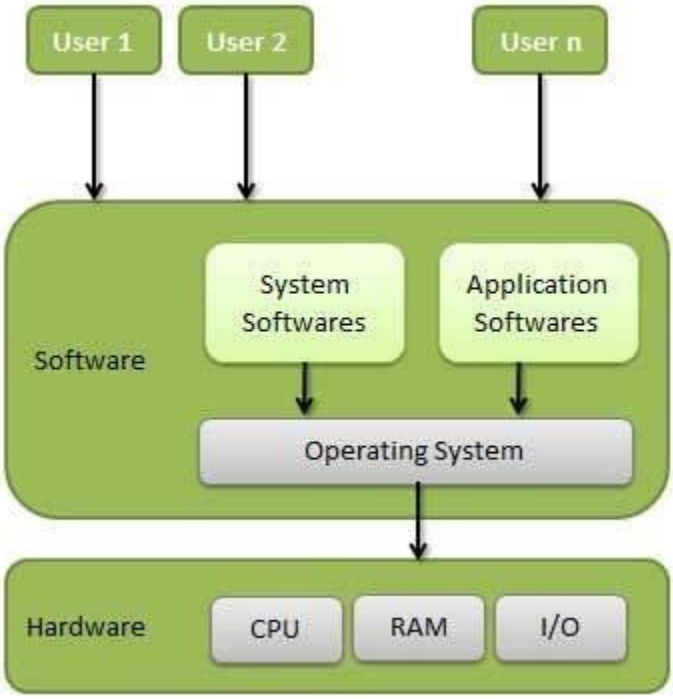
FUNCTION OF OPERATING SYSTEM

An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

Definition

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.



Following are some of important functions of an operating System.

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

FUNCTION OF OPERATING SYSTEM

Memory Management

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.

Main memory provides a fast storage that can be accessed directly by the CPU. For a program to be executed, it must in the main memory. An Operating System does the following activities for memory management −

- Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part are not in use.
- In multiprogramming, the OS decides which process will get memory when and how much.
- Allocates the memory when a process requests it to do so.
- De-allocates the memory when a process no longer needs it or has been terminated.

Processor Management

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

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

Device Management

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

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

File Management

FUNCTION OF OPERATING SYSTEM

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

An Operating System does the following activities for file management −

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

Other Important Activities

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

- **Security** − By means of password and similar other techniques, it prevents unauthorized access to programs and data.
- **Control over system performance** − Recording delays between request for a service and response from the system.
- **Job accounting** − Keeping track of time and resources used by various jobs and users.
- **Error detecting aids** − Production of dumps, traces, error messages, and other debugging and error detecting aids.
- **Coordination between other softwares and users** − Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

TYPES OF OPERATING SYSTEM

Operating systems are there from the very first computer generation and they keep evolving with time. In this chapter, we will discuss some of the important types of operating systems which are most commonly used.

Batch operating system

The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.

The problems with Batch Systems are as follows −

- Lack of interaction between the user and the job.
- CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
- Difficult to provide the desired priority.

Time-sharing operating systems

Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.

Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation. That is, if **n** users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.

The operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems are as follows −

- Provides the advantage of quick response.
- Avoids duplication of software.

- Reduces CPU idle time.

Disadvantages of Time-sharing operating systems are as follows −

- Problem of reliability.
- Question of security and integrity of user programs and data.
- Problem of data communication.

Distributed operating System

Distributed systems use multiple central processors to serve multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as **loosely coupled systems** or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.

The advantages of distributed systems are as follows −

- With resource sharing facility, a user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- If one site fails in a distributed system, the remaining sites can potentially continue operating.
- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing.

Network operating System

A Network Operating System runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.

Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

The advantages of network operating systems are as follows −

- Centralized servers are highly stable.
- Security is server managed.

- Upgrades to new technologies and hardware can be easily integrated into the system.
- Remote access to servers is possible from different locations and types of systems.

The disadvantages of network operating systems are as follows −

- High cost of buying and running a server.
- Dependency on a central location for most operations.
- Regular maintenance and updates are required.

Real Time operating System

A real-time system is defined as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. The time taken by the system to respond to an input and display of required updated information is termed as the **response time**. So in this method, the response time is very less as compared to online processing.

Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail. For example, Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

There are two types of real-time operating systems.

Hard real-time systems

Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems, secondary storage is limited or missing and the data is stored in ROM. In these systems, virtual memory is almost never found.

Soft real-time systems

Soft real-time systems are less restrictive. A critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems. For example, multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers, etc.

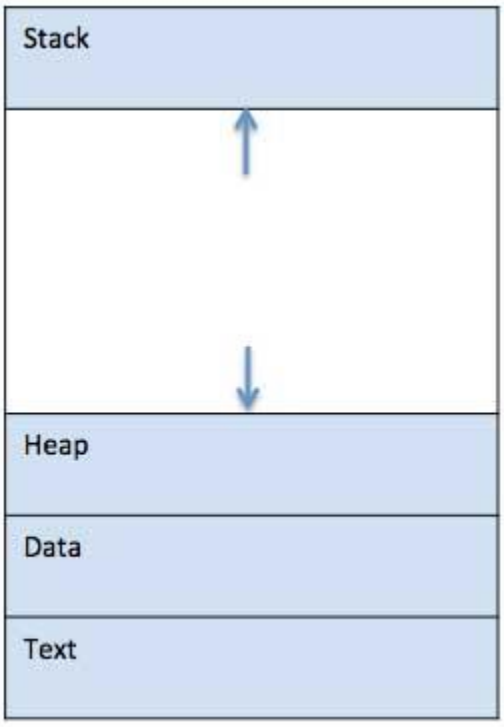
Process

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

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

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

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



S.N.	Component & Description
1	**Stack** The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2	**Heap**

	This is dynamically allocated memory to a process during its run time.
3	**Text** This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4	**Data** This section contains the global and static variables.

