

# **FIRST MODULE**

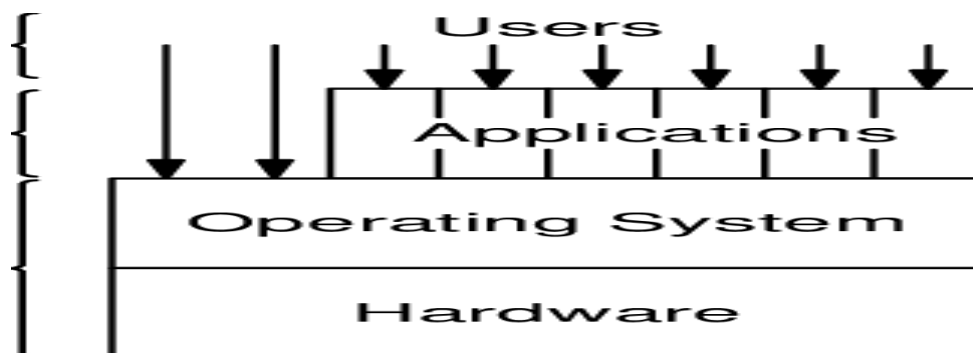
# What is an Operating System?

An **Operating System** can be defined as an **interface between user and hardware**. The purpose of an operating system is to provide an environment in which a user can execute programs in convenient and efficient manner.

Operating system goals:

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

## Computer System Structure



Computer system can be divided into four components:

- Hardware – provides basic computing resources
  - CPU, memory, I/O devices

- Operating system

Controls and coordinates use of hardware among various applications and users

- Application programs – define the ways in which the system resources are used to solve the computing problems of the users

- Word processors, compilers, web browsers, database systems, video games

- Users

- People, machines, other computers

# How OS takes System Control

## →Four Ways in Which an Operating System Controls the Hardware of a Computer

### Drivers

In order for the operating system to talk to your hardware, it needs drivers. Drivers teach the operating system to interact with each bit of hardware. Graphics cards, sound cards, networking cards, USB peripherals, and everything else you connect to your computer relies on drivers. The operating system then uses these drivers to ensure correct operation of each device.

### Memory Management

All computers have random access memory, or RAM. The operating system works with your RAM sticks to control how much memory is used by each application, including the OS itself.

### CPU Control

Your computer's processes are executed by its central processing unit, or CPU. Signals sent to and from the CPU determine what happens, and in what order. The operating system works with the CPU to create a checklist of processes to execute and ensures that each gets done. A CPU can only perform one task at a time; because CPUs are so incredibly fast, it gives the illusion of many simultaneous tasks. The operating system controls the priority of each task and sees them through to completion.

### Fans and Cooling

One of the most important components of your computer is its cooling system. The CPU fan keeps the CPU from overheating in moments of high stress. Overheating can cause permanent damage to a CPU. The operating system communicates with both the CPU and the fan to help ensure the PC stays cool.

## →How does application communicate with operating system?

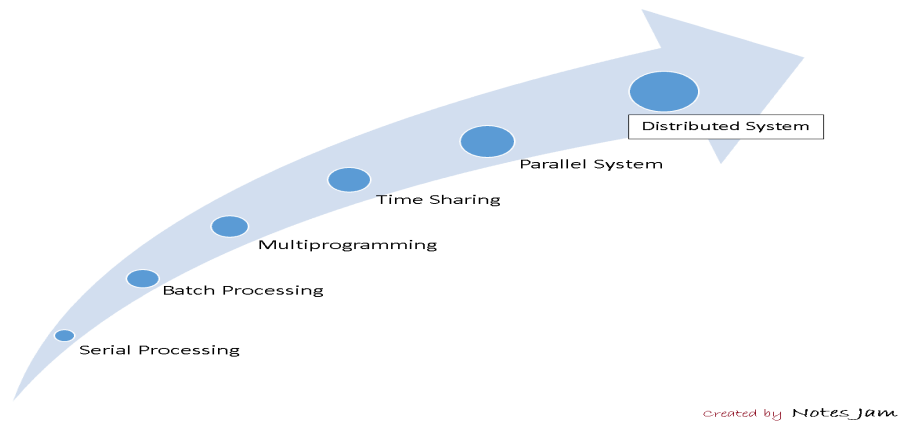
It can only communicate with the operating system through the API or Application Program Interface. The API is a set of common functions that the application calls upon to get the operating system to do what it wants. The operating system never communicates directly with the hardware.

# Functions of Operating System

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.
3. **File management:-** It manages all the file-related activities such as organization, storage, retrieval, naming, sharing, and protection of files.
4. **Device Management:**  
OS performs the task of allocation and de-allocation of the devices. One of the main objects of any OS is to hide the complexity of those hardware devices from the user.
5. **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 be executed.
6. **Security:-** Security module protects the data and information of a computer system against malware threat and unauthorized access.
7. **Command interpretation:** This module is interpreting commands given by the user and acting system resources to process that commands.
8. **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.
9. **Job accounting:** Keeping track of time & resources used by various jobs and users.
10. **Communication management:** Coordination and assignment of compilers, interpreters, and another software resource of the various users of the computer systems.

# Evolution of OS

Here we will discuss six main operating system types evaluated over the past 70 years.



## →Serial Processing

History of the operating system started in 1950. Before 1950, the programmers directly interact with the hardware there was no operating system at that time. If a programmer wishes to execute a program on those days, the following serial steps are necessary.

- Type the program or punched card.
- Convert the punched card to a card reader.
- submit to the computing machine, is there any errors, the error was indicated by the lights.
- The programmer examined the register and main memory to identify the cause of an error
- Take outputs on the printers.
- Then the programmer ready for the next program.

### **Drawback:**

This type of processing is difficult for users, it takes much time and the next program should wait for the completion of the previous one. The programs are submitted to the machine one after one, therefore the method is said to be **serial processing**.

## →Batch Processing

Before 1960, it is difficult to execute a program using a computer because of the computer located in three different rooms, one room for the card reader, one room for executing the program and another room for printing the result.

The user/machine operator runs between three rooms to complete a job. We can solve this problem by using **batch processing**.

In batch processing technique, the same type of jobs batch together and execute at a time. The carrier carries the group of jobs at a time from one room to another.

Therefore, the programmer need not run between these three rooms several times.

## →Multiprogramming

Multiprogramming is a technique to execute the number of programs simultaneously by a single processor. In multiprogramming, a number of processes reside in main memory at a time. The OS(Operating System) picks and begins to execute one of the jobs in main memory.

In the non-multiprogramming system, the CPU can execute only one program at a time, if the running program is waiting for any I/O device, the CPU becomes idle so it will effect on the performance of the CPU.

But in a multiprogramming environment, if any I/O wait happened in a process, then the CPU switches from that job to another job in the job pool. So, the CPU is not idle at any time.

### **Advantages:**

- Can get efficient memory utilization.
- CPU is never idle so the performance of CPU will increase.
- The throughput of CPU may also increase.
- In the non-multiprogramming environment, the user/program has to wait for CPU much time. But waiting time is limited in multiprogramming.

## →Time Sharing System

**Time-sharing** or **multitasking** is a logical extension of multiprogramming. Multiple jobs are executed by the CPU switching between them. The CPU scheduler selects a job from the ready queue and switches the CPU to that job. When the time slot expires, the CPU switches from this job to another.

In this method, the CPU time is shared by different processes. So, it is said to be "**Time-Sharing System**". Generally, time slots are defined by the operating system.

### **Advantages:**

- The main **advantage of the time-sharing system** is efficient CPU utilization. It was developed to provide interactive use of a computer system at a reasonable cost. A time shared OS uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer.
- Another advantage of the time-sharing system over the batch processing system is, the user can interact with the job when it is executing, but it is not possible in batch systems.

## →Parallel System

There is a trend multiprocessor system, such system have more than one processor in close communication, sharing the computer bus, the clock, and sometimes memory and peripheral devices.

These systems are referred to as "Tightly Coupled" system. Then the system is called a **parallel system**. In the parallel system, a number of processors are executing their job in parallel.

### **Advantages:**

- It increases the throughput.
- By increasing the number of processors (CPU), to get more work done in a shorter period of time.

## →Distributed System

In a **distributed operating system**, the processors cannot share a memory or a clock, each processor has its own local memory. The processor communicates with one another through various communication lines, such as high-speed buses. These systems are referred to as "Loosely Coupled" systems.

### **Advantages:**

- If a number of sites connected by high-speed communication lines, it is possible to share the resources from one site to another site, for example, s1 and s2 are two sites. These are connected by some communication lines. The site s1 having a printer, but the site does not have any print. Then the system can be altered without moving from s2 to s1. Therefore, resource sharing is possible in the distributed operating system.
- A big computer that is partitioned into a number of partitions, these sub-partitions are run concurrently in distributed systems.
- If a resource or a system fails in one site due to technical problems, we can use other systems/resources in some other sites. So, the reliability will increase in the distributed system.



# Generations of Operating System

## →The First Generation (1940 to early 1950s)

When the first electronic computer was developed in 1940, it was created without any operating system. In early times, users have full access to the computer machine and write a program for each task in absolute machine language. The programmer can perform and solve only simple mathematical calculations during the computer generation, and this calculation does not require an operating system.

## →The Second Generation (1955 - 1965)

The first operating system (OS) was created in the early 1950s and was known as **GMOS. General Motors** has developed OS for the **IBM** computer. The second-generation operating system was based on a single stream batch processing system because it collects all similar jobs in groups or batches and then submits the jobs to the operating system using a punch card to complete all jobs in a machine. At each completion of jobs (either normally or abnormally), control transfer to the operating system that is cleaned after completing one job and then continues to read and initiates the next job in a punch card. After that, new machines were called mainframes, which were very big and used by professional operators.

## →The Third Generation (1965 - 1980)

During the late 1960s, operating system designers were very capable of developing a new operating system that could simultaneously perform multiple tasks in a single computer program called multiprogramming. The introduction of **multiprogramming** plays a very important role in developing operating systems that allow a CPU to be busy every time by performing different tasks on a computer at the same time. During the third generation, there was a new development of minicomputer's phenomenal growth starting in 1961 .

### →The Fourth Generation (1980 - Present Day)

The fourth generation of operating systems is related to the development of the personal computer. However, the personal computer is very similar to the minicomputers that were developed in the third generation. The cost of a personal computer was very high at that time; there were small fractions of minicomputers costs. A major factor related to creating personal computers was the birth of Microsoft and the Windows operating system. Microsoft created the first **window** operating system in 1975. After introducing the Microsoft Windows OS, Bill Gates and Paul Allen had the vision to take personal computers to the next level. Therefore, they introduced the **MS-DOS** in 1981; however, it was very difficult for the person to understand its cryptic commands. Today, Windows has become the most popular and most commonly used operating system technology. And then, Windows released various operating systems such as Windows 95, Windows 98, Windows XP and the latest operating system, Windows 7. Currently, most Windows users use the Windows 10 operating system. Besides the Windows operating system, Apple is another popular operating system built in the 1980s, and this operating system was developed by Steve Jobs, a co-founder of Apple. They named the operating system Macintosh OS or Mac OS.

# **SECOND MODULE**

# Services of Operating System

- Program execution
- I/O operations
- File system manipulation
- Communication
- Error detection
- Resource allocation
- Protection

## 1) Program Execution

A process includes the complete execution of the written program or code.

There are some of the activities which are performed by the operating system:

- The operating system Loads program into memory
- It also Executes the program
- It Handles the program's execution
- It Provides a mechanism for process synchronization
- It Provides a mechanism for process communication

## 2) I/O Operations

- The communication between the user and devices drivers are managed by the operating system.
- I/O devices are required for any running process. In I/O a file or I/O devices can be involved.
- I/O operations are the read or write operations which are done with the help of input-output devices.
- Operating system gives the access to the I/O devices when it required.

## 3) Communication

There are some major activities that are carried by an operating system with respect to communication.

- Two processes may require data to be transferred between the processes.
- Both the processes can be on one computer or a different computer, but are connected through a computer network.

#### **4) File system manipulation**

There are some major activities which are performed by an operating system with respect to file management.

- The operating system gives an access to the program for performing an operation on the file.
- Programs need to read and write a file.
- The user can create/delete a file by using an interface provided by the operating system.
- The operating system provides an interface to the user creates/ delete directories.
- The backup of the file system can be created by using an interface provided by the operating system.

#### **5) Error handling**

There are some activities that are performed by an operating system with respect to Error handling:

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

#### **6) Protection**

The owners of information stored in a multi-user computer system want to control its use. When several disjoint processes execute concurrently it should not be possible for any process to interfere with another process. Every process in the computer system must be secured and controlled.

#### **7) Resource management**

When there are multiple users or multiple jobs running at the same time resources must be allocated to each of them. There are some major activities that are performed by an operating system:

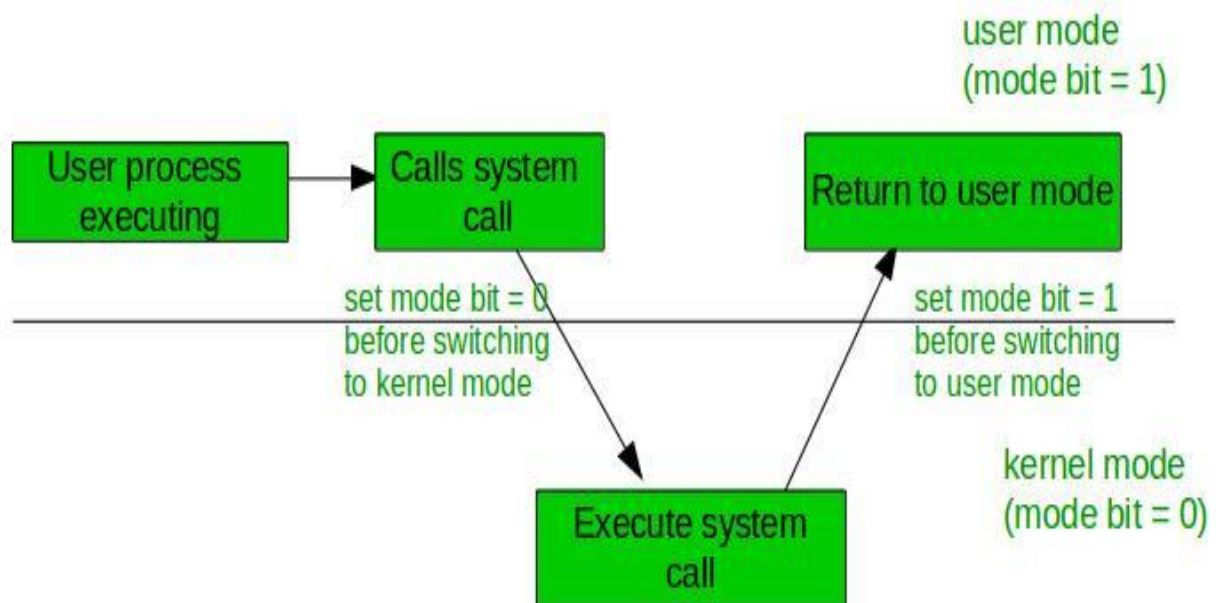
- The OS manages all kinds of resources using schedulers.
- CPU scheduling algorithm is used for better utilization of CPU.

# Dual Mode operations in OS

To ensure the proper execution of the operating system there are two modes of operation:

## →User Mode

When the computer system runs user application like creating a text document or using any application program, then the system is in user mode. When the user application requests for a service from the operating system or an interrupt occurs or system call, then there will be a transition from user to kernel mode to fulfill the requests. To switch from kernel mode to user mode, mode bit should be 1.



## →Kernel Mode

To provide protection to the hardware, we have privileged instructions which execute only in kernel mode. Some of the privileged instructions are:

1. Handling Interrupts
2. To switch from user mode to kernel mode.
3. Input Output management.

# **System calls and types**

The interface between a process and an operating system is provided by system calls. In general, system calls are available as assembly language instructions. System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.

## **Types of System Calls**

There are mainly five types of system calls. These are explained in detail as follows:

- **Process Control**
- **File Management**
- **Device Management**
- **Information Maintenance**
- **Communication**

Here are the types of system calls:

### **Process Control**

These system calls deal with processes such as process creation, process termination etc.

### **File Management**

These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.

### **Device Management**

These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.

### **Information Maintenance**

These system calls handle information and its transfer between the operating system and the user program.

### **Communication**

These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

Types of System Calls	Windows	Linux
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Management	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()

### **wait()**

In some systems, a process may wait for another process to complete its execution. The suspending of the parent process occurs with a wait() system call. When the child process completes execution, the control is returned back to the parent process.

### **exec()**

This system call runs an executable file in the context of an already running process.

### **fork()**

Processes use the fork() system call to create processes that are a copy of themselves.

### **exit()**

The exit() system call is used by a program to terminate its execution.

### **kill()**

The kill() system call is used by the operating system to send a termination signal to a process that urges the process to exit.



# Process

A process is a program in execution. A process is an 'active' entity as opposed to program which is considered to be a 'passive' entity. **Process memory** is divided into four sections for efficient working :

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

## Process Control Block in Operating System

- In an Operating System, we have a number of processes present in it. Each process has some information that is needed by the CPU for the execution of the process. So, we need some kind of data structure to store information about a particular process.
- A Process Control Block or simple PCB is a data structure that is used to store the information of a process that might be needed to manage the scheduling of a particular process.
- So, each process will be given a PCB which is a kind of identification card for a process. All the processes present in the system will have a PCB associated with it and all these PCBs are connected in a Linked List.

# Attributes of a process

The Attributes of the process are used by the Operating System to create the process control block (PCB) for each of them.

## **1. Process ID**

When a process is created, a unique id is assigned to the process which is used for unique identification of the process in the system.

## **2. Program counter**

A program counter stores the address of the last instruction of the process on which the process was suspended. The CPU uses this address when the execution of this process is resumed.

## **3. Process State**

The Process, from its creation to the completion, goes through various states which are new, ready, running and waiting. Ready queue

## **4. Priority**

Every process has its own priority. The process with the highest priority among the processes gets the CPU first. This is also stored on the process control block.

## **5. General Purpose Registers**

Every process has its own set of registers which are used to hold the data which is generated during the execution of the process.

