Unit-2 CPU Scheduler and Scheduling

What is CPU Scheduling?

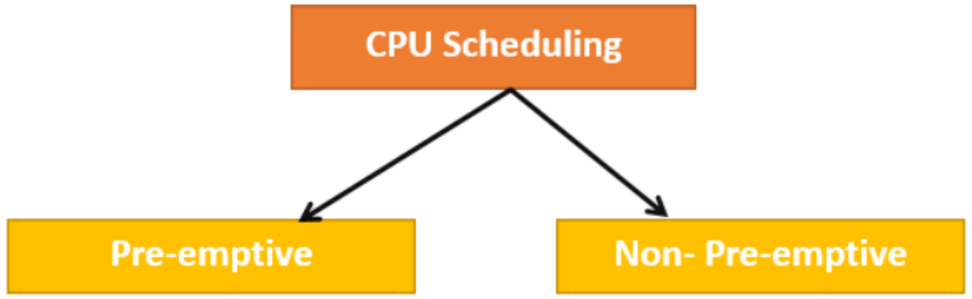
CPU Scheduling is a process of determining which process will own CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution. The selection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

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.

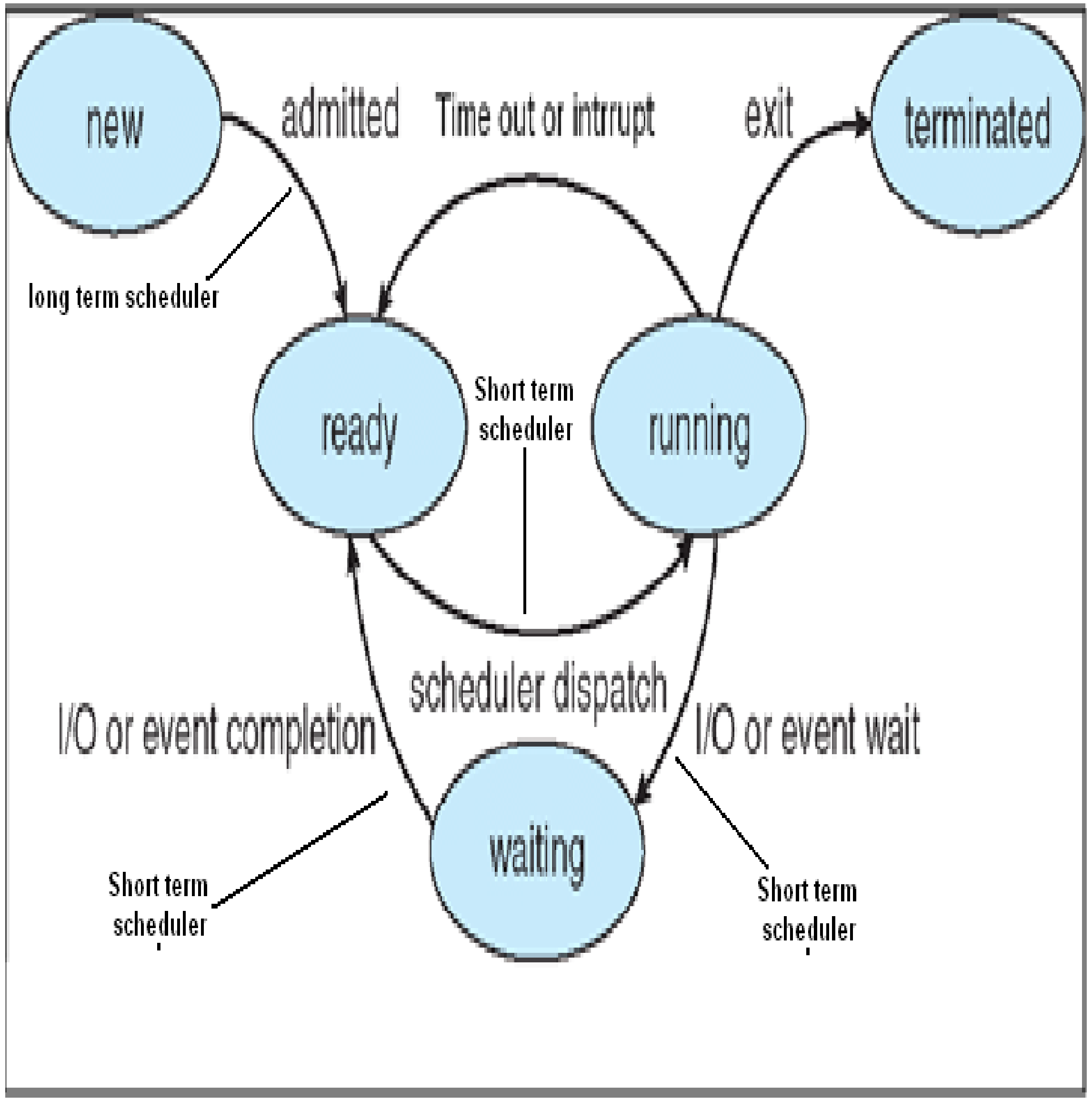
All other scheduling are preemptive.

Scheduler

- Scheduler makes a choice of which process run next whenever more than one processes are simultaneously in the ready queue.
- A process migrates between the various scheduling queues throughout its lifetime. Scheduler selects processes from these queues.
- There are three types of schedulers depending upon the scheduling queue and operating system:

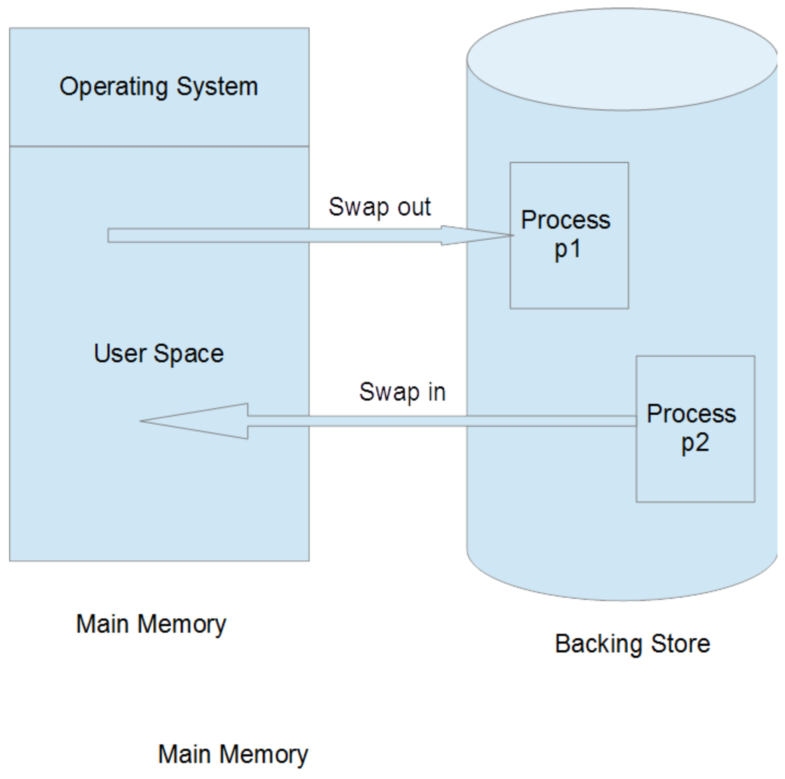
- **Long term scheduler/Job Scheduler**
- **Medium term scheduler**
- **Short term scheduler/Process Scheduler**

Scheduler



- Long term scheduler/Job Scheduler
  - It is a first level scheduler. It selects processes from new state which are in main memory to ready state. It executes less frequently compared to other schedulers
- Medium term scheduler
  - It is a second level scheduler. It can be found in operating system where there is a support for "**Swapping**".
- Short term scheduler/Process Scheduler
  - It is third level scheduler. It selects process, which is ready for the execution, from ready queue to run queue. It executes much frequently than other scheduler

Swapping



“**Swapping**”.

- Swapping is process of swap-in and swap-out.

- **Swap-out:** Whenever there is a limited memory in main memory, then it will be swapped out from main memory to backing store (standard hard disk), that is known as swap out process. So, process is temporary removed from the main memory.
- **Swap-in:** Whenever a process is swap in from backing store to main memory. It is known as swap in.

