Process Management in OS

A Program does nothing unless its instructions are executed by a CPU. A program in execution is called a process. In order to accomplish its task, process needs the computer resources.

There may exist more than one process in the system which may require the same resource at the same time. Therefore, the operating system has to manage all the processes and the resources in a convenient and efficient way.

Some resources may need to be executed by one process at one time to maintain the consistency otherwise the system can become inconsistent and deadlock may occur.

The operating system is responsible for the following activities in connection with Process Management

1. Scheduling processes and threads on the CPUs.
2. Creating and deleting both user and system processes.
3. Suspending and resuming processes.
4. Providing mechanisms for process synchronization.
5. Providing mechanisms for process communication.

Process Management in OS

A Program does nothing unless its instructions are executed by a CPU. A program in execution is called a process. In order to accomplish its task, process needs the computer resources.

There may exist more than one process in the system which may require the same resource at the same time. Therefore, the operating system has to manage all the processes and the resources in a convenient and efficient way.

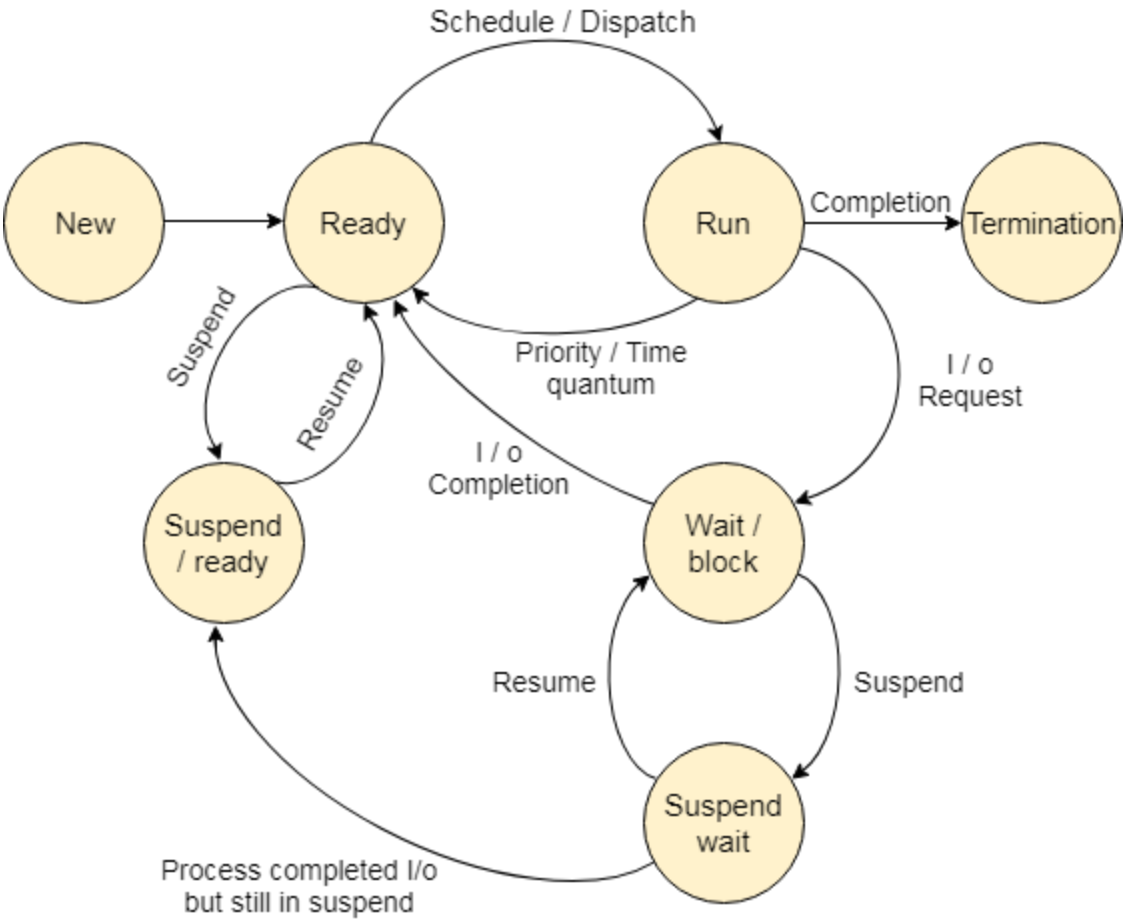
Some resources may need to be executed by one process at one time to maintain the consistency otherwise the system can become inconsistent and deadlock may occur.

The operating system is responsible for the following activities in connection with Process Management

1. Scheduling processes and threads on the CPUs.
2. Creating and deleting both user and system processes.
3. Suspending and resuming processes.
4. Providing mechanisms for process synchronization.
5. Providing mechanisms for process communication.

Process States

State Diagram



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

The names of the states are not standardized although the process may be in one of the following states during execution.

1. New

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

2. Ready

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

The processes which are ready for the execution and reside in the main memory are called ready state processes. There can be many processes present in the ready state.

3. Running

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

4. Block or wait

From the Running state, a process can make the transition to the block or wait state depending upon the scheduling algorithm or the intrinsic behavior of the process.

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

5. Completion or termination

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

6. Suspend ready

A process in the ready state, which is moved to secondary memory from the main memory due to lack of the resources (mainly primary memory) is called in the suspend ready state.

If the main memory is full and a higher priority process comes for the execution then the OS have to make the room for the process in the main memory by throwing the lower priority process out into the secondary memory. The suspend ready processes remain in the secondary memory until the main memory gets available.

