OPERATING SYSTEM

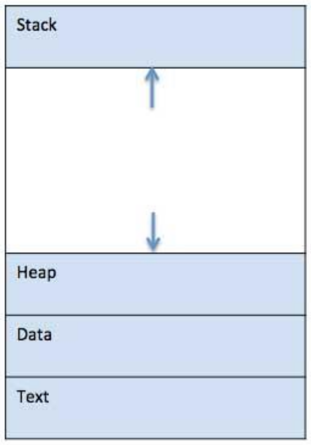
UNIT-2

CPU SCHEDULING

Process:-

A program in execution is called process. The processer **(A processor, is a small chip that resides in computers and other electronic devices. Its basic job is to receive input and provide the appropriate output.)** has to manage several activities at one time. Each activity is correspond to one process. Process execution must progress in sequential fashion. The OS must maintain a data structure for each process, which describes the state and resource ownership of that process, and which enables the OS to exert control over each process.

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 –



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.

Difference between Process and a Program:-

```
Main()
{
 int i, prod=1;
for(i=0;i<100;i++)
prod=prod*i;
}
```

- This is a program that contain one multiplication statement(prod=prod*i) but the process will execute 100 multiplication.
- Process is active entity.
- Program is passive entity.

SR.NO.	PROGRAM	PROCESS
1.	Program contains a set of instructions designed to complete a specific task.	Process is an instance of an executing program.
2.	Program is a passive entity as it resides in the secondary memory.	Process is a active entity as it is created during execution and loaded into the main memory.
3.	Program exists at a single place and continues to exist until it is deleted.	Process exists for a limited span of time as it gets terminated after the completion of task.
4.	Program is a static entity.	Process is a dynamic entity.
5.	Program does not have any resource requirement, it only requires memory space for storing the instructions.	Process has a high resource requirement, it needs resources like CPU, memory address, I/O during its lifetime.
6.	Program does not have any control block.	Process has its own control block called Process Control Block.

Process States or Process Life Cycle:-

When a process comes in execution it change its states. The state of a process is defined in part by the current activity of that process. Each process may be one of the following state during the time of execution.

New: In this state the process is being created.

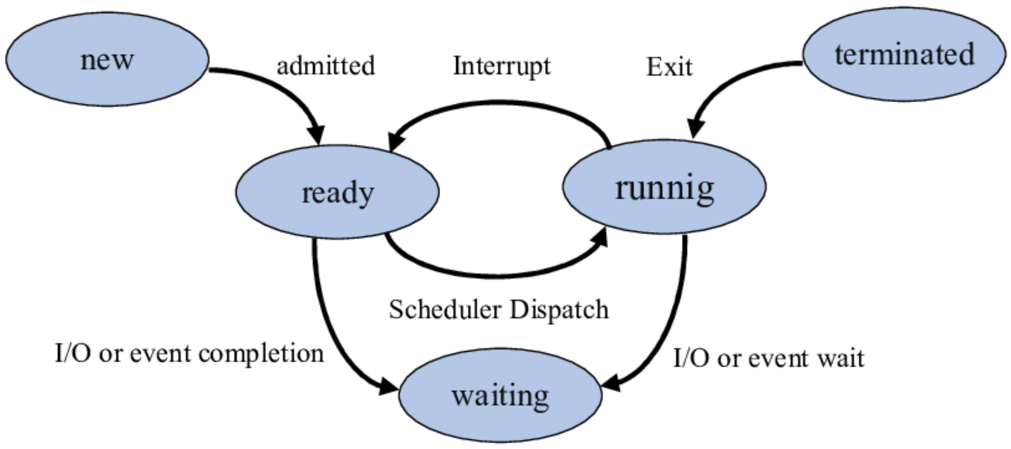
Ready: In this state the process is to be assign a duty by the processor.

Waiting: In this state the process is waiting or blocked for some event to occur such as input/output completion.

Running: In this state Instructions are executed.

Terminated: In this state the process has finished the execution

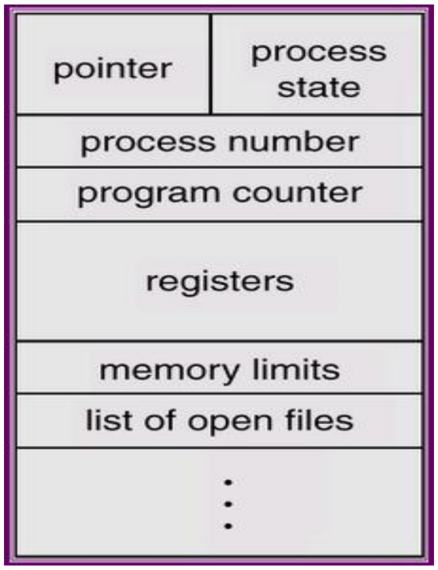
State Transition Diagram:-



Process Control Block (PCB):-

In OS each process is represented by a process control block or task control block. It is a data structure that physically represent a process in the memory of a computer system. It contains pieces of information associate with a specific process that include following:-

PCB



Pointer:- A pointer to parent process.

