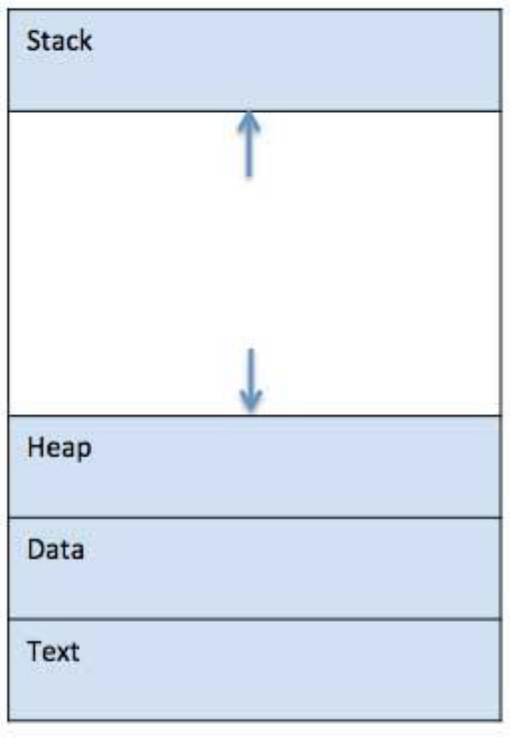
Process

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory −



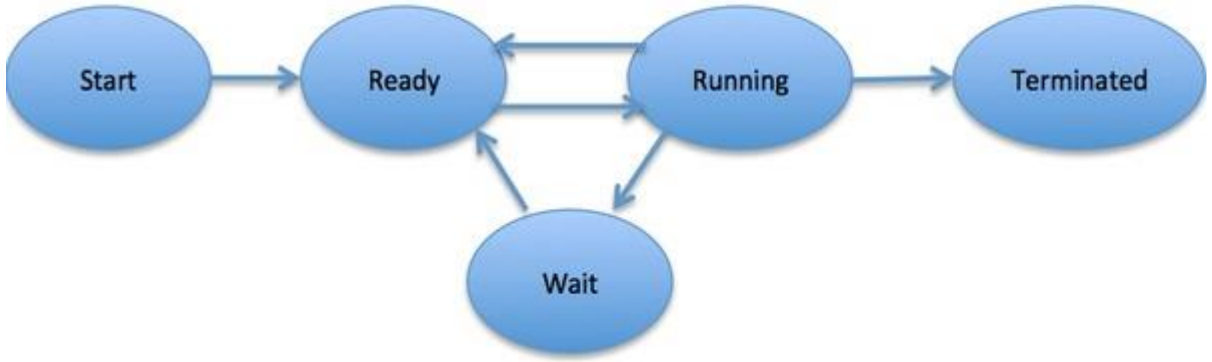
S.N.	Component & Description
1	**Stack** The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2	**Heap** This is dynamically allocated memory to a process during its run time.
3	**Text** This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4	**Data** This section contains the global and static variables.

Process States

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

In general, a process can have one of the following five states at a time.

S.N.	State & Description
1	**Start** This is the initial state when a process is first started/created.
2	**Ready** The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process.
3	**Running** Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.
4	**Waiting** Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
5	**Terminated or Exit** Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.



Representation of Process

Process Control Block (PCB)

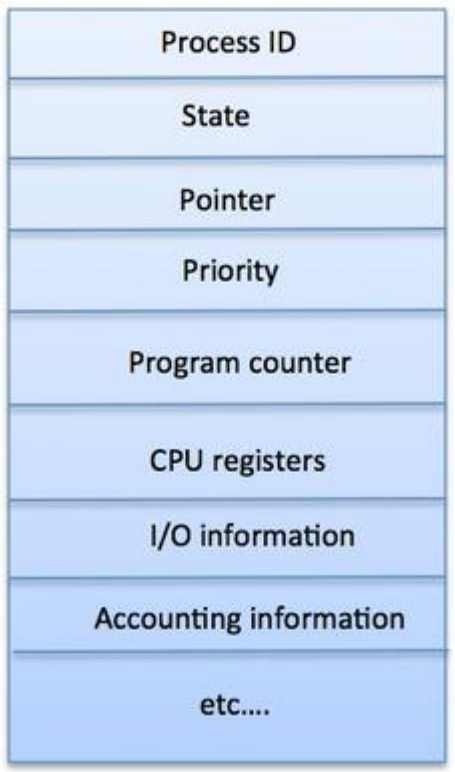
A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table –

