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, the 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 synchronisation.
5. Providing mechanisms for process communication.

Attributes of a process

The Attributes of the process are used by the Operating System to create the **process control block (PCB) for each of them. This is also called context of the process. Attributes which are stored in the PCB are described below.**

1. Process ID

When a process is created, a unique id is assigned to the process which is used for unique identification of the process in the system.

2. Program counter

A program counter stores the address of the last instruction of the process on which the process was suspended. The CPU uses this address when the execution of this process is resumed.

3. Process State

The Process, from its creation to the completion, goes through various states which are new, ready, running and waiting. We will discuss about them later in detail.

4. Priority

Every process has its own priority. The process with the highest priority among the processes gets the CPU first. This is also stored on the process control block.

5. General Purpose Registers

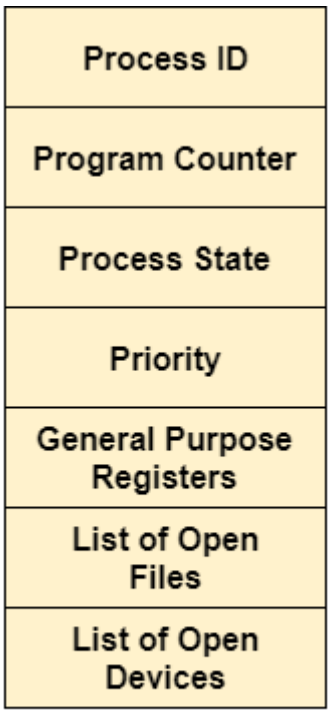
Every process has its own set of registers which are used to hold the data which is generated during the execution of the process.

6. List of open files

During the Execution, Every process uses some files which need to be present in the main memory. OS also maintains a list of open files in the PCB.

7. List of open devices

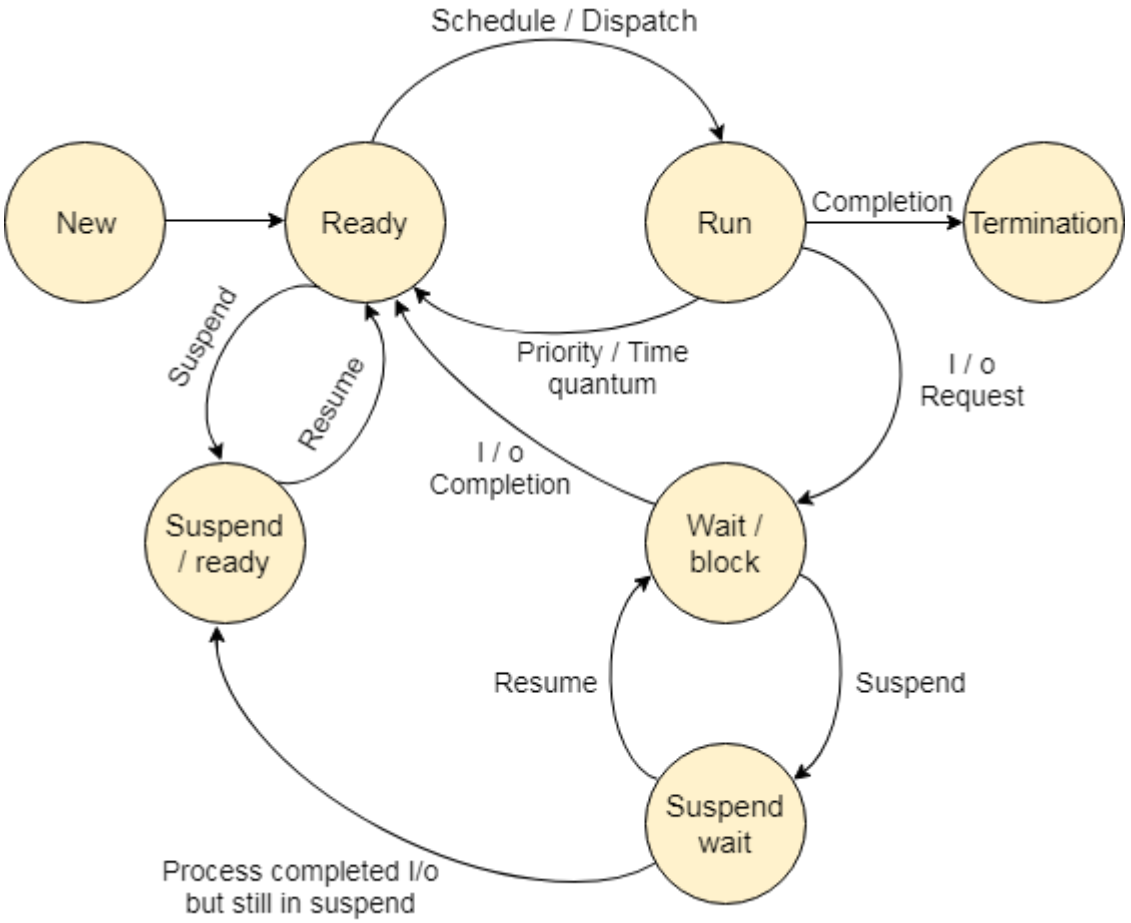
OS also maintains the list of all open devices which are used during the execution of the process.



Process Attributes

Process States (https://www.youtube.com/watch?v=OP4WG5dRIXA)

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 standardised 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 behaviour 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 and 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 suspended ready state.

If the main memory is full and a higher priority process comes for the execution then the OS has to make room for the process in the main memory by throwing the lower priority process out into the secondary memory. **The suspended 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 makes 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. Processes 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.

Process Scheduling in OS (Operating System)

Operating system uses various schedulers for the process scheduling described below.

1. Long term scheduler

Long term scheduler is also known as job scheduler. It chooses the processes from the pool (secondary memory) and keeps them in the ready queue maintained in the primary memory.

Long Term scheduler mainly controls the degree of Multiprogramming. The purpose of long term scheduler is to choose a perfect mix of IO bound and CPU bound processes among the jobs present in the pool.

If the job scheduler chooses more IO bound processes then all of the jobs may reside in the blocked state all the time and the CPU will remain idle most of the time. This will reduce the degree of Multiprogramming. Therefore, the Job of long term scheduler is very critical and may affect the system for a very long time.

2. Short term scheduler

Short term scheduler is also known as CPU scheduler. It selects one of the Jobs from the ready queue and dispatch to the CPU for the execution.

A scheduling algorithm is used to select which job is going to be dispatched for the execution. The Job of the short term scheduler can be very critical in the sense that if it selects job whose CPU burst time is very high then all the jobs after that, will have to wait in the ready queue for a very long time.

This problem is called starvation which may arise if the short term scheduler makes some mistakes while selecting the job.

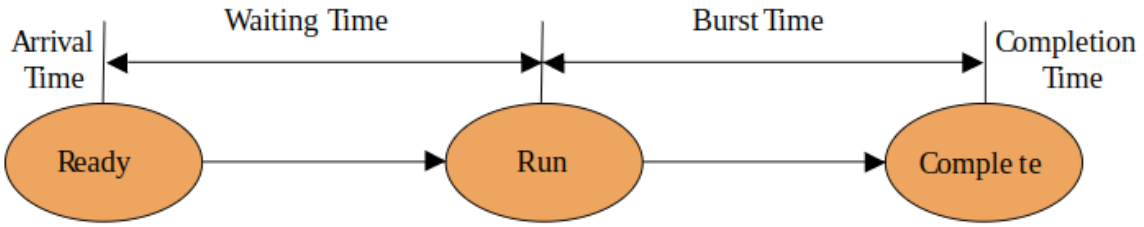
3. Medium term scheduler

Medium term scheduler takes care of the swapped out processes.If the running state processes needs some IO time for the completion then there is a need to change its state from running to waiting.

Medium term scheduler is used for this purpose. It removes the process from the running state to make room for the other processes. Such processes are the swapped out processes and this procedure is called swapping. The medium term scheduler is responsible for suspending and resuming the processes.

It reduces the degree of multiprogramming. The swapping is necessary to have a perfect mix of processes in the ready queue.

Various Times related to the Process



$$CT - AT = WT + BT$$

$$TAT = CT - AT$$

$$\text{Waiting Time} = TAT - BT$$

TAT ⟶ Turn around time

BT ⟶ Burst time

AT ⟶ Arrival time

1. Arrival Time

The time at which the process enters into the ready queue is called the arrival time.

2. Burst Time

The total amount of time required by the CPU to execute the whole process is called the Burst Time. This does not include the waiting time. It is confusing to calculate the execution time for a process even before executing it hence the scheduling problems based on the burst time cannot be implemented in reality.

3. Completion Time

The Time at which the process enters into the completion state or the time at which the process completes its execution, is called completion time.

4. Turnaround time

The total amount of time spent by the process from its arrival to its completion, is called Turnaround time.

5. Waiting Time

The Total amount of time for which the process waits for the CPU to be assigned is called waiting time.

6. Response Time

The difference between the arrival time and the time at which the process first gets the CPU is called Response Time.

CPU Scheduling

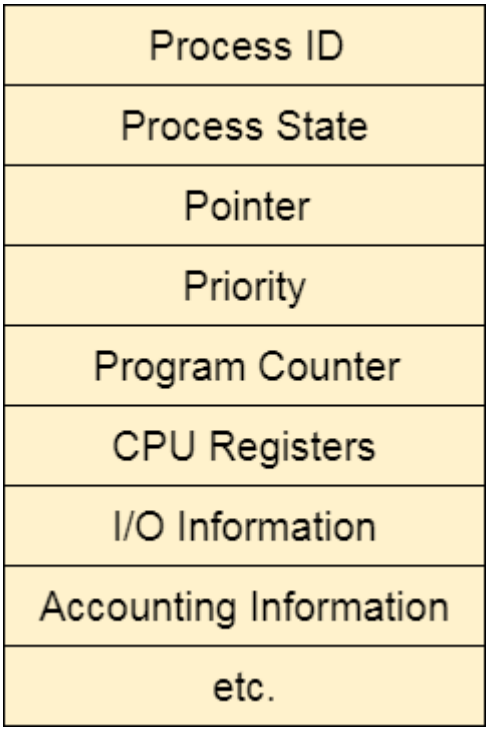
In uniprogramming systems like MS DOS, when a process waits for any I/O operation to be done, the CPU remains idle. This is an overhead since it wastes the time and causes the problem of starvation. However, In Multiprogramming systems, the CPU doesn't remain idle during the waiting time of the Process and it starts executing other processes. The 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 utilisation of it and this procedure is called **CPU scheduling**. The Operating System uses various scheduling algorithms 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, the 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 in OS (Operating System)

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 Algorithms in OS (Operating System)

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 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.

