5. CPU Scheduling

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

In a single-processor system, only one process can run at a time.

Any others must wait until the CPU is freed 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 OS takes the CPU away from that process and gives the CPU to another process and this pattern continues.

CPU and I/O Burst Cycles

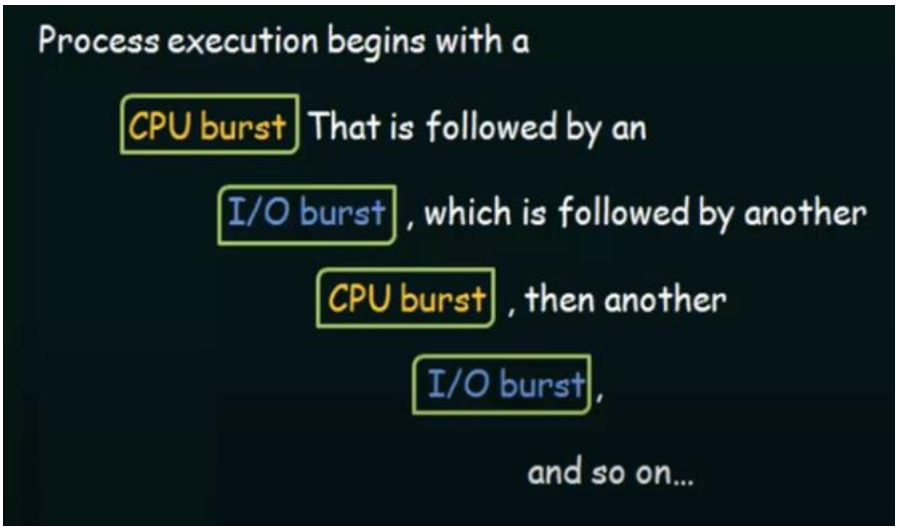
Process execution consists of a cycle of CPU execution and I/O wait.

After a process starts execution, it will be either in CPU execution or I/O wait.

Process alternates between these two states.

CPU execution is where 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.

Preemptive and Non-Preemptive Scheduling

CPU Scheduler

Whenever the CPU becomes idle, the OS 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 allocate 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 star 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 situation 2 and 3

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

Scheduling Criteria

There are several main criteria to compare scheduling algorithms.

1. **CPU utilization**
2. **Throughput**
3. **Turnaround time**
4. **Waiting time**
5. **Response time**

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% (for a lightly loaded system) to 90% (for a heavily loaded system).

Throughput

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

Turnaround time

From the point of view of a 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 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 Algorithms

First-Come, First-Served Scheduling (FCFS)

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.

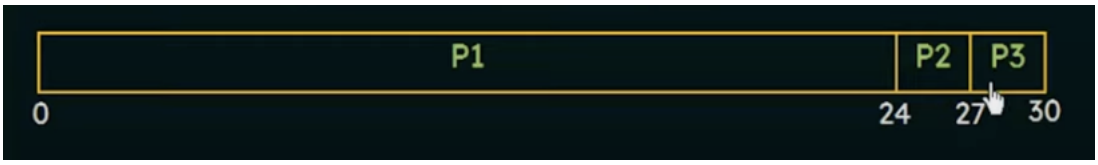
The running process is then removed from the queue.

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

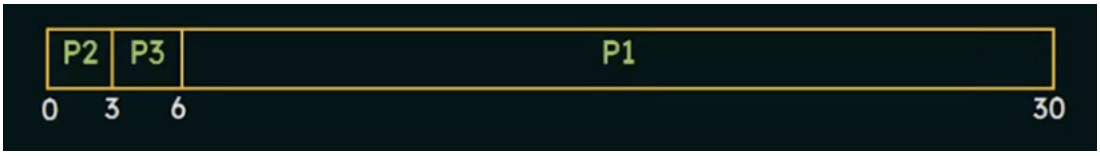
Consider the following set of processes that arrive at time 0

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:



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



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 non preemptive.

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.