7. Suspend wait

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

Operations on the Process

1. Creation

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

2. Scheduling

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

3. Execution

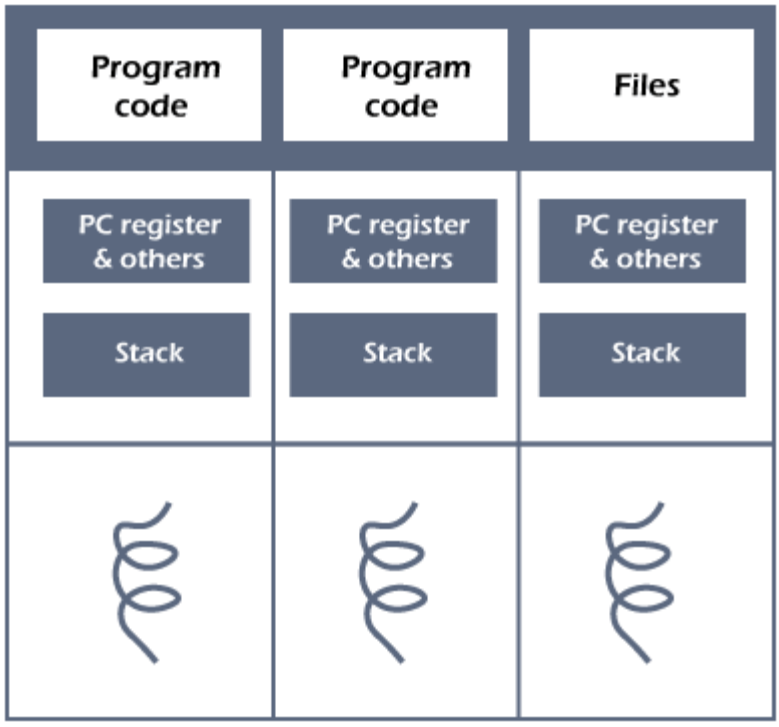
Once the process is scheduled for the execution, the processor starts executing it. Process may come to the blocked or wait state during the execution then in that case the processor starts executing the other processes.

4. Deletion/killing

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

Threads in Operating System

A thread is a single sequential flow of execution of tasks of a process so it is also known as thread of execution or thread of control. There is a way of thread execution inside the process of any operating system. Apart from this, there can be more than one thread inside a process. Each thread of the same process makes use of a separate program counter and a stack of activation records and control blocks. Thread is often referred to as a lightweight process.



Three threads of same process

The process can be split down into so many threads. **For example**, in a browser, many tabs can be viewed as threads. MS Word uses many threads - formatting text from one thread, processing input from another thread, etc.

Need of Thread:

- It takes far less time to create a new thread in an existing process than to create a new process.

- Threads can share the common data, they do not need to use Inter- Process communication.
- Context switching is faster when working with threads.
- It takes less time to terminate a thread than a process.

Types of Threads

In the operating system, there are two types of threads.

1. Kernel level thread.
2. User-level thread.

User-level thread

The operating system does not recognize the user-level thread. User threads can be easily implemented and it is implemented by the user. If a user performs a user-level thread blocking operation, the whole process is blocked. The kernel level thread does not know nothing about the user level thread. The kernel-level thread manages user-level threads as if they are single-threaded processes?examples: Java thread, POSIX threads, etc.

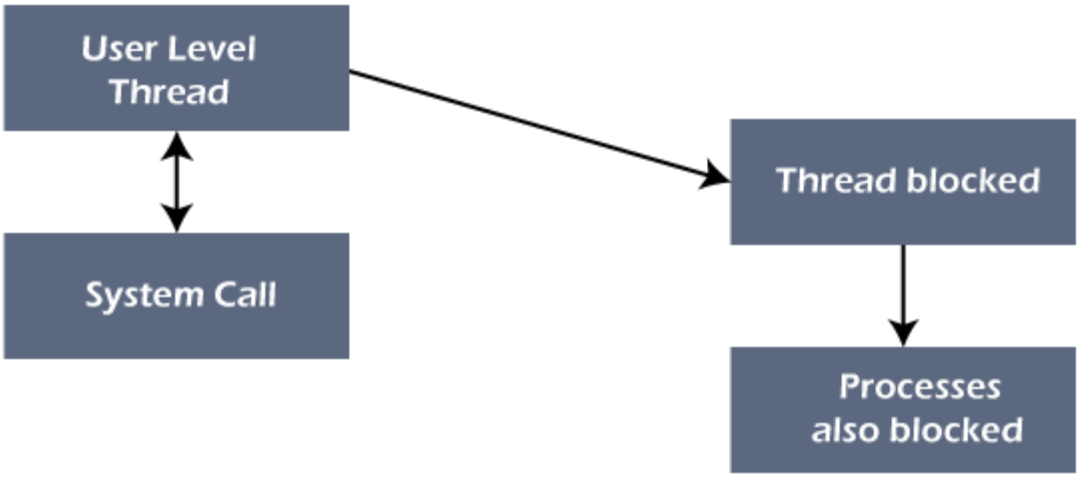
Advantages of User-level threads

1. The user threads can be easily implemented than the kernel thread.
2. User-level threads can be applied to such types of operating systems that do not support threads at the kernel-level.
3. It is faster and efficient.
4. Context switch time is shorter than the kernel-level threads.
5. It does not require modifications of the operating system.
6. User-level threads representation is very simple. The register, PC, stack, and mini thread control blocks are stored in the address space of the user-level process.
7. It is simple to create, switch, and synchronize threads without the intervention of the process.