Operations on Process:

- **Process Creation**
- Whenever we are working with multiple programs, several processes are created and deleted dynamically. They can be a user process or system process.
- A running process may create several new processes via create process system call. The creating process is called a parent process, whereas the new processes are called the children of the process.
- Process needs certain resources to accomplish its task. When process creates a sub process, there are two possibilities:
- Child process may obtain its resources directly from the parent process.
- Parent process may share their resources among their child processes.
- When a process creates a new process, two possibilities may exist in term of execution:
  - The parent process and child process execute concurrently.
  - The parent process waits until some or all of its children have terminated.
- When a process creates a new process, two possibilities may exist in term of address space:
  - The parent process and child process may share the same address space.
  - The Parent process and child process may have their individual address space.
- Each process is identified by its process identifier, which is unique integer.
- New process is created by the **fork** system call. If parent process and child process share the address space, then they can communicate easily. After fork system call, newly created (child) process has process id 0 and parent process has always process id nonzero

Process Termination

- Whenever process finish its execution it will be terminated by operating system. Process may finish its execution either normally or abnormally. Process will terminated by the **exit** system call. Whenever process terminates, all of its resources will be deallocated.
- When child process terminates, it may return data to its parent process.
- A Parent process may terminate the execution of its children for different reasons:
- The task assigned to the child process is no longer required.
- In some cases, suppose parent process terminates first, then OS will not allow a child to continue their execution. OS will explicitly terminate all its child processes.

Important CPU scheduling Terminologies

- **Burst Time/Execution Time:** It is a time required by the process to complete execution. It is also called running time.
- **Arrival Time:** when a process enters in a ready state
- **Finish Time:** when process complete and exit from a system
- **Multiprogramming:** A number of programs which can be present in memory at the same time.
- **Jobs:** It is a type of program without any kind of user interaction.
- **User:** It is a kind of program having user interaction.
- **Process:** It is the reference that is used for both job and user.
- **CPU/IO burst cycle:** Characterizes process execution, which alternates between CPU and I/O activity. CPU times are usually shorter than the time of I/O.

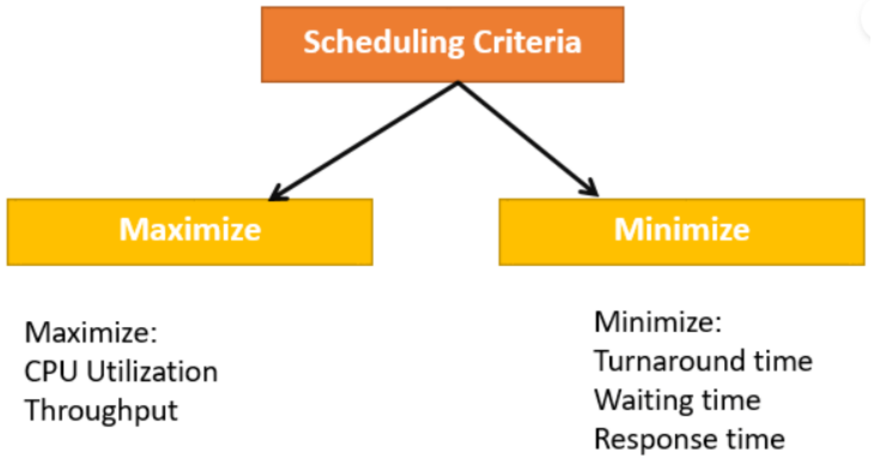
CPU Scheduling Criteria

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

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. 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.



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.

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.

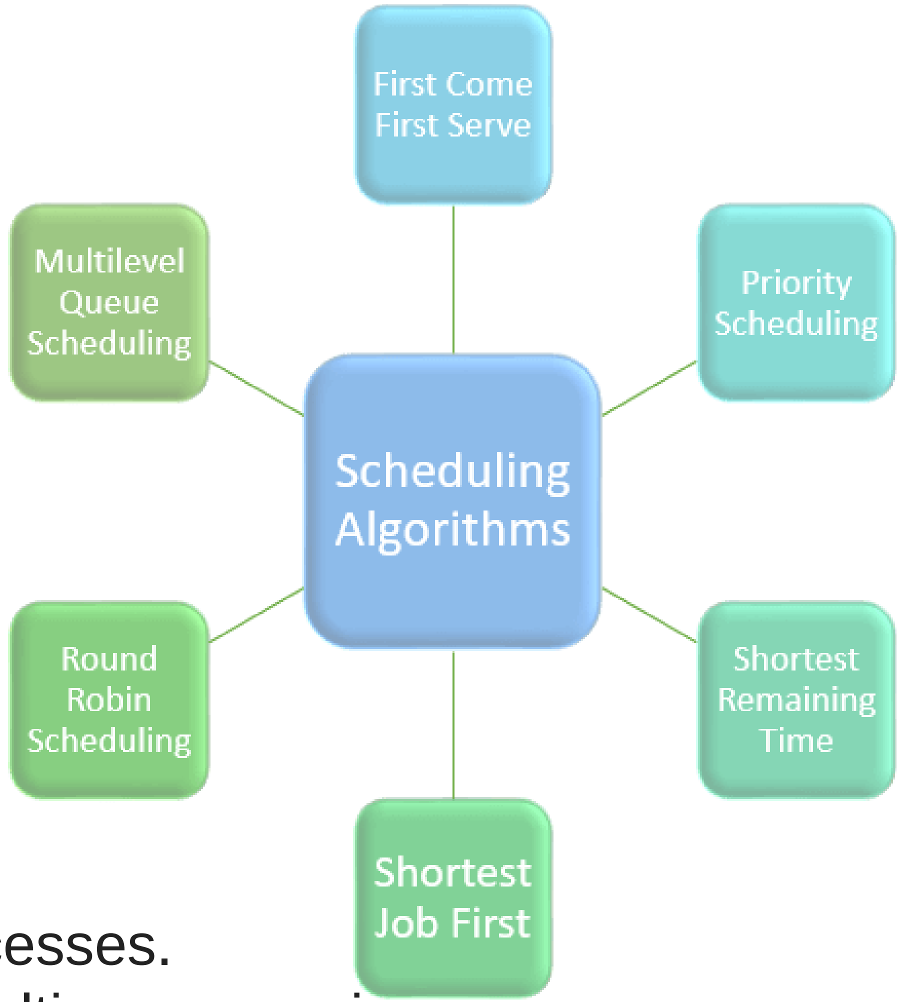
What is Dispatcher?
It is a module that provides control of the CPU to the process. The Dispatcher should be fast so that it can run on every context switch. Dispatch latency is the amount of time needed by the CPU scheduler to stop one process and start another.

Functions performed by Dispatcher:
• Context Switching
• Switching to user mode
• Moving to the correct location in the newly loaded program.

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



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.

Ref. Link: https://youtu.be/exlaEOVRWQM

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.

Ref. Link: https://youtu.be/iiBij98FtHg

Characteristics of First Come First Serve 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.

FCFS Advantage and Disadvantage:

Advantages:

- FCFS algorithm doesn't include any complex logic, it just puts the process requests in a queue and executes it one by one.
- Hence, FCFS is pretty simple and easy to implement.
- Eventually, every process will get a chance to run, so starvation doesn't occur.

Disadvantages:

- There is no option for pre-emption of a process. If a process is started, then CPU executes the process until it ends.
- Because there is no pre-emption, if a process executes for a long time, the processes in the back of the queue will have to wait for a long time before they get a chance to be executed.

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.

Ref. Link: https://youtu.be/Nr4nANJjddg

Priority Based Scheduling

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.

Ref. Link: https://youtu.be/cNdEQKw4apM

Priority based Scheduling

Advantages:
• The priority of a process can be selected based on memory requirement, time requirement or user preference. For example, a high end game will have better graphics, that means the process which updates the screen in a game will have higher priority so as to achieve better graphics performance.

Disadvantages:
• A second scheduling algorithm is required to schedule the processes which have same priority.
• In preemptive priority scheduling, a higher priority process can execute ahead of an already executing lower priority process. If lower priority process keeps waiting for higher priority processes, starvation occurs.

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.

Ref. Link: https://youtu.be/3N2t9_6Co3U

Tutorial: https://youtu.be/aWlQYllBZDs

Round Robin (RR)

Advantages:

• Each process is served by the CPU for a fixed time quantum, so all processes are given the same priority.