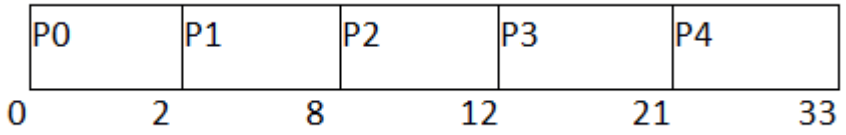
The Turnaround time and the waiting time are calculated by using the following formula.

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

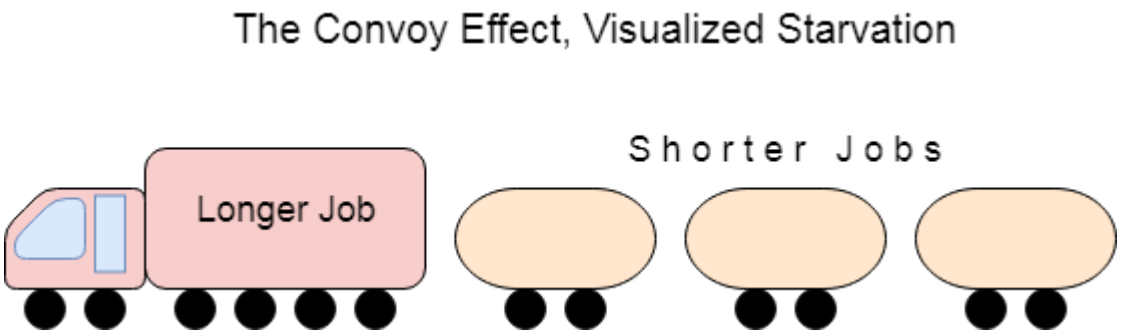


(Gantt chart)

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.

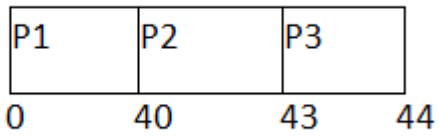
The Turnaround time and the waiting time in the following table, are calculated by the formula,

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



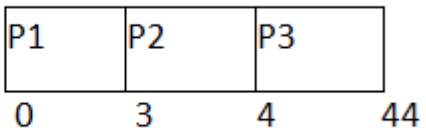
Avg waiting Time = 81/3

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

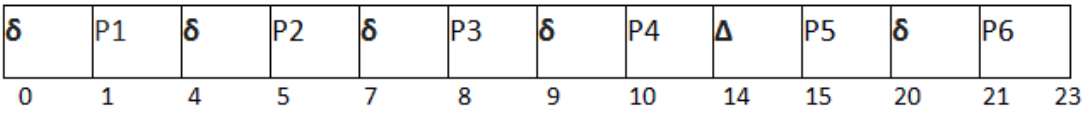
In the following Example, we are considering five processes P1, P2, P3, P4, P5 and P6. Their arrival time and Burst time are given below.

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. Efficiency = (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 Time
	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).

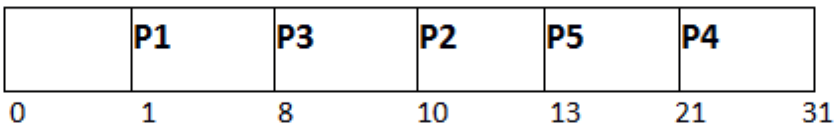
According to the algorithm, the OS schedules the process which is having the lowest burst time among the available processes in the ready queue.

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,

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

Where, 0<=i<=n and ∑ 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,

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

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

Shortest Remaining Time First (SRTF) Scheduling Algorithm

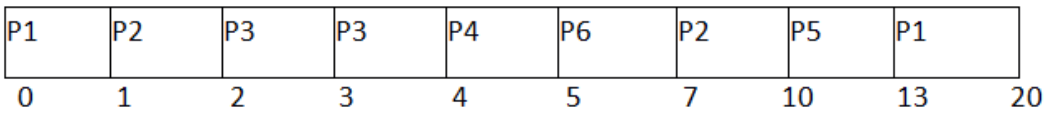
This Algorithm is the **preemptive version** of **SJF scheduling**. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes 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.
5. 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.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.

SRTF GATE 2011 Example

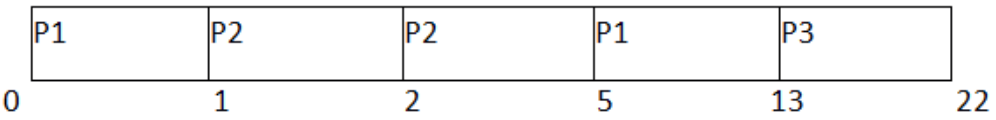
If we talk about scheduling algorithm from the GATE point of view, they generally ask simple numerical questions about finding the average waiting time and Turnaround Time. Let's discuss the question asked in GATE 2011 on SRTF.

Q. Given the arrival time and burst time of 3 jobs in the table below. Calculate the Average waiting time of the system.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	0	9	13	13	4
2	1	4	5	4	0
3	2	9	22	20	11

There are three jobs P1, P2 and P3. P1 arrives at time unit 0; it will be scheduled first for the time until the next process arrives. P2 arrives at 1 unit of time. Its burst time is 4 units which is least among the jobs in the queue. Hence it will be scheduled next.

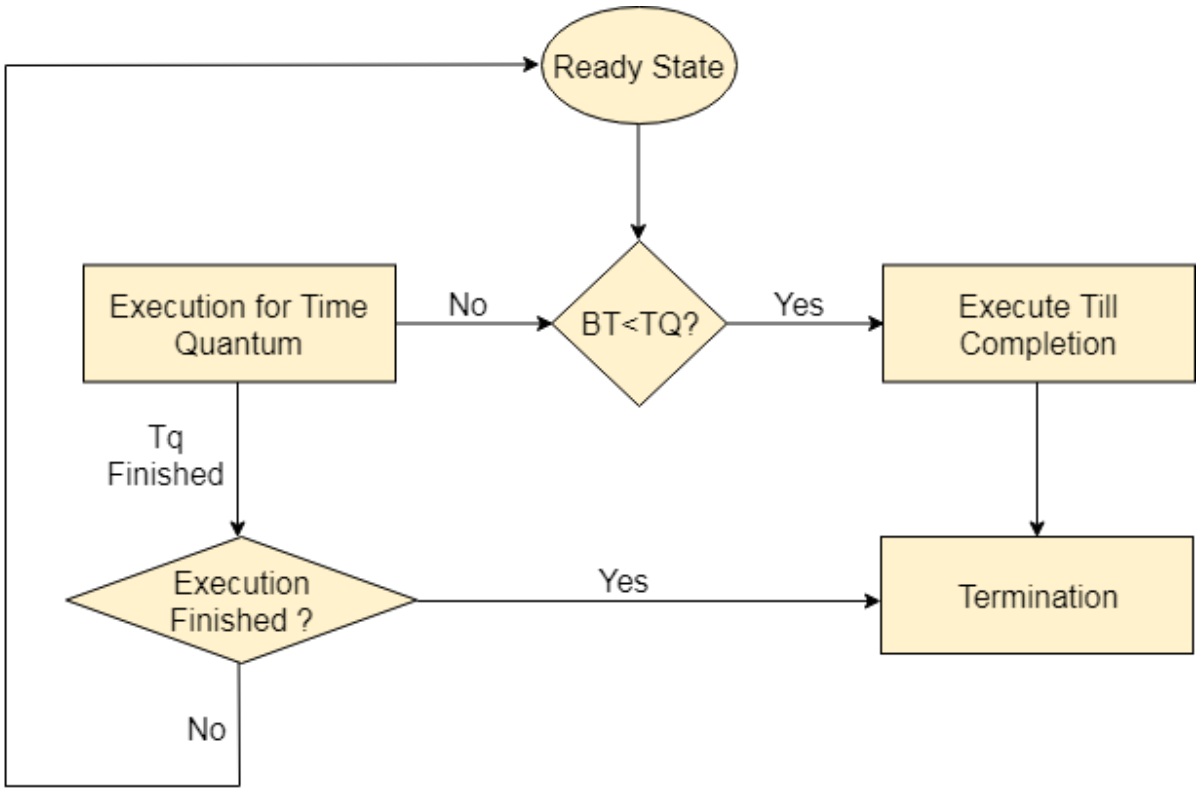
At time 2, P3 will arrive with burst time 9. Since remaining burst time of P2 is 3 units which are least among the available jobs. Hence the processor will continue its execution till its completion. Because all the jobs have been arrived so no preemption will be done now and all the jobs will be executed till the completion according to SJF.



Avg Waiting Time = (4+0+11)/3 = 5 units

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.

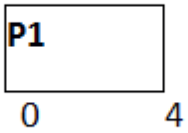
Ready Queue:

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.



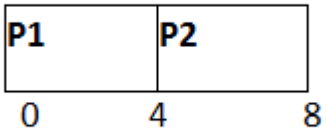
Ready Queue

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.



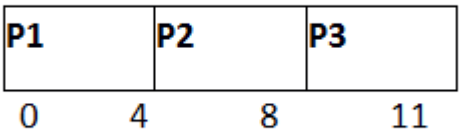
Ready Queue

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.



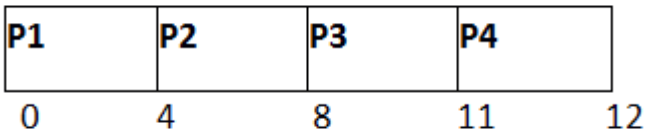
Ready Queue

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.



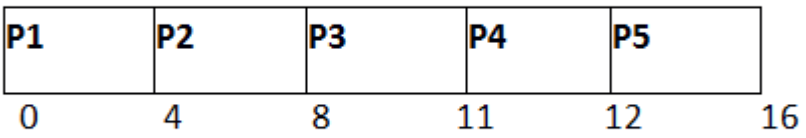
Ready Queue

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.



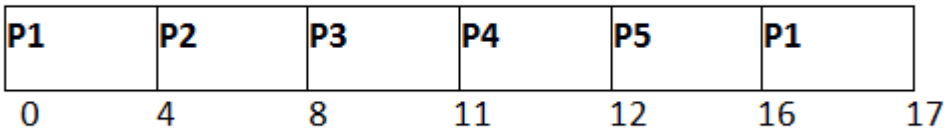
Ready Queue

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.



