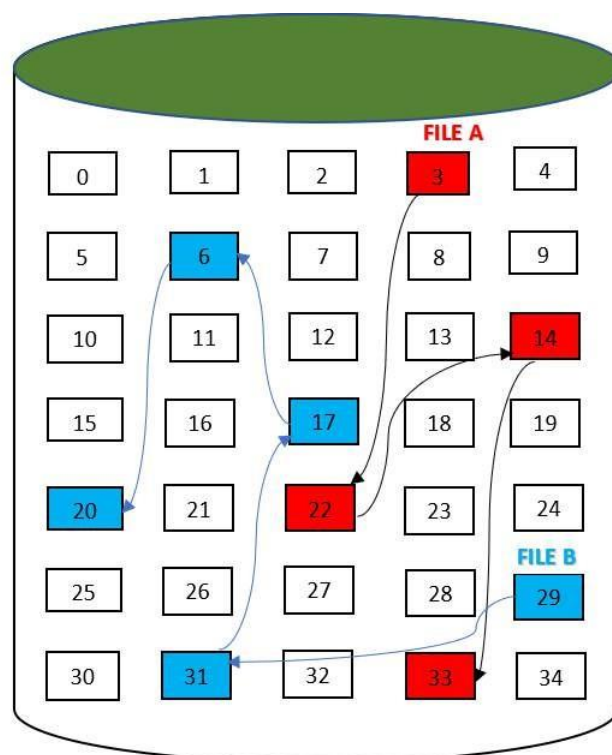
- It is simple
- Easy to implement.
- We will get Excellent read performance.
- Supports Random Access into files.

Disadvantage

- External fragmentation is a major issue with this type of allocation technique.
- It may be difficult to have a file grow.

2. LINKED ALLOCATION (NON-CONTIGUOUS ALLOCATION):

- This method solves the problems associated with contiguous allocation. Here the blocks of a single file can be scattered anywhere on the disk.
- The entire file is implemented as a Linked List. Each file is a linked list of disk blocks.
- The directory maintained by the Operating System contains the starting block address.
- Each block of a file contains a pointer to the next block after it in the list.
- For creating a new file, we need to just create a new entry in the directory and not to search for sufficient space as in contiguous.



File Allocation Table

File Name	Start Block	Length	✓ Linked Blocks
File A	3	4	3→22→14→33
File B	29	5	29→31→17→6→20

Advantages

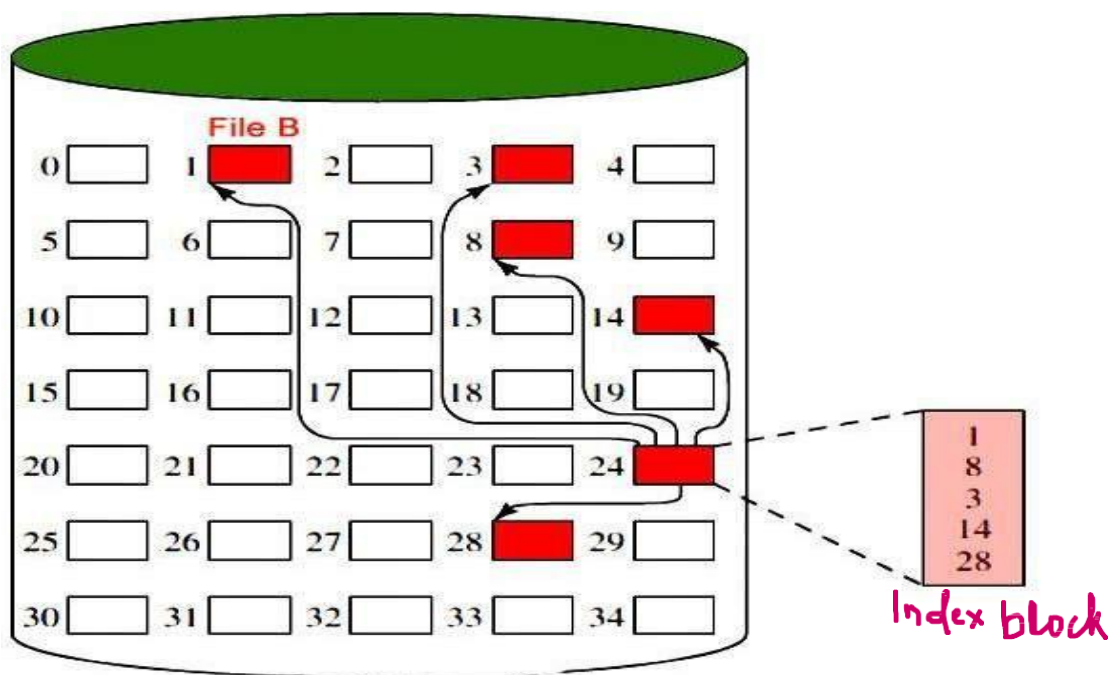
- No external fragmentation
- Effectively used in sequential access file.
- Any free block can be utilized in order to satisfy the file block requests.
- File can continue to grow as long as the free blocks are available.

Disadvantage:

- There is an overhead of maintaining the pointer in every disk block.
- If the pointer of any disk block is lost, the file will be truncated.
- Does not support direct access of file.

3. INDEXED ALLOCATION:

- Provides solutions to problems of contiguous and linked allocation.
- An index block is created having all pointers to files.
- Each file has its own index block which stores the addresses of disk space occupied by the file.
- Directory contains the addresses of index blocks of files.