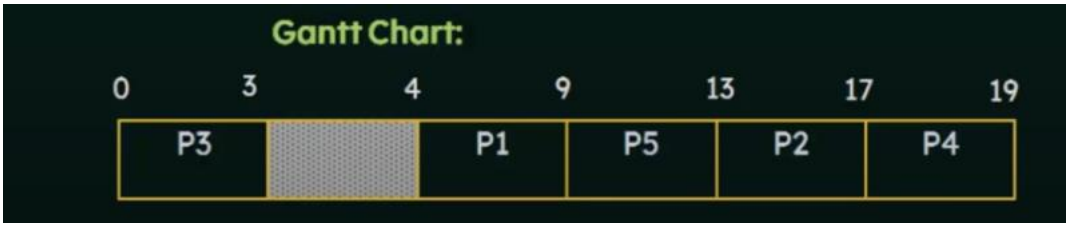
It would be disastrous to allow one process to keep the CPU for an extended period.

Problem

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 is followed. If two processes arrive at the same time, process with the smaller PID will be scheduled first.

Solution:



The shaded box represents the idle time of CPU

Turnaround time = Completion time – Arrival time
(Waiting time + Execution time)

Waiting time = Turnaround time – Burst time

Process ID	Completion Time	Turnaround Time	Waiting Time
P1	9	9 - 4 = 5	5 - 5 = 0
P2	17	17 - 6 = 11	11 - 4 = 7
P3	3	3 - 0 = 3	3 - 3 = 0
P4	19	19 - 6 = 13	13 - 2 = 11
P5	13	13 - 5 = 8	8 - 4 = 4

Average Turn Around time	= (5 + 11 + 3 + 13 + 8) / 5 = 40 / 5 = 8 units
Average waiting time	= (0 + 7 + 0 + 11 + 4) / 5 = 22 / 5 = 4.4 units

Problem

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 I unit of overhead in scheduling the processes, find the efficiency of the algorithm.

Solution:

Gantt Chart:

Here, δ denotes the unit overhead in scheduling the processes

SO ACADEMY

Useless time or Wasted time	= 6 x δ = 6 x 1 = 6 units	Efficiency (η)	= Useful time / Total Time = 17 units / 23 units = 0.7391 = 73.91%
Total time	= 23 units		
Useful time	= 23 units - 6 units = 17 units		

Scheduling Algorithms
Shortest-Job-first Scheduling (SJF)

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.

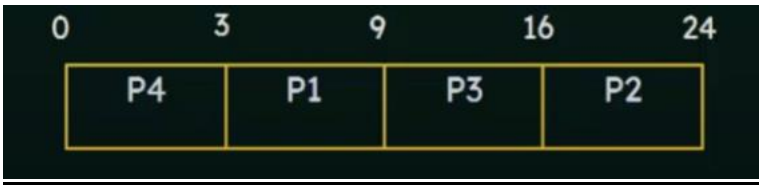
If the next CPU burst of two processes are the same, FCFS scheduling is used to break the tie.

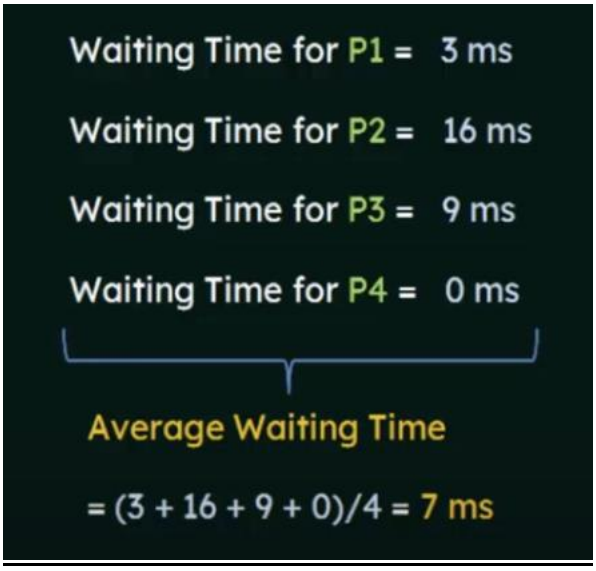
The SJF algorithm can be either preemptive or non-preemptive.