Program

A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. For example, here is a simple program written in C programming language −

```
#include <stdio.h>

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

A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.

A part of a computer program that performs a well-defined task is known as an **algorithm**. A collection of computer programs, libraries and related data are referred to as a **software**.

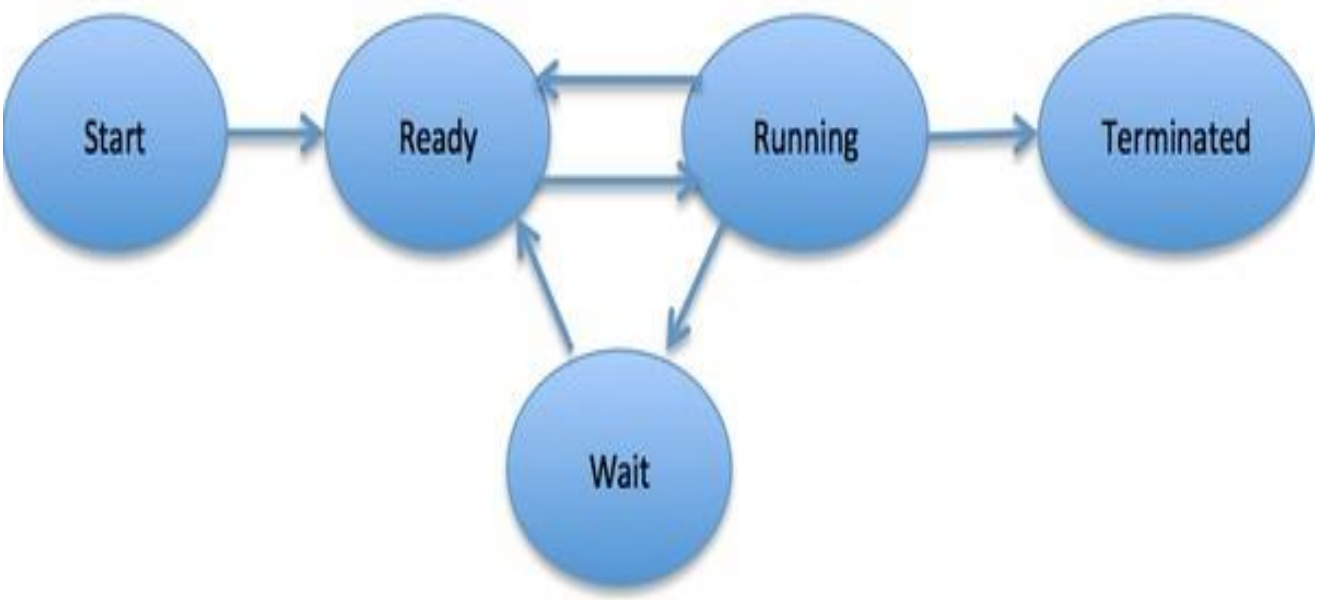
Process Life Cycle

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

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

S.N.	State & Description
1	**Start**

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



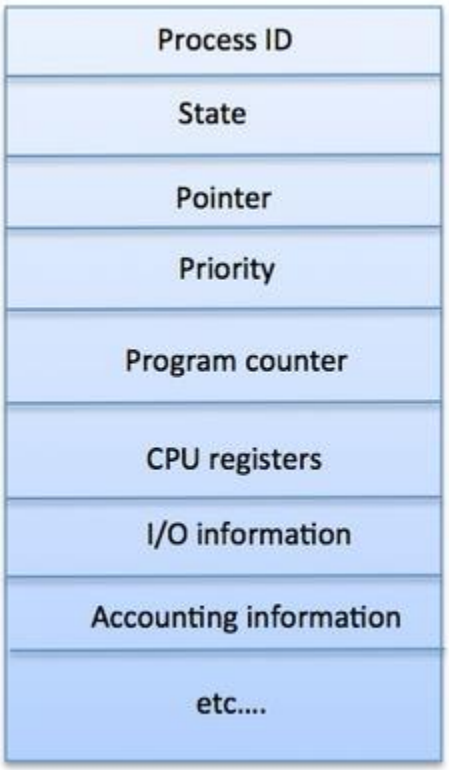
Process Control Block (PCB)

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

S.N.	Information & Description
1	**Process State** The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	**Process privileges** This is required to allow/disallow access to system resources.
3	**Process ID** Unique identification for each of the process in the operating system.
4	**Pointer** A pointer to parent process.
5	**Program Counter** Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	**CPU registers** Various CPU registers where process need to be stored for execution for running state.
7	**CPU Scheduling Information**

	Process priority and other scheduling information which is required to schedule the process.
8	**Memory management information** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	**Accounting information** This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	**IO status information** This includes a list of I/O devices allocated to the process.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB −



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

PROCESS SCHEDULING:

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

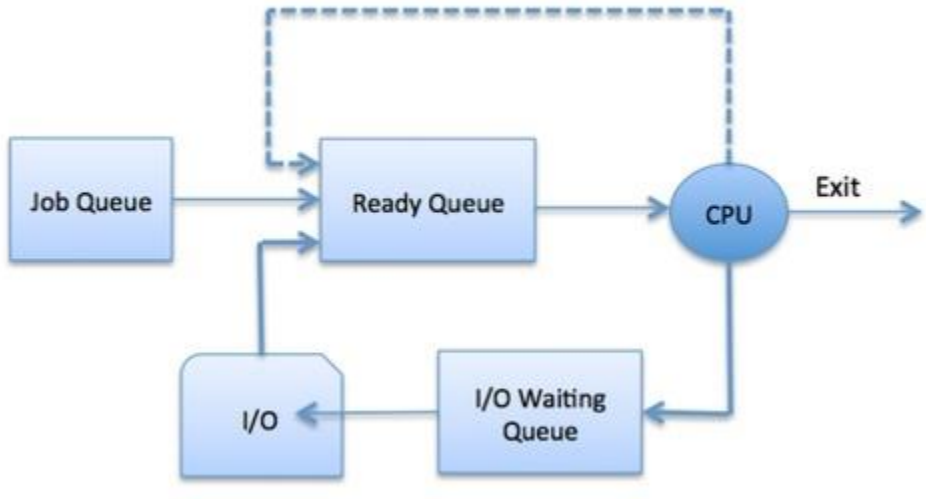
Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Process Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

- **Job queue** − This queue keeps all the processes in the system.
- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.



The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

Two-State Process Model

Two-state process model refers to running and non-running states which are described below −

S.N.	State & Description
1	**Running** When a new process is created, it enters into the system as in the running state.
2	**Not Running** Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute.

Schedulers

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

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

Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

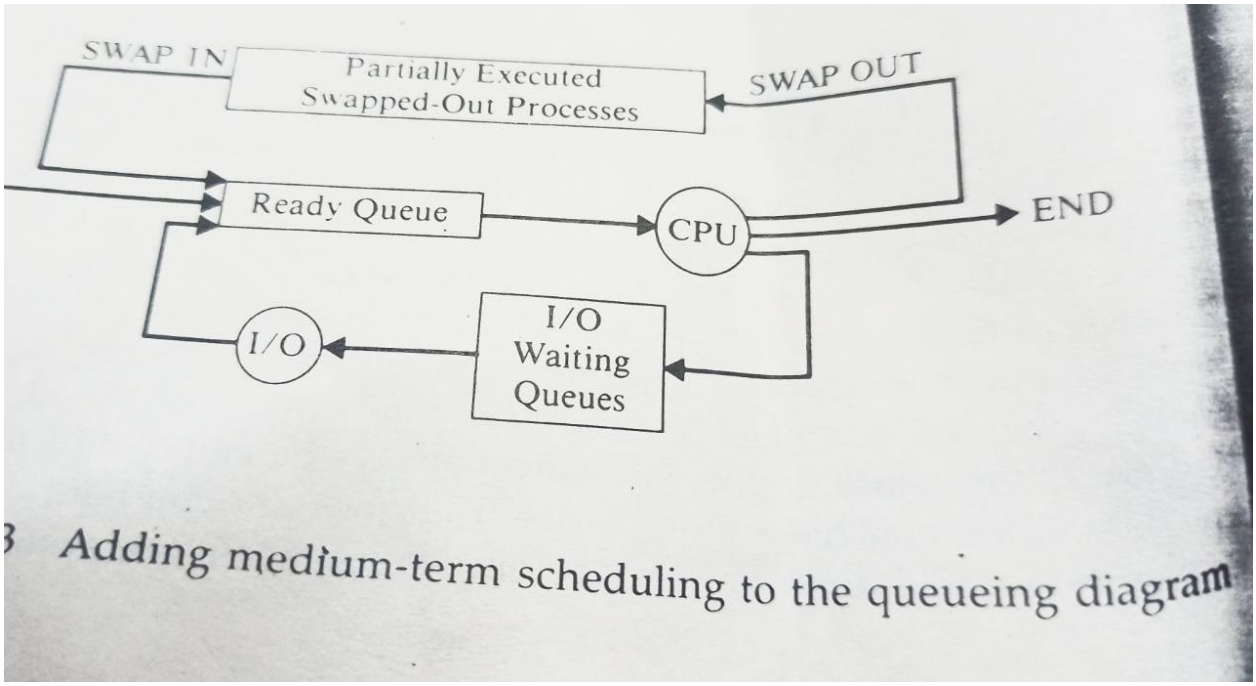
It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.



Adding medium-term scheduling to the queueing diagram

Comparison among Scheduler

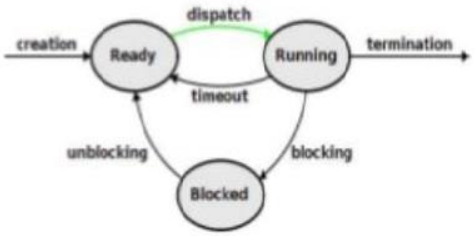
S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the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

Dispatcher

The dispatcher is the module that gives control of the CPU to the process selected by the scheduler. This function involves:

- Switching context.
- Switching to user mode.
- Jumping to the proper location in the newly loaded program.

The dispatcher needs to be as fast as possible, as it is run on every context switch. Dispatch latency is the amount of time required for the scheduler to stop one process and start another.



Dispatcher

When the processes are in the ready state, then the CPU applies some process scheduling algorithm and choose one process from a list of processes that will be executed at a particular instant of time. This is done by a scheduler i.e. selecting one process from a number of processes is done by a scheduler.

Now, the selected process has to be transferred from the current state to the desired or scheduled state. So, it is the duty of the dispatcher to dispatch or transfer a process from one state to another. A dispatcher is responsible for context switching and switching to user mode.

For example, if we have three processes P1, P2, and P3 in the ready state. The arrival time of all these processes is T0, T1, and T2 respectively. If we are using the First Come First Serve approach, then the scheduler will first select the process P1 and the dispatcher will transfer the process P1 from the ready state to the running state. After completion of the execution of the process P1, the scheduler will then select the process P2 and the dispatcher will transfer the process P2 from ready to running state and so on.

Difference between Dispatcher and Scheduler

Till now, we are familiar with the concept of dispatcher and scheduler. Now in this section of the blog, we will see the difference between a dispatcher and a scheduler.

- The scheduler selects a process from a list of processes by applying some process scheduling algorithm. On the other hand, the dispatcher transfers the process selected by the short-term scheduler from one state to another.

- The scheduler works independently, while the dispatcher has to be dependent on the scheduler i.e. the dispatcher transfers only those processes that are selected by the scheduler.
- For selecting a process, the scheduler uses some process scheduling algorithm like FCFS, Round-Robin, SJF, etc. But the dispatcher doesn't use any kind of scheduling algorithms.
- The only duty of a scheduler is to select a process from a list of processes. But apart from transferring a process from one state to another, the dispatcher can also be used for switching to user mode. Also, the dispatcher can be used to jump to a proper location when the process is restarted.

- In the Operating System, there are cases when you have to bring back the process that is in the running state to some other state like ready state or wait/block state.
- If the running process wants to perform some I/O operation, then you have to remove the process from the running state and then put the process in the I/O queue.
- Sometimes, the process might be using a round-robin scheduling algorithm where after every fixed time quantum, the process has to come back to the ready state from the running state.
- So, these process switchings are done with the help of Context Switching. In this blog, we will learn about the concept of Context Switching in the Operating System and we will also learn about the advantages and disadvantages of Context Switching. So, let's get started.

What is Context Switching?

A context switching is a process that involves switching of the CPU from one process or task to another. In this phenomenon, the execution of the process that is present in the running state is suspended by the kernel and another process that is present in the ready state is executed by the CPU.

It is one of the essential features of the multitasking operating system. The processes are switched so fastly that it gives an illusion to the user that all the processes are being executed at the same time.

But the context switching process involved a number of steps that need to be followed. You can't directly switch a process from the running state to the ready state. You have to save the context of that process. If you are not saving the context of any process P then after some time, when the process P comes in the CPU for execution again, then the process will start executing from starting. But in reality, it should continue from that point where it left the CPU in its previous

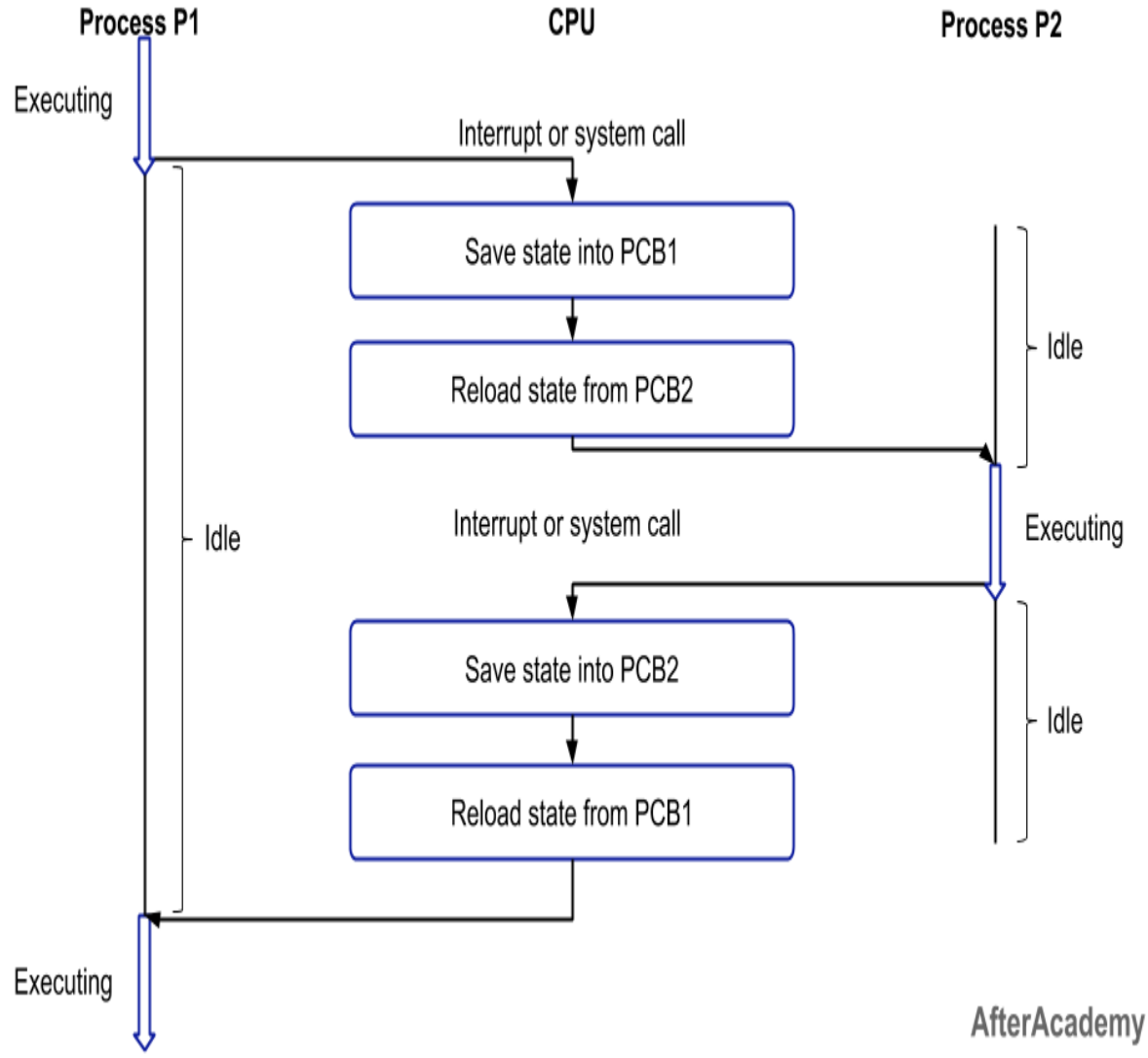
execution. So, the context of the process should be saved before putting any other process in the running state.