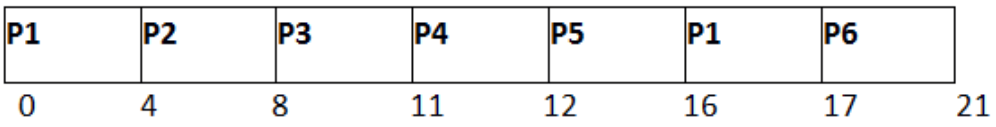
Ready Queue

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.



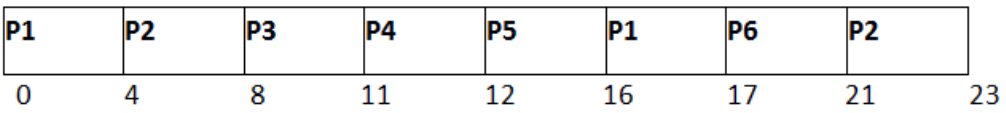
Ready Queue

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.



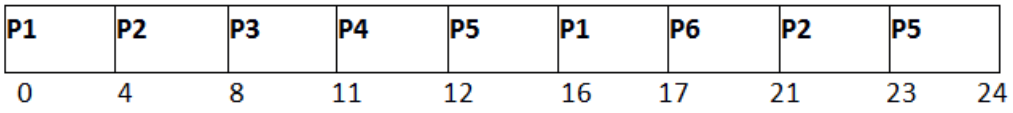
Ready Queue

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.

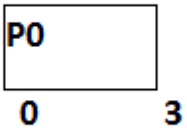
HRNN Example

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

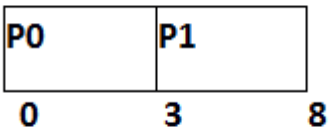
Process ID	Arrival Time	Burst Time
0	0	3
1	2	5
2	4	4
3	6	1

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.



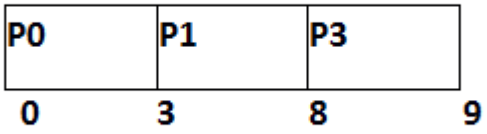
P0 is executed for 3 units, meanwhile, only one process P1 arrives at time 3. This will get scheduled immediately since the OS doesn't have a choice.



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

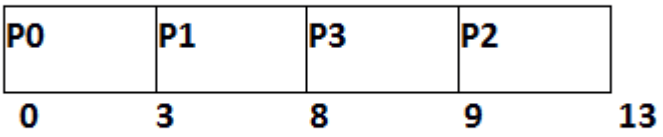
Since, the Response ratio of P3 is higher hence P3 will be scheduled first.



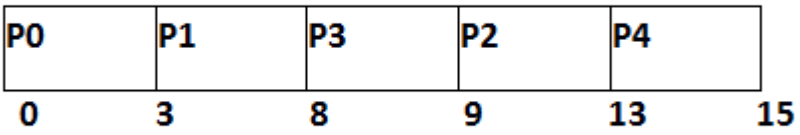
P3 is scheduled for 1 unit. The next available processes are P2 and P4. Let's calculate their Response ratio.

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.

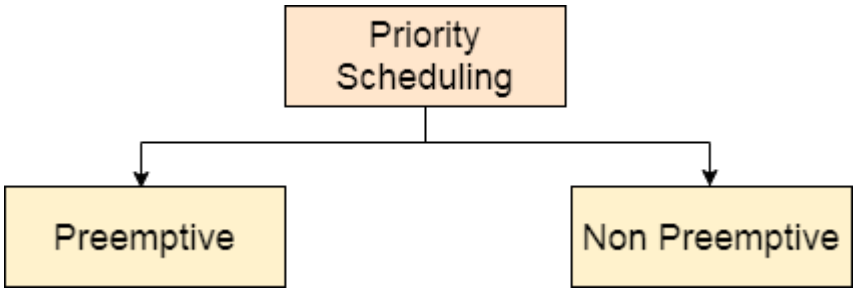


Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
0	0	3	3	3	0
1	2	5	8	6	1
2	4	4	13	9	5
3	6	1	9	3	2
4	8	2	15	7	5

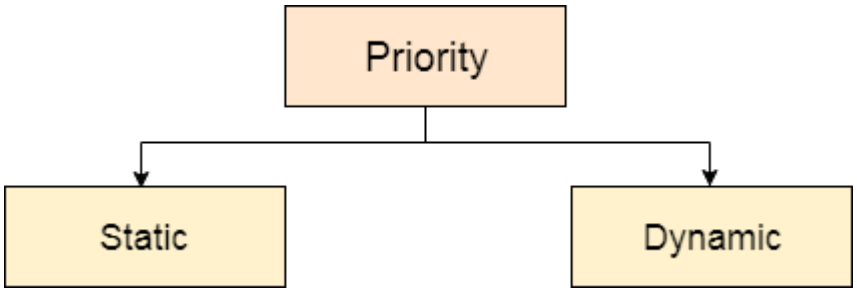
Average Waiting Time = 13/5

Priority Scheduling Algorithm in OS (Operating System)

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**.



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.

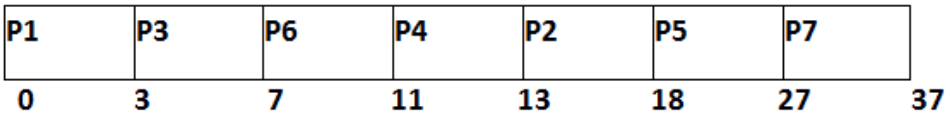
The Process P1 arrives at time 0 with the burst time of 3 units and the priority number 2. Since No other process has arrived till now hence the OS will schedule it immediately.

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	Turnaround 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

Preemptive Priority Scheduling

In Preemptive Priority Scheduling, at the time of arrival of a process in the ready queue, its Priority is compared with the priority of the other processes present in the ready queue as well as with the one which is being executed by the CPU at that point of time. The One with the highest priority among all the available processes will be given the CPU next.

The difference between preemptive priority scheduling and non preemptive priority scheduling is that, in the preemptive priority scheduling, the job which is being executed can be stopped at the arrival of a higher priority job.

Once all the jobs get available in the ready queue, the algorithm will behave as non-preemptive priority scheduling, which means the job scheduled will run till the completion and no preemption will be done.

Example

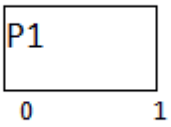
There are 7 processes P1, P2, P3, P4, P5, P6 and P7 given. Their respective priorities, Arrival Times and Burst times are given in the table below.

Process Id	Priority	Arrival Time	Burst Time
1	2(L)	0	1
2	6	1	7
3	3	2	3
4	5	3	6
5	4	4	5
6	10(H)	5	15

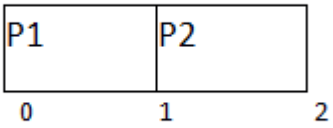
7	9	15	8

GANTT chart Preparation

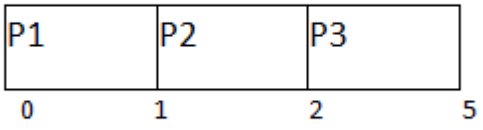
At time 0, P1 arrives with the burst time of 1 units and priority 2. Since no other process is available hence this will be scheduled till next job arrives or its completion (whichever is lesser).



At time 1, P2 arrives. P1 has completed its execution and no other process is available at this time hence the Operating system has to schedule it regardless of the priority assigned to it.

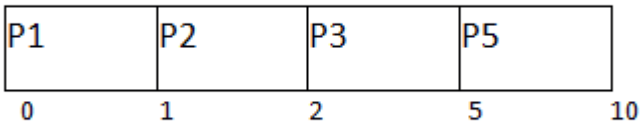


The Next process P3 arrives at time unit 2, the priority of P3 is higher to P2. Hence the execution of P2 will be stopped and P3 will be scheduled on the CPU.

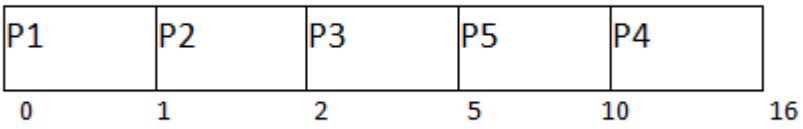


During the execution of P3, three more processes P4, P5 and P6 becomes available. Since, all these three have the priority lower to the process in execution so PS can't

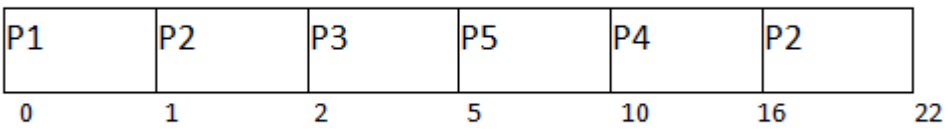
preempt the process. P3 will complete its execution and then P5 will be scheduled with the priority highest among the available processes.



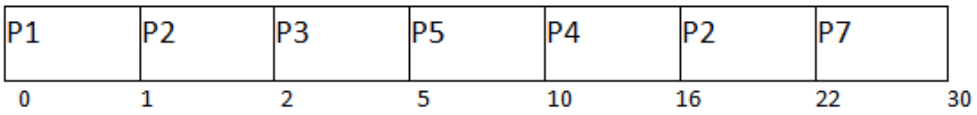
Meanwhile the execution of P5, all the processes got available in the ready queue. At this point, the algorithm will start behaving as Non Preemptive Priority Scheduling. Hence now, once all the processes get available in the ready queue, the OS just took the process with the highest priority and execute that process till completion. In this case, P4 will be scheduled and will be executed till the completion.



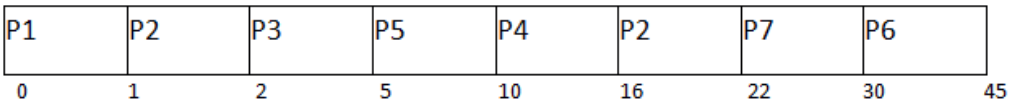
Since P4 is completed, the other process with the highest priority available in the ready queue is P2. Hence P2 will be scheduled next.



P2 is given the CPU till the completion. Since its remaining burst time is 6 units hence P7 will be scheduled after this.



The only remaining process is P6 with the least priority, the Operating System has no choice unless of executing it. This will be executed at the last.



The Completion Time of each process is determined with the help of GANTT chart. The turnaround time and the waiting time can be calculated by the following formula.