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-preemptive)

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

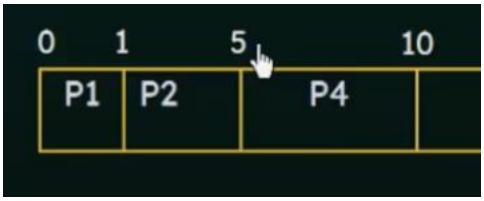




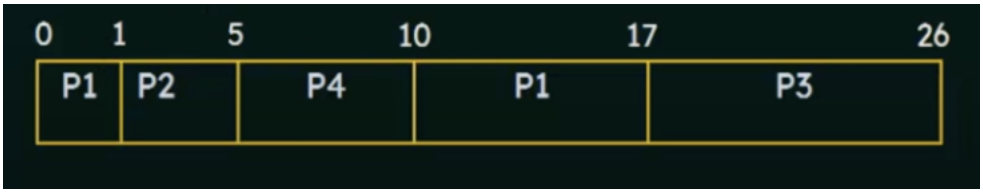
(FCFS would take an average waiting time of 10.25 milliseconds)

Example of SJF scheduling (preemptive)

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



Process ID	Arrival Time	Burst Time
P1	0	~~8~~ 7
P2	1	4
P3	2	9
P4	3	5



Waiting time = Total waiting time – No. of time units 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 exact length of the next CPU burst.

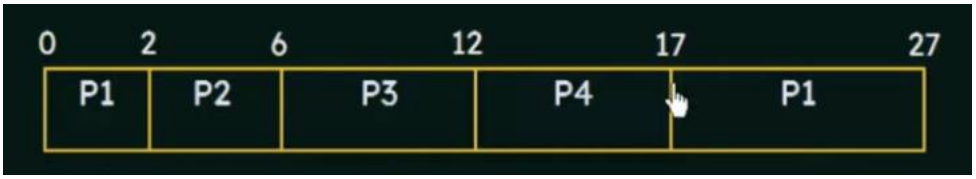
One approach is;

- 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.

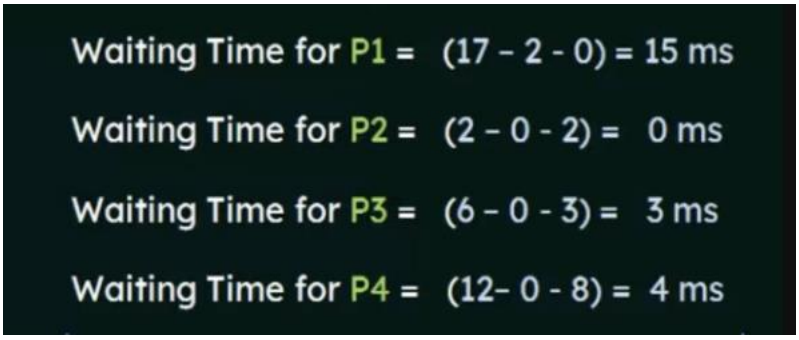
Problem

An OS uses shortest remaining time first scheduling algorithm for preemptive scheduling.

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



Waiting time = Total waiting time – No. of time units process executed – Arrival time

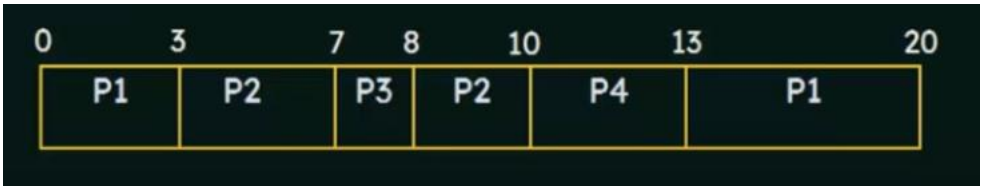


Average waiting time = (15 + 0 + 3 + 4)/4 = 5.5ms

Problem

Preemptive shortest job remaining first.

Process ID	Arrival Time	Burst Time
P1	0	~~10~~ 7
P2	3	~~6~~ 2
P3	7	1
P4	8	3



Turnaround time = Completion time – Arrival time
(Waiting time + Execution time)

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 = 8.25ms

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.

Priority scheduling can be either preemptive or non-preemptive

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.

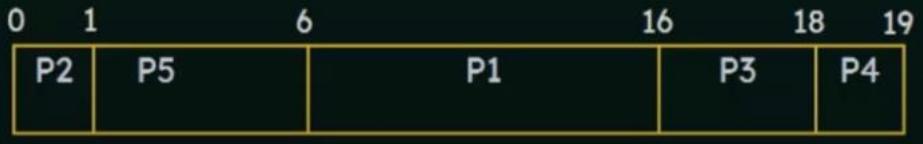
A non-preemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.