Process State:- It indicates the information about the state of process such as blocked ready running etc.

Process ID:- Each process is assigned a unique identification number, when it is entered into the system.

Program Counter:- It indicates the address of the next instruction to be executed.

CPU Registers:- It indicates the information about the contents of the CPU registers. The information of CPU registers must be saved when an interrupt occurs, so that the process can be continued correctly afterward. Registers hold the processed the result of calculations or addresses pointing to the memory locations of desired data.

CPU Scheduling Information:- It indicates the information needed for CPU scheduling such as process priority, pointers to scheduling queues and other scheduling parameters.

Memory Management Information:- It indicates the information needed for memory management such as value of the base and limit registers, page tables or segment tables, amount of memory units allocated to the process etc.

Accounting Information:- It indicates the information about process number, CPU used by the process time limits etc.

I/O Status Information:- It indicates the information about I/O devices allocated to the process a list of open files access rights of files opened and so on,

Link to Parent Process:- A new process can be created from existing process; the existing process is called the parent process of the newly created process. The address of the PCB of parent process is stored.

Link to Child Process:- The addresses of the PCBs of the child processes in the main memory are stored.

CPU Scheduling:-

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

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

Schedulers:-

A scheduler is a type of system software that allows you to handle process scheduling.

There are mainly three types of Process Schedulers:

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

Long Term Scheduler or Job Scheduler :-

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

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

Medium Term Scheduler:-

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

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

Short Term Scheduler:-

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

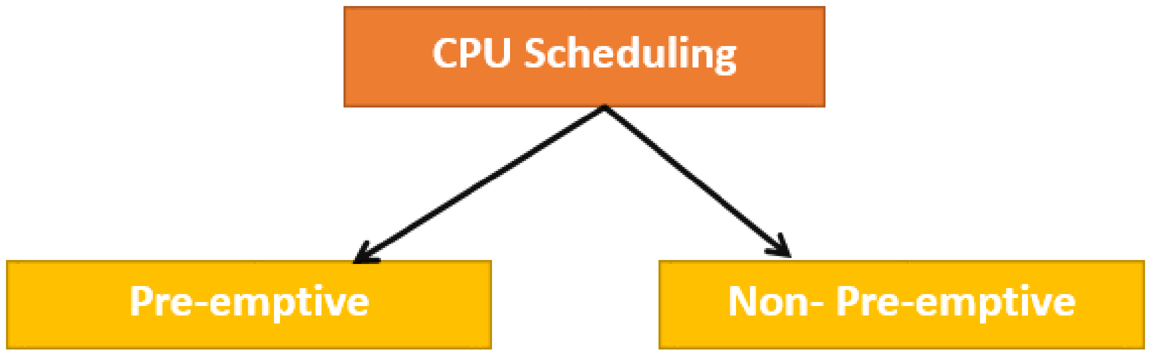
Difference between Schedulers:-

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

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

Types of CPU Scheduling

Here are two kinds of Scheduling methods:



Preemptive Scheduling

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

Non-Preemptive Scheduling

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

When scheduling is Preemptive or Non-Preemptive?

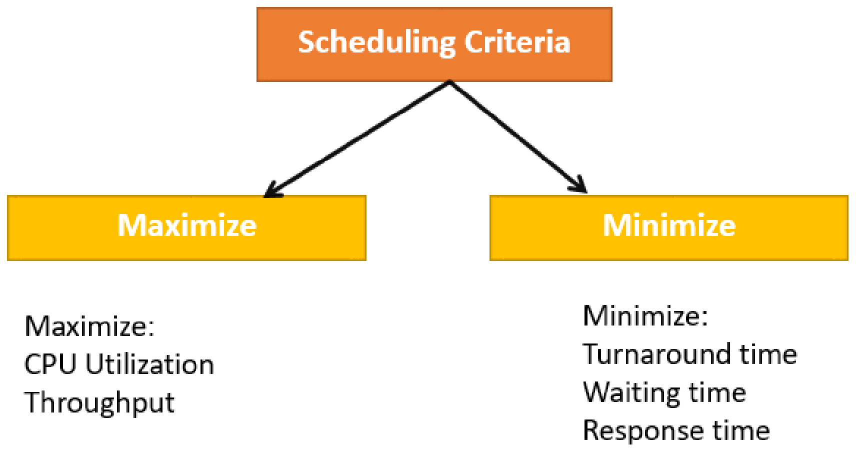
To determine if scheduling is preemptive or non-preemptive, consider these four parameters:

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

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

CPU Scheduling Criteria

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



<u>Maximize:</u>

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

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

<u>Minimize:</u>

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

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

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

Interval Timer

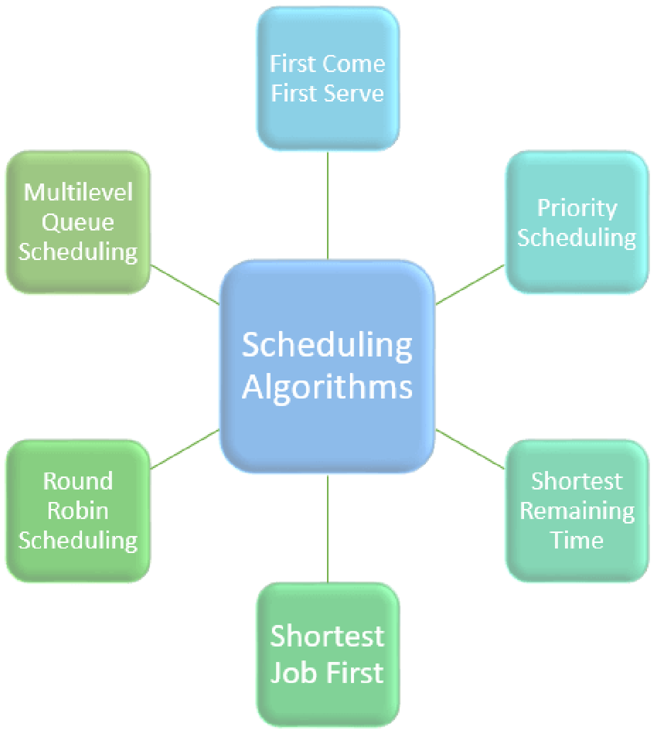
Timer interruption is a method that is closely related to preemption. When a certain process gets the CPU allocation, a timer may be set to a specified interval. Both timer interruption and preemption force a process to return the CPU before its CPU burst is complete.