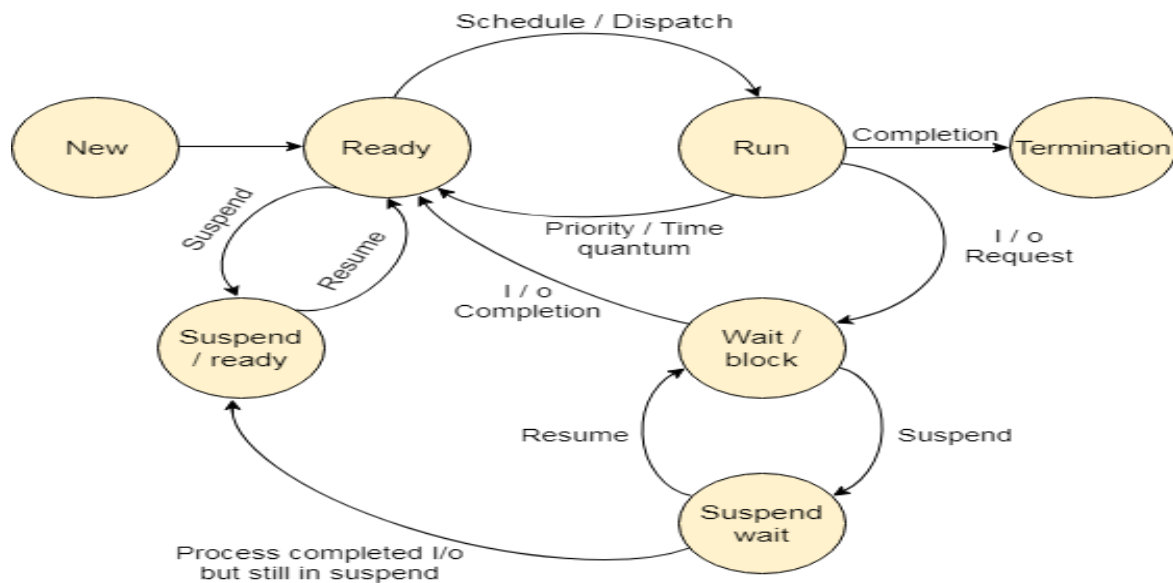
## **6. List of open files**

During the Execution, Every process uses some files which need to be present in the main memory. OS also maintains a list of open files in the PCB.

## **7. List of open devices**

OS also maintain the list of all open devices which are used during the execution of the process.

# Process States



The process, from its creation to completion, passes through various states. The minimum number of states is five.

## → New

A program which is going to be picked up by the OS into the main memory is called a new process.

## → Ready

Whenever a process is created, it directly enters in the ready state, in which, it waits for the CPU to be assigned. The OS picks the new processes from the secondary memory and put all of them in the main memory.

## → Running

One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm. Hence, if we have only one CPU in our system, the number of running processes for a particular time will always be one. If we have  $n$  processors in the system then we can have  $n$  processes running simultaneously.

### **→Block or Wait**

When a process waits for a certain resource to be assigned or for the input from the user then the OS move this process to the block or wait state and assigns the CPU to the other processes.

### **→Completion or termination**

When a process finishes its execution, it comes in the termination state. All the context of the process (Process Control Block) will also be deleted the process will be terminated by the Operating system.

### **→Suspend ready**

A process in the ready state, which is moved to secondary memory from the main memory due to lack of the resources (mainly primary memory) is called in the suspend ready state. If the main memory is full and a higher priority process comes for the execution then the OS have to make the room for the process in the main memory by throwing the lower priority process out into the secondary memory. The suspend ready processes remain in the secondary memory until the main memory gets available.

### **→Suspend wait**

Instead of removing the process from the ready queue, it's better to remove the blocked process which is waiting for some resources in the main memory. Since it is already waiting for some resource to get available hence it is better if it waits in the secondary memory and makes room for the higher priority process. These processes complete their execution once the main memory gets available and their wait is finished.

# Operations on the Process

## →Creation

Once the process is created, it will be ready and come into the ready queue (main memory) and will be ready for the execution.

## →Scheduling

Out of the many processes present in the ready queue, the Operating system chooses one process and start executing it. Selecting the process which is to be executed next, is known as scheduling.

## →Execution

Once the process is scheduled for the execution, the processor starts executing it. Process may come to the blocked or wait state during the execution then in that case the processor starts executing the other processes.

## →Deletion/killing

Once the purpose of the process gets over then the OS will kill the process. The Context of the process (PCB) will be deleted and the process gets terminated by the Operating system.

## System Program:

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.

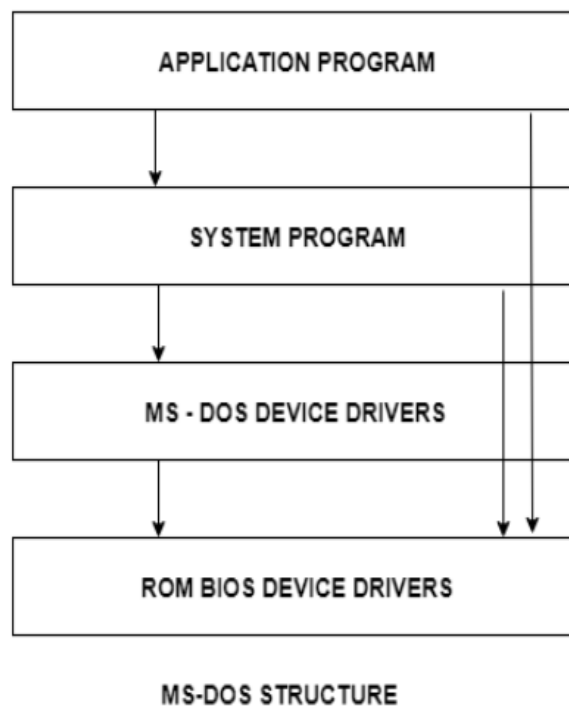
Operating system, interpreter, compiler, editor etc. are all system programs. They provide a built-in environment to the user where several necessary functions like the ones for compiling, editing, interrupt handling, memory management etc. are already defined and hence the user does not need to write any code for such functions.

# OS Structure

Since the operating system is such a complex structure, it should be created with utmost care so it can be used and modified easily.

## 1) Simple structure

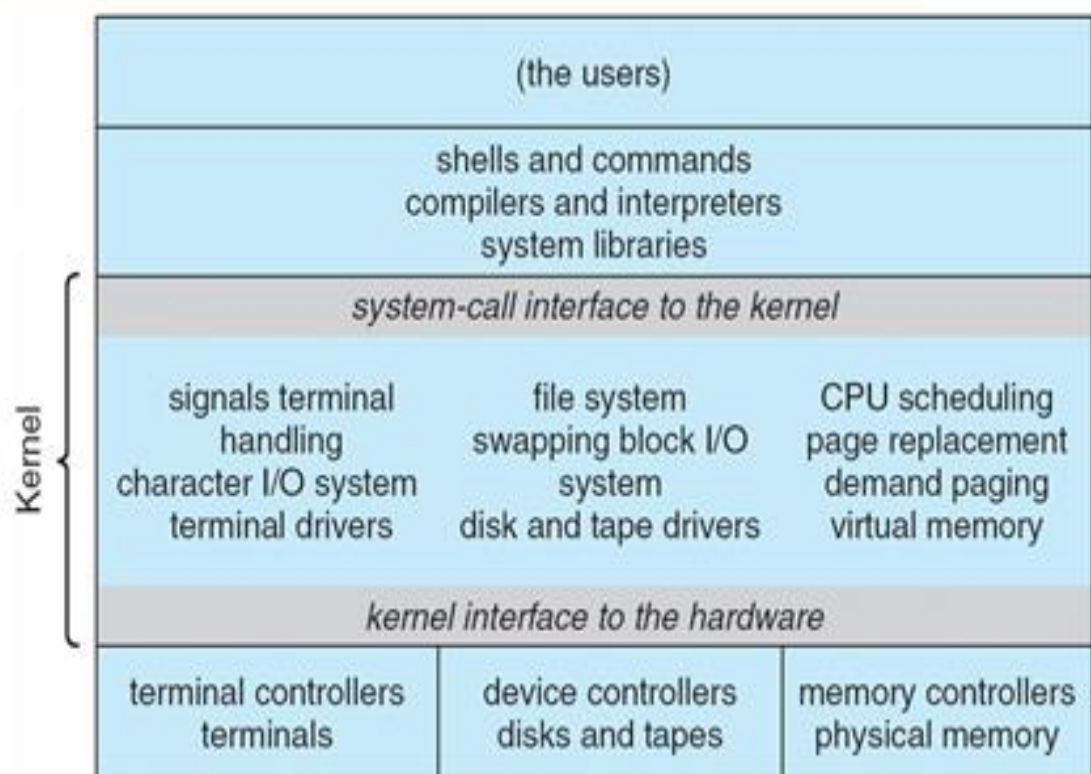
MS-DOS has some simple structure, its interface and levels of functionalities are not well separated. Application programs are able to access I/O devices. If user program fails entire system may crash. It is not well structured, not well defined and not well protected.



## 2) Monolithic structure

The original UNIX OS has limited structure. The Unix OS has two separate programs ie System programs and Kernel. The file system, CPU scheduling, memory management and other operating system functions under one layer. A set of system services that carry out operating system procedures/calls. Here implementation and maintenance is difficult. If we add or modify something the entire kernel has to modify.

### Traditional UNIX System Structure



### **Advantage:**

→ Directly user program can't access hardware. It sends the request to kernel and after execution the response is back to the user. So it is protected.

→ Accessing is faster as kernel contains all the services.

### **Disadvantages:**

→ One service can affect other as no isolation is there.

→ One service is failed entire kernel is crashed.

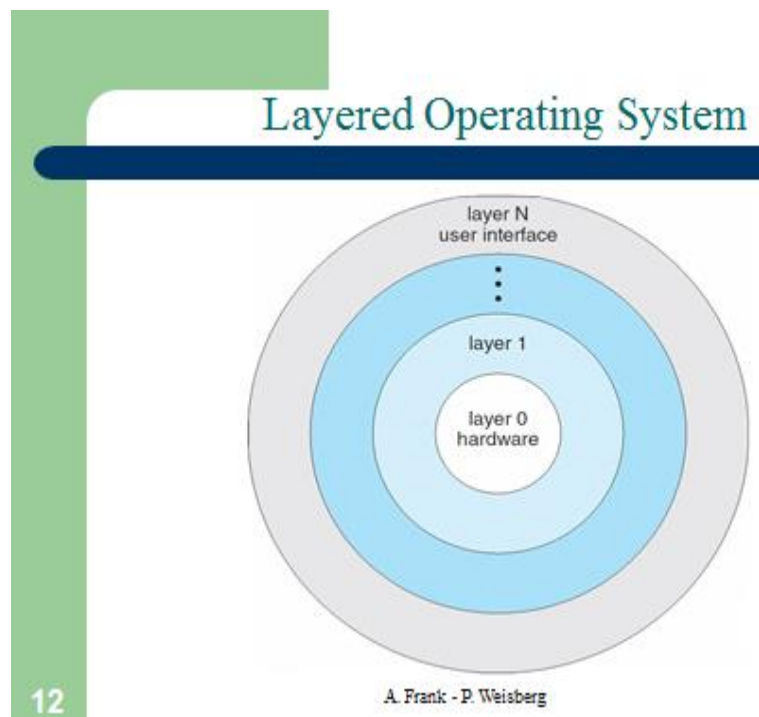
→ Implementation and maintainance is difficult.

→ Size of kernel is large

### **3) Layered structure**

OS divides into multiple layers. Functionalities are separated into multiple layers. Easy to implement and debug. Only we debug that layer when problem is found. When we design the layer the user has to be very particular which layer is placed above which layer. System call is going one by one.

Here hardware is protected.





## **Advantages**

:

There are several advantages to this design :

### **1. Modularity**

:

This design promotes modularity as each layer performs only the tasks it is scheduled to perform.

### **2. Easy debugging :**

As the layers are discrete so it is very easy to debug. Suppose an error occurs in the CPU scheduling layer, so the developer can only search that particular layer to debug, unlike the Monolithic system in which all the services are present together.

### **3. Easy update :**

A modification made in a particular layer will not affect the other layers.

### **4. No direct access to hardware :**

The hardware layer is the innermost layer present in the design. So a user can use the services of hardware but cannot directly modify or access it, unlike the Simple system in which the user had direct access to the hardware.

### **5. Abstraction :**

Every layer is concerned with its own functions. So the functions and implementations of the other layers are abstract to it.

## **Disadvantages :**

Though this system has several advantages over the Monolithic and Simple design, there are also some disadvantages as follows.

### **1. Complex and careful implementation :**

As a layer can access the services of the layers below it, so the arrangement of the layers must be done carefully. For example, the backing storage layer uses the services of the memory management layer. So it must be kept below the memory management layer. Thus with great modularity comes complex implementation.

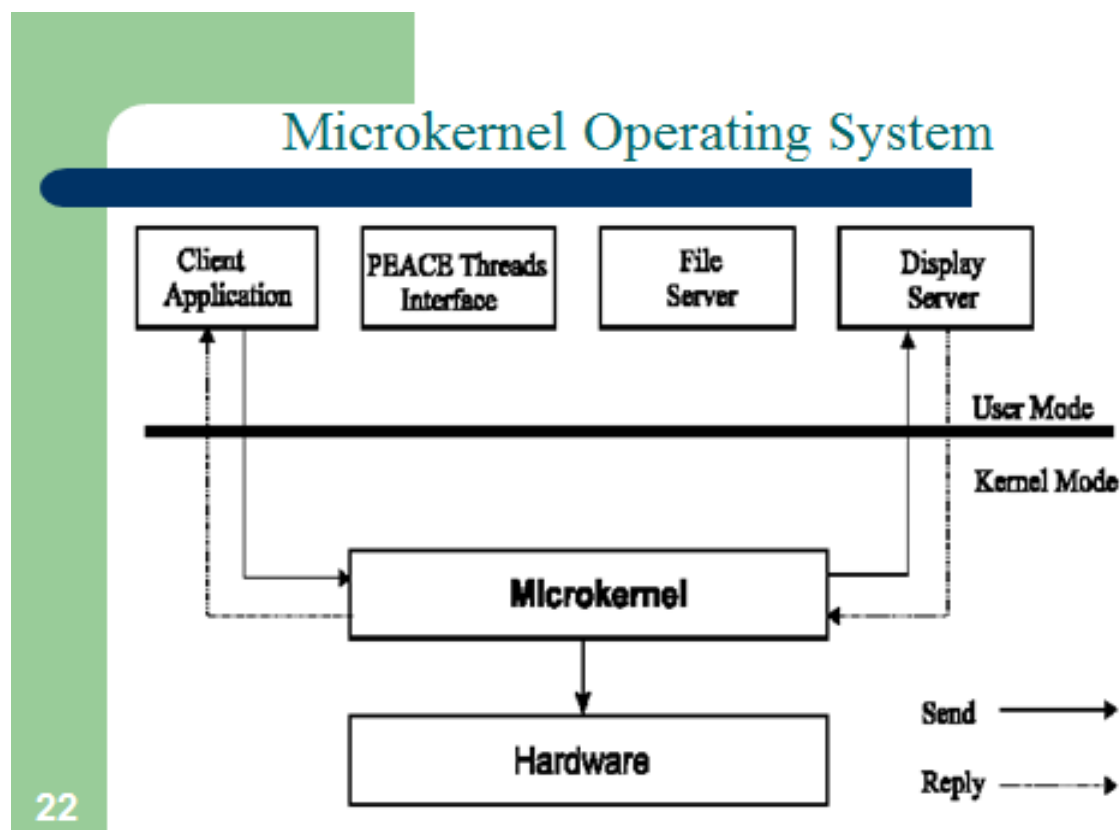
### **2. Slower in execution :**

If a layer wants to interact with another layer, it sends a request that has to travel through all the layers present in between the two interacting layers. Thus it increases response time, unlike the Monolithic system which is faster than this. Thus an increase in the number of layers may lead to a very inefficient design.

#### 4)Micro Kernel structure

In a microkernel, the **user services** and **kernel services** are implemented in different address space. The user services are kept in **user address space**, and kernel services are kept under **kernel address space**, thus also reduces the size of kernel and size of operating system as well.

The communication between client program/application and services running in user address space is established through message passing, reducing the speed of execution microkernel. The Operating System **remains unaffected** as user services and kernel services are isolated so if any user service fails it does not affect kernel service. Thus it adds to one of the advantages in a microkernel. It is easily **extendable** i.e. if any new services are to be added they are added to user address space and hence requires no modification in kernel space. It is also portable, secure and reliable.



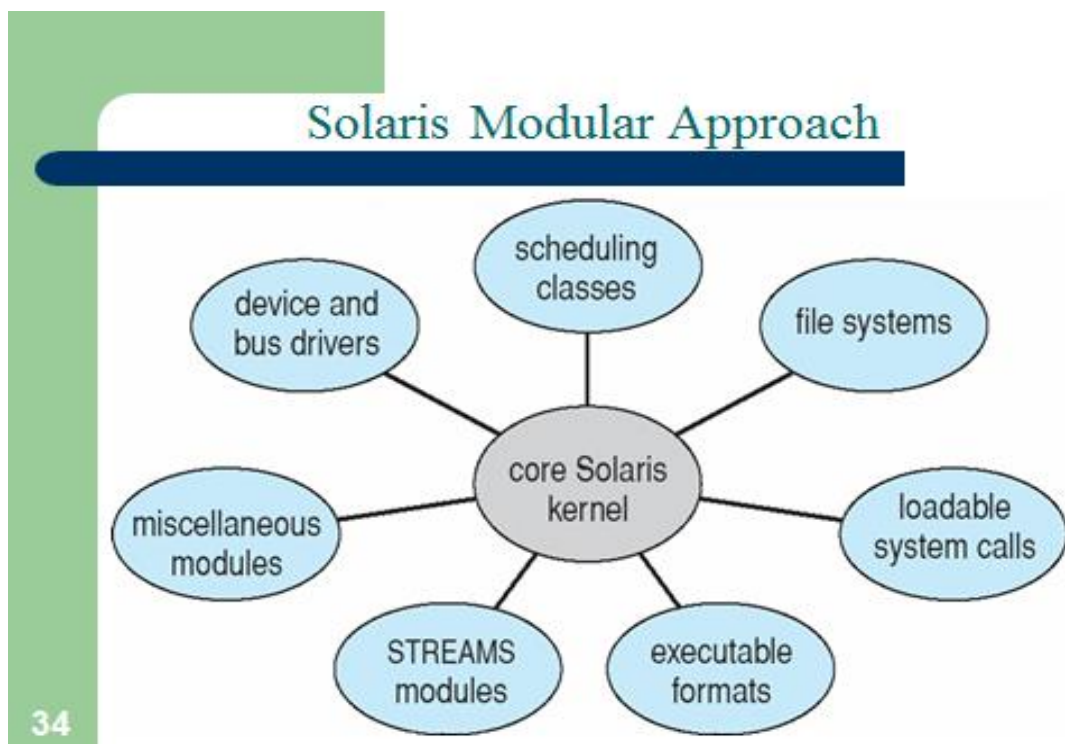
- Providing services in a microkernel system are expensive compared to the normal monolithic system.
- Context switch or a function call needed when the drivers are implemented as procedures or processes, respectively.

## 5) Kernel Modular approach

Most modern operating systems implement kernel modules.

- Uses object oriented approach.
- Each core component is separate
- Each talks to others over known interfaces.
- Each is loadable as needed within the kernel.

Overall ,similar to layers but with more flexibility.