• Starvation doesn't occur because for each round robin cycle, every process is given a fixed time to execute. No process is left behind.

Disadvantages:

• The throughput in RR largely depends on the choice of the length of the time quantum. If time quantum is longer than needed, it tends to exhibit the same behavior as FCFS.

• If time quantum is shorter than needed, the number of times that CPU switches from one process to another process, increases. This leads to decrease in CPU efficiency.

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.

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.

Ref. Link: https://youtu.be/t0g9b3SJECg

Shortest Job First (SJF)

Advantages:

• According to the definition, short processes are executed first and then followed by longer processes.

• The throughput is increased because more processes can be executed in less amount of time.

Disadvantages:

• The time taken by a process must be known by the CPU beforehand, which is not possible.

• Longer processes will have more waiting time, eventually they'll suffer starvation.

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.

Ref. Link:

Usage of Scheduling Algorithms in Different Situations

Every scheduling algorithm has a type of a situation where it is the best choice. Let's look at different such situations:

Situation 1:

The incoming processes are short and there is no need for the processes to execute in a specific order.

In this case, FCFS works best when compared to SJF and RR because the processes are short which means that no process will wait for a longer time. When each process is executed one by one, every process will be executed eventually.

Situation 2:

The processes are a mix of long and short processes and the task will only be completed if all the processes are executed successfully in a given time.

Round Robin scheduling works efficiently here because it does not cause starvation and also gives equal time quantum for each process.

Situation 3:

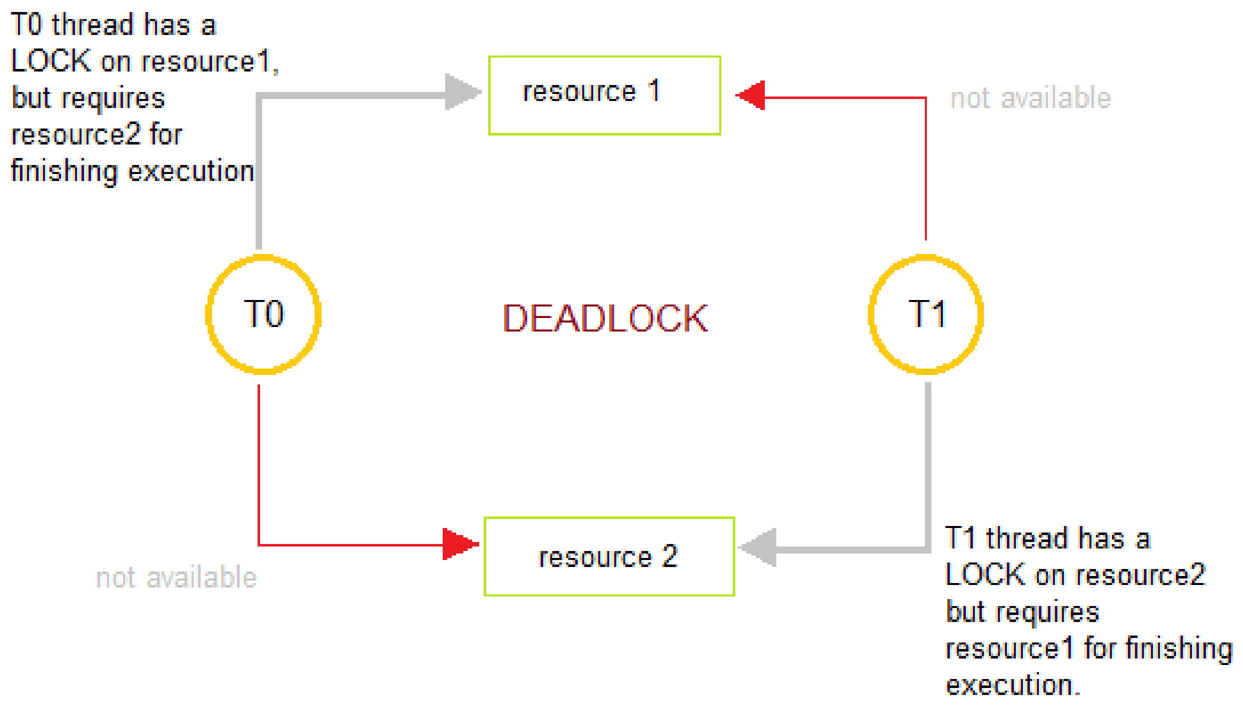
The processes are a mix of user based and kernel based processes.

Priority based scheduling works efficiently in this case because generally kernel based processes have higher priority when compared to user based processes.

For example, the scheduler itself is a kernel based process, it should run first so that it can schedule other processes

Introduction to Deadlocks in Operating System

Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.


T0 thread has a LOCK on resource1, but requires resource2 for finishing execution
resource 1
not available
T0
DEADLOCK
T1
not available
resource 2
T1 thread has a LOCK on resource2 but requires resource1 for finishing execution.

In the above figure, process T0 has resource1, it requires resource2 in order to finish its execution.

Similarly, process T1 has resource2 and it also needs to acquire resource1 to finish its execution.

In this way, T0 and T1 are in a deadlock because each of them needs the resource of others to complete their execution but neither of them is willing to give up their resources.

In General, a process must request a resource before using it and it must release the resource after using it. And any process can request as many resources as it requires in order to complete its designated task. And there is a condition that the number of resources requested may not exceed the total number of resources available in the system.

Basically in the Normal mode of Operation utilization of resources by a process is in the following sequence:

1.Request: Firstly, the process requests the resource. In a case, if the request cannot be granted immediately(e.g: resource is being used by any other process), then the requesting process must wait until it can acquire the resource.

2.Use: The Process can operate on the resource (e.g: if the resource is a printer then in that case process can print on the printer).

3.Release: The Process releases the resource.

Necessary Conditions
The deadlock situation can only arise if all the following four conditions hold simultaneously:

1. Mutual Exclusion

According to this condition, at least one resource should be non-shareable (non-shareable resources are those that can be used by one process at a time.)

This condition must hold for non-sharable resources. For example, a printer cannot be simultaneously shared by several processes.

In contrast, Sharable resources do not require mutually exclusive access and thus cannot be involved in a deadlock.

A good example of a sharable resource is **Read-only files** because if several processes attempt to open a read-only file at the same time, then they can be granted simultaneous access to the file.

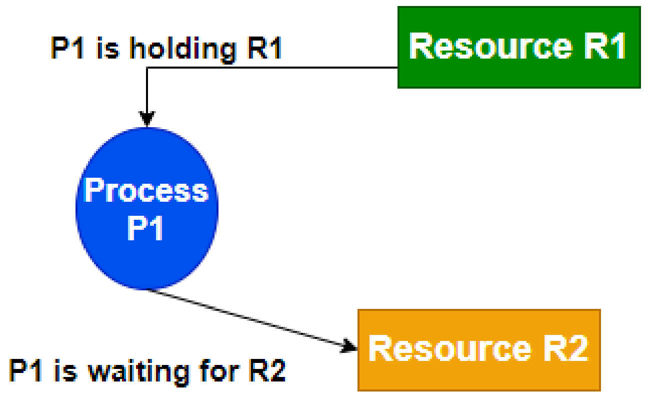
Necessary Conditions

The deadlock situation can only arise if all the following four conditions hold simultaneously:

2. Hold and Wait

According to this condition, A process is holding at-least one resource and is waiting for additional resources.

Hold and Wait condition occurs when a process holds a resource and is also waiting for some other resource in order to complete its execution. Thus if we did not want the occurrence of this condition then we must guarantee that when a process requests a resource, it does not hold any other resource.



Hold and wait condition

2. Hold and Wait (Continue…)

There are some protocols that can be used in order to ensure that the Hold and Wait condition never occurs:

- According to the first protocol; Each process must request and gets all its resources before the beginning of its execution.
- The second protocol allows a process to request resources only when it does not occupy any resource.

Let us illustrate the difference between these two protocols:

We will consider a process that mainly copies data from a DVD drive to a file on disk, sorts the file, and then prints the results to a printer. If all the resources must be requested at the beginning of the process **according to the first protocol**, then the process requests the DVD drive, disk file, and printer initially. It will hold the printer during its entire execution, even though the printer is needed only at the end.

