BCA Fourth Semester

Operating Systems

Unit -3 Process Management (Part 3) [5 Hrs]

Process Scheduling

- The act of determining which process is in the **ready** state, and should be moved to the **running** state is known as **Process Scheduling**.
- The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
- **The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs**.
- For achieving this, the **scheduler** must apply appropriate rules for swapping processes IN and OUT of CPU.

Types of Scheduling

Non-Preemptive Scheduling

- **Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.**
- This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems.
- It is the only method that can be used on certain hardware platforms, because It does not require the special hardware(for example: a timer) needed for preemptive scheduling.

Preemptive Scheduling

- **In this type of Scheduling, the tasks are usually assigned with priorities**.
- At times it is necessary to run a certain task that has a higher priority before another task although it is running.
- Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

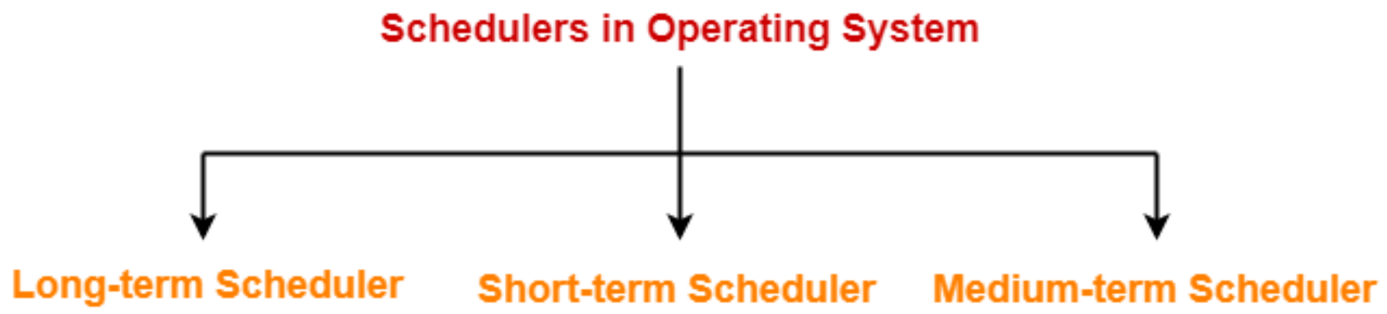
Types of Scheduling (cont.)

Batch System Scheduling (First-Come First-Served, Shortest Job First, Shortest Remaining Time Next) – works on batch mode, once assigned doesn’t need human interaction.

Interactive System Scheduling (Round-Robin Scheduling, Priority Scheduling, Multiple Queues)

Real Time System Scheduling (we will discuss later).

CPU Schedulers



1. **Long-term Scheduler-**

- **The primary objective of long-term scheduler is to maintain a good degree of multiprogramming.**
- Long-term scheduler is also known as **Job Scheduler**.
- It selects a balanced mix of I/O bound and CPU bound processes from the secondary memory (new state).
- Then, it loads the selected processes into the main memory (ready state) for execution.

CPU Schedulers (cont.)

2. Short-term Scheduler-

- **The primary objective of short-term scheduler is to increase the system performance.**
- Short-term scheduler is also known as **CPU Scheduler**.
- It decides which process to execute next from the ready queue.
- After short-term scheduler decides the process, **Dispatcher** assigns the decided process to the CPU for execution.

3. Medium-term Scheduler-

- **The primary objective of medium-term scheduler is to perform swapping**.
- Medium-term scheduler swaps-out the processes from main memory to secondary memory to free up the main memory when required.
- Thus, medium-term scheduler reduces the degree of multiprogramming.
- After some time when main memory becomes available, medium-term scheduler swaps-in the swapped-out process to the main memory and its execution is resumed from where it left off.

Comparison Of Schedulers-

Long-term scheduler	**Medium-term scheduler**	**Short-term scheduler**
It is a job scheduler	It is a process swapping scheduler.	It is a CPU scheduler
It controls the degree of multiprogramming.	It reduces the degree of multiprogramming.	It provides lesser control over degree of multiprogramming.
Speed is lesser than short-term scheduler	Speed is in between the long-term and short-term schedulers.	Speed is fastest among the other two.
It is minimal or almost absent in time sharing system.	It is a part of time sharing system.	It is also minimal in time sharing system.
It selects processes from new state and loads them into ready state.	It swaps-out processes from main memory to secondary memory and later swaps in.	It selects processes from the ready state and assigns to the CPU.
Operates less frequently since processes are not rapidly created.	Operates more frequently than long-term scheduler but less frequently than short-term scheduler.	Operates very frequently to match the speed of CPU since CPU rapidly switches from one process to another

CPU Scheduling Criteria

There are many different criteria's to check when considering the **"best"** scheduling algorithm, they are:

CPU Utilization

- To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

Throughput

- **It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time.** This may range from 10/second to 1/hour depending on the specific processes.

Turnaround Time

- **It is the amount of time taken to execute a particular process.**
- Turnaround time= Time of completion of job - Time of submission of job. (waiting time + service time or burst time)

CPU Scheduling Criteria (cont.)

Waiting Time

- The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

Load Average

- It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

Response Time

- Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

First Come First Serve (FCFS)

- FCFS is the simplest non-preemptive algorithm. Processes are assigned the CPU in the order they request it.
- That is the process that requests the CPU first is allocated the CPU first.
- The implementation of FCFS is policy is managed with a FIFO(First in first out) queue.

Advantages:

- Easy to understand and program. With this algorithm a single linked list keeps track of all ready processes.
- Equally fair.,
- Suitable specially for Batch Operating system.

Disadvantages:

- FCFS is not suitable for time-sharing systems where it is important that each user should get the CPU for an equal amount of arrival time.

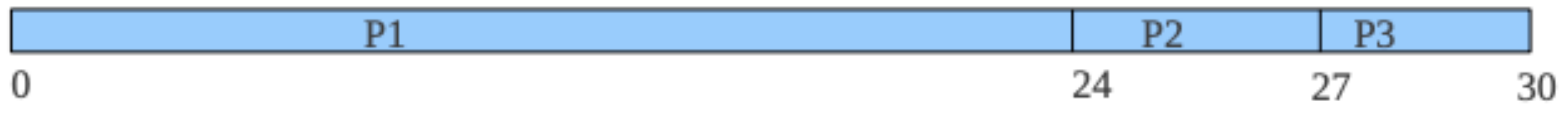
FCFS Example 1

Consider the following set of processes having their burst time mentioned in milliseconds. CPU burst time indicates that for how much time, the process needs the cpu.

Process	Burst Time
P1	24
P2	3
P3	3

Calculate the average waiting time if the processes arrive in the order of:
a). P1, P2, P3

a. The processes arrive the order P1, P2, P3. Let us assume they arrive in the same time at 0 ms in the system. We get the following gantt chart.

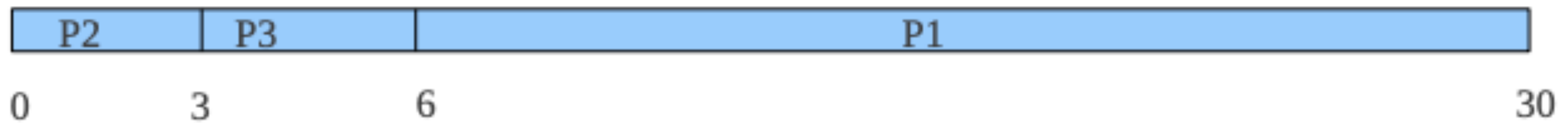


Waiting time for P1= 0ms , for P2 = 24 ms for P3 = 27ms
Avg waiting time: (0+24+27)/3= 17

