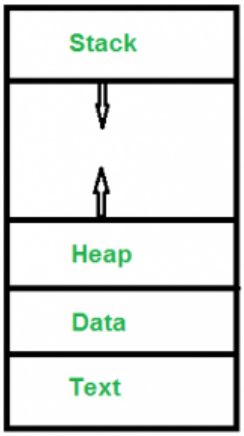
Process management - Introduction to process

Introduction to process: A process is a program in execution or a process is an 'active' entity, as opposed to a program, which is considered to be a 'passive' entity.
A single program can create many processes when running multiple times; when we open a .exe or binary file multiple times, multiple instances begin (multiple processes are created).
For example,
When we write a program in C or C++ and compile it, the compiler creates binary code. The original code and binary code are both programs. When we actually run the binary code, it becomes a process.

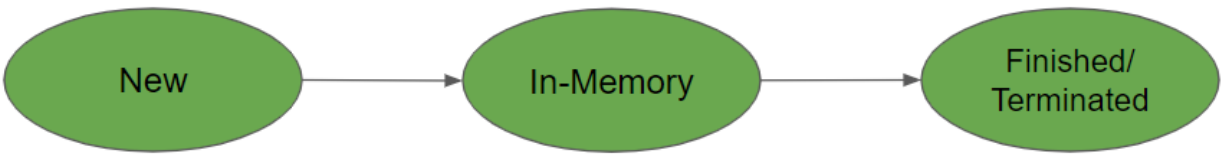
What does a process look like in memory?



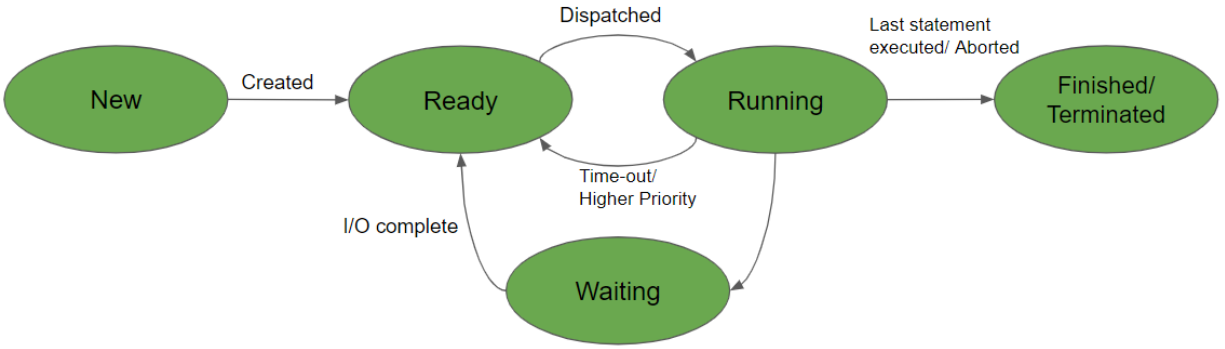
Text Section: A Process, sometimes known as the Text Section, also includes the current activity represented by the value of the *Program Counter*.
Stack: The Stack contains the temporary data such as function parameters, returns addresses, and local variables.
Data Section: Contains the global variable.
Heap Section: Dynamically allocated memory to process during its run time.
Refer to this for more details on sections.