1. Turnaround 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
1	2	0	1	1	1	0
2	6	1	7	22	21	14
3	3	2	3	5	3	0
4	5	3	6	16	13	7
5	4	4	5	10	6	1
6	10	5	15	45	40	25

7	9	6	8	30	24	16

Avg Waiting Time = (0+14+0+7+1+25+16)/7 = 63/7 = 9 units

SRTF with Processes contains CPU and IO Time

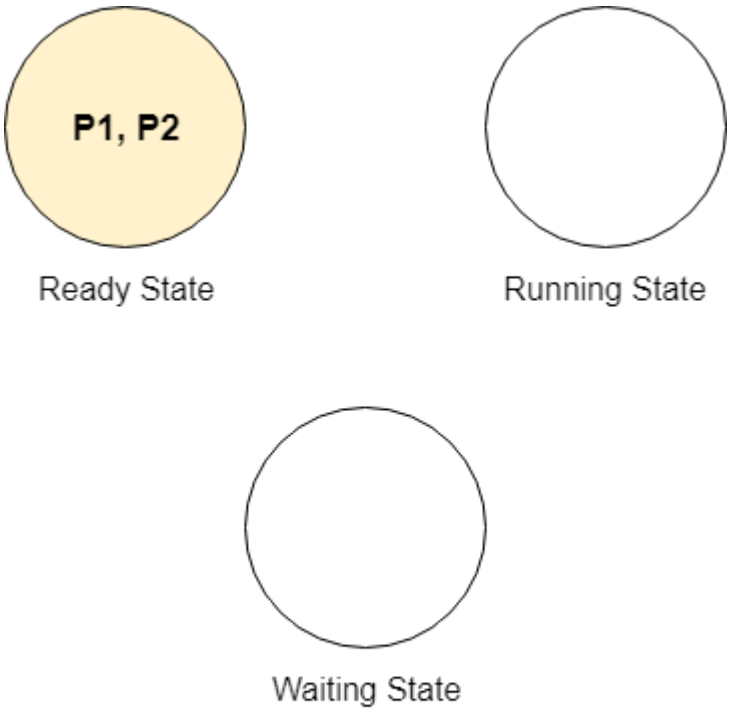
Till now, we were considering the CPU bound jobs only. However, the process might need some IO operation or some resource to complete its execution. In this Example, we are considering, the IO bound processes.

In the Example, there are four jobs with process ID P1, P2, P3 and P4 are available. Their Arrival Time, and the CPU Burst time are given in the table below.

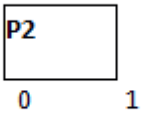
Process Id	Arrival Time	(Burst Time, IO Burst Time, Burst Time)
1	0	(3,2,2)
2	0	(1,3,1)
3	3	(3,1,2)
4	6	(5,4,5)

GANTT Chart Preparation

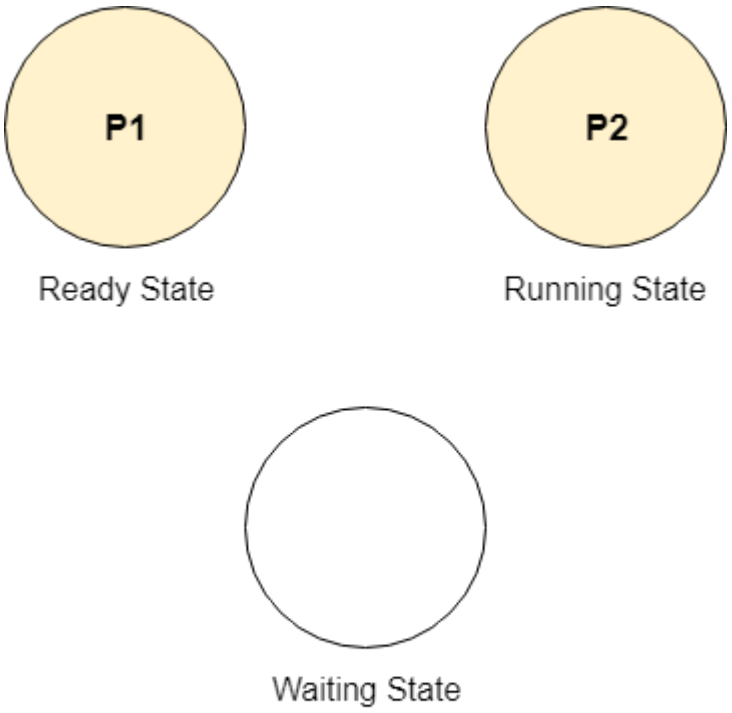
At time 0, the process P1 and P2 arrives. Since the algorithm we are using is SRTF hence, the process with the shortest burst time will be scheduled on the CPU. In this case, it is P2.



Before Execution



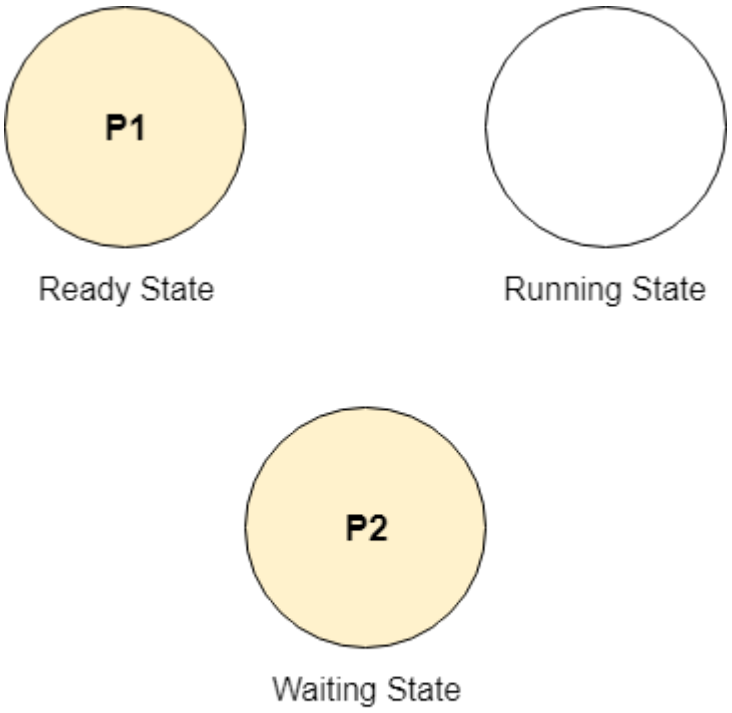
From time 0 to time 1, P2 will be in running state.



Time 0 to Time 1

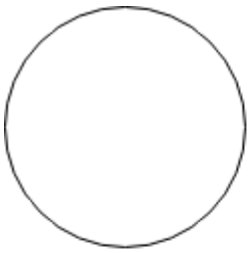
P2 also needs some IO time in order to complete its execution. After 1 unit of execution, P2 will change its state from running to waiting. The processor becomes free to execute other jobs. Since No other process is available at this point of time other than P1 so P1 will get executed.

The following diagram illustrates the processes and states at Time 1. The process P2 went to waiting state and the CPU becomes idol at this time.

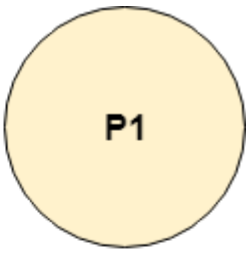


At Time 1

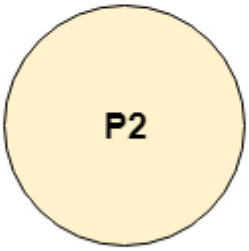
From time 1 to 3, since P2 is being in waiting state, and no other process is available in ready queue, hence the only available process P1 will be executed in this period of time.



Ready State

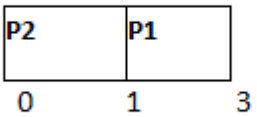


Running State

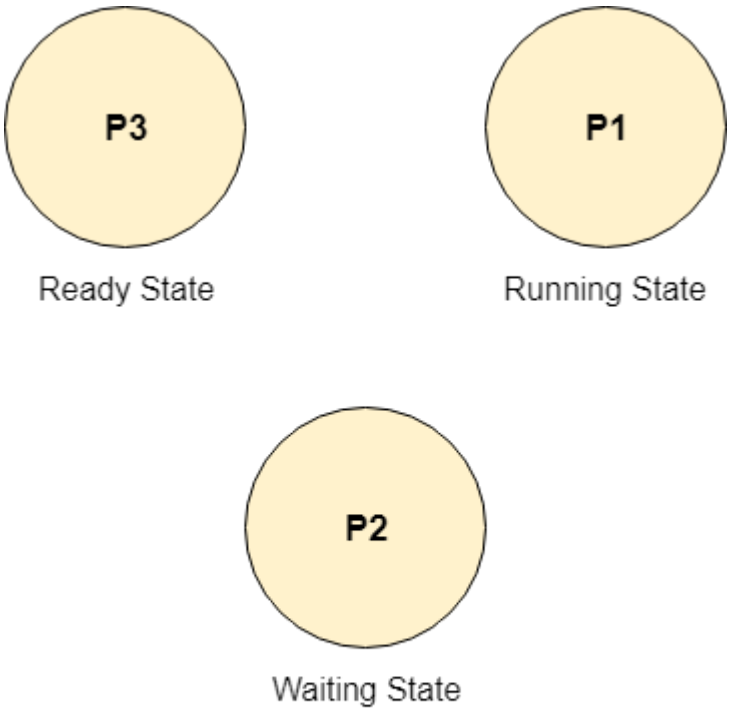


Waiting State

Time 1 to Time 3

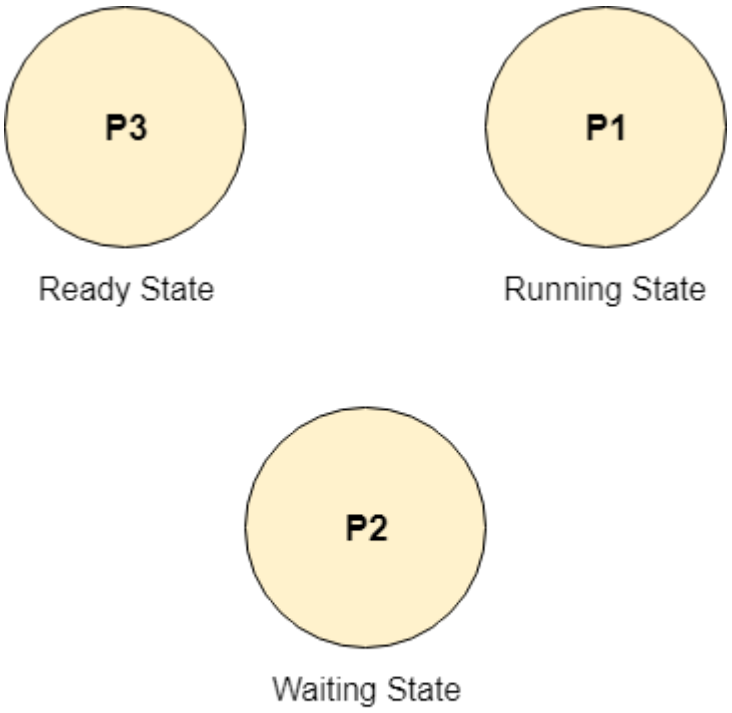


At time 3, the process P3 arrived with the total CPU burst time of 5 units. Since the remaining burst time of P1 is lesser then P3 hence CPU will continue its execution.