S.N.	Information & Description
1	**Process State** The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	**Process privileges** This is required to allow/disallow access to system resources.
3	**Process ID**

	Unique identification for each of the process in the operating system.
4	**Pointer** A pointer to parent process.
5	**Program Counter** Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	**CPU registers** Various CPU registers where process need to be stored for execution for running state.
7	**CPU Scheduling Information** Process priority and other scheduling information which is required to schedule the process.
8	**Memory management information** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	**Accounting information** This includes the amount of CPU used for process execution, time limits, execution ID etc.

10	**IO status information** This includes a list of I/O devices allocated to the process.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB −



The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

Process Scheduling

Definition

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.

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

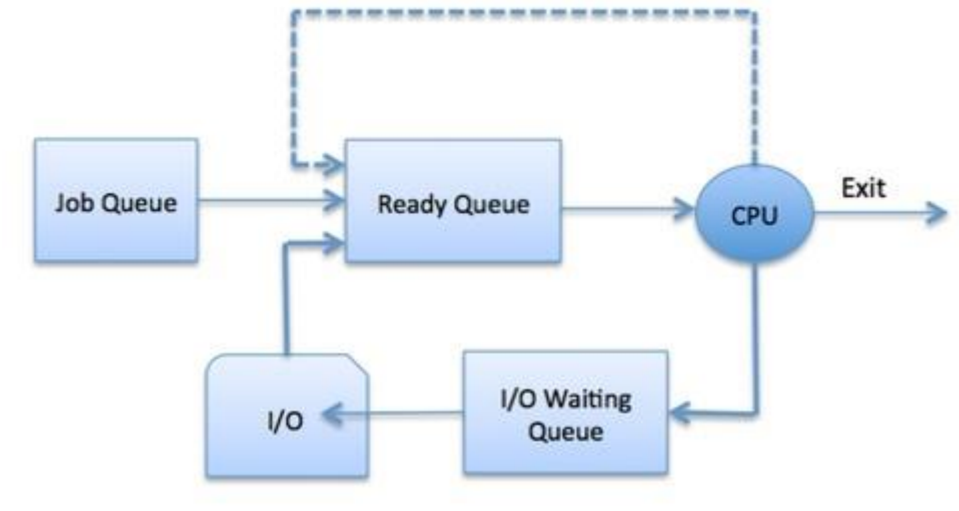
Process Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

- **Job queue** − This queue keeps all the processes in the system.
- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.

- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.



The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

Two-State Process Model

Two-state process model refers to running and non-running states which are described below −

S.N.	State & Description
1	**Running** When a new process is created, it enters into the system as in the running state.

2	**Not Running** Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute.

Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long-term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended process cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

Comparison among Scheduler

S.N.	**Long-Term Scheduler**	**Short-Term Scheduler**	**Medium-Term Scheduler**
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal	It is also minimal in time sharing system	It is a part of Time sharing systems.

	in time sharing system		
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

CPU Scheduling

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

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 from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

Types of CPU Scheduling

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

1. When a process switches from the **running** state to the **waiting** state (for I/O request or invocation of wait for the termination of one of the child processes).
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, completion of I/O).
4. When a process **terminates**.

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, i.e. The interval from time of submission of the process to the time of completion of the process(Wall clock time).

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).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

Scheduling Algorithms

To decide which process to execute first and which process to execute last to achieve maximum CPU utilization, computer scientists have defined some algorithms, they are:

1. First Come First Serve (FCFS) Scheduling
2. Shortest-Job-First (SJF) Scheduling
3. Priority Scheduling
4. Round Robin (RR) Scheduling
5. Multilevel Queue Scheduling
6. Multilevel Feedback Queue Scheduling

1.First Come First Serve Scheduling

In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.