While the second method allows the process to request initially only the DVD drive and disk file. It copies the data from the DVD drive to the disk and then releases both the DVD drive and the disk file. The process must then again request the disk file and printer. After copying the disk file to the printer, the process releases these two resources as well and then terminates.

3. NO preemption: Resources cannot be taken from the process because resources can be released only voluntarily by the process holding them.

The third necessary condition for deadlocks is that there should be no preemption of resources that have already been allocated. In order to ensure that this condition does not hold the following protocols can be used :

• According to the First Protocol: "If a process that is already holding some resources requests another resource and if the requested resources cannot be allocated to it, then it must release all the resources currently allocated to it.“
• According to the Second Protocol: "When a process requests some resources, if they are available, then allocate them. If in case the requested resource is not available then we will check whether it is being used or is allocated to some other process waiting for other resources. If that resource is not being used, then the operating system preempts it from the waiting process and allocate it to the requesting process. And if that resource is being used, then the requesting process must wait".

The second protocol can be applied to those resources whose state can be easily saved and restored later for example CPU registers and memory space, and cannot be applied to resources like printers and tape drivers.

4. Circular wait
In this condition, the set of processes are waiting for each other in the circular form.

Assign a priority number to each resource. There will be a condition that any process cannot request for a lesser priority resource. This method ensures that not a single process can request a resource that is being utilized by any other process and due to which no cycle will be formed.

Example: Assume that R5 resource is allocated to P1, if next time P1 asks for R4, R3 that are lesser than R5; then such request will not be granted. Only the request for resources that are more than R5 will be granted.

Deadlock conditions can be avoided with the help of a number of methods. Let us take a look at some of the methods.

Methods For Handling Deadlocks

Methods that are used in order to handle the problem of deadlocks are as follows:

1. Ignoring the Deadlock

According to this method, it is assumed that deadlock would never occur. This approach is used by many operating systems where they assume that deadlock will never occur which means operating systems simply ignores the deadlock. This approach can be beneficial for those systems that are only used for browsing and for normal tasks. Thus ignoring the deadlock method can be useful in many cases but it is not perfect in order to remove the deadlock from the operating system.

2.Deadlock Prevention

As we have discussed in the above section, that all four conditions: Mutual Exclusion, Hold and Wait, No preemption, and circular wait if held by a system then causes deadlock to occur. The main aim of the deadlock prevention method is to violate any one condition among the four; because if any of one condition is violated then the problem of deadlock will never occur. As the idea behind this method is simple but the difficulty can occur during the physical implementation of this method in the system.

3.Avoiding the Deadlock

This method is used by the operating system in order to check whether the system is in a safe state or in an unsafe state. This method checks every step performed by the operating system. Any process continues its execution until the system is in a safe state. Once the system enters into an unsafe state, the operating system has to take a step back.

Basically, with the help of this method, the operating system keeps an eye on each allocation, and make sure that allocation does not cause any deadlock in the system.

4.Deadlock detection and recovery

With this method, the deadlock is detected first by using some algorithms of the resource-allocation graph. This graph is mainly used to represent the allocations of various resources to different processes. After the detection of deadlock, a number of methods can be used in order to recover from that deadlock.

One way is **preemption** by the help of which a resource held by one process is provided to another process.

The second way is to **roll back**, as the operating system keeps a record of the process state and it can easily make a process roll back to its previous state due to which deadlock situation can be easily eliminate.

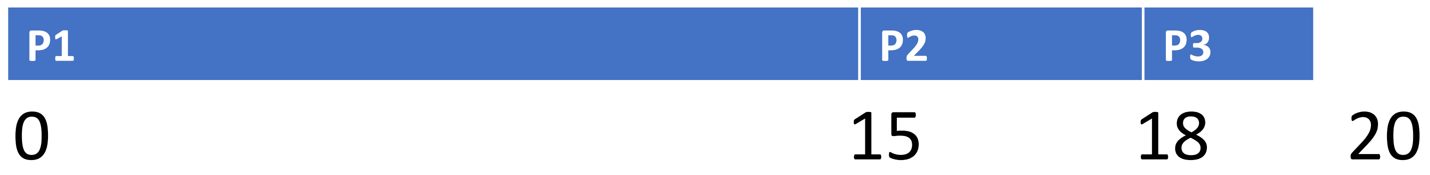
The third way to overcome the deadlock situation is by killing one or more processes.

FCFS

- Consider we have following set of processes that arrive at time 0 with execution time in milliseconds:

Process	Burst Time(Execution Time)
P1	15
P2	3
P3	2

Gantt Chart(Time Line Chart) :



- Here in ready queue, we have three processes P1, P2, and P3. Now,   first P1 process is selected first and will be executed first, then P2, and P3. Here if any new process comes, then it will be added to the tail part of the ready queue.
- If all three jobs arrive almost simultaneously, we can calculate that the turnaround time for each Processes is as follows:

P1=15 milliseconds

P2=18 milliseconds

P3=20 milliseconds

Average Turnaround time: (15+18+20)/3

=53/3

=17.66 ms

Average Waiting time for each processes is as follows:

- P1=0 milliseconds
- P2=15 milliseconds
- P3=18 milliseconds

Average Turnaround time: (0+15+18)/3

=33/3

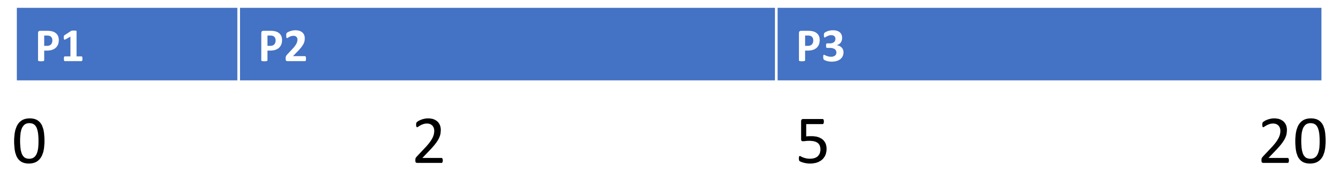
=11 ms

SJF/SJN: (Shortest Job First Scheduling/ Shortest Job Next Scheduling)

- Consider we have following set of processes that arrive at time 0 with execution time in milliseconds:

Process	Burst Time(Execution Time)
P1	15
P2	3
P3	2

Gantt Chart(Time Line Chart) :



- Here in ready queue, we have three processes P1, P2, and P3. Now,   first P3 process is selected first because it has shortest execution time, then P2, and P1.
- If all three jobs arrive almost simultaneously, we can calculate that the turnaround time for each Processes is as follows:

P1=20 milliseconds

P2=5 milliseconds

P3=2 milliseconds

Average Turnaround time : (20+5+2)/3

=27/3

=9 ms

Average Waiting time for each processes is as follows:

- P1=5 milliseconds
- P2=2 milliseconds
- P3=0 milliseconds

Average Turnaround time: (5+2+0)/3

=7/3

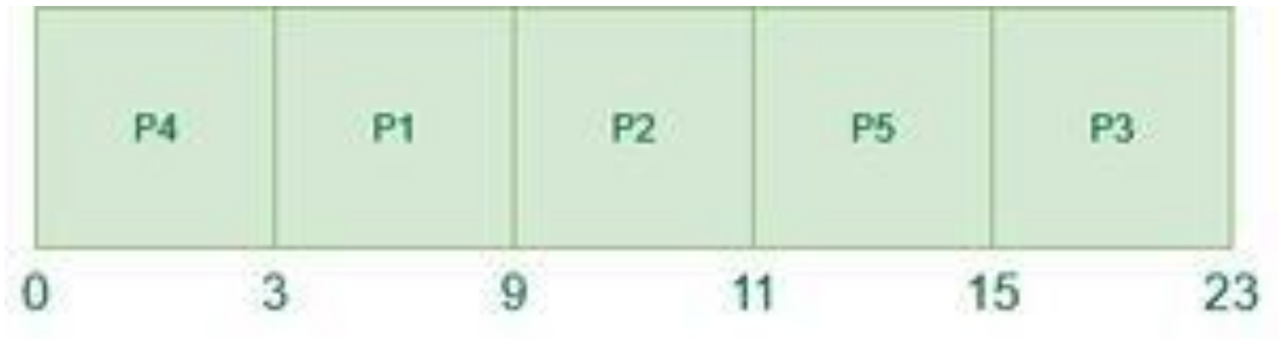
=2.33 ms

SJF Example

- **Example-1:** Consider the following table of arrival time and burst time for five processes **P1, P2, P3, P4** and **P5**.