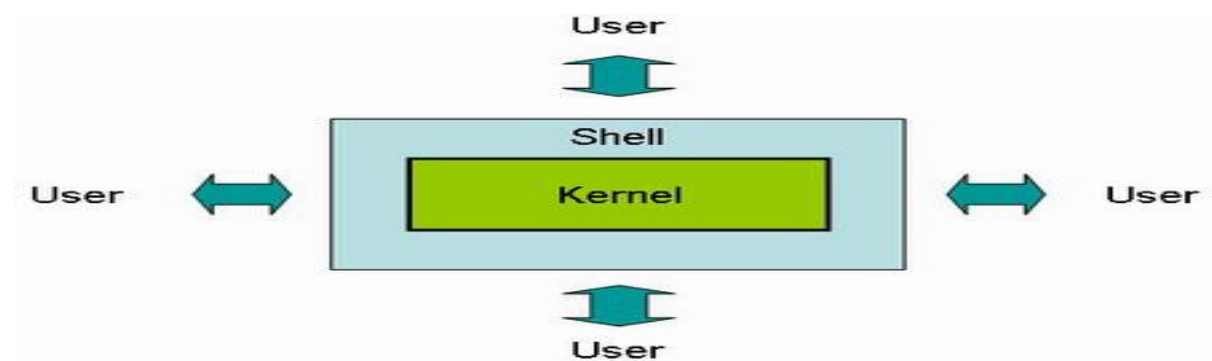


# COMPONENTS OF OPERATING SYSTEM

## Command Interpreter

In order to perform the actions requested by the users, an Operating System must be able to communicate with those users. The portion of an Operating System that handles this communication is often called the shell. Older shells communicate with users through textual messages using a keyboard and monitor screen.

Modern shells perform this task by means of a Graphical User Interface (GUI) in which objects are represented pictorially on the monitor screen as icons. These systems allow the users to issue commands by manipulating these icons with a mouse by pointing and clicking.



*The shell as the interface between the kernel and the user.*

## The Kernel

In contrast to the shell, the internal part of an Operating System is called **the kernel**. The kernel contains those software components that perform the very fundamental functions of a computer:

## File Manager

One of such unit is the file manager, which coordinates the use of the machine's storage facilities. The file manager maintains records of all files stored, which users are allowed to access them and which sections of the storage device are free for new files - these records are kept on the individual storage devices.

File managers generally allow files to be grouped into a bundle called a *folder* (or *directory*), which allows the user to group related files according to their purpose. Furthermore, directories can contain other directories (called sub-directories) so a hierarchical organization of files is possible. A chain of directories is called a directory path.

### **Device Drivers**

Another component consists of a collection of device drivers, which communicate with the peripherals attached to the machine. Each device driver is uniquely designed for its type (such as a printer or video card) and translates the generic requests of the Operating System into the detailed instructions of the device. In this way, the design of the Operating System is independent of the characteristics of the particular devices.

### **Memory Manager**

In multitasking environments, the duties of the memory manager are extensive as the computer is asked to address many needs at the same time. In these cases, many programs and their associated data must reside in main memory concurrently. Thus, the memory manager must assign memory for these needs and ensure that the actions of each program are restricted to their allocated space.

The work of memory manager is complicated further when the total main memory space required exceeds the space actually available in the computer. In this case, the memory manager creates the illusion of additional memory space by swapping programs and data between main memory and disk storage. This large ‘fictional’ memory space created by paging is called virtual memory.

### **Scheduler**

The scheduler determines the order in which activities (called processes) are to be considered for execution in multiprogramming systems, while the dispatcher is in charge of dispatching processes to run on the CPU.

# THIRD MODULE

# Process scheduling in operating system

Process scheduling is a task of operating system to schedules the processes of different states like ready, running, waiting. It is very important in multiprogramming and multitasking operating system, where multiple processes execute simultaneously. At first, the processes that are to be executed are placed in a queue called Job queue. The processes which are already placed in the main memory and are ready for CPU allocation, are placed in a queue called Ready queue. If the process is waiting for input / output, then that process is placed in the queue called Device queue.

**An operating system uses a program scheduler to schedules the processes of computer system. The schedulers are of following types:**

- Long term scheduler
- Mid - term scheduler
- Short term scheduler

## 1) Long Term Scheduler

It selects the process that is to be placed in ready queue. The long term scheduler basically decides the priority in which processes must be placed in main memory.

## 2) Mid – Term Scheduler

It places the blocked and suspended processes in the secondary memory of a computer system. The task of moving from main memory to secondary memory is called **swapping out**. The task of moving back a swapped out process from secondary memory to main memory is known as **swapping in**. The swapping of processes is performed to ensure the best utilization of main memory.

## 3) Short Term Scheduler

It decides the priority in which processes is in the ready queue are allocated the central processing unit (CPU) time for their execution. The short term scheduler is also referred as central processing unit (CPU) scheduler.

## **Scheduling algorithm:**

An operating system uses two types of scheduling processes execution, **pre-emptive** and **non - pre-emptive**.

**1. Pre-emptive process:**In pre-emptive scheduling policy, a low priority process has to be suspending its execution if high priority process is waiting in the same queue for its execution.

**2. Non - pre-emptive process:**In non - pre-emptive scheduling policy, processes are executed in first come first serve basis, which means the next process is executed only when currently running process finishes its execution.

## **CPU Scheduling: Scheduling Criteria**

### **→CPU Utilization**

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time.

### **→Throughput**

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time.

### **→Turnaround Time**

It is the amount of time taken to execute a particular process, i.e. the interval from time of submission of the process to the time of completion.

### **→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.

### **→Load Average**

It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

### **→Response Time**

Amount of time it takes from when a request was submitted until the first response is produced.



# CPU Scheduling

## ->(FCFS Scheduling)

**First come first serve** (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first.

### Advantages of FCFS

- Simple
- Easy
- First come, First serve

### Disadvantages of FCFS

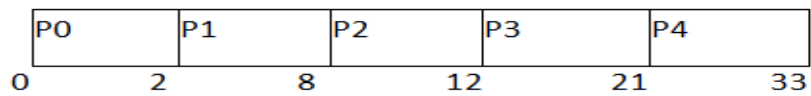
- The scheduling method is non-pre-emptive; the process will run to the completion.
- Due to the non-pre-emptive nature of the algorithm, the problem of starvation may occur.
- Although it is easy to implement, but it is poor in performance since the average waiting time is higher as compare to other scheduling algorithms.

## Example

There are 5 processes with process ID **P0, P1, P2, P3 and P4**. The processes and their respective Arrival and Burst time are given in the following table. Calculate Completion time, Turnaround time, waiting time, average waiting time.

<b>Turn Around Time = Completion Time - Arrival Time</b>
<b>Waiting Time = Turnaround time - Burst Time</b>

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
0	0	2	2	2	0
1	1	6	8	7	1
2	2	4	12	10	6
3	3	9	21	18	9
4	4	12	33	29	17



**(Gantt chart)**

**Avg Waiting Time=(Total Waiting time)/(No of processes)**

**=33/5=6.6**

## → Shortest job first (SJF)

In this scheduling algorithm the process which requires shortest CPU time to execute is processed first.

### Advantages of SJF

- Maximum throughput
- Minimum average waiting and turnaround time

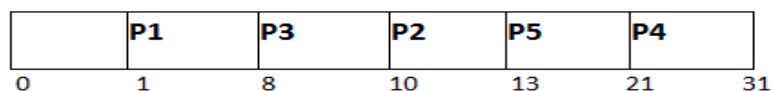
### Disadvantages of SJF

- May suffer with the problem of starvation
- It is not implementable because the exact Burst time for a process can't be known in advance.

## Example

In the following example, there are five jobs named as P1, P2, P3, P4 and P5. Their arrival time and burst time are given in the table below. Calculate Completion time, Turnaround time, waiting time, average waiting time.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	1	7	8	7	0
2	3	3	13	10	7
3	6	2	10	4	2
4	7	10	31	24	14
5	9	8	21	12	4



$$\text{Avg Waiting Time} = 27/5 = 5.4$$

## ➔ Highest Response Ratio Next (HRRN) Scheduling

Highest Response Ratio Next (HRNN) is one of the most optimal scheduling algorithms. This is a non-pre-emptive algorithm in which, the scheduling is done on the basis of an extra parameter called Response Ratio. A Response Ratio is calculated for each of the available jobs and the Job with the highest response ratio is given priority over the others.

**Response Ratio =  $(W+S)/S$  Where,**  
**W → Waiting Time**

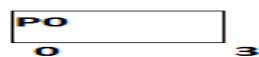
**S → Service Time or Burst Time**

### HRNN Example

In the following example, there are 5 processes given. Their arrival time and Burst Time are given in the table. Calculate Completion time, Turnaround time, waiting time, average waiting time.

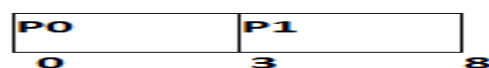
Process ID	Arrival Time	Burst Time
0	0	3
1	2	5
2	4	4
3	6	1
4	8	2

At time 0, The Process P0 arrives with the CPU burst time of 3 units. Since it is the only process arrived till now hence this will get scheduled immediately.



P0 is executed for 3 units, meanwhile, only one process P1 arrives at time 3.

This will get scheduled immediately since the OS doesn't have a choice.



P1 is executed for 5 units. Meanwhile, all the processes get available. We have to calculate the Response Ratio for all the remaining jobs.

$$RR (P2) = ((8-4) +4)/4 = 2$$

$$RR (P3) = ((8-6)+1)/1 = 3$$

$$RR (P4) = ((8-8)+2)/2 = 1$$

Since, the Response ratio of P3 is higher hence P3 will be scheduled first.

P0	P1	P3
0	3	8
		9

P3 is scheduled for 1 unit. The next available processes are P2 and P4. Let's calculate their Response ratio.

$$RR (P2) = ((9-4)+4)/4 = 2.25$$

$$RR (P4) = ((9-8)+2)/2 = 1.5$$

The response ratio of P2 is higher hence P2 will be scheduled.

P0	P1	P3	P2
0	3	8	9
			13

Now, the only available process is P4 with the burst time of 2 units, since there is no other process available hence this will be scheduled.

P0	P1	P3	P2	P4
0	3	8	9	13
				15

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
0	0	3	3	3	0
1	2	5	8	6	1
2	4	4	13	9	5
3	6	1	9	3	2
4	8	2	15	7	5

$$\text{Average Waiting Time} = (0+1+5+2+5)/5 = 13/5 = 2.6$$

## → Priority Scheduling

In Priority scheduling, there is a priority number assigned to each process. In some systems, the lower the number, the higher the priority. While, in the others, the higher the number, the higher will be the priority. The Process with the higher priority among the available processes is given the CPU. There are two types of priority scheduling algorithm exists. One is **Pre-emptive** priority scheduling while the other is **Non Pre-emptive** Priority scheduling.

### 1. Non Pre-emptive Priority Scheduling

In the Non Pre-emptive Priority scheduling, the Processes are scheduled according to the priority number assigned to them. Once the process gets scheduled, it will run till the completion.

#### Example

In the Example, there are 7 processes P1, P2, P3, P4, P5, P6 and P7. Their priorities, Arrival Time and burst time are given in the table. Calculate Completion time, Turnaround time, waiting time, average waiting time.

Condition: Lower number is the higher priority

Process ID	Priority	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	2	0	3	3	3	0
2	6	1	5	18	17	12
3	3	2	4	7	5	1
4	5	3	2	13	10	8
5	7	4	9	27	23	14
6	4	5	4	11	6	2
7	10	6	10	37	31	21

P1	P3	P6	P4	P2	P5	P7	
0	3	7	11	13	18	27	37

**Ganttchart**

**Avg Waiting Time =  $(0+11+2+7+12+2+18)/7 = 52/7$  units**

## 2. Pre-emptive Priority Scheduling

In Pre-emptive Priority Scheduling, at the time of arrival of a process in the ready queue, its Priority is compared with the priority of the other processes present in the ready queue as well as with the one which is being executed by the CPU at that point of time. The One with the highest priority among all the available processes will be given the CPU next.

### Example:

There are 7 processes P1, P2, P3, P4, P5, P6 and P7 given. Their respective priorities, Arrival Times and Burst times are given in the table below. Calculate Completion time, Turnaround time, waiting time, average waiting time.

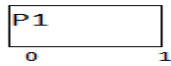
**LOWER NUMBER IS THE HIGHEST PRIORITY**

Process ID	Priority	Arrival Time	Burst Time
1	2	0	1
2	6	1	7
3	3	2	3
4	5	3	6
5	4	4	5
6	10	5	15
7	9	6	8

P1	P2	P3	P5	
0	1	2	5	10

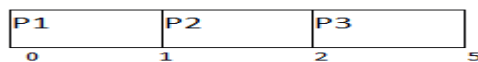
## GANTT chart Preparation

→ At time 0, P1 arrives with the burst time of 1 units and priority 2. Since no other process is available hence this will be scheduled till next job arrives or its completion (whichever is lesser).



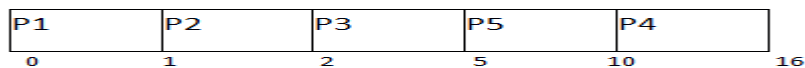
→ At time 1, P2 arrives. P1 has completed its execution and no other process is available at this time hence the Operating system has to schedule it regardless of the priority assigned to it.

→ The Next process P3 arrives at time unit 2, the priority of P3 is higher to P2. Hence the execution of P2 will be stopped and P3 will be scheduled on the CPU.



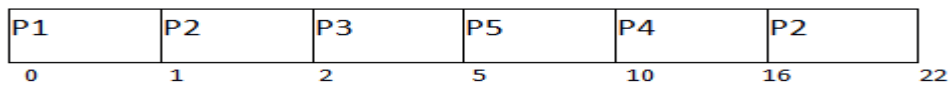
→ During the execution of P3, three more processes P4, P5 and P6 becomes available. Since, all these three have the priority lower to the process in execution so P3 will complete its execution and then P5 will be scheduled with the priority highest among the available processes.

→ Meanwhile the execution of P5, all the processes got available in the ready queue .The OS just took the process with the highest priority and execute that process till completion. In this case, P4 will be scheduled and will be executed till the completion.

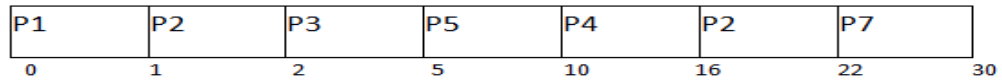


→ Since P4 is completed, the other process with the highest priority available in the ready queue is P2. Hence P2 will be scheduled next.

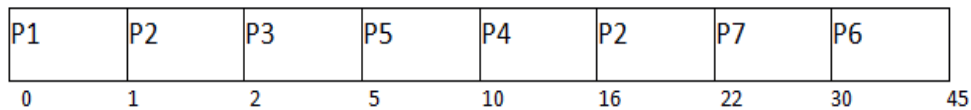




→P2 is given the CPU till the completion. Since its remaining burst time is 6 units hence P7 will be scheduled after this.



→The only remaining process is P6 with the least priority, the Operating System has no choice unless of executing it. This will be executed at the last.



**Turnaround Time = Completion Time - Arrival Time**

**Waiting Time = Turn Around Time - Burst Time**

Process ID	Priority	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	2	0	1	1	1	0
2	6	1	7	22	21	14
3	3	2	3	5	3	0
4	5	3	6	16	13	7
5	4	4	5	10	6	1
6	10	5	15	45	40	25
7	9	6	8	30	24	16

**Avg Waiting Time =  $(0+14+0+7+1+25+16)/7 = 63/7 = 9$  units**

**→Round Robin Scheduling Algorithm**

This is the **pre-emptive version** of first come first serve scheduling. A certain time slice is defined in the system which is called time **quantum**. Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will **terminate** else the process will go back to the **ready queue** and waits for the next turn to complete the execution.

### Advantages

1. It can be actually implementable in the system because it is not depending on the burst time.
2. It doesn't suffer from the problem of starvation or convoy effect.
3. All the jobs get a fair allocation of CPU.

### Disadvantages

1. The higher the time quantum, the higher the response time in the system.
2. The lower the time quantum, the higher the context switching overhead in the system.
3. Deciding a perfect time quantum is really a very difficult task in the system.

### Example of Round Robin Scheduling

In this example, we will take six processes P1, P2, P3, P4, P5 and P6 whose arrival and burst time are given in the table. The time quantum is 4 units. Calculate Completion time, Turnaround time, waiting time, average waiting time.

Process	Arrival Time	Burst time	Completion time	Turn around time	Waiting time
P1	0	5	17	17	12
P2	1	6	23	22	16
P3	2	3	11	9	6
P4	3	1	12	9	8
P5	4	5	24	20	15
P6	6	4	21	15	11

## GANTT chart

The P1 will be executed for 4 units first.

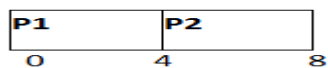


## Ready Queue

P2	P3	P4	P5	P1
6	3	1	5	1(5-4)

## GANTT chart

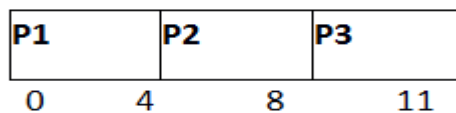
After P1, P2 will be executed for 4 units of time which is shown in the Gantt chart.



## Ready Queue

P3	P4	P5	P1	P6	P2
3	1	5	1	4	2

## GANTT chart

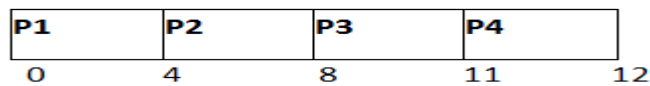


## Ready Queue

P4	P5	P1	P6	P2
1	5	1	4	2

## GANTT chart

After, P1, P2 and P3, P4 will get executed. Its burst time is only 1 unit which is lesser than the time quantum hence it will be completed.



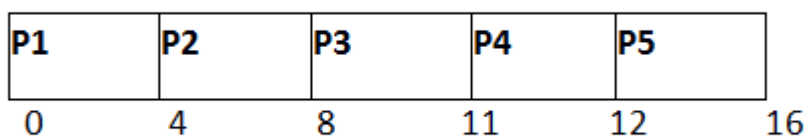
## Ready Queue

The next process in the ready queue is P5 with 5 units of burst time. Since P4 is completed hence it will not be added back to the queue.

P3	P4	P5	P1	P6	P2
3	1	5	1	4	2

## GANTT chart

P5 will be executed for the whole time slice because it requires 5 units of burst time which is higher than the time slice.



## Ready Queue

P5 has not been completed yet; it will be added back to the queue with the remaining burst time of 1 unit.

P1	P6	P2	P5
----	----	----	----

1	4	2	1(5-4)
---	---	---	--------

### GANTT Chart

The process P1 will be given the next turn to complete its execution. Since it only requires 1 unit of burst time hence it will be completed.

<b>P1</b>	<b>P2</b>	<b>P3</b>	<b>P4</b>	<b>P5</b>	<b>P1</b>	
0	4	8	11	12	16	17

### Ready Queue

P1 is completed and will not be added back to the ready queue. The next process P6 requires only 4 units of burst time and it will be executed next.

