1. In c/c++ there are 4 steps of running a program

* Preprocessor—inclusion of files ,expansion of micros
* Compiler—high level language is converted into machine leve(object file with .out extension)
* Linker—here the object file is linked with other oject files and forms executable file with(.exe extension) this is stored in hard disk
* Loader—the loader work is to load the exexutable file into ram and run it

1. In java there is not 1 step compilation .This requires 2 steps

* Compilation step
* Java file is passed through a compiler and it is converted into machine independent code which is (bytecode or .class files)
* Execution step
* These class files can be run on any machine they are independent of machine
* To run the class containing the main method is passed to the jvm and then further three steps takes place
* Class Loader – the class containing main function is loaded by passing passed its .class file to jvm

Class loader is basically an object

There are two types of class loaders—primordial and non primordial.Primordial are embedded into jvm and is the default class loader where as non primordial is user defined and its priority is higher than default of primordial

* ByteCode verifier—after loading class it checks that the instructions don’t perform damaging actions like
* Run time stack overflow
* Variables should be initialized before they are used
* Rules for accessing private data and methods are not violated
* Just In Time(JIT) compiler—this is the final stage .Its job is to convert the loaded byte code into machine code.This component is responsible for improving the speed of java application at run time

When a method has been compiled, the JVM calls the compiled code of that method directly instead of interpreting it.

1. Diff between multiprogramming and multitasking

* Multiprogramming—this is non preemptive method .here all the processes are keps in an area called job pool.The process is taken one by one and completed whole .when the process goes for input output then the cpu chooses the next process.OS remians ideal for a very less time.
* Multitasking –this is preemptive types.One Process is run for a particular quantum .This improves the responsiveness

Process

* Program in execution is called process
* When the program is loaded into memory it becomes the process.

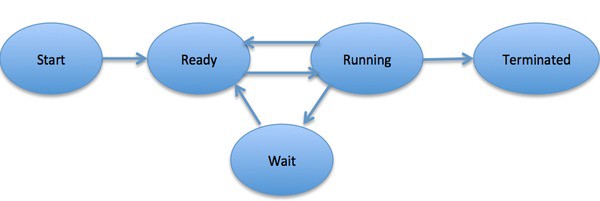
The process is divided into 4 sections

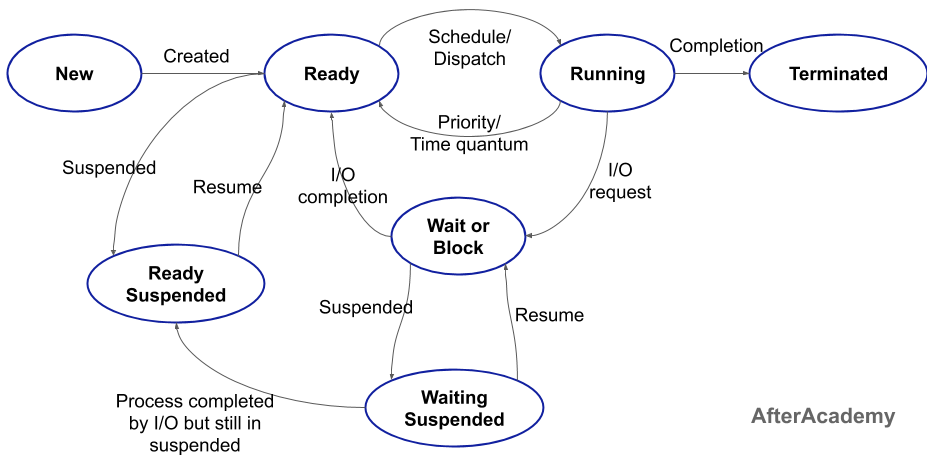
Stack—tempopary data method functions ,variables