Disadvantages of User-level threads

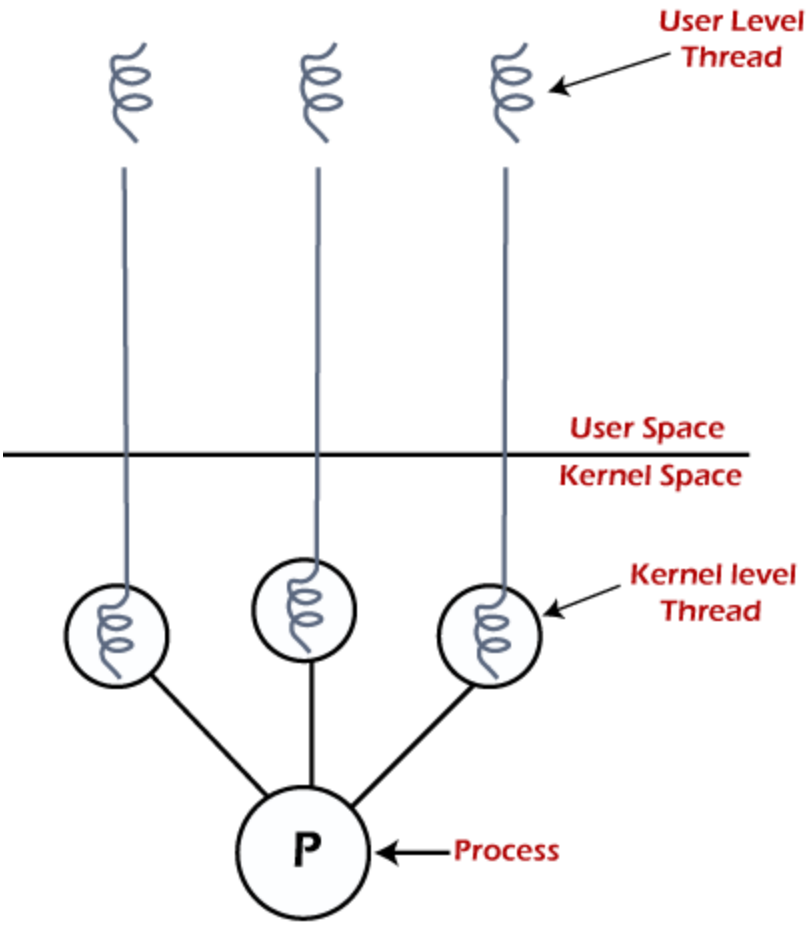
1. User-level threads lack coordination between the thread and the kernel.

2. If a thread causes a page fault, the entire process is blocked.



Kernel level thread

The kernel thread recognizes the operating system. There is a thread control block and process control block in the system for each thread and process in the kernel-level thread. The kernel-level thread is implemented by the operating system. The kernel knows about all the threads and manages them. The kernel-level thread offers a system call to create and manage the threads from user-space. The implementation of kernel threads is more difficult than the user thread. Context switch time is longer in the kernel thread. If a kernel thread performs a blocking operation, the Banky thread execution can continue. Example: Window Solaris.



Advantages of Kernel-level threads

1. The kernel-level thread is fully aware of all threads.
2. The scheduler may decide to spend more CPU time in the process of threads being large numerical.
3. The kernel-level thread is good for those applications that block the frequency.

Disadvantages of Kernel-level threads

1. The kernel thread manages and schedules all threads.
2. The implementation of kernel threads is difficult than the user thread.
3. The kernel-level thread is slower than user-level threads.

Components of Threads

Any thread has the following components.

1. Program counter
2. Register set
3. Stack space

Benefits of Threads

- **Enhanced throughput of the system:** When the process is split into many threads, and each thread is treated as a job, the number of jobs done in the unit time increases. That is why the throughput of the system also increases.
- **Effective Utilization of Multiprocessor system:** When you have more than one thread in one process, you can schedule more than one thread in more than one processor.
- **Faster context switch:** The context switching period between threads is less than the process context switching. The process context switch means more overhead for the CPU.
- **Responsiveness:** When the process is split into several threads, and when a thread completes its execution, that process can be responded to as soon as possible.
- **Communication:** Multiple-thread communication is simple because the threads share the same address space, while in process, we adopt just a few exclusive communication strategies for communication between two processes.
- **Resource sharing:** Resources can be shared between all threads within a process, such as code, data, and files. Note: The stack and register cannot be shared between threads. There is a stack and register for each thread.

CPU Scheduling

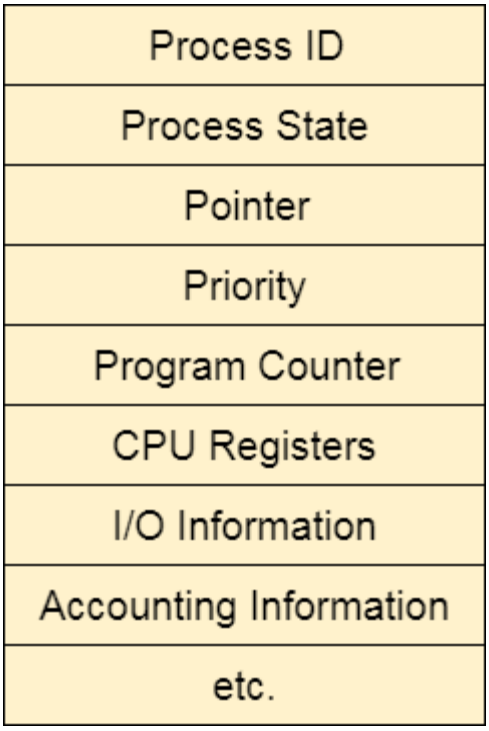
In the uniprogrammming systems like MS DOS, when a process waits for any I/O operation to be done, the CPU remains idol. This is an overhead since it wastes the time and causes the problem of starvation. However, In Multiprogramming systems, the CPU doesn't remain idle during the waiting time of the Process and it starts executing other processes. Operating System has to define which process the CPU will be given.

In Multiprogramming systems, the Operating system schedules the processes on the CPU to have the maximum utilization of it and this procedure is called **CPU scheduling**. The Operating System uses various scheduling algorithm to schedule the processes.

This is a task of the short term scheduler to schedule the CPU for the number of processes present in the Job Pool. Whenever the running process requests some IO operation then the short term scheduler saves the current context of the process (also called PCB) and changes its state from running to waiting. During the time, process is in waiting state; the Short term scheduler picks another process from the ready queue and assigns the CPU to this process. This procedure is called **context switching**.

What is saved in the Process Control Block?

The Operating system maintains a process control block during the lifetime of the process. The Process control block is deleted when the process is terminated or killed. There is the following information which is saved in the process control block and is changing with the state of the process.



Why do we need Scheduling?

In Multiprogramming, if the long term scheduler picks more I/O bound processes then most of the time, the CPU remains idol. The task of Operating system is to optimize the utilization of resources.

If most of the running processes change their state from running to waiting then there may always be a possibility of deadlock in the system. Hence to reduce this overhead, the OS needs to schedule the jobs to get the optimal utilization of CPU and to avoid the possibility to deadlock.

Scheduling Algorithms

There are various algorithms which are used by the Operating System to schedule the processes on the processor in an efficient way.