File allocation table

File name	Index block
• • • File B • • •	• • • 24 • • •

Advantages

- Supports direct access
- A bad data block causes the loss of only that block.

Disadvantages

- A bad index block could cause the loss of entire file.

- Size of a file depends upon the number of pointers, an index block can hold.
- Having an index block for a small file is totally wastage.
- More pointer overhead.

6.4.1 FILE PROTECTION

File protection is provided by the operating system, which can designate files as read only. This allows both regular (read/write) and read only files to be stored on the same disk volume. Files can also be designated as hidden files, which makes them invisible to most software programs.

A computer file needs protection which is of two types

1. Reliability

- Protection from physical damage. File systems can be damaged by
 - Hardware problems (errors in r/w)
 - Power surges or failures
 - Head crashes
 - Dirt and temperature
 - Bugs in file system software

Reliability can be provided by

- duplicate copies of files
- Take backups at regular intervals (daily/weekly/monthly)

2. Security

- Protection from improper access
 - Protecting files from unauthorized access
 - More important in a multi user system
 - Provided by controlling access to files

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.

6.4.2 SECONDARY STORAGE MANAGEMENT

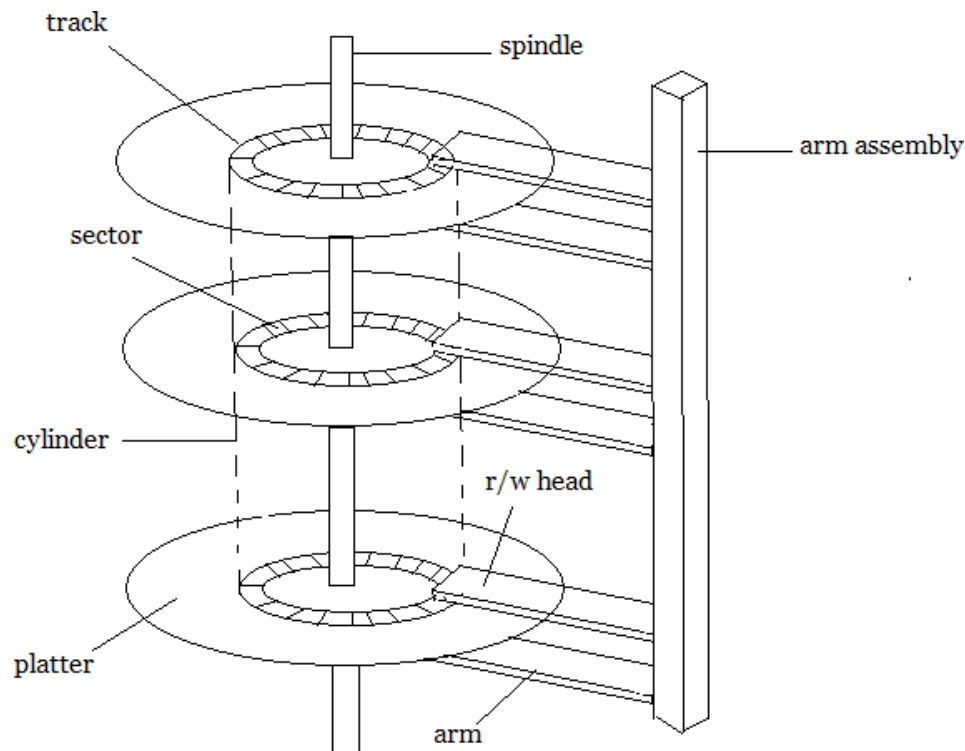
Brief:

Secondary storage devices are those devices whose memory is non volatile i.e., 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.

DISK SCHEDULING ALGORITHMS

On a typical multiprogramming system, there will usually be multiple disk access requests at any point of time. So those requests must be scheduled to achieve good efficiency. Disk scheduling is similar to process scheduling. Some of the disk scheduling algorithms are:

- First Come First Served (FCFS) or FIFO
- Shortest Service Time First (SSTF)
- SCAN—back and forth over disk
- CSCAN—circular SCAN or one way SCAN and fast return
- LOOK—look for a request before moving in that direction
- CLOOK—circular LOOK