P6	P2	P5
4	2	1

### GANTT chart

P6 will be executed for 4 units of time till completion.

P1	P2	P3	P4	P5	P1	P6	
0	4	8	11	12	16	17	21

### Ready Queue

Since P6 is completed, hence it will not be added again to the queue. There are only two processes present in the ready queue. The Next process P2 requires only 2 units of time.

P2	P5
2	1

### GANTT Chart

P2 will get executed again, since it only requires only 2 units of time hence this will be completed.

<b>P1</b>	<b>P2</b>	<b>P3</b>	<b>P4</b>	<b>P5</b>	<b>P1</b>	<b>P6</b>	<b>P2</b>	
0	4	8	11	12	16	17	21	23

### Ready Queue

Now, the only available process in the queue is P5 which requires 1 unit of burst time. Since the time slice is of 4 units hence it will be completed in the next burst.

P5
1

### GANTT chart

P5 will get executed till completion.

<b>P1</b>	<b>P2</b>	<b>P3</b>	<b>P4</b>	<b>P5</b>	<b>P1</b>	<b>P6</b>	<b>P2</b>	<b>P5</b>	
0	4	8	11	12	16	17	21	23	24

### RESPONSE TIME

**P1:0   P2:4   P3:8   P4:11   P5:12   P6:17**

### **GANTTCHART**

P1	P2	P3	P4	P5	P1	P6	P2	P5	
0	4	8	11	12	16	17	21	23	24

**Average waiting time=  $(12+16+6+8+15+11)/6 = 76/6 = 12.66$  units**

## →Shortest Remaining Time First (SRTF) scheduling

This scheduling Algorithm is the pre-emptive version of the SJF scheduling algorithm. In this, the process which is left with the least processing time is executed first. Once all the processes are available in the **ready queue**, No pre-emption will be done and the algorithm will work as **SJF scheduling**. The context of the process is saved in the **Process Control Block** when the process is removed from the execution and the next process is scheduled. This PCB is accessed on the **next execution** of this process.

### Example

In this Example, there are five jobs P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below in the table. Calculate Completion time, Turnaround time, waiting time, average waiting time.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	0	8	20	20	12
2	1	4	10	9	5
3	2	2	4	2	0
4	3	1	5	2	1
5	4	3	13	9	6
6	5	2	7	2	0

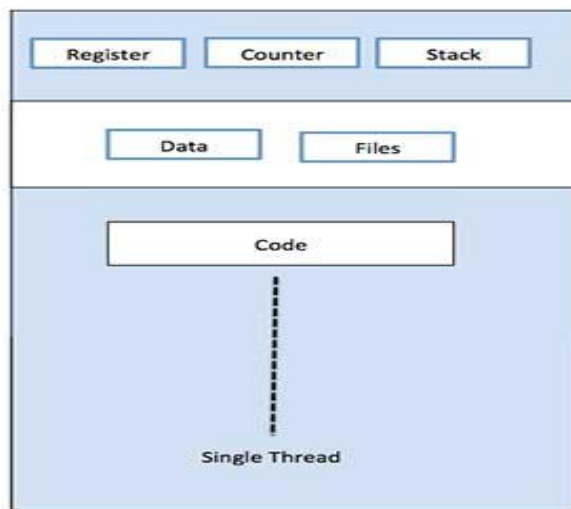
P1	P2	P3	P3	P4	P6	P2	P5	P1	
0	1	2	3	4	5	7	10	13	20

$$\text{Avg Waiting Time} = 24/6$$

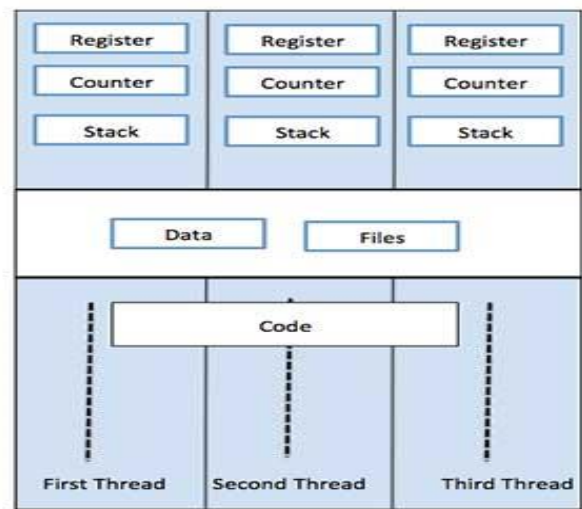


# Multithreading

Thread is an execution unit that is part of a process. A process can have multiple threads, all executing at the same time. It is a unit of execution in concurrent programming. A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history. Threads provide a way to improve application performance through parallelism.



Single Process P with single thread



Single Process P with three threads

## Advantages of Thread

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

# Difference between Process and Thread

Process	Thread
Process is heavy weight.	Thread is light weight.
Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
Context switching time is more.	Context switching time is less.
In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.

## Types of Thread

Threads are implemented in following two ways –

- **User Level Threads** – User managed threads.
- **Kernel Level Threads** – Operating System managed threads.

### →User Level Threads

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

### Advantages

- Thread switching does not require Kernel mode .so context switching time is less.
- User level thread can run on any operating system.
- User level threads are fast to create and manage.

## **Disadvantages**

- If one user level thread is block entire process will be blocked.

### **→Kernel Level Threads**

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

## **Advantages**

- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

## **Disadvantages**

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

## **Multithreading Models**

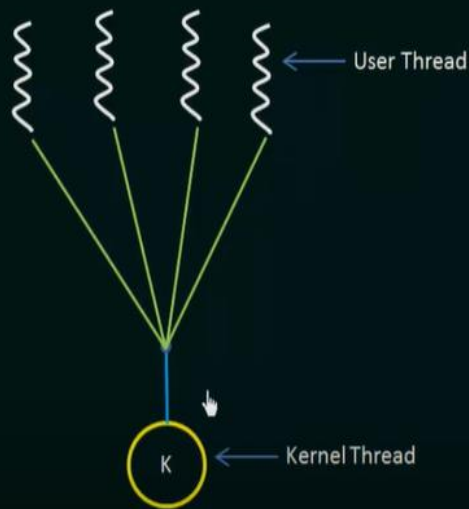
To function a system there must exist a relationship between kernel thread and user thread.

There are three types of models.

- One-to-One model
- Many-to-one model
- Many-to-many model

### **→Many to One Model**

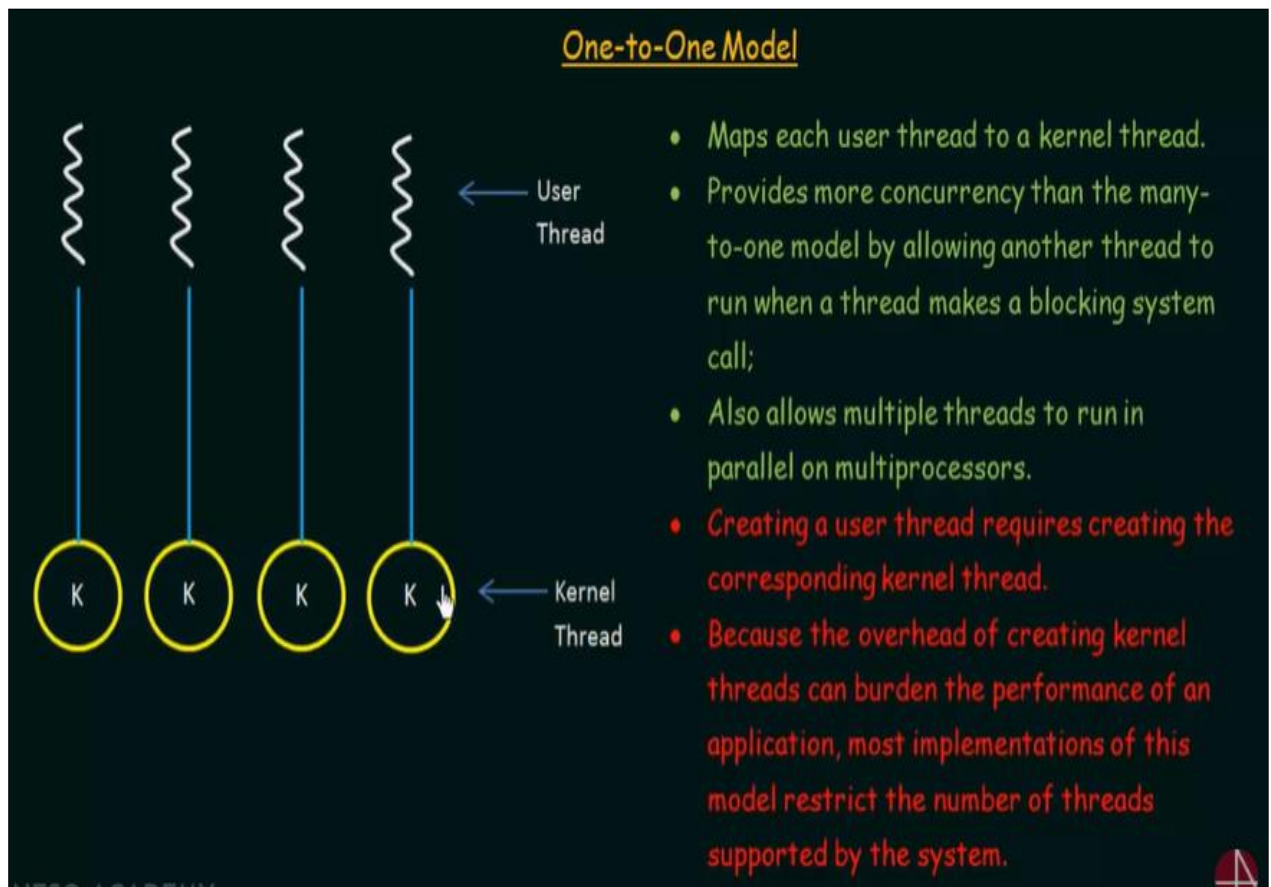
### Many-to-One Model



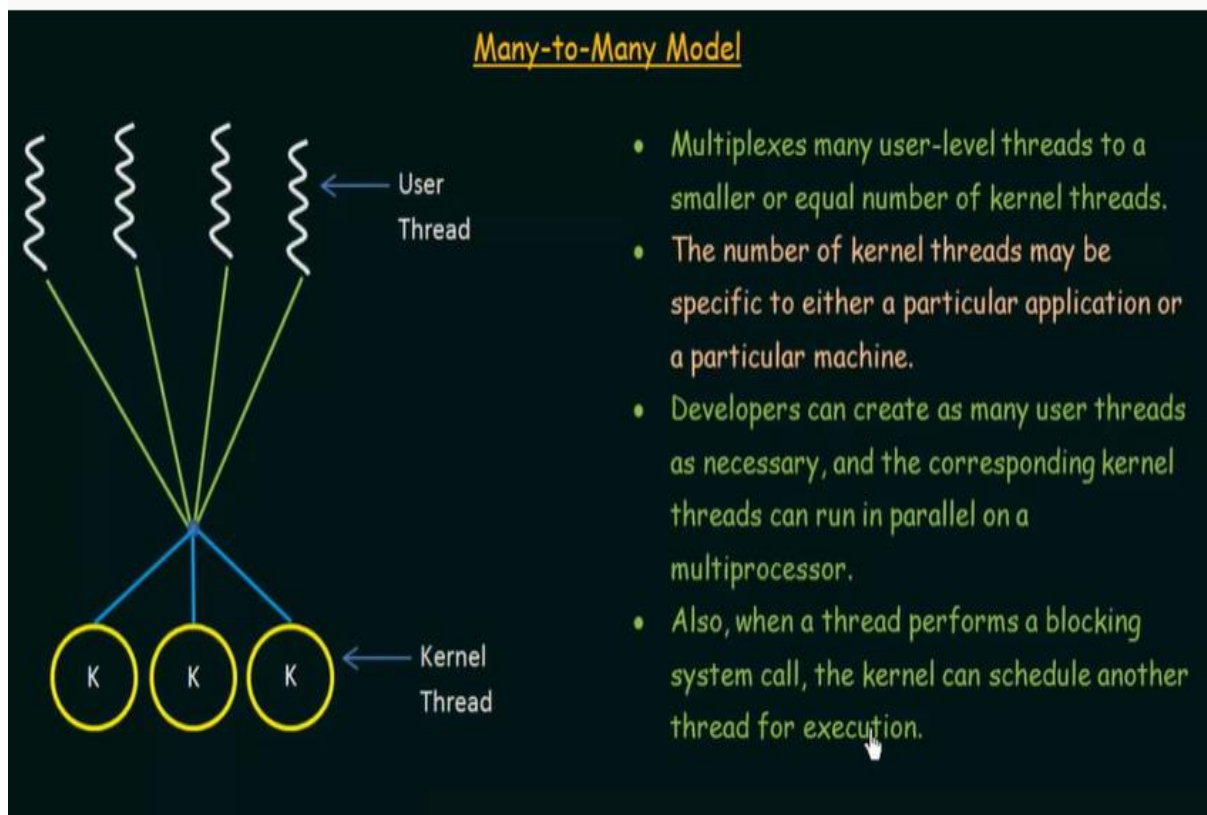
- Maps many user-level threads to one kernel thread.
- Thread management is done by the thread library in user space, so it is efficient.
- The entire process will block if a thread makes a blocking system call.
- Because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.



## → One to One Model



## → Many to Many Model



# Process Creation and Deletions in OS

There are two basic operations that can be performed on a process: Creation and Deletion. They are explained as following below.

## 1 . Process creation:

- (i). When a new process is created, operating system assigns a unique Process Identifier (PID) to it and inserts a new entry in primary process table.
- (ii). Then the required memory space for all the elements of process such as program, data and stack is allocated including space for its Process Control Block (PCB).
- (iii). Next, the various values in PCB are initialized such as,
  - 1. Process identification part is filled with PID assigned to it in step (1) and also its parent's PID.
  - 2. The processor register values are mostly filled with zeroes, except for the stack pointer and program counter. Stack pointer is filled with the address of stack allocated to it in step (ii) and program counter is filled with the address of its program entry point.
  - 3. The process state information would be set to 'New'.
  - 4. Priority would be lowest by default, but user can specify any priority during creation.

In the beginning, process is not allocated to any I/O devices or files. The user has to request them or if this is a child process it may inherit some resources from its parent.

(vi). Then the operating system will link this process to scheduling queue and the process state would be changed from 'New' to 'Ready'. Now process is competing for the CPU.

(v). Additionally, operating system will create some other data structures such as log files or accounting files to keep track of processes activity.

## **2 . Process Deletion:**

Processes are terminated by themselves when they finish<sup>1</sup> executing their last statement, then operating system USES exit( ) system call to delete its context. Then all the resources held by that process like physical and virtual memory, buffers, open files etc., are taken back by the operating system. A process P can be terminated either by operation system or by the parent process of P.

A parent may terminate a process due to one of the following reasons,

- (i). When task given to the child is not required now.
- (ii). When child has taken more resources than its limit.
- (iii). The parent of the process is exiting, as a result all its children are deleted. This is called as cascaded termination.



# **FOURTH MODULE**

# Introduction to Deadlock

A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

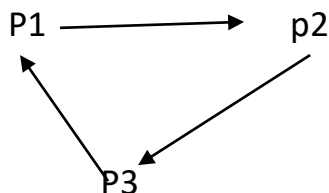
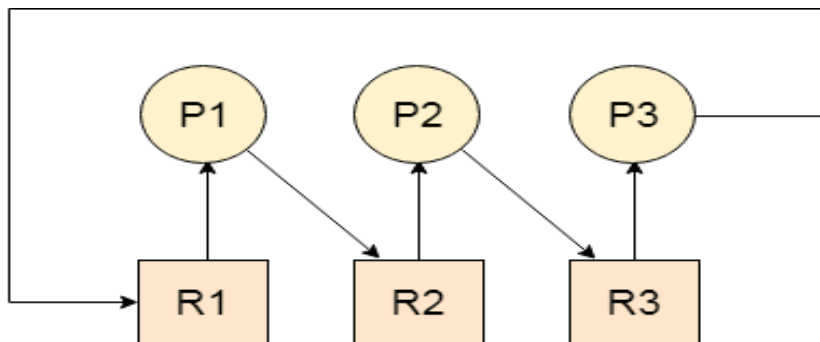
In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they are all waiting. The computer becomes unresponsive since all the processes got blocked.

P1  $\longrightarrow$  R1 (EX-1)

Here P1 is waiting for R1 to complete its execution.

P1  $\longleftarrow$  R1(Ex-2)

Here P1 is getting R1 or R1 resource is assigned to P1.



## Difference between Starvation and Deadlock

Deadlock	Starvation
Deadlock is a situation where no process got blocked and no process proceeds	Starvation is a situation where the low priority process got blocked and the high priority processes proceed.
Deadlock is an infinite waiting.	Starvation is a long waiting but not infinite.
Every Deadlock is always a starvation.	Every starvation need not be deadlock.
The requested resource is blocked by the other process.	The requested resource is continuously be used by the higher priority processes.
Deadlock happens when Mutual exclusion, hold and wait, No pre-emption and circular wait occurs simultaneously.	It occurs due to the uncontrolled priority and resource management.

## Necessary conditions for Deadlocks

### 1. Mutual Exclusion

A resource can only be shared in mutually exclusive manner. It implies, if two processes cannot use the same resource at the same time.

### 2. Hold and Wait

A process waits for some resources while holding another resource at the same time.

### 3. No pre-emption

The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

### 4. Circular Wait

All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

# Strategies for handling Deadlock

## 1. Deadlock Ignorance

Deadlock Ignorance is the most widely used approach among all the mechanism. In this approach, the Operating system assumes that deadlock never occurs. It simply ignores deadlock. The operating systems like Windows and Linux mainly focus upon performance. However, the performance of the system decreases if it uses deadlock handling mechanism all the time if deadlock happens. In these types of systems, the user has to simply restart the computer in the case of deadlock. Windows and Linux are mainly using this approach.

## 2. Deadlock prevention

Deadlock happens only when Mutual Exclusion, hold and wait, No pre-emption and circular wait holds simultaneously. If it is possible to violate one of the four conditions at any time then the deadlock can never occur in the system.

### →Mutual Exclusion

If a resource could have been used by more than one process at the same time then the process would have never been waiting for any resource.

### Spooling

For a device like printer, spooling can work. There is a memory associated with the printer which stores jobs from each of the process into it. Later, Printer collects all the jobs and prints each one of them according to FCFS. So the process doesn't have to wait for the printer. Although, Spooling can be an effective approach to violate mutual exclusion but it suffers from two kinds of problems.

1. This cannot be applied to every resource.
2. After some point of time, there may arise a race condition between the processes to get space in that spool.