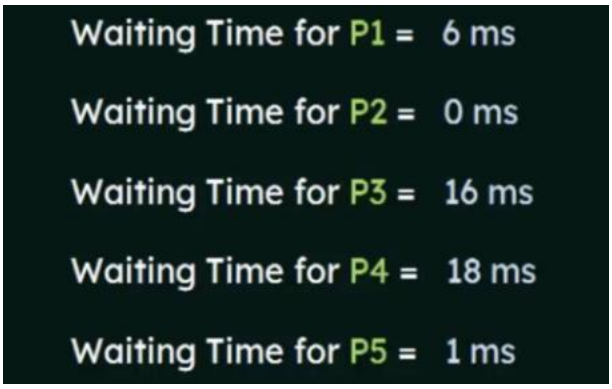
Example

Consider the following set of processes, assumed to have arrived to have arrive at time 0

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

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





Average waiting time = (6 + 0 + 16 + 18 + 1)/5 = 8.2ms

Problem with Priority Scheduling

- A major problem with priority scheduling algorithm 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.
- Example;
  If priorities range from 127 (low) to 0 (high), we could increase the priority of the 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.

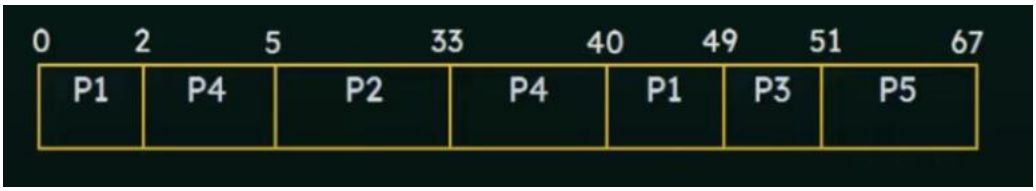
Problem

0 is the highest priority.

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…?

Process ID	Arrival Time	Burst Time	Priority
P1	0	~~11~~ ~~9~~	2
P2	5	~~28~~	0
P3	12	~~2~~	3
P4	2	~~10~~ ~~7~~	1
P5	9	~~16~~	4



Waiting time = Total waiting time – No. of time units process executed – Arrival time

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

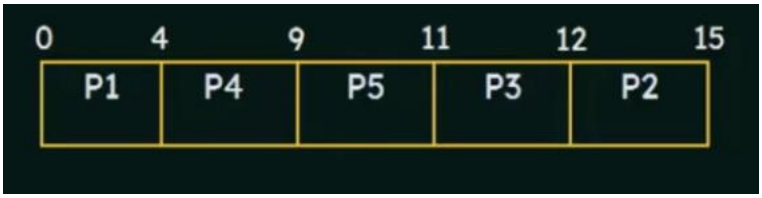
Waiting Time for P5 = (51 – 0 - 9) = 42 ms

Average waiting time = (38 + 0 + 37 + 28 + 42)/5 = 29ms

Problem

Higher number means higher priority. The CPU scheduling is priority non-preemptive. Calculate average waiting time? average turnaround 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



Turnaround time = Completion time – Arrival time

(Waiting time + Execution time)

Waiting time = Turnaround time – Burst time

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

Average turnaround time = (4 + 14 + 10 + 6 + 7)/5 = 8.2ms

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

Scheduling Algorithms

Round-Robin Scheduling

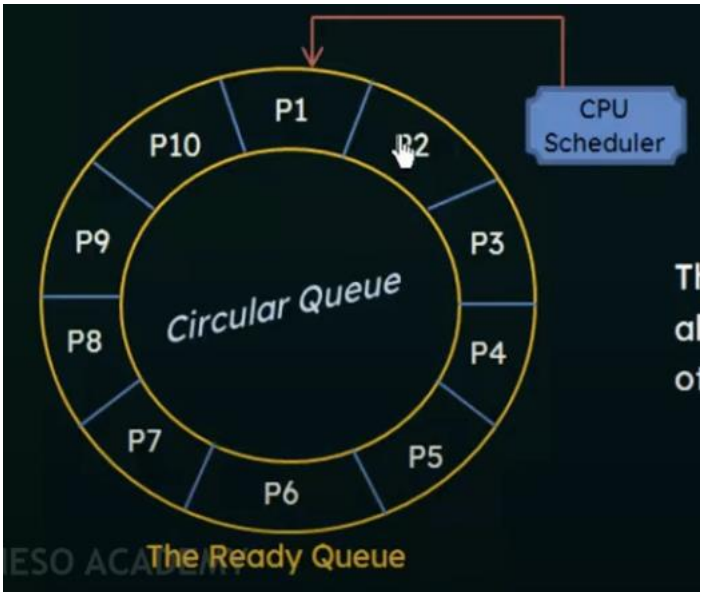
The round-robin (RR) scheduling algorithm is designed especially for timesharing systems.