heap—dynamically allocated data

data—global and static data

Text – current activity represented by program counter





# When a process is executed it passes through 5 states basically

* **New State:** This is the state when the process is just created. It is the first state of a process.
* **Ready State:** After the creation of the process, when the process is ready for its execution then it goes in the ready state. In a ready state, the process is ready for its execution by the CPU but it is waiting for its turn to come. There can be more than one process in the ready state.
* **Ready Suspended State:** There can be more than one process in the ready state but due to memory constraint, if the memory is full then some process from the ready state gets placed in the ready suspended state.
* **Running State:** Amongst the process present in the ready state, the CPU chooses one process amongst them by using some CPU scheduling algorithm. The process will now be executed by the CPU and it is in the running state.
* **Waiting or Blocked State:** During the execution of the process, the process might require some I/O operation like writing on file or some more priority process might come. In these situations, the running process will have to go into the waiting or blocked state and the other process will come for its execution. So, the process is waiting for something in the waiting state.
* **Waiting Suspended State:** When the waiting queue of the system becomes full then some of the processes will be sent to the waiting suspended state.
* **Terminated State:** After the complete execution of the process, the process comes into the terminated state and the information related to this process is deleted.

PCB

*A Process Control Block or simple PCB is a data structure that is used to store the information of a process that might be needed to manage the scheduling of a particular process.*

So, each process will be given a PCB which is a kind of identification card for a process. All the processes present in the system will have a PCB associated with it and all these PCBs are connected in a Linked List

#### Attributes of a Process Control Block

There are various attributes of a PCB that helps the CPU to execute a particular process. These attributes are:

* **Process Id:** A process id is a unique identity of a process. Each process is identified with the help of the process id.
* **Program Counter:** The program counter, points to the next instruction that is to be executed by the CPU. It is used to find the next instruction that is to be executed.
* **Process State:** A process can be in any state out of the possible states of a process. So, the CPU needs to know about the current state of a process, so that its execution can be done easily. You can learn more about process state from [here](https://afteracademy.com/blog/what-is-a-process-in-operating-system-and-what-are-the-different-states-of-a-process).
* **Priority:** There is a priority associated with each process. Based on that priority the CPU finds which process is to be executed first. Higher priority process will be executed first.
* **General-purpose Registers:** During the execution of a process, it deals with a number of data that are being used and changed by the process. But in most of the cases, we have to stop the execution of a process to start another process and after some times, the previous process should be resumed once again. Since the previous process was dealing with some data and had changed the data so when the process resumes then it should use that data only. These data are stored in some kind of storage units called registers.
* **CPU Scheduling Information:** It indicates the information about the process scheduling algorithms that are being used by the CPU for the process.
* **List of opened files:** A process can deal with a number of files, so the CPU should maintain a list of files that are being opened by a process to make sure that no other process can open the file at the same time.
* **List of I/O devices:** A process may need a number of I/O devices to perform various tasks. So, a proper list should be maintained that shows which I/O device is being used by which process.

#### What is Context Switching?

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

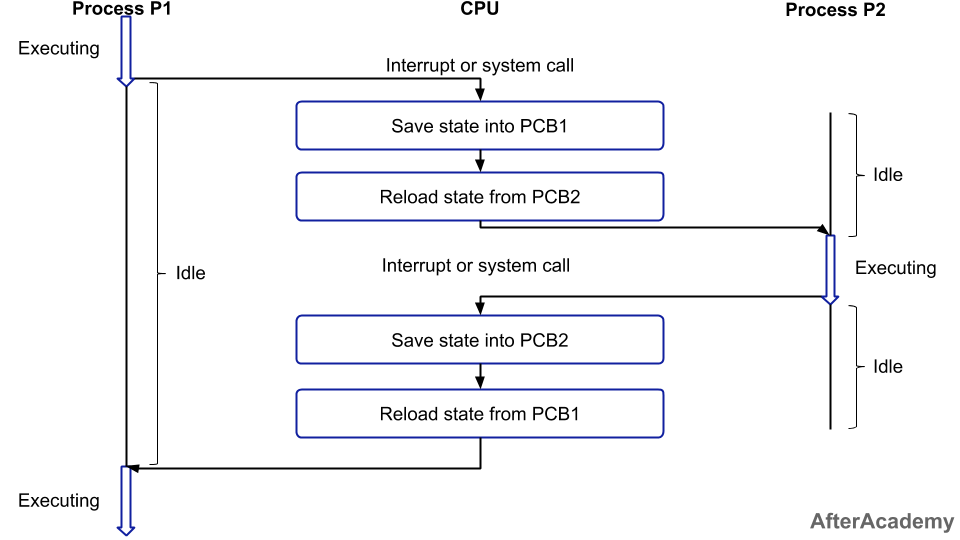
It is one of the essential features of the multitasking operating system. The processes are switched so fastly that it gives an illusion to the user that all the processes are being executed at the same time.