### →Hold and Wait

Hold and wait condition lies when a process holds a resource and waiting for some other resource to complete its task. However, we have to find out some mechanism by which a process either doesn't hold any resource or doesn't wait. That means, a process must be assigned all the necessary resources before the execution starts.

The problem with the approach is:

1. Practically not possible.
2. Possibility of getting starved will be increases due to the fact that some process may hold a resource for a very long time.

### →No Pre-emption

Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock. This is not a good approach at all since if we take a resource away which is being used by the process then all the work which it has done till now can become inconsistent.

### →Circular Wait

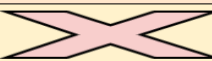
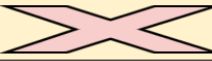
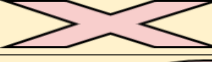

To violate circular wait, we can assign a priority number to each of the resource. A process can't request for a lesser priority resource. This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed.

**IF LATEST RN<NEW REQUEST**

**GRANTED**

**ELSE**

**NOT GRANTED**

Condition	Approach	Is Practically Possible?
Mutual Exclusion	Spooling	
Hold and Wait	Request for all the resources initially	
No Preemption	Snatch all the resources	
Circular Wait	Assign priority to each resources and order resources numerically	

Among all the methods, violating Circular wait is the only approach that can be implemented practically.

### 3. Deadlock avoidance

In deadlock avoidance, the operating system checks whether the system is in safe state or in unsafe state at every step which the operating system performs. The process continues until the system is in safe state. Once the system moves to unsafe state, the OS has to backtrack one step.

#### → Banker's Algorithm in Operating System

Banker's Algorithm is a resource allocation and deadlock avoidance algorithm. It is generally used to find if a safe sequence exists or not. But here we will determine the total number of safe sequences and print all safe sequences. In simple terms, it checks if allocation of any resource will lead to deadlock or not, OR is it safe to allocate a resource to a process and if not then resource is not allocated to that process. Determining a safe sequence (even if there is only 1) will assure that system will not go into deadlock.

The data structure used are:

- Available vector
- Max Matrix
- Allocation Matrix
- Need Matrix

**Example:****Input:**

Total Resources		R1 10	R2 5	R3 7		
Process	Allocation			Max		
	R1	R2	R3	R1	R2	R3
P1	0	1	0	7	5	3
P2	2	0	0	3	2	2
P3	3	0	2	9	0	2
P4	2	1	1	2	2	2

**Explanation:**

Total resources are **R1 = 10, R2 = 5, R3 = 7** and allocated resources are R1 = (0+2+3+2 =) 7, R2 = (1+0+0+1 =) 2, R3 = (0+0+2+1 =) 3. Therefore, remaining resources are R1 = (10 – 7 =) 3, R2 = (5 – 2 =) 3, R3 = (7 – 3 =) 4.

**Remaining available = Total resources – allocated resources**  
**and**

**Remaining need = max – allocated**

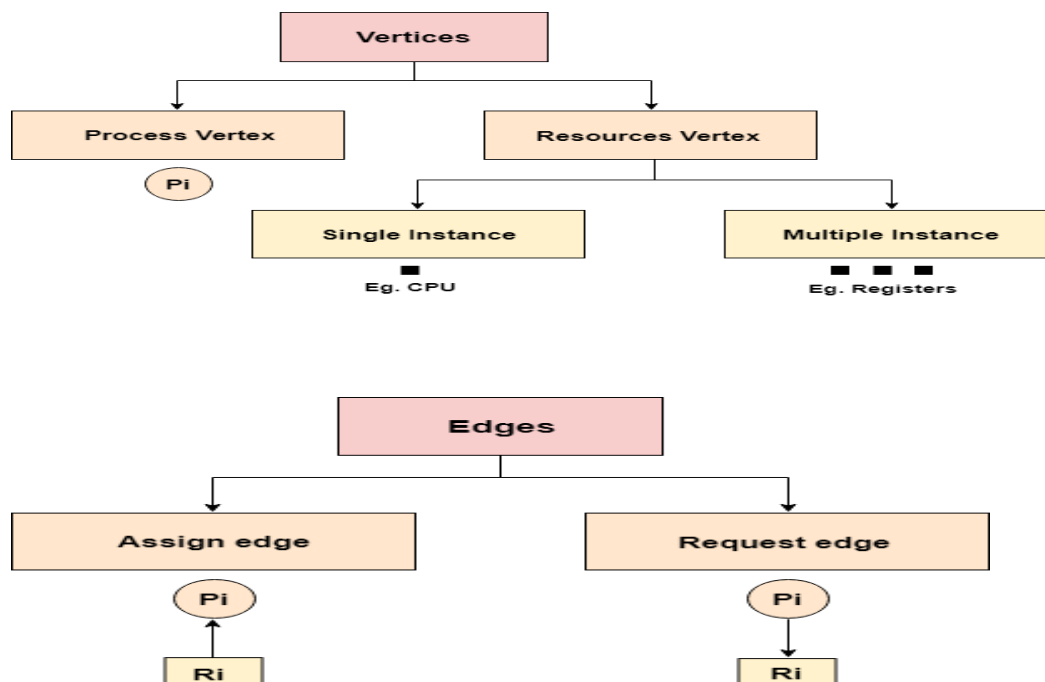
Process	Max			Allocation			Available			Needed		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	7	5	3	0	1	0	3	3	4	7	4	3
P2	3	2	2	2	0	0				1	2	2
P3	9	0	2	3	0	2				6	0	0
P4	2	2	2	2	1	1				0	1	1
							7	2	3			

So, we can start from either P2 or P4. We cannot satisfy remaining need from available resources of either P1 or P3 in first or second attempt step of Banker's algorithm. There are only four possible safe sequences. These are :

P2→ P4→ P1→ P3  
P2→ P4→ P3→ P1  
P4→ P2→ P1→ P3  
P4→ P2→ P3→ P1

## →Resource Allocation Graph

The resource allocation graph is the pictorial representation of the state of a system. It gives complete information about all the processes which are holding some resources or waiting for some resources. In Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle. Let's see the types of vertices and edges in detail. Vertices are mainly of two types, Resource and process. Each of them will be represented by a different shape. Circle represents process while rectangle represents resource.



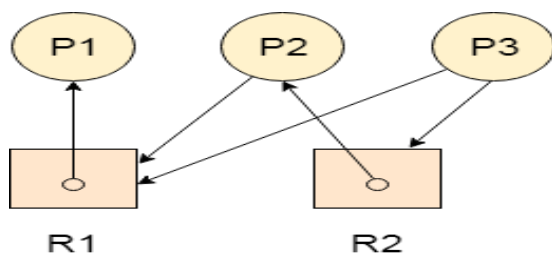


## Example

Let's consider 3 processes P1, P2 and P3, and two types of resources R1 and R2.

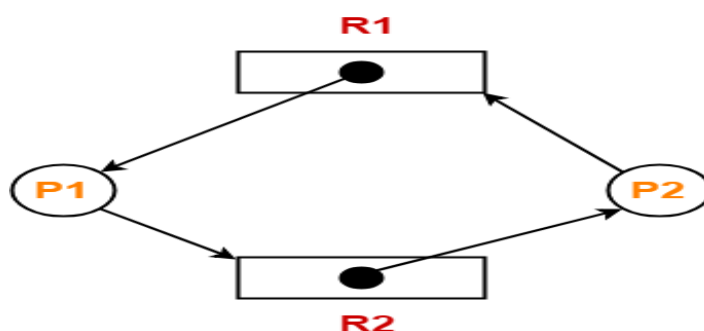
The resources are having 1 instance each.

According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 is waiting for R1 as well as R2. The graph is deadlock free since no cycle is being formed in the graph.



## PROBLEMS BASED ON DETECTING DEADLOCK USING RAG-

**Problem-01:** Consider the resource allocation graph in the figure-



Find if the system is in a deadlock state otherwise find a safe sequence.

**Solution-**

**Method-01:**

The given resource allocation graph is single instance with a cycle contained in it. Thus, the system is definitely in a deadlock state.

### Method-02:

Using the given resource allocation graph, we have-

	Allocation		Need	
	R1	R2	R1	R2
Process P1	1	0	0	1
Process P2	0	1	1	0

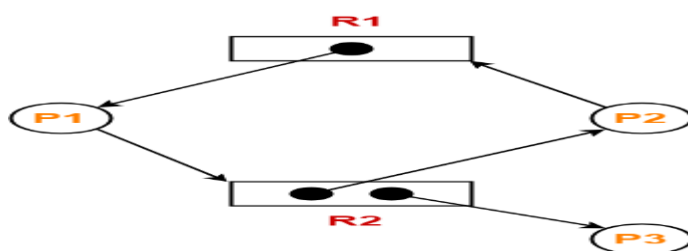
Available = [ R1 R2 ] = [ 0 0 ]

Now,

There are no instances available currently and both the processes require a resource to execute. Therefore, none of the process can be executed and both keep waiting infinitely. Thus, the system is in a deadlock state.

### Problem-02:

Consider the resource allocation graph in the figure-



Find if the system is in a deadlock state otherwise find a safe sequence.

### Solution-

The given resource allocation graph is multi instance with a cycle contained in it. So, the system may or may not be in a deadlock state. Using the given resource allocation graph, we have-

	Allocation		Need	
	R1	R2	R1	R2
<b>Process P1</b>	1	0	0	1
<b>Process P2</b>	0	1	1	0
<b>Process P3</b>	0	1	0	0

Available = [ R1 R2 ] = [ 0 0 ]

**Step-01:**

Since process P3 does not need any resource, so it executes. After execution, process P3 releases its resources.

**Available**

$$= [ 0 0 ] + [ 0 1 ]$$

$$= [ 0 1 ]$$

**Step-02:**

With the instances available currently, only the requirement of the process P1 can be satisfied. So, process P1 is allocated the requested resources. It completes its execution and then free up the instances of resources held by it.

**Available**

$$= [ 0 1 ] + [ 1 0 ]$$

$$= [ 1 1 ]$$

**Step-03:**

With the instances available currently, the requirement of the process P2 can be satisfied. So, process P2 is allocated the requested resources. It completes its execution and then free up the instances of resources held by it.

**Available**

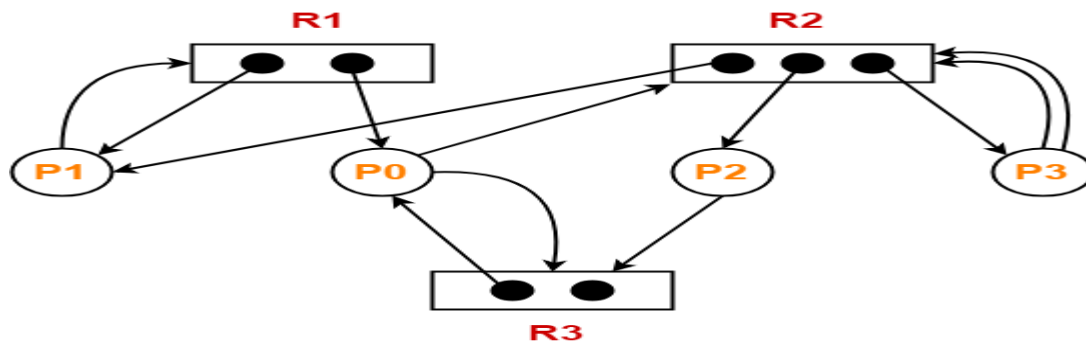
$$= [ 1 \ 1 ] + [ 0 \ 1 ]$$

$$= [ 1 \ 2 ]$$

Thus,

There exists a safe sequence P3, P1, P2 in which all the processes can be executed. So, the system is in a safe state.

**Problem-03:** Consider the resource allocation graph in the figure-



Find if the system is in a deadlock state otherwise find a safe sequence.

**Solution-**

The given resource allocation graph is multi instance with a cycle contained in it. So, the system may or may not be in a deadlock state.

Using the given resource allocation graph, we have-

	Allocation			Need		
	R1	R2	R3	R1	R2	R3
Process P0	1	0	1	0	1	1
Process P1	1	1	0	1	0	0
Process P2	0	1	0	0	0	1
Process P3	0	1	0	0	2	0

Available = [ R1 R2 R3 ] = [ 0 0 1 ]

**Step-01:**

With the instances available currently, only the requirement of the process P2 can be satisfied. So, process P2 is allocated the requested resources. It completes its execution and then free up the instances of resources held by it.

**Available**

$$= [0\ 0\ 1] + [0\ 1\ 0]$$

$$= [0\ 1\ 1]$$

**Step-02:**

With the instances available currently, only the requirement of the process P0 can be satisfied. So, process P0 is allocated the requested resources. It completes its execution and then free up the instances of resources held by it.

**Available**

$$= [0\ 1\ 1] + [1\ 0\ 1]$$

$$= [1\ 1\ 2]$$

**Step-03:**

With the instances available currently, only the requirement of the process P1 can be satisfied. So, process P1 is allocated the requested resources. It completes its execution and then free up the instances of resources held by it.

**Available**

$$= [1\ 1\ 2] + [1\ 1\ 0]$$

$$= [2\ 2\ 2]$$

**Step-04:**

With the instances available currently, the requirement of the process P3 can be satisfied. So, process P3 is allocated the requested resources. It completes its execution and then free up the instances of resources held by it.

**Available**

$$= [2\ 2\ 2] + [0\ 1\ 0]$$

$$= [2\ 3\ 2]$$

## 4. Deadlock detection and recovery

This approach let the processes fall in deadlock and then periodically check whether deadlock occur in the system or not. If it occurs then it applies some of the recovery methods to the system to get rid of deadlock.

**In single instanced resource types**, if a cycle is being formed in the system then there will definitely be a deadlock. We have to apply the safety algorithm on the system by converting the resource allocation graph into the allocation matrix and request matrix.

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

### →For Resource

- **Preempt the resource:-**We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner.
- **Rollback to a safe state:-**The operating system can rollback the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.

### →For Process

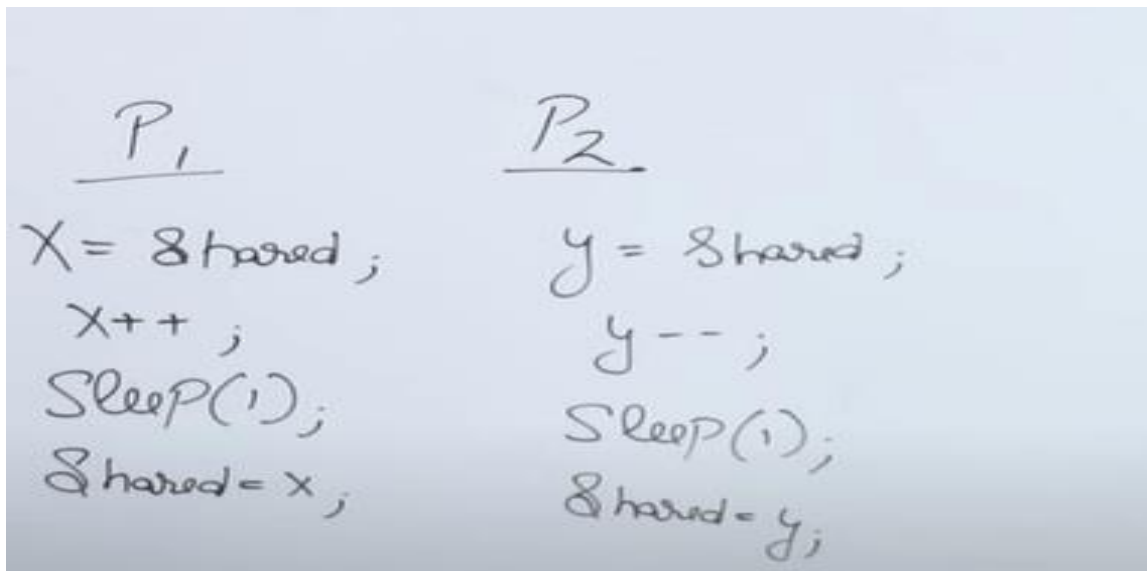
- **Kill a process:-**Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.
- **Kill all process:-**Killing all process will lead to inefficiency in the system because all the processes will execute again from starting.

# Process Synchronization

On the basis of synchronization, processes are categorized as one of the following two types:

- **Independent Process:** Execution of one process does not affect the execution of other processes.
- **Cooperative Process:** Execution of one process affects the execution of other processes.

Process synchronization problem arises in the case of Cooperative processes because of sharing resources among them.



## Race Condition

When more than one processes are executing the same code or accessing the same memory or any shared variable in that condition there is a possibility that the output or the value of the shared variable is wrong. So all the processes doing race to say that my output is correct. This condition is known as race condition.



# Critical Section Problem

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.



## Solution to Critical Section Problem

A solution to the critical section problem must satisfy the following three conditions:

### 1. Mutual Exclusion

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

### 2. Progress

Progress means that if one process doesn't need to execute into critical section then it should not stop other processes to get into the critical section.

### 3. Bounded Waiting

The process must not be endlessly waiting for getting into the critical section.

### 4. Architectural Neutrality

Our mechanism must be architectural natural. It means that if our solution is working fine on one architecture then it should also run on the other ones as well.

# Peterson Solution

This is a software mechanism implemented at user mode. It is a busy waiting solution can be implemented for only two processes. It uses two variables that are turn variable and interested variable.

**The Code of the solution is given below**

```
# define N 2
# define TRUE 1
# define FALSE 0
int interested[N] = FALSE, turn;
voidEntry_Section (int process)
{
    int other;
    other = 1-process;
    interested[process] = TRUE;
    turn = process;
    while (interested [other] ==True && turn==process);
}
voidExit_Section (int process)
{
    interested [process] = FALSE;
}
```

## →Mutual Exclusion

In entry section, the while condition involves the criteria for two variables therefore a process cannot enter in the critical section until the other process is interested and the process is the last one to update turn variable.

### **→Progress**

An uninterested process will never stop the other interested process from entering in the critical section. If the other process is also interested then the process will wait.

### **→Bounded waiting**

In Peterson solution, A deadlock can never happen because the process which first sets the turn variable will enter in the critical section for sure. Therefore, if a process is pre-empted after executing line number 4 of the entry section then it will definitely get into the critical section in its next chance.

### **→Portability**

This is the complete software solution and therefore it is portable on every hardware.

# InterProcess Communication

Interprocess communication (IPC) method allows the processes to exchange data along with various information. There are two primary models of interprocess communication:

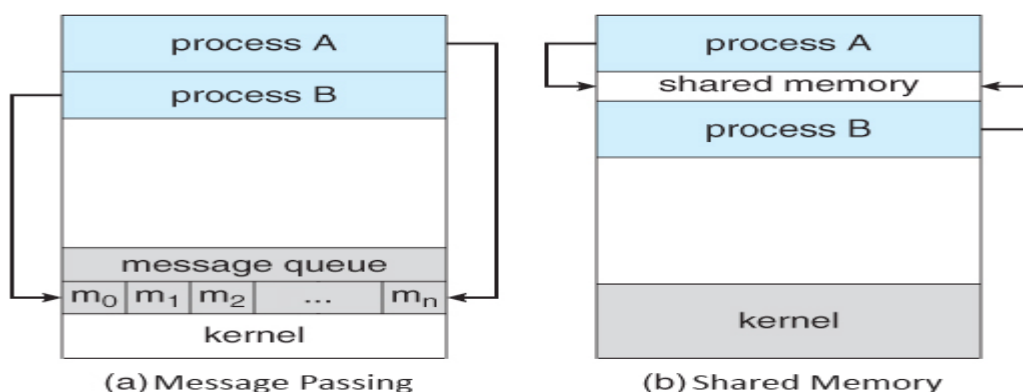
→ **Shared memory**: -A region of memory which is shared by cooperating processes gets established. Processes can be then able to exchange information by reading and writing all the data to the shared region.