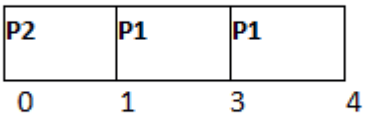


At Time 3

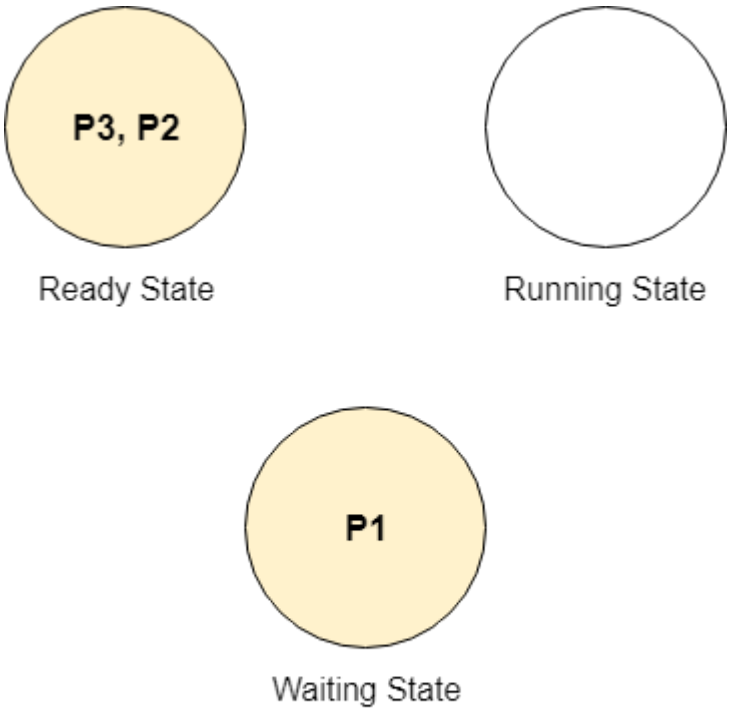
Hence, P1 will remain in the running state from time 3 to time 4.



Time 3 to Time 4

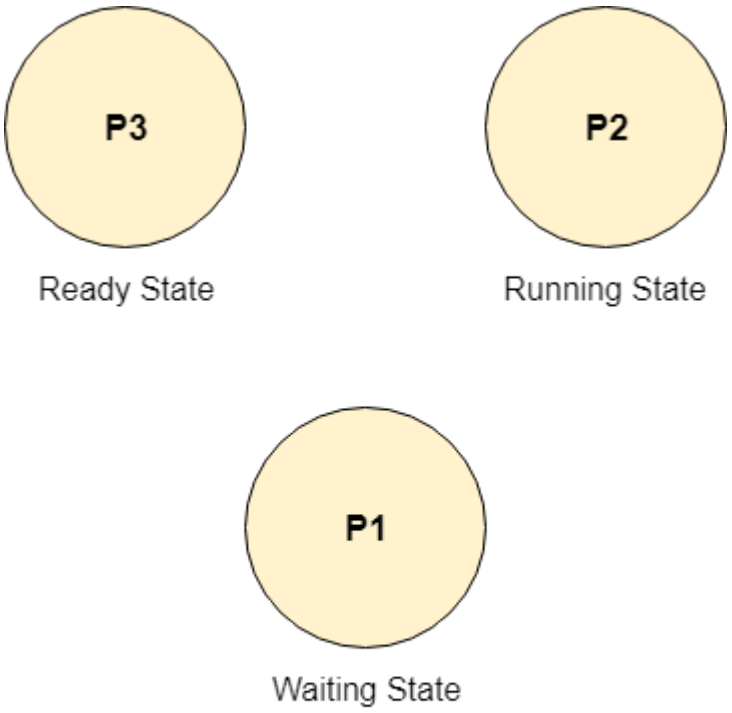


Since P1 is an IO bound process. At time unit 4, it will change its state from running to waiting. Processor becomes free for the execution of other jobs. Since P2 also becomes available at time 4 because it has completed the IO operation and now it needs another 1 unit of CPU burst time. P3 is also available and requires 5 units of total CPU burst time.

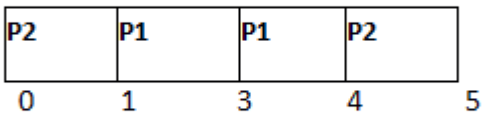


At Time 4

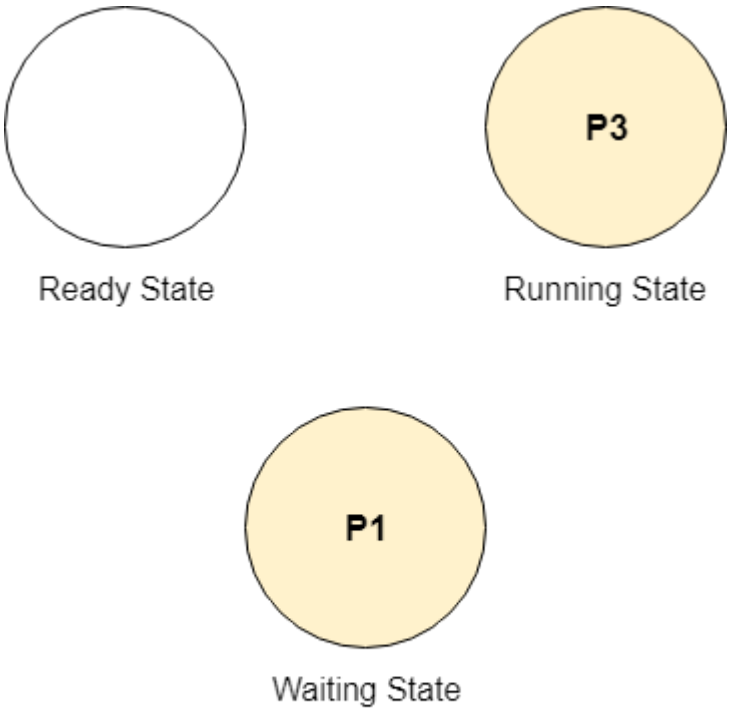
The process with the least remaining CPU burst time among the available processes will get executed. In our case, such process is P2 which requires 1 unit of burst time hence it will be given the CPU.



Time 4 to Time 5



At time 5, P2 is finished. P1 is still in waiting state. At this point of time, the only available process is P3, hence it will be given the CPU.

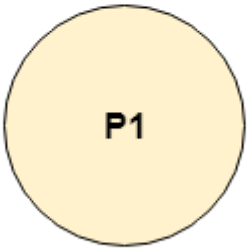


At Time 5

From Time 5 to time 6, P3 will be in the running state; meanwhile, P1 will still be in waiting state.

Ready State

Running State

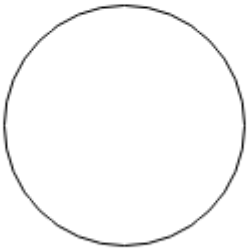


Waiting State

Time 5 to Time 6

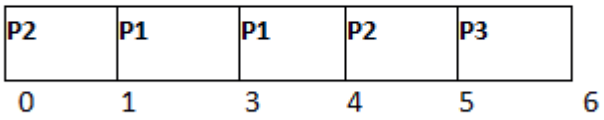
Ready State

Running State



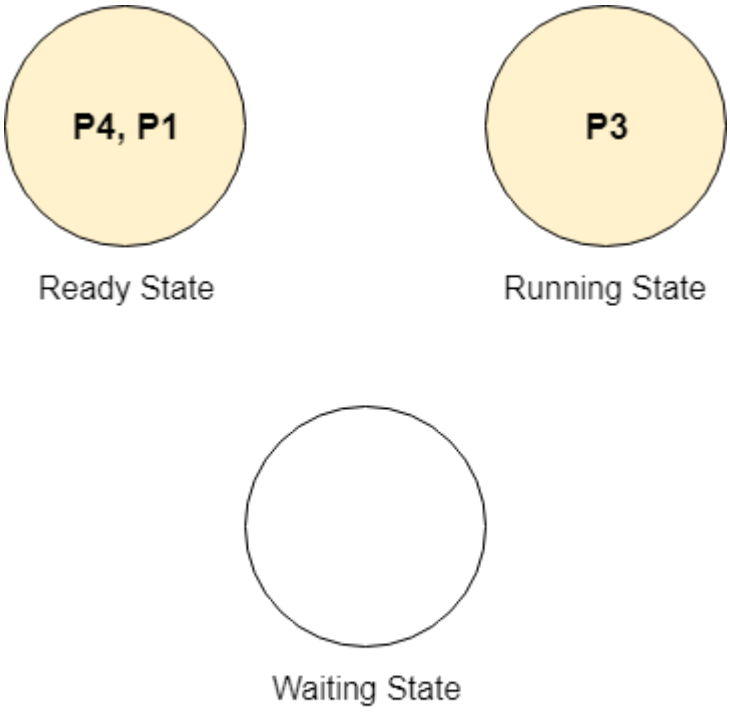
Waiting State

At Time 6

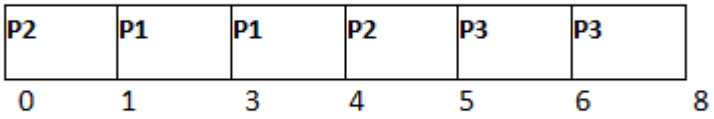


At time 6, the Process P4 arrives in the ready queue. The P1 has also done with the IO and becomes available for the execution. P3 is not yet finished and still needs another 2 unit of CPU burst time.