The Purpose of a Scheduling algorithm

1. Maximum CPU utilization
2. Fare allocation of CPU
3. Maximum throughput

4. Minimum turnaround time
5. Minimum waiting time
6. Minimum response time

There are the following algorithms which can be used to schedule the jobs.

1. First Come First Serve

It is the simplest algorithm to implement. The process with the minimal arrival time will get the CPU first. The lesser the arrival time, the sooner will the process gets the CPU. It is the non-preemptive type of scheduling.

2. Round Robin

In the Round Robin scheduling algorithm, the OS defines a time quantum (slice). All the processes will get executed in the cyclic way. Each of the process will get the CPU for a small amount of time (called time quantum) and then get back to the ready queue to wait for its next turn. It is a preemptive type of scheduling.

3. Shortest Job First

The job with the shortest burst time will get the CPU first. The lesser the burst time, the sooner will the process get the CPU. It is the non-preemptive type of scheduling.

4. Shortest remaining time first

It is the preemptive form of SJF. In this algorithm, the OS schedules the Job according to the remaining time of the execution.

5. Priority based scheduling

In this algorithm, the priority will be assigned to each of the processes. The higher the priority, the sooner will the process get the CPU. If the priority of the two processes is same then they will be scheduled according to their arrival time.

6. Highest Response Ratio Next

In this scheduling Algorithm, the process with highest response ratio will be scheduled next. This reduces the starvation in the system.

FCFS Scheduling

First come first serve (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU. FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

Advantages of FCFS

- Simple
- Easy
- First come, First serv

Disadvantages of FCFS

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

Example

Let's take an example of The FCFS scheduling algorithm. In the Following schedule, there are 5 processes with process ID **P0, P1, P2, P3 and P4**. P0 arrives at time 0, P1 at time 1, P2 at time 2, P3 arrives at time 3 and Process P4 arrives at time 4 in the ready queue. The processes and their respective Arrival and Burst time are given in the following table.

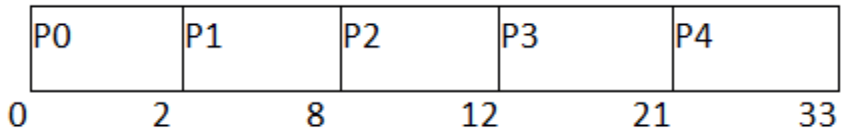
The Turnaround time and the waiting time are calculated by using the following formula.

1. Turn Around Time = Completion Time - Arrival Time
2. Waiting Time = Turnaround time - Burst Time

The average waiting Time is determined by summing the respective waiting time of all the processes and divided the sum by the total number of processes.

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

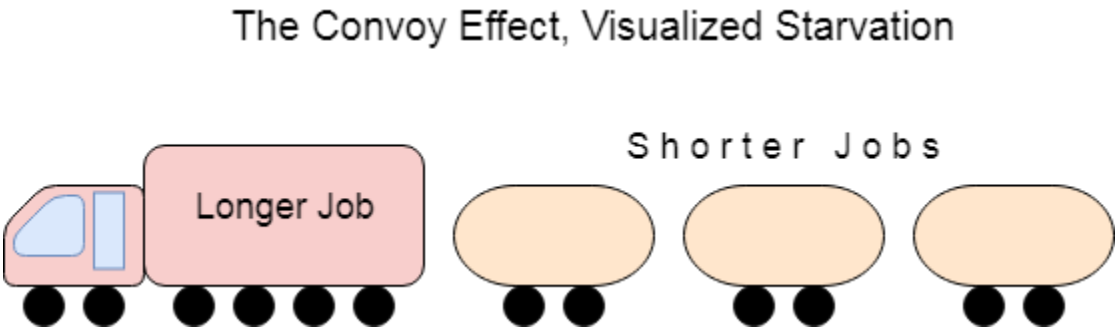
Avg Waiting Time=31/5



Convoy Effect in FCFS

FCFS may suffer from the **convoy effect** if the burst time of the first job is the highest among all. As in the real life, if a convoy is passing through the road then the other persons may get blocked until it passes completely. This can be simulated in the Operating System also.

If the CPU gets the processes of the higher burst time at the front end of the ready queue then the processes of lower burst time may get blocked which means they may never get the CPU if the job in the execution has a very high burst time. This is called **convoy effect** or **starvation**.



Example

In the Example, We have 3 processes named as **P1, P2 and P3**. The Burt Time of process P1 is highest.

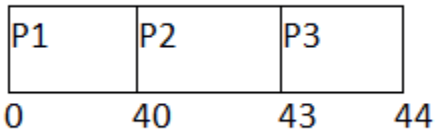
The Turnaround time and the waiting time in the following table, are calculated by the formula,

1. Turn Around Time = Completion Time - Arrival Time
2. Waiting Time = Turn Around Time - Burst Time

In the First scenario, The Process P1 arrives at the first in the queue although; the burst time of the process is the highest among all. Since, the Scheduling algorithm, we are following is FCFS hence the CPU will execute the Process P1 first.

In this schedule, the average waiting time of the system will be very high. That is because of the convoy effect. The other processes P2, P3 have to wait for their turn for 40 units of time although their burst time is very low. This schedule suffers from starvation.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	0	40	40	40	0
2	1	3	43	42	39
3	1	1	44	43	42

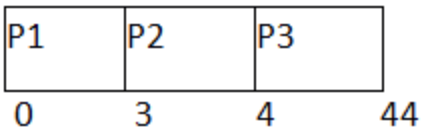


Avg waiting Time = 81/3

In the Second scenario, If Process P1 would have arrived at the last of the queue and the other processes P2 and P3 at earlier then the problem of starvation would not be there.

Following example shows the deviation in the waiting times of both the scenarios. Although the length of the schedule is same that is 44 units but the waiting time will be lesser in this schedule.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	1	40	44	43	3
2	0	3	3	3	0
3	0	1	4	4	3



Avg Waiting Time=6/3

FCFS with Overhead

In the above Examples, we are assuming that all the processes are the CPU bound processes only. We were also neglecting the context switching time.

However if the time taken by the scheduler in context switching is considered then the average waiting time of the system will be increased which also affects the efficiency of the system.

Context Switching is always an overhead. The Following Example describeshow the efficiency will be affected if the context switching time is considered in the system.