Process	**Burst Time**	**Arrival Time**
P1	6 ms	2 ms
P2	2 ms	5 ms
P3	8 ms	1 ms
P4	3 ms	0 ms
P5	4 ms	4 ms

Gantt chart for above execution:



Gantt chart

Now, let’s calculate the average waiting time for above example:

P4 = 0 – 0 = 0

P1 = 3 – 2 = 1

P2 = 9 – 5 = 4

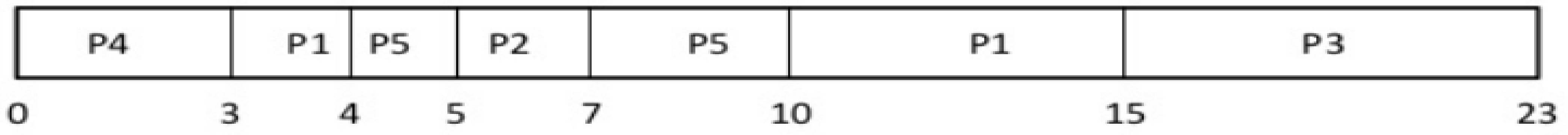
P5 = 11 – 4 = 7

P3 = 15 – 1 = 14

Average Waiting Time = 0 + 1 + 4 + 7 + 14/5 = 26/5 = 5.2

Preemptive SJF

Process Queue	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4



Let’s calculate the average waiting time for above example.
Wait time P4= 0-0=0 P1= (3-2) + 6 =7 P2= 5-5 = 0 P5= 4-4+2 =2 P3= 15-1 = 14

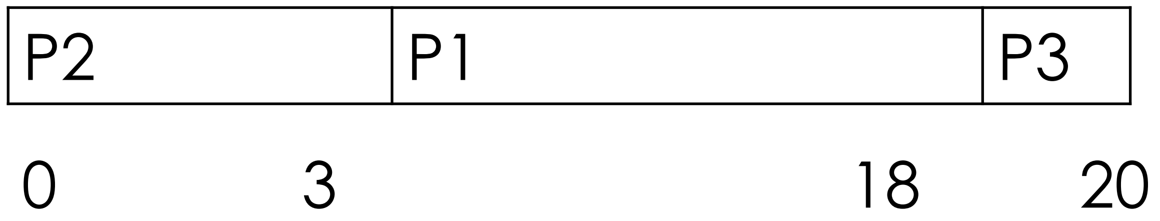
Average Waiting Time = 0+7+0+2+14/5 = 23/5 =4.6

Priority Scheduling:-

✓Consider we have following set of processes that :

Process	Burst Time(Execution Time)	Priority
P1	15	2
P2	3	1
P3	2	2

- **Gantt Chart(Time Line Chart) :**



- If all three jobs arrive almost simultaneously, we can calculate that the turnaround time for each Processes is as follows:
-
- P1=18 milliseconds
- P2=3 milliseconds
- P3=20 milliseconds

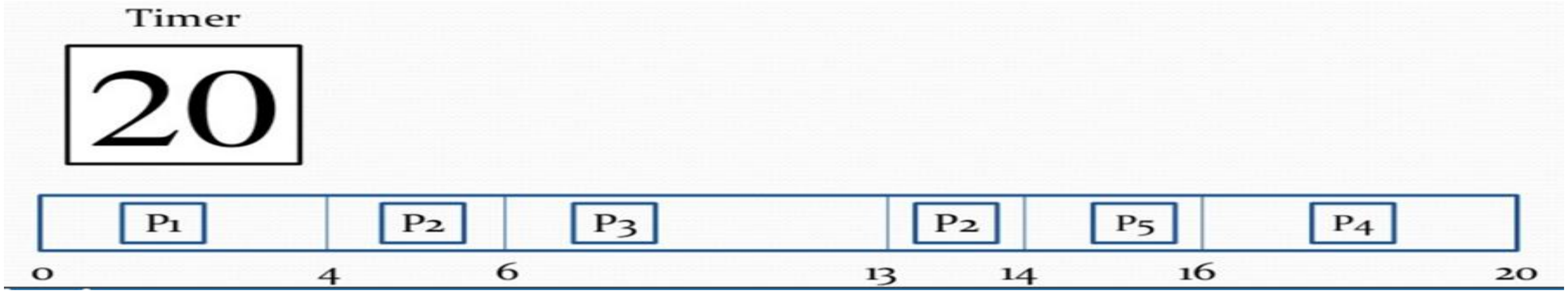
  Average Turn around time : (3+18+20)/3 =41/3

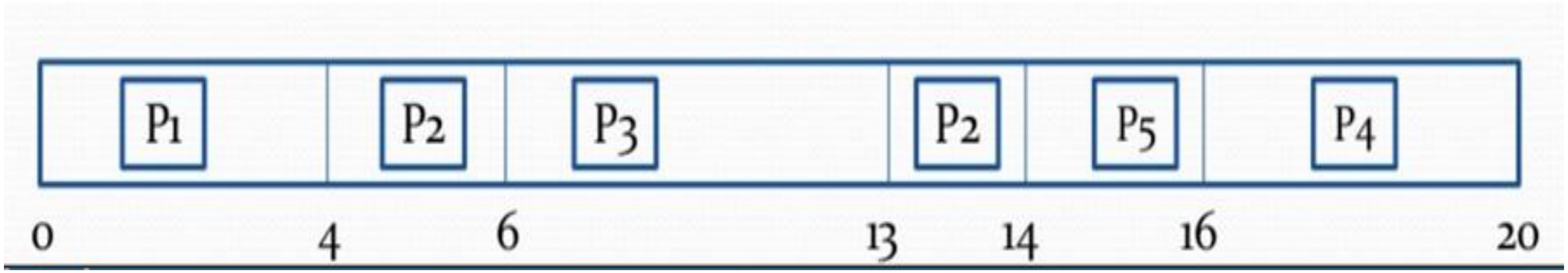
  =13.66 ms
- Average Waiting time for each processes is as follows:
  - P1=3 milliseconds
  - P2=0 milliseconds
  - P3=18 milliseconds
- Average Turn around time: (0+15+18)/3 =21/3

  =7 ms

Consider following five processes P1 to P5. Each process has its unique priority, burst time, and arrival time.

Process	Priority	Burst time	Arrival time
P1	1	4	0
P2	2	3	0
P3	1	7	6
P4	3	4	11
P5	2	2	12





Let’s calculate the average waiting time for the above example.

Waiting Time = start time – arrival time + wait time for next burst

P1 = 0 - 0 = 0

P2 =4 - 0 + 7 =11

P3= 6-6=0

P4= 16-11=5

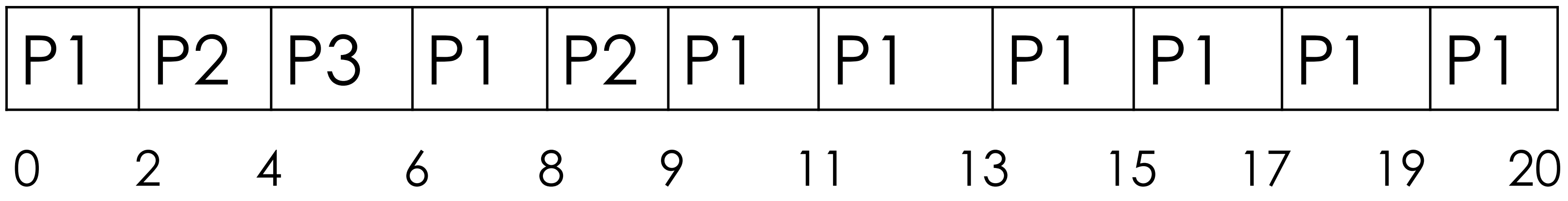
Average Waiting time = (0+11+0+5+2)/5 = 18/5= 3.6

Round Robin Scheduling:-

- Consider we have following set of processes, and time slice=2 ms:

Process	Burst Time(Execution Time)
`P1	15
P2	3
P3	2

- **Gantt Chart(Time Line Chart) :**



- Here in ready queue, we have three processes P1, P2, and P3. We have 2 milliseconds time slice. first P1 process will be executed. After 2 millisecond (completion of time slice) P2 process will be executed. P1 will be arranged at the end of queue and so on.
- If all three jobs arrive almost simultaneously, we can calculate that the

  Turnaround time for each Processes is as follows:

  P1=20 milliseconds

  P2=9 milliseconds

  P3=6 milliseconds

  Average Turnaround time: (6+9+20)/3 =35/3

  =11.66 ms
- Average Waiting time for each processes is as follows:

  P1= (9-4)=5 milliseconds

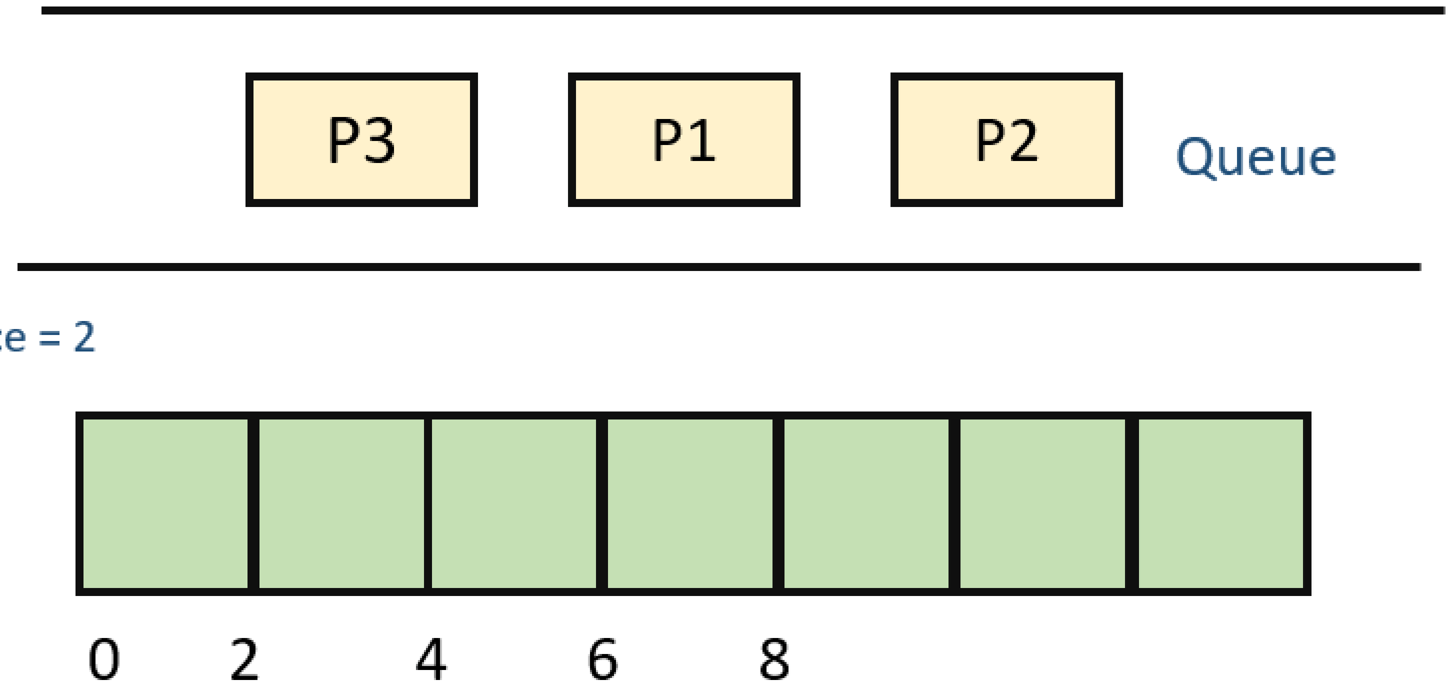
  P2= (8-2)=6 milliseconds

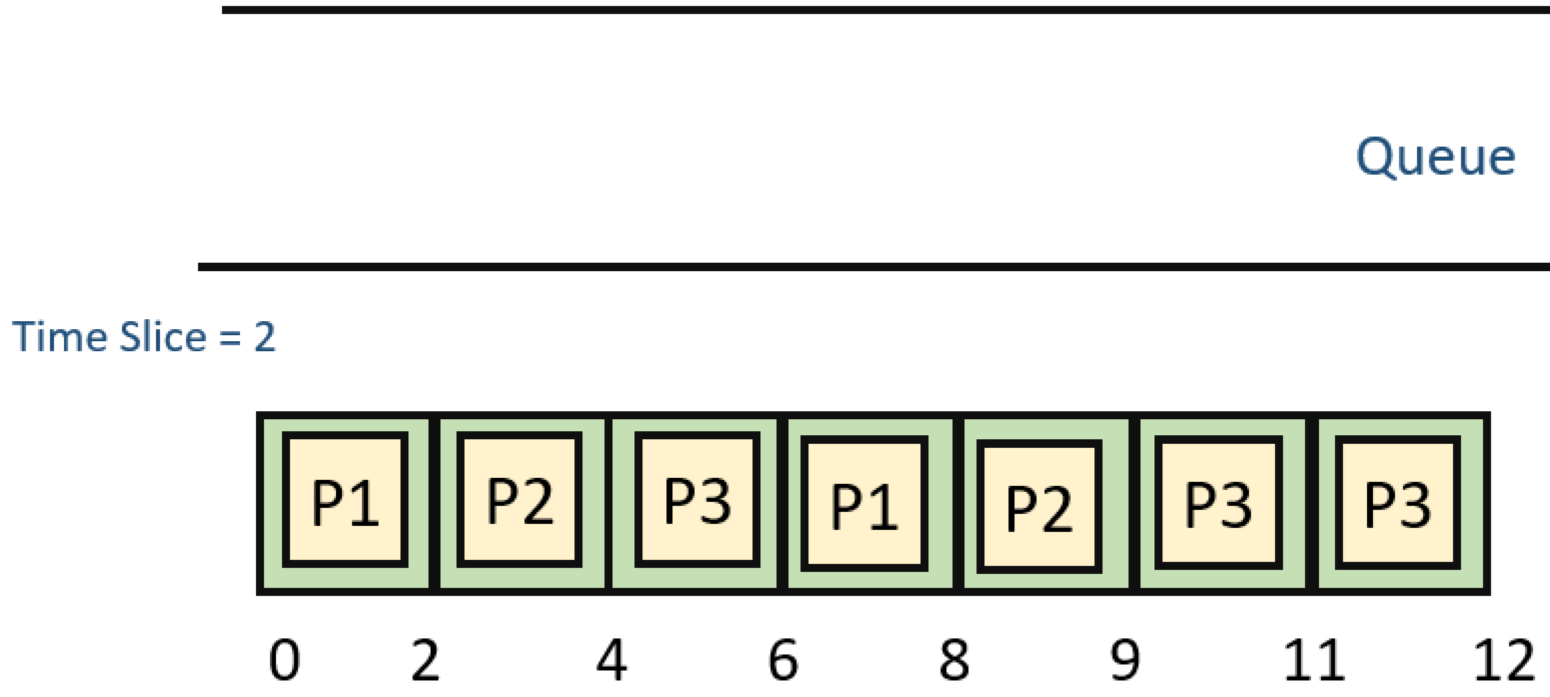
  P3= 4 milliseconds
- Average Turnaround time: (5+6+4)/3 =15/3

  =5 ms

Consider this following three processes

Process Queue	Burst time
P1	4
P2	3
P3	5





Let’s calculate the average waiting time for above example.
Wait time
P1= 0+ 4= 4
P2= 2+4= 6
P3= 4+3= 7
AWT = (4+6+7) /3
= 5.6 ms

Consider the following set of processes, with the arrival times and the CPU-burst times given in milliseconds

