CHAPTER - 5

CPU Scheduling

Neso Academy

CPU Scheduling

CPU scheduling is the basis of multiprogrammed operating systems.

By switching the CPU among processes, the operating system can make the computer more productive.

Topics to be covered:

- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems.
- To describe various CPU-scheduling algorithms

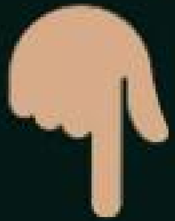
Let's get the basics right

- In a single-processor system, only one process can run at a time.
- Any others must wait until the CPU is free and can be rescheduled.
    - The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
    - A process is executed until it must wait, typically for the completion of some I/O request.

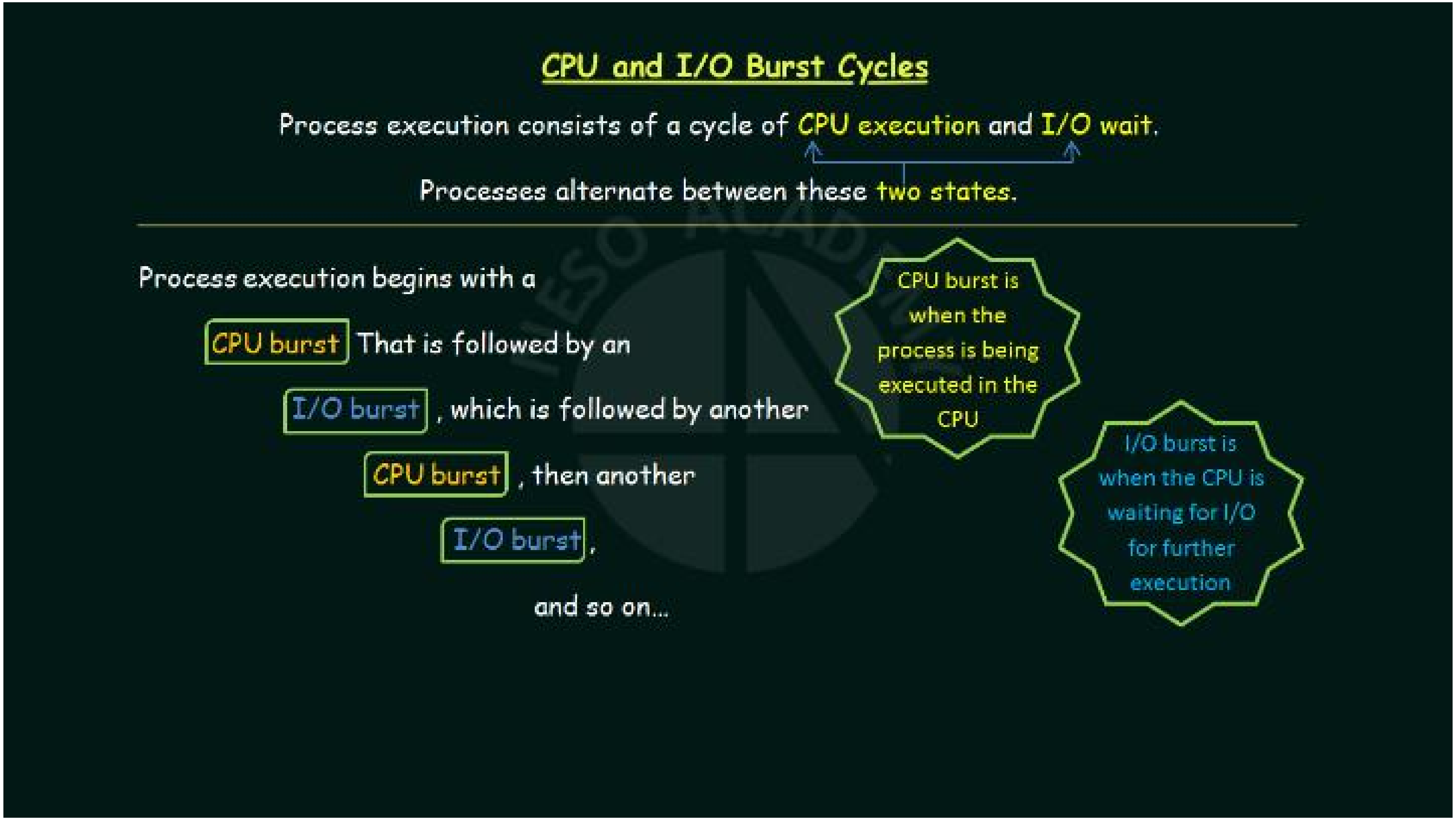
In a simple computer system, the CPU then just sits idle.

All this waiting time is wasted; no useful work is accomplished.

- With multiprogramming, we try to use this time productively.
    - Several processes are kept in memory at one time.