Process States

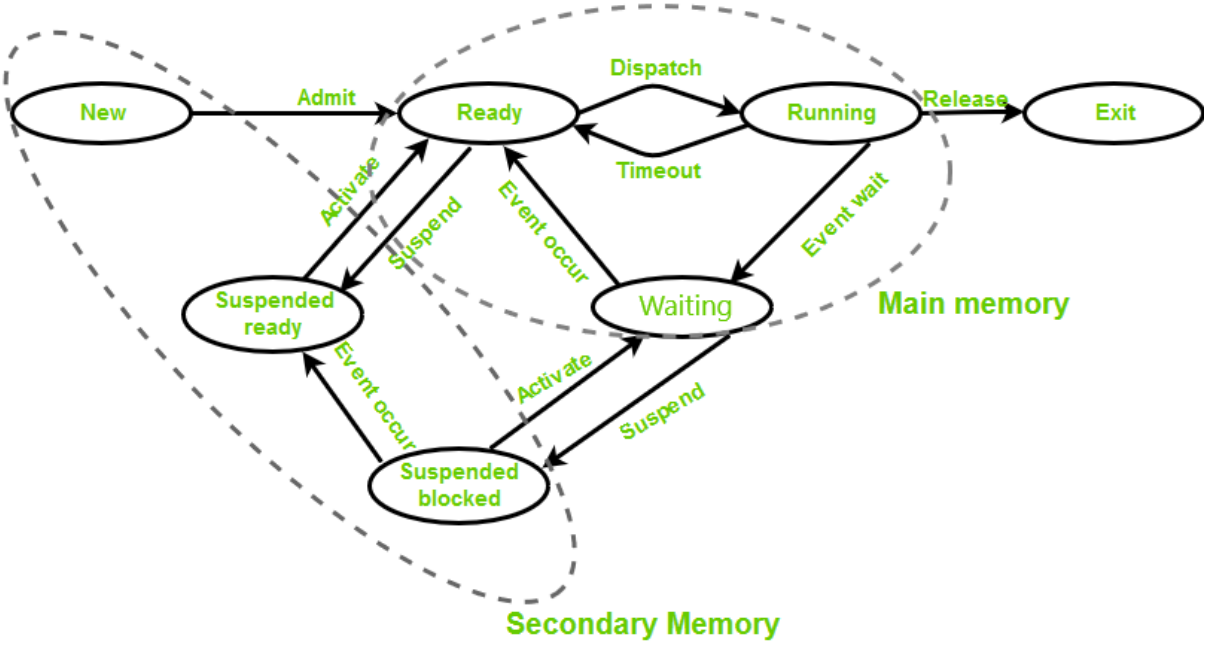
Single Tasking Systems(MS-DOS): In this kind of system, the second process begins only when the first process ends. By the time one process completes there might be other I/O devices, waiting for the first process to complete its task. This might lead to a delay in the process of the operating system, which is not feasible from the user's point of view. Therefore, the need arose for a multiprogramming system that can execute multiple processes at a given time. Given below is the process state diagram of the following:



Multiple programming System: In this system, multiple programs share single hardware resources. However, fake parallelism is achieved by context-switching between the processes. Responsiveness between multiple programs is achieved in such a manner. The number of programs that can simultaneously run thus depends upon the size of the main memory.
There are various states through which a processor passes to complete a particular or multiple executions. This is explained below using the process state diagram.
Here is the basic diagram showing a 5-state model and an advanced diagram showing the 7-state model.



5-State Model



7-State Model

States of a process are as following:

- **New (Create) -** In this step, the process is about to be created, but it is not yet created. It is the program that is present in secondary memory that will be picked up by OS to create the process.

- **Ready -** New -> Ready to run. After the creation of a process, the process enters the ready state i.e. the process is loaded into the main memory. The process here is ready to run and is waiting to get the CPU time for its execution. Processes that are ready for execution by the CPU are maintained in a queue for ready processes.

- **Run -** The process is chosen by the CPU for execution and the instructions within the process are executed by any one of the available CPU cores.

- **Blocked or wait -** Whenever the process requests access to I/O or needs input from the user or needs access to a critical region(the lock for which is already acquired) it enters the blocked or waits for the state. The process continues to wait in the main memory and does not require a CPU. Once the I/O operation is completed the process goes to the ready state.

- **Terminated or completed -** Process is killed as well as PCB is deleted.

- **Suspend ready -** Process that was initially in the ready state but were swapped out of main memory(refer Virtual Memory topic) and placed onto external storage by scheduler are said to be in suspending ready state. The process will transition back to the ready state whenever the process is again brought onto the main memory. Linux uses swap space and partition to do this task.

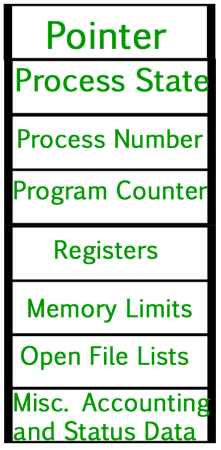
- **Suspend wait or suspend blocked -** Similar to suspend ready but uses the process which was performing I/O operation and lack of main memory caused them to move to secondary memory.

When work is finished it may go to suspend ready.