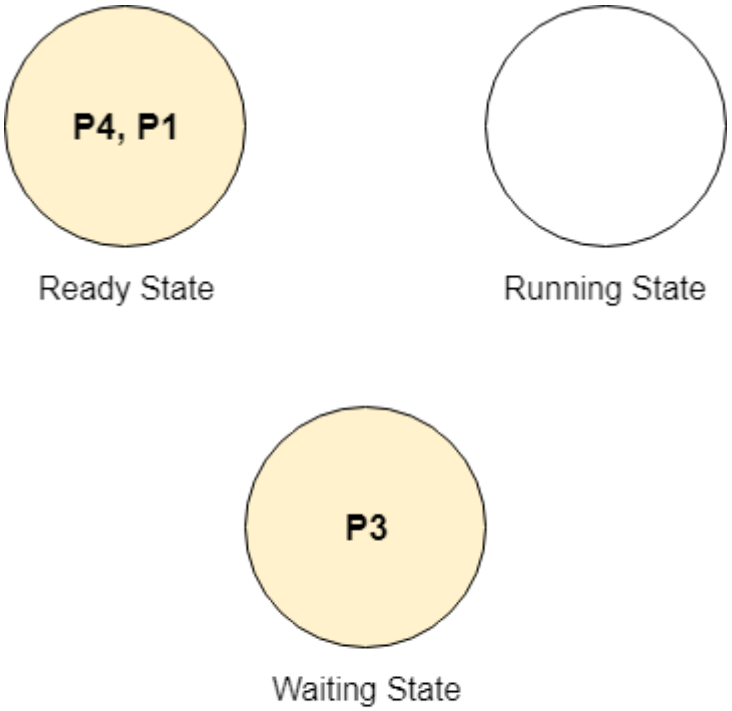
From time 6 to time 8, the reaming CPU burst time of Process P3 is least among the available processes, hence P3 will be given the CPU.



Time 6 to Time 8



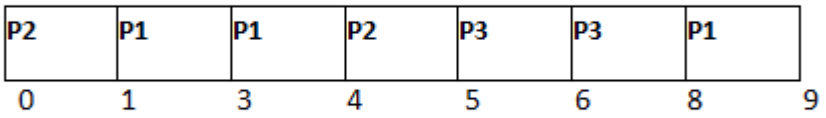
P3 needs some IO operation in order to complete its execution. At time 8, P3 will change its state from running to waiting. The CPU becomes free to execute the other processes. Process P4 and P1 are available out of which, the process with the least remaining burst time will get executed.



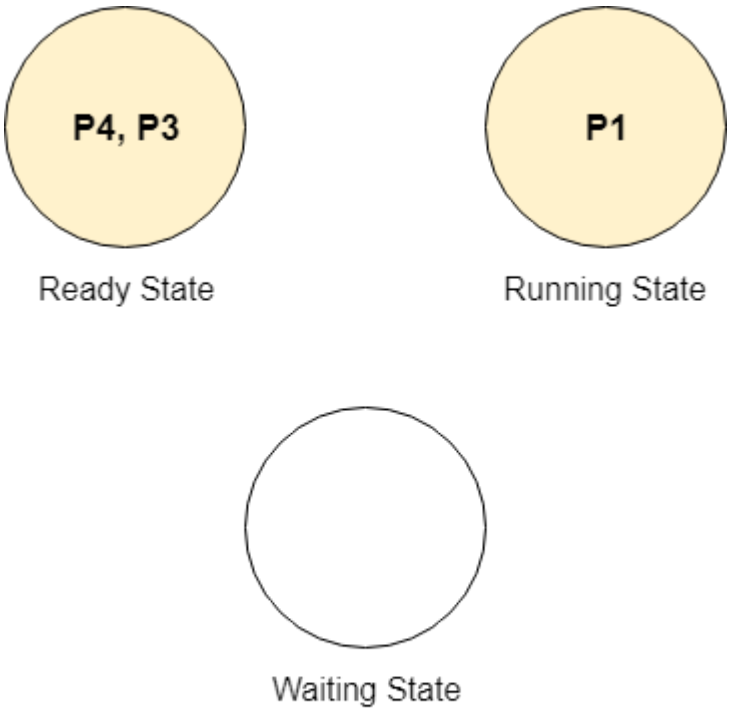
At Time 8

From time 8 to time 9, the process P1 will get executed.

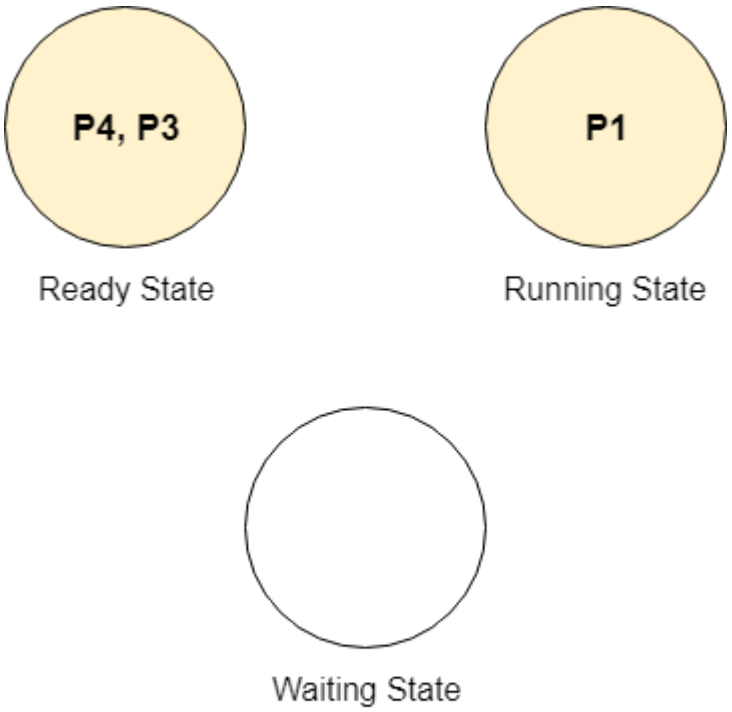
Time 8 to Time 9



At time 9, the IO of process P3 is finished and it will now be available in the ready state along with P4 which is already waiting there for its turn. In order to complete its execution, it needs another 2 unit of burst time. P1 is in running state at this point of the time while no process is present in the waiting state.

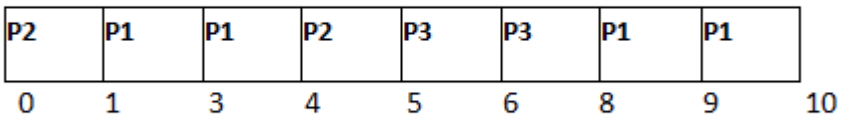


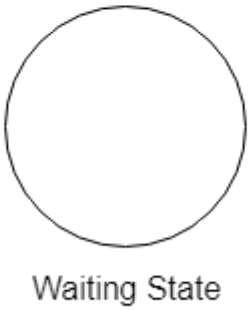
At Time 9



Time 9 to Time 10

from time 9 to 10 , the process P1 will get executed since its remaining CPU burst time is lesser then the processes P4 and P3 available in the ready queue.

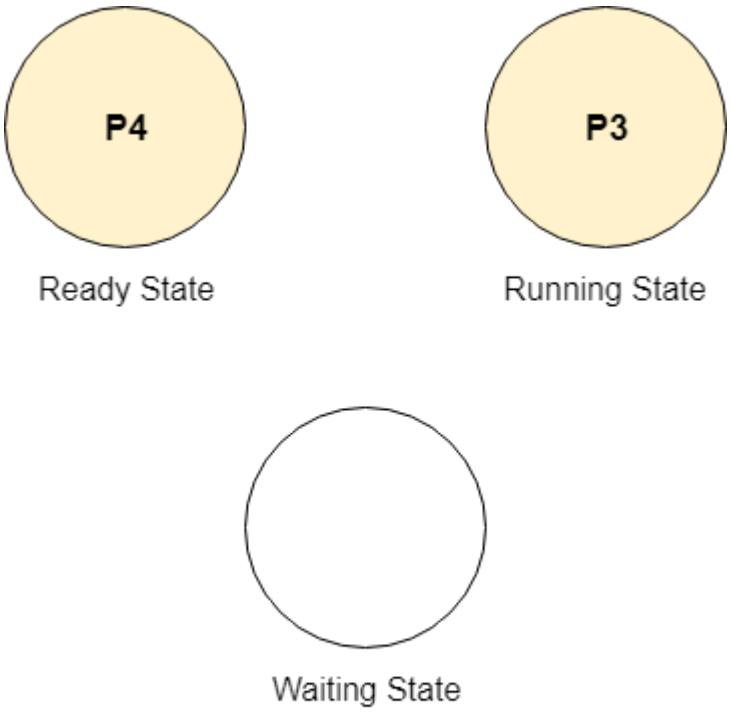




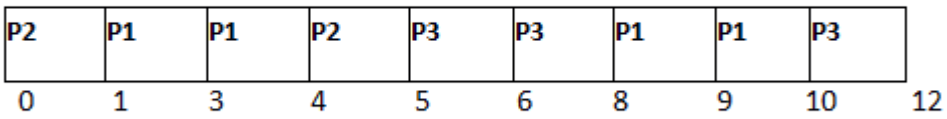
At Time 10

At time 10, execution of P1 is finished, and now the CPU becomes idol. The process with the lesser CPU burst time among the ready processes will get the CPU turn.

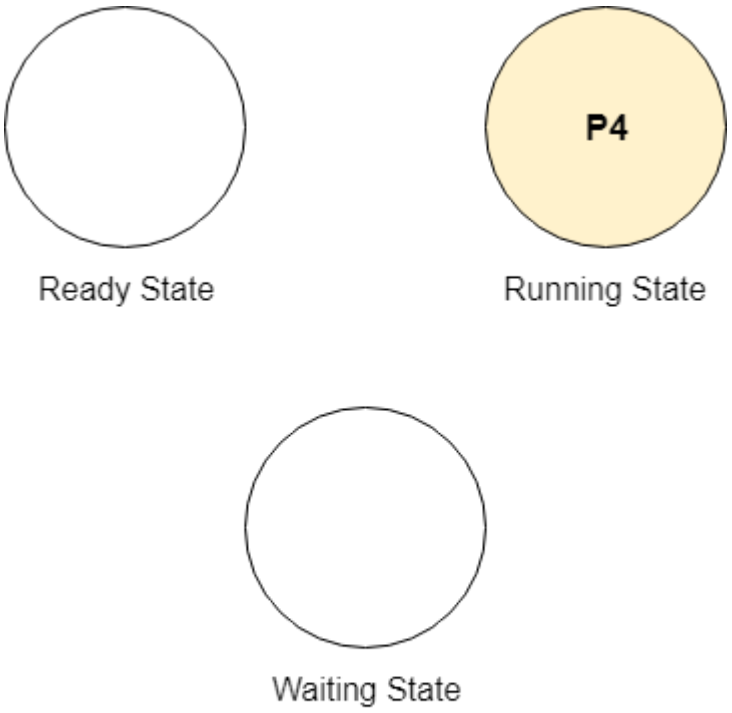
From time 10 to 12, the process P3 will get executed till its completion because of the fact that its remaining CPU burst time is the between the two available processes. It needs 2 units of more CPU burst time, since No other process will be arrived in the ready state hence No preemption will be done and it will be executed till the completion.