A context is the contents of a CPU's registers and program counter at any point in time. Context switching can happen due to the following reasons:

- When a process of high priority comes in the ready state. In this case, the execution of the running process should be stopped and the higher priority process should be given the CPU for execution.
- When an interruption occurs then the process in the running state should be stopped and the CPU should handle the interrupt before doing something else.
- When a transition between the user mode and kernel mode is required then you have to perform the context switching.

Steps involved in Context Switching

The process of context switching involves a number of steps. The following diagram depicts the process of context switching between the two processes P1 and P2.



In the above figure, you can see that initially, the process P1 is in the running state and the process P2 is in the ready state. Now, when some interruption occurs then you have to switch the process P1 from running to the ready state after saving the context and the process P2 from ready to running state. The following steps will be performed:

1. Firstly, the context of the process P1 i.e. the process present in the running state will be saved in the Process Control Block of process P1 i.e. PCB1.
2. Now, you have to move the PCB1 to the relevant queue i.e. ready queue, I/O queue, waiting queue, etc.
3. From the ready state, select the new process that is to be executed i.e. the process P2.
4. Now, update the Process Control Block of process P2 i.e. PCB2 by setting the process state to running. If the process P2 was earlier executed by the CPU, then you can get the position of last executed instruction so that you can resume the execution of P2.
5. Similarly, if you want to execute the process P1 again, then you have to follow the same steps as mentioned above(from step 1 to 4).

For context switching to happen, two processes are at least required in general, and in the case of the round-robin algorithm, you can perform context switching with the help of one process only.

The time involved in the context switching of one process by other is called the Context Switching Time.

Advantage of Context Switching

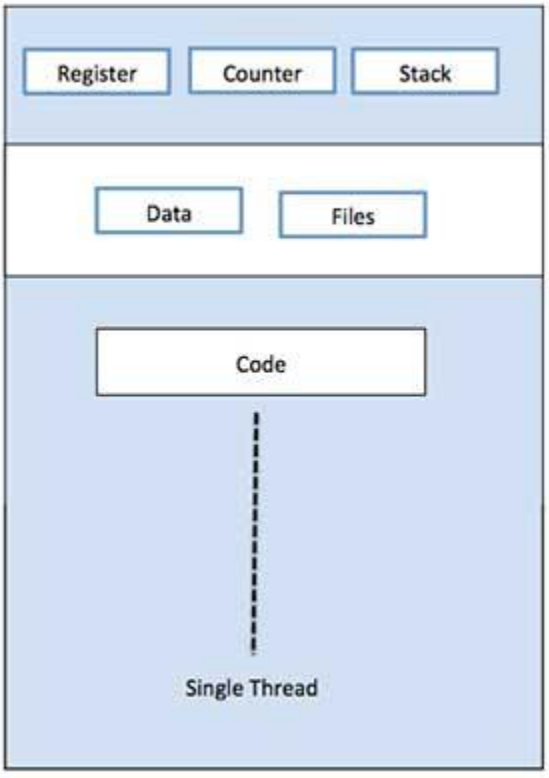
- Context switching is used to achieve multitasking i.e. multiprogramming with time-sharing.
- Multitasking gives an illusion to the users that more than one process are being executed at the same time.
- But in reality, only one task is being executed at a particular instant of time by a processor. Here, the context switching is so fast that the user feels that the CPU is executing more than one task at the same time.

The disadvantage of Context Switching

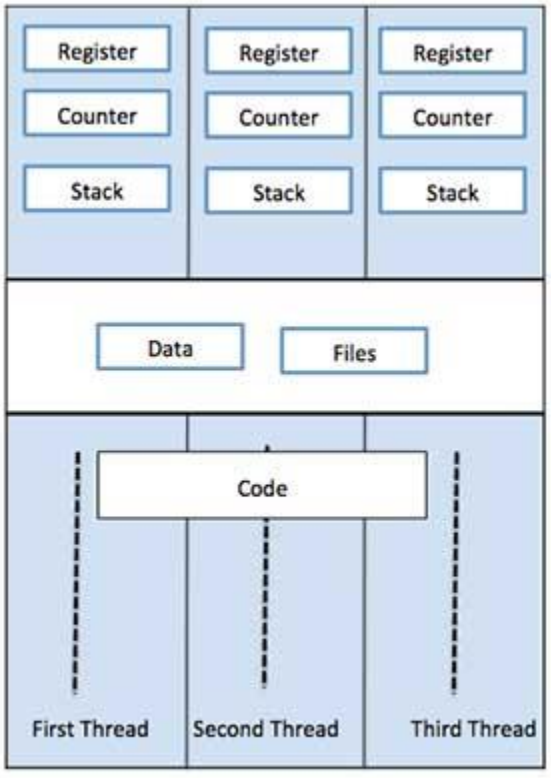
- The disadvantage of context switching is that it requires some time for context switching i.e. the context switching time.
- Time is required to save the context of one process that is in the running state and then getting the context of another process that is about to come in the running state.
- During that time, there is no useful work done by the CPU from the user perspective. So, context switching is pure overhead in this condition.

What is Thread?

- A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.
- A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.
- A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.
- Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. The following figure shows the working of a single-threaded and a multithreaded process.



Single Process P with single thread



Single Process P with three threads

Difference between Process and Thread

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

		use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Advantages of Thread

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

Types of Thread

Threads are implemented in following two ways −

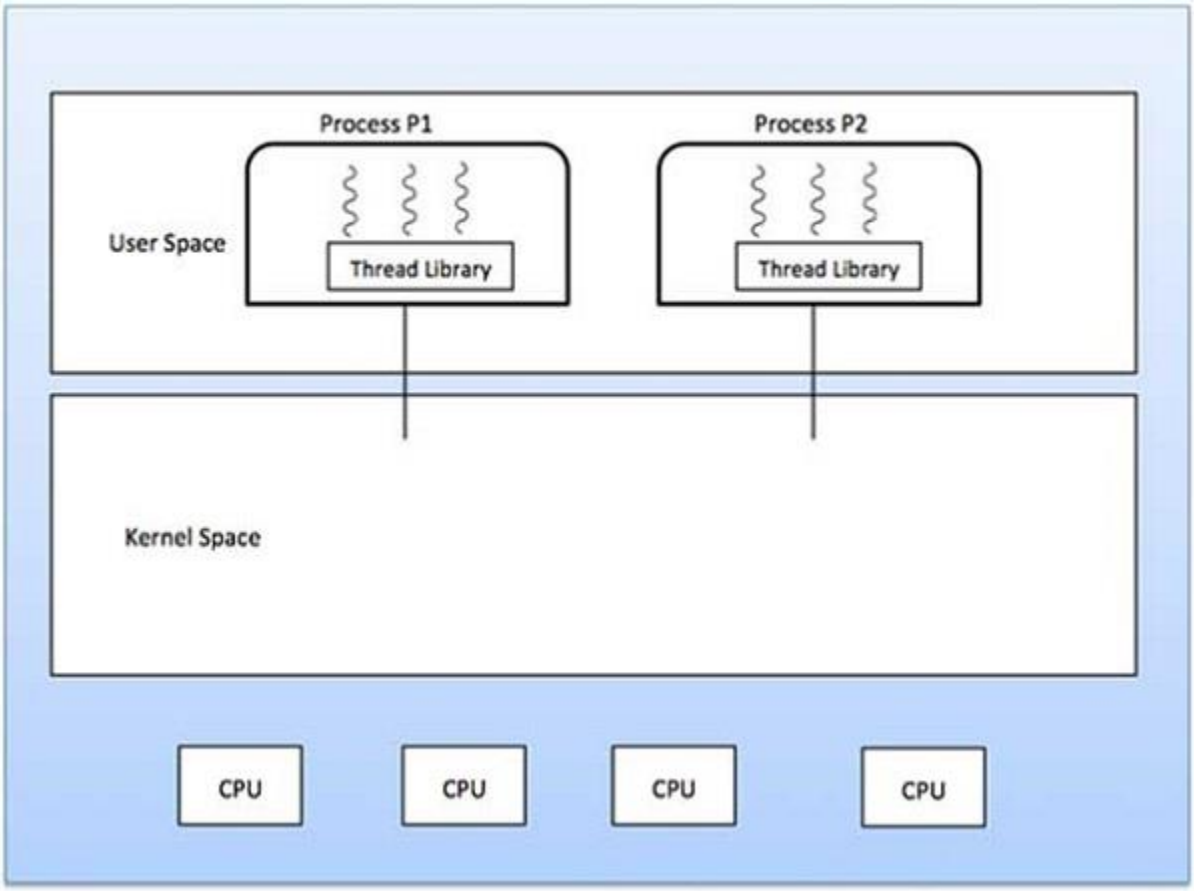
- **User Level Threads** − User managed threads.
- **Kernel Level Threads** − Operating System managed threads acting on kernel, an operating system core.

- ## Difference between User-Level & Kernel-Level Thread

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

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. The application starts with a single thread.



Advantages

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

Disadvantages

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

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. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.
- The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis.
- The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

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

Disadvantages

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

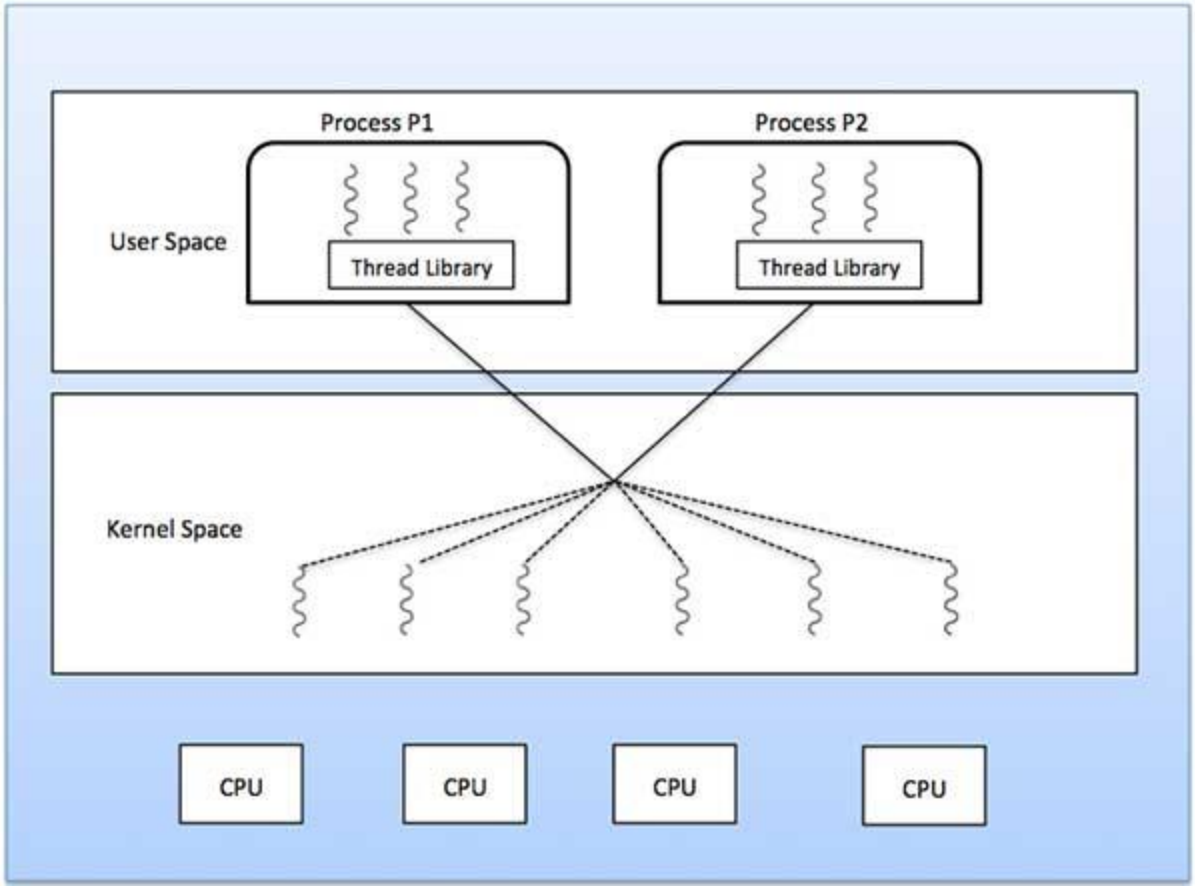
Multithreading Models

Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- Many to many relationship.
- Many to one relationship.
- One to one relationship.

Many to Many Model

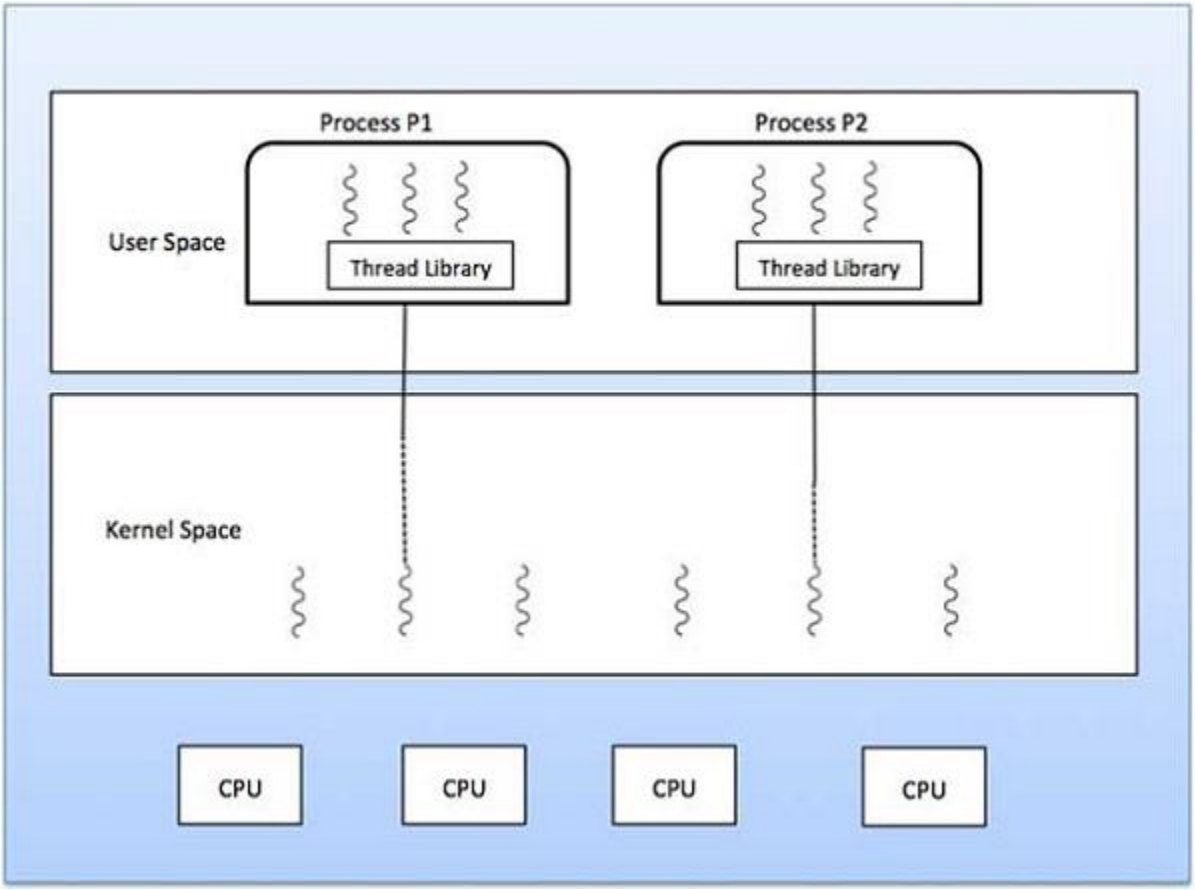
- The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.
- The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads.
- In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine.
- This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.



Many to One Model

Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

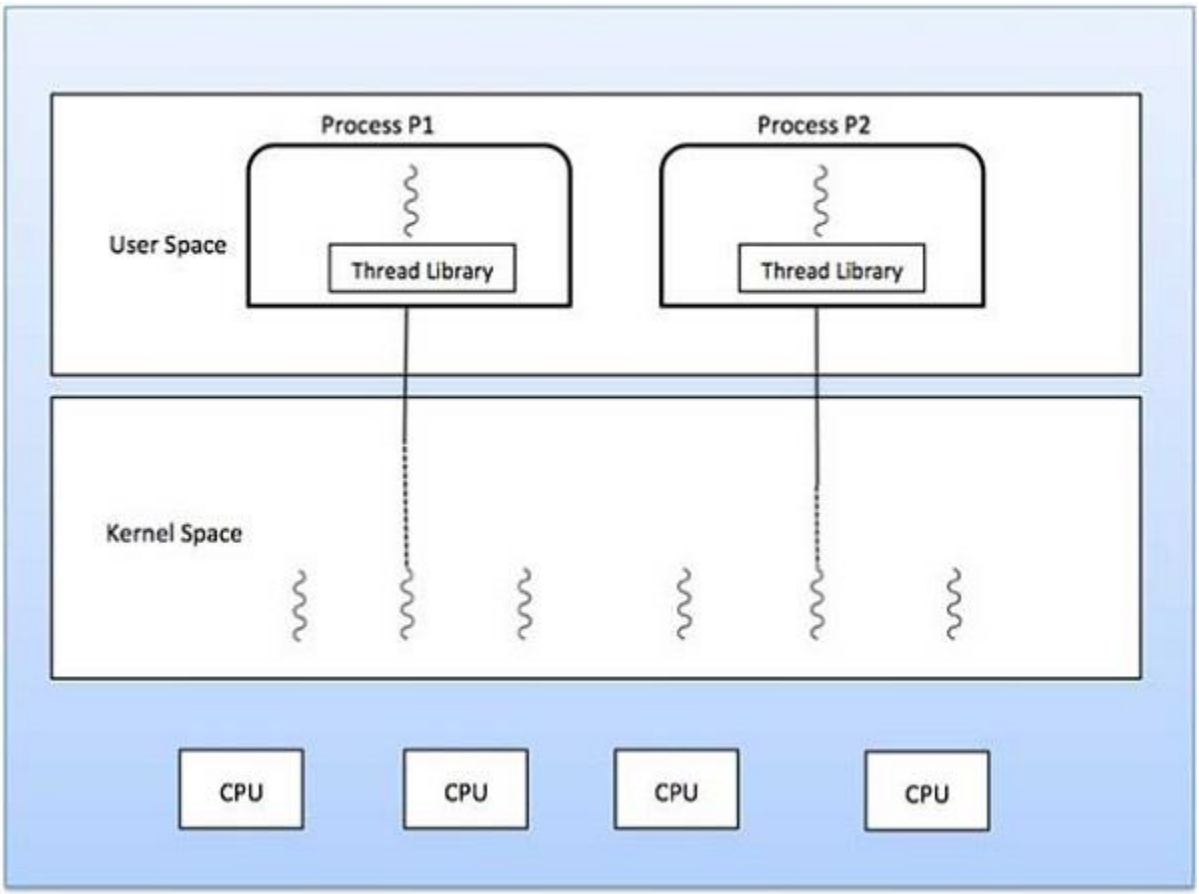
If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.



One to One Model

There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

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



Process Management

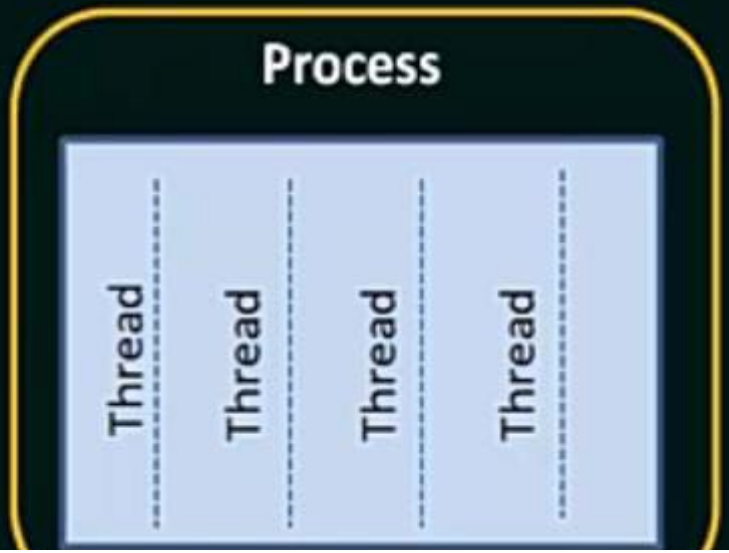
(Processes and Threads)

Process:

A process can be thought of as a program in execution.

Thread:

A thread is the unit of execution within a process. A process can have anywhere from just one thread to many threads.



Threads

A thread is a basic unit of CPU utilization.

It comprises

- A thread ID
- A program counter
- A register set and
- A stack

It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.

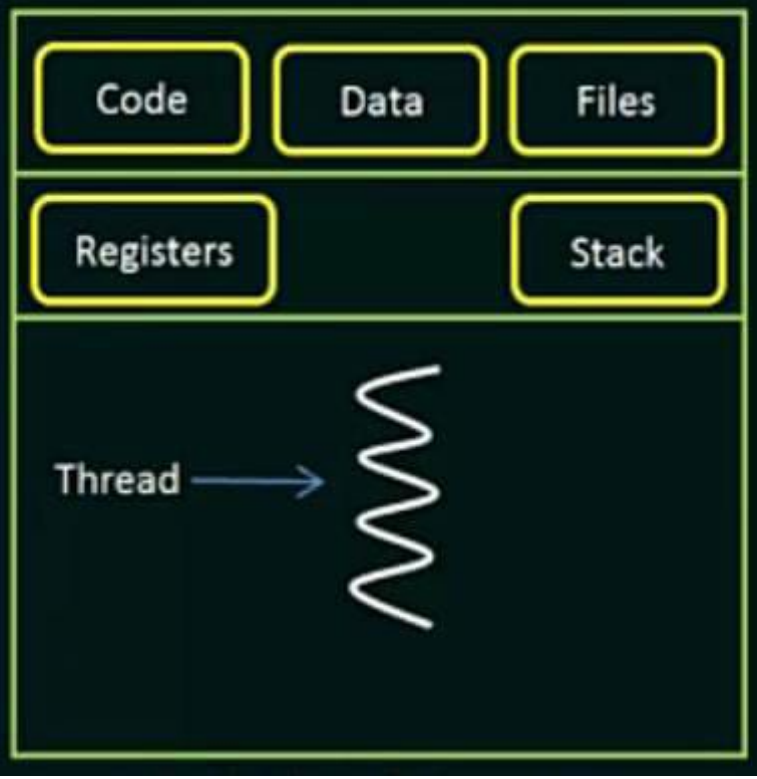
A traditional / heavyweight process has a single thread of control.

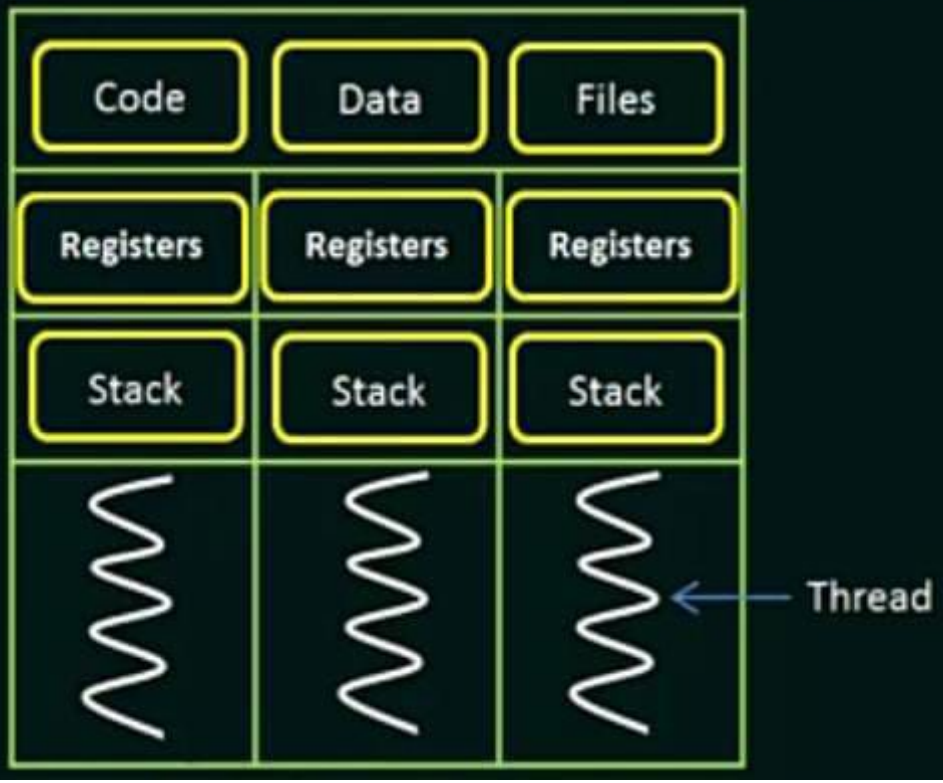
If a process has multiple threads of control, it can perform more than one task at a time.

It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.

A traditional / heavyweight process has a single thread of control.

If a process has multiple threads of control, it can perform more than one task at a time.





The benefits of multithreaded programming can be broken down into four major categories:

Responsiveness	Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.
Resource sharing	By default, threads share the memory and the resources of the process to which they belong. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.
Economy	Allocating memory and resources for process creation is costly. Because threads share resources of the process to which they belong, it is more economical to create and context-switch threads.
Utilization of multiprocessor architectures	The benefits of multithreading can be greatly increased in a multiprocessor architecture, where threads may be running in parallel on different processors. A single-threaded process can only run on one CPU, no matter how many are available. Multithreading on a multi-CPU machine increases

Threading Issues (Part-2)

Thread Cancellation

Thread cancellation is the task of terminating a thread before it has completed.