- First Come First Serve, is just like **FIFO**(First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first.
- This is used in Batch Systems.
- It's **easy to understand and implement** programmatically, using a Queue data structure, where a new process enters through the **tail** of the queue, and the scheduler selects process from the **head** of the queue.

- A perfect real-life example of FCFS scheduling is **buying tickets at ticket counter**.

Calculating Average Waiting Time

For every scheduling algorithm, **Average waiting time** is a crucial parameter to judge it's performance.

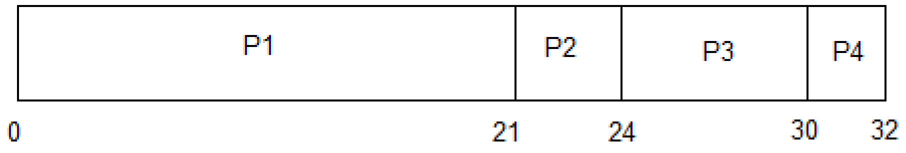
AWT or Average waiting time is the average of the waiting times of the processes in the queue, waiting for the scheduler to pick them for execution.

Lower the Average Waiting Time, better the scheduling algorithm.

Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with **Arrival Time** 0, and given **Burst Time**, let's find the average waiting time using the FCFS scheduling algorithm.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 + 21 + 24 + 30)/4 = 18.75 ms



This is the GANTT chart for the above processes

The average waiting time will be 18.75 ms

For the above given processes, first **P1** will be provided with the CPU resources,

- Hence, waiting time for **P1** will be 0
- **P1** requires 21 ms for completion, hence waiting time for **P2** will be 21 ms
- Similarly, waiting time for process **P3** will be execution time of **P1** + execution time for **P2**, which will be (21 + 3) ms = 24 ms.
- For process **P4** it will be the sum of execution times of **P1**, **P2** and **P3**.

The **GANTT chart** above perfectly represents the waiting time for each process.

Problems with FCFS Scheduling

Below we have a few shortcomings or problems with the FCFS scheduling algorithm:

1. It is **Non-Pre-emptive** algorithm, which means the **process priority** doesn't matter.

   If a process with very least priority is being executed, more like **daily routine backup** process, which takes more time, and all of a sudden, some other high priority process arrives, like **interrupt to avoid system crash**, the high priority process will have to wait, and hence in this case, the system will crash, just because of improper process scheduling.
2. Not optimal Average Waiting Time.
3. Resources utilization in parallel is not possible, which leads to **Convoy Effect**, and hence poor resource (CPU, I/O etc.) utilization.

What is Convoy Effect?

Convoy Effect is a situation where many processes, who need to use a resource for short time are blocked by one process holding that resource for a long time.

This essentially leads to poor utilization of resources and hence poor performance.

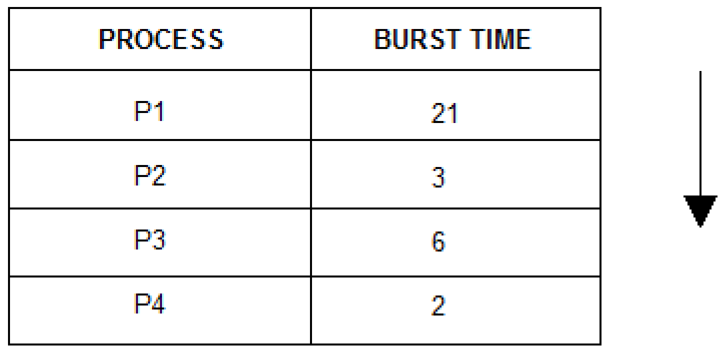
2.Shortest Job First (SJF) Scheduling

Shortest Job First scheduling works on the process with the shortest **burst time** or **duration** first.