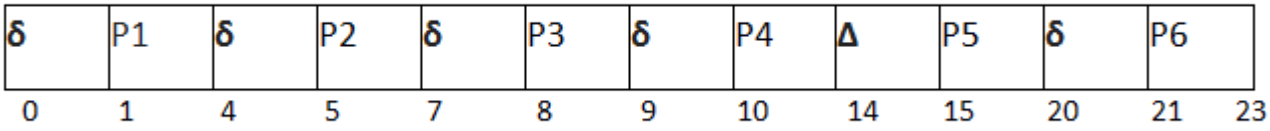
Example

In the following Example, we are considering five processes P1, P2, P3, P4, P5 and P6. Their arrival time and Burst time are given below.

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

If the context switching time of the system is 1 unit then the Gantt chart of the system will be prepared as follows.

Given **δ=1 unit;**



The system will take extra 1 unit of time (overhead) after the execution of every process to schedule the next process.

1. Inefficiency= (6/23) X 100 %
2. Efficiencyи = (1-6/23) X 100 %

Shortest Job First (SJF) Scheduling

Till now, we were scheduling the processes according to their arrival time (in FCFS scheduling). However, SJF scheduling algorithm, schedules the processes according to their burst time.

In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.

However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.

Advantages of SJF

1. Maximum throughput
2. Minimum average waiting and turnaround time

Disadvantages of SJF

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

There are different techniques available by which, the CPU burst time of the process can be determined. We will discuss them later in detail.

Example

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

PID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Ti
1	1	7	8	7	0
2	3	3	13	10	7

3	6	2	10	4	2
4	7	10	31	24	14
5	9	8	21	12	4

Since, No Process arrives at time 0 hence; there will be an empty slot in the **Gantt chart** from time 0 to 1 (the time at which the first process arrives).

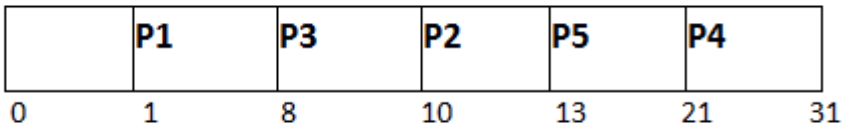
According to the algorithm, the OS schedules the process which is having the lowest burst time among the available processes in the ready queue.

Till now, we have only one process in the ready queue hence the scheduler will schedule this to the processor no matter what is its burst time.

This will be executed till 8 units of time. Till then we have three more processes arrived in the ready queue hence the scheduler will choose the process with the lowest burst time.

Among the processes given in the table, P3 will be executed next since it is having the lowest burst time among all the available processes.

So that's how the procedure will go on in **shortest job first (SJF)** scheduling algorithm.



Avg Waiting Time = 27/5

Prediction of CPU Burst Time for a process in SJF

The SJF algorithm is one of the best scheduling algorithms since it provides the maximum throughput and minimal waiting time but the problem with the algorithm is, the CPU burst time can't be known in advance.

We can approximate the CPU burst time for a process. There are various techniques which can be used to assume the CPU Burst time for a process. Our Assumption needs to be accurate in order to utilize the algorithm optimally.

There are the following techniques used for the assumption of CPU burst time for a process.

1. Static Techniques

Process Size

We can predict the Burst Time of the process from its size. If we have two processes **T_OLD** and **T_New** and the actual burst time of the old process is known as **20 secs** and the size of the process is **20 KB**. We know that the size of **P_NEW is 21 KB**. Then the probability of **P_New** having the similar burst time as **20 secs** is maximum.

1. If, P_OLD → 20 KB
2. P_New → 21 KB
3. BT(P_OLD) → 20 Secs
4. Then,
5. BT(P_New) → 20 secs

Hence, in this technique, we actually predict the burst time of a new process according to the burst time of an old process of similar size as of new process.

Process Type

We can also predict the burst time of the process according to its type. A Process can be of various types defined as follows.

- **OS Process**

A Process can be an Operating system process like schedulers, compilers, program managers and many more system processes. Their burst time is generally lower for example, 3 to 5 units of time.

- **User Process**

  The Processes initiated by the users are called user processes. There can be three types of processes as follows.

- **Interactive Process**

  The Interactive processes are the one which interact with the user time to time or Execution of which totally depends upon the User inputs for example various games are such processes. There burst time needs to be lower since they don't need CPU for a large amount of time, they mainly depend upon the user's interactivity with the process hence they are mainly IO bound processes.

- **Foreground process**

  Foreground processes are the processes which are used by the user to perform their needs such as MS office, Editors, utility software etc. These types of processes have a bit higher burst time since they are a perfect mix of CPU and IO bound processes.

- **Background process**

  Background processes supports the execution of other processes. They work in hidden mode. For example, key logger is the process which records the keys pressed by the user and activities of the user on the system. They are mainly CPU bound processes and needs CPU for a higher amount of time.

2. Dynamic Techniques

Simple Averaging

In simple averaging, there are given list of n processes P(i).....P(n). Let T(i) denotes the burst time of the process P(i). Let τ(n) denotes the predicted burst time of Pth process. Then according to the simple averaging, the predicted burst time of process n+1 will be calculated as,

$$\tau(n+1) = (1/n) \sum T(i)$$

Where, $0<=i<=n$ and $\sum T(i)$ is the summation of actual burst time of all the processes available till now.

Exponential Averaging or Aging

Let, Tn be the actual burst time of nth process.τ(n) be the predicted burst time for nth process then the CPU burst time for the next process (n+1) will be calculated as,

$$\tau(n+1) = \alpha. Tn + (1-\alpha) . \tau(n)$$

Where, α is the smoothing. Its value lies between 0 and 1.

Shortest Remaining Time First (SRTF) Scheduling Algorithm

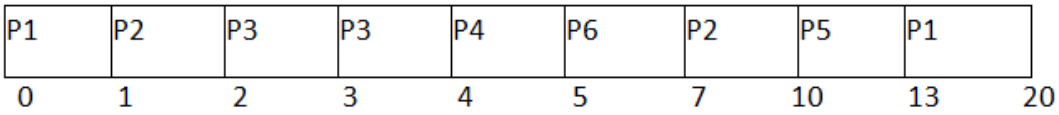
This Algorithm is the **preemptive version** of **SJF scheduling**. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.