→ **Message passing**: -In the message-passing form, communication takes place by way of messages exchanged among the cooperating processes.

There are numerous reasons for providing an environment or situation which allows process co-operation:

- **Information sharing**: Since some users may be interested in the same piece of information (for example, a shared file), you must provide a situation for allowing concurrent access to that information.
- **Computation speedup**: If you want a particular work to run fast, you must break it into sub-tasks where each of them will get executed in parallel with the other tasks.
- **Convenience**: Even a single user may work on many tasks at a time. For example, a user may be editing, formatting, printing, and compiling in parallel.

The two communications models are contrasted in the figure below:



# Semaphores in Process Synchronization

Semaphore was proposed by Dijkstra in 1965 which is a very significant technique to manage concurrent processes by using a simple integer value, which is known as semaphore. Semaphore is simply a variable which is non-negative and shared between threads with operations wait and signal, this variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment. The two most common kinds of semaphores are counting semaphores and binary semaphores. Counting semaphore can take non-negative integer values and Binary semaphore can take the value 0 & 1. Only.

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

- **Wait:** Decrements the value of its argument  $S$ , as soon as it would become non-negative (greater than or equal to 1).
- **Signal:** Increments the value of its argument  $S$ , as there is no more process blocked on the queue.

## Limitations of Semaphores

- **Priority Inversion** is a big limitation of semaphores.
- Their use is not enforced, but is by convention only.
- With improper use, a process may block indefinitely. Such a situation is called **Deadlock**.

## → Binary Semaphore or Mutex

In counting semaphore, Mutual exclusion was not provided because we have the set of processes which required executing in the critical section simultaneously. However, Binary Semaphore strictly provides mutual exclusion. Here, instead of having more than 1 slot available in the critical section, we can only have at most 1 process in the critical section. The semaphore can have only two values, 0 or 1.

### Down (Bsemaphore S)

```
{
    if (s.value == 1)
        S.value = 0;
    else
    {
        Block the process (PCB) in Suspend list;
        sleep();
    }
}
```

### Up (Bsemaphore S)

```
{
    if (Suspend list is empty)
        S.Value = 1;
    else
    {
        Select a process from Suspend list ;
        Wakeup();
    }
}
```

## → Counting Semaphore

There are the scenarios in which more than one processes need to execute in critical section simultaneously. However, counting semaphore can be used when we need to have more than one process in the critical section at the same time.

### Down (Semaphore S)

```
{
S.value = S.value - 1;
if (S.value < 0)
{
    put_process(PCB) in L;
    Sleep();
}
else
    return;
}
```

### Up (Semaphore s)

```
{
S.value = S.value + 1;
if (S.value <= 0)
{
    select a process from L;
    wake-up()
}
}
```

A process which wants to enter in the critical section first decrease the semaphore value by 1 and then check whether it gets negative or not. If it gets negative then the process is pushed in the list of blocked processes (i.e. q) otherwise it gets enter in the critical section. When a process exits from the critical section, it increases the counting semaphore by 1 and then checks whether it is negative or zero. If it is negative then that means that at least one process is waiting in the blocked state hence, to ensure bounded waiting, the first process among the list of blocked processes will wake up and gets enter in the critical section.

# Classical Problems of Synchronization

Below are some of the classical problems depicting flaws of process synchronization in systems where cooperating processes are present?

1. Bounded Buffer (Producer-Consumer) Problem
2. Dining Philosophers Problem
3. The Readers Writers Problem
4. Sleeping Barber Problem

## →The Readers Writers Problem

1. The readers-writers problem relates to an object such as a file that is shared between multiple processes. Some of these processes are readers i.e. they only want to read the data from the object and some of the processes are writers i.e. they want to write into the object.
2. To solve this situation, when a writer is accessing the object, no reader or writer may access it. However, multiple readers can access the object at the same time. This can be implemented using semaphores.

**Initially int rc=0;**

**Semaphore mutex=1;**

**Semaphore db=1;**



### Reader Process

```
{
    While(true)
    {
        Down(mutex);
        rc=rc+1;
        If(rc==1) then Down(db);
        Up(mutex);
        FILE;    //CRITICAL SECTION
        Down(mutex);
        rc=rc-1;
        If(rc==0) then Up(db);
        Up(mutex);
        Process_data();
    }
}
```

### Writer Process

```
{
    While(true)
    {
        Down(db);
        FILE; //CRITICAL SECTION
        Up(db);
    }
}
```

## → Bounded Buffer Problem

1. This problem is generalised in terms of the **Producer Consumer problem**, where a **finite** buffer pool is used to exchange messages between producer and consumer processes. Because the buffer pool has a maximum size, this problem is often called the **Bounded buffer problem**.
2. Solution to this problem is, creating two counting semaphores "full" and "empty" to keep track of the current number of full and empty buffers respectively.

## Problem Statement

There is a buffer of  $n$  slots and each slot is capable of storing one unit of data. There are two processes running, namely, **producer** and **consumer**, which are operating on the buffer. A producer tries to insert data into an empty slot of the buffer. A consumer tries to remove data from a filled slot in the buffer.



## Issues

- Mutual Exclusion
- Overflow: Buffer is full
- Underflow: Buffer is empty

## Solution

One solution of this problem is to use semaphores.

- **mutex**, a binary semaphore which is used to acquire and release the lock.
- **Empty**, a counting semaphore whose initial value is the number of slots in the buffer, since, initially all slots are empty.
- **Full**, a counting semaphore whose initial value is 0.

--	--	--

**Empty=3Full=0mutex=1**

## The Producer Operation

```
do
{
    // Produce the item;
    If(buffer is not full)
    {
        Down(Empty);
        Down(mutex);
        Add item to the buffer; //CRITICAL SECTION
    }
    Else
    {
        Print "Overflow";
        Exit(0);
    }
    Up(mutex);
    Up(Full);
} while(TRUE);
```

1. We can see that a producer first waits until there is at least one empty slot.
2. Then it decrements the **empty** semaphore because, there will now be one less empty slot, since the producer is going to insert data in one of those slots.
3. Then, it acquires lock on the buffer, so that the consumer cannot access the buffer until producer completes its operation.
4. After performing the insert operation, the lock is released and the value of **full** is incremented because the producer has just filled a slot in the buffer.

# The Consumer Operation

```
do
{
    If(buffer is empty)
    {
        Print "Underflow";
        Exit(0);
    }
    Else
    {
        Down(Full);
Down(mutex);
//Remove item from the buffer //Critical section
    }
Up(mutex);
Up(Empty);
} while(TRUE);
```

- The consumer waits until there is at least one full slot in the buffer.
- Then it decrements the **full** semaphore because the number of occupied slots will be decreased by one, after the consumer completes its operation.
- After that, the consumer acquires lock on the buffer.
- Following that, the consumer completes the removal operation so that the data from one of the full slots is removed.
- Then, the consumer releases the lock.
- Finally, the **empty** semaphore is incremented by 1, because the consumer has just removed data from an occupied slot, thus making it empty.

## → Dining Philosophers Problem

- The dining philosopher's problem involves the allocation of limited resources to a group of processes in a deadlock-free and starvation-free manner.
- There are five philosophers sitting around a table, in which there are five chopsticks/forks kept beside them and a bowl of rice in the centre, When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.

```
Void Philosopher(void)
{
    While(true)
    {
        Thinking();
        Take_fork(i);

Take_fork((i+1)%N);
        EAT();
        Put_fork(i);

        Put_fork((i+1)%N);
    }
}
```

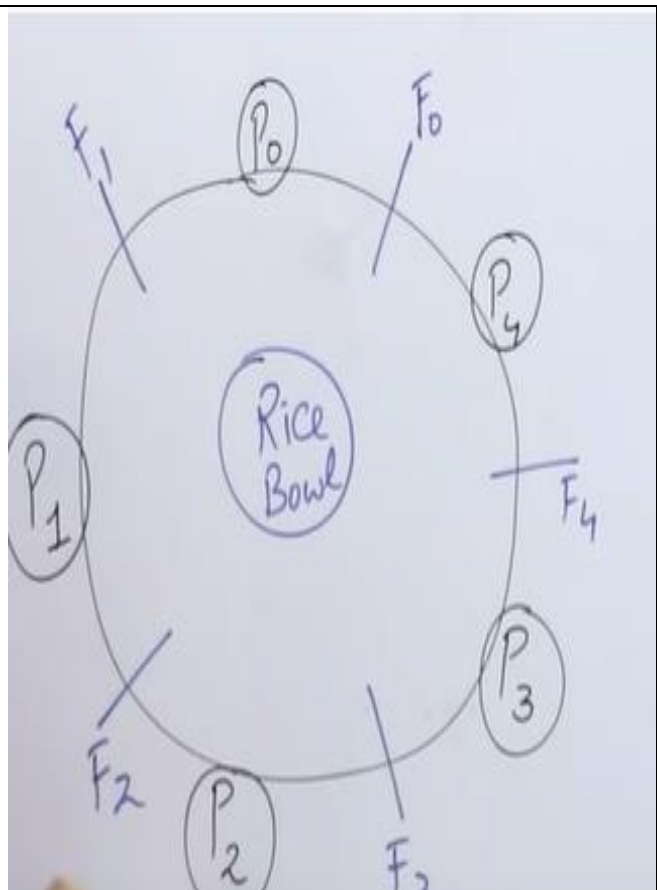
P0->F0 & F1

P1->F1 & F2

P2->F2 & F3

P3->F3 & F4

P4->F4 & F0



## Solution

A solution of the Dining Philosophers Problem is to use a semaphore to represent a chopstick. A chopstick can be picked up by executing a wait operation on the semaphore and released by executing a signal semaphore.

The structure of the chopstick is shown below:

### **Semaphore chopstick [5];**

Initially the elements of the chopstick are initialized to 1 as the chopsticks are on the table and not picked up by a philosopher.

Let's modify the above code of the Dining Philosopher Problem by using semaphore operations wait and signal, the desired code looks like

**S[5]=1    S0=1   S1=1   S2=1   S3=1   S4=1    Initially**

<pre> <b>void</b> Philosopher { <b>while</b>(1) {     Down( take_chopstickC[i] );     Down( take_chopstickC[(i+1) % 5] ) ;     ..     . <b>EATING</b>     .     Up( put_chopstickC[i] );     Up( put_chopstickC[ (i+1) % 5] ) ;     .     . THINKING } } </pre>	<b>P0</b>	<b>S0</b>	<b>S1</b>
	<b>P1</b>	<b>S1</b>	<b>S2</b>
	<b>P2</b>	<b>S2</b>	<b>S3</b>
	<b>P3</b>	<b>S3</b>	<b>S4</b>
	<b>P4</b>	<b>S4</b>	<b>S0</b>

→ In the above structure, first wait operation is performed on chopstick[i] and chopstick[ (i+1) % 5]. This means that the philosopher i have picked up the chopsticks on his sides. Then the eating function is performed.

→ After that, signal operation is performed on chopstick[i] and chopstick[ (i+1) % 5]. This means that the philosopher i has eaten and put down the chopsticks on his sides. Then the philosopher goes back to thinking.

## **Difficulty with the solution**

The above solution makes sure that no two neighbouring philosophers can eat at the same time. But this solution can lead to a deadlock. This may happen if all the philosophers pick their left chopstick simultaneously. Then none of them can eat and deadlock occurs.

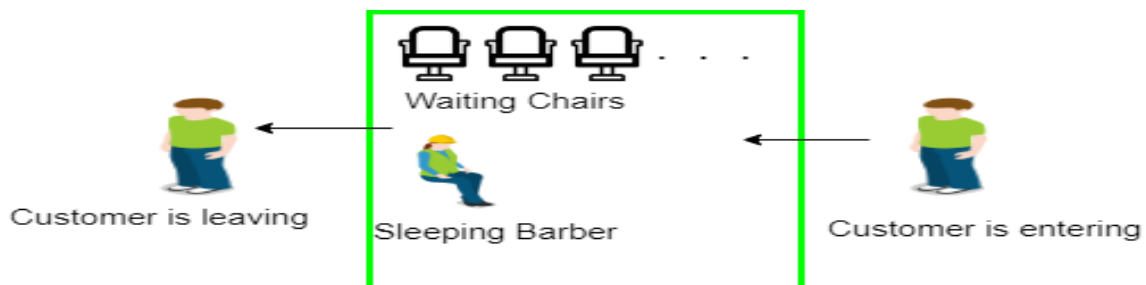
Some of the ways to avoid deadlock are as follows:

1. There should be at most four philosophers on the table.
2. An even philosopher should pick the right chopstick and then the left chopstick while an odd philosopher should pick the left chopstick and then the right chopstick.
3. A philosopher should only be allowed to pick their chopstick if both are available at the same time.

## →Sleeping Barber problem

**Problem:** The analogy is based upon a hypothetical barber shop with one barber. There is a barber shop which has one barber, one barber chair, and  $n$  chairs for waiting for customers if there are any to sit on the chair.

- If there is no customer, then the barber sleeps in his own chair.
- When a customer arrives, he has to wake up the barber.
- If there are many customers and the barber is cutting a customer's hair, then the remaining customers either wait if there are empty chairs in the waiting room or they leave if no chairs are empty.



This solution uses three semaphores, one for customers (counts waiting customers), one for the barber (idle - 0 or busy - 1) and a mutual exclusion semaphore, mutex. When the barber arrives for work, the barber procedure is executed blocking the barber on the customer semaphore until a customer arrives. When a customer arrives, the customer procedure is executed which begins by acquiring mutex to enter a critical region. Subsequent arriving customers have to wait until the first customer has released mutex. After acquiring mutex, a customer checks to see if the number of waiting customers is less than the number of chairs. If not, mutex is released and the customer leaves without a haircut. If there is an available chair, the waiting counter is incremented, the barber is awakened, the customer releases mutex, the barber grabs mutex, and begins the haircut. Once the customer's hair is cut, the customer leaves. The barber then checks to see if there is another customer. If not, the barber takes a nap.



```
#define CHAIRS 5           //No of waiting chairs=5
semaphore customers = 0;   //No of customers
semaphore barbers=0;
semaphore mutex=1;
int waiting=0;//No of waiting customers
```

```
void barber(void)
{
    while (TRUE)
    {
        P(customers);
        P(mutex);
        waiting=waiting-1;
        V(barbers);
        V(mutex);
        Cut_hair();
    }
}
```

```
void customer(void)
{
    P(mutex);
    if (waiting < CHAIRS)
    {
        waiting=waiting+1;
        V(customers);
        V(mutex);
        P(barbers);
        get_haircut();
    }
    else
        V(mutex);
}
```

# **FIFTH MODULE**

# Introduction to Memory Management

Memory management is the functionality of operating system which manages and keep track of processes moving from secondary memory to main memory.

Main Memory refers to a physical memory that is the internal memory to the computer. Every program we execute and every file we access must be copied from a storage device into main memory.

## Functionality

- Keep track of what memory is in use and what memory is free.
- Allocate free memory when processes needed and deallocate when they don't need it.
- Prohibiting user program to enter in operating system area and other user program area.
- Keep updating the status of processes.

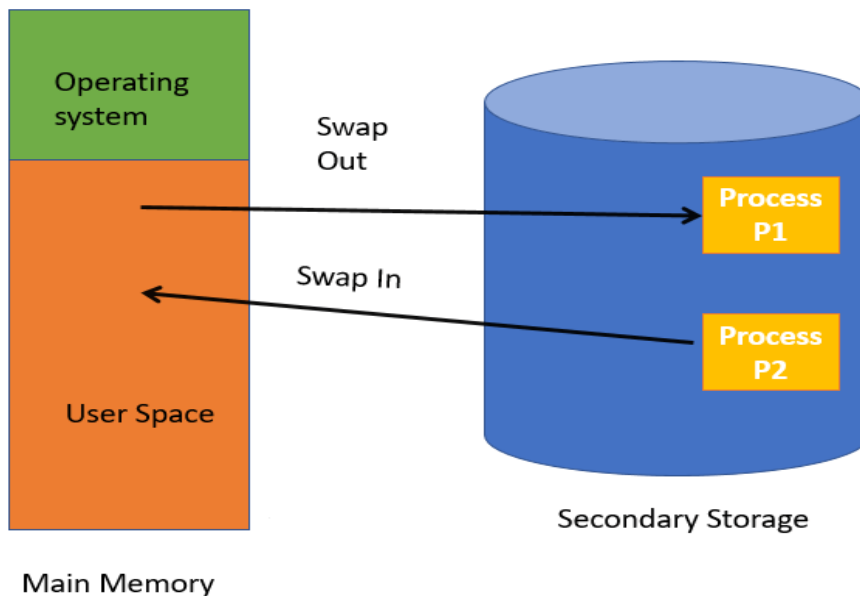
## Goals

- Maximise CPU utilisation
- Minimise response time
- Security to user programs and operating system
- Prioritise important processes

All the programs are loaded in the main memory for execution. Sometimes complete program is loaded into the memory, but sometimes a certain part or routine of the program is loaded into the main memory only when it is called by the program, this mechanism is called **Dynamic Loading**, this enhance the performance. Also, at times one program is dependent on some other program. In such a case, rather than loading all the dependent programs, CPU links the dependent programs to the main executing program when it's required. This mechanism is known as **Dynamic Linking**.

# Swapping

A process needs to be in memory for execution. But sometimes there is not enough main memory to hold all the currently active processes in a timesharing system. So, excess process are kept on disk and brought in to run dynamically. Swapping is the process of bringing in each process in main memory, running it for a while and then putting it back to the disk.



## Benefits of Swapping

- It offers a higher degree of multiprogramming. By multiprogramming we mean there will be more than one process in the main memory and if the running process wants to wait for an event like I/O then instead of sitting ideal CPU will make a context switch and will pick another process.
- It helps to get better utilization of memory.
- Minimum wastage of CPU time on completion so it can easily be applied to a priority-based scheduling method to improve its performance.

# Memory Management Techniques

- Contiguous memory allocation
- Non-contiguous memory allocation

## a) Contiguous memory allocation

→Contiguous memory allocation is a memory allocation technique.

→It allows storing the process only in a contiguous fashion.

→Thus, entire process has to be stored as a single entity at one place inside the memory.