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

Types of CPU scheduling Algorithm

There are mainly six types of process scheduling algorithms

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



<u>Scheduling Algorithms</u>

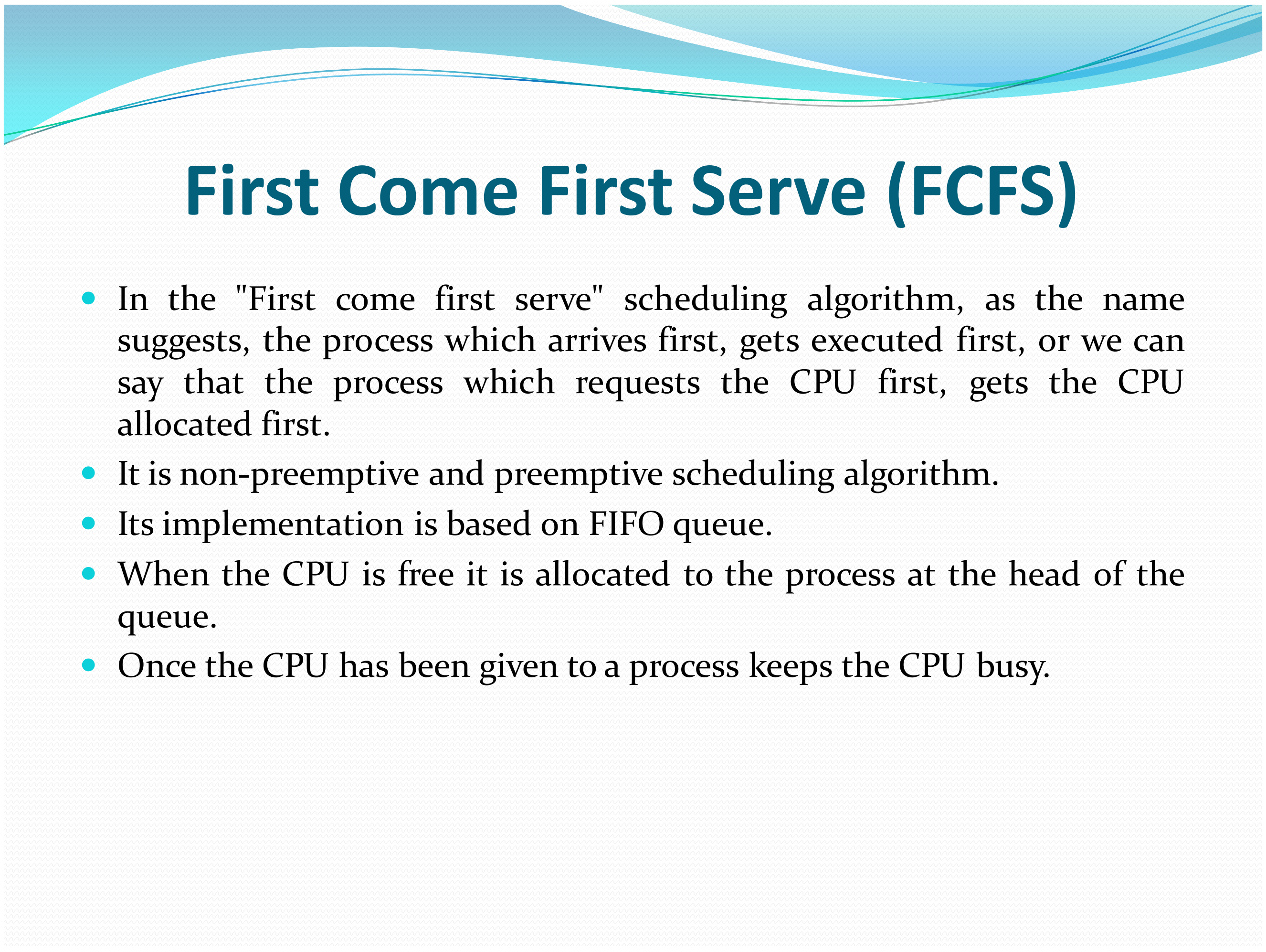
First Come First Serve

First Come First Serve is the full form of FCFS. It is the easiest and most simple CPU scheduling algorithm. In this type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be managed with a FIFO queue.

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

Characteristics of FCFS method:

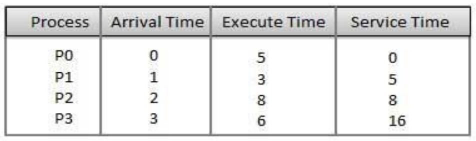
- It offers non-preemptive and pre-emptive scheduling algorithm.
- Jobs are always executed on a first-come, first-serve basis.
- It is easy to implement and use.
- However, this method is poor in performance, and the general wait time is quite high.



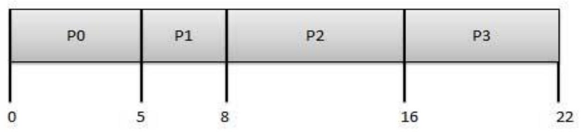
First Come First Serve (FCFS)

- In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.
- It is non-preemptive and preemptive scheduling algorithm.
- Its implementation is based on FIFO queue.
- When the CPU is free it is allocated to the process at the head of the queue.
- Once the CPU has been given to a process keeps the CPU busy.

First Come First Serve (FCFS)



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



Wait time of each process is as follows –

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

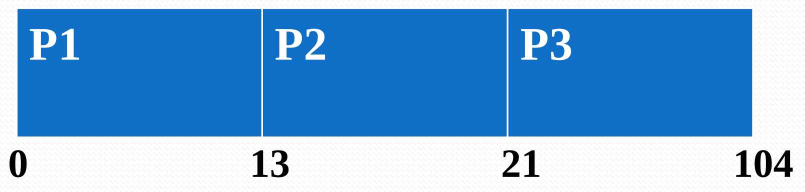
Lets take an example:-

Q:-Draw the Gantt chart for FCFS at arrival time zero and burst-time is given in micro second calculate average waiting time and also calculate turn around time for the same process.

Process	Burst Time
P1	13
P2	08
P3	83

Solution:-

Gantt Chart:-



Waiting time for each process is-

Process	Waiting Time
P1	00
P2	13
P3	21

- **Average Waiting Time=** $\frac{0+13+21}{3} = \frac{34}{3}$ = 11.33 Micro-second
- **Turn Around Time:-**

Process	Turn Around Time
P1	13-0=13
P2	21-13=08
P3	104-21=83

- **Average Turn Around Time** $= \frac{13+8+83}{3} = \frac{104}{3}$

$=$ 3.46 Micro-second

Problems with FCFS Scheduling:-

Below we have a few shortcomings or problems with the FCFS
scheduling algorithm:

- It is **Non Pre-emptive** algorithm, which means the **process priority** doesn't matter. If a process with very least priority is being executed, more like **daily routine backup** process, which takes more time, and all of a sudden some other high priority process arrives, like **interrupt to avoid system crash**, the high priority process will have to wait, and hence in this case, the system will crash, just because of improper process scheduling.
- Not optimal Average Waiting Time.
- Resources utilization in parallel is not possible, which leads to **Convoy Effect**, and hence poor resource(CPU, I/O etc) utilization.