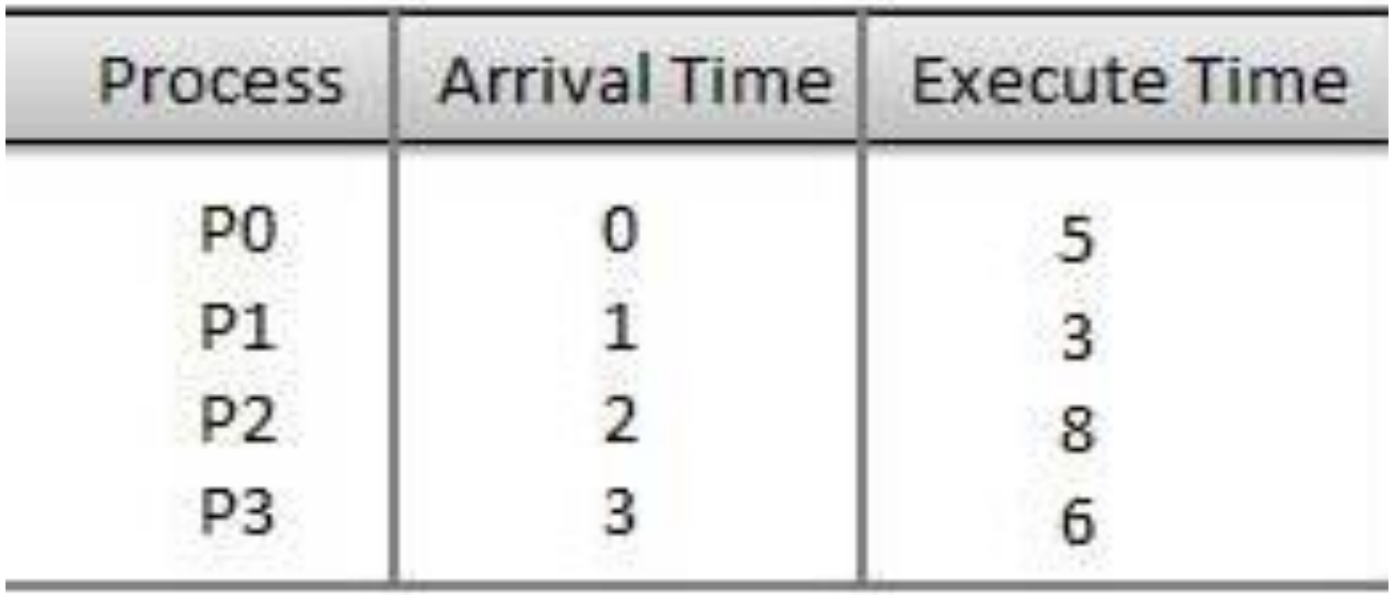
Process	Arrival time	Burst Time
P1	0 ms	5 ms
P2	1 ms	3 ms
P3	2 ms	3 ms
P4	4 ms	1 ms

1.What is the average turnaround time for these processes with the preemptive shortest remaining processing time first (SRPT) algorithm ?

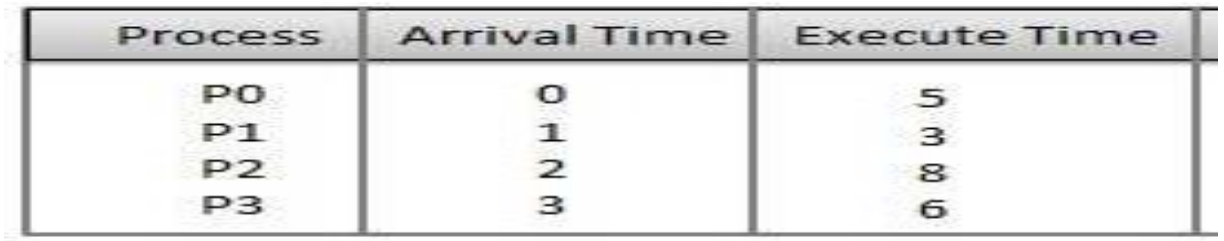
P1	P2	P4	P3	P1
1	4	5	8	12

Turn Around Time = Completion Time – Arrival Time Avg
Turn Around Time = (12 + 3 + 6+ 1)/4 = 5.50

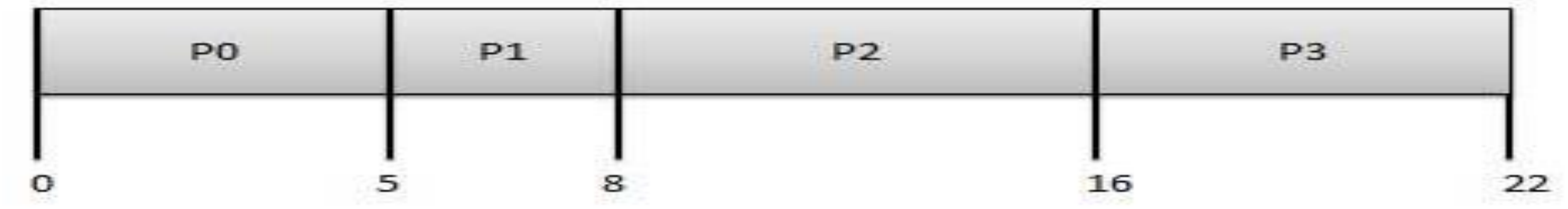
First Come First Serve (FCFS)



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



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



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

Wait time of each process is as follows −
Average Wait Time: (0+4+6+13) / 4 = 5.75

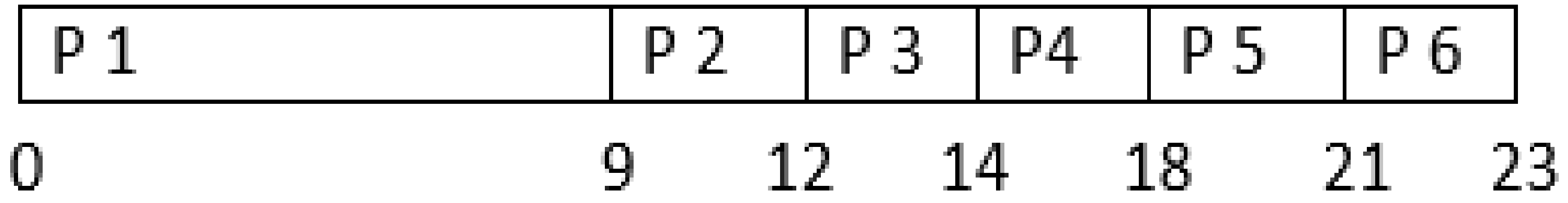
First Come First Serve (FCFS)

Example

S. No	Process ID	Process Name	Arrival Time	Burst Time
1	P 1	A	0	9
2	P 2	B	1	3
3	P 3	C	1	2
4	P 4	D	1	4
5	P 5	E	2	3
6	P 6	F	3	2

Non Pre Emptive Approach

Now, let us solve this problem with the help of the Scheduling Algorithm named First Come First Serve in a Non Preemptive Approach.
Gantt chart for the above Example is:



Turn Around Time = Completion Time - Arrival Time
Waiting Time = Turn Around Time - Burst Time

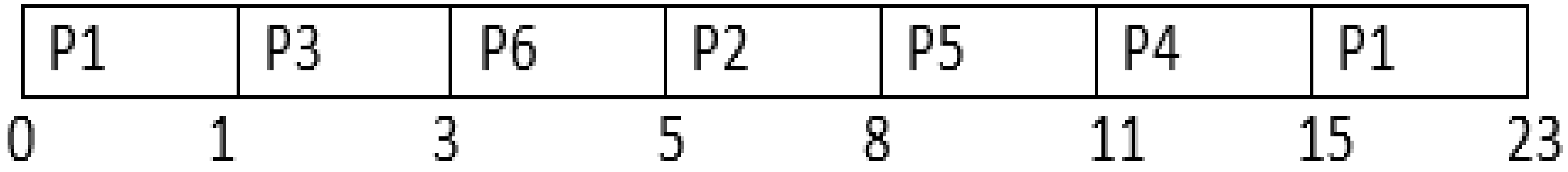
The Average Completion Time is:
Average CT = (9 + 12 + 14 + 18 + 21 + 23) / 6
Average CT = 97 / 6
Average CT = 16.16667
The Average Waiting Time is:
Average WT = (0 + 8 + 11 + 13 + 16 + 18) /6
Average WT = 66 / 6
Average WT = 11
The Average Turn Around Time is:
Average TAT = (9 + 11 + 13 + 17 + 19 +20) / 6
Average TAT = 89 / 6
Average TAT = 14.83334

Pre Emptive Approach

Now, let us solve this problem with the help of the Scheduling Algorithm named First Come First Serve in a Pre Emptive Approach.