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

A context is the contents of a CPU's registers and program counter at any point in time.

Context switching can happen due to the following reasons:

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



The following steps will be performed:

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

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

#### Advantage of Context Switching

Context switching is used to achieve multitasking i.e. multiprogramming with time-sharing(learn more about multitasking from [here](https://afteracademy.com/blog/multiprogramming-vs-multiprocessing-vs-multitasking)). Multitasking gives an illusion to the users that more than one process are being executed at the same time. But in reality, only one task is being executed at a particular instant of time by a processor. Here, the context switching is so fast that the user feels that the CPU is executing more than one task at the same time.

#### The disadvantage of Context Switching

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

#### Priority Scheduling Algorithm

Before getting into the details of Starvation, let's have a quick overview of Priority Scheduling algorithms.

In Priority scheduling technique, we assign some priority to every process we have and based on that priority, the CPU will be allocated and the process will be executed. Here, the CPU will be allocated to the process that is having the highest priority. **We don't care about the burst time here.** Even if the burst time is low, the CPU will be allocated to the process having the highest priority.

## What is Long-Term, Short-Term, and Medium-Term Scheduler?

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

Long-Term schedulers are those schedulers whose decision will have a long-term effect on the performance. The duty of the long-term scheduler is to bring the process from the JOB pool to the Ready state for its execution.

***Long-Term Scheduler is also called Job Scheduler and is responsible for controlling the Degree of Multiprogramming i.e. the total number of processes that are present in the ready state.***

So, the long-term scheduler decides which process is to be created to put into the ready state.

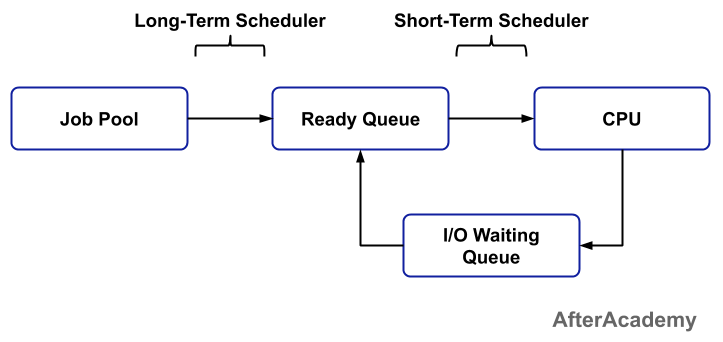
**Effect on performance**

* The long term scheduler is responsible for creating a balance between the I/O bound(a process is said to be I/O bound if the majority of the time is spent on the I/O operation) and CPU bound(a process is said to be CPU bound if the majority of the time is spent on the CPU). So, if we create processes which are all I/O bound then the CPU might not be used and it will remain idle for most of the time. This is because the majority of the time will be spent on the I/O operation.
* So, if we create processes that are having high a CPU bound or a perfect balance between the I/O and CPU bound, then the overall performance of the system will be increased
* Short-term schedulers are those schedulers whose decision will have a short-term effect on the performance of the system. The duty of the short-term scheduler is to schedule the process from the ready state to the running state. This is the place where all the scheduling algorithms are used i.e. it can be FCFS or Round-Robin or SJF or any other scheduling algorithm.