- When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process and this pattern continues.



CPU and I/O Burst Cycles

Process execution consists of a cycle of CPU execution and I/O wait.

Processes alternate between these two states.

Process execution begins with a

CPU burst That is followed by an

I/O burst , which is followed by another

CPU burst , then another

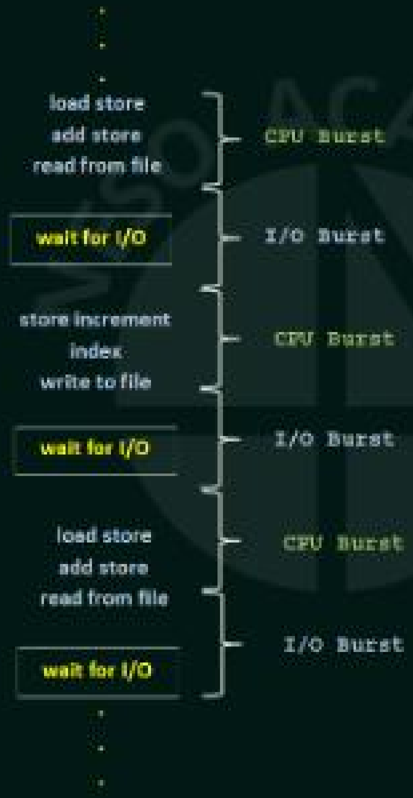
I/O burst,

and so on...

CPU burst is when the process is being executed in the CPU

I/O burst is when the CPU is waiting for I/O for further execution

Eventually, the final CPU burst ends with a system request to terminate execution.



Alternating sequence of
CPU
&
I/O bursts

Preemptive and Non-Preemptive Scheduling

CPU Scheduler

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.

Dispatcher

The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler.

The time it takes for the dispatcher to stop one process and start another running is known as the dispatch latency.

CPU-scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state.

2. When a process switches from the running state to the ready state (for example, when an interrupt occurs).

3. When a process switches from the waiting state to the ready state (for example, at completion of I/O).

4. When a process terminates.

For situations 1 and 4, there is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution.
However, there is a choice for situations 2 and 3.

When scheduling takes place only under circumstances 1 and 4, we say that the scheduling scheme is nonpreemptive or cooperative; otherwise, it is preemptive.

Scheduling Criteria

CPU utilization

We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).

Throughput

If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit, called throughput.

Turnaround time

From the point of view of a particular process, the important criterion is how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

Waiting time

The CPU scheduling algorithm does not affect the amount of time during which a process executes or does I/O; it affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue.

Response time

In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early and can continue computing new results while previous results are being output to the user. Thus, another measure is the time from the submission of a request until the first response is produced. This measure, called response time, is the time it takes to start responding, not the time it takes to output the response. The turnaround time is generally limited by the speed of the output device.

Scheduling Criteria

CPU utilization

Throughput

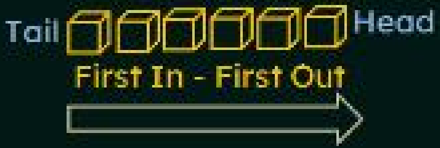
Turnaround time

Waiting time

Response time

Scheduling Algorithms
(First-Come, First-Served Scheduling)

- By far the simplest CPU-scheduling algorithm.
- The process that requests the CPU first is allocated the CPU first.
- The implementation of the FCFS policy is easily managed with a **FIFO queue**.



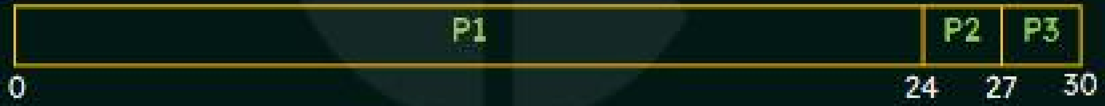
- When a process enters the ready queue, its PCB is linked onto the tail of the queue.
- When the CPU is free, it is allocated to the process at the head of the queue.
- The running process is then removed from the queue.

The average waiting time under the FCFS policy, however, is often quite long.

Consider the following set of processes that arrive at time 0

Process	Burst Time (ms)
P1	24
P2	3
P3	3

If the processes arrive in the order P1, P2, P3, and are served in FCFS order, we get the result shown in the following Gantt chart:



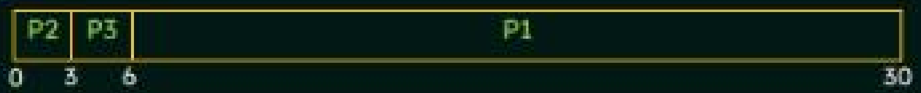
Waiting Time for P1 = 0 ms

Waiting Time for P2 = 24 ms

Waiting Time for P3 = 27 ms

Average Waiting Time = (0 + 24 + 27)/3 = 17 ms

If the processes arrive in the order P2, P3, P1, however the result will be shown in the following Gantt chart:



Waiting Time for P1 = 6 ms

Waiting Time for P2 = 0 ms

Waiting Time for P3 = 3 ms

Average Waiting Time = (6 + 0 + 3)/3 = 3 ms

This reduction is substantial. Thus, the average waiting time under an FCFS policy is generally not minimal and may vary substantially if the process's CPU burst times vary greatly.