- This is the best approach to minimize waiting time.
- This is used in Batch Systems.
- It is of two types:
    1. Non-Pre-emptive
    2. Pre-emptive
- To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time.
- This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (either Arrival time is 0 for all, or Arrival time is same for all)

Non-Pre-emptive Shortest Job First

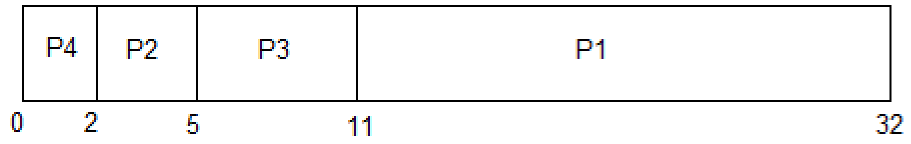
Consider the below processes available in the ready queue for execution, with **arrival time** as 0 for all and given **burst times**.



PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :



Now, the average waiting time will be = (0 + 2 + 5 + 11)/4 = 4.5 ms

As you can see in the **GANTT chart** above, the process **P4** will be picked up first as it has the shortest burst time, then **P2**, followed by **P3** and at last **P1**.

We scheduled the same set of processes using the First come first serve algorithm in the previous tutorial, and got average waiting time to be 18.75 ms, whereas with SJF, the average waiting time comes out 4.5 ms.

Problem with Non-Pre-emptive SJF

If the **arrival time** for processes are different, which means all the processes are not available in the ready queue at time 0, and some jobs arrive after some time, in such situation, sometimes process with short burst time have to wait for the current process's execution to finish, because in Non Pre-emptive SJF, on arrival of a process with short duration, the existing job/process's execution is not halted/stopped to execute the short job first.

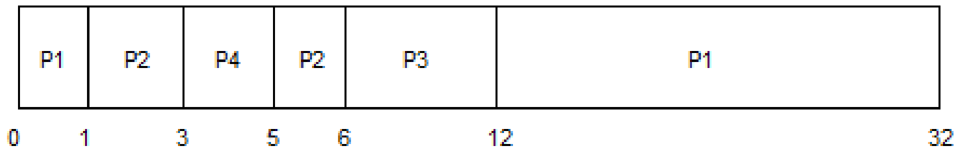
This leads to the problem of **Starvation**, where a shorter process has to wait for a long time until the current longer process gets executed. This happens if shorter jobs keep coming, but this can be solved using the concept of **aging**.

Pre-emptive Shortest Job First

In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with **short burst time** arrives, the existing process is preempted or removed from execution, and the shorter job is executed first.

PROCESS	BURST TIME	ARRIVAL TIME
P1	21	0
P2	3	1
P3	6	2
P4	2	3

The GANTT chart for Preemptive Shortest Job First Scheduling will be,



The average waiting time will be, ((5-3) + (6-2) + (12-1))/4 = 4.25 ms

The average waiting time for preemptive shortest job first scheduling is less than both, non-preemptive SJF scheduling and FCFS scheduling.

As you can see in the **GANTT chart** above, as **P1** arrives first, hence its execution starts immediately, but just after 1 ms, process **P2** arrives with a **burst time** of 3 ms which is less than the burst time of **P1**, hence the process **P1**(1 ms done, 20 ms left) is preempted and process **P2** is executed.

As **P2** is getting executed, after 1 ms, **P3** arrives, but it has a burst time greater than that of **P2**, hence execution of **P2** continues. But after another millisecond, **P4** arrives with a burst time of 2 ms, as a result **P2**(2 ms done, 1 ms left) is preempted and **P4** is executed.

After the completion of **P4**, process **P2** is picked up and finishes, then **P2** will get executed and at last **P1**.

The Pre-emptive SJF is also known as **Shortest Remaining Time First**, because at any given point of time, the job with the shortest remaining time is executed first.

3.Priority CPU Scheduling

In this tutorial we will understand the priority scheduling algorithm, how it works and its advantages and disadvantages.

In the Shortest Job First scheduling algorithm, the priority of a process is generally the inverse of the CPU burst time, i.e. the larger the burst time the lower is the priority of that process.