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)

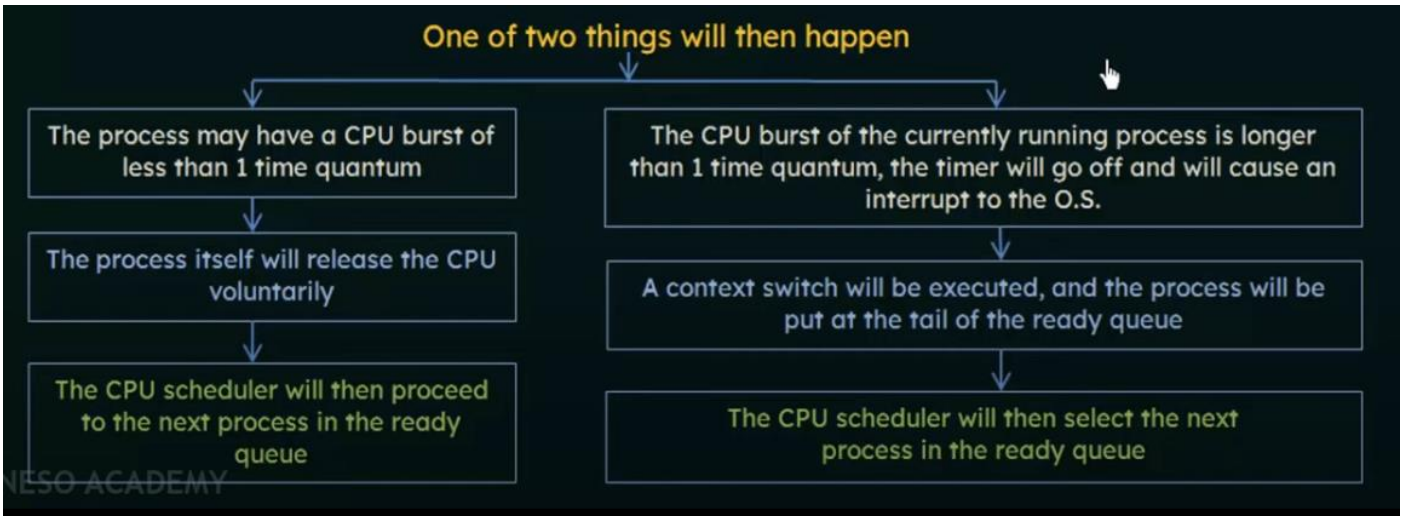
The ready queue is treated as a circular queue.

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.

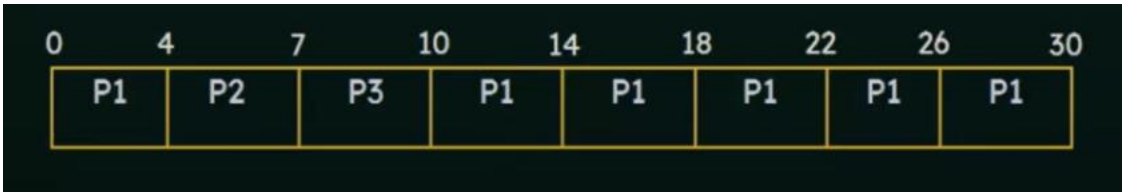


With large time slices, **starvation** could occur. With small time slices, **too many context switches** occur. Both are **bad.** We have to strike a balance between two.

Round Robin Scheduling (Turnaround time and waiting time)

Processes arrive at time 0, time quantum taken as 4 milliseconds for RR scheduling.