The FCFS scheduling algorithm is nonpreemptive

- Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or by requesting I/O.
- The FCFS algorithm is thus particularly troublesome for time-sharing systems, where it is important that each user get a share of the CPU at regular intervals.
- It would be disastrous to allow one process to keep the CPU for an extended period.

First-Come, First-Served Scheduling

Solved Problem -1

Convoy Effect

If processes with higher burst time arrived before the processes with smaller burst time, then, smaller processes have to wait for a long time for longer processes to release the CPU.

Consider the set of 5 processes whose arrival time and burst time are given below:

Process ID	Arrival Time	Burst Time
P1	4	5
P2	6	4
P3	0	3
P4	6	2
P5	5	4

Calculate the average waiting time and average turnaround time, if FCFS Scheduling Algorithm if followed.

First-Come, First-Served Scheduling

Solved Problem - 2

The arrival times and burst times for a set of 6 processes are given in the table below:

Process ID	Arrival Time	Burst Time
P1	0	3
P2	1	2
P3	2	1
P4	3	4
P5	4	5
P6	5	2

If FCFS Scheduling Algorithm is followed and there is 1 unit of overhead in scheduling the processes, find the efficiency of the algorithm.

Scheduling Algorithms
(Shortest-Job-First Scheduling)

- This algorithm associates with each process the length of the process's next CPU burst.
- When the CPU is available, it is assigned to the process that has the smallest next CPU burst.
- If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.

The SJF algorithm can be either preemptive or nonpreemptive

A more appropriate term for this scheduling method would be the
Shortest-Next-CPU-Burst Algorithm
because scheduling depends on the length of the next CPU burst of a process, rather than its total length.

Example of SJF Scheduling (Non-Premptive)

Consider the following set of processes,with the length of the CPU burst given in milliseconds:

Process ID	Burst Time
P1	6
P2	8
P3	7
P4	3

Waiting Time for P1 = 3 ms

Waiting Time for P2 = 16 ms

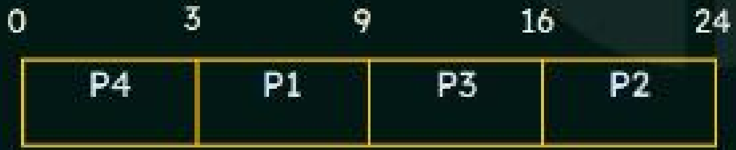
Waiting Time for P3 = 9 ms

Waiting Time for P4 = 0 ms

Average Waiting Time

= (3 + 16 + 9 + 0)/4 = 7 ms

Gantt Chart:



By comparison, if we were using the FCFS scheduling scheme, the average waiting time would be 10.25 milliseconds.

Example of SJF Scheduling (Premptive)

Consider the following four processes, with the length of the CPU burst given in milliseconds and the processes arrive at the ready queue at the times shown:

Process ID	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

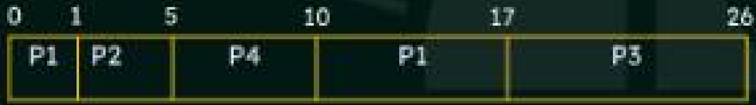
Waiting Time for P1 = (10 - 1 - 0) = 9 ms

Waiting Time for P2 = (1 - 0 - 1) = 0 ms

Waiting Time for P3 = (17 - 0 - 2) = 15 ms

Waiting Time for P4 = (5 - 0 - 3) = 2 ms

Gantt Chart:



Average Waiting Time

= (9 + 0 + 15 + 2)/4 = 6.5 ms

Waiting Time = Total waiting Time – No. of milliseconds Process executed – Arrival Time

Preemptive SJF scheduling is sometimes called Shortest-Remaining-Time-First Scheduling.

Problems with SJF Scheduling:

- The real difficulty with the SJF algorithm is knowing the length of the next CPU request.
- Although the SJF algorithm is optimal, it cannot be implemented at the level of short-term CPU scheduling.
- There is no way to know the length of the next CPU burst.

One approach is:

- To try to approximate SJF scheduling.
- We may not know the length of the next CPU burst, but we may be able to predict its value.
- We expect that the next CPU burst will be similar in length to the previous ones.
- Thus, by computing an approximation of the length of the next CPU burst, we can pick the process with the shortest predicted CPU burst.