FCFS Example 1

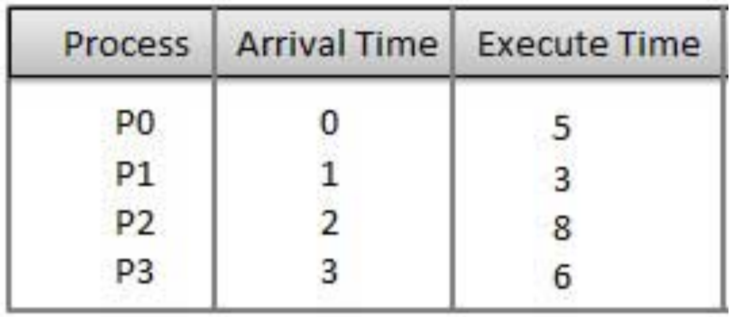
Process	Burst Time
P1	24
P2	3
P3	3

b.) If the process arrive in the order P2,P3, P1

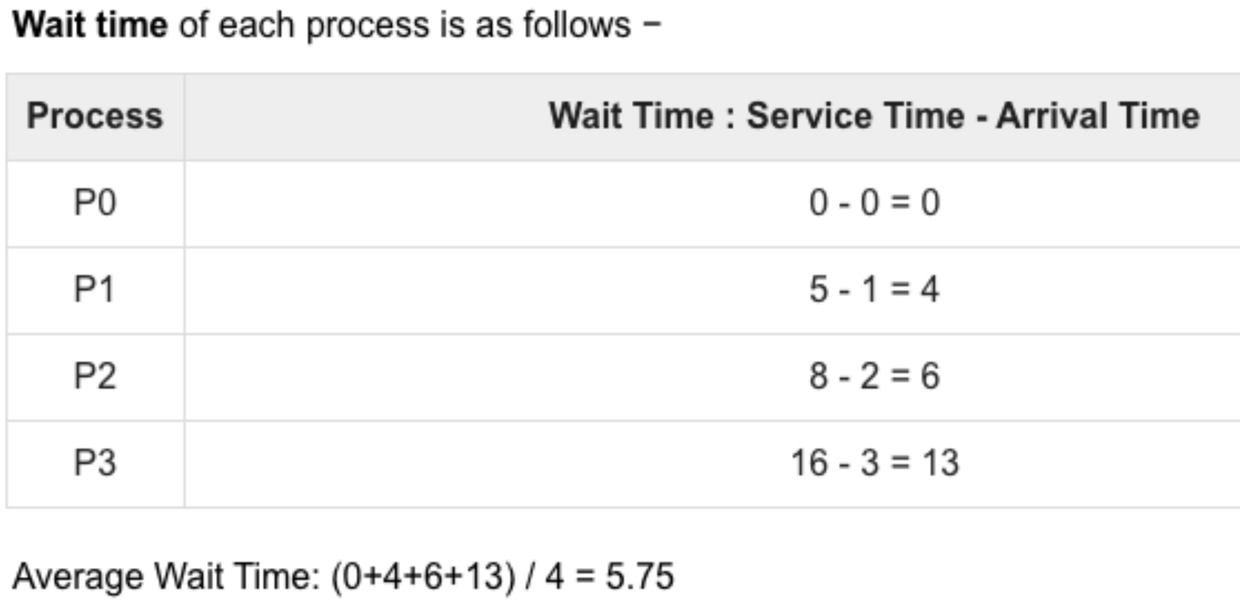


Average waiting time: (0+3+6)/3=3. Average waiting time vary substantially if the process CPU burst time vary greatly.

FCFS Example 2



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

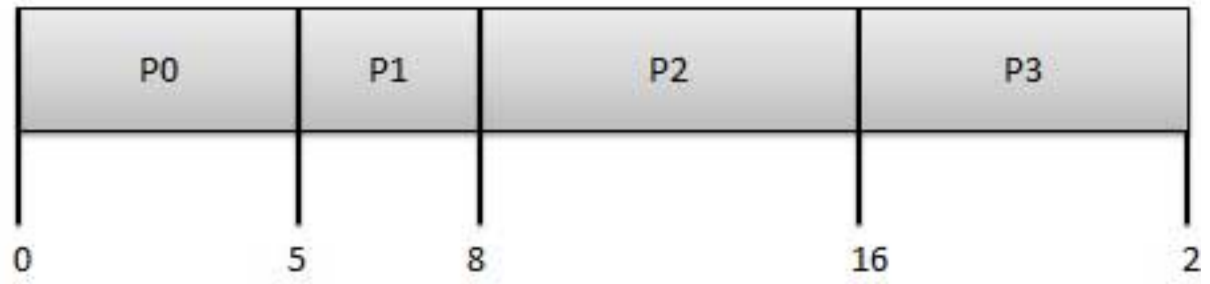


Wait time of each process is as follows −

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

Average Wait Time: (0+4+6+13) / 4 = 5.75

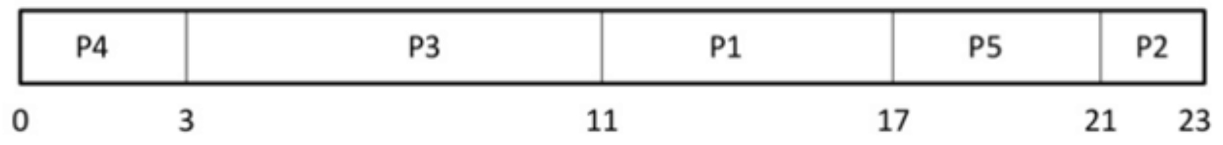
Gnatt Chart:



Calculate Turnaround time for each process
Turnaround time = Burst time + Waiting time

FCFS Example 3

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



P4 = 0-0 = 0

P3 = 3-1 = 2

PI = 11-2 = 9

P5= 17-4 = 13

P2= 21-5= 16

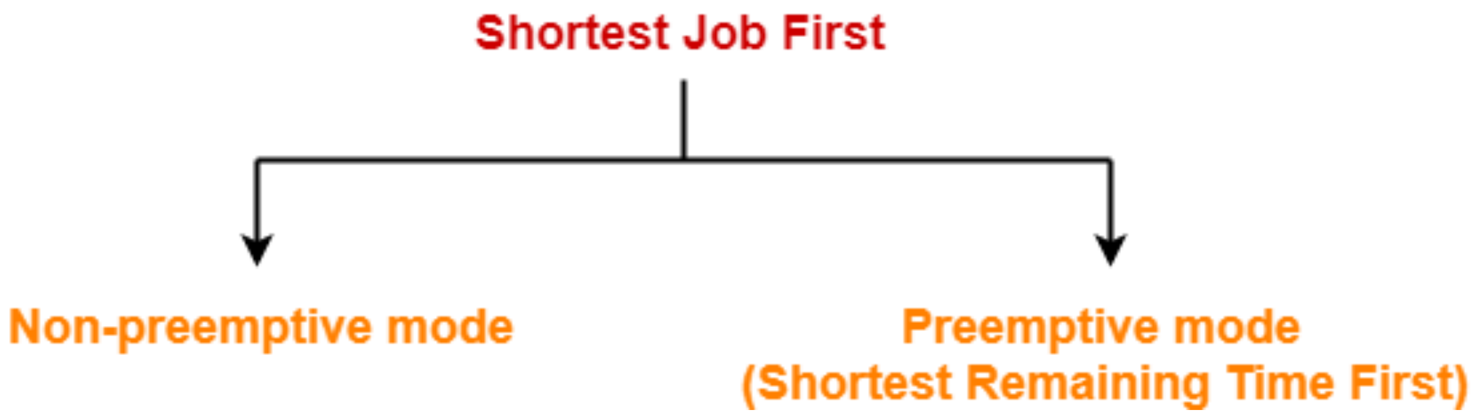
Average Waiting Time

$$\frac{0+2+9+13+16}{5}$$

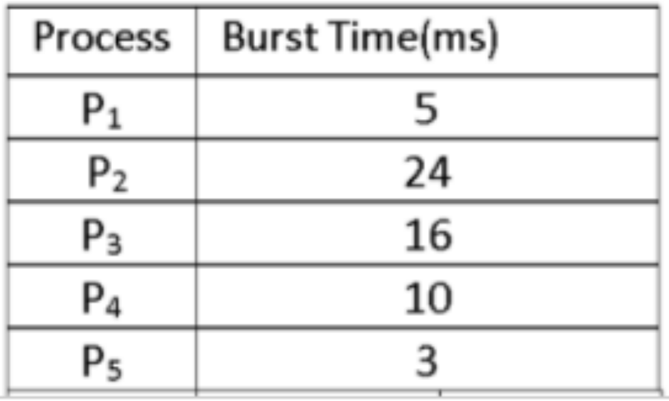
= 40/5= 8

Shortest Job First (SJF) Scheduling

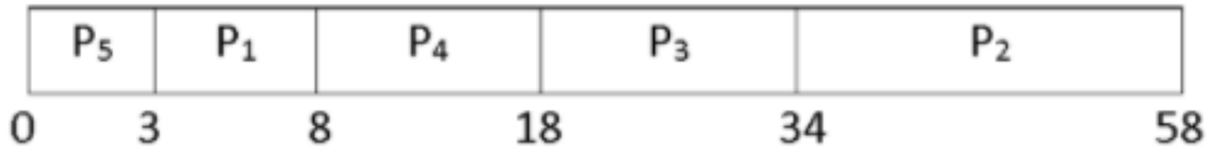
- **This is also known as shortest job next, or SJN**
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.