Shortest Remaining Time

The full form of SRT is Shortest remaining time. It is also known as SJF preemptive scheduling. In this method, the process will be allocated to the task, which is closest to its completion. This method prevents a newer ready state process from holding the completion of an older process.

Characteristics of SRT scheduling method:

- This method is mostly applied in batch environments where short jobs are required to be given preference.
- This is not an ideal method to implement it in a shared system where the required CPU time is unknown.
- Associate with each process as the length of its next CPU burst. So that operating system uses these lengths, which helps to schedule the process with the shortest possible time.

Shortest Job First

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

Shortest Job First scheduling works on the process with the shortest **burst time** or **duration** first.

- This is the best approach to minimize waiting time.
- This is used in Batch Systems.
- It is of two types:
    1. Non Pre-emptive

2. Pre-emptive

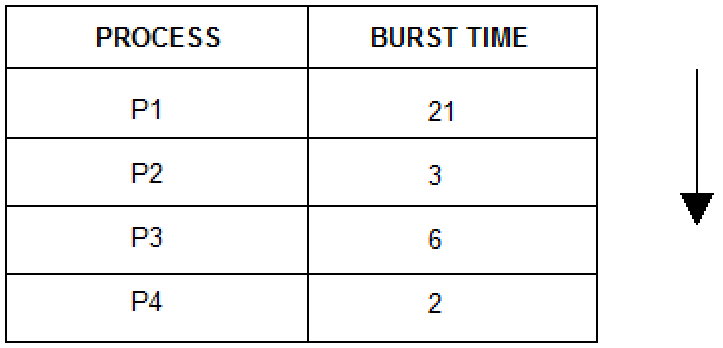
- To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time.
- This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (either Arrival time is 0 for all, or Arrival time is same for all.

Characteristics of SJF Scheduling

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

Non Pre-emptive Shortest Job First

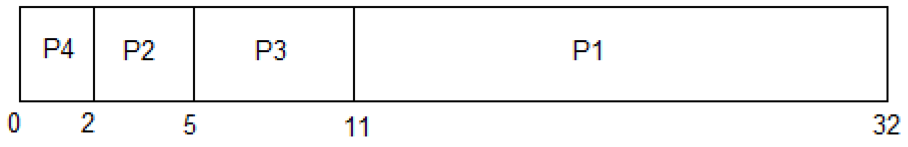
Consider the below processes available in the ready queue for execution, with **arrival time** as 0 for all and given **burst times**.



PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :



Now, the average waiting time will be = (0 + 2 + 5 + 11)/4 = 4.5 ms

As you can see in the **GANTT chart** above, the process **P4** will be picked up first as it has the shortest burst time, then **P2**, followed by **P3** and at last **P1**.

We scheduled the same set of processes using the First come first serve algorithm in the previous tutorial, and got average waiting time to be 18.75 ms, whereas with SJF, the average waiting time comes out 4.5 ms.

Problem with Non Pre-emptive SJF

If the **arrival time** for processes are different, which means all the processes are not available in the ready queue at time 0, and some jobs arrive after some time, in such situation, sometimes process with short burst time have to wait for the current process's execution to finish, because in Non Pre-emptive SJF, on arrival of a process with short duration, the existing job/process's execution is not halted/stopped to execute the short job first.

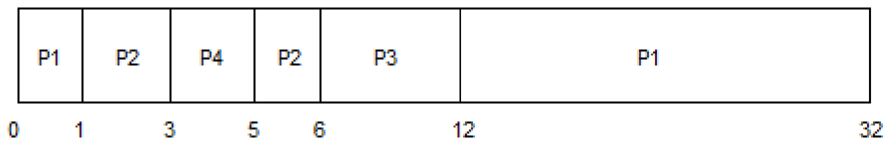
This leads to the problem of **Starvation**, where a shorter process has to wait for a long time until the current longer process gets executed. This happens if shorter jobs keep coming, but this can be solved using the concept of **aging**.

Pre-emptive Shortest Job First

In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with **short burst time** arrives, the existing process is preempted or removed from execution, and the shorter job is executed first.

PROCESS	BURST TIME	ARRIVAL TIME
P1	21	0
P2	3	1
P3	6	2
P4	2	3

The GANTT chart for Preemptive Shortest Job First Scheduling will be,



The average waiting time will be, ((5-3) + (6-2) + (12-1))/4 = 4.25 ms

The average waiting time for preemptive shortest job first scheduling is less than both, non-preemptive SJF scheduling and FCFS scheduling.