SECONDARY STORAGE MANAGEMENT ISSUES

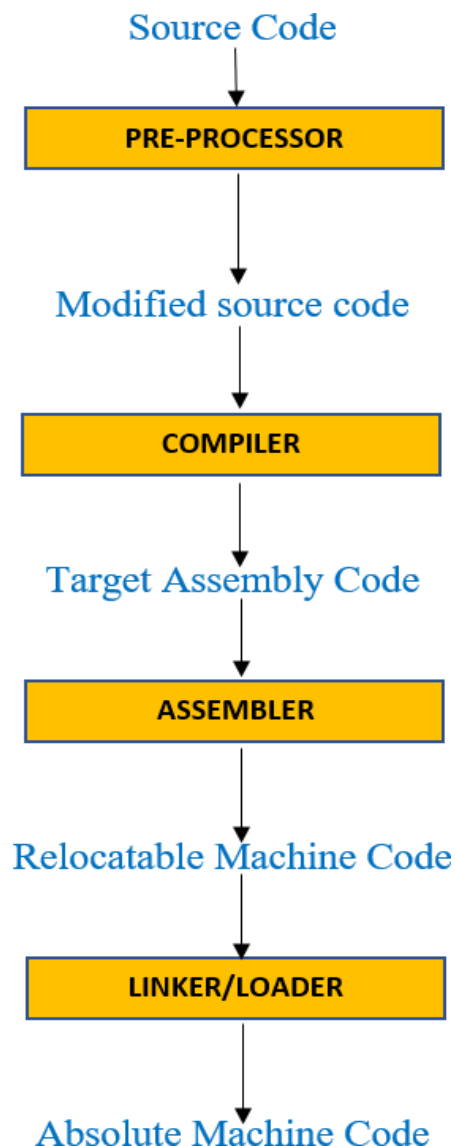
- Formatting
 - Physical: divide the blank slate into sectors identified by headers containing such information as sector number; sector interleaving
 - Logical: marking bad blocks; partitioning (optional) and writing a blank directory on disk; installing file allocation tables and other relevant information (file system initialization)
- Reliability
 - disk interleaving or striping
 - RAIDs (Redundant Array of Inexpensive Disks): various levels, e.g., level 0 is disk striping)
- Controller caches
 - newer disks have on-disk caches (128KB—512KB)

Chapter 7- SYSTEM PROGRAMMING

7.0 Concept: *System programming* involves designing and writing computer programs that allow the computer hardware to interface with the programmer and the user, leading to the effective execution of application software on the computer system.

Typical system programs include the operating system and firmware, programming tools such as compilers, assemblers, I/O routines, interpreters, scheduler, loaders and linkers as well as the runtime libraries of the computer programming languages.

CONTEXTS/ COUSINS OF A COMPILER



Pre-processor:

Pre-processor produces input to compiler. They may perform the following functions,

1. Macro processing: A preprocessor may allow user to define macros that are shorthand for longer constructs.
2. File inclusion: A preprocessor may include the header file into the program text.

Assembler:

Assembler is a translator which takes the assembly program as an input and generates the machine code as a output. An assembly is a mnemonic version of machine code, in which names are used instead of binary codes for operations.

An assembler is a program that takes basic computer instructions and converts them into a pattern of bits that the computer's processor can use to perform its basic operations. Some people call these instructions assembler language and others use the term assembly language.

For example, **L 8,3000**

a "Load" instruction causes the processor to move a string of bits from a location in the processor's memory to a special holding place called a register. Assuming the processor has at least eight registers, each numbered, the following instruction would move the value (string of bits of a certain length) at memory location 3000 into the holding place called register 8.

Linker:

Linker allows us to make a single program from a several files of relocatable machine code. These file may have been the result of several different compilation, and one or more may be library files of routine provided by a system.

Loader:

The process of loading consists of taking relocatable machine code, altering the relocatable address and placing the altered instructions and data in memory at the proper location.

7.1 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:

A **compiler** is a computer program that translates a program in a source language into an equivalent program in a target language.

A **source program/code** is a program/code written in the source language, which is usually a high-level language.

A **target program/code** is a program/code written in the target language, which often is a machine language or an intermediate code.

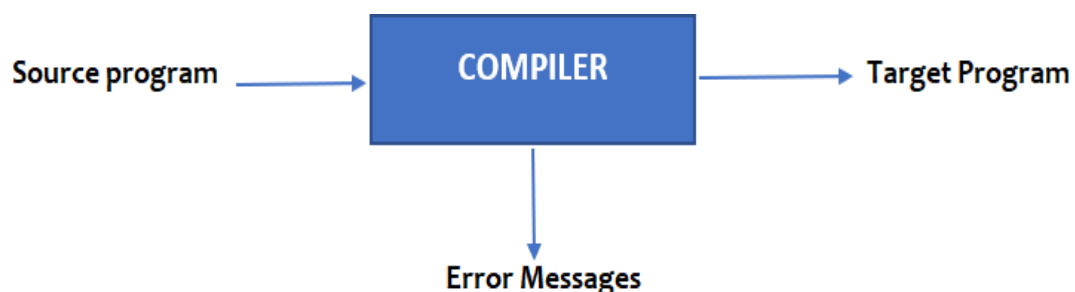


Fig. 1.0. Block diagram of a Compiler

7.2.1 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) used in program instructions.
- iii. Converts a source code written using high level programming language into machine code.
- iv. Generates executable (.exe) object code file.

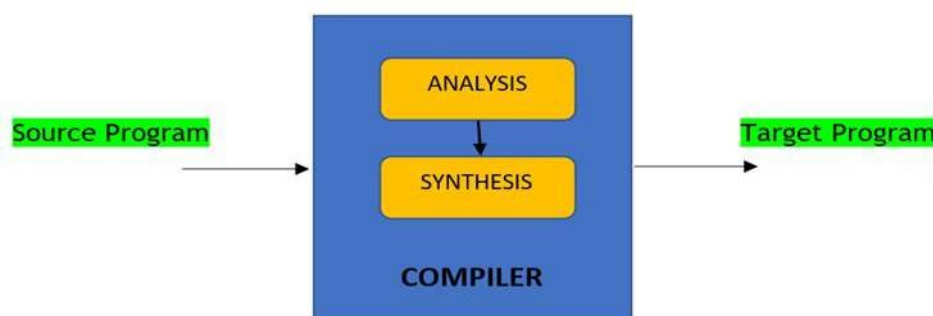
7.3 Differences between Compiler and Interpreter:

COMPILER	INTERPRETER
1. It translates the full source code at a time.	1. It translates one line of code at a time.
2. It translates one line of code at a time.	2. Comparatively slower.
3. It uses more memory to perform.	3. The interpreter uses less memory than the compiler to perform.
4. Error detection is difficult for the compiler.	4. Error detection is easier for the interpreter.
5. It shows error alert after scanning the full program.	5. Whenever it finds any error it stops there.