#### Short-Term Scheduler

Short-term schedulers are those schedulers whose decision will have a short-term effect on the performance of the system. The duty of the short-term scheduler is to schedule the process from the ready state to the running state. This is the place where all the scheduling algorithms are used i.e. it can be FCFS or Round-Robin or SJF or any other scheduling algorithm.

***Short-Term Scheduler is also known as CPU scheduler and is responsible for selecting one process from the ready state for scheduling it on the running state.***



**Effect on performance**

* The choice of the short-term scheduler is very important for the performance of the system. If the short-term scheduler only selects a process that is having very high burst time(learn more about burst time from [here](https://afteracademy.com/blog/what-is-burst-arrival-exit-response-waiting-turnaround-time-and-throughput)) then the other process may go into the condition of starvation(learn more about starvation from [here](https://afteracademy.com/blog/what-is-starvation-and-aging)). So, be specific when you are choosing short-term scheduler because the performance of the system is our highest priority.

The following image shows the scheduling of processes using the long-term and short-term schedulers.

#### Dispatcher

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

Now, the selected process has to be transferred from the current state to the desired or scheduled state. So, it is the duty of the dispatcher to dispatch or transfer a process from one state to another. A dispatcher is responsible for context switching and switching to user mode(learn more about context switching from [here](https://afteracademy.com/blog/what-is-context-switching-in-operating-system)).

#### Difference between Dispatcher and Scheduler

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

* The scheduler selects a process from a list of processes by applying some process scheduling algorithm. On the other hand, the dispatcher transfers the process selected by the short-term scheduler from one state to another.
* The scheduler works independently, while the dispatcher has to be dependent on the scheduler i.e. the dispatcher transfers only those processes that are selected by the scheduler.
* For selecting a process, the scheduler uses some process scheduling algorithm like FCFS, Round-Robin, SJF, etc. But the dispatcher doesn't use any kind of scheduling algorithms.
* The only duty of a scheduler is to select a process from a list of processes. But apart from transferring a process from one state to another, the dispatcher can also be used for switching to user mode. Also, the dispatcher can be used to jump to a proper location when the process is restarted.
* In the Operating System, the process scheduling algorithms can be divided into two broad categories i.e. Preemptive Scheduling and Non-Preemptive Scheduling. In this blog, we will learn the difference between these two. So, let's get started.

#### Non-preemptive Scheduling

In non-preemptive scheduling, if some resource is allocated to a process then that resource will not be taken back until the completion of the process. Other processes that are present in the ready queue have to wait for its turn and it cann't forcefully get the CPU. Once the CPU is allocated to a process, then it will be held by that process until it completes its execution or it goes in the waiting state for I/O operation.

#### Preemptive Scheduling

* In preemptive scheduling, the CPU will execute a process but for a limited period of time and after that, the process has to wait for its next turn i.e. in preemptive scheduling, the state of a process gets changed i.e. the process may go to the ready state from running state or from the waiting state to the ready state. The resources are allocated to the process for a limited amount of time and after that, they are taken back and the process goes to the ready queue if it still has some CPU burst time remaining. Some of the preemptive scheduling algorithms are Round-robin, SJF (preemptive), etc.