If multiple threads are concurrently searching through a database and one thread returns the result, the remaining threads might be canceled.

When a user presses a button on a web browser that stops a web page from loading any further, all threads loading the page are canceled.

A thread that is to be canceled  is often referred to as the target thread.

Cancellation of a target thread may occur in two different scenarios:

1. Asynchronous cancellation: One thread immediately terminates the target thread.

2. Deferred cancellation: The target thread periodically checks whether it should terminate, allowing it an opportunity to terminate itself in an orderly fashion.

Where the difficulty with cancellation lies:

Where the difficulty with cancellation lies:

In situations where:

- Resources have been allocated to a canceled thread
- A thread is canceled while in the midst of updating data it is sharing with other threads.

Often, the OS will reclaim system resources from a canceled thread
but will not reclaim all resources.

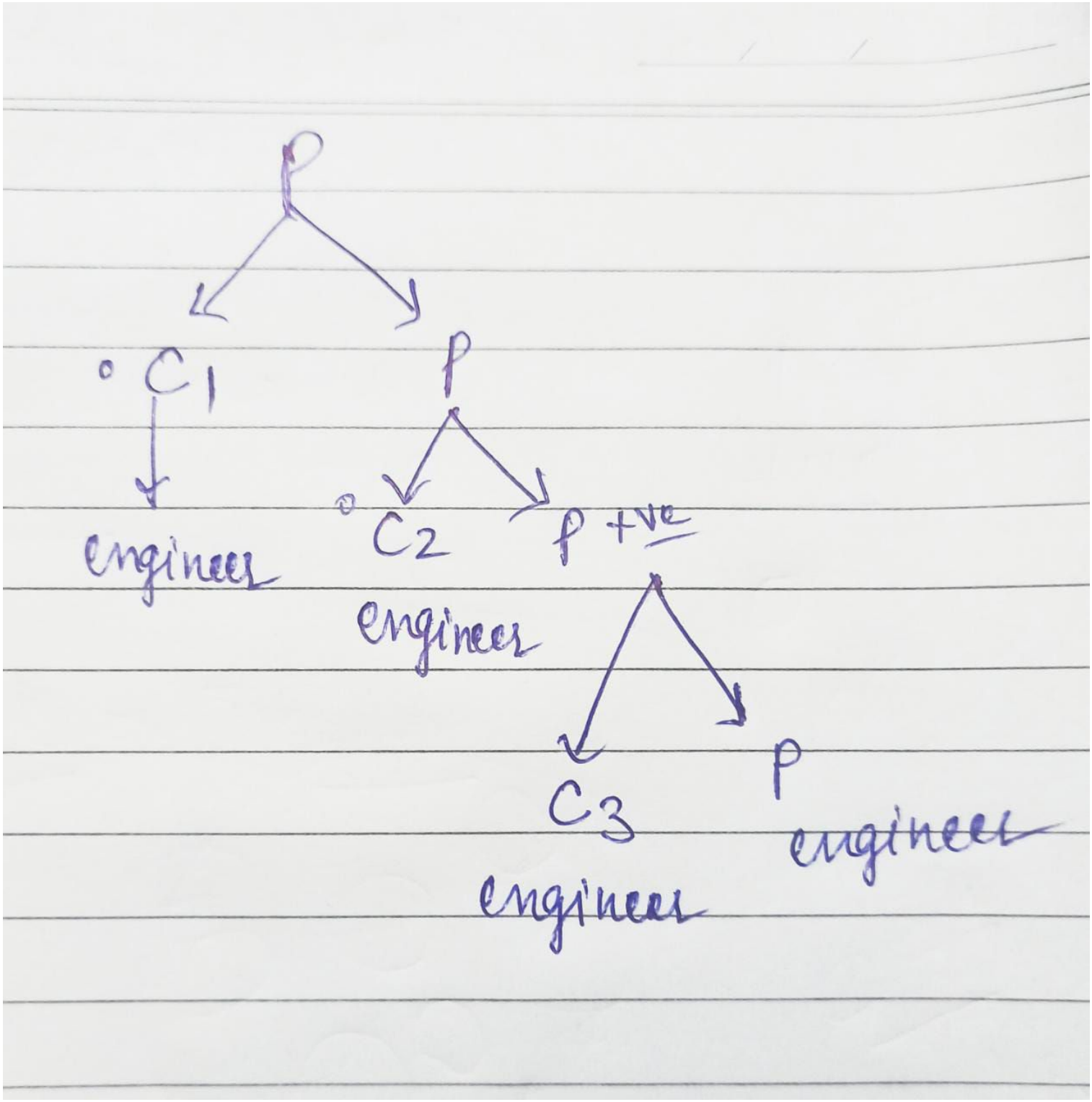
Therefore, canceling a thread asynchronously
may not free a necessary system-wide resource.

Often, the OS will reclaim system resources from a canceled thread
but will not reclaim all resources.
Therefore, canceling a thread asynchronously
may not free a necessary system-wide resource.

With deferred cancellation:

One thread indicates that a target thread is to be canceled.
But cancellation occurs only after the target thread has checked a flag to determine if it should be canceled or not.
This allows a thread to check whether it should be canceled at a point when it can be canceled safely.

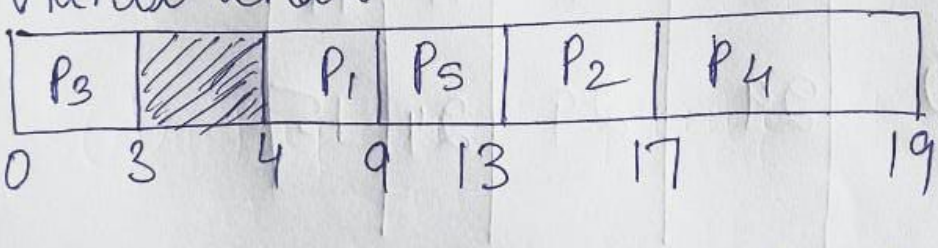
```
# include <stdio.h>
# include <conio.h>
int main()
{
if (fork() && fork())
    fork();
printf ("engineer");
return 0;
}
```


P
C1
P
C2
P +ve
C3
P
engineer
engineer
engineer
engineer

FCFS

Process	Arrival time	Burst time	Completion time	Turnaround time	Waiting time
$P_1 \rightarrow$	4	5	9	5	0
P_2	6	4	17	11	7
P_3	0	3	3	3	0
P_4	6	2	19	13	11
P_5	5	4	13	8	4

Gantt chart



average turn around time = 40/5 = 8 units

Avg. waiting = 22/5 = 4.4 units

Round Robin

Five process are given, content Switch time is 1 unit. calculate average WT, TAT, RT, No. of content switches and CPU utilization.

TQ = 3 unit

	Burst time	AT	CT	TAT	WT	RT
P_1	~~852~~	0	32	32	24	0
P_2	2	0	6	6	4	4
P_3	7 ~~$\frac{1}{4}$~~ 1	0	34	34	27	7
P_4	3	0	14	14	11	11
P_5	~~5~~ 2	0	29	29	24	15

Ready Queue

P1	P2	P3	P4	P5	P1	P3	P5	P1	P3

Gantt chart.

P1	P2	P3	P4	P5	P1	P3	P5	P1	P3

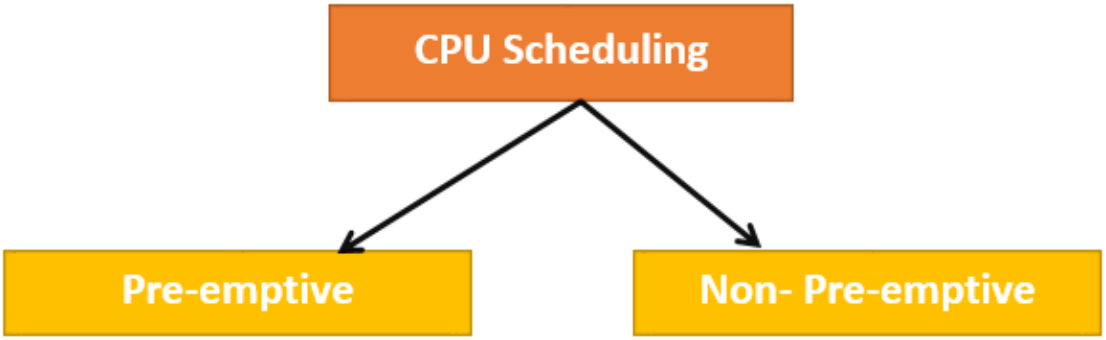
3 4 6 7 10 11 14 15 18 19 22 23 26 27 29 30 32 33 34

No. of content switches → 9

CPU utilization → Expected / Actual $\frac{25}{34} \times 100 = 73.5$

What is CPU Scheduling?

CPU Scheduling is a process of determining which process will own CPU for execution while another process is on hold.
The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution.
The selection process will be carried out by the CPU scheduler. It selects one of the processes in memory that are ready for execution.



Preemptive Scheduling

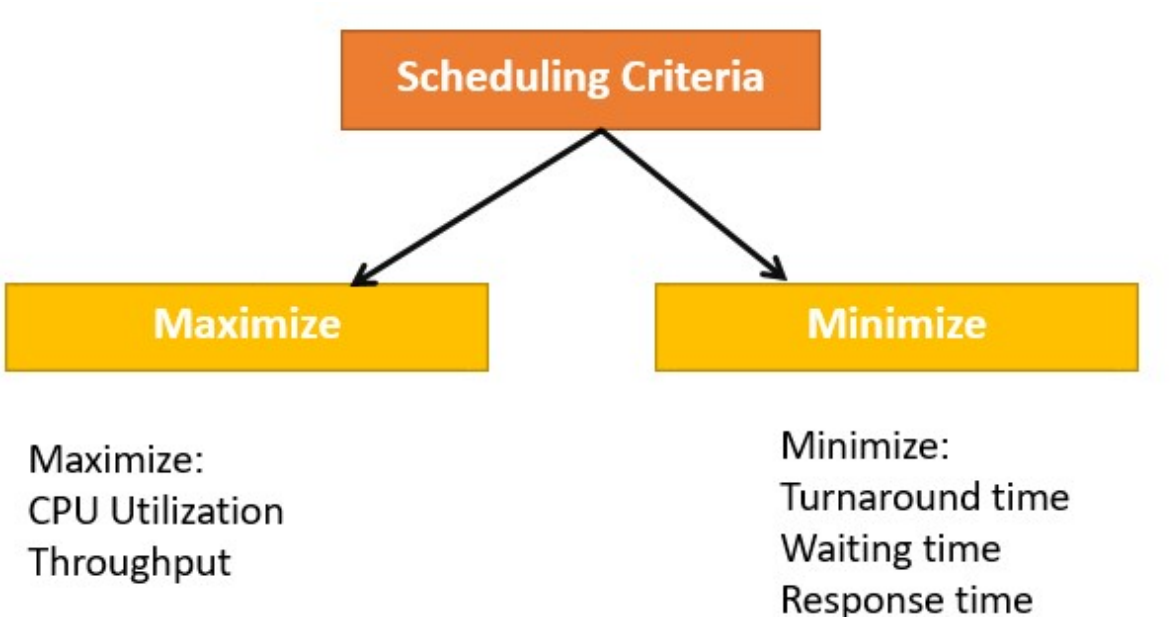
In Preemptive Scheduling, the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running.

The lower priority task holds for some time and resumes when the higher priority task finishes its execution.

Non-preemptive Scheduling

is a CPU scheduling technique the process takes the resource (CPU time) and holds it till the process gets terminated or is pushed to the waiting state.

No process is interrupted until it is completed, and after that processor switches to another process.



Maximize:

CPU utilization: CPU utilization is the main task in which the operating system needs to make sure that CPU remains as busy as possible. It can range from 0 to 100 percent.

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

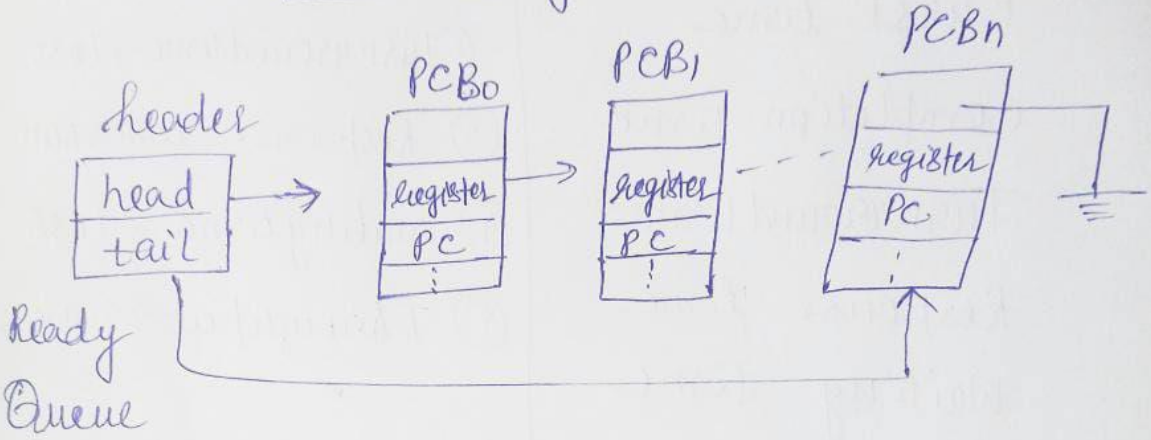
Minimize:

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

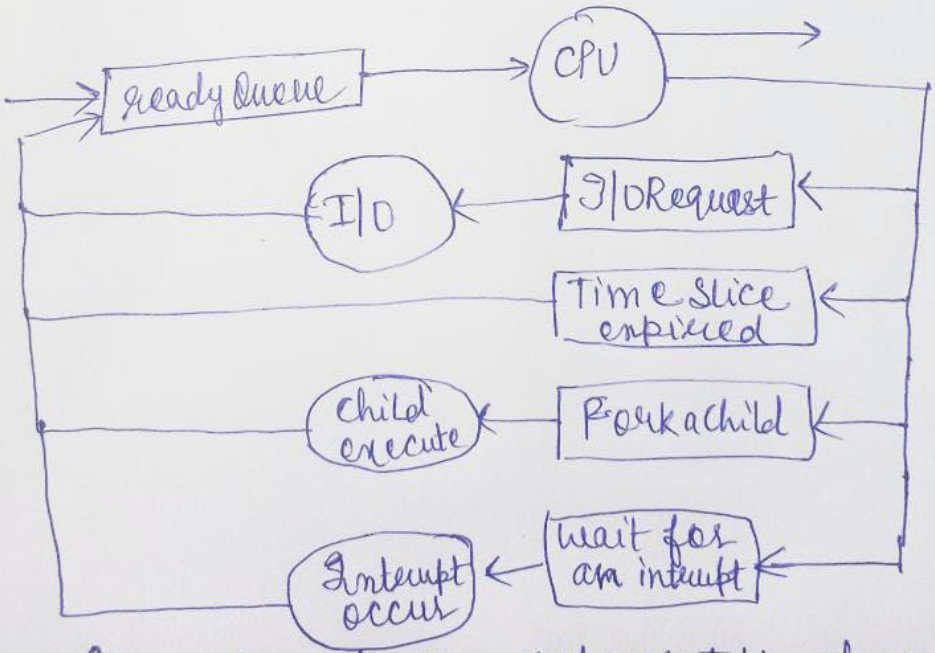
Response time: It is an amount to time in which the request was submitted until the first response is produced.

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

Scheduling Queues



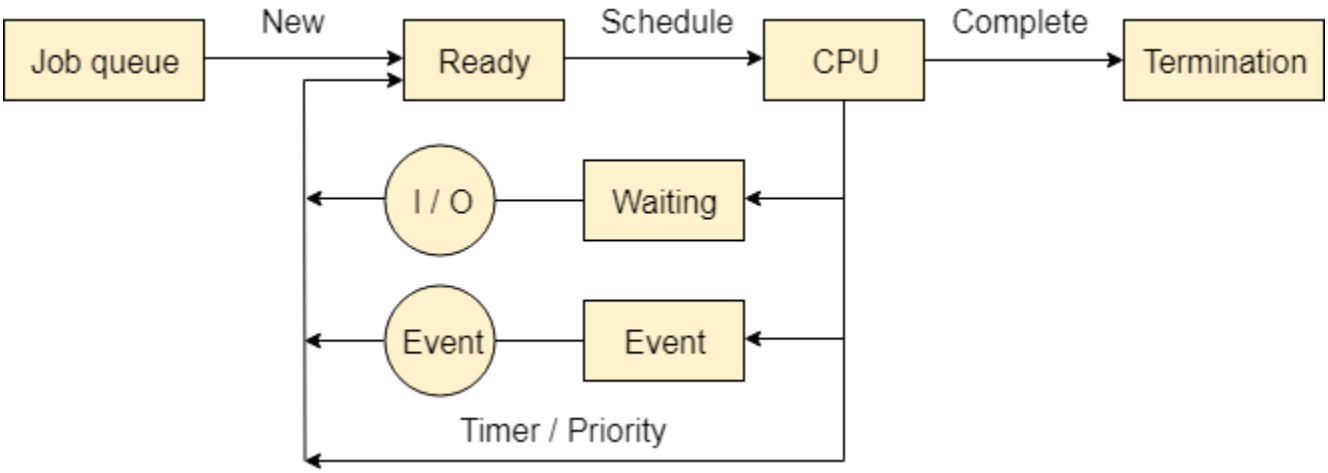
1. Job queue
2. Ready queue
3. device queue



Queueing diagram representation of CPU scheduling

Scheduling Queues

The Operating system manages various types of queues for each of the process states. The PCB related to the process is also stored in the queue of the same state. If the Process is moved from one state to another state then its PCB is also unlinked from the corresponding queue and added to the other state queue in which the transition is made.



There are the following queues maintained by the Operating system.

1. Job Queue

In starting, all the processes get stored in the job queue. It is maintained in the secondary memory. The long term scheduler (Job scheduler) picks some of the jobs and put them in the primary memory.

2. Ready Queue

Ready queue is maintained in primary memory. The short term scheduler picks the job from the ready queue and dispatch to the CPU for the execution.

3. Waiting Queue

When the process needs some IO operation in order to complete its execution, OS changes the state of the process from running to waiting. The context (PCB) associated with the process gets stored on the waiting queue which will be used by the Processor when the process finishes the IO.

(Shortest-Job-First Scheduling)

- This algorithm associates with each process the length of the process's next CPU burst.
- When the CPU is available, it is assigned to the process that has the smallest next CPU burst.
- If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.

The SJF algorithm can be either preemptive or nonpreemptive

A more appropriate term for this scheduling method would be the

Shortest-Next-CPU-Burst Algorithm

because scheduling depends on the length of the next CPU burst of a process, rather than its total length.

Example of SJF Scheduling (Non-Premptive)

Consider the following set of processes,with the length of the CPU burst given in milliseconds:

Process ID	Burst Time
P1	6
P2	8
P3	7
P4	3

Waiting Time for P1 = 3 ms

Waiting Time for P2 = 16 ms

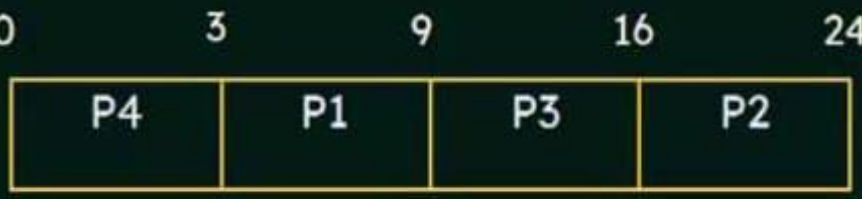
Waiting Time for P3 = 9 ms

Waiting Time for P4 = 0 ms

Average Waiting Time

= (3 + 16 + 9 + 0)/4 = 7 ms

Gantt Chart:



By comparison, if we were using the FCFS scheduling scheme, the average waiting time would be 10.25 milliseconds

Example of SJF Scheduling (Premptive)

Consider the following four processes, with the length of the CPU burst given in milliseconds and the processes arrive at the ready queue at the times shown:

Process ID	Arrival Time	Burst Time
P1	0	~~8~~ 7
P2	1	4
P3	2	9
P4	3	5

Waiting Time for P1 = (10 – 1 - 0) = 9 ms

Waiting Time for P2 = (1 – 0 - 1) = 0 ms

Waiting Time for P3 = (17 – 0 - 2) = 15 ms

Waiting Time for P4 = (5 – 0 - 3) = 2 ms

Gantt Chart:

Average Waiting Time

= (9 + 0 + 15 + 2)/4 = 6.5 ms

Waiting Time =Total waiting Time – No. of milliseconds Process executed – Arrival Time

Example of SJF Scheduling (Premptive)

Consider the following four processes, with the length of the CPU burst given in milliseconds and the processes arrive at the ready queue at the times shown:

Process ID	Arrival Time	Burst Time
P1	0	~~8~~ 7
P2	1	4
P3	2	9
P4	3	5

Waiting Time for P1 = (10 – 1 - 0) = 9 ms

Waiting Time for P2 = (1 – 0 - 1) = 0 ms

Waiting Time for P3 = (17 – 0 - 2) = 15 ms

Waiting Time for P4 = (5 – 0 - 3) = 2 ms

Gantt Chart:

Average Waiting Time

= (9 + 0 + 15 + 2)/4 = 6.5 ms

Waiting Time =Total waiting Time – No. of milliseconds Process executed – Arrival Time