Process ID	Burst Time
P1	24
P2	3
P3	3



Turnaround time = Completion time – Arrival time
Waiting time = Turnaround time – Burst time

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

NESO ACADEMY

Average turnaround time = (30 + 7 + 10)/3 = 15.66ms
Average waiting time = (6 + 4 + 7)/3 = 5.66ms

Method 2

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

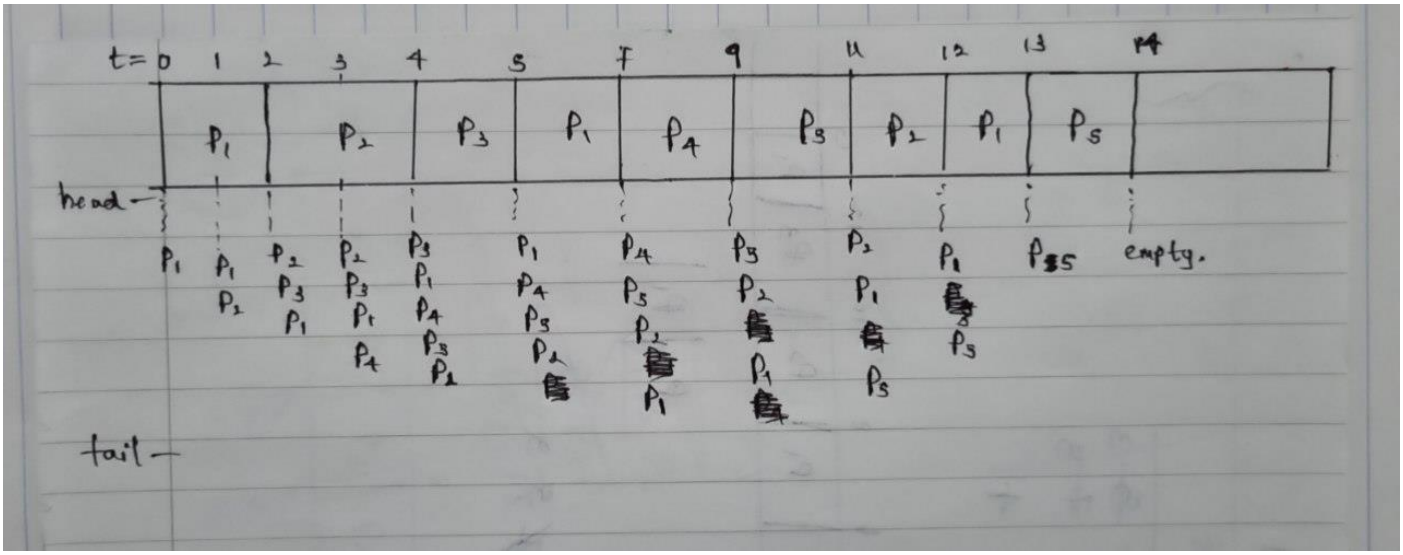
Process ID	Waiting Time
P1	26 - 0 - (5x4) = 6
P2	4 - 0 - (0x4) = 4
P3	7 - 0 - (0x4) = 7

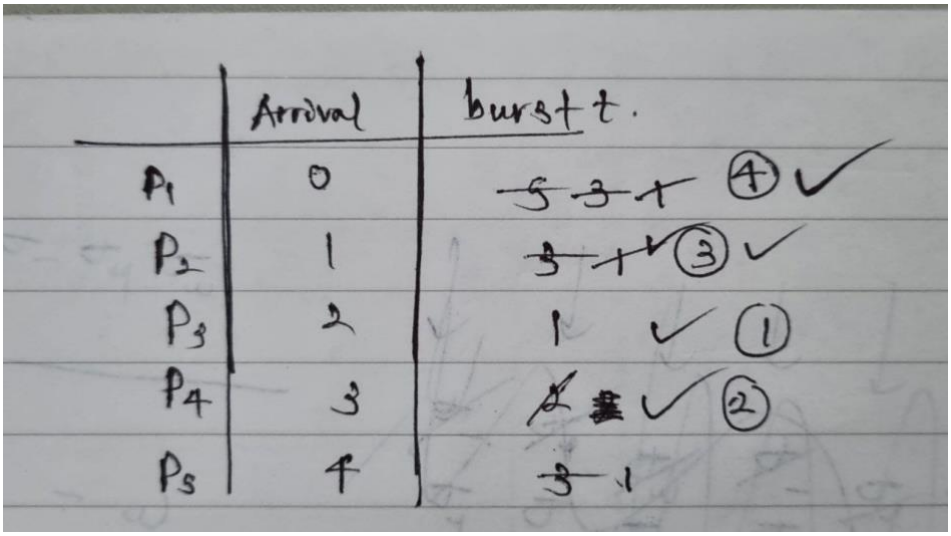
Average waiting time = (6 + 4 + 7)/3 = 5.66ms