|  |  |
| --- | --- |
| **Preemptive Scheduling** | **Non-preemptive Scheduling** |
| A processor can be preempted to execute the different processes in the middle of any current process execution. | Once the processor starts its execution, it must finish it before executing the other. It can't be paused in the middle. |
| CPU utilization is more efficient compared to Non-Preemptive Scheduling. | CPU utilization is less efficient compared to preemptive Scheduling. |
| Waiting and response time of preemptive Scheduling is less. | Waiting and response time of the non-preemptive Scheduling method is higher. |
| Preemptive Scheduling is prioritized. The highest priority process is a process that is currently utilized. | When any process enters the state of running, the state of that process is never deleted from the scheduler until it finishes its job. |
| Preemptive Scheduling is flexible. | Non-preemptive Scheduling is rigid. |
| Examples: - Shortest Remaining Time First, Round Robin, etc. | Examples: First Come First Serve, Shortest Job First, Priority Scheduling, etc. |
| Preemptive Scheduling algorithm can be pre-empted that is the process can be Scheduled | In non-preemptive scheduling process cannot be Scheduled |
| In this process, the CPU is allocated to the processes for a specific time period. | In this process, CPU is allocated to the process until it terminates or switches to the waiting state. |
| Preemptive algorithm has the overhead of switching the process from the ready state to the running state and vice-versa. | Non-preemptive Scheduling has no such overhead of switching the process from running into the ready state. |

#### Burst time

Every process in a computer system requires some amount of time for its execution. This time is both the CPU time and the I/O time. The CPU time is the time taken by CPU to execute the process. While the I/O time is the time taken by the process to perform some I/O operation. In general, we ignore the I/O time and we consider only the CPU time for a process. So, **Burst time is the total time taken by the process for its execution on the CPU.**

#### Arrival time

Arrival time is the time when a process enters into the ready state and is ready for its execution.

#### Exit time

Exit time is the time when a process completes its execution and exit from the system.

#### Response time

Response time is the time spent when the process is in the ready state and gets the CPU for the first time.

**Response time = Time at which the process gets the CPU for the first time - Arrival time**

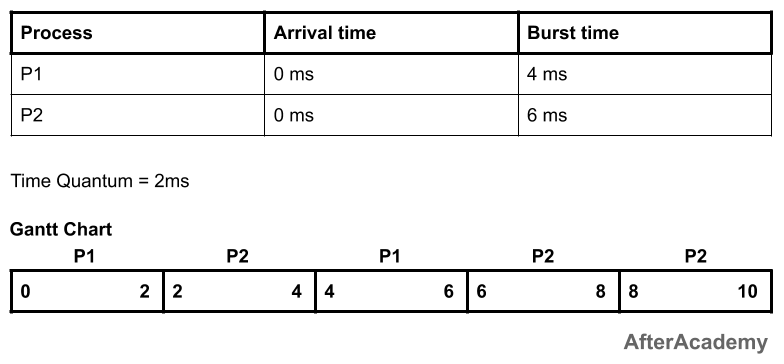
#### Waiting time

Waiting time is the total time spent by the process in the ready state waiting for CPU.

**Waiting time = Turnaround time - Burst time**

In the above example, the processes have to wait only once. But in many other scheduling algorithms, the CPU may be allocated to the process for some time and then the process will be moved to the waiting state and again after some time, the process will get the CPU and so on.

There is a difference between waiting time and response time. Response time is the time spent between the ready state and getting the CPU for the first time. But the waiting time is the total time taken by the process in the ready state. Let's take an example of a round-robin scheduling algorithm. The time quantum is 2 ms.



In the above example, the response time of the process P2 is 2 ms because after 2 ms, the CPU is allocated to P2 and the waiting time of the process P2 is 4 ms i.e turnaround time - burst time (10 - 6 = 4 ms).

#### Turnaround time

Turnaround time is the total amount of time spent by the process from coming in the ready state for the first time to its completion.