As you can see in the **GANTT chart** above, as **P1** arrives first, hence it's execution starts immediately, but just after 1 ms, process **P2** arrives with a **burst time** of 3 ms which is less than the burst time of **P1**, hence the process **P1**(1 ms done, 20 ms left) is preemptied and process **P2** is executed.

As **P2** is getting executed, after 1 ms, **P3** arrives, but it has a burst time greater than that of **P2**, hence execution of **P2** continues. But after another millisecond, **P4** arrives with a burst time of 2 ms, as a result **P2**(2 ms done, 1 ms left) is preemptied and **P4** is executed.

After the completion of **P4**, process **P2** is picked up and finishes, then **P2** will get executed and at last **P1**.

The Pre-emptive SJF is also known as **Shortest Remaining Time First**, because at any given point of time, the job with the shortest remaining time is executed first.

<u>Priority Based Scheduling</u>

Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler selects the tasks to work as per the priority.

Priority scheduling also helps OS to involve priority assignments. The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis. Priority can be decided based on memory requirements, time requirements, etc.

In case of priority scheduling the priority is not always set as the inverse of the CPU burst time, rather it can be internally or externally set, but yes the scheduling is done on the basis of priority of the process where the process which is most urgent is processed first, followed by the ones with lesser priority in order.

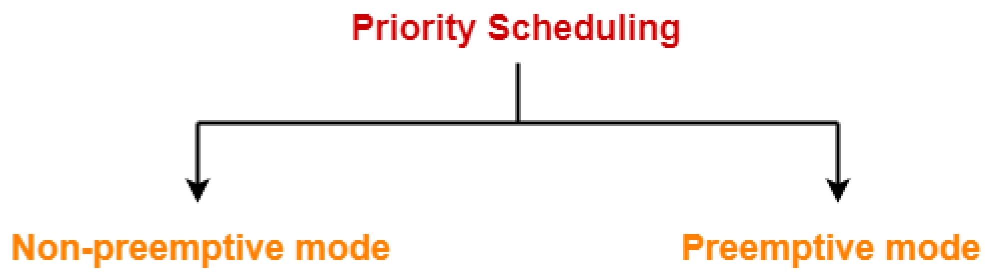
Processes with same priority are executed in FCFS manner.

The priority of process, when internally defined, can be decided based on **memory requirements**, **time limits** ,**number of open files**, **ratio of I/O burst to CPU burst** etc.

Whereas, external priorities are set based on criteria outside the operating system, like the importance of the process, funds paid for the computer resource use, makrte factor etc.

Types of Priority Scheduling Algorithm

Priority scheduling can be of two types:



1. **Preemptive Priority Scheduling**: If the new process arrived at the ready queue has a higher priority than the currently running process, the CPU is preempted, which means the processing of the current process is stoped and the incoming new process with higher priority gets the CPU for its execution.
2. **Non-Preemptive Priority Scheduling**: In case of non-preemptive priority scheduling algorithm if a new process arrives with a higher priority than the current running process, the incoming process is put at the head of the ready queue, which means after the execution of the current process it will be processed.

Advantages-

- It considers the priority of the processes and allows the important processes to run first.
- Priority scheduling in preemptive mode is best suited for real time operating system.

Disadvantages-

- Processes with lesser priority may starve for CPU.
- There is no idea of response time and waiting time.

Important Notes-

Note-01:

- The waiting time for the process having the highest priority will always be zero in preemptive mode.
- The waiting time for the process having the highest priority may not be zero in non-preemptive mode.

Note-02:

Priority scheduling in preemptive and non-preemptive mode behaves exactly same under following conditions-

- The arrival time of all the processes is same
- All the processes become available

PRACTICE PROBLEMS BASED ON PRIORITY SCHEDULING-

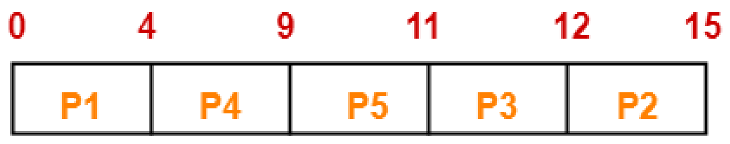
Problem-01:

Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time	Priority
P1	0	4	2
P2	1	3	3
P3	2	1	4
P4	3	5	5
P5	4	2	5

If the CPU scheduling policy is priority non-preemptive, calculate the average waiting time and average turn around time. (Higher number represents higher priority)