Shortest-Job-First Scheduling

Solved Problem -1

GATE 2014

An operating system uses shortest remaining time first scheduling algorithm for pre-emptive scheduling of processes. Consider the following set of processes with their arrival times and CPU burst times (in milliseconds):

Process ID	Arrival Time	Burst Time
P1	0	12
P2	2	4
P3	3	6
P4	8	5

The average waiting time (in milliseconds) of the processes is ______________.

(A) 4.5 (B) 5.0 (C) 5.5 (D) 6.5

Process ID	Arrival Time	Burst Time
P1	0	12
P2	2	4
P3	3	6
P4	8	5

Waiting Time for P1 = (17 – 2 - 0) = 15 ms

Waiting Time for P2 = (2 – 0 - 2) = 0 ms

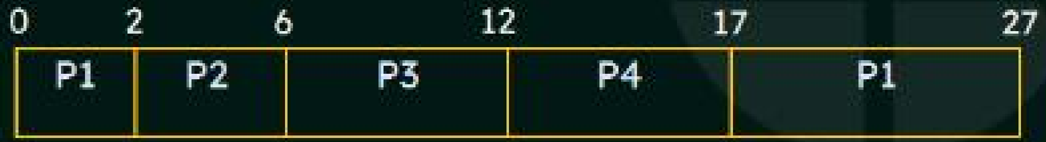
Waiting Time for P3 = (6 – 0 - 3) = 3 ms

Waiting Time for P4 = (12– 0 - 8) = 4 ms

Average Waiting Time

= (15 + 0 + 3 + 4)/4 = 5.5 ms

<u>Solution</u>: Gantt Chart:



Waiting Time = Total waiting Time – No. of milliseconds Process executed – Arrival Time

Shortest-Job-First Scheduling

Solved Problem - 2

GATE 2016

Consider the following processes, with the arrival time and the length of the CPU burst given in milliseconds. The scheduling algorithm used is preemptive shortest remaining-time first.

Process ID	Arrival Time	Burst Time
P1	0	10
P2	3	6
P3	7	1
P4	8	3

The average turn around time of these processes is ____________ milliseconds.

(A) 8.25 (B) 10.25 (C) 6.35 (D) 4.25

Process ID	Arrival Time	Burst Time
P1	0	10
P2	3	6
P3	7	1
P4	8	3

Turnaround Time for P1 = (20 – 0) = 20 ms

Turnaround Time for P2 = (10 – 3) = 7 ms

Turnaround Time for P3 = (8 – 7) = 1 ms

Turnaround Time for P4 = (13– 8) = 5 ms

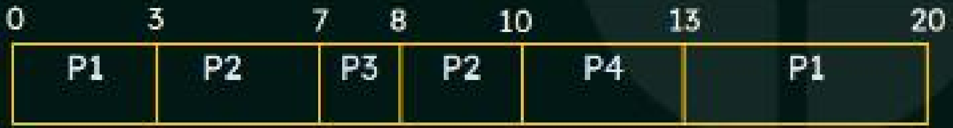
Average Turnaround Time

= (20 + 7 + 1 + 5)/4

= 33/4

= 8.25 ms

<u>Solution:</u> **Gantt Chart:**



Turn Around time = Completion time – Arrival time

Scheduling Algorithms
(Priority Scheduling)

- A priority is associated with each process, and the CPU is allocated to the process with the highest priority.

- Equal-priority processes are scheduled in FCFS order.

- An SJF algorithm is simply a priority algorithm where the priority is the inverse of the (predicted) next CPU burst.
  The larger the CPU burst, the lower the priority, and vice versa.

Priority scheduling can be either preemptive or nonpreemptive.

A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.

A nonpreemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.

Consider the following set of processes, assumed to have arrived at time 0, in the order P1, P2, P3, P4, P5, with the length of the CPU burst given in milliseconds:

Process ID	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Waiting Time for P1 = 6 ms

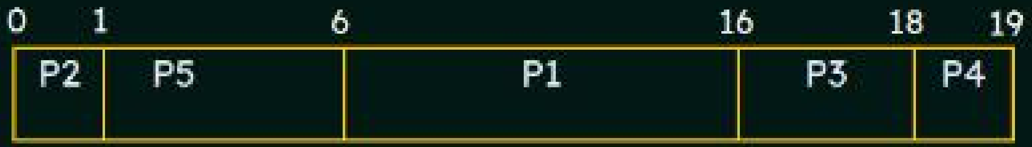
Waiting Time for P2 = 0 ms

Waiting Time for P3 = 16 ms

Waiting Time for P4 = 18 ms

Waiting Time for P5 = 1 ms

Using Priority Scheduling, we would schedule these processes according to the following Gantt Chart:



Average Waiting Time

= (6 + 0 + 16 + 18 + 1) / 5

= 41 / 5 ms

= 8.2 ms

Problem with Priority Scheduling

- A major problem with priority scheduling algorithms is indefinite blocking,
- or starvation.
- A process that is ready to run but waiting for the CPU can be considered blocked.
- A priority scheduling algorithm can leave some low priority processes waiting indefinitely.
- In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.