Problem

Round robin with time quantum = 2 units

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





Turnaround time = Completion time – Arrival time
Waiting time = Turnaround time – Burst time

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

Process ID	Completion Time	Turnaround Time	Waiting Time
P1	13	13 – 0 = 13	13 – 5 = 8
P2	12	12 – 1 = 11	11 – 3 = 8
P3	5	5 – 2 = 3	3 – 1 = 2
P4	9	9 – 3 = 6	6 – 2 = 4
P5	14	14 – 4 = 10	10 – 3 = 7

Average turnaround time = (13 + 11 + 3 + 6 + 10)/5 = 8.6 units
Average waiting time = (8 + 8 + 2 + 4 + 7)/5 = 5.8 units

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

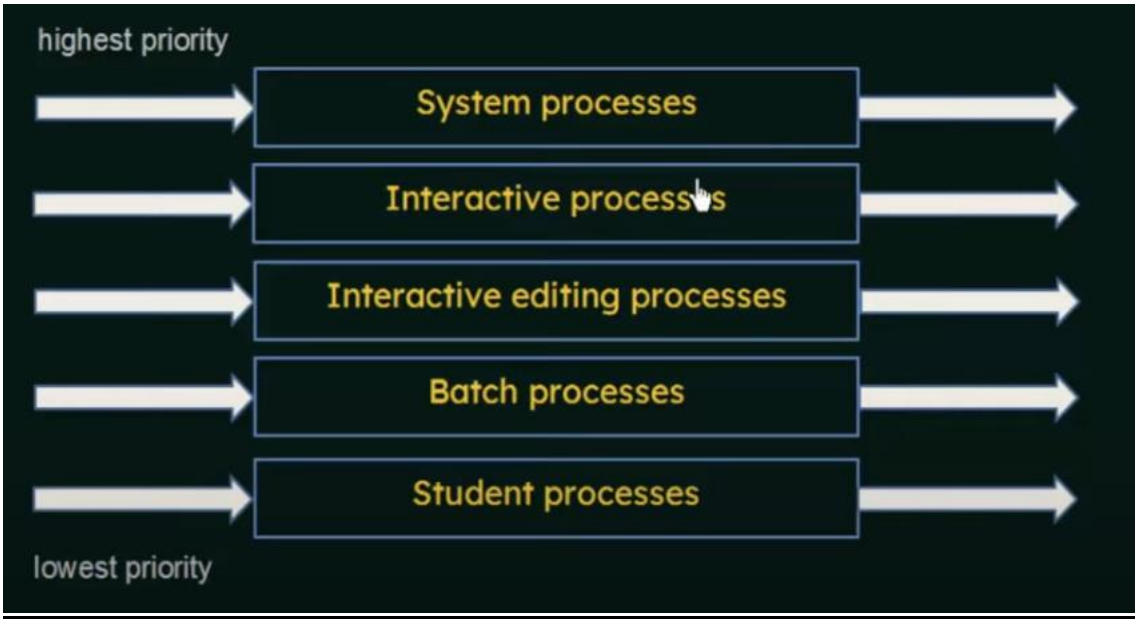
Separate queues might be used for foreground and background processes.

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.

Example



Multilevel Feedback-Queue Scheduling

This 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 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