SJF (Non-Preemptive)Example 1



Process	Burst Time(ms)
P_1	5
P_2	24
P_3	16
P_4	10
P_5	3



Waiting Time for

P1 = 3 - 0 = 3ms

P2 = 34 - 0 = 34ms

P3 = 18 - 0 = 18ms

P4 = 8 - 0 = 8ms

P5 = 0ms

Now, Average Waiting Time = (3 + 34 + 18 + 8 + 0) / 5 = 12.6ms

Turnaround Time of

P1 = 3 + 5 = 8ms

P2 = 34 + 24 = 58ms

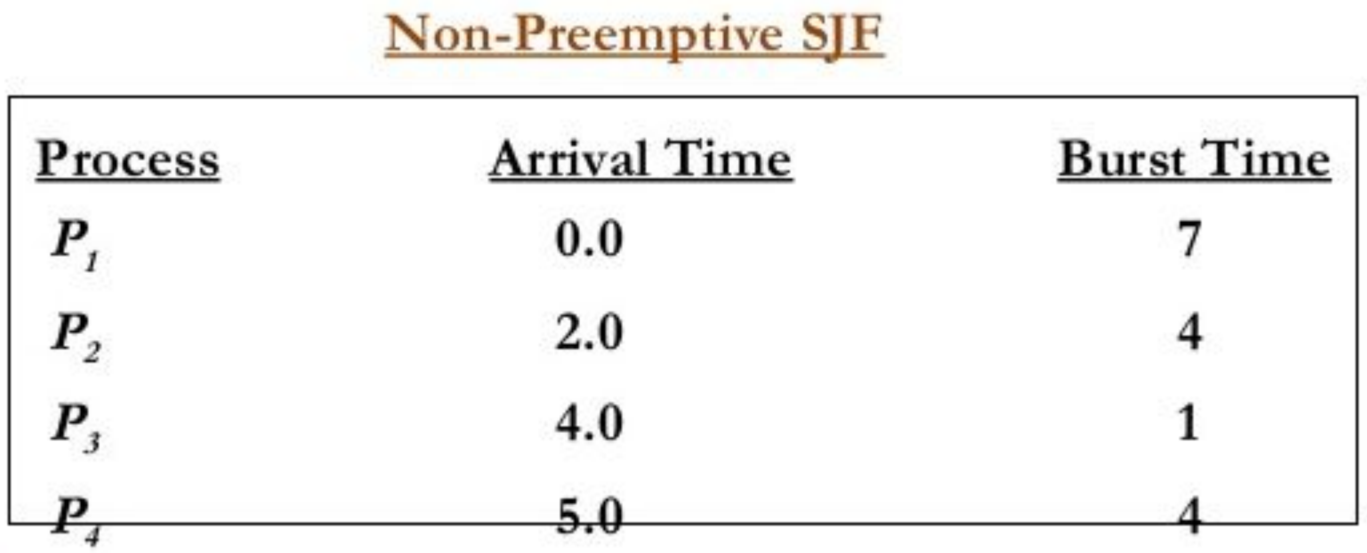
P3 = 18 + 16 = 34ms

P4 = 8 + 10 = 18ms

P5 = 0 + 3 = 3ms

Therefore, Average Turnaround Time = (8 + 58 + 34 + 18 + 3) / 5 = 24.2ms

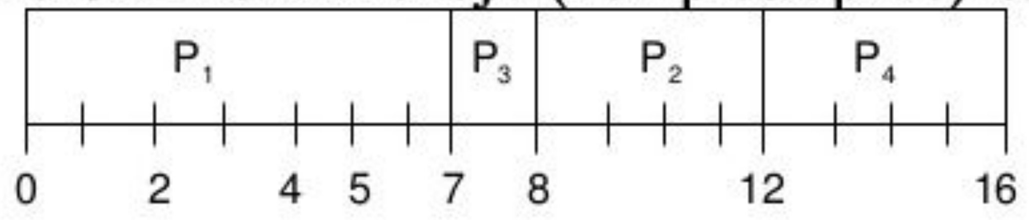
SJF (Non-Preemptive)Example 2



Non-Preemptive SJF

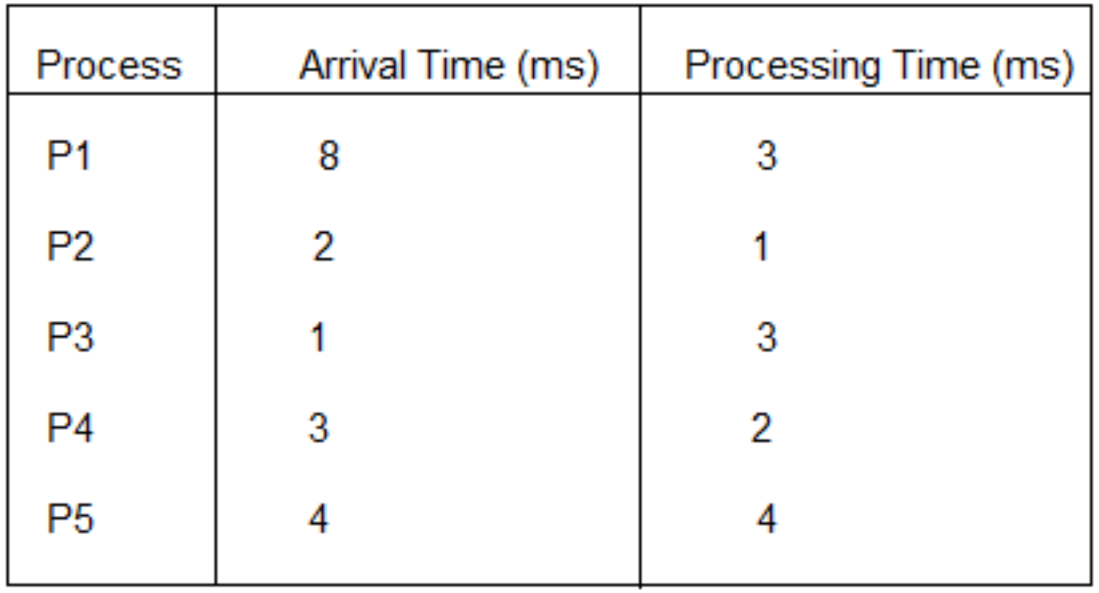
Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- The Gantt Chart for SJF (non-preemptive) is:

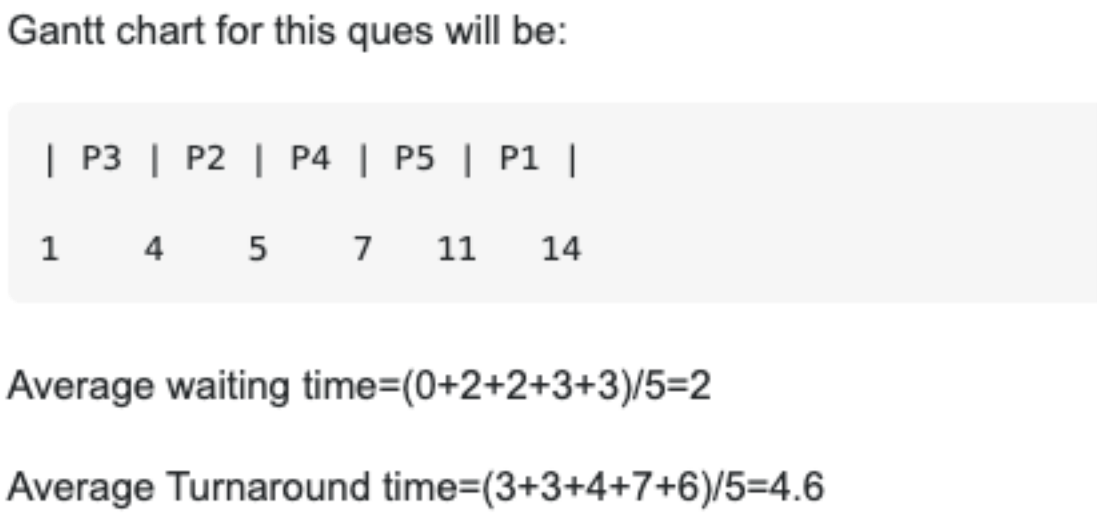


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

SJF (Non Preemptive)Example 3



Process	Arrival Time (ms)	Processing Time (ms)
P1	8	3
P2	2	1
P3	1	3
P4	3	2
P5	4	4