Solution to the Problem

- A solution to the problem of indefinite blockage of low-priority processes is **aging**.
- Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.
- For example,
  If priorities range from 127 (low) to 0 (high), we could increase the priority of a waiting process by 1 every 15 minutes.
- Eventually, even a process with an initial priority of 127 would have the highest priority in the system and would be executed.

Priority Scheduling

Solved Problem -1

GATE 2017

Consider the set of processes with arrival time (in milliseconds), CPU burst time (in milliseconds), and priority (0 is the highest priority) shown below. None of the processes have I/O burst time.

Process ID	Arrival Time	Burst Time	Priority
P1	0	11	2
P2	5	28	0
P3	12	2	3
P4	2	10	1
P5	9	16	4

The average waiting time (in milliseconds) of all the processes using preemptive priority scheduling algorithm is________.

(A) 29 (B) 30 (C) 31 (D) 32

Process ID	Arrival Time	Burst Time	Priority
P1	0	11	2
P2	5	28	0
P3	12	2	3
P4	2	10	1
P5	9	16	4

Waiting Time for P1 = (40 – 2 - 0) = 38 ms

Waiting Time for P2 = (5 – 0 - 5) = 0 ms

Waiting Time for P3 = (49 - 0 - 12) = 37 ms

Waiting Time for P4 = (33 – 3 - 2) = 28 ms

Waiting Time for P5 = (51 – 0 - 9) = 42 ms

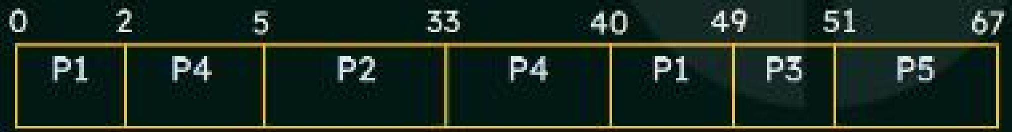
Average Waiting Time

= (38 + 0 + 37 + 28 + 42) / 5

= 145/5 ms

= 29 ms

Solution: Gantt Chart:



Waiting Time = Total waiting Time – No. of milliseconds Process executed – Arrival Time

Priority Scheduling

Solved Problem - 2

Consider the set of processes with arrival time (in milliseconds), CPU burst time (in milliseconds), and priority shown below: (Higher number represents higher priority)

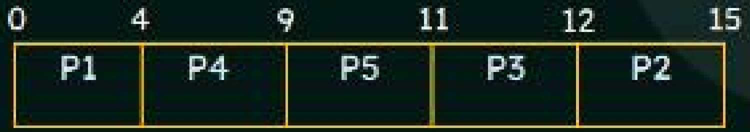
Process ID	Arrival Time	Burst Time	Priority
P1	0	4	2
P2	1	3	3
P3	2	1	4
P4	3	5	5
P5	4	2	5

If the CPU scheduling policy is priority non-preemptive, calculate the average waiting time and average turn around time.

Process ID	Arrival Time	Burst Time	Priority
P1	0	4	2
P2	1	3	3
P3	2	1	4
P4	3	5	5
P5	4	2	5

Process ID	Completion Time	Turnaround Time	Waiting Time
P1	4	4 - 0 = 4	4 - 4 = 0
P2	15	15 - 1 = 14	14 - 3 = 11
P3	12	12 - 2 = 10	10 - 1 = 9
P4	9	9 - 3 = 6	6 - 5 = 1
P5	11	11 - 4 = 7	7 - 2 = 5

Solution: **Gantt Chart:**



Turn Around time = Completion time - Arrival time

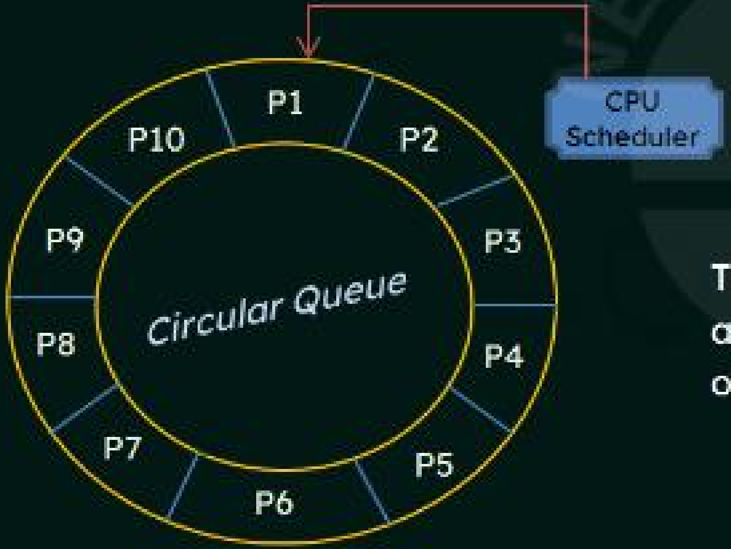
Waiting time = Turn Around time - Burst time