Solution- Gantt Chart-



Gantt Chart

Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	4	4 – 0 = 4	4 – 4 = 0
P2	15	15 – 1 = 14	14 – 3 = 11
P3	12	12 – 2 = 10	10 – 1 = 9

P4	9	9 – 3 = 6	6 – 5 = 1
P5	11	11 – 4 = 7	7 – 2 = 5

- Average Turn Around time = (4 + 14 + 10 + 6 + 7) / 5 = 41 / 5 = 8.2 unit
- Average waiting time = (0 + 11 + 9 + 1 + 5) / 5 = 26 / 5 = 5.2 unit

Problem-02:

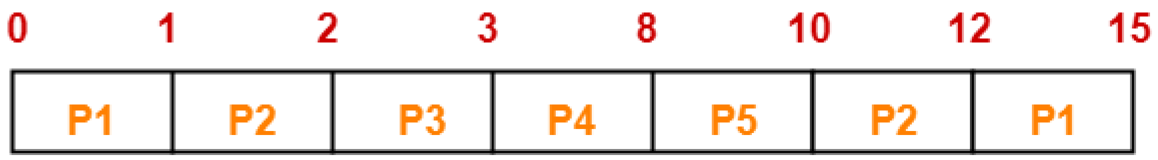
Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	**Arrival time**	**Burst time**	**Priority**
P1	0	4	2
P2	1	3	3
P3	2	1	4
P4	3	5	5
P5	4	2	5

If the CPU scheduling policy is priority preemptive, calculate the average waiting time and average turn around time. (Higher number represents higher priority)

Solution-

Gantt Chart-



Gantt Chart

Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	**Exit time**	**Turn Around time**	**Waiting time**
P1	15	15 – 0 = 15	15 – 4 = 11
P2	12	12 – 1 = 11	11 – 3 = 8
P3	3	3 – 2 = 1	1 – 1 = 0
P4	8	8 – 3 = 5	5 – 5 = 0
P5	10	10 – 4 = 6	6 – 2 = 4

Now,

- Average Turn Around time = (15 + 11 + 1 + 5 + 6) / 5 = 38 / 5 = 7.6 unit
- Average waiting time = (11 + 8 + 0 + 0 + 4) / 5 = 23 / 5 = 4.6 unit

Problem:3

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

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

Waiting time of each process is as follows –

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

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

Problem with Priority Scheduling Algorithm

In priority scheduling algorithm, the chances of **indefinite blocking** or **starvation**.

A process is considered blocked when it is ready to run but has to wait for the CPU as some other process is running currently.

But in case of priority scheduling if new higher priority processes keeps coming in the ready queue then the processes waiting in the ready queue with lower priority may have to wait for long durations before getting the CPU for execution.

In 1973, when the IBM 7904 machine was shut down at MIT, a low-priority process was found which was submitted in 1967 and had not yet been run.\

Round-Robin Scheduling

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

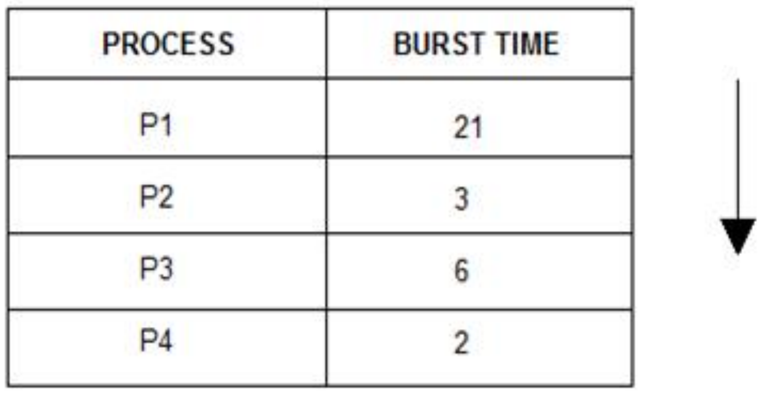
Characteristics of Round-Robin Scheduling

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

Key Points:-

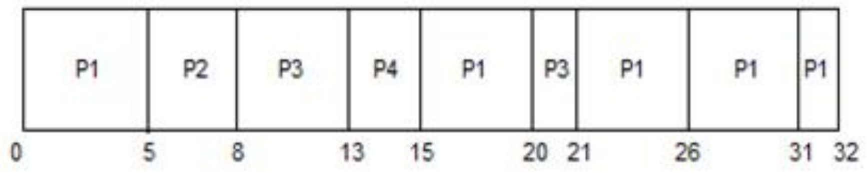
- A fixed time is allotted to each process, called **quantum**, for execution.
- Once a process is executed for given time period that process is preempted and other process executes for given time period.
- Context switching is used to save states of preempted processes.