### Techniques-

There are two popular techniques used for contiguous memory allocation-

→**Static Partitioning/Fixed Size Partitioning**

→**Dynamic Partitioning/Variable Size Partitioning**

## Fixed Partitioning

In this technique, the main memory is divided into partitions of equal or different sizes. The operating system always resides in the first partition while the other partitions can be used to store user processes. The memory is assigned to the processes in contiguous way.

In fixed partitioning,

- The partitions cannot overlap.
- A process must be contiguously present in a partition for the execution.

There are various cons of using this technique.

### **1. Internal Fragmentation**

If the size of the process is lesser than the total size of the partition then some size of the partition get wasted and remain unused. This wastage of the memory is called internal fragmentation.

As shown in the image below, the 4 MB partition is used to load only 3 MB process and the remaining 1 MB got wasted.

## 2. External Fragmentation

The total unused space of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form. This is called as external fragmentation.

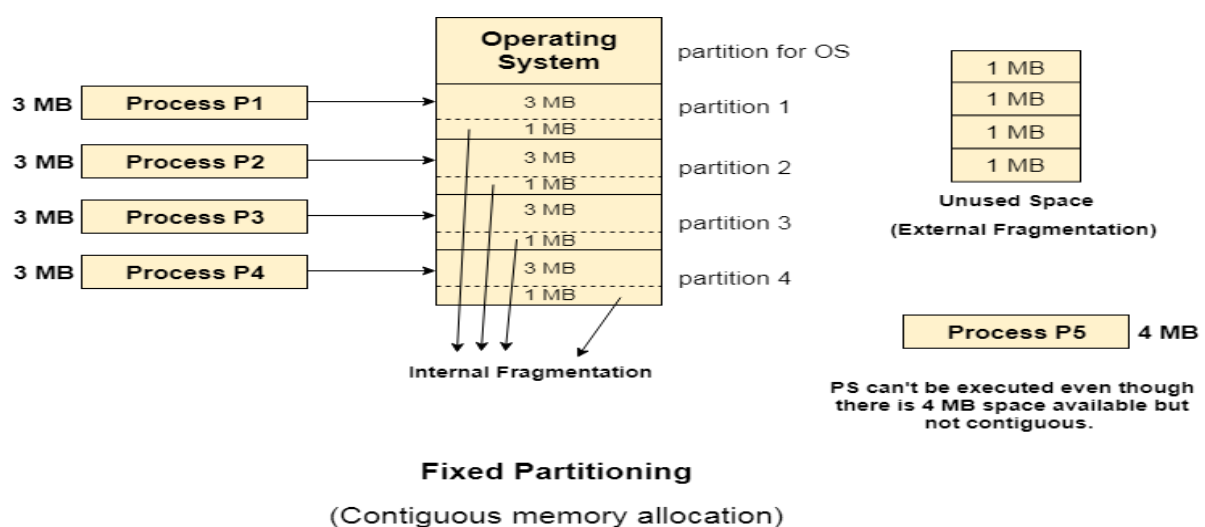
As shown in the image below, the remaining 1 MB space of each partition cannot be used as a unit to store a 4 MB process. Despite of the fact that the sufficient space is available to load the process, process will not be loaded.

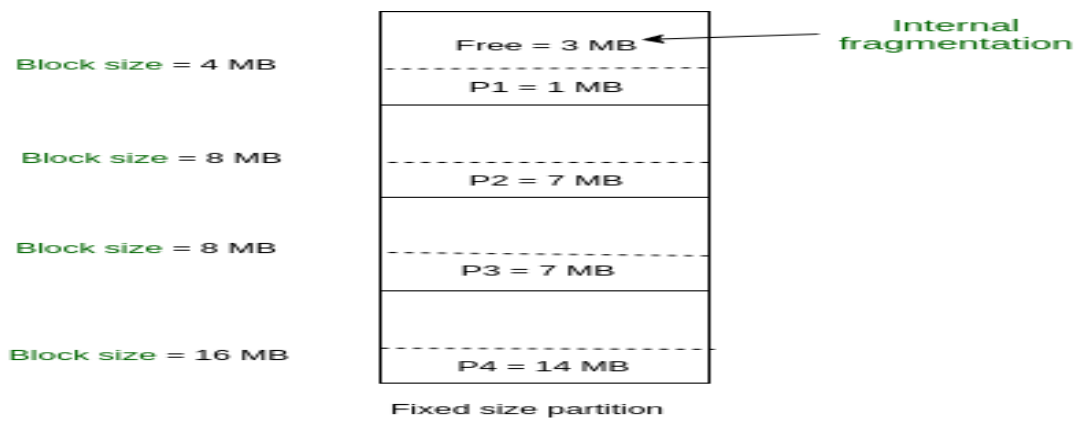
## 3. Limitation on the size of the process

If the process size is larger than the size of maximum sized partition then that process cannot be loaded into the memory. Therefore, a limitation can be imposed on the process size that is it cannot be larger than the size of the largest partition.

## 4. Degree of multiprogramming is less

By Degree of multi programming, we simply mean the maximum number of processes that can be loaded into the memory at the same time. In fixed partitioning, the degree of multiprogramming is fixed and very less due to the fact that the size of the partition cannot be varied according to the size of processes.

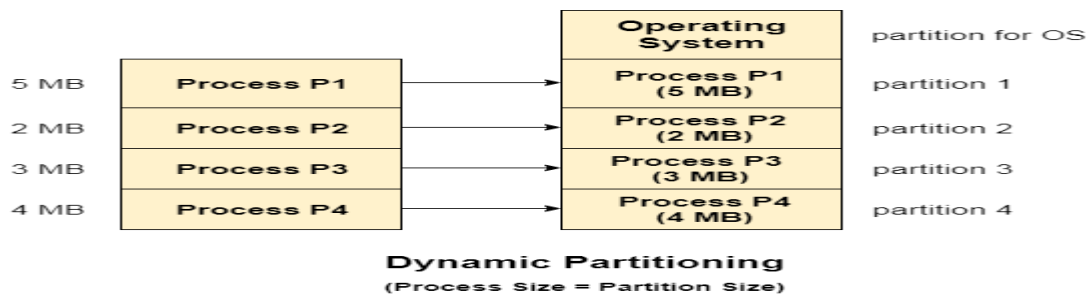




Internal Fragmentation	External Implementation
The difference between the memory allocated and the required memory is called internal fragmentation.	The unused spaces formed between non-contiguous memory fragments are too small to serve a new process request. This is called external fragmentation.
It occurs when main memory is divided into fixed size blocks regardless of the size of the process.	It occurs when memory is allocated to process dynamically based on process requests.
It refers to the unused space in the partition which resides within an allocated region.	It refers to the unused memory blocks that are too small to handle a request.
Internal fragmentation can be eliminated by allocating memory to processes dynamically.	External fragmentation can be eliminated by compaction, segmentation and paging.

# Dynamic Partitioning

In this technique, the partition size is not declared initially. It is declared at the time of process loading. The first partition is reserved for the operating system. The remaining space is divided into parts. The size of each partition will be equal to the size of the process.



## Advantages of Dynamic Partitioning

### 1. No Internal Fragmentation

It is clear that there will not be any internal fragmentation because there will not be any unused remaining space in the partition.

### 2. No Limitation on the size of the process

Here, In Dynamic partitioning, the process size can't be restricted since the partition size is decided according to the process size.

### 3. Degree of multiprocessing is dynamic

Due to the absence of internal fragmentation, there will not be any unused space in the partition hence more processes can be loaded in the memory at the same time.

## Disadvantages

### External Fragmentation

Absence of internal fragmentation doesn't mean that there will not be external fragmentation. The rule says that the process must be contiguously present in the main memory to get executed. We need to change this rule to avoid external fragmentation.



## Algorithms for Partition Allocation-

In **Partition Allocation**, when there is more than one partition freely available to accommodate a process's request, a partition must be selected. To choose a particular partition, a partition allocation method is needed.

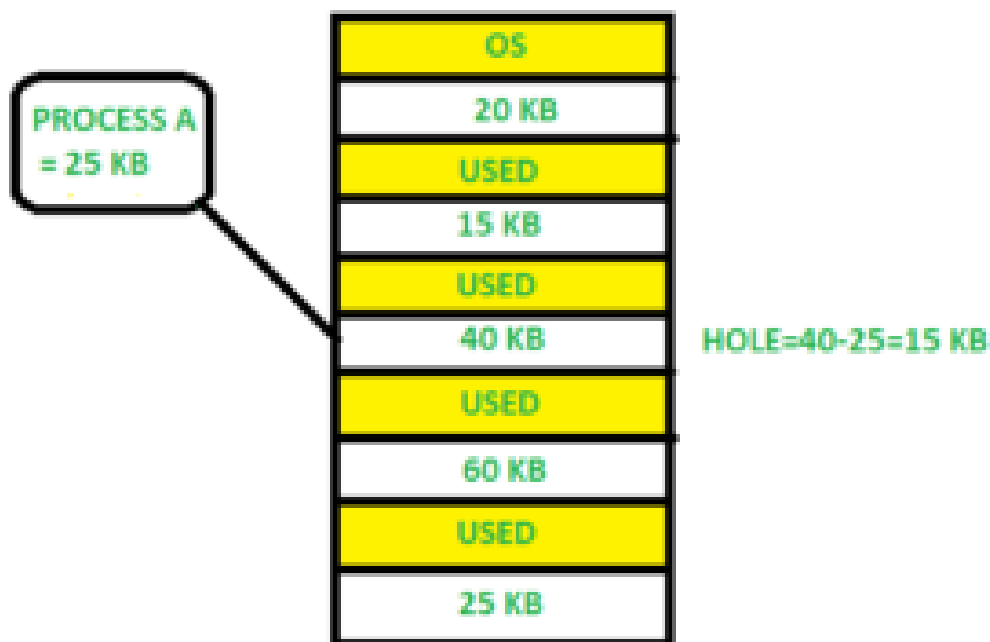
→ **First Fit Algorithm**

→ **Best Fit Algorithm**

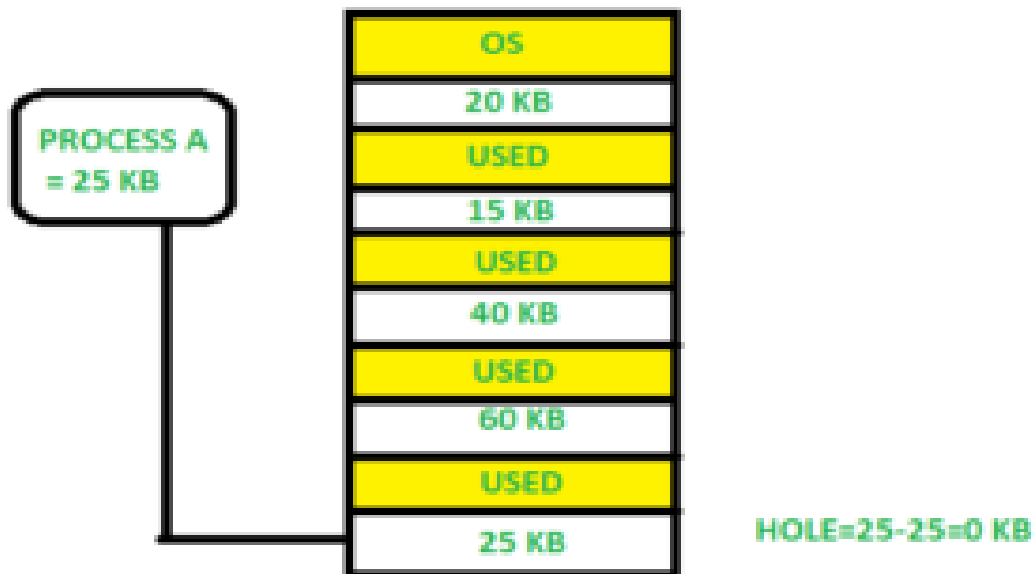
→ **Worst Fit Algorithm**

→ **Next Fit**

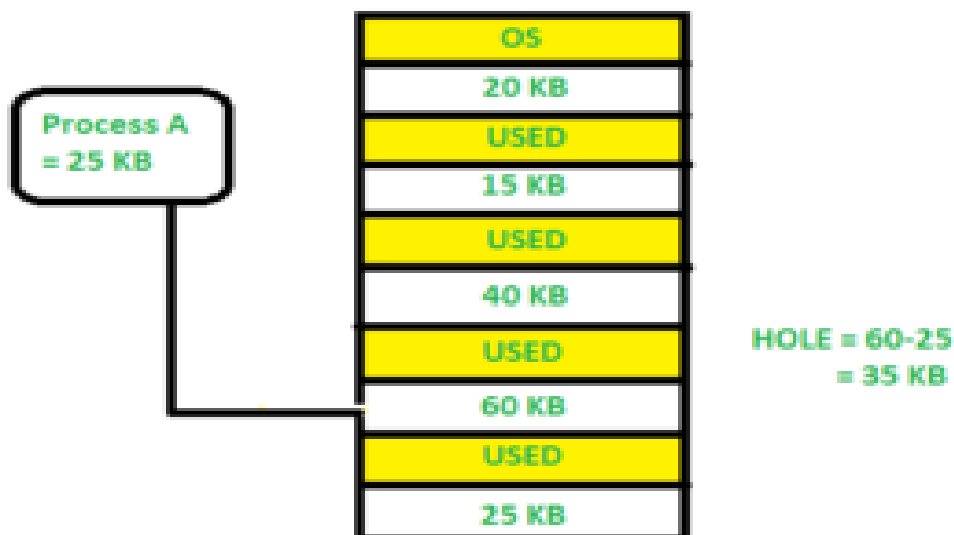
**1. First Fit:** In the first fit, the partition is allocated which is first sufficient block from the top of Main Memory. It scans memory from beginning and chooses the first available block that is large enough. Thus it allocates the first hole that is large enough.



**2. Best Fit** Allocate the process to the partition which is the first smallest sufficient partition among the free available partition. It search the



**3. Worst Fit :** It search the entire list of hole to find the largest hole and allocate it to process.



**4. Next Fit** Next fit is similar to the first fit but it will search for the first sufficient partition from the last allocation point.

**Question:** Consider the requests from processes in given order 300K, 25K, 125K and 50K. Let there be two blocks of memory available of size 150K followed by a block size 350K.

Which of the following partition allocation schemes can satisfy above requests?

- A) Best fit but not first fit.
- B) First fit but not best fit.
- C) Both First fit & Best fit.
- D) neither first fit nor best fit.

**Solution:**

→**Best Fit:**

300K is allocated from block of size 350K. 50 is left in the block. 25K is allocated from the remaining 50K block. 25K is left in the block. 125K is allocated from 150 K block. 25K is left in this block also. 50K can't be allocated even if there is 25K + 25K space available.

→**First Fit:**

300K request is allocated from 350K block, 50K is left out. 25K is be allocated from 150K block, 125K is left out. Then 125K and 50K are allocated to remaining left out partitions. So, first fit can handle requests.

**So option B is the correct choice.**

## **b) Non-contiguous memory allocation**

In the non-contiguous memory allocation, a process will acquire the memory space but it is not at one place it is at the different locations according to the process requirement. This technique of non-contiguous memory allocation reduces the wastage of memory which leads to internal and external fragmentation.

**Non-contiguous memory allocation** is of different types,

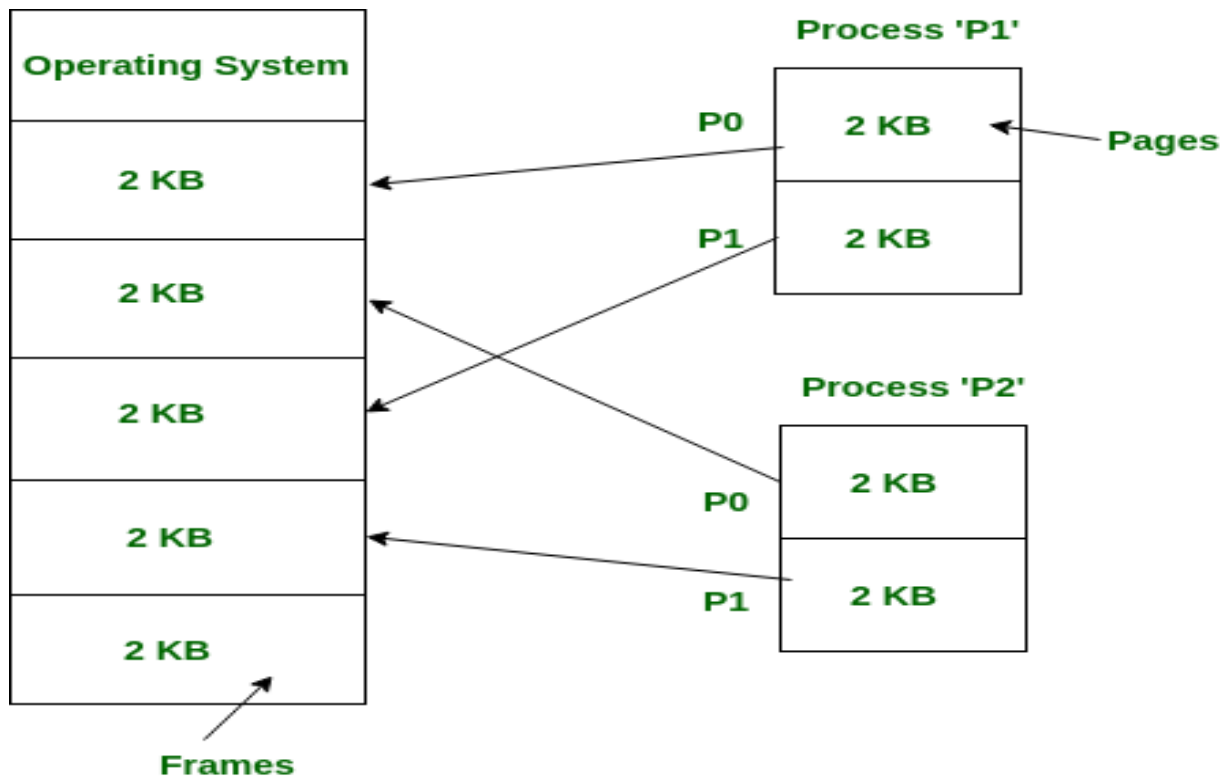
1. Paging

2. Segmentation

<b>Logical Address</b>	<b>Physical Address</b>
Logical Address Space is the set of all logical addresses generated by CPU for a program.	The set of all physical address mapped to corresponding logical addresses is called Physical Address Space.
The logical address does not exist physically in the memory.	Physical address is a location in the memory that can be accessed physically.
The logical address is generated by the CPU while the program is running.	The physical address is computed by the Memory Management Unit (MMU).

## →Paging

- Paging is a fixed size partitioning scheme.
- In paging, secondary memory and main memory are divided into equal fixed size partitions.
- The partitions of secondary memory are called as **pages**.
- The partitions of main memory are called as **frames**.
- Each process is divided into parts where size of each part is same as page size.
- The size of the last part may be less than the page size.
- The pages of process are stored in the frames of main memory depending upon their availability.