Once all the processes are available in the **ready queue**, No preemption will be done and the algorithm will work as **SJF scheduling**. The context of the process is saved in the **Process Control Block** when the process is removed from the execution and the next process is scheduled. This PCB is accessed on the **next execution** of this process.

Example

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

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



Avg Waiting Time = 24/6

The Gantt chart is prepared according to the arrival and burst time given in the table.

1. Since, at time 0, the only available process is P1 with CPU burst time 8. This is the only available process in the list therefore it is scheduled.
2. The next process arrives at time unit 1. Since the algorithm we are using is SRTF which is a preemptive one, the current execution is stopped and the scheduler checks for the process with the least burst time.
   Till now, there are two processes available in the ready queue. The OS has executed P1 for one unit of time till now; the remaining burst time of P1 is 7 units. The burst time of Process P2 is 4 units. Hence Process P2 is scheduled on the CPU according to the algorithm.
3. The next process P3 arrives at time unit 2. At this time, the execution of process P3 is stopped and the process with the least remaining burst time is searched. Since the process P3 has 2 unit of burst time hence it will be given priority over others.
4. The Next Process P4 arrives at time unit 3. At this arrival, the scheduler will stop the execution of P4 and check which process is having least burst time among the available processes (P1, P2, P3 and P4). P1 and P2 are having the remaining burst time 7 units and 3 units respectively.

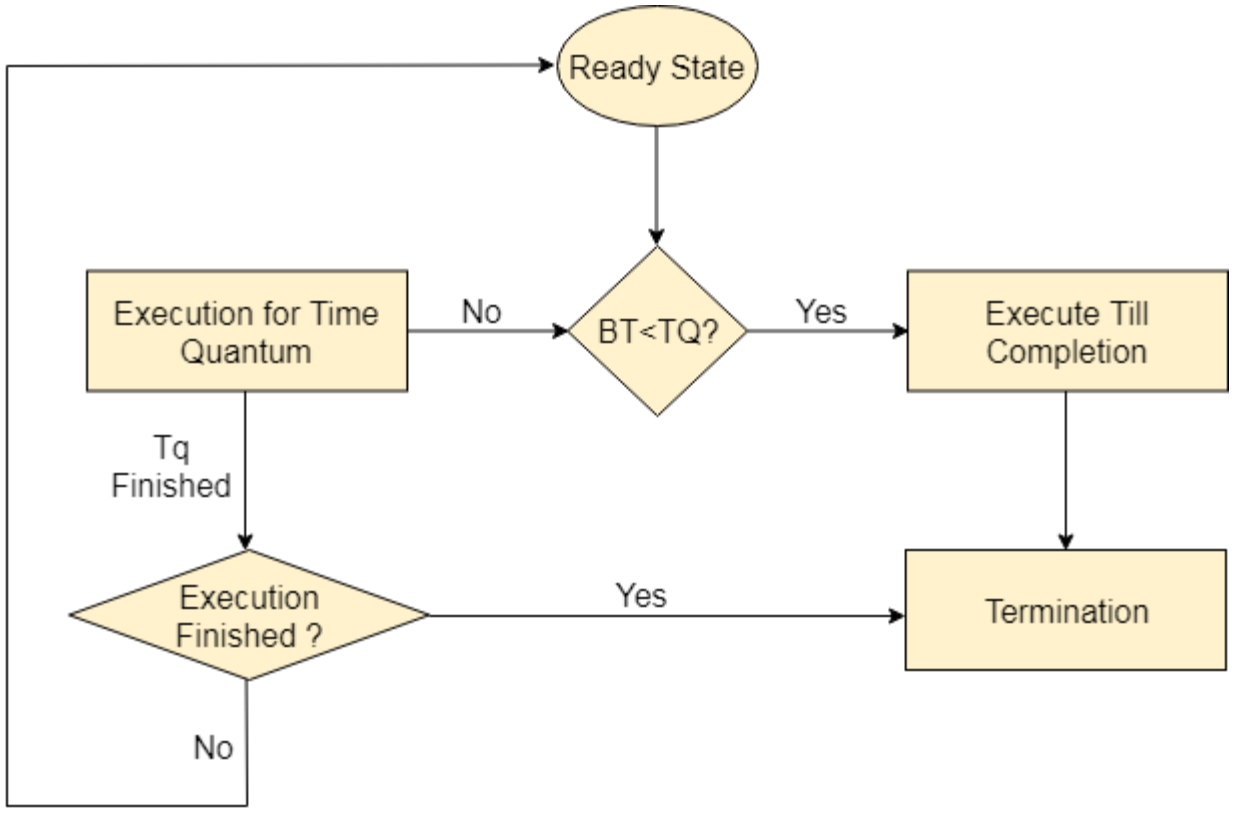
   P3 and P4 are having the remaining burst time 1 unit each. Since, both are equal hence the scheduling will be done according to their arrival time. P3 arrives earlier than P4 and therefore it will be scheduled again.
5. The Next Process P5 arrives at time unit 4. Till this time, the Process P3 has completed its execution and it is no more in the list. The scheduler will compare the remaining burst time of all the available processes. Since the burst time of process P4 is 1 which is least among all hence this will be scheduled.

6. The Next Process P6 arrives at time unit 5, till this time, the Process P4 has completed its execution. We have 4 available processes till now, that are P1 (7), P2 (3), P5 (3) and P6 (2). The Burst time of P6 is the least among all hence P6 is scheduled. Since, now, all the processes are available hence the algorithm will now work same as SJF. P6 will be executed till its completion and then the process with the least remaining time will be scheduled.

Once all the processes arrive, No preemption is done and the algorithm will work as SJF.

Round Robin Scheduling Algorithm

Round Robin scheduling algorithm is one of the most popular scheduling algorithm which can actually be implemented in most of the operating systems. This is the **preemptive version** of first come first serve scheduling. The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a **cyclic way**. A certain time slice is defined in the system which is called time **quantum**. Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will **terminate** else the process will go back to the **ready queue** and waits for the next turn to complete the execution.