Average Turn Around time
= (4 + 14 + 10 + 6 + 7) / 5
= 41 / 5 = 8.2 ms

Average waiting time
= (0 + 11 + 9 + 1 + 5) / 5
= 26 / 5 = 5.2 ms

Scheduling Algorithms
(Round-Robin Scheduling)

- The round-robin (RR) scheduling algorithm is designed especially for timesharing systems.
- It is similar to FCFS scheduling, but preemption is added to switch between processes.
- A small unit of time, called a time quantum or time slice, is defined (generally from 10 to 100 milliseconds)



The Ready Queue

- The ready queue is treated as a circular queue.

The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.

Implementation of Round Robin scheduling:

- We keep the ready queue as a FIFO queue of processes.
- New processes are added to the tail of the ready queue.
- The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process.

Tail
Head
First In - First Out

One of two things will then happen

The process may have a CPU burst of less than 1 time quantum

↓

The process itself will release the CPU voluntarily

↓

The CPU scheduler will then proceed to the next process in the ready queue

The CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the O.S.

↓

A context switch will be executed, and the process will be put at the tail of the ready queue

↓

The CPU scheduler will then select the next process in the ready queue

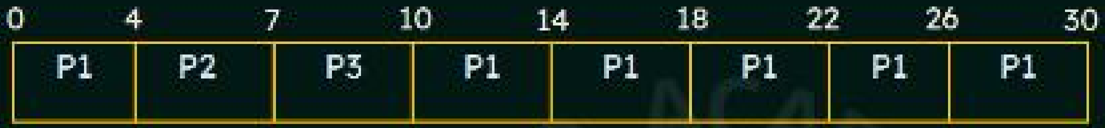
Round-Robin Scheduling
(Turnaround Time and Waiting Time)

Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds and time quantum taken as 4 milliseconds for RR Scheduling:

Process ID	Burst Time
P1	24
P2	3
P3	3

Time Quantum

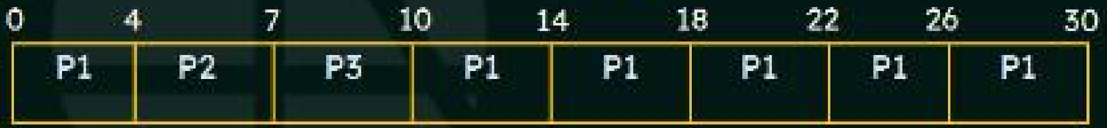
Gantt Chart:



Process ID	Burst Time
P1	24
P2	3
P3	3

Gantt Chart:

Time Quantum 4 ms



Method 1

Turn Around time = Completion time - Arrival time

Waiting time = Turn Around time – Burst time

Process ID	Completion Time	Turnaround Time	Waiting Time
P1	30	30 – 0 = 30	30 – 24 = 6
P2	7	7 – 0 = 7	7 – 3 = 4
P3	10	10 – 0 = 10	10 – 3 = 7

Average Turn Around time

= (30 + 7 + 10) / 3

= 47 / 3 = 15.66 ms

Average waiting time

= (6 + 4 + 7) / 3

= 17 / 3 = 5.66 ms

Method 2

Waiting time = Last Start Time - Arrival Time – (Preemption x Time Quantum)

Process ID	Waiting Time
P1	26 - 0 - (5x4) = 6
P2	4 - 0 - (0x4) = 4
P3	7 - 0 - (0x4) = 7

Average waiting time

= (6 + 4 + 7) / 3

= 17 / 3 = 5.66 ms

Round Robin Scheduling

Solved Problem

(Part - 1)

Consider the set of 5 processes whose arrival time and burst time are given below:

Process ID	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

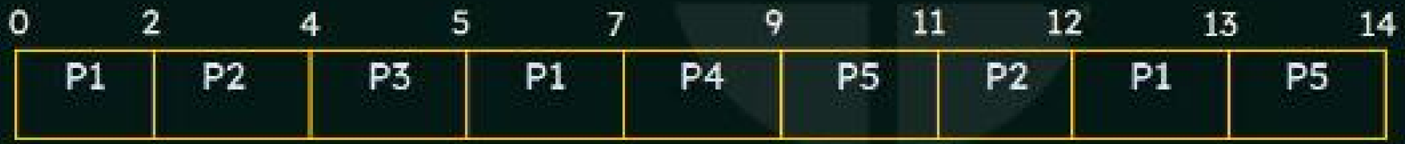
If the CPU scheduling policy is Round Robin with time quantum = 2 units, calculate the average waiting time and average turn around time.