Process Control Block or PCB Various Attributes or Characteristics of a Process are:

```
1. Process Id: A unique identifier assigned by the operating system

2. Process State: Can be ready, running, etc.

3. CPU registers: Like the Program Counter (CPU registers must be saved and

                        restored when a process is swapped in and out of CPU)

4. Accounts information: This includes the amount of CPU used for the process

                        execution, time limits, execution ID etc.

5. I/O status information: For example, devices allocated to the process,

                        open files, etc.

6. CPU scheduling information: For example, Priority (Different processes

                        may have different priorities, for example, a short process maybe

                        assigned a low priority in the shortest job first scheduling)
```

All of the above attributes of a process are also known as the *context of the process*.
Every process has its own program control block(PCB), i.e each process will have a unique PCB. All of the above attributes are part of the PCB.
A process control block (PCB) contains information about the process, i.e. registers, quantum, priority, etc. The process table is an array of PCBs, which means logically contains a PCB for all of the current processes in the system.



Process Control Block

- Pointer - It is a stack pointer that is required to be saved when the process is switched from one state to another to retain the current position of the process.
- Process state - It stores the respective state of the process.
- Process number - Every process is assigned with a unique id known as process ID or PID which stores the process identifier.
- Program counter - It stores the counter which contains the address of the next instruction that is to be executed for the process.
- Register - These are the CPU registers which include: accumulator, base, registers, and general-purpose registers.
- Memory limits - This field contains information about the memory management system used by the operating system. This may include the page tables, segment tables etc.

- **Open files list -** This information includes the list of files opened for a process.

Context Switching The process of saving the context of one process and loading the context of another process is known as Context Switching. In simple terms, it is like loading and unloading the process from the running state to the ready state.

When does context switching happen? 1. When a high-priority process comes to a ready state (i.e. with higher priority than the running process).
2. An Interrupt occurs.
3. User and kernel-mode switch (It is not necessary though).
4. Preemptive CPU scheduling is used.

Context Switch vs Mode Switch A mode switch occurs when the CPU privilege level is changed, for example when a system call is made or a fault occurs. The kernel works in more a privileged mode than a standard user task. If a user process wants to access things that are only accessible to the kernel, a mode switch must occur. The currently executing process need not be changed during a mode switch.
A mode switch typically occurs for a process context switch to occur. Only the kernel can cause a context switch.

- Process management - Process scheduling (Algorithms and Criteria)

CPU and IO Bound Processes If the process is intensive in terms of CPU operations then it is called the CPU-bound process. Similarly, If the process is intensive in terms of I/O operations then it is called an IO-bound process.
There are three types of process schedulers.

1. **Long Term or job scheduler:** It brings the new process to the 'Ready State'. It controls the ***Degree of Multi-programming***, i.e., the number of processes present in the ready state at any point in time. It is important that the long-term scheduler makes a careful selection of both IO and CPU-bound processes.

2. **Short term or CPU scheduler:** It is responsible for selecting one process from the ready state for scheduling it on the running state.
   Note: Short-term scheduler only selects the process to schedule it doesn't load the process on running. The *Dispatcher* is responsible for loading the process selected by the Short-term scheduler on the CPU (Ready to Running State) Context switching is done by dispatcher only. A dispatcher does the following:
   1. Switching context.
   2. Switching to user mode.
   3. Jumping to the proper location in the newly loaded program.

3. **Medium-term scheduler** It is responsible for suspending and resuming the process. It mainly does swapping (moving processes from main memory to disk and vice versa). Swapping may be necessary to improve the process mix or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up.

Multiprogramming - We have many processes ready to run. There are two types of multiprogramming:

1. **Pre-emption -** Process is forcefully removed from CPU. Pre-emption is also called as time sharing or multitasking.

2. **Non pre-emption -** Processes are not removed until they complete the execution.

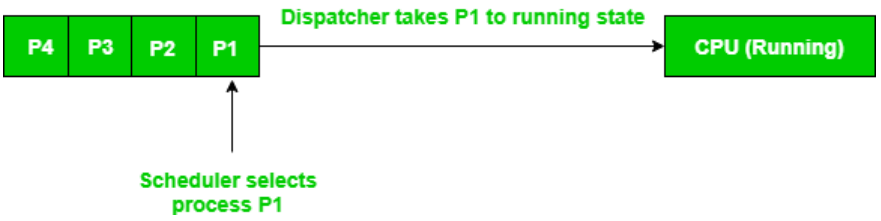
Degree of multiprogramming -
The number of processes that can reside in the ready state at maximum decides the degree of multiprogramming, e.g., if the degree of programming = 100 means 100 processes can reside in the ready state at maximum.