**Turnaround time = Burst time + Waiting time**

**or**

**Turnaround time = Exit time - Arrival time**

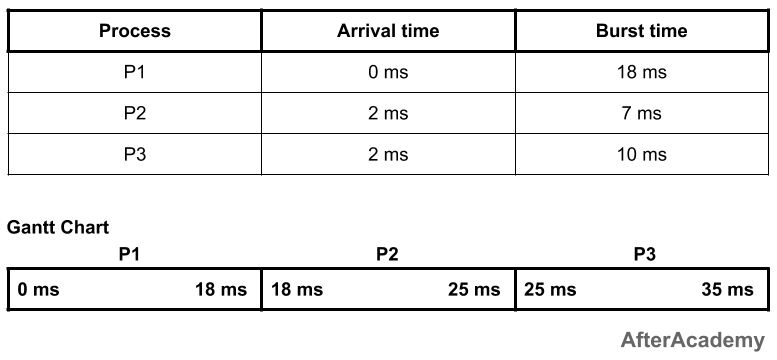
#### Throughput

Throughput is a way to find the efficiency of a CPU. It can be defined as the number of processes executed by the CPU in a given amount of time. For example, let's say, the process P1 takes 3 seconds for execution, P2 takes 5 seconds, and P3 takes 10 seconds. So, throughput, in this case, the throughput will be (3+5+10)/3 = 18/3 = 6 seconds.

#### First Come First Serve (FCFS)

As the name suggests, the process coming first in the ready state will be executed first by the CPU irrespective of the burst time or the priority. This is implemented by using the **First In First Out (FIFO)** queue. So, what happens is that, when a process enters into the ready state, then the PCB of that process will be linked to the tail of the queue and the CPU starts executing the processes by taking the process from the head of the queue (learn more about PCB from [here](https://afteracademy.com/blog/process-control-block-in-operating-system)). If the CPU is allocated to a process then it can't be taken back until it finishes the execution of that process.

**Example:**



In the above example, you can see that we have three processes P1, P2, and P3, and they are coming in the ready state at 0ms, 2ms, and 2ms respectively. So, based on the arrival time, the process P1 will be executed for the first 18ms. After that, the process P2 will be executed for 7ms and finally, the process P3 will be executed for 10ms. One thing to be noted here is that if the arrival time of the processes is the same, then the CPU can select any process.

---------------------------------------------

| Process | Waiting Time | Turnaround Time |

---------------------------------------------

| P1 | 0ms | 18ms |

| P2 | 16ms | 23ms |

| P3 | 23ms | 33ms |

---------------------------------------------

Total waiting time: (0 + 16 + 23) = 39ms

Average waiting time: (39/3) = 13ms

Total turnaround time: (18 + 23 + 33) = 74ms

Average turnaround time: (74/3) = 24.66ms

**Advantages of FCFS:**

* It is the most simple scheduling algorithm and is easy to implement.

**Disadvantages of FCFS:**

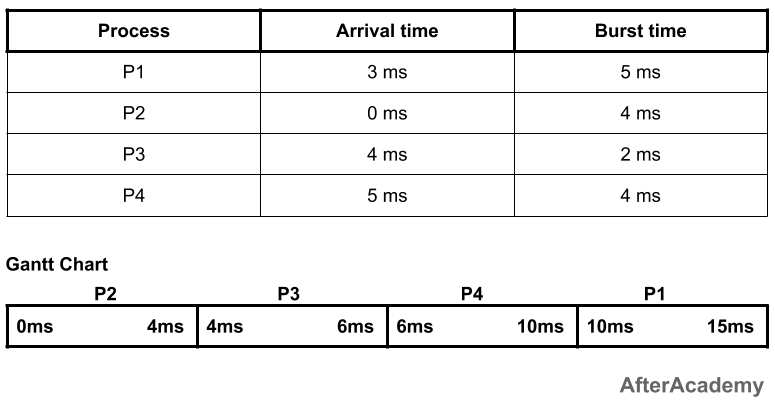
* This algorithm is non-preemptive so you have to execute the process fully and after that other processes will be allowed to execute.
* Throughput is not efficient.
* FCFS suffers from the **Convey effect** i.e. if a process is having very high burst time and it is coming first, then it will be executed first irrespective of the fact that a process having very less time is there in the ready state.

#### Shortest Job First (Non-preemptive)

In the FCFS, we saw if a process is having a very high burst time and it comes first then the other process with a very low burst time have to wait for its turn. So, to remove this problem, we come with a new approach i.e. Shortest Job First or SJF.