COMPILER	INTERPRETER
6. Before execution of a program, the compilation is done.	6. Compilation and execution for a program are done simultaneously.
7. It generates intermediate object code.	7. It does not produce any intermediate object code.
8. Example: C Compiler	8. Example: BASIC

7.4 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.

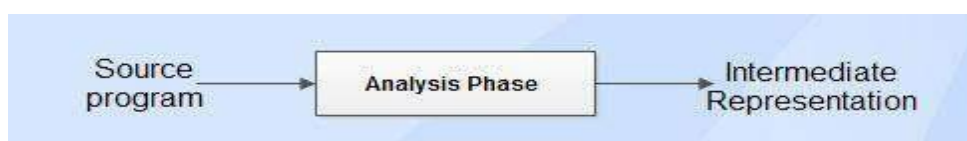


Fig. 2.0. Diagram of Analysis phase

2. Synthesis phase

- Synthesis part takes the intermediate representation as input and transforms it to the target program.
- It is also termed as back end of compiler.

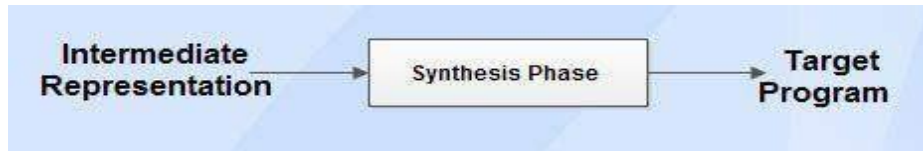


Fig. 3.0. Diagram of Synthesis phase

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

} — Analyzer / Front end

} back end synthesis

All of the aforementioned phases involve the following tasks:

- Symbol table management.
- Error handling.