Advantages

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

Disadvantages

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

RR Scheduling Example

In the following example, there are six processes named as P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below in the table. The time quantum of the system is 4 units.

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

According to the algorithm, we have to maintain the ready queue and the Gantt chart. The structure of both the data structures will be changed after every scheduling.

Ready Queue:

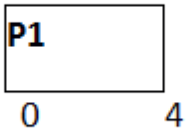
Initially, at time 0, process P1 arrives which will be scheduled for the time slice 4 units. Hence in the ready queue, there will be only one process P1 at starting with CPU burst time 5 units.

P1
5

GANTT chart

The P1 will be executed for 4 units first.

Difference between structure and union in HindiKeep Watching



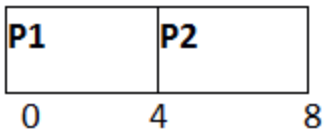
Ready Queue

Meanwhile the execution of P1, four more processes P2, P3, P4 and P5 arrives in the ready queue. P1 has not completed yet, it needs another 1 unit of time hence it will also be added back to the ready queue.

P2	P3	P4	P5	P1
6	3	1	5	1

GANTT chart

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



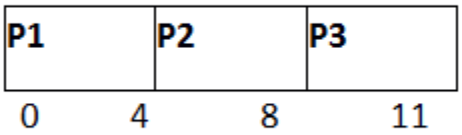
Ready Queue

During the execution of P2, one more process P6 is arrived in the ready queue. Since P2 has not completed yet hence, P2 will also be added back to the ready queue with the remaining burst time 2 units.

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

GANTT chart

After P1 and P2, P3 will get executed for 3 units of time since its CPU burst time is only 3 seconds.



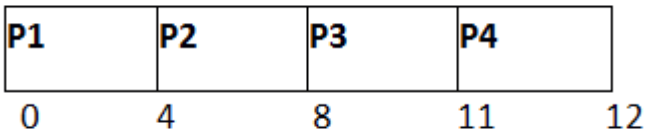
Ready Queue

Since P3 has been completed, hence it will be terminated and not be added to the ready queue. The next process will be executed is P4.

P4	P5	P1	P6	P2
1	5	1	4	2

GANTT chart

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



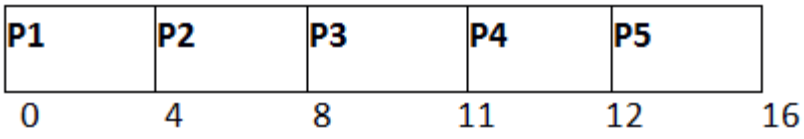
Ready Queue

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

P5	P1	P6	P2
5	1	4	2

GANTT chart

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



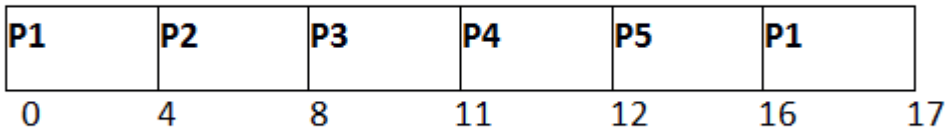
Ready Queue

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

P1	P6	P2	P5
1	4	2	1

GANTT Chart

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



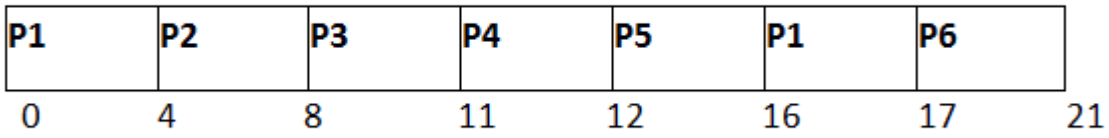
Ready Queue

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

P6	P2	P5
4	2	1

GANTT chart

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



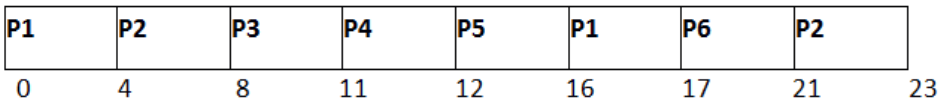
Ready Queue

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

P2	P5
2	1

GANTT Chart

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



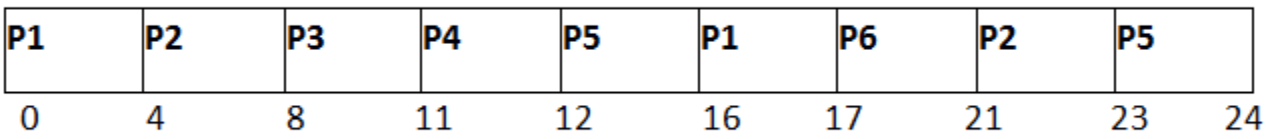
Ready Queue

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

P5
1

GANTT chart

P5 will get executed till completion.



The completion time, Turnaround time and waiting time will be calculated as shown in the table below.

As, we know,

1. Turn Around Time = Completion Time - Arrival Time
2. Waiting Time = Turn Around Time - Burst Time

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	0	5	17	17	12
2	1	6	23	22	16
3	2	3	11	9	6
4	3	1	12	9	8
5	4	5	24	20	15
6	6	4	21	15	11

Avg Waiting Time = (12+16+6+8+15+11)/6 = 76/6 units

Highest Response Ratio Next (HRRN) Scheduling

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

Response Ratio is calculated by the given formula.

1. Response Ratio = (W+S)/S

Where,

1. W → Waiting Time
2. S → Service Time or Burst Time

If we look at the formula, we will notice that the job with the shorter burst time will be given priority but it is also including an extra factor called waiting time. Since,

1. HRNN α W
2. HRNN α (1/S)

Hence,

1. This algorithm not only favors shorter job but it also concern the waiting time of the longer jobs.
2. Its mode is non preemptive hence context switching is minimal in this algorithm.

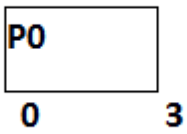
HRNN Example

In the following example, there are 5 processes given. Their arrival time and Burst Time are given in the table.