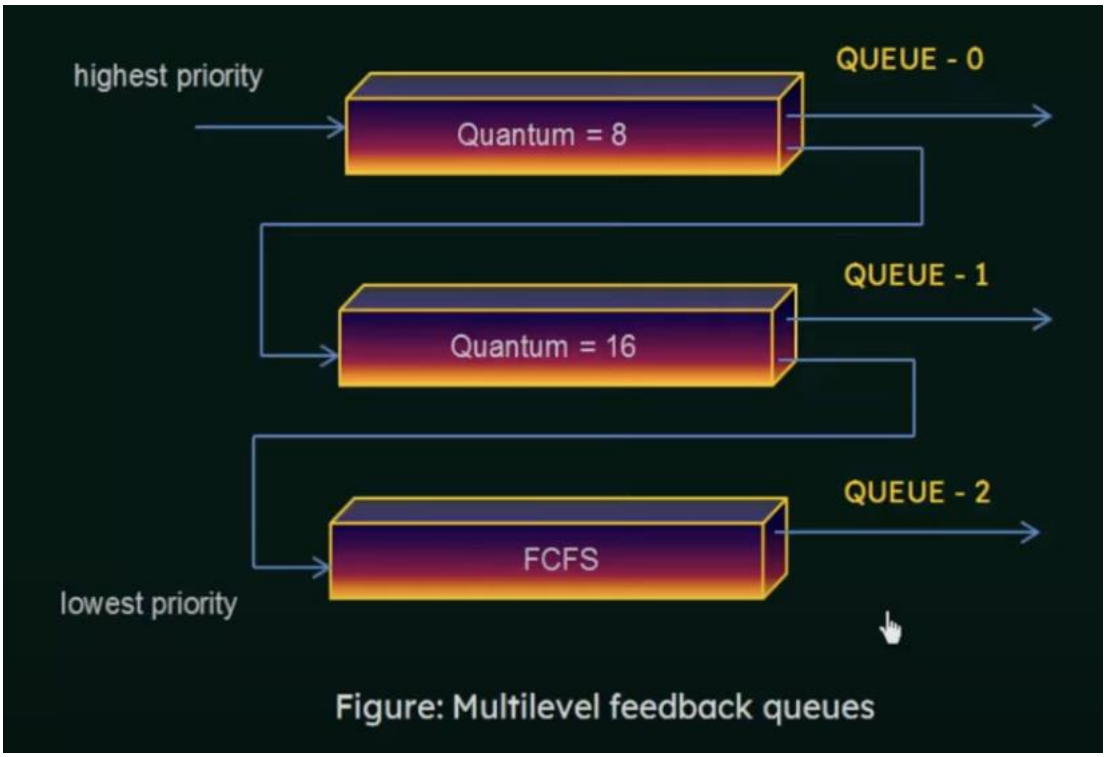


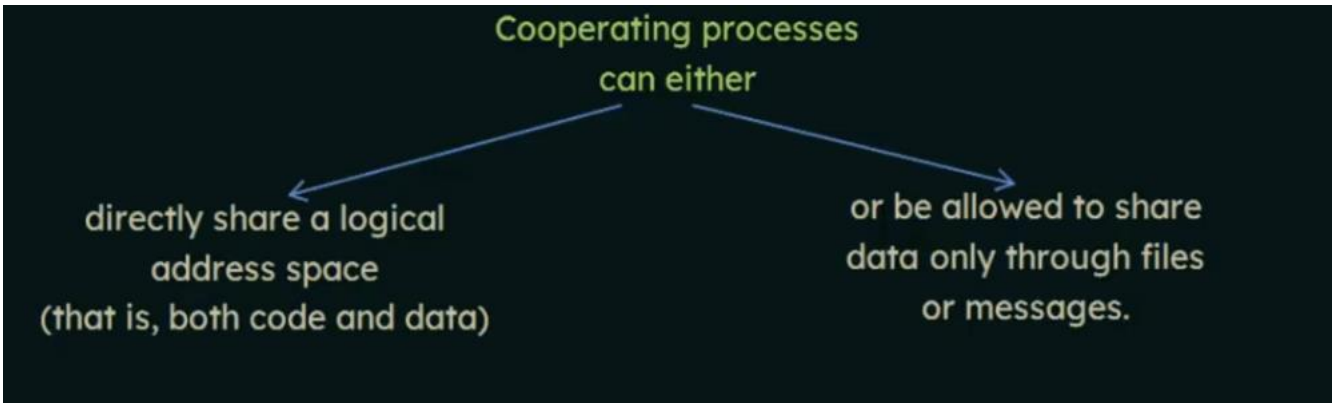
Figure: Multilevel feedback queues

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.

Process Synchronization.

A cooperating process is a process that can affect for get affected by other processes.



Concurrent access to share data may result in data inconsistency!

We will discuss various mechanisms to ensure the orderly execution of cooperating processes that share a logical space, so that data consistency is maintained.

Producer Consumer Problem

A producer process produces information that is consumed by a consumer process.

Ex: a compiler may produce assembly code, which is consumed by an assembler.

- One solution to this problem is shared memory.
- To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer.
- This buffer will reside in a region of memory that is shared by the two processes.
- A producer can produce one item while the consumer is consuming another item.
- These two must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

Two kinds of buffers

- **Unbounded buffer**
  No practical limit of size. The consumer may have to wait for new items, but producer can always produce new items.
- **Bounded buffer**
  Fixed finite size buffer. Consumer must wait if the buffer is empty, and producer must wait if the buffer is full.