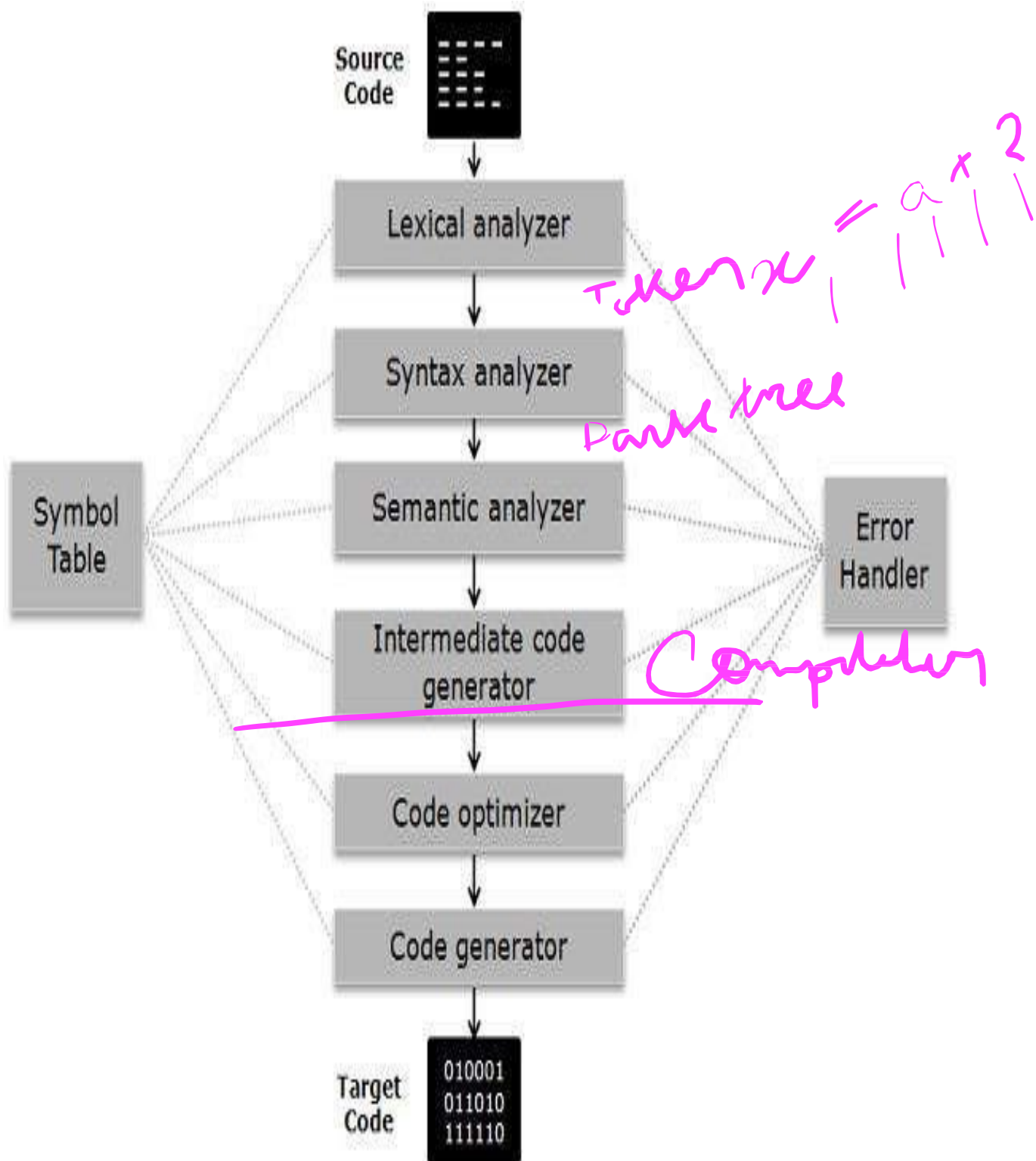


Fig. 4.0. Block diagram of phases of compiler

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 the structure that must be matched by strings.
- Once a token is generated the corresponding entry is made in the symbol table.

Input: stream of characters

Output: 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

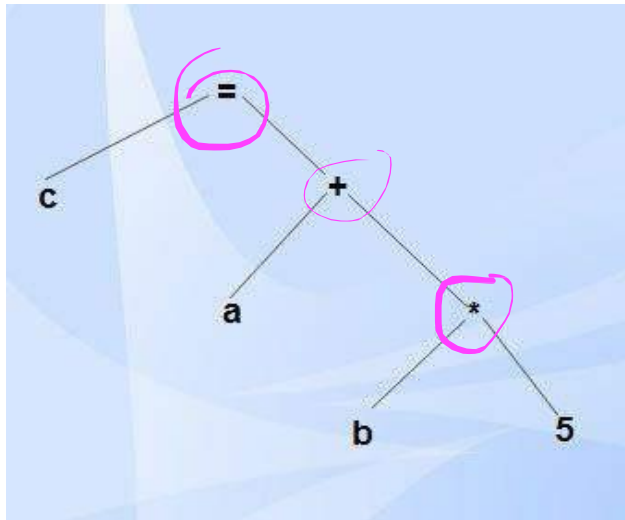


Fig. 5.0. Diagram of Syntax tree

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.

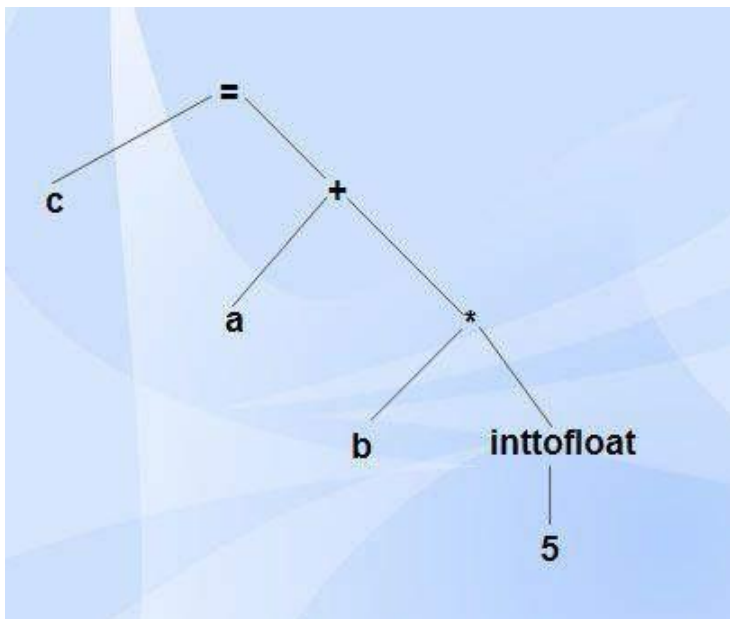


Fig. 6.0. Diagram of Semantic Tree

4. INTERMEDIATE CODE GENERATION

- Intermediate code generation produces intermediate representations for the source program which are of the following forms:
 - o Postfix notation

- o Three address code
- o Syntax tree

$$C = \underline{a + b \times 5}$$

Most commonly used form is the three-address code.

$t_1 = \text{inttofloat}(5)$
 $t_2 = id_3 * t_1$
 $t_3 = id_2 + t_2$
 $id_1 = t_3$

$$\begin{array}{l}
 t_1 = 5 \\
 t_2 = id_3 \times t_1 \quad \text{--- } t_1 = id_3 \times 5 \\
 t_3 = id_2 + t_2 \quad \text{--- } t_2 = id_2 + t_1
 \end{array}$$

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.

$t_1 = id_3 * 5.0$

$id_1 = id_2 + t_1$

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 R_2, id_3
MULF $R_2, \# 5.0$
LDF R_1, id_2
ADDF R_1, R_2
STF id_1, R_1

7(a) SYMBOL TABLE MANAGEMENT

- Symbol table is used to store all the information about identifiers used in the program.
- 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 no meaning
 - (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 somehow deal 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 are handled 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 should be 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.