Gantt chart for this ques will be:

```
| P3 | P2 | P4 | P5 | P1 |
1    4    5    7    11   14
```

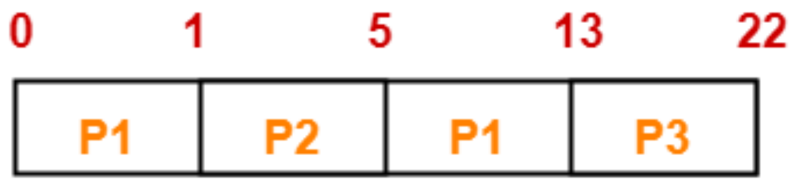
Average waiting time=(0+2+2+3+3)/5=2

Average Turnaround time=(3+3+4+7+6)/5=4.6

First process P3 has arrived so it will execute first. Since the burst time of P3 is 3sec after the completion of P3, processes P2,P4, and P5 has been arrived. Among P2,P4, and P5 the shortest burst time is 1sec for P2, so P2 will execute next. Then P4 and P5. At last P1 will be executed.

Shortest Remaining Time First (SRTF) Example 1

Process Id	Arrival time	Burst time
P1	0	~~9~~ 8
P2	1	4
P3	2	9



Gantt Chart

Process Id	Exit time	Turn Around time	Waiting time
P1	13	4+9 = 13	(0-0)+(5-1) = 4
P2	5	0+4 = 4	1 – 1 = 0
P3	22	11+9 = 20	13-2 = 11

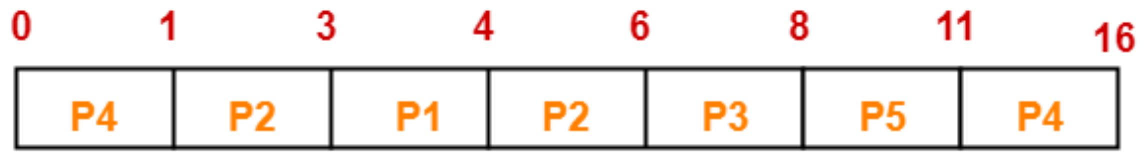
Now,

- Average Turn Around time = (13 + 4 + 20) / 3 = 37 / 3 = 12.33 unit
- Average waiting time = (4 + 0 + 11) / 3 = 15 / 3 = 5 unit

Shortest Remaining Time First (SRTF) Example 2

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

Process Id	Exit time	Turn Around time	Waiting time
P1	4	0+1 = 1	(3-3) = 0
P2	6	1+4 = 5	(1-1)+(4-3) = 1
P3	8	2+2 = 4	(6-4) = 2
P4	16	10+6 = 16	(0-0)+(11-1) = 10
P5	11	6+3 = 9	(8-2) = 6



Gantt Chart

Now,

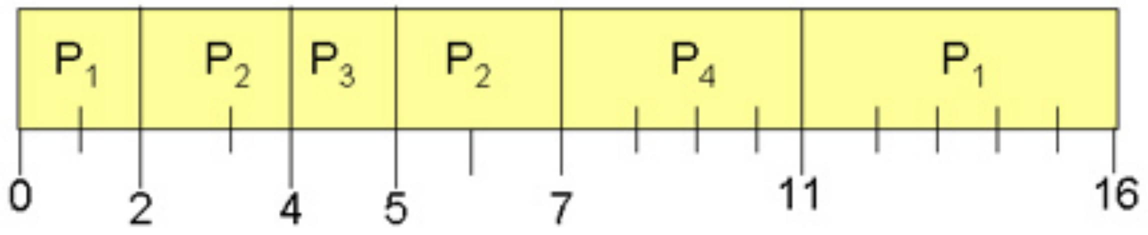
- Average Turn Around time = (1 + 5 + 4 + 16 + 9) / 5 = 35 / 5 = 7 unit
- Average waiting time = (0 + 1 + 2 + 10 + 6) / 5 = 19 / 5 = 3.8 unit

Shortest Remaining Time First (SRTF) Example 3

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

Waiting Time,
P1=(0-0)+(11-2)=9
P2=(2-2)+(5-4)=1
P3=(4-4)=0
P4=(7-5)=2

Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a quantum.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.

Advantages of round-robin:

- No starvation will be there in round-robin because every process will get chance for its execution.
- Used in time-sharing systems.

Disadvantages of round-robin:

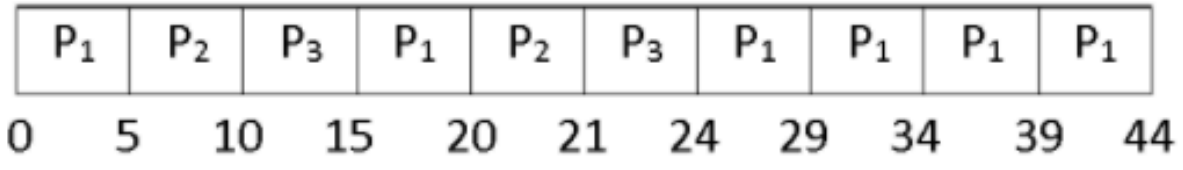
- We have to perform a lot of context switching here, which will keep the CPU idle

Round Robin Example 1

Here is the Round Robin scheduling example with gantt chart. Time Quantum is 5ms.