Time 10 to Time 12

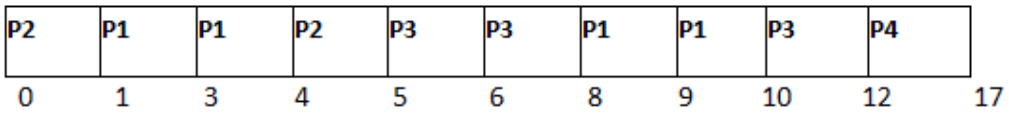


At time 12, the process P3 will get completed, since there is only one process P4 available in the ready state hence P4 will be given the CPU.

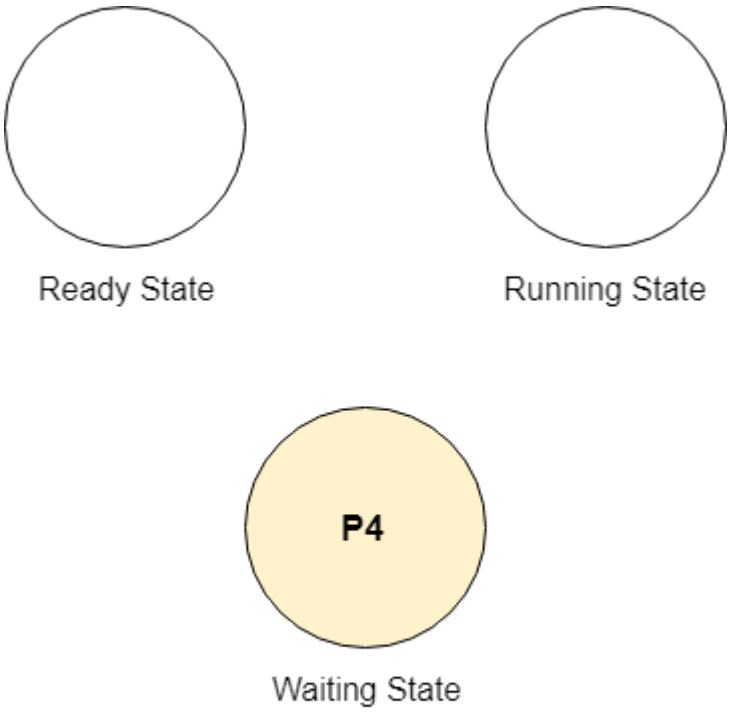


Time 12 to Time 17

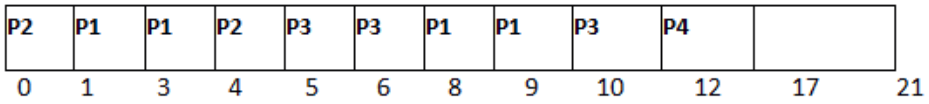
P4 needs 5 units of CPU burst time before IO, hence it will be executed till time 17 (for 5 units) and then it will change its state from running to waiting.



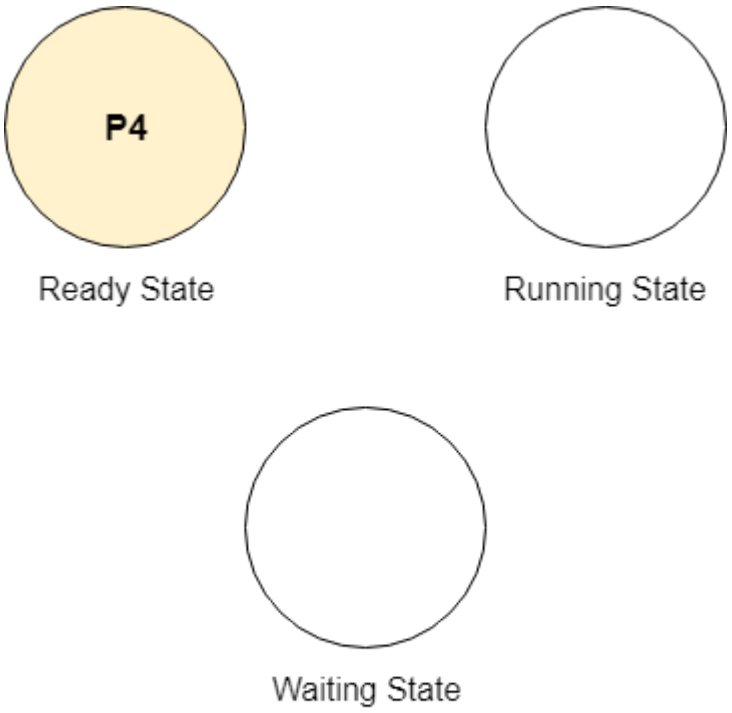
At time 17, the Process P4 changes its state from running to waiting. Since this is the only process in the system hence the CPU will remain idol until P4 becomes available again.



Time 17 to Time 21

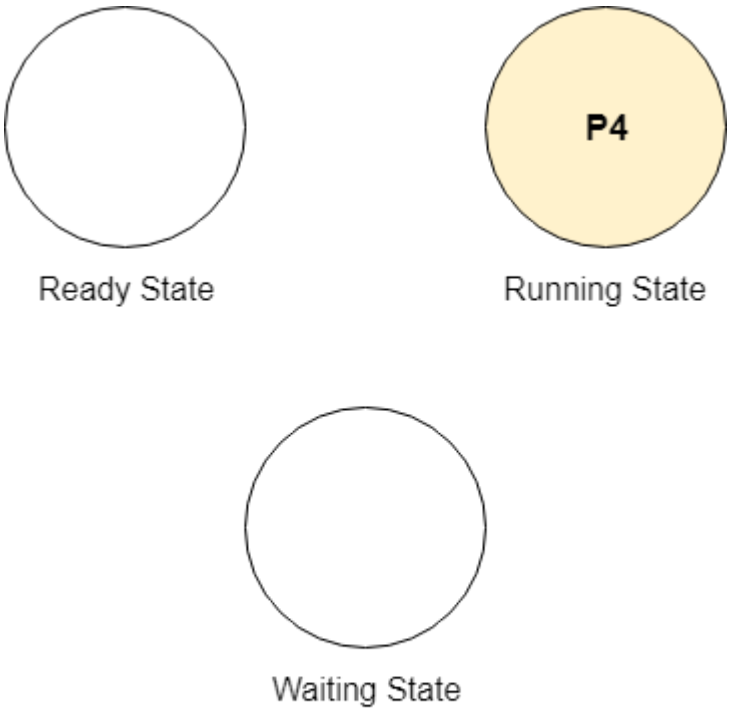


At time 21, P4 will be done with the IO operation and becomes available in the ready state.



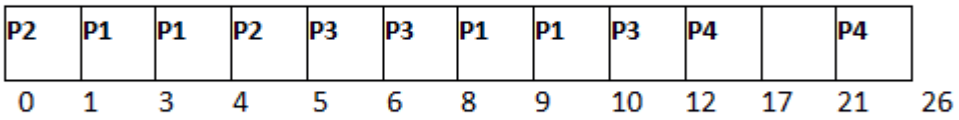
At Time 21

From time 21, the process P4 will get scheduled. Since No other process is in ready queue hence the processor don't have any choice. It will be executed till completion.



Time 21 to Time 26

Final Gantt chart:



Process Id	Arrival Time	Total CPU Burst Time	Completion Time	Turn Around Time	Waiting Time
1	0	5	10	10	5
2	0	2	5	5	3
3	3	5	12	9	4

4	6	10	26	20	10

Average waiting Time = (5+3+4+10)/4 = 22/4 units

Process Synchronization in OS (Operating System)

When two or more process cooperates with each other, their order of execution must be preserved otherwise there can be conflicts in their execution and inappropriate outputs can be produced.

A cooperative process is the one which can affect the execution of other process or can be affected by the execution of other process. Such processes need to be synchronized so that their order of execution can be guaranteed.

The procedure involved in preserving the appropriate order of execution of cooperative processes is known as Process Synchronization. There are various synchronization mechanisms that are used to synchronize the processes.

Race Condition

A Race Condition typically occurs when two or more threads try to read, write and possibly make the decisions based on the memory that they are accessing concurrently.

Critical Section

The regions of a program that try to access shared resources and may cause race conditions are called critical section. To avoid race condition among the processes, we need to assure that only one process at a time can execute within the critical section.

Critical Section Problem in OS (Operating System)

Critical Section is the part of a program which tries to access shared resources. That resource may be any resource in a computer like a memory location, Data structure, CPU or any IO device.

The critical section cannot be executed by more than one process at the same time; operating system faces the difficulties in allowing and disallowing the processes from entering the critical section.

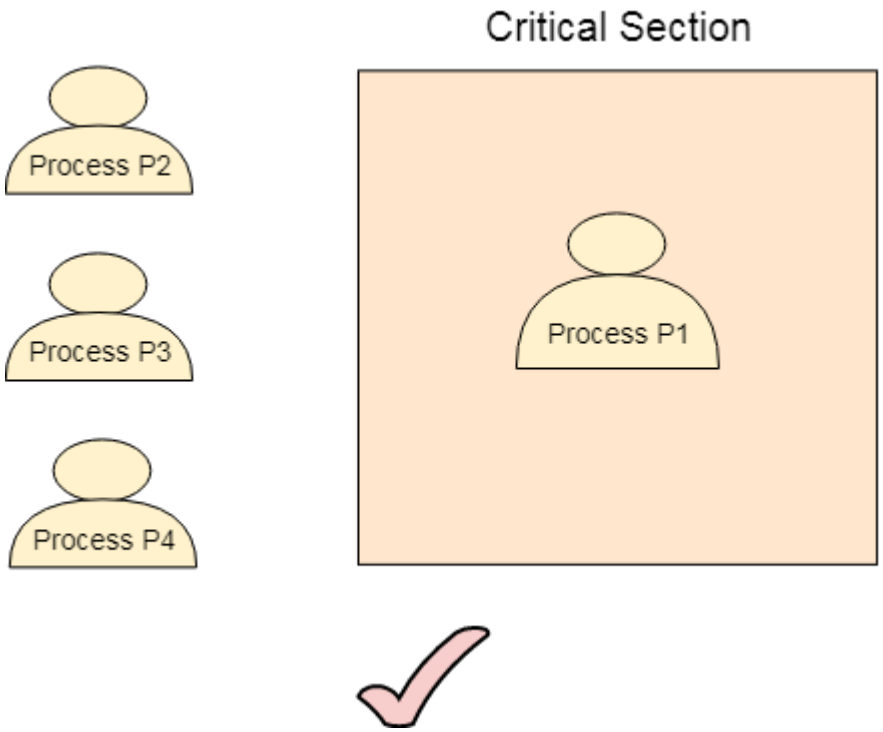
The critical section problem is used to design a set of protocols which can ensure that the Race condition among the processes will never arise.