Process ID	Arrival Time	Burst Time
P1	0	~~5~~ ~~3~~ ~~1~~
P2	1	~~3~~ ~~1~~
P3	2	~~1~~
P4	3	~~2~~
P5	4	~~3~~ ~~1~~

Time Quantum
2 Units

Clock
14

Gantt Chart:



Ready Queue

Round Robin Scheduling

Solved Problem

(Part - 2)

Consider the set of 5 processes whose arrival time and burst time are given below:

Process ID	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

If the CPU scheduling policy is Round Robin with time quantum = 2 units, calculate the average waiting time and average turn around time.

Scheduling Algorithms
(Multilevel Queue Scheduling)

A class of scheduling algorithms has been created for situations in which processes are easily classified into different groups.

Example:

Foreground Processes (Interactive)	They have: - Different response-time requirements - Different scheduling needs	Background Processes (Batch)

In addition, foreground processes may have priority (externally defined) over background processes.

A multilevel 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.

Example:

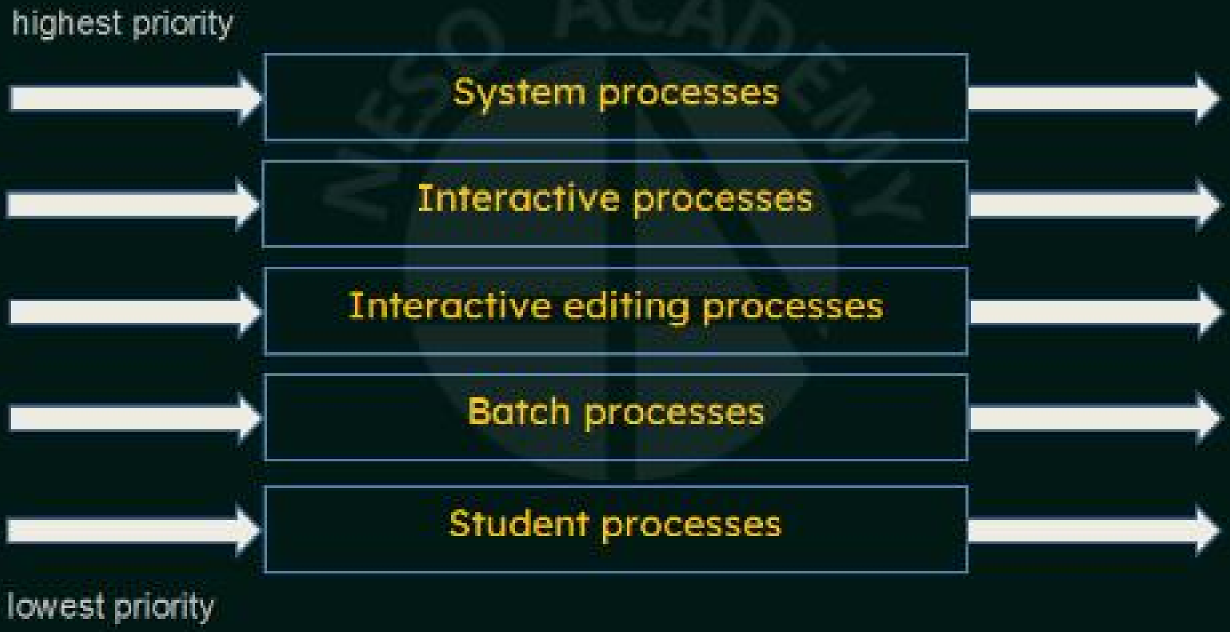
Separate queues might be used for foreground and background processes.

The foreground queue might be scheduled by an RR algorithm, while the background queue is scheduled by an FCFS algorithm.

In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling.

For example, the foreground queue may have absolute priority over the background queue.

An example of a multilevel queue scheduling algorithm with five queues, listed below in order of priority:



Scheduling Algorithms
(Multilevel Feedback-Queue Scheduling)

The multilevel feedback-queue scheduling algorithm allows a process to move between queues.

- The idea is to separate processes according to the characteristics of their CPU bursts.
- If a process uses too much CPU time, it will be moved to a lower-priority queue.
- This scheme leaves I/O-bound and interactive processes in the higher-priority queues.
- In addition, 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.