Process	CPU Burst Time
P_1	30
P_2	6
P_3	8

Gantt Chart



Waiting Time of

P1 = 0 + (15 - 5) + (24 - 20) = 14ms

P2 = 5 + (20 - 10) = 15ms

P3 = 10 + (21 - 15) = 16ms

Therefore, Average Waiting Time = (14 + 15 + 16) / 3 = 15ms

Turnaround Time of

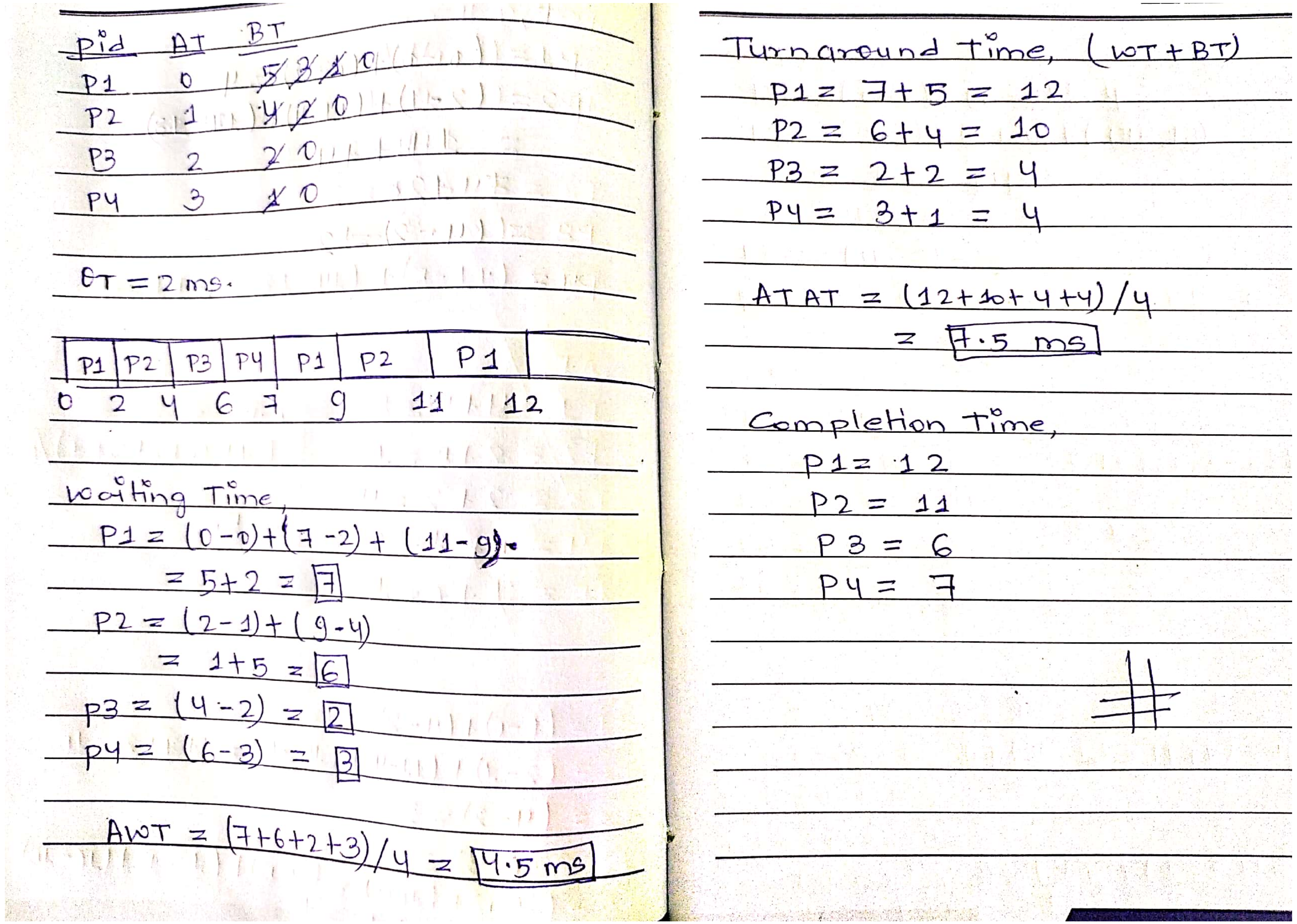
P1 = 14 + 30 = 44ms

P2 = 15 + 6 = 21ms

P3 = 16 + 8 = 24ms

Therefore, Average Turnaround Time = (44 + 21 + 24) / 3 = 29.66ms

Round Robin Example 2



Pid	AT	BT
P1	0	~~5~~ ~~3~~ ~~1~~ 0
P2	1	~~4~~ ~~2~~ 0
P3	2	~~2~~ 0
P4	3	~~1~~ 0

QT = 2 ms.

P1	P2	P3	P4	P1	P2	P1

0 2 4 6 7 9 11 12

Waiting Time,

P1 = (0-0)+(7-2) + (11-9).

= 5+2 = $\boxed{7}$

P2 = (2-1)+(9-4)

= 1+5 = $\boxed{6}$

P3 = (4-2) = $\boxed{2}$

P4 = (6-3) = $\boxed{3}$

AWT = (7+6+2+3)/4 = $\boxed{4.5 \text{ ms}}$

Turnaround time, (WT+BT)

P1 = 7+5 = 12

P2 = 6+4 = 10

P3 = 2+2 = 4

P4 = 3+1 = 4

ATAT = (12+10+4+4)/4

= $\boxed{7.5 \text{ ms}}$

Completion Time,

P1 = 12

P2 = 11

P3 = 6

P4 = 7

#

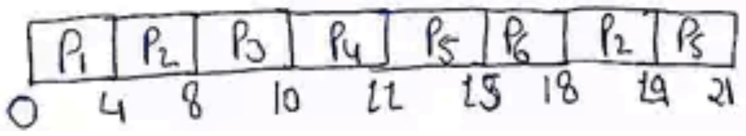
Round Robin Example 3

TQ=4

P.No	AT	BT	
1	0	4	X
2	1	~~5~~	~~1~~X
3	2	2	X
4	3	1	X
5	4	~~6~~	2
6	6	3	X

CT	TAT	WT
4	4	0
19	18	13
10	8	6
11	8	7
21	17	11
18	12	9
	67	46

~~P1~~ ~~P2~~ ~~P3~~ ~~P4~~ ~~P5~~ ~~P6~~ P2 P5

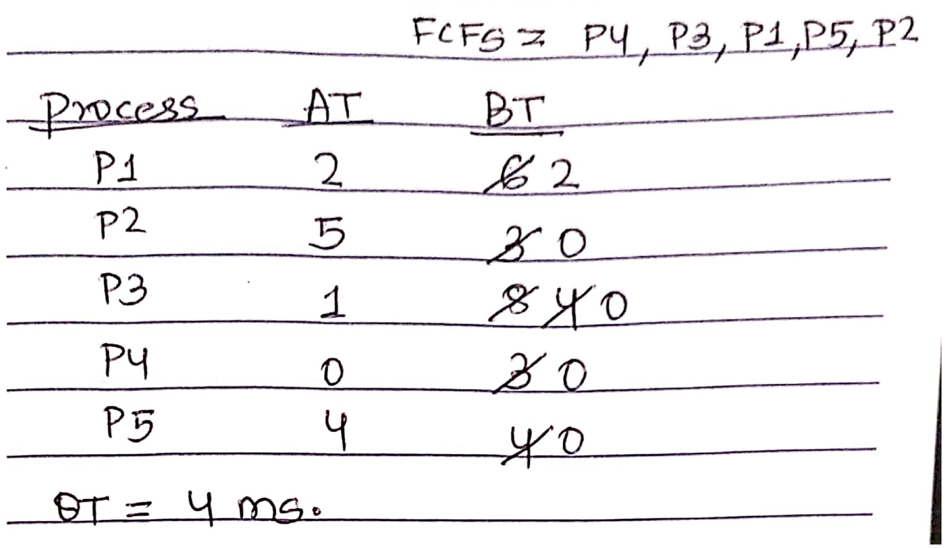


$ATAT = \frac{67}{6} \Rightarrow 11.16$

$AWT = \frac{46}{6} \Rightarrow 7.66$

Waiting Time Calculation,
P1=(0-0)=0
P2=(4-1)+(18-8)=13
P3=(8-2)=6
P4=(10-3)=7
P5=(11-4)+(19-15)=11
P6=(15-6)=9

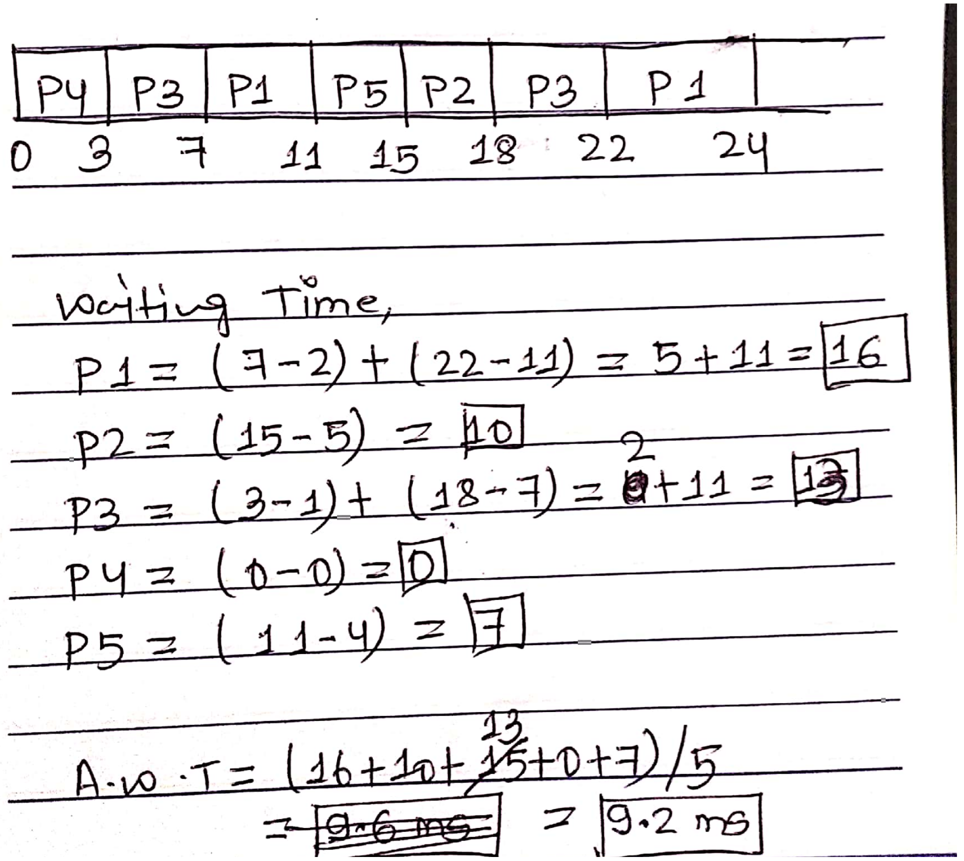
Round Robin Example 4



FCFS = P4, P3, P1, P5, P2

Process	AT	BT
P1	2	~~6~~ 2
P2	5	~~3~~ 0
P3	1	~~8~~ ~~4~~ 0
P4	0	~~3~~ 0
P5	4	~~4~~ 0

QT = 4 ms.