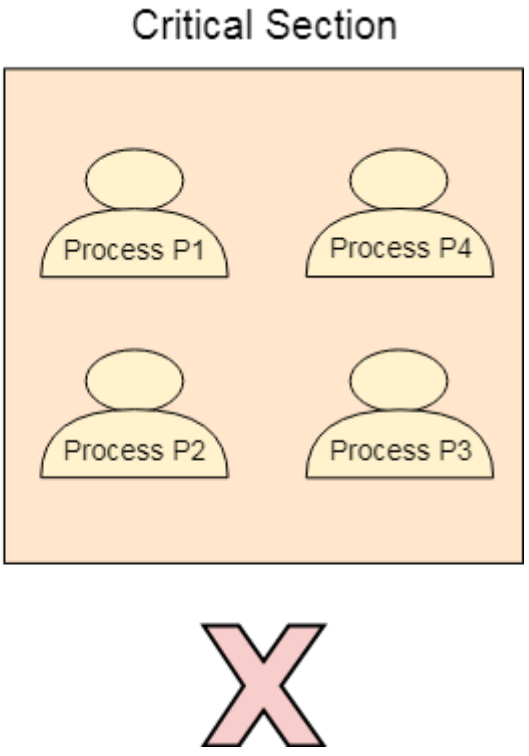
In order to synchronize the cooperative processes, our main task is to solve the critical section problem. We need to provide a solution in such a way that the following conditions can be satisfied.

Requirements of Synchronization mechanisms

Primary

1. **Mutual Exclusion**
2. Our solution must provide mutual exclusion. By Mutual Exclusion, we mean that if one process is executing inside critical section then the other process must not enter in the critical section.





3. **Progress**

Progress means that if one process doesn't need to execute into critical section then it should not stop other processes to get into the critical section.

Secondary

1. **Bounded Waiting :** We should be able to predict the waiting time for every process to get into the critical section. The process must not be endlessly waiting for getting into the critical section.
2. **Architectural Neutrality :** Our mechanism must be architectural natural. It means that if our solution is working fine on one architecture then it should also run on the other ones as well.

Lock Variable

This is the simplest synchronization mechanism. This is a Software Mechanism implemented in User mode. This is a busy waiting solution which can be used for more than two processes.

In this mechanism, a Lock variable **lock**is used. Two values of lock can be possible, either 0 or 1. Lock value 0 means that the critical section is vacant while the lock value 1 means that it is occupied.

A process which wants to get into the critical section first checks the value of the lock variable. If it is 0 then it sets the value of lock as 1 and enters into the critical section, otherwise it waits.

The pseudo code of the mechanism looks like following.

1. Entry Section →
2. While (lock! = 0);
3. Lock = 1;
4. //Critical Section
5. Exit Section →
6. Lock =0;

If we look at the Pseudo Code, we find that there are three sections in the code. Entry Section, Critical Section and the exit section.

Initially the value of **lock variable** is **0**. The process which needs to get into the **critical section**, enters into the entry section and checks the condition provided in the while loop.

The process will wait infinitely until the value of **lock** is 1 (that is implied by while loop). Since, at the very first time critical section is vacant hence the process will enter the critical section by setting the lock variable as 1.

When the process exits from the critical section, then in the exit section, it reassigns the value of **lock** as 0.

Every Synchronization mechanism is judged on the basis of four conditions.

1. Mutual Exclusion
2. Progress
3. Bounded Waiting
4. Portability

Out of the four parameters, Mutual Exclusion and Progress must be provided by any solution. Let?s analyze this mechanism on the basis of the above mentioned conditions.

Mutual Exclusion

The lock variable mechanism doesn't provide Mutual Exclusion in some of the cases. This can be better described by looking at the pseudo code by the Operating System point of view I.E. Assembly code of the program. Let's convert the Code into the assembly language.

1. *Load Lock, R0*
2. *CMP R0, #0*
3. *JNZ Step 1*
4. *Store #1, Lock*
5. *Store #0, Lock*

Let us consider that we have two processes P1 and P2. The process P1 wants to execute its critical section. P1 gets into the entry section. Since the value of lock is 0 hence P1 changes its value from 0 to 1 and enters into the critical section.

Meanwhile, P1 is preempted by the CPU and P2 gets scheduled. Now there is no other process in the critical section and the value of lock variable is 0. P2 also wants to execute its critical section. It enters into the critical section by setting the lock variable to 1.

Now, CPU changes P1's state from waiting to running. P1 is yet to finish its critical section. P1 has already checked the value of lock variable and remembers that its value was 0 when it previously checked it. Hence, it also enters into the critical section without checking the updated value of lock variable.

Now, we got two processes in the critical section. According to the condition of mutual exclusion, morethan one process in the critical section must not be present at the same time. Hence, the lock variable mechanism doesn't guarantee the mutual exclusion.

The problem with the lock variable mechanism is that, at the same time, more than one process can see the vacant tag and more than one process can enter in the critical section. Hence, the lock variable doesn't provide the mutual exclusion that's why it cannot be used in general.

Since, this method is failed at the basic step; hence, there is no need to talk about the other conditions to be fulfilled.

Test Set Lock Mechanism

Modification in the assembly code

In lock variable mechanism, Sometimes Process reads the old value of lock variable and enters the critical section. Due to this reason, more than one process might get into critical section. However, the code shown in the part one of the following section can be replaced with the code shown in the part two. This doesn't affect the algorithm but, by doing this, we can manage to provide the mutual exclusion to some extent but not completely.

In the updated version of code, the value of Lock is loaded into the local register R0 and then value of lock is set to 1.

However, in step 3, the previous value of lock (that is now stored into R0) is compared with 0. if this is 0 then the process will simply enter into the critical section otherwise will wait by executing continuously in the loop.

The benefit of setting the lock immediately to 1 by the process itself is that, now the process which enters into the critical section carries the updated value of lock variable that is 1.

In the case when it gets preempted and scheduled again then also it will not enter the critical section regardless of the current value of the lock variable as it already knows what the updated value of lock variable is.

Section 1	Section 2
1. Load Lock, R0 2. CMP R0, #0 3. JNZ step1 4. store #1, Lock	1. Load Lock, R0 2. Store #1, Lock 3. CMP R0, #0 4. JNZ step 1

TSL Instruction

However, the solution provided in the above segment provides mutual exclusion to some extent but it doesn't make sure that the mutual exclusion will always be there. There is a possibility of having more than one process in the critical section.

What if the process gets preempted just after executing the first instruction of the assembly code written in section 2? In that case, it will carry the old value of lock variable with it and it will enter into the critical section regardless of knowing the current value of lock variable. This may make the two processes present in the critical section at the same time.

To get rid of this problem, we have to make sure that the preemption must not take place just after loading the previous value of lock variable and before setting it to 1. The problem can be solved if we can be able to merge the first two instructions.

In order to address the problem, the operating system provides a special instruction called **Test Set Lock (TSL)** instruction which simply loads the value of lock variable into the local register R0 and sets it to 1 simultaneously

The process which executes the TSL first will enter into the critical section and no other process after that can enter until the first process comes out. No process can execute the critical section even in the case of preemption of the first process.

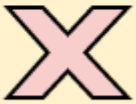
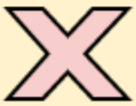
The assembly code of the solution will look like following.

1. *TSL Lock, R0*
2. *CMP R0, #0*
3. *JNZ step 1*

Let's examine TSL on the basis of the four conditions.

- **Mutual Exclusion**
- Mutual Exclusion is guaranteed in TSL mechanism since a process can never be preempted just before setting the lock variable. Only one process can see the lock variable as 0 at a particular time and that's why, the mutual exclusion is guaranteed.**Progress**
- According to the definition of the progress, a process which doesn't want to enter in the critical section should not stop other processes to get into it. In TSL mechanism, a process will execute the TSL instruction only when it wants to get into the critical section. The value of the lock will always be 0 if no process doesn't want to enter into the critical section hence the progress is always guaranteed in TSL.**Bounded Waiting**
- Bounded Waiting is not guaranteed in TSL. Some process might not get a chance for so long. We cannot predict for a process that it will definitely get a chance to enter in critical section after a certain time.**Architectural Neutrality**

TSL doesn't provide Architectural Neutrality. It depends on the hardware platform. The TSL instruction is provided by the operating system. Some platforms might not provide that. Hence it is not Architectural natural.

Mutual Exclusion	
Progress	
Bounded Waiting	
Portability	

Priority Inversion

In TSL mechanism, there can be a problem of priority inversion. Let?s say that there are two cooperative processes, P1 and P2.

The priority of P1 is 2 while that of P2 is 1. P1 arrives earlier and got scheduled by the CPU. Since it is a cooperative process and wants to execute in the critical section hence it will enter in the critical section by setting the lock variable to 1.

Now, P2 arrives in the ready queue. The priority of P2 is higher than P1 hence according to priority scheduling, P2 is scheduled and P1 got preempted. P2 is also a cooperative process and wants to execute inside the critical section.

Although, P1 got preempted but it the value of lock variable will be shown as 1 since P1 is not completed and it is yet to finish its critical section.

P1 needs to finish the critical section but according to the scheduling algorithm, CPU is with P2. P2 wants to execute in the critical section, but according to the synchronization mechanism, critical section is with P1.

This is a kind of lock where each of the process neither executes nor completes. Such kind of lock is called **Spin Lock**.

This is different from deadlock since they are not in a blocked state. One is in ready state and the other is in running state, but neither of the two is being executed.