In this technique, the process having the minimum burst time at a particular instant of time will be executed first. It is a non-preemptive approach i.e. if the process starts its execution then it will be fully executed and then some other process will come.

**Example:**



In the above example, at 0ms, we have only one process i.e. process P2, so the process P2 will be executed for 4ms. Now, after 4ms, there are two new processes i.e. process P1 and process P3. The burst time of P1 is 5ms and that of P3 is 2ms. So, amongst these two, the process P3 will be executed first because its burst time is less than P1. P3 will be executed for 2ms. Now, after 6ms, we have two processes with us i.e. P1 and P4 (because we are at 6ms and P4 comes at 5ms). Amongst these two, the process P4 is having a less burst time as compared to P1. So, P4 will be executed for 4ms and after that P1 will be executed for 5ms. So, the waiting time and turnaround time of these processes will be:

---------------------------------------------

| Process | Waiting Time | Turnaround Time |

---------------------------------------------

| P1 | 7ms | 12ms |

| P2 | 0ms | 4ms |

| P3 | 0ms | 2ms |

| P4 | 1ms | 5ms |

---------------------------------------------

Total waiting time: (7 + 0 + 0 + 1) = 8ms

Average waiting time: (8/4) = 2ms

Total turnaround time: (12 + 4 + 2 + 5) = 23ms

Average turnaround time: (23/4) = 5.75ms

**Advantages of SJF (non-preemptive):**

* Short processes will be executed first.

**Disadvantages of SJF (non-preemptive):**

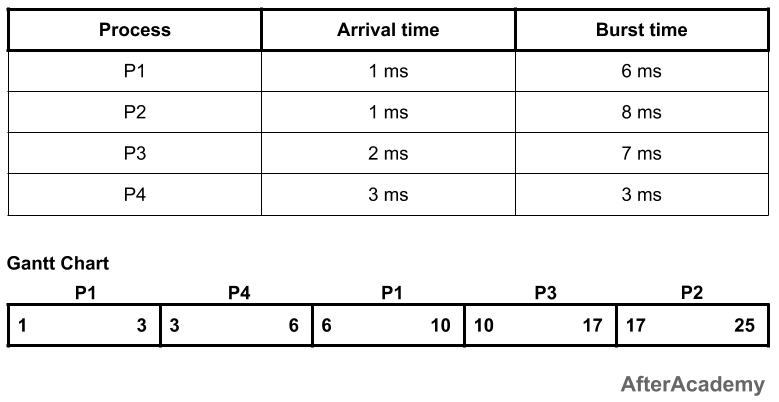
* It may lead to starvation if only short burst time processes are coming in the ready state(learn more about starvation from [here](https://afteracademy.com/blog/what-is-starvation-and-aging)).

#### Shortest Job First (Preemptive)

This is the preemptive approach of the Shortest Job First algorithm. Here, at every instant of time, the CPU will check for some shortest job. For example, at time 0ms, we have P1 as the shortest process. So, P1 will execute for 1ms and then the CPU will check if some other process is shorter than P1 or not. If there is no such process, then P1 will keep on executing for the next 1ms and if there is some process shorter than P1 then that process will be executed. This will continue until the process gets executed.

*This algorithm is also known as Shortest Remaining Time First i.e. we schedule the process based on the shortest remaining time of the processes.*

**Example:**



---------------------------------------------

| Process | Waiting Time | Turnaround Time |

---------------------------------------------

| P1 | 3ms | 9ms |

| P2 | 16ms | 24ms |

| P3 | 8ms | 15ms |

| P4 | 0ms | 3ms |

---------------------------------------------

Total waiting time: (3 + 16 + 8 + 0) = 27ms

Average waiting time: (27/4) = 6.75ms

Total turnaround time: (9 + 24 + 15 + 3) = 51ms

Average turnaround time: (51/4) = 12.75ms

**Advantages of SJF (preemptive):**

* Short processes will be executed first.

**Disadvantages of SJF (preemptive):**

* It may result in starvation if short processes keep on coming.