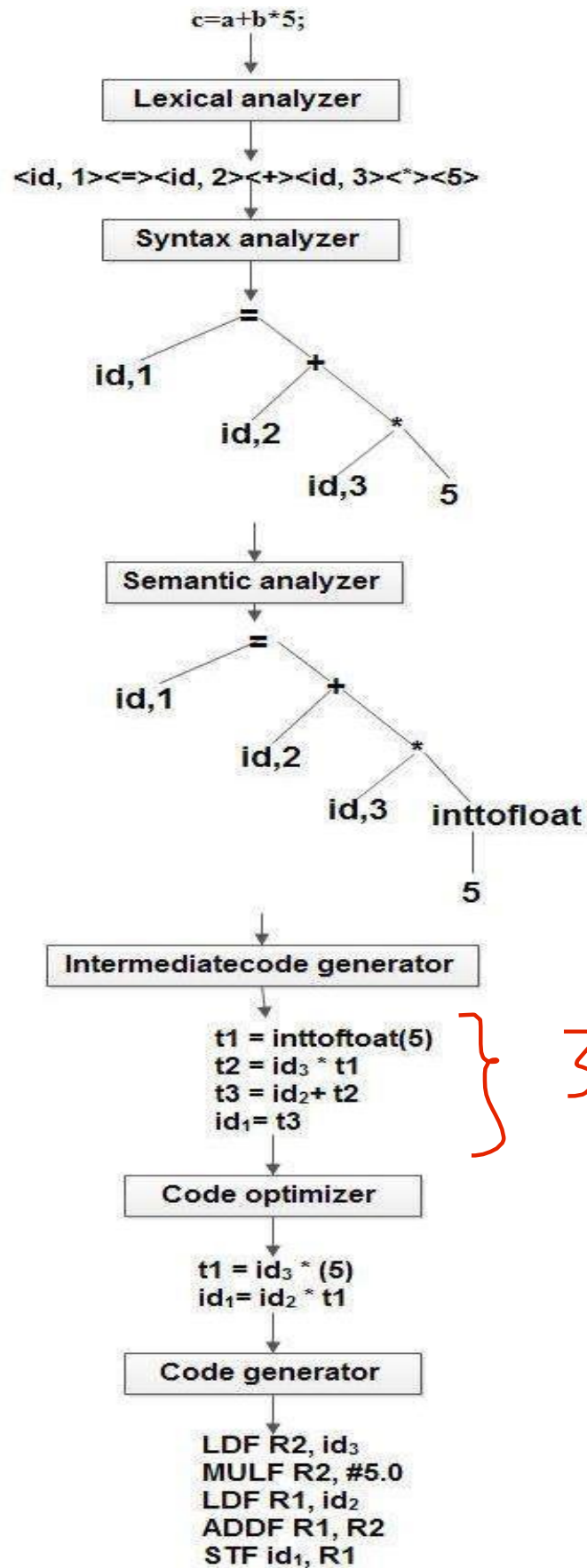
Semantical Errors

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.

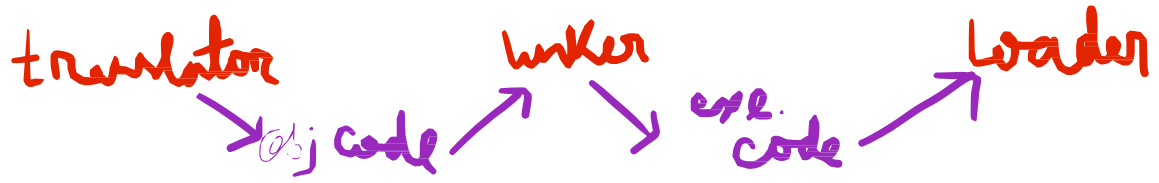
Logical errors

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



} 3 addr code

Fig.7.0 Flowchart of Phases of Compiler



LOADER

A loader is a system software program that performs the loading function. Loading is the process of placing the program into memory for execution. Loader is responsible for initiating the execution of the process.

Types of Loader:

- Compile-Go Loader
- General Loader
- Absolute Loader
- Relocating Loader
- Linking Loader

COMPILE AND GO LOADER

- In compile and go loader is a link editor/program loader in which the assembler itself places the assembled instruction directly into the designated memory locations for execution.
- The instruction are read line by line, its machine code is obtained and it is directly put in the main memory at some known address.
- After completion of assembly process, it assigns the starting address of the program to the location counter.
- The assembler is first executed , when it is finished, causes a branch straight to the first instruction of the program.
- There is no stop between the compilation, link editing, loading, and execution of the program.
- It is also called an **assemble-and-go** or a **load-and-go** system.

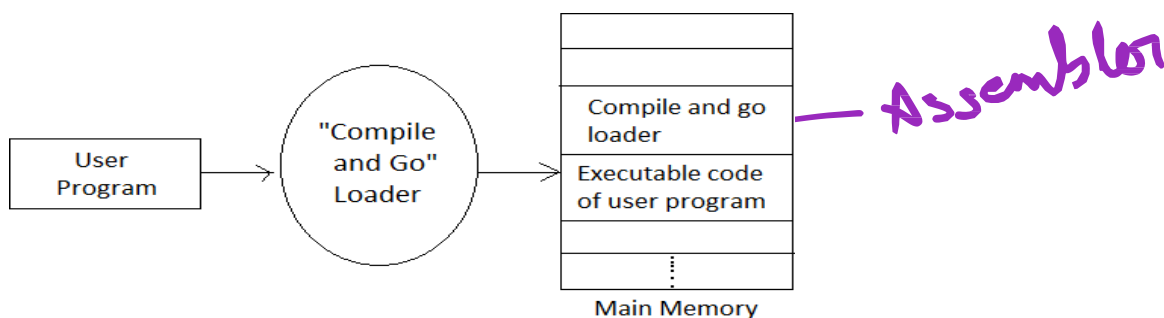


Fig. 8.1. Block diagram of Compile and Go loader

:: :: Advantages :: ::

- They are simple and easier to implement.
- No additional routines are required to load the compiled code into the memory.