In case of priority scheduling the priority is not always set as the inverse of the CPU burst time, rather it can be internally or externally set, but yes the scheduling is done on the basis of priority of the process where the process which is most urgent is processed first, followed by the ones with lesser priority in order.

Processes with same priority are executed in FCFS manner.

The priority of process, when internally defined, can be decided based on **memory requirements**, **time limits** ,**number of open files**, **ratio of I/O burst to CPU burst** etc.

Whereas, external priorities are set based on criteria outside the operating system, like the importance of the process, funds paid for the computer resource use, market factor etc.

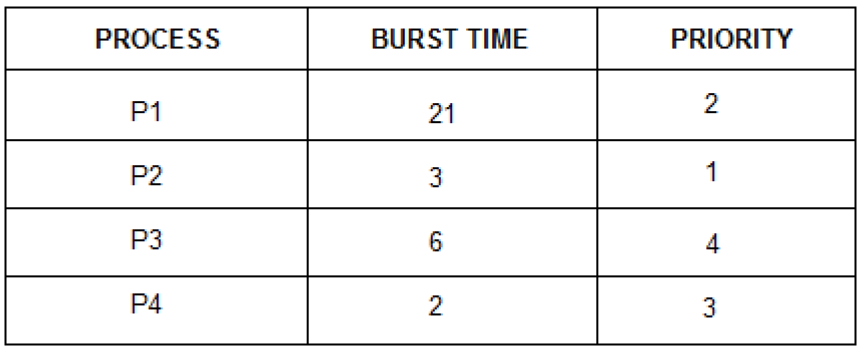
Types of Priority Scheduling Algorithm

Priority scheduling can be of two types:

1. **Preemptive Priority Scheduling**: If the new process arrived at the ready queue has a higher priority than the currently running process, the CPU is preempted, which means the processing of the current process is stoped and the incoming new process with higher priority gets the CPU for its execution.
2. **Non-Preemptive Priority Scheduling**: In case of non-preemptive priority scheduling algorithm if a new process arrives with a higher priority than the current running process, the incoming process is put at the head of the ready queue, which means after the execution of the current process it will be processed.

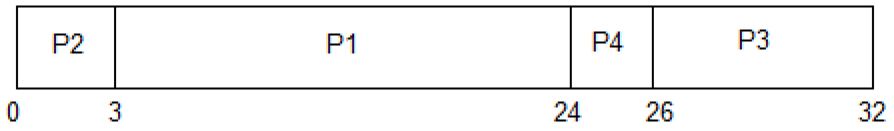
Example of Priority Scheduling Algorithm

Consider the below table fo processes with their respective CPU burst times and the priorities.



PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

The GANTT chart for following processes based on Priority scheduling will be,



The average waiting time will be, (0 + 3 + 24 + 26)/4 = 13.25 ms

As you can see in the GANTT chart that the processes are given CPU time just on the basis of the priorities.

Problem with Priority Scheduling Algorithm

In priority scheduling algorithm, the chances of **indefinite blocking** or **starvation**.

A process is considered blocked when it is ready to run but has to wait for the CPU as some other process is running currently.

But in case of priority scheduling if new higher priority processes keeps coming in the ready queue then the processes waiting in the ready queue with lower priority may have to wait for long durations before getting the CPU for execution.

In 1973, when the IBM 7904 machine was shut down at MIT, a low-priority process was found which was submitted in 1967 and had not yet been run.

Using Aging Technique with Priority Scheduling

To prevent starvation of any process, we can use the concept of **aging** where we keep on increasing the priority of low-priority process based on the its waiting time.