P4	P3	P1	P5	P2	P3	P1

0 3 7 11 15 18 22 24

Waiting Time,

$P1 = (7-2) + (22-11) = 5 + 11 = \boxed{16}$

$P2 = (15-5) = \boxed{10}$

$P3 = (3-1) + (18-7) = \overset{2}{\cancel{0}} + 11 = \boxed{13}$

$P4 = (0-0) = \boxed{0}$

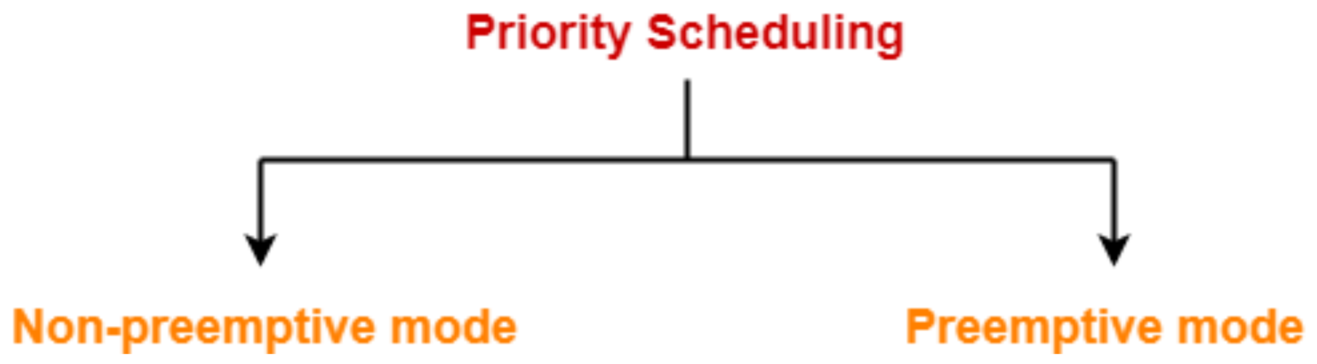
$P5 = (11-4) = \boxed{7}$

$A.W.T = (16 + 10 + \overset{13}{\cancel{15}} + 0 + 7)/5$

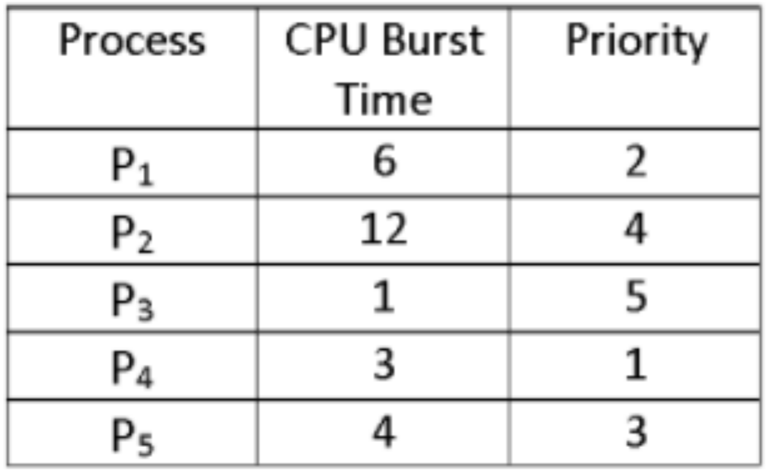
$= \cancel{\boxed{9.6\text{ ms}}} = \boxed{9.2\text{ ms}}$

Priority Based Scheduling

- Priority scheduling is a **non-preemptive algorithm by default** and one of the most common scheduling algorithms in batch systems.
- **Each process is assigned a priority. Process with highest priority is to be executed first and so on.**
- Processes with **same priority are executed on first come first served** basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

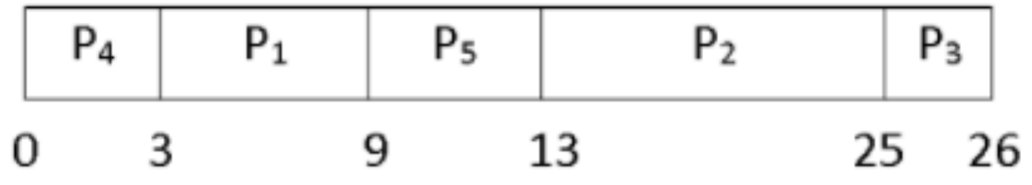


Priority Based Scheduling Example 1



Process	CPU Burst Time	Priority
P_1	6	2
P_2	12	4
P_3	1	5
P_4	3	1
P_5	4	3

Gantt Chart



Turnaround Time of process

P1 = (3 + 6) = 9ms

P2 = (13 + 12) = 25ms

P3 = (25 + 1) = 26ms

P4 = (0 + 3) = 3ms

P5 = (9 + 4) = 13ms

Therefore, Average Turnaround Time = (9 + 25 + 26 + 3 + 13) / 5 = 15.2ms

Waiting Time of process

P1 = 3ms

P2 = 13ms

P3 = 25ms

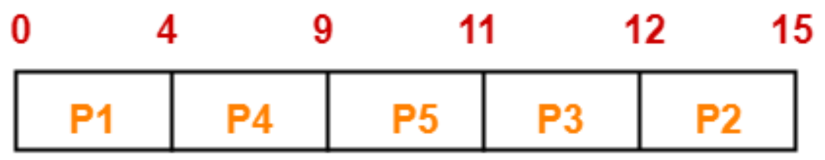
P4 = 0ms

P5 = 9ms

Therefore, Average Waiting Time = (3 + 13 + 25 + 0 + 9) / 5 = 10ms

Priority Based Scheduling Example 2

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



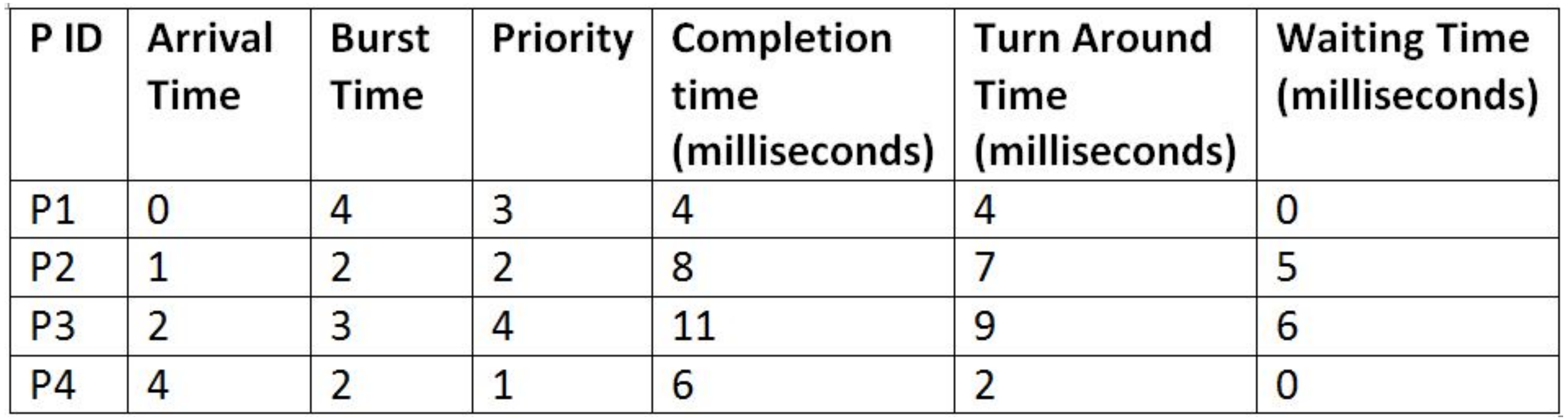
Gantt Chart

Process Id	Completion time	Turn Around time	Waiting time
P1	4	0 + 4 = 4	0 – 0 = 0
P2	15	11 + 3 = 14	12 – 1 = 11
P3	12	9 + 1 = 10	11 – 2 = 9
P4	9	1 + 5 = 6	4 – 3 = 1
P5	11	5 + 2 = 7	9 – 4 = 5