# Translating Logical Address into Physical Address-

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

## Step-01:

CPU generates a logical address consisting of two parts-

→**Page Number:-** Page Number specifies the specific page of the process from which CPU wants to read the data

→**Page Offset:-** Page Offset specifies the specific word on the page that CPU wants to read.

## Step-02:

For the page number generated by the CPU, **Page Table** provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.

## Step-03:

→**The frame number** combined with the page offset forms the required physical address. Frame number specifies the specific frame where the required page is stored.

→**Page Offset** specifies the specific word that has to be read from that page.

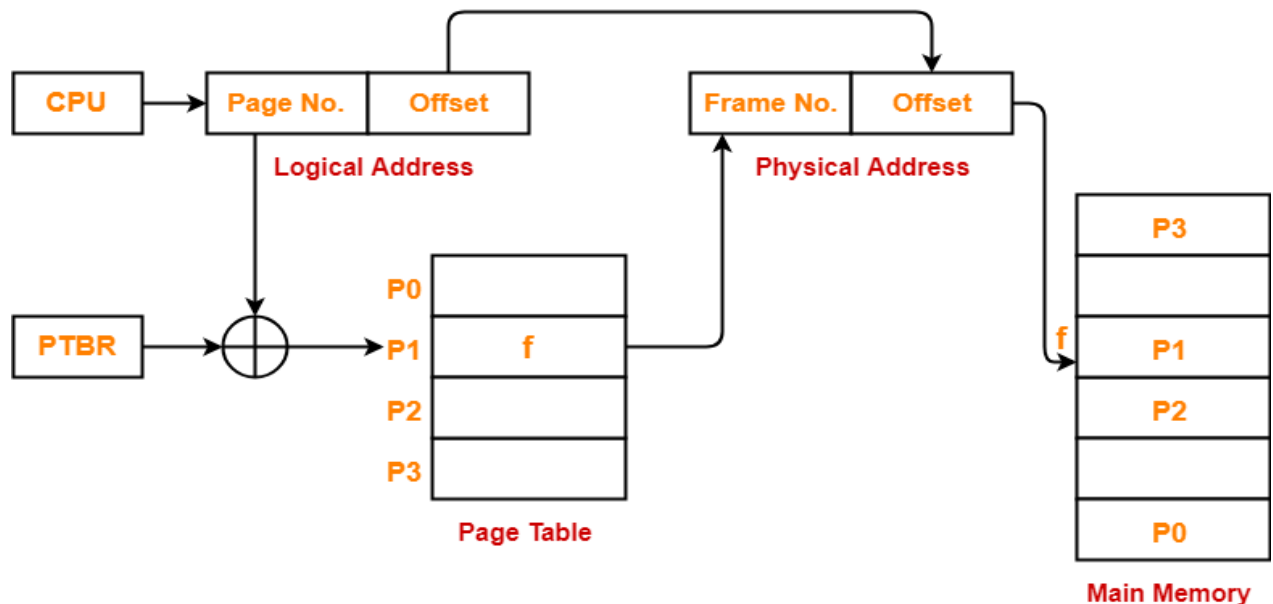
## Advantages-

- It allows to store parts of a single process in a non-contiguous fashion.
- It solves the problem of external fragmentation.

## Disadvantages-

- It suffers from internal fragmentation.
- There is an overhead of maintaining a page table for each process.
- The time taken to fetch the instruction increases since now two memory accesses are required.

Diagram-



**Translating Logical Address into Physical Address**

## Page Table-

- Page table is a data structure.
- It maps the page number referenced by the CPU to the frame number where that page is stored.

### Characteristics-

Page table is stored in the main memory.

- Number of entries in a page table = Number of pages in which the process is divided.
- Page Table Base Register (PTBR) contains the base address of page table.
- Each process has its own independent page table.

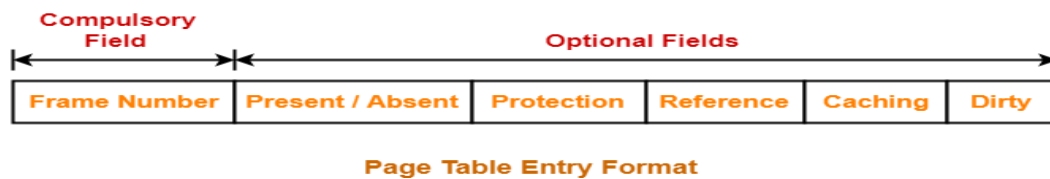
## Page Table Entry-

A page table entry contains several information about the page.

- The information contained in the page table entry varies from operating system to operating system.

- The most important information in a page table entry is frame number.

In general, each entry of a page table contains the following information-



### 1. Frame Number-

→Frame number specifies the frame where the page is stored in the main memory.

→The number of bits in frame number depends on the number of frames in the main memory.

### 2. Present / Absent Bit-

→This bit is also sometimes called as **valid / invalid bit**.

→This bit specifies whether that page is present in the main memory or not.

→If the page is not present in the main memory, then this bit is set to 0 otherwise set to 1.

### 3. Protection Bit-

→This bit is also sometimes called as “**Read / Write bit**”.

→It specifies the permission to perform read and write operation on the page.

→If only read operation is allowed to be performed and no writing is allowed, then this bit is set to 0.

→If both read and write operations are allowed to be performed, then this bit is set to 1.

### 4. Reference Bit-

→Reference bit specifies whether that page has been referenced in the last clock cycle or not.

→If the page has been referenced recently, then this bit is set to 1 otherwise set to 0.



## 5.Caching Enabled/Disabled-

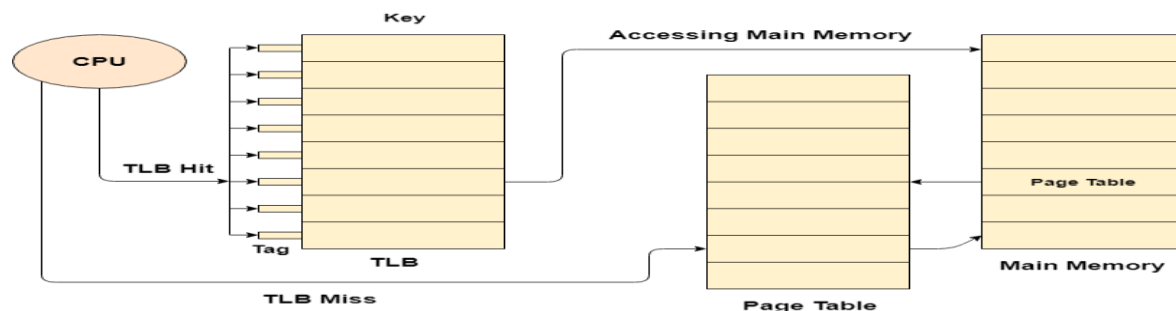
- This bit enables or disables the caching of page.
- If caching of the page is disabled, then this bit is set to 1 otherwise set to 0.

## 6. Dirty Bit-

- This bit is also sometimes called as “**Modified bit**”.
- This bit specifies whether that page has been modified or not.
- If the page has been modified, then this bit is set to 1 otherwise set to 0.

## **Translation look aside buffer (TLB)**

A Translation look aside buffer can be defined as a memory cache which can be used to reduce the time taken to access the page table again and again. TLB follows the concept of locality of reference which means that it contains only the entries of those many pages that are frequently accessed by the CPU.



In translation look aside buffers, there are tags and keys with the help of which, the mapping is done. TLB hit is a condition where the desired entry is found in translation look aside buffer. If this happens then the CPU simply access the actual location in the main memory. However, if the entry is not found in TLB (TLB miss) then CPU has to access page table in the main memory and then access the actual frame in the main memory. Therefore, in the case of TLB hit, the effective access time will be lesser as compare to the case of TLB miss.

If the probability of TLB hit is P% (TLB hit rate) then the probability of TLB miss (TLB miss rate) will be (1-P) %. Therefore, the effective access time can be defined as;

$$\text{EAT} = P(t + m) + (1 - p)(t + k.m + m)$$

Where,  $p \rightarrow$  TLB hit rate,  $t \rightarrow$  time taken to access TLB,  $m \rightarrow$  time taken to access main memory  $k = 1$ , if the single level paging has been implemented.

By the formula, we come to know that

1. Effective access time will be decreased if the TLB hit rate is increased.
2. Effective access time will be increased in the case of multilevel paging.

**Q: Consider a paging hardware with a TLB. Assume that the entire page table and all the pages are in the physical memory. It takes 10 milliseconds to search the TLB and 80 milliseconds to access the physical memory. If the TLB hit ratio is 0.6, the effective memory access time (in milliseconds) is \_\_\_\_\_.**

**Given,**

1. TLB hit ratio = 0.6
2. Therefore, TLB miss ratio = 0.4
3. Time taken to access TLB (t) = 10 ms
4. Time taken to access main memory (m) = 80 ms

$$\text{Effective Access Time (EAT)} = 0.6 (10 + 80) + 0.4 (10 + 80 + 80) = 90 \times 0.6 + 0.4 \times 170 = 122$$

## Questions 1:

Consider a system having LA=7 bits, physical address=6 bits

page size=8 words/byte

Calculate no of pages and frames.

**Ans:**

page size=8 byte= $2^3$  3 bits are required to represent page size

To represent page no= $7-3=4$  bits are required

No of pages= $2^4=16$

To represent frame no= $6-3=3$  bits are required (as frame size=page size)

No of frames= $2^3=8$

## Questions 2: (Memory is byte addressable)

Logical Address Space=4GB

Physical Address Space=64MB

Page Size=4KB No of pages=? No of frames=?

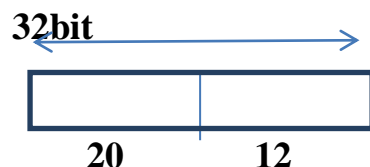
**Ans:**

**Step 1 :**

LA= $2^2 * 2^{30} = 2^{32}$  byte 32 no of bits are required

PA=64MB= $2^6 * 2^{20} = 2^{26}$  byte 26 no of bits are required

**Step2:**



Page no      Page size/offset

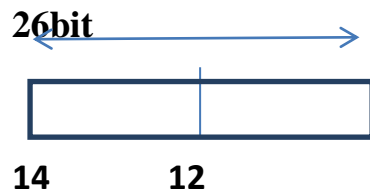
Page size=4KB

$=2^2 \times 2^{10} = 2^{12}$  bytes 12 bits are required to represent page size

No of pages  $= 2^{20}$

### Step-3

.....



Frame no      Frame size/offset      No of frames  $= 2^{14}$

## Question 3:

Calculate the size of memory if its address consists of 20 bits and the memory is 4-byte addressable.

### Solution-

Number of locations possible with 20 bits  $= 2^{20}$  locations

It is given that the size of one location  $= 4$  bytes

Thus, Size of memory

$$= 2^{20} \times 4 \text{ bytes}$$

$$= 2^{22} \text{ bytes} = 2^{20} \times 2^2$$

$$= 4 \text{ MB}$$

## Question 4:

Calculate the number of bits required in the address for memory having size of 16 GB. Assume the memory is 4-byte addressable.

### Solution-

Let 'n' number of bits are required. Then, Size of memory  $= 2^n \times 4$  bytes.

Since, the given memory has size of 16 GB, so we have-

$$2^n \times 4 \text{ bytes} = 16 \text{ GB}$$

$$2^n \times 2^2 = 2^{34}$$

$$2^n = 2^{32} \therefore n = 32 \text{ bits}$$

## Question 5:

Consider a machine with 64 MB physical memory and a 32 bit virtual address space. If the page size is 4 KB, what is the approximate size of the page table?

### Solution-

1)Size of main memory = 64 MB

2)Number of bits in virtual address space = 32 bits

3)Page size = 4 KB=frame size

### Number of Bits in Physical Address-

Size of main memory

= 64 MB

$= 2^6 * 2^{20} = 2^{26} \text{ B}$

Thus, Number of bits in physical address = 26 bits

### Number of Bits in Page Offset-

We have,

Page size

= 4 KB =  $2^{12} \text{ B}$

Thus, Number of bits in page/frame offset = 12 bits



Logical address space=32 bits

Page size/offset=12bits

Page size=32-12=20 bits

No of pages= $2^n = 2^{20}$

Number of entries in page table(input) =  $2^{20}$  entries

Page table size= Number of entries in page table x Number of bits in frame number

$= 2^{20} \text{ byte} * 14 \text{ bits}$     8bits=1byte    14bits=16bits(2byte)

$= 2^{20} * 2^1 = 2^{21} \text{ byte} = 2 \text{ MB}$



# Virtual Memory

Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.

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

## Benefits of having Virtual Memory

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

## Demand Paging

According to the concept of Virtual Memory, in order to execute some process, only a part of the process needs to be present in the main memory which means that only a few pages will only be present in the main memory at any time.

However, deciding, which pages need to be kept in the main memory and which need to be kept in the secondary memory, is going to be difficult because we cannot say in advance that a process will require a particular page at particular time. Therefore, to overcome this problem, there is a concept called Demand Paging is introduced. It suggests keeping all pages of the frames in the secondary memory until they are required. In other words, it says that do not load any page in the main memory until it is required.

**Page Fault** – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

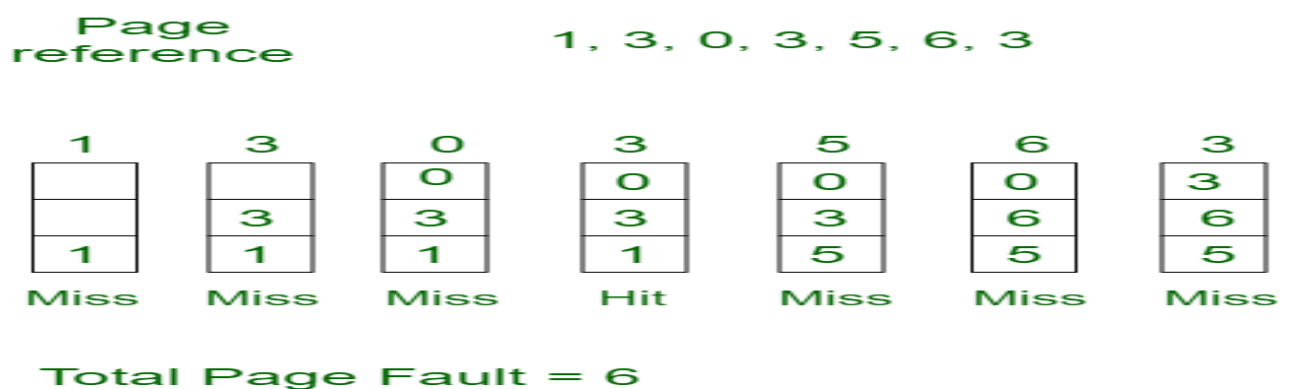
## Page Replacement Algorithms:

The page replacement algorithm decides which memory page is to be replaced. The process of replacement is sometimes called swap out or write to disk. Page replacement is done when the requested page is not found in the main memory (page fault).

### → First in First out (FIFO) –

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue; the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

**Example-1** Consider page reference string 1, 3, 0, 3, 5, 6 with 3 page frames. Find number of page faults.





## →Optimal Page replacement –

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

**Example-2:** Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.

Page reference	7,0,1,2,0,3,0,4,2,3,0,3,2,3													No. of Page frame - 4
7	0	1	2	0	3	0	4	2	3	0	3	2	3	
			2	2	2	2	2	2	2	2	2	2	2	
		1	1	1	1	1	4	4	4	4	4	4	4	
	0	0	0	0	0	0	0	0	0	0	0	0	0	
7	7	7	7	7	3	3	3	3	3	3	3	3	3	
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit	
Total Page Fault = 6														

## →Least Recently Used –

In this algorithm page will be replaced which is least recently used.

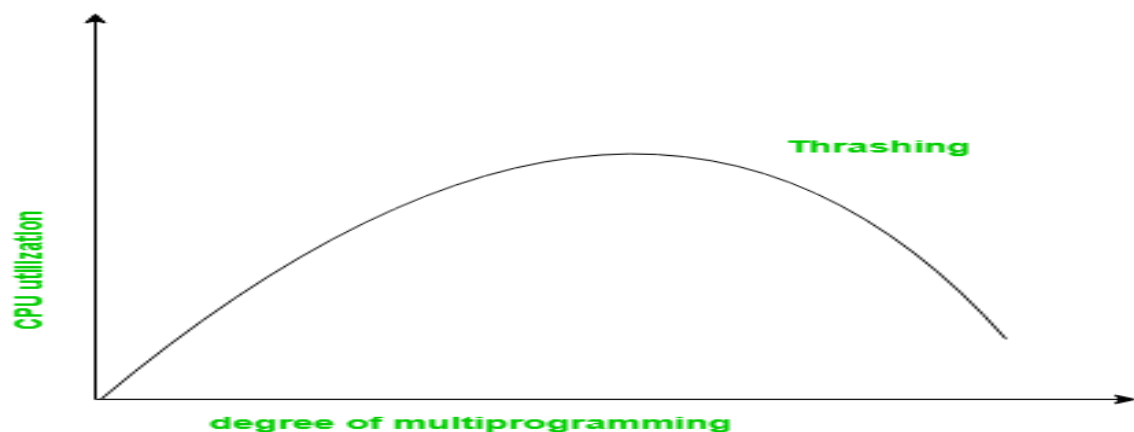
**Example-3** Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frames. Find number of page faults.

Page reference	7,0,1,2,0,3,0,4,2,3,0,3,2,3													No. of Page frame - 4
7	0	1	2	0	3	0	4	2	3	0	3	2	3	
			2	2	2	2	2	2	2	2	2	2	2	
		1	1	1	1	1	4	4	4	4	4	4	4	
	0	0	0	0	0	0	0	0	0	0	0	0	0	
7	7	7	7	7	3	3	3	3	3	3	3	3	3	
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit	
Total Page Fault = 6														

Here LRU has same number of page fault as optimal but it may differ according to question.

# Thrashing

Thrashing is a condition or a situation when the system is spending a major portion of its time in servicing the page faults, but the actual processing done is very negligible. If the number of frames which are allocated to a process is not sufficient or accurate then there can be a problem of thrashing. . As a result, no useful work would be done by the CPU and the CPU utilisation would fall drastically.



## Cause of Thrashing

1. **High degree of multiprogramming:** -The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming as a result. The new process tries to get started by taking frames from running processes, causing more page faults and a longer queue for the paging device. As a result, CPU utilization drops even further, and the CPU scheduler tries to increase the degree of multiprogramming even more
2. **Lack of frames:-** If a process has less number of frames then less pages of that process will be able to reside in memory and hence it would result in more frequent swapping. This may lead to thrashing. Hence sufficient amount of frames must be allocated.

## **Recovery**

- Do not allow the system to go into thrashing by instructing the long term scheduler not to bring the processes into memory after the threshold.
- If the system is already in thrashing then instruct the midterm scheduler to suspend some of the processes so that we can recover the system from thrashing.