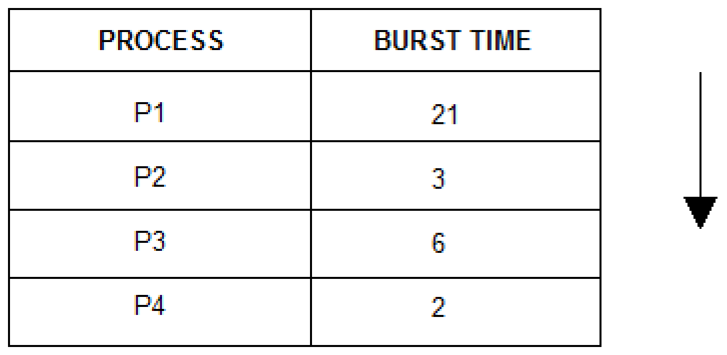
For example, if we decide the aging factor to be **0.5** for each day of waiting, then if a process with priority **20**(which is comparatively low priority) comes in the ready queue. After one day of waiting, its priority is increased to **19.5** and so on.

Doing so, we can ensure that no process will have to wait for indefinite time for getting CPU time for processing.

4.Round Robin Scheduling

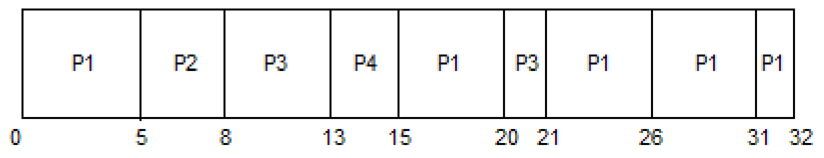
- A fixed time is allotted to each process, called **quantum**, for execution.
- Once a process is executed for given time period that process is preempted and other process executes for given time period.

- Context switching is used to save states of preemptied processes.



PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The GANTT chart for round robin scheduling will be,



The average waiting time will be, 11 ms.

5.Multilevel Queue Scheduling

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

For example: A common division is made between foreground (or interactive) processes and background (or batch) processes. These two types of processes have different response-time requirements, and so might have different scheduling needs. In addition, foreground processes may have priority over background processes.

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.

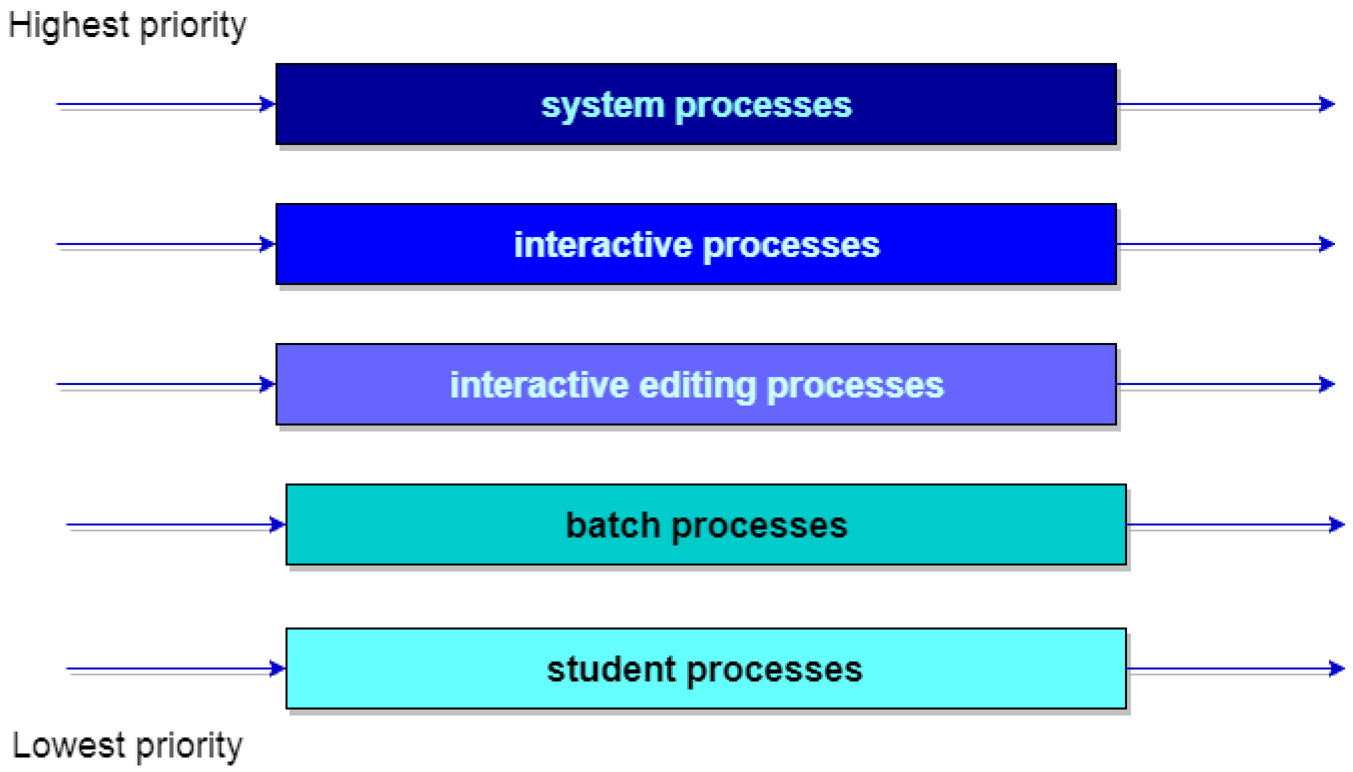
For example: separate queues might be used for foreground and background processes. The foreground queue might be scheduled by Round Robin algorithm, while the background queue is scheduled by an FCFS algorithm.