Problems with SJF Scheduling:

- The real difficulty with the SJF algorithm is knowing the length of the next CPU request.
- Although the SJF algorithm is optimal, it cannot be implemented at the level of short-term CPU scheduling.
- There is no way to know the length of the next CPU burst.

One approach is:

- To try to approximate SJF scheduling.
- We may not know the length of the next CPU burst, but we may be able to predict its value.
- We expect that the next CPU burst will be similar in length to the previous ones.
- Thus, by computing an approximation of the length of the next CPU burst, we can pick the process with the shortest predicted CPU burst.

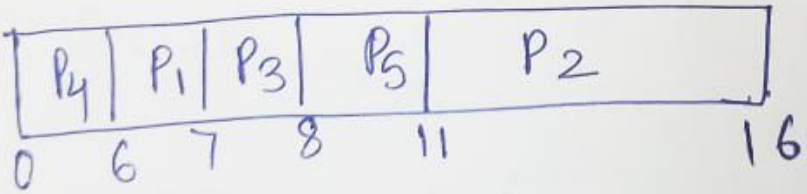
1.

SJF
→ Pre-emptive (SRTF)
→ Non-preemptive

	Arrival time	Burst time	C.T	TAT	W.T	R.T
P_1	2	1	7	5	4	4
P_2	1	5	16	15	10	10
P_3	4	1	8	4	3	3
P_4	0	6	6	6	0	0
P_5	2	3	11	9	6	6

Ready
P_1 P_2 P_3 P_5

Gantt chart



DISADvantage:

→ can not be implemented
→ starvation problem with process having large burst time.

Pre-emptive

Whenever new process arrives, there may be pre-emption of the running process.

	AT	BT	CT	TAT	WT	Response time
P_1	2	1	3	1	$P_1 \rightarrow 0$	0
P_2	1	5	16	15	$P_2 \rightarrow 10$	10
P_3	4	1	5	1	$P_3 \rightarrow 0$	0
P_4	0	6 $\cancel{5}_4$	11	11	$P_4 \rightarrow 5$	0
P_5	2	3	7	5	$P_5 \rightarrow 2$	1

P_4, P_2, P_1, P_5, P_3

Gantt-chart

P_4	P_4	P_1	P_5	P_3	P_5	P_4	P_2

0 1 2 3 4 5 7 11 16

Advantage → Min. average WT & min average Tat.

→ Better response time than FCFS

→ Max. throughput

→ Provide a standard for other Algo in case of avg. WT.

Scheduling Algorithms
(Priority Scheduling)

- A priority is associated with each process, and the CPU is allocated to the process with the highest priority.

- Equal-priority processes are scheduled in FCFS order.

- An SJF algorithm is simply a priority algorithm where the priority is the inverse of the (predicted) next CPU burst.
  The larger the CPU burst, the lower the priority, and vice versa.

Priority scheduling can be either preemptive or nonpreemptive.

A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.

A nonpreemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.

Consider the following set of processes, assumed to have arrived at time 0, in the order P1, P2, P3, P4, P5, with the length of the CPU burst given in milliseconds:

Process ID	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Using Priority Scheduling, we would schedule these processes according to the following Gantt Chart:

Waiting Time for P1 = 6 ms

Waiting Time for P2 = 0 ms

Waiting Time for P3 = 16 ms

Waiting Time for P4 = 18 ms

Waiting Time for P5 = 1 ms

Average Waiting Time

= (6 + 0 + 16 + 18 + 1) / 5

= 41 / 5 ms

= 8.2 ms

Problem with Priority Scheduling

A major problem with priority scheduling algorithms is indefinite blocking, or starvation.

A process that is ready to run but waiting for the CPU can be considered blocked.

A priority scheduling algorithm can leave some low priority processes waiting indefinitely.

In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.

Solution to the Problem

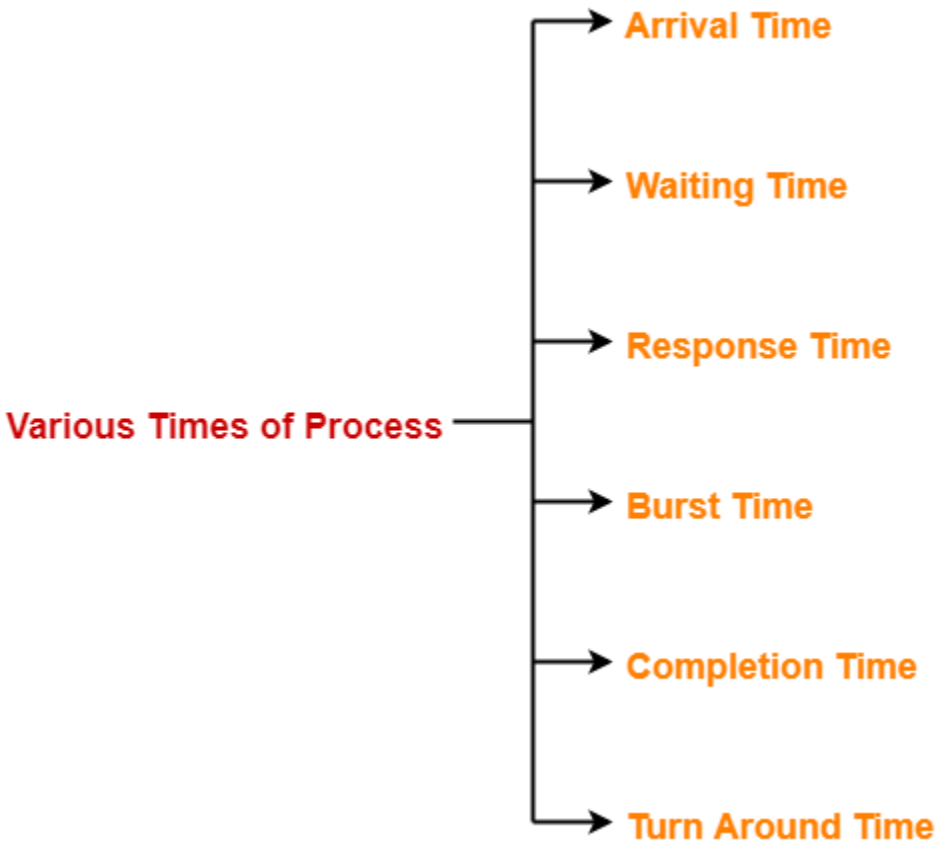
A solution to the problem of indefinite blockage of low-priority processes is aging.

Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.

For example,

If priorities range from 127 (low) to 0 (high), we could increase the priority of a waiting process by 1 every 15 minutes.

Eventually, even a process with an initial priority of 127 would have the highest priority in the system and would be executed.



1. Arrival time
2. Waiting time
3. Response time
4. Burst time
5. Completion time
6. Turn Around Time

1. Arrival Time-

- Arrival time is the point of time at which a process enters the ready queue.

2. Waiting Time-

- Waiting time is the amount of time spent by a process waiting in the ready queue for getting the CPU.

Waiting time = Turn Around time – Burst time

3. Response Time-

- Response time is the amount of time after which a process gets the CPU for the first time after entering the ready queue.

Response Time = Time at which process first gets the CPU – Arrival time

4. Burst Time-

- Burst time is the amount of time required by a process for executing on CPU.
- It is also called as **execution time** or **running time**.
- Burst time of a process can not be known in advance before executing the process.
- It can be known only after the process has executed.

5. Completion Time-

- Completion time is the point of time at which a process completes its execution on the CPU and takes exit from the system.
- It is also called as **exit time**.

6. Turn Around Time-

- Turn Around time is the total amount of time spent by a process in the system.
- When present in the system, a process is either waiting in the ready queue for getting the CPU or it is executing on the CPU.

Turn Around time = Burst time + Waiting time

OR

Turn Around time = Completion time – Arrival time

Contiguous allocation (1)

Fixed Partition:

→ No. of Partition are fixed

→ Size of each Partition may or may not same

→ Contiguous allocation so spanning is not allowed.

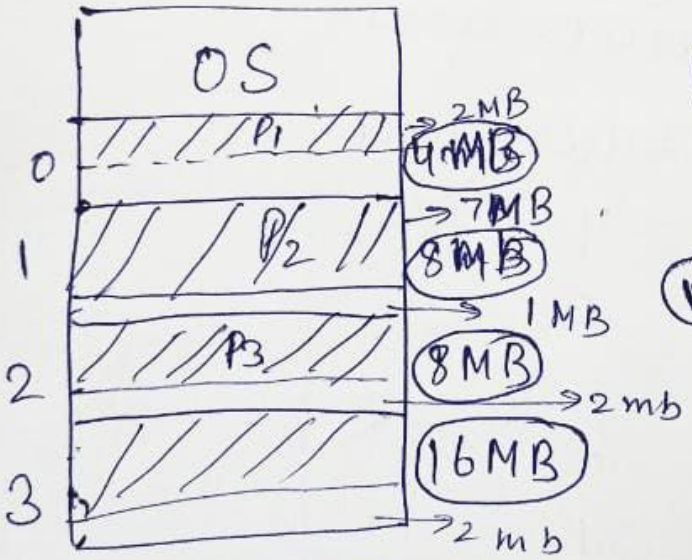
disadvantage:

(A) Internal Fragmentation.

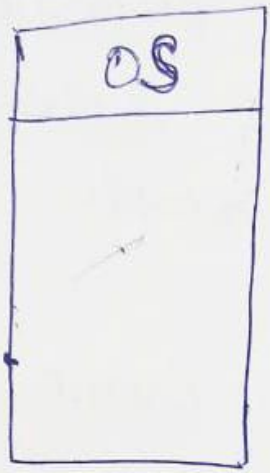
(A) limit in process size.

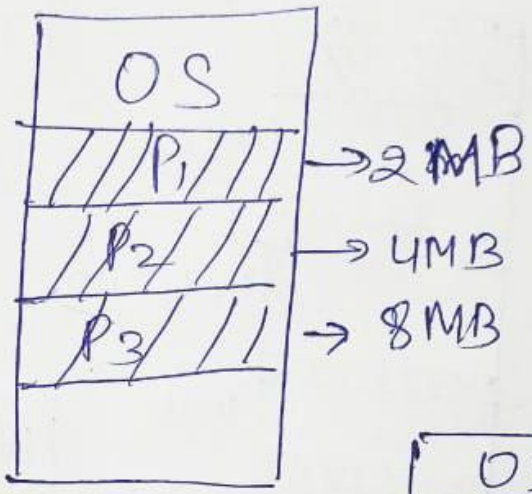
(A) limitation on degree of multiprogramming

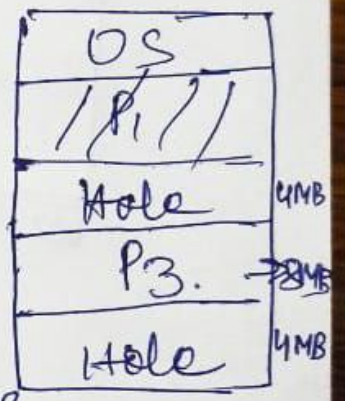
(A) External fragmentation



Dynamic Partitioning (2)
or
Variable Partitioning







Advantage

→ No Internal fragmentation

→ No. limitation on no. of process.

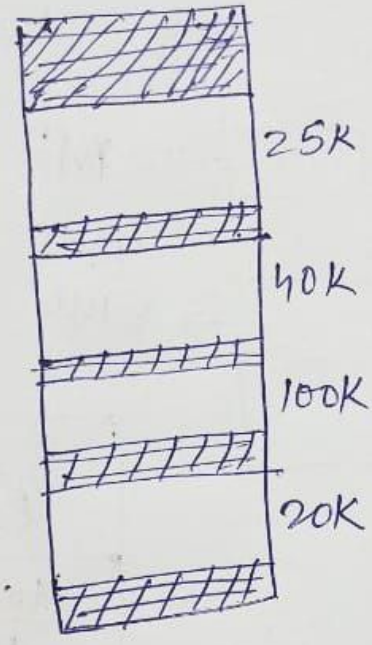
→ No. limitation on process size

Disadvantage

① External fragmentation

② Allocation/deallocation is complex

Memory Allocation →



First Fit → Allocate the first hole that is big enough.

Next fit → Same as first fit but start search always from last allocated hole

BEST FIT Allocate the smallest hole that is big enough.

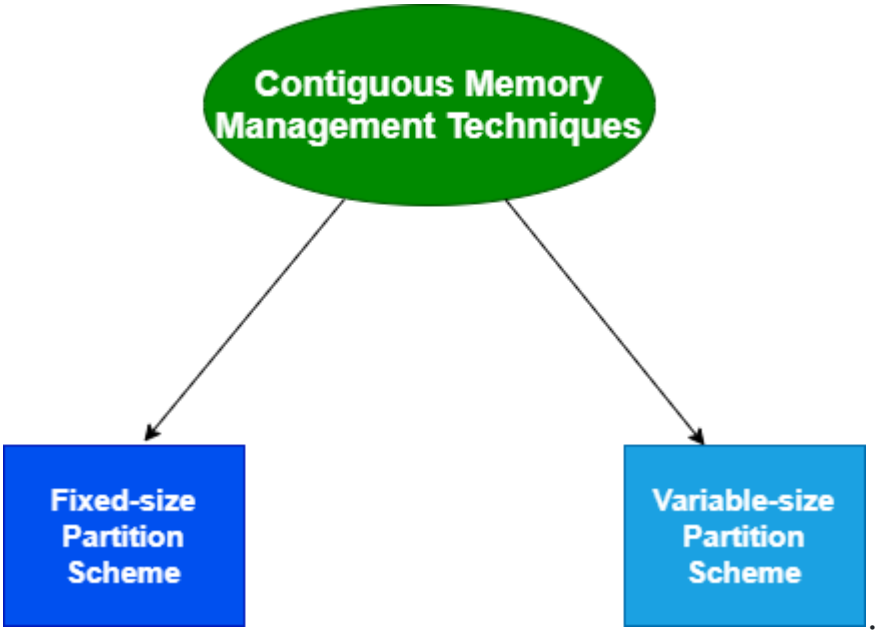
WORST FIT → Allocate the largest hole.

In the Contiguous Memory Allocation, each process is contained in a single contiguous section of memory. In this memory allocation, all the available memory space remains together in one place which implies that the freely available memory partitions are not spread over here and there across the whole memory space.

In **Contiguous memory allocation** which is a memory management technique, whenever there is a request by the user process for the memory then a single section of the contiguous memory block is given to that process according to its requirement.

Contiguous Memory allocation is achieved just by dividing the memory into the **fixed-sized partition.**

The memory can be divided either in the **fixed-sized partition or in the variable-sized partition** in order to allocate contiguous space to user processes.



Fixed-size Partition Scheme

This technique is also known as **Static partitioning**. In this scheme, the system divides the memory into fixed-size partitions. The partitions may or may not be the same size. The size of each partition

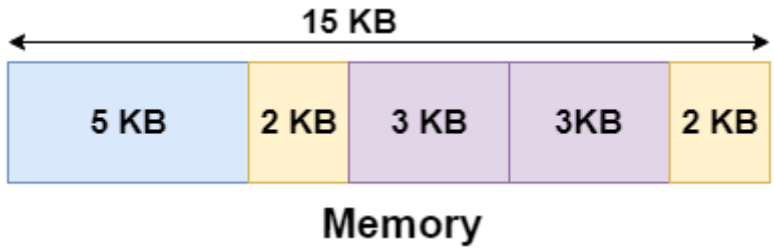
is fixed as indicated by the name of the technique and it cannot be changed.

In this partition scheme, each partition may contain exactly one process. There is a problem that this technique will limit the degree of multiprogramming because the number of partitions will basically decide the number of processes.

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

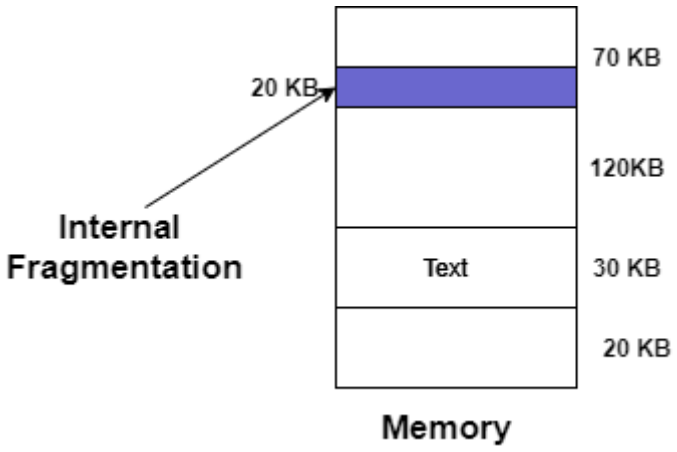
Example

Let's take an example of fixed size partitioning scheme, we will divide a memory size of 15 KB into fixed-size partitions:



It is important to note that these partitions are allocated to the processes as they arrive and the partition that is allocated to the arrived process basically depends on the algorithm followed.

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



Advantages of Fixed-size Partition Scheme

- This scheme is simple and is easy to implement
- It supports multiprogramming as multiple processes can be stored inside the main memory.
- Management is easy using this scheme

Disadvantages of Fixed-size Partition Scheme

Some disadvantages of using this scheme are as follows:

1. Internal Fragmentation

Suppose the size of the process is lesser than the size of the partition in that case some size of the partition gets wasted and remains unused. This wastage inside the memory is generally termed as Internal fragmentation

As we have shown in the above diagram the 70 KB partition is used to load a process of 50 KB so the remaining 20 KB got wasted.

2. Limitation on the size of the process

If in a case size of a process is more than that of a maximum-sized partition then that process cannot be loaded into the memory. Due to this, a condition is imposed on the size of the process and it is:

the size of the process cannot be larger than the size of the largest partition.

3. External Fragmentation

It is another drawback of the fixed-size partition scheme as total unused space by various partitions cannot be used in order to load the processes even though there is the availability of space but it is not in the contiguous fashion.

4. Degree of multiprogramming is less

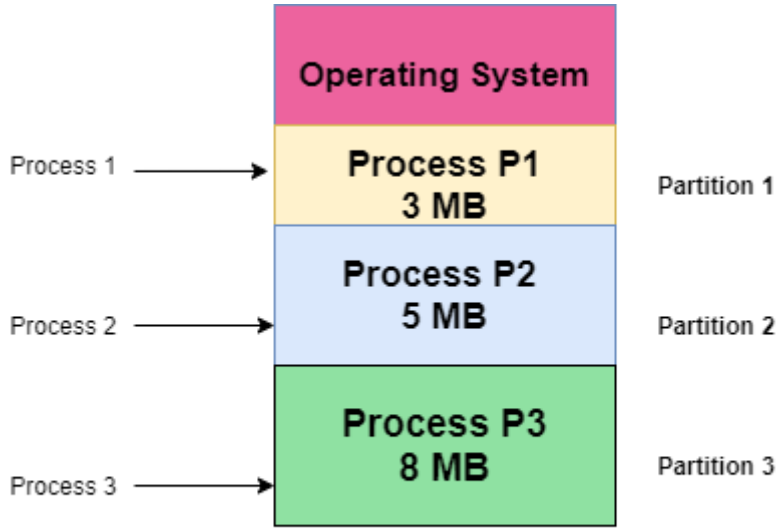
In this partition scheme, as the size of the partition cannot change according to the size of the process. Thus the degree of multiprogramming is very less and is fixed.

Variable-size Partition Scheme

This scheme is also known as **Dynamic partitioning** and is came into existence to overcome the drawback i.e internal fragmentation that is caused by **Static partitioning**. In this partitioning, scheme allocation is done dynamically.

The size of the partition is not declared initially. Whenever any process arrives, a partition of size equal to the size of the process is created and then allocated to the process. Thus the size of each partition is equal to the size of the process.

As partition size varies according to the need of the process so in this partition scheme there is no **internal fragmentation.**



Size of Partition =Size of Process

Advantages of Variable-size Partition Scheme

Some Advantages of using this partition scheme are as follows:

1. **No Internal Fragmentation** As in this partition scheme space in the main memory is allocated strictly according to the requirement of the process thus there is no chance of internal fragmentation. Also, there will be no unused space left in the partition.
2. **Degree of Multiprogramming is Dynamic** As there is no internal fragmentation in this partition scheme due to which there is no unused space in the memory. Thus more processes can be loaded into the memory at the same time.
3. **No Limitation on the Size of Process** In this partition scheme as the partition is allocated to the process dynamically thus the size of the process cannot be restricted because the partition size is decided according to the process size.

Disadvantages of Variable-size Partition Scheme

Some Disadvantages of using this partition scheme are as follows:

1. **External Fragmentation** As there is no internal fragmentation which is an advantage of using this partition scheme does not mean there will no external fragmentation.

   Let us understand this with the help of an example: In the above diagram- process P1(3MB) and process P3(8MB) completed their execution. Hence there are two spaces left i.e. 3MB and 8MB. Let's there is a Process P4 of size 15 MB comes. But the empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation.

   Because the rule says that process must be continuously present in the main memory in order to get executed. Thus it results in External Fragmentation.

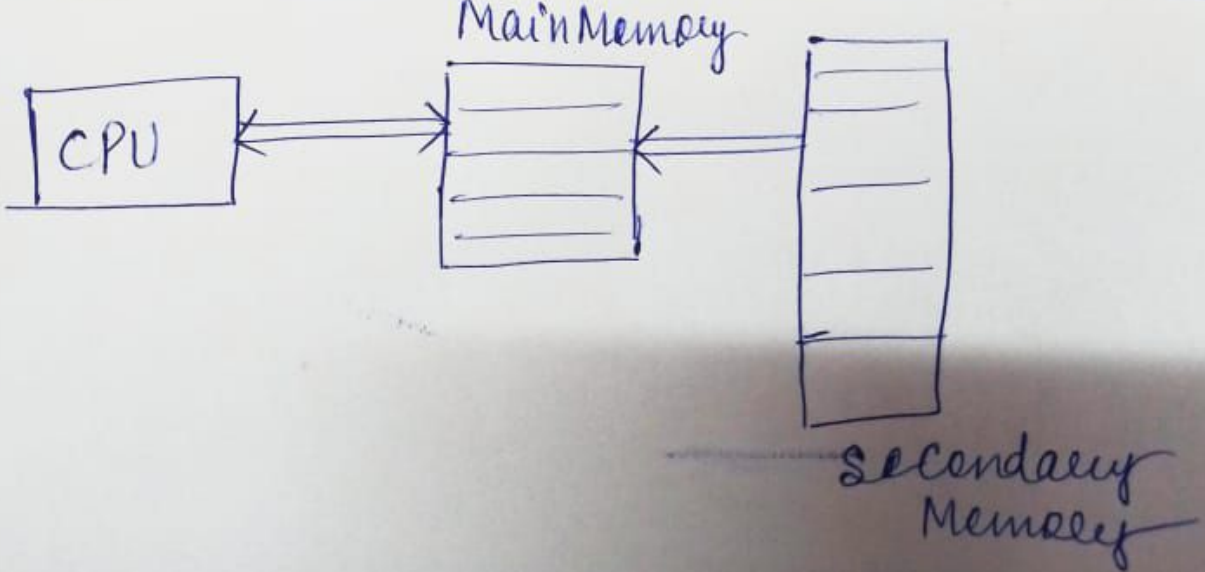
2. **Difficult Implementation** The implementation of this partition scheme is difficult as compared to the Fixed Partitioning scheme as it involves the allocation of memory at run-time rather than during the system configuration.

   As we know that OS keeps the track of all the partitions but here allocation and deallocation are done very frequently and partition size will be changed at each time so it will be difficult for the operating system to manage everything.

Memory Management Techniques

in Multiprogramming

- Contiguous Allocation
  - Fixed Partitioning or Static Partitioning or M.F.T
  - Dynamic Partitioning or M.V.T or Variable Partitioning
- Noncontiguous Allocation
  - Paging
  - Segmentation
  - Inverted Paging
  - Segmented Paging



Memory Management

- The operating system manages the Primary Memory or Main Memory. Main memory is made up of a large array of bytes or words where each byte or word is assigned a certain address.
- Main memory is a fast storage and it can be accessed directly by the CPU. For a program to be executed, it should be first loaded in the main memory. An Operating System performs the following activities for memory management:
- It keeps tracks of primary memory, i.e., which bytes of memory are used by which user program.
- The memory addresses that have already been allocated and the memory addresses of the memory that has not yet been used. In multi programming, the OS decides the order in which process are granted access to memory, and for how long.
- It Allocates the memory to a process when the process requests it and deallocates the memory when the process has terminated or is performing an I/O operation.

Definition: It Is The Functionality Of The Operating System Which Manages The 'Main Memory' And Keep Track Of Process Moving From ' Secondary Memory' To 'CPU' And Vice Versa.

Functionality :

- Keep Track Of What Memory Is In Use And What Memory Is Free
- Allocate Memory To A Process When Memory Is Free And DeAllocate Memory When They Do Not Required.
- Managing The Transfer Of Memory - Main Memory And Secondary Memory
- Prohibiting User Program To Enter In Others User Program Area And Operating System Area.
- Keep Updating Status Of Process

Functionality :

• Keep Track Of What Memory Is In Use And What Memory Is Free

• Allocate Memory To A Process When Memory Is Free And DeAllocate Memory When They Do Not Required.

• Managing The Transfer Of Memory - Main Memory And Secondary Memory

• Prohibiting User Program To Enter In Others User Program Area And Operating System Area.

• Keep Updating Status Of Process

GOALS :

• Maximize CPU Utilization

• Minimize Response Time

• Maximize Memory Management

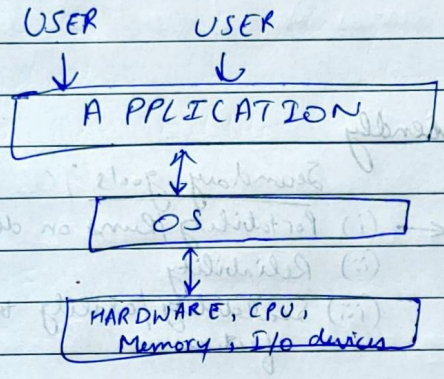
• Prioritize Important Process

classmate
Date 17/01/23
Page

Introduction to Operating Systems (OS)

* 1) This is a interface between user and Hardware.

Kernel → Core part of OS
Shell → GUI → Graphical user interface
↳ Interface → Command line interpreter → Copy file, etc



* 2) Software abstracting Hardware.
↳ like C, D, E kis ke through Dekh sakte hain
↳ OS ke through

* 3) OS is Set of Utilities to simplify application, execution.

* 4) OS is a controlled program & it act like a government.

Services of OS :-

* 1) It provide a user interface.
Eg Command line, GUI

* 2) [redacted] program execution.

* 3) I/O operation

* 4) File System manipulation.

* 5) Inter-process communication
eg word ki file ko print karna
next is printer driver ko bolega Printer kardo.

* 6) Error detection

Goals of OS :-