:: :: Disadvantages :: ::

- There is wastage in memory space due to the presence of the assembler.
- There is no production of object file, the source code is directly converted to executable form. Hence even though there is no modification in the source program it needs to be assembled and executed each time.

GENERAL LOADER

- In general loader scheme, the source program is converted to object program by some translator (assembler).
- The loader accepts these object modules and puts machine instruction and data in an executable form at their assigned memory.
- The loader occupies some portion of main memory.
- Generally, the size of loader is less than that of assembler.

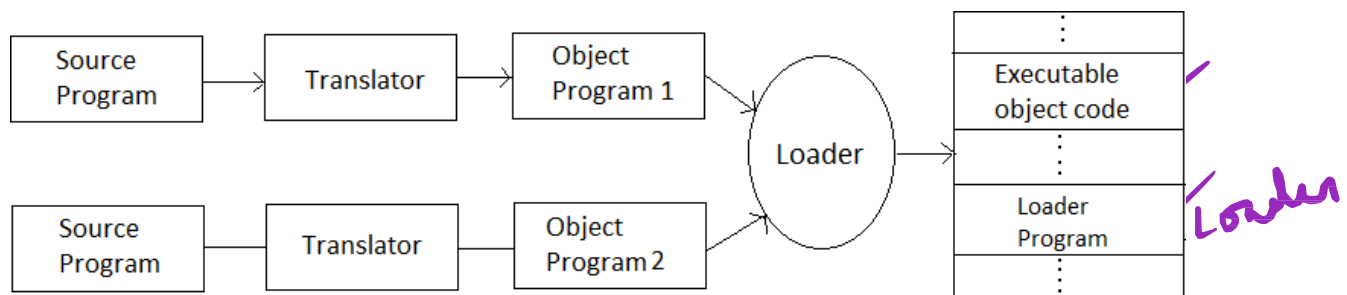


Fig. 8.2. Block diagram of General loader

:: :: Advantages :: ::

- Reassembly of program is not needed
- The translator of the source program produce compatible object program files
- More memory is available to the user.

:: :: Disadvantage :: ::

- The loader cannot handle different object Module Obtained from different kinds of computers.

ABSOLUTE LOADER

- In absolute loader scheme the assembler outputs the machine language translation of the source program in almost the same form as in the “Compile and go”, except that the data is punched on cards. Here it will directly placed in memory.
- The loader in turn simply accepts the machine language text and places it into core at the location prescribed by the assembler.

- The MAIN program is assigned to locations 100-247 and the SQRT subroutine is assigned locations 400-477. If changes were made to MAIN that increasing length to more than 300 bytes.

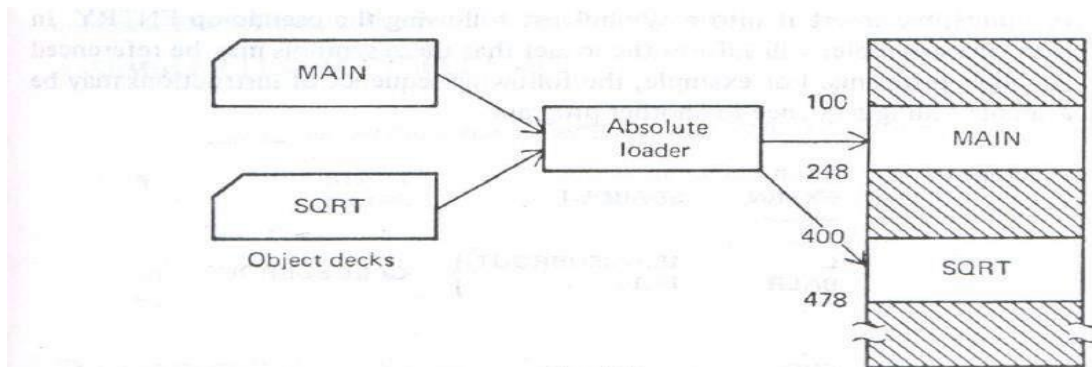


Fig. 8.3. Block diagram of Absolute loader

- The end of MAIN is overlap with the start of SQRT. It would then necessary to assign SQRT to a new location by changing its START.

:: :: Advantages :: ::

- Simple and efficient.

:: :: Disadvantages :: ::

- The programmer must specify to the assembler the address where the program is to be loaded.
- If there are multiple subroutines, the programmer must remember the address of each.

RELOCATING LOADERS

- To avoid possible reassembling of all subroutines when a single subroutine is changed and to perform the tasks of allocation and linking for the programmer the relocating loaders is introduced.
- The execution of the object program is done using any part of the available & sufficient memory.
- The object program is loaded into memory wherever there is room for it.
- The assembler assembles each procedures segment independently and passes to loader the text and information as to relocation and intersegment references.
- The assembler would also provide the loader with additional information, such as the length of the entire program and the length of the transfer vector.