Now,

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

Priority Based Scheduling



P ID	Arrival Time	Burst Time	Priority	Completion time (milliseconds)	Turn Around Time (milliseconds)	Waiting Time (milliseconds)
P1	0	4	3	4	4	0
P2	1	2	2	8	7	5
P3	2	3	4	11	9	6
P4	4	2	1	6	2	0

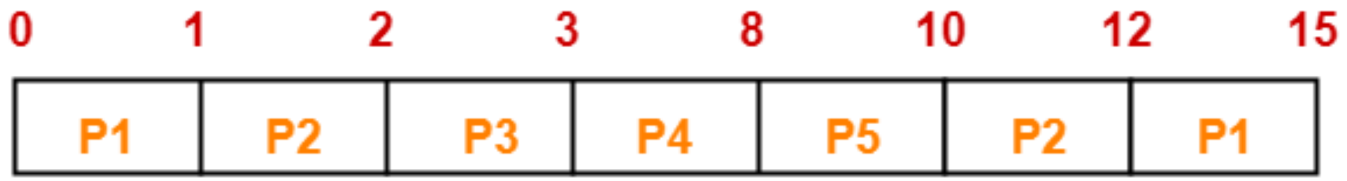
- There is only **P1** available at time 0, so it will be executed first irrespective of the priority, and it cannot be preempted in between before its completion.
- When it is completed at 4th-time unit, we have all **P2**, **P3**, and **P4** available. So, they are executed according to their priorities.

Solve Yourself !

Priority Based Scheduling (Pre-emptive)

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

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



Gantt Chart

•Average Turn Around time = (15 + 11 + 1 + 5 + 6) / 5 = 38 / 5 = 7.6 unit

•Average waiting time = (11 + 8 + 0 + 0 + 4) / 5 = 23 / 5 = 4.6 unit

Highest Response Ratio Next (HRRN) Scheduling

In HRRN Scheduling,

- Out of all the available processes, **CPU is assigned to the process having highest response ratio.**
- **In case of a tie, it is broken by FCFS Scheduling**.
- It operates only in **non-preemptive mode**.

Response Ratio (RR) for any process is calculated by using the formula-

$$\boxed{\text{Response Ratio} = \frac{W + B}{B}}$$

W = Waiting time of the process so far

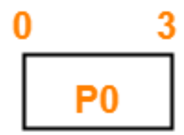
B = Burst time or Service time of the process

HRRN Scheduling Example

Process Id	Arrival time	Burst time
P0	0	3
P1	2	6
P2	4	4
P3	6	5
P4	8	2

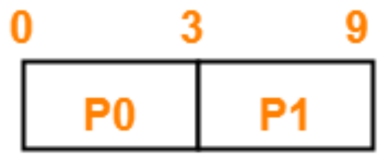
Step-01:

- At t = 0, only the process P0 is available in the ready queue.
- So, process P0 executes till its completion.



Step-02:

- At t = 3, only the process P1 is available in the ready queue.
- So, process P1 executes till its completion.

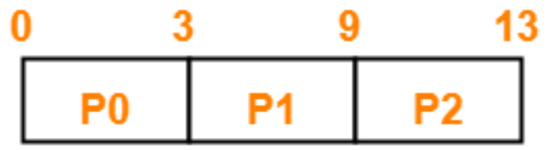


Step-03:

- At t = 9, the processes P2, P3 and P4 are available in the ready queue.
- The process having the highest response ratio will be executed next.

The response ratio are-

- Response Ratio for process P2 = [(9-4) + 4] / 4 = 9 / 4 = 2.25
- Response Ratio for process P3 = [(9-6) + 5] / 5 = 8 / 5 = 1.6
- Response Ratio for process P4 = [(9-8) + 2] / 2 = 3 / 2 = 1.5



Since process P2 has the highest response ratio, so process P2 executes till completion.

HRRN Scheduling Example

Process Id	Arrival time	Burst time
P0	0	3
P1	2	6
P2	4	4
P3	6	5
P4	8	2

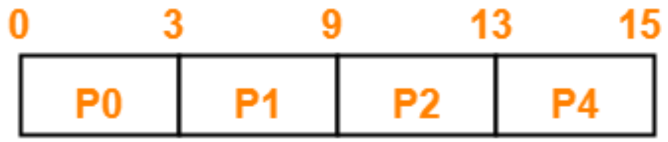
Step-04:

- At t = 13, the processes P3 and P4 are available in the ready queue.

The response ratio are calculated as-

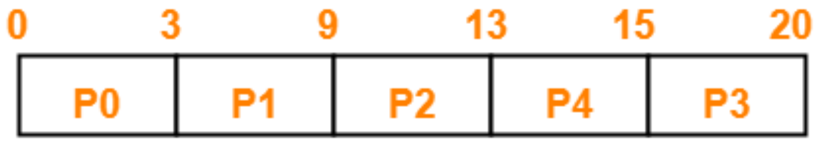
- Response Ratio for process P3 = [(13-6) + 5] / 5 = 12 / 5 = 2.4
- Response Ratio for process P4 = [(13-8) + 2] / 2 = 7 / 2 = 3.5

Since process P4 has the highest response ratio, so process P4 executes till completion.



Step-05:

- At t = 15, only the process P3 is available in the ready queue.
- So, process P3 executes till its completion.



•Average Turn Around time = (3 + 7 + 9 + 14 + 7) / 5 = 40 / 5 = 8 units

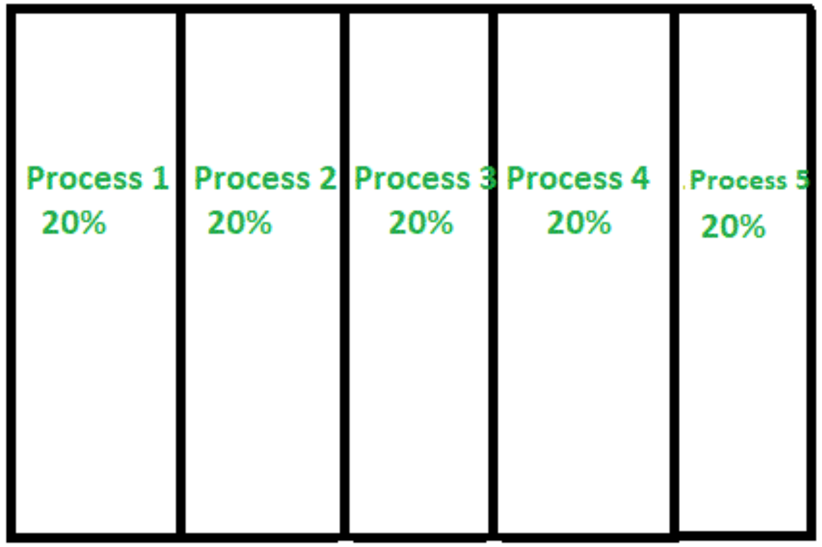
•Average waiting time = (0 + 1 + 5 + 9 + 5) / 5 = 20 / 5 = 4 units

Priority Fair Share Scheduling

- Fair-share scheduling is a scheduling algorithm that was first designed by Judy Kay and Piers Lauder at Sydney University in the 1980s.
- **It is a scheduling algorithm for computer operating systems that dynamically distributes the time quantam "equally" to its users.**
- **Time quantam is the processor time allowed for a process to run.**
- But in a round-robin scheduling where the time slices or time quanta are allocated equally in a circular order, due to the distribution of time slices in a circular manner, any equal amount of time quantam will produce a similar output, therefore, an arbitrary distribution is required in this scenario.