Primary
1) Convinience or User friendly
2) Efficient / CPU utilization
apple has no portability ← works on apple stuff only.

Secondary goals :
(i) Portability / Runs on diff hardware
(ii) Reliability
(iii) Scalability / ability to add new features

Diff. Functions of OS :-

1) Memory Management
2) Processor Management
3) Input/Output Management
4) Device Management
5) Protection & Security Management

Types of OS :-

1) Batch OS

2) Unifrogramming OS

3) Multiprogramming OS

4) Multitasking / Time sharing / Microprogramming with Round Robin / ~~Fair sha~~ Fair share OS

5) Multiuser OS

6) Multiprocessing OS

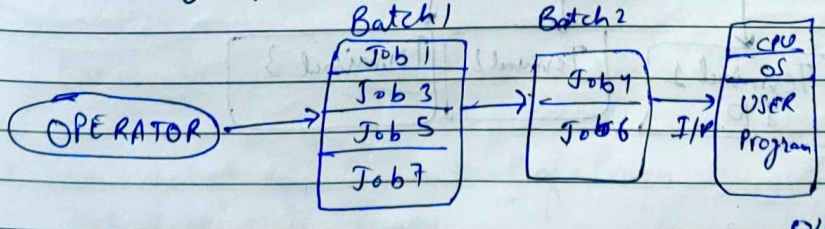
7) Embedded OS

8) Real-time OS

9) Hand-Held device OS (like mobile, tablet, etc.)

* Job → Program + Input data + Control Instruction (like ADD, SUB)

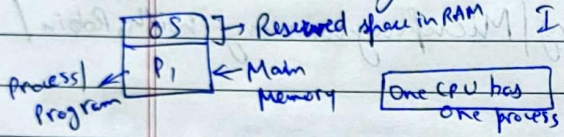
(i) Batch OS :- Jobs with similar needs are batched together and executed through processor as a group.



* In batch OS, Jobs execute one after another, saving time from activities like loading compiler.

(ii) Uniprogramming OS :- OS allows one process to reside in the main memory.

Disadvantage → CPU utilization is not good as both CPU & I/O devices are not used at same time

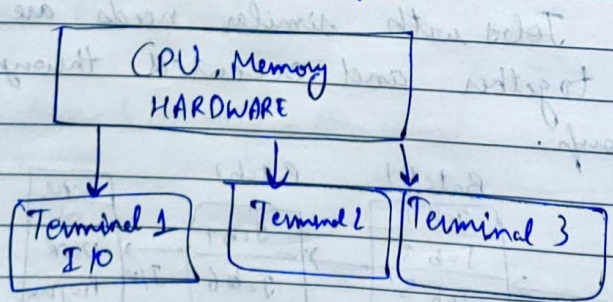


(iii) Multiprogramming OS :- OS allow multiple process to reside in the main memory.

* Degree of Multiprogramming - No. of Program (or Processes) in the main memory.

(iv) Multitasking / Time sharing :- Multitasking is multiprogram with time sharing.
There is only one CPU but switches b/w process so quickly that. It gives illusion that they are happening at same time.

(v) Multiuser OS :- Diff. Version of Unix, Linux



(vi) **Multiprocessing OS** :- Multiple CPU, Single memory
eg. DUAL CORE, QUAD CORE

- Tightly coupled → shared memory
- Loosely coupled → memory is not shared

(vii) **Embedded OS** :-

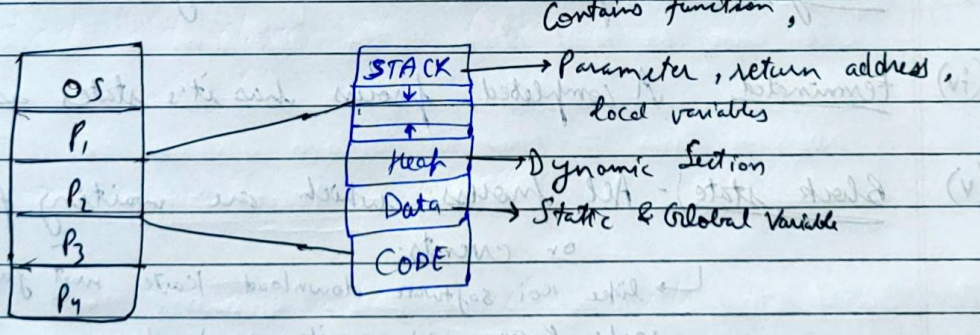
(viii) **Real-time OS** :- Real time OS are used in an environment with a large no. of events, mostly external to the computer system & it must be processed in a short time or within certain deadline.

Types :-

(i) Hard-real time OS → eg Missile launching / weather forecasting

(ii) Soft-real time OS :- Has delay
↳ Relaxation in deadline

*



Representation of Process in main memory

* **Process Descriptor / Process Control Block (PCB)** :-

↳ The content of PCB of a Process are collectively known as context of that process.

* Attributes of a Process : [Context of Process]

1) Process ID
2) Program Counter
3) GPR (or General Purpose Register)
4) List of devices
5) Size
6) Memory limit (starting, ending address, etc.)
7) Priority
8) States
9) List of files

* Different states of Process or Process life :-

(i) New :- All installed process are known to be in new state

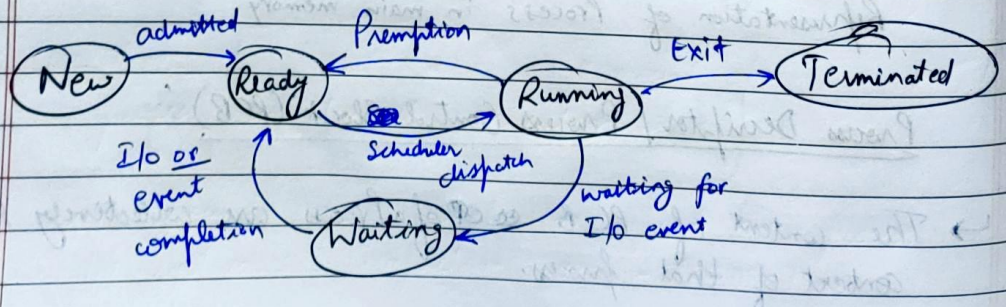
(ii) Ready :- All installed process are known to be in Ready.

(iii) Running :- The process which is running in CPU.

(iv) terminated :- A completed process has it's state as terminated.

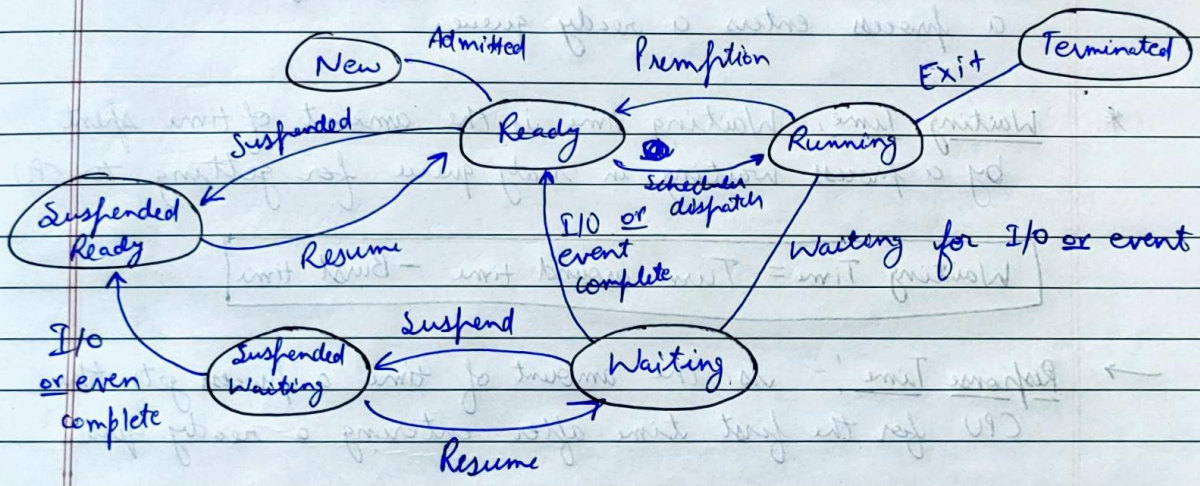
(v) Block state :- All process which are waiting for I/O device or events.

↳ like koi software download karte waqt jab tak apka nahi karoge woh waiting mein hoga.



* Long term scheduler → also control • degree of multiprogramming.

* Pehle long term New se Ready leke jaata
Phir short term select karta hai konse process ko allocate karna hai execution ke liye
Phir Dispatcher ready se running mein leke jaata hai aur execute karwa deta hai programs ko.
Aur mid term scheduler swap in yaan • swap out karwata hai.



* Types of Processes :-

(i) CPU bound :- If the process is intensive in terms of CPU operations then it is called the CPU bound.

(ii) I/O bound :- If the process is intensive in terms of I/O operations then it is called the I/O bound.

* **Context Switching**:- Content of a PCB of a process is collectively known as Context of that process.

→ Context Switching
Context of a Current running process is saved to its PCB and context of the other process is loaded to the CPU from its PCB.
(Context saved; Context load)

* **Arrival Time**:- Arrival time is the point of time at which a process enters a ready queue.

* **Waiting Time**:- Waiting time is the amount of time spent by a process waiting in ready queue for getting to CPU.

Waiting Time = Turn around time − Burst time

→ **Response Time**:- is the amount of time a process gets the CPU for the first time after entering a ready queue.

Response time = time at which process gets CPU at first time − Arrival time.

→ **Burst time**:- Execution time/Running time.
Amount of time required by a process for executing on CPU.

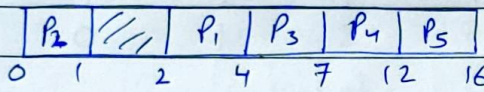
→ **Completion time**:- Exact time is amount of time at which process completes its execution.

→ **Turn around time**:- Total time spent by a process in the system = Completion Time − Arrival Time

= Burst time + Waiting Time

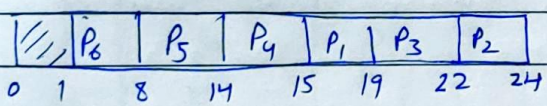
FCFS → Non-Primitive Scheduling

Process	AT	BT	CT	TAT (CT-AT)	WT (TAT-BT)	RT
P_1	2	2	4	2	0	0
P_2	0	1	1	1	0	0
P_3	2	3	7	5	2	2
P_4	3	5	12	9	4	4
P_5	4	4	16	12	8	8



Q.

Process	AT	BT	CT	TAT (CT-AT)	WT (TAT-BT)	RT
P_1	4	4	19	15	11	11
P_2	8	2	24	16	14	14
P_3	6	3	22	16	13	13
P_4	3	1	15	12	11	11
P_5	2	6	14	12	6	6
P_6	1	7	8	7	0	0



* when a process is stored in ready queue from secondary memory, it is stored as stack, heap, data, code.

as in malloc

* Different types of queue:-

(i) Job Queue :- In Hard disk (secondary memory)

(ii) Ready Queue :- In RAM (main memory)

(iii) Waiting queue :- In RAM (main memory)

WT = TAT - BT

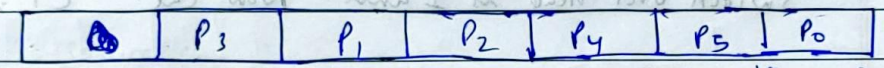
TAT = CT - AT

Q.

	AT	BT	CT	TAT	WT	RT
P_0	6	4	22	16	12	12
P_1	2	5	7	5	0	0
P_2	3	3	10	7	4	4
P_3	1	1	2	1	0	0
P_4	4	2	12	8	6	6
P_5	5	6	18	13	7	7

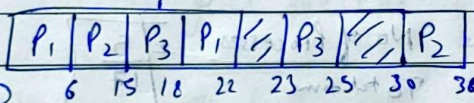
RT = WT for FCFS

Gantt chart



Q. 3 Process P_1, P_2, P_3 with process time 20, 30, 10. Each process use 30% of its process time in CPU then 50% in I/O & last 20% in CPU. Find avg. WT, CT, RT, TAT.

P_1 returns if



$P_1 - 16$
$P_2 - 30$
$P_3 - 23$

	Process time	AT	CPU BT	I/O BT	CPU BT
P_1	20	0	6	10	4
P_2	30	0	9	15	6
P_3	10	0	3	5	2

~~CPU bog~~ Disadvantage of FCFS

CPU bound process dominates I/O bound.

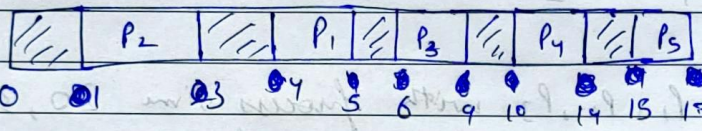
eg. as P_1 returns from I/O at t = 16 but it has to wait for 2 time units for P_3 CPU bound process to end.

CPU Utilization = $\frac{30}{36} \times 100 = 83\%$.

Q.

	AT	BT	CT	TAT	WT	RT
P_1	2	1	5	3	2	2
P_2	0	2	3	3	1	1
P_3	2	3	9	7	4	4
P_4	3	4	14	11	7	7
P_5	4	2	17	13	11	11

Assume for these process there is no I/O request & context switch over head is 1 unit. Now Cal. CT, TAT, WT



CPU Utilization = $\frac{12}{17} \times 100$

CPU throughput = $\frac{\text{No. of Processes}}{\text{Total time}} = \boxed{\frac{5}{17}}$

* Convey effect :- Short program ko bhi wait karna padta hai Long program ke liye

SJF :- (Shortest Job first) → Non Primitive, Primitive

[Optimal Algorithm]

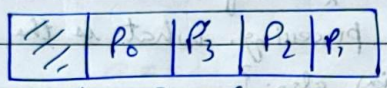
Shortest remaining time first [Primitive]

↳ Real life mein implement nahi kar sakte kyunki humein nahi pata kitni der movie dekhoge aap. Lekin iss algo ka avg. WT aur RT kam hai isliye padhte hain isko.

Starvation	AT	BT	CT	TAT	WT	RT
P_0	1	2	3	2	0	
P_1	2	5	12	10	5	
P_2	4	1	7	3	2	
P_3	3	3	6	3		

Premitive → chalte program se bhi nikal sakte hain

Non Premitive → chalte program se nikal nahi sakte

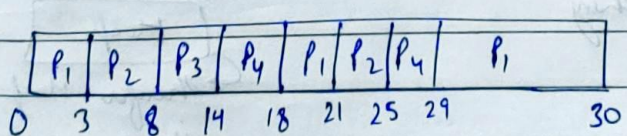


Issue :- Starvation :- bade program ki kabhi baari nahi aayegi. chote programs execute hote rahenge