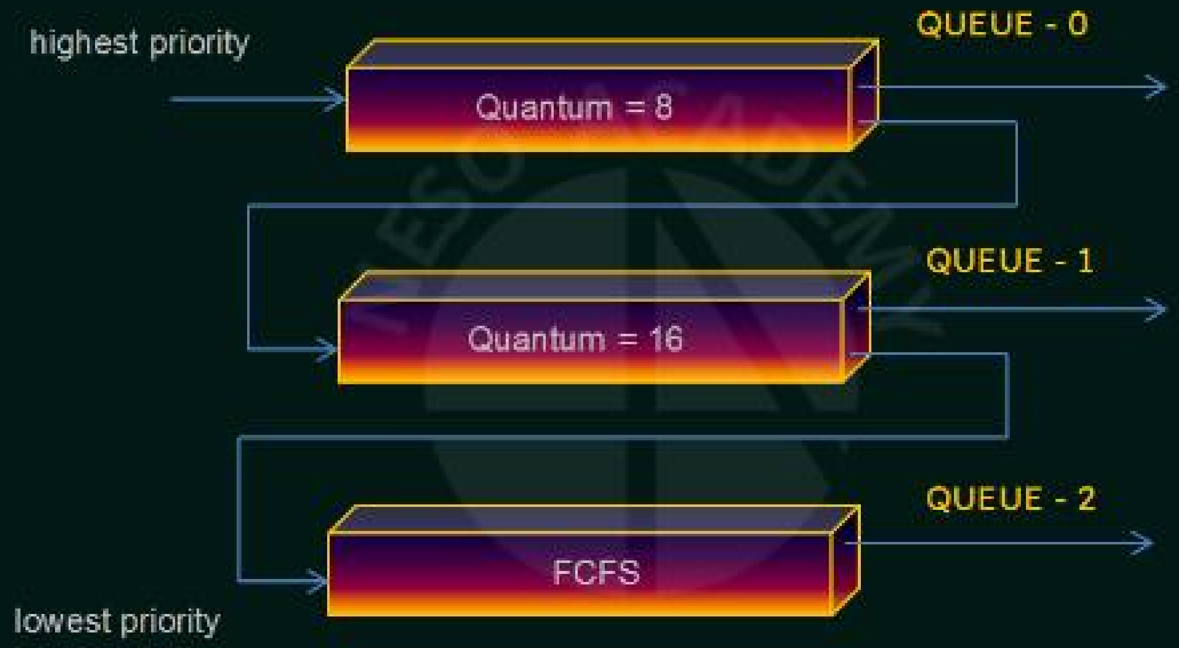


Figure: Multilevel feedback queues

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

Scheduling Algorithms
(Solved Problems)

Question: 1 GATE 2001

Consider a set of n tasks with known runtimes $r_1, r_2, \ldots r_n$ to be run on a uniprocessor machine. Which of the following processor scheduling algorithms will result in the maximum throughput?

(a) Round-Robin

(b) Shortest-Job-First

(c) Highest-Response-Ratio-Next

(d) First-Come-First-Served

Question: 2 GATE 2002

Which of the following scheduling algorithms is non-preemptive?

(a) Round-Robin

(b) First In First Out

(c) Multilevel Queue Scheduling

(d) Multilevel Queue Scheduling with Feedback

Question: 3

GATE 2010

Which of the following statements are true?

I. Shortest remaining time first scheduling may cause starvation

II. Preemptive scheduling may cause starvation

III. Round Robin is better than FCFS in terms of response time

(a) I only

(b) I and III only

(c) II and III only

(d) I, II and III

Question: 4

GATE 2012

Consider the 3 processes, P1, P2 and P3 shown in the table.

Process ID	Arrival Time	Time Units Required
P1	0	5
P2	1	7
P3	3	4

The completion order of the 3 processes under the policies FCFS and RR2 (round robin scheduling with CPU quantum of 2 time units) are:

(a) FCFS: P1, P2, P3
RR2: P1, P2, P3

(b) FCFS: P1, P3, P2
RR2: P1, P3, P2

(c) FCFS: P1, P2, P3
RR2: P1, P3, P2

(d) FCFS: P1, P3, P2
RR2: P1, P2, P3

Question: 5

GATE 2013

A scheduling algorithm assigns priority proportional to the waiting time of a process. Every process starts with priority zero (the lowest priority). The scheduler re-evaluates the process priorities every T time units and decides the next process to schedule. Which one of the following is TRUE if the processes have no I/O operations and all arrive at time zero?

(a) This algorithm is equivalent to the first-come-first-serve algorithm.

(b) This algorithm is equivalent to the round-robin algorithm.

(c) This algorithm is equivalent to the shortest-job-first algorithm.

(d) This algorithm is equivalent to the shortest-remaining-time-first algorithm.