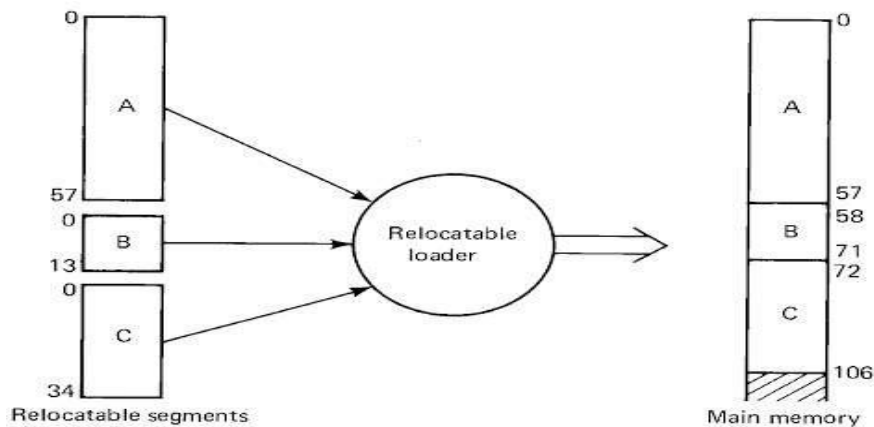


Fig. 8.4. Block diagram of Relocating loader

:: :: Advantages :: ::

- Avoids possible reassembling of all Subroutines when a single Subroutine is changed
- Perform the tasks of allocation and linking for the programmer.

:: :: Disadvantages :: ::

- Difficult to implement
- Algorithm is depending on File structure of the object program.
- Slower than Absolute loader.

LINKING LOADERS

- In this type of loaders first load, links and then Relocates segments.
- Complete freedom in referencing data or instructions contained in other segments
- Perform all linking and relocation at load time.
- A linking loader is known as **general relocatable loader** or **direct-linking loader**.

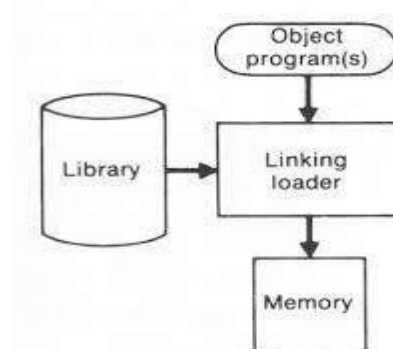


Fig. 8.5.1 Block diagram of Linking loader

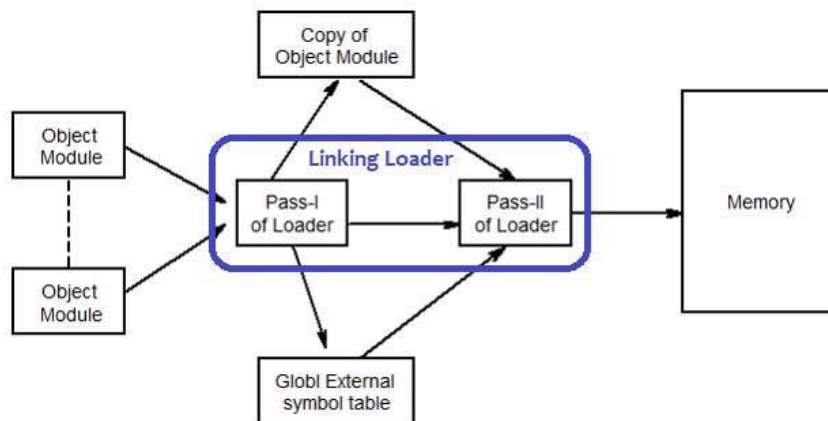


Fig. 8.5.2 Block diagram of Two pass Linking loader

Two pass linking loader

- ▣ **PASS-ONE:** accepts object file and produces Global external symbol table (GEST) and copies of the object programs as O/P.
- ▣ **PASS-TWO:** object programs are loaded to the primary memory after proper translation of address.

One pass linking loader

- ▣ One pass linking loader performs all operations that are specified in a two-pass linking loaders

:: :: Advantages :: ::

- ▣ It allows multiple procedure and data segment.
- ▣ It provides flexible inter-segment referencing and Accessing ability, at same time independent translation of programs.

:: :: Disadvantages :: ::

- ▣ It is necessary for allocation, relocation, linking and loading.
- ▣ All the subroutine are required each time in order to execute a program.
- ▣ It is time consuming.

DIFFERENCE BETWEEN LINKERS AND LOADERS

Linkers	Loaders
1. Combine the target code with the code of other programs and library routines in order to generate a binary program.	1. Load executable code into the memory.
2. Carried out with the help of compiler	2. Part of operating system
3. The execution of linking can be during compilation or during execution of the program.	3. The loads the program into the memory when it is scheduled for execution.

FUNCTIONS OF LOADER: (There are four functions involving in the Absolute loading)

- **Allocation**- It is the space for program is allocated in the main memory, by calculating the size of the program.
- **Linking**- It is in which combines two or more separate object programs and supplies the necessary information.
- **Relocation** - It modifies the object program so that it can be loaded at an address different from the location originally specified.
- **Loading** - It brings the object program into memory for execution.

Note:

A **translator**, in software programming terms, is a generic term that could refer to a compiler, assembler, or interpreter; anything that converts higher level code into another high-level code (e.g., Basic, C++, Fortran, Java) or lower-level (i.e., a language that the processor can understand), such as assembly language or machine code.

Types of translators:

- compiler,
- assembler or
- interpreter