Consider process P_1 goes for I/O operation for 5 unit of time after every 3 unit of execution in CPU & process P_2 goes for I/O for 2 unit after 5 unit of execution in CPU. P_3 is a CPU bound process with no I/O & P_4 goes for I/O for 1 unit after every 4 units of execution in CPU. Calculate C.T. of each process.

FCFS

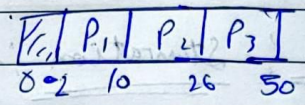
	AT	ET	I/O + CPU time
P_1	0	7	3 + (5) + 3 + (5) + 1 (I/O time)
P_2	2	9	5 + (2) + 4
P_3	4	6	No I/O time
P_4	6	8	4 + (1) + 4



Thread in OS :-

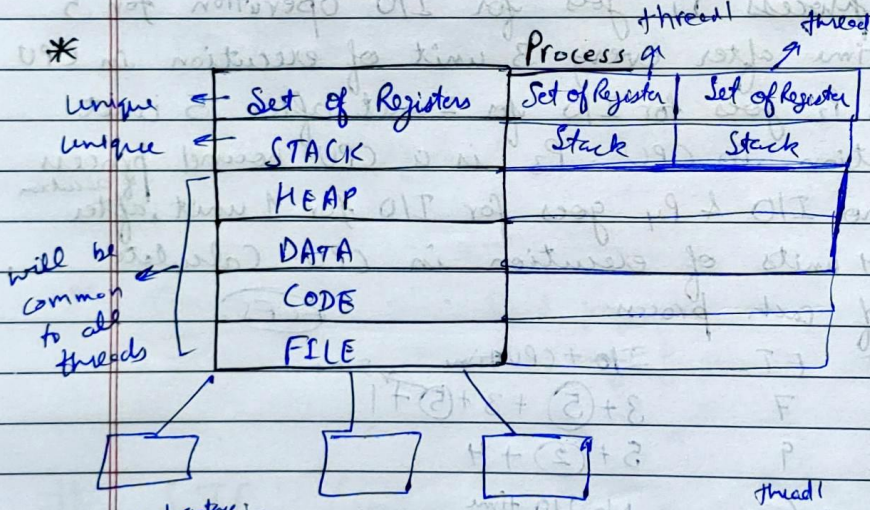
Q. 3 process P_1, P_2, P_3 arrive at time 0, the total time spent by the Process in the system 10, 20 and 30 ms respectively. They spent first 20% of execution time in I/O & rest 80% in CPU processing. what is the CPU utilization using FCFS Scheduling algo.?

	I/O	CPU
P_1	2	8
P_2	4	16
P_3	6	24



CPU Uti. = $\frac{48}{50} \times 100 = 96\%$

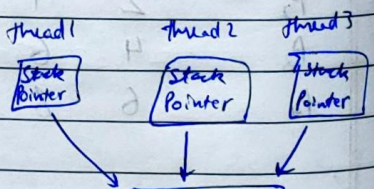
*



Disadvantages :-

① Lot of Memory

② Context Switching



Therefore Heap section is common to all

(Kyunki Current working data isimein store kiya jaata hai

* Heap is not a per thread attribute

* Stack ✓ is a per thread attribute
  Set of Register

eg ① main() f1() → ⑤ f2
② f2 f1() → ④ f3 main()

main()
{
int a → 4
a = a+1 → 5

* Multithreading : web browser, word

	User level thread	Kernel level thread
1. Creation of thread	It is created by user or application developer.	User process will make a system call
2. Context Switching	It will take less time.	It will take more time.
3. [illegible]	No fair sharing of time quantum in user level thread eg P1: 10 threads, 10 unit; P2: 1 thread, 10 unit	
	level of concurrency is decreased in user level thread.	

classmate
Date
Page

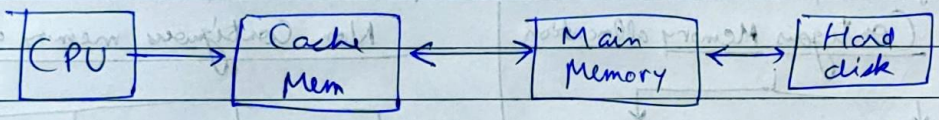
Memory management → RAM

* Wastage of space is called Fragmentation.

CPU Utilization
Responsiveness
} Benefits of Scheduling algos

Note :- Disk Management → Secondary memory

* RAM → Size ↑
Access time ↓
Cost ↓



↳ Collection of ~~Program is~~ instruction is program. OS ka module

Collection of address is addres space → Physical address
→ Logical address

↳ Collection of RAM address is physical address

↳ Collection of address of process is Logical addres / Virtual address

1 word = 4 byte / 8 byte

* Memory management ke function :-

(i) Allocation
(ii) Deallocation
(iii) Protection → Ek process doosre process ke memory space mein interfere nahi kar sakta

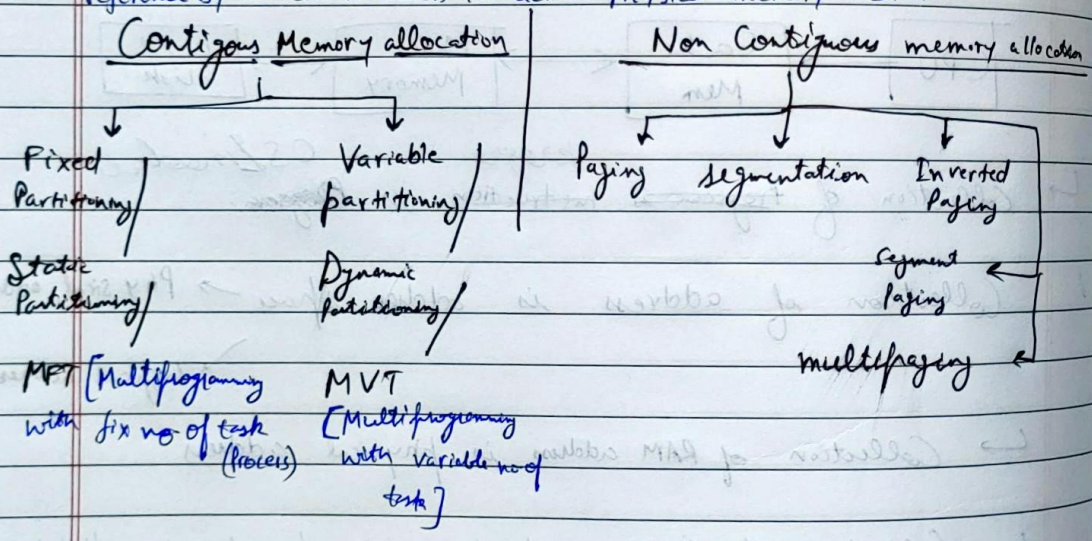
* **Memory management goals :-**
  - ↳ Max utilization of space
  - ↳ Ability of larger program/process with limited space.

Q. Why Hit ratio is more then 90%

Local Spatiality (something something)

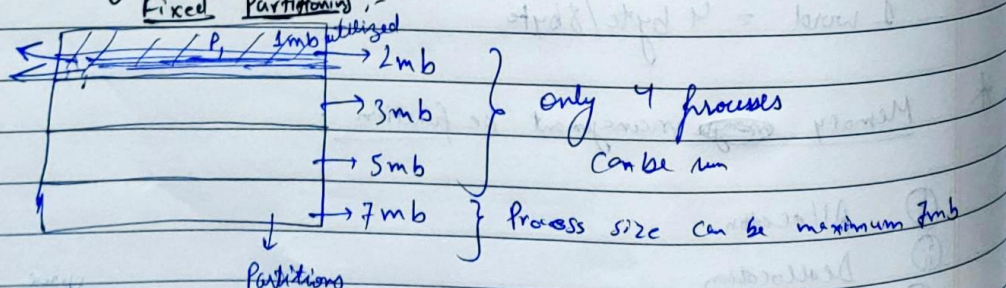
Q. Why CPU generates logical address but not physical address.

A logical address is generated by CPU while a program is running. Since a logical address does not physically exists it is known as a virtual address. This address is used as a reference by the CPU to access the actual physical memory location.



* **Contiguous Memory allocation :-**

Fixed Partitioning :-



* **Disadvantages of Fixed Partitioning :-**

(i) Internal fragmentation

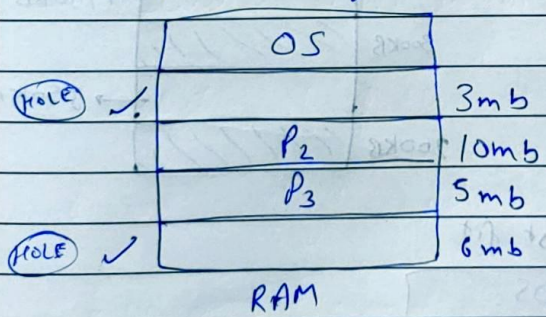
(ii) Degree of multiprogramming is limiting by no. of processes

(iii) Process size is limited by the size of largest partitioning.

→ One partition can only hold one process.

* **Variable Partitioning :-** In Variable Partitioning the O.S. keeps a table indicating which parts of memory are available & which are occupied. Initially all memories are available for user process and consider one large block of available memory called Hole.

* External fragmentation exists when there is enough total memory or space to satisfy a request but the available space are not contiguous.
Storage is fragmented into large no. of small holes.



* **Advantages of Variable Partitioning :-**

(i) No internal fragmentation.

(ii) Degree of multiprogramming is not limited

(iii) Size of a process is not limited by the size of

largest partitioning rather than it is limited by size of RAM.

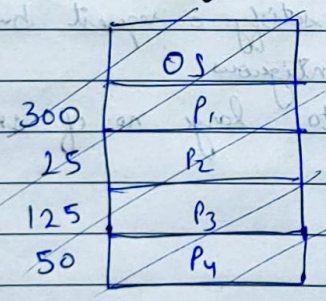
* **Disadvantages of Variable Partitioning :-**

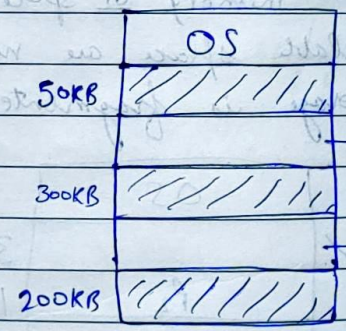
(i) External Fragmentation

☆ When available (memory holes) are more than one and they are big enough to allocate the process then fits are:

(i) first fit
(ii) Next fit
(iii) best fit
(iv) worst fit

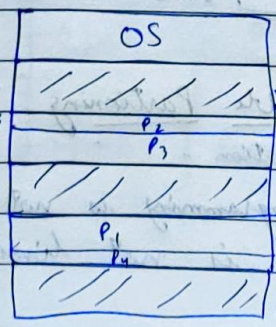
Q. The sequence of request for block of size 300, 25, 125, 50 can be satisfied if we use all allocation fits.

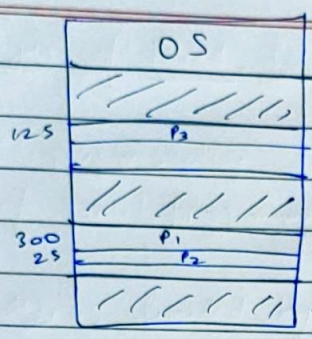




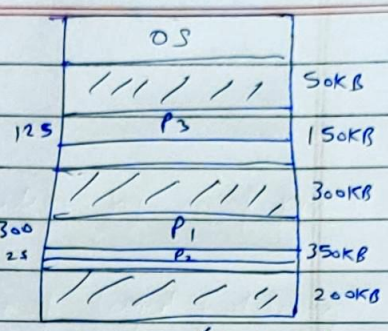
P1 P2 P3 P4
300, 25, 125, 50
↓
First
Worst

First fit



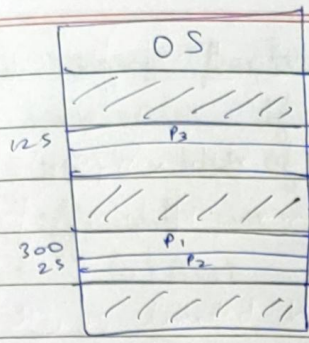


Next fit (Can't fit)

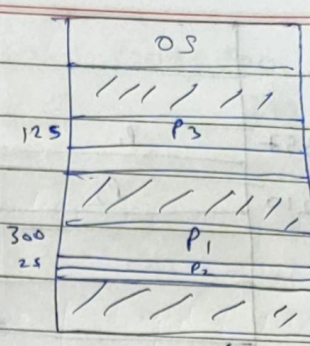


Best fit (Can't fit)

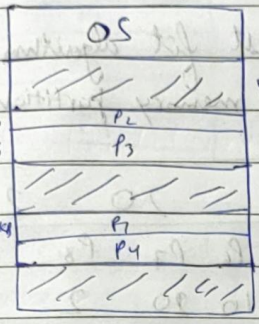
Worst fit



Next fit (Can't fit)

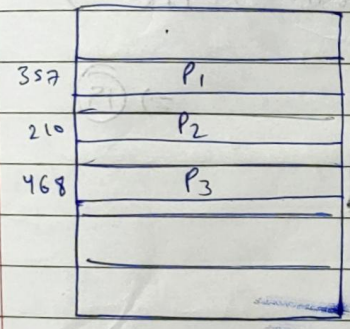


Best fit (Can't fit)

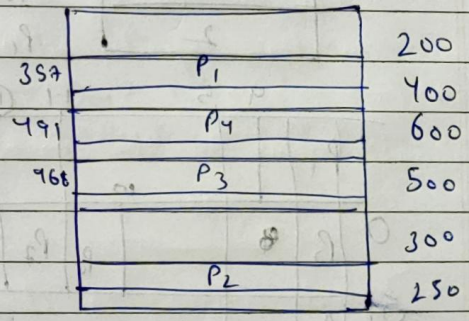


Worst fit

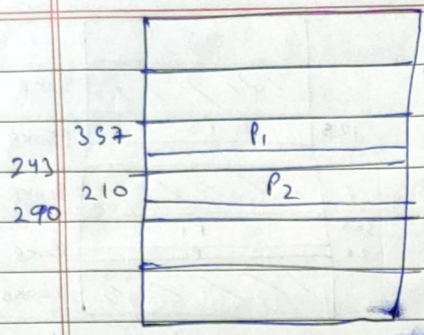
Q. Consider 6 memory partition of size of 200KB, 400, 600, 500, 300 & 250 These partition need to be allocated to 4 process of size of 357 (P_1), 210 (P_2), 468 (P_3), 491 (P_4). Perform the allocation of process using first, best & worst.



first (Not Possible)



best

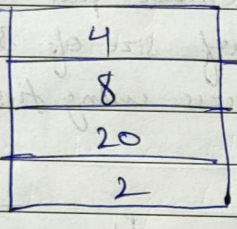


Worst (Not Possible)

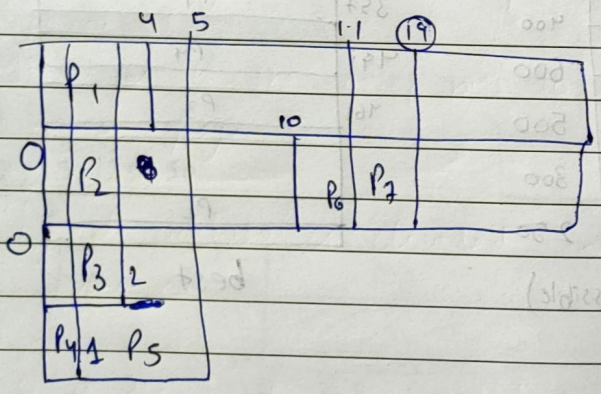
Q.. In a Computer system, best fit algorithm is used for allocating job (process) to memory partition. (Fixed partitioning)

Partition Size		4	8	20	2
Process	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
Size	2KB	14	3	6	6	10	20	2
Execution time	4	10	2	1	4	1	8	6

When P_7 process will finish the execution?



P_3 (0-2)
P_4 (0-1), P_5 (1-5)
P_2 (0-10), P_6 (10-11), P_7 (11-19)
P_1 (0-4)

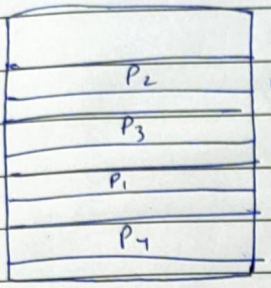


⇒ 19

Q. Consider 5 memory partition of size 100 KB, 500, 200, 450 & 600 in same order. If sequence of process, 212, 417, 112, 426 KB. which of the following algorithm makes efficient use of memory (Fixed partition as partition is given)

(i) Best (ii) First (iii) Worst (iv) Next

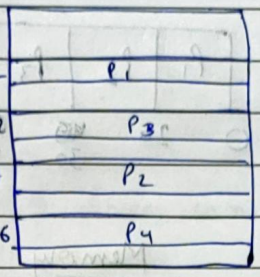
(i)



83
88
238
174
583

Best Fit

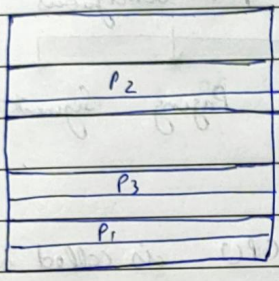
(ii)



288
88
33
174
583

First

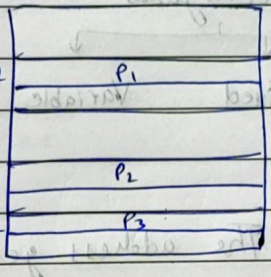
(iii)



83
338
388

Worst (Not Possible)

(iv)



288
33
488

Next (Not Possible)

SJF (Non ~~Preemptive~~ Preemptive)

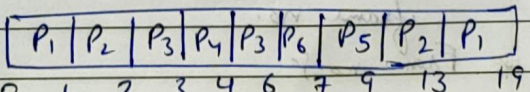
SRTF (Preemptive) → Don't suffer from convoy effect
↳ Has less waiting & Response time

~~Advantages~~ :→

↓
Shortest Remaining Time First

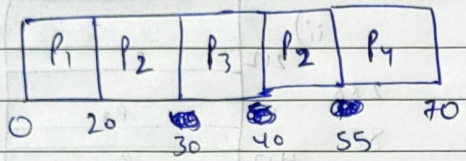
Q.

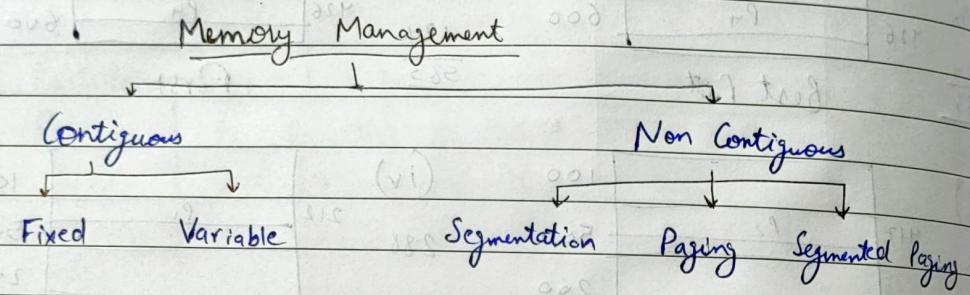
	AT	BT
P_1	0	~~7~~ 6
P_2	1	~~5~~ 4
P_3	2	~~3~~ 1
P_4	3	1
P_5	4	2
P_6	5	1



Q.

	AT	BT	CT	RT
P_1	0	20	20	0
P_2	15	25	55	5
P_3	30	10	40	0
P_4	45	15	70	10





* → The address generated by the CPU is called logical address. It is divided into the page numbers & page offsets.

* Page number :-
  ↳ It is the no. of bits needed to represent the pages in the logical address space or page number.

* Page offset :-
  It refers to the no. of bits to represent a certain word in a page.

* → Physical address is divided into frame offset & frame no.

* Frame no :-
  ↳ It is the no. of bits needed to indicate a frame of the physical address space or frame no.

classmate
Date
Page

* Frame offset :
refers to the no of bits necessary to represent a certain words in a frame

* $\log_2$ (No. of Pages in process) ⟶ Bits in page no.

* No. of bits in Page offset → $\log_2$ (size of Page)

* bits in frame no. ⟶ $\log_2$ (no. of frames in main memory)

$\frac{\text{Memory Size}}{\text{Page Size}}$ ⟶ No. of frames

No. of Pages ⟶ $\frac{\text{Process Size}}{\text{Page Size}}$