In Pre Emptive Approach we search for the best process which is available

Gantt chart for the above Example 1 is:



The Average Completion Time is:

Average CT = (23 + 8 + 3 + 15 + 11 + 5) / 6

Average CT = 65 / 6

Average CT = 10.83333

The Average Waiting Time is:

Average WT = (14 + 4 + 0 + 10 + 7 + 0) /6

Average WT = 35 / 6

Average WT = 5.83333

The Average Turn Around Time is:

Average TAT = (23 + 7 + 2 + 14 + 9 +2) / 6

Average TAT = 57 / 6

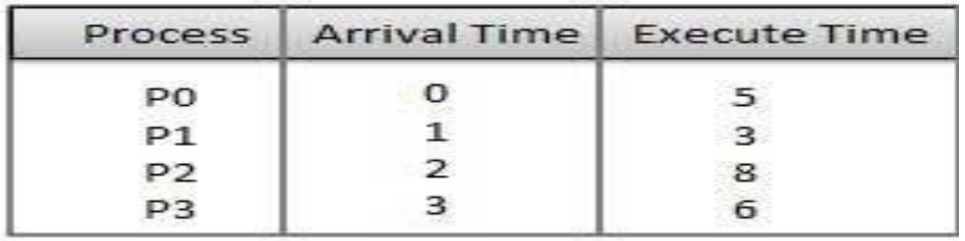
Average TAT = 9.5

This is how the FCFS is solved in Pre Emptive Approach.

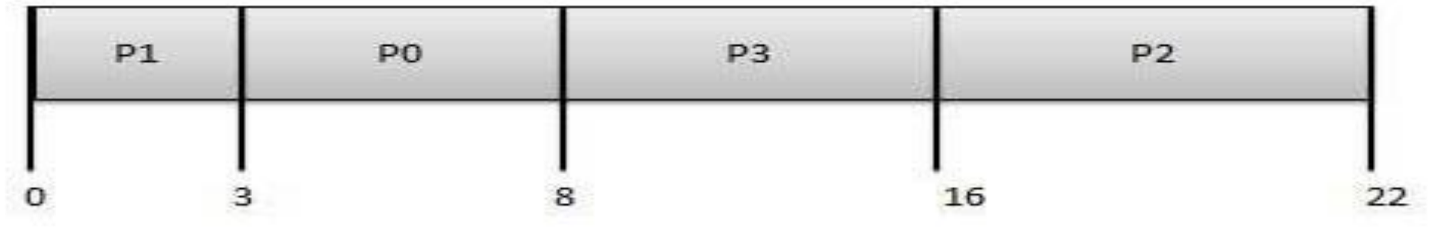
Shortest Job Next (SJN)

Given: Table of processes, and their Arrival time, Execution time

Process	Arrival Time	Execution Time
P0	0	5
P1	1	3
P2	2	8
P3	3	6



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



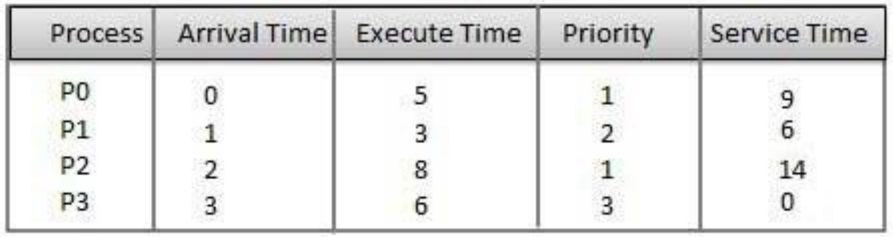
Process	**Waiting Time**
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	14 - 2 = 12
P3	8 - 3 = 5

Waiting time of each process is as follows −
Average Wait Time: (0 + 4 + 12 + 5)/4 = 21 / 4 = 5.25

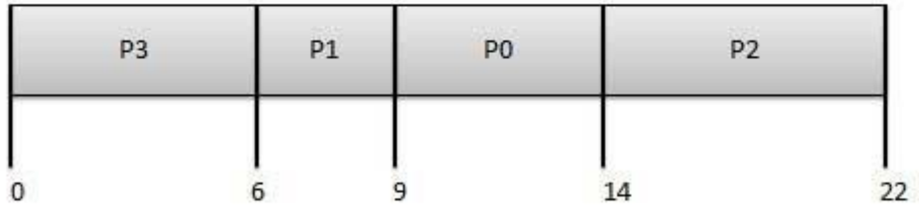
Priority Based Scheduling

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



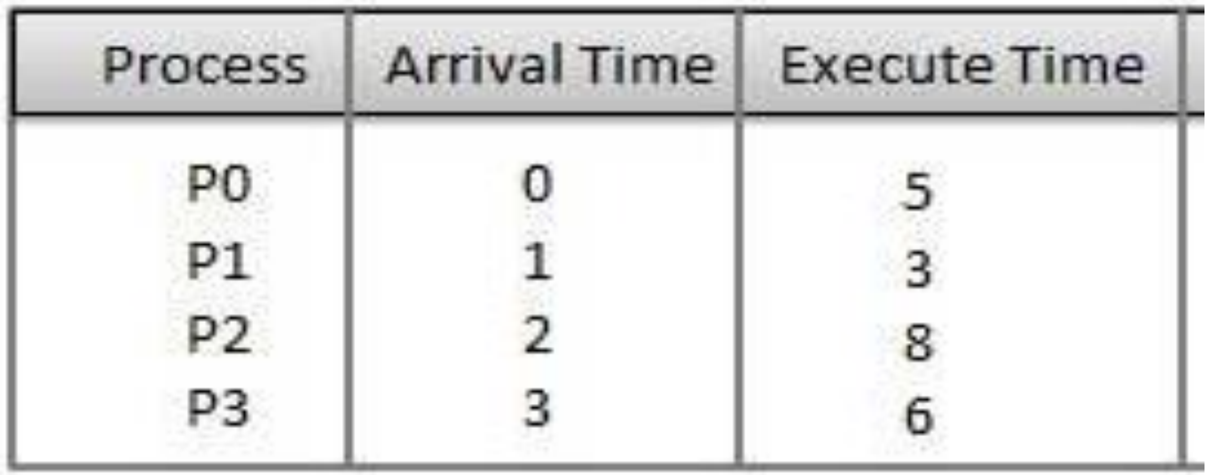
Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



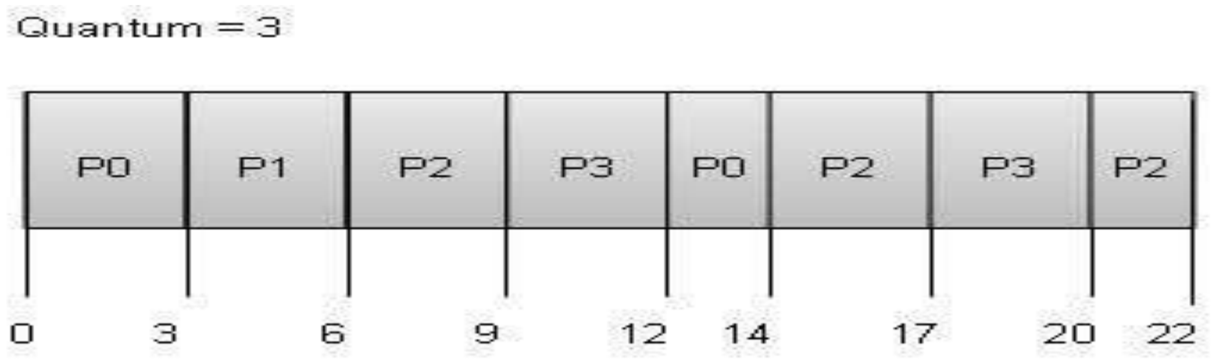
Process	Waiting Time
P0	0 - 0 = 0
P1	11 - 1 = 10
P2	14 - 2 = 12
P3	5 - 3 = 2

Waiting time of each process is as follows −
Average Wait Time: (0 + 10 + 12 + 2)/4 = 24 / 4 = 6

Round Robin Scheduling



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



Wait time of each process is as follows −
Average Wait Time: (9+2+12+11) / 4 = 8.5

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