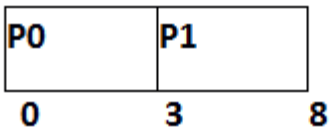
Process ID	Arrival Time	Burst Time

0	0	3
1	2	5
2	4	4
3	6	1
4	8	2

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



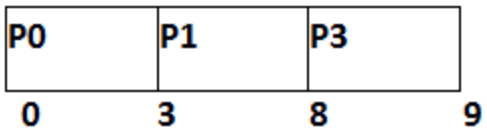
P0 is executed for 3 units, meanwhile, only one process P1 arrives at time 3. This will get scheduled immediately since the OS doesn't have a choice.



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

1. RR (P2) = ((8-4) +4)/4 = 2
2. RR (P3) = (2+1)/1 = 3
3. RR (P4) = (0+2)/2 = 1

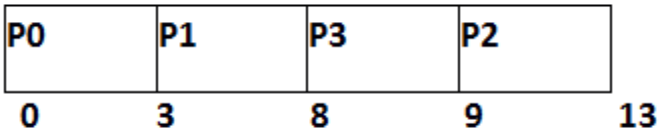
Since, the Response ratio of P3 is higher hence P3 will be scheduled first.



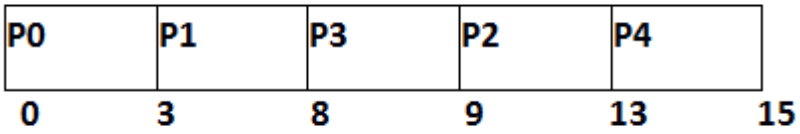
P3 is scheduled for 1 unit. The next available processes are P2 and P4. Let's calculate their Response ratio.

1. RR (P2) = (5+4)/4 = 2.25
2. RR (P4) = (1+2)/2 = 1.5

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



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



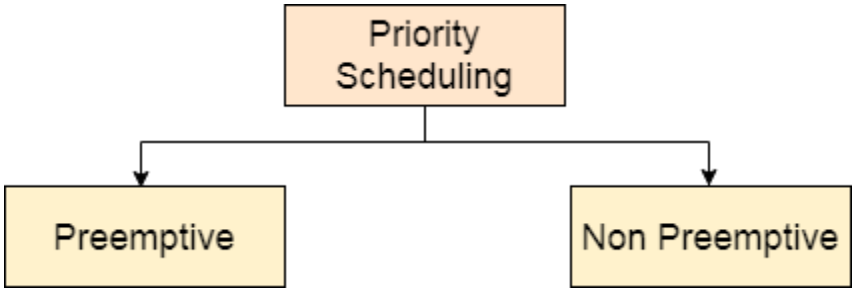
Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
0	0	3	3	3	0

1	2	5	8	6	1
2	4	4	13	9	5
3	6	1	9	3	2
4	8	2	15	7	5

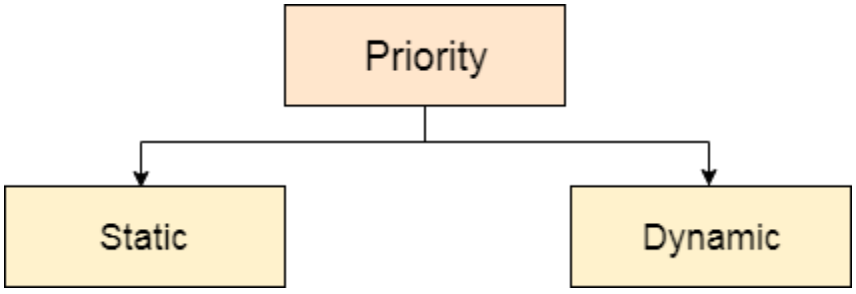
Average Waiting Time = 13/5

Priority Scheduling

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



The priority number assigned to each of the process may or may not vary. If the priority number doesn't change itself throughout the process, it is called **static priority**, while if it keeps changing itself at the regular intervals, it is called **dynamic priority**.



Next →← Prev

Non Preemptive Priority Scheduling

In the Non Preemptive Priority scheduling, The Processes are scheduled according to the priority number assigned to them. Once the process gets scheduled, it will run till the completion. Generally, the lower the priority number, the higher is the priority of the

process. The people might get confused with the priority numbers, hence in the GATE, there clearly mention which one is the highest priority and which one is the lowest one.

Example

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

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

We can prepare the Gantt chart according to the Non Preemptive priority scheduling.

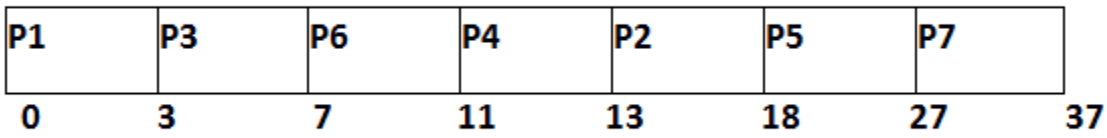
The Process P1 arrives at time 0 with the burst time of 3 units and the priority number 2. Since No other process has arrived till now hence the OS will schedule it immediately.

Meanwhile the execution of P1, two more Processes P2 and P3 are arrived. Since the priority of P3 is 3 hence the CPU will execute P3 over P2.

Meanwhile the execution of P3, All the processes get available in the ready queue. The Process with the lowest priority number will be given the priority. Since P6 has priority number assigned as 4 hence it will be executed just after P3.

After P6, P4 has the least priority number among the available processes; it will get executed for the whole burst time.

Since all the jobs are available in the ready queue hence All the Jobs will get executed according to their priorities. If two jobs have similar priority number assigned to them, the one with the least arrival time will be executed.



From the GANTT Chart prepared, we can determine the completion time of every process. The turnaround time, waiting time and response time will be determined.

1. Turn Around Time = Completion Time - Arrival Time
2. Waiting Time = Turn Around Time - Burst Time

Process Id	Priority	Arrival Time	Burst Time	Completion Time	Turn around Time	Waiting Time	Response Time
1	2	0	3	3	3	0	0
2	6	2	5	18	16	11	13
3	3	1	4	7	6	2	3
4	5	4	2	13	9	7	11
5	7	6	9	27	21	12	18
6	4	5	4	11	6	2	7
7	10	7	10	37	30	18	27

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