Specificity of Fair-share scheduling :

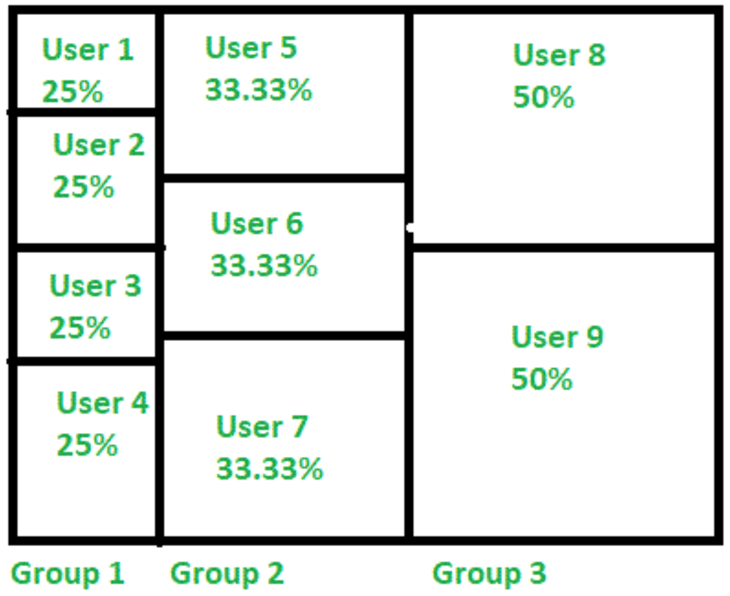
- This algorithm equally distributes the processor time to its users, for instance, there are 5 users (A, B, C, D, E)each of them are simultaneously executing a process, the scheduler divides the CPU periods such that all the users get the same share of the CPU cycles (100%/5) that is 20%.
- **Even though, a user moves onto the second while the other at the first, the algorithm is so specific that it ensures that this user is attributed with only 10% for the second process making it a total of 20%.**



1 CPU time slices are equally by 5 processses in Fair-share scheduling operating system.

Specificity of Fair-share scheduling (cont.):

- The scheduler logically divides an equal amount even though, another layer of partition is added, for example, if there were 3 groups present with different number of people in each group, the algorithm would still divide the same time for those groups, 100%/3= 33.33%, this 33.33% would be shared equally in the respective group depending on the number of users present in the group.



33.33 perecent of a single CPU time quanta is shared.

Note that all the users share the 33.33% of the group share among them.

Fair-share scheduling is an efficient strategy that creates a consistent user experience.

Guaranteed Scheduling

- **Make real promises to the users about performance.**
- **If there are n users logged in while you are working, you will receive about 1 /n of the CPU power.**
- **Similarly, on a single-user system with n processes running, all things being equal, each one should get 1 /n of the CPU cycles.**
- To make good on this promise, the system must keep track of how much CPU each process has had since its creation.
- **CPU Time entitled= (Time Since Creation)/n**
- Then compute the ratio of Actual CPU time consumed to the CPU time entitled.
- A ratio of 0.5 means that a process has only had half of what it should have had, and a ratio of 2.0 means that a process has had twice as much as it was entitled to.
- The algorithm is then to run the process with the lowest ratio until its ratio has moved above its closest competitor.

Lottery Scheduling

- **Lottery Scheduling is a probabilistic scheduling algorithm for processes in an operating system**.
- Processes are each assigned some number of lottery tickets for various system resources such as CPU time and the scheduler draws a random ticket to select the next process.
- The distribution of tickets need not be uniform; granting a process more tickets provides it a relative higher chance of selection.
- This technique can be used to approximate other scheduling algorithms, such as Shortest job next and Fair- share scheduling.
- **Lottery scheduling solves the problem of starvation. Giving each process at least one lottery ticket guarantees that it has non-zero probability of being selected at each scheduling operation.**
- More important process can be given extra tickets to increase their odd of winning. If there are 100 tickets outstanding, & one process holds 20 of them it will have 20% chance of winning each lottery. In the long run it will get 20% of the CPU. A process holding a fraction f of the tickets will get about a fraction of the resource in questions.

Real Time Scheduling

Time is crucial and plays an essential role.
eg. the computer in a compact disc player gets the bits as they come off the drive and must convert them into music with a very tight time interval. If the calculation takes too long the music sounds peculiar. Other example includes

- Auto pilot in Aircraft
- Robot control in automated factory.
- Patient monitoringin Factory. (ICU)

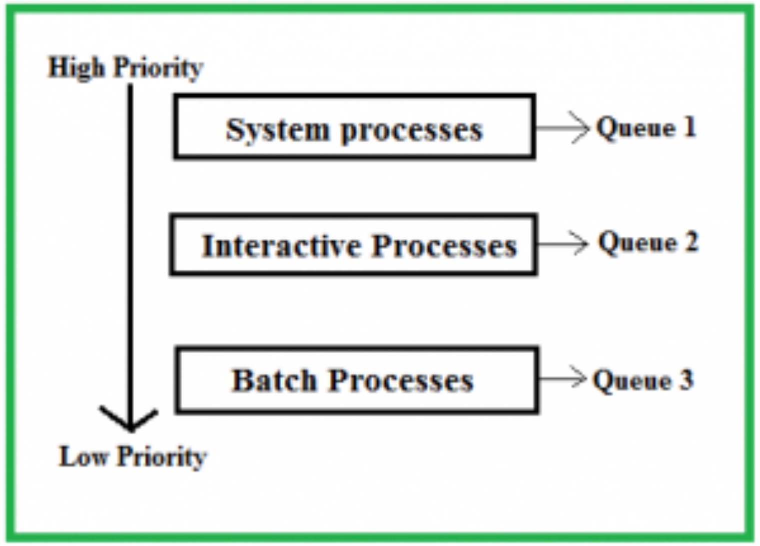
Two types:

1. **Hard Real Time system:** There are absolute deadline that must be met.
2. **Soft Real Time system:** Missing an occasional deadline is undesirable but nevertheless tolerable.

In both cases real time behavior is achieved by dividing the program into a no. of processes, each of whose behavior is predictable & known in advance. These processes are short lived and can run to completion. Its the job of schedulers to schedule the process in such a way that all deadlines are met.

Multilevel Queue Scheduling

- **A multi-level queue scheduling algorithm partitions the ready queue into several separate queues.**
- The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type.
- **Each queue has its own scheduling algorithm.**
- **For example:** separate queues might be used for **foreground and background processes**.
- **The foreground queue might be scheduled by Round Robin algorithm, while the background queue is scheduled by an FCFS algorithm.**

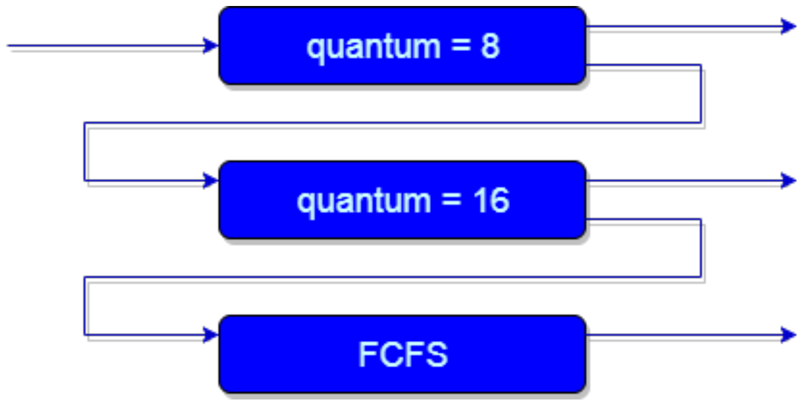


All three different type of processes have there own queue. Each queue have its own Scheduling algorithm. For example, queue 1 and queue 2 uses **Round Robin** while queue 3 can use **FCFS** to schedule there processes.

Multilevel Feedback Queue Scheduling

- **In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system.** Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.

- **Multilevel feedback queue scheduling, however, allows a process to move between queues**.
- The idea is to separate processes with different CPU-burst characteristics.
- If a process uses too much CPU time, it will be moved to a lower-priority queue.
- Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.



An example of a multilevel feedback queue.

Multilevel Feedback Queue Scheduling (cont.)

In general, a multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues.
- The scheduling algorithm for each queue.
- The method used to determine when to upgrade a process to a higher-priority queue.
- The method used to determine when to demote a process to a lower-priority queue.
- The method used to determine which queue a process will enter when that process needs service.

The definition of a multilevel feedback queue scheduler makes it the most general CPU-scheduling algorithm. It can be configured to match a specific system under design. Unfortunately, it also requires some means of selecting values for all the parameters to define the best scheduler. Although a multilevel feedback queue is the **most general scheme**, it is also the **most complex**.