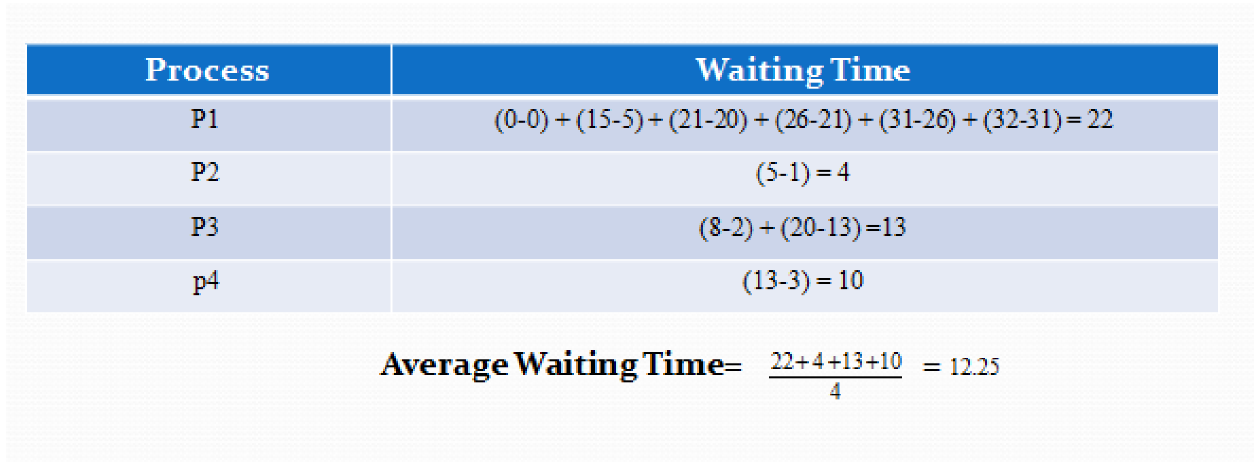
Example:- In the given table Process(P1, P2, P3, P4) and burst time(21, 3, 6, 2) is given. The time quantum is 5 make a gantt chart and also calculate average waiting time.



PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

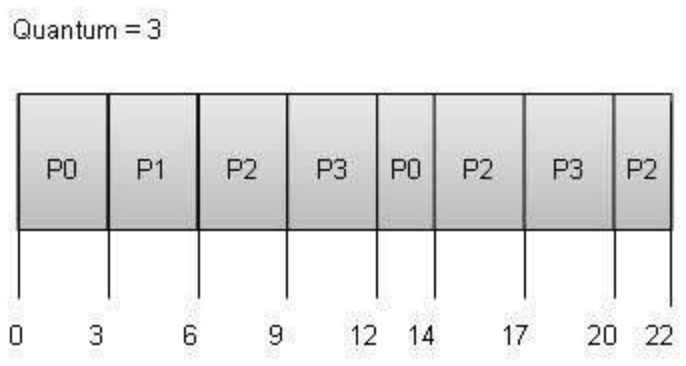
The GANTT chart for round robin scheduling will be,





Process	Waiting Time
P1	(0-0) + (15-5) + (21-20) + (26-21) + (31-26) + (32-31) = 22
P2	(5-1) = 4
P3	(8-2) + (20-13) =13
p4	(13-3) = 10

Average Waiting Time= $\frac{22+4+13+10}{4}$ = 12.25



Q2.

Wait time of each process is as follows −

Process	**Wait Time : Service Time - Arrival Time**
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

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

Multiple-Level Queues Scheduling

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

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

Characteristic of Multiple-Level Queues Scheduling:

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

The Purpose of a Scheduling algorithm

Here are the reasons for using a scheduling algorithm:

- The CPU uses scheduling to improve its efficiency.

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

Multiple-Processor Scheduling in Operating System

In multiple-processor scheduling **multiple CPU's** are available and hence **Load Sharing** becomes possible. However multiple processor scheduling is more **complex** as compared to single processor scheduling. In multiple processor scheduling there are cases when the processors are identical i.e. HOMOGENEOUS, in terms of their functionality, we can use any processor available to run any process in the queue.

Approaches to Multiple-Processor Scheduling –

One approach is when all the scheduling decisions and I/O processing are handled by a single processor which is called the **Master Server** and the other processors executes only the **user code**. This is simple and reduces the need of data sharing. This entire scenario is called **Asymmetric Multiprocessing**.

A second approach uses **Symmetric Multiprocessing** where each processor is **self scheduling**. All processes may be in a common ready queue or each processor may have its own private queue for ready processes. The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.