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.

Let us consider an example of a multilevel queue-scheduling algorithm with five queues:

1. System Processes
2. Interactive Processes
3. Interactive Editing Processes
4. Batch Processes
5. Student Processes

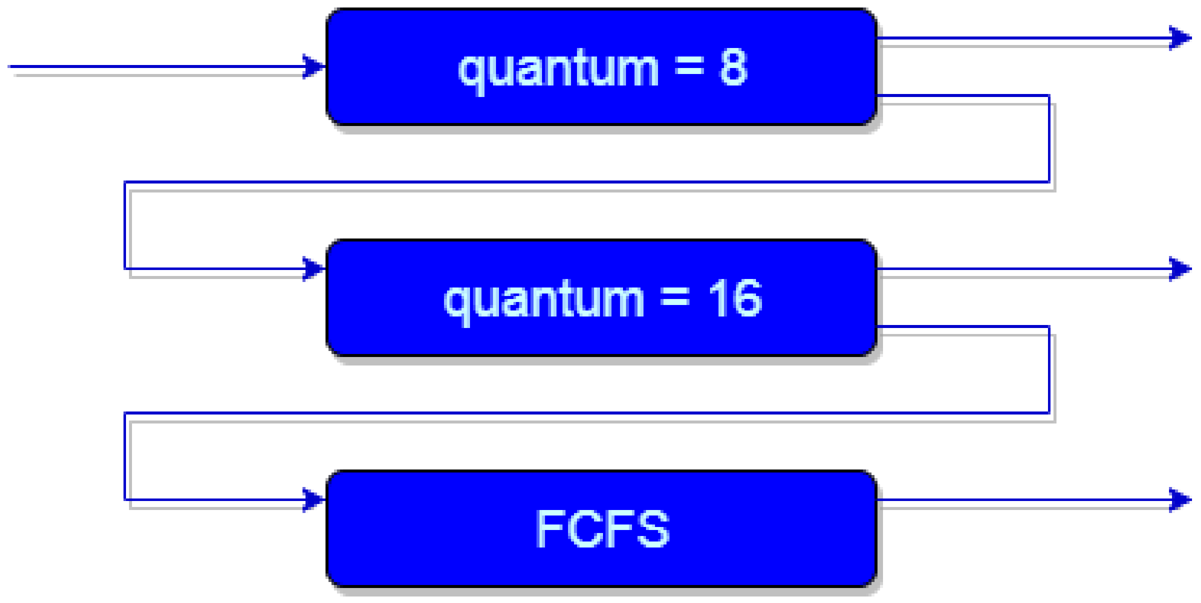
Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process will be preempted.



6.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 can be seen in the below figure.

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**.