There are different queues of processes (in an operating system):

- **Ready Queue:** The set of all processes that are in main memory and are waiting for CPU time is kept in the ready queue.
- **Job Queue:** Each new process goes into the job queue. Processes in the job queue reside on mass storage and await the allocation of main memory.
- **Waiting (Device) Queues:** The set of processes waiting for allocation of certain I/O devices is kept in the waiting (device) queue. The short-term scheduler (also known as CPU scheduling) selects a process from the ready queue and yields control of the CPU to the process

Short-Term Scheduler and Dispatcher - Consider a situation, where various processes residing in the ready queue and waiting for execution. But CPU can't execute all the processes of ready queue simultaneously, the operating system has to choose a particular process on the basis of scheduling algorithm used. So, this procedure of selecting a process among various processes is done by **scheduler**. Now here the task of scheduler completed. Now **dispatcher** comes into the picture as scheduler has decided a process for execution, it is dispatcher who takes that process from ready queue to the running status, or you can say that providing CPU to that process is the task of the dispatcher.

Example - There are 4 processes in ready queue, i.e., P1, P2, P3, P4; They all have arrived at t0, t1, t2, t3 respectively. First in First out scheduling algorithm is used. So, scheduler decided that first of all P1 has come, so this is to be executed first. Now dispatcher takes P1 to the running state.



Difference between the Scheduler and Dispatcher:

Properties	DISPATCHER	SCHEDULER
Definition:	Dispatcher is a module that gives control of CPU to the process selected by short term scheduler	Scheduler is something which selects a process among various processes
Types:	There are no diifrent types in dispatcher.It is just a code segment.	There are 3 types of scheduler i.e. Long-term, Short-term, Medium-term
Dependency:	Working of dispatcher is dependednt on scheduler.Means dispatcher have to wait untill scheduler selects a process.	Scheduler works idependently.It works immediately when needed
Algorithm:	Dispatcher has no specific algorithm for its implementation	Scheduler works on various algorithm such as FCFS, SJF, RR etc.
Time Taken:	The time taken by dispatcher is called dispatch latency.	TIme taken by scheduler is usually negligible.Hence we neglect it.
Functions:	Dispatcher is also responsible for:Context Switching, Switch to user mode, Jumping to proper location when process again restarted	The only work of scheduler is selection of processes.

Below are different time with respect to a process.

1. **Arrival Time:** Time at which the process arrives in the ready queue.
2. **Completion Time:** Time at which process completes its execution.
3. **Burst Time:** Time required by a process for CPU execution.
4. **Turn Around Time:** Time Difference between completion time and arrival time.

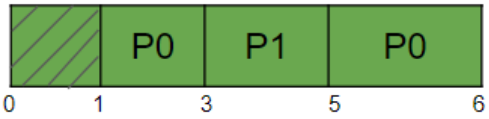
```
Turn Around Time = Completion Time - Arrival Time
```

5. **Waiting Time(W.T):** Time Difference between turn around time and burst time.

```
Waiting Time = Turn Around Time - Burst Time
```

6. **Response Time:** Time difference between arrival time and the first time the process gets CPU.

Example: Suppose we are given to calculate each type of time for the process P0 for the Gantt chart:



In the diagram from time 0 to 1, the CPU is free. Time slot 1 to 3 has been scheduled for P0, 3 to 5 has been scheduled for P1 and 5 to 6 are again scheduled to P0.
So for P0:

- Arrival Time (time of arrival into PC) = 1
- Completion Time (time of completion at PC) = 6
- Burst Time (Time taken by P0) = From 1 to 3 P0 takes 2 units of time and from 5 to 6 P0 takes 1 unit of time. So, in all Burst Time for P0 is 3.
- Turn Around Time = Completion Time - Arrival Time = 6 - 1 = 5.
- Waiting Time = Turn Around Time - Burst Time = 5 - 3 = 2.
- Response Time = 0 since the process got the CPU immediately.

Goals of a Scheduling Algorithm:

1. Maximum CPU Utilization: CPU must not be left idle and should be used to the full potential
2. Maximum Throughput: Throughput refers to the number of jobs completed per unit time. So the CPU must try to execute maximum number of jobs per unit time.
3. Minimum Turnaround time: Time between arrival and completion must be minimum.
4. Minimum Waiting Time: Total amount of time in the ready queue must be minimum.
5. Minimum Response Time: Difference between the arrival time and the time at which the process gets the CPU must be mininmum.
6. Fair CPU Allocation: No job must be starved.

Different Scheduling Algorithms

- *First Come First Serve (FCFS)* - Simplest scheduling algorithm that schedules according to arrival times of processes. First come first serve scheduling algorithm process that requests the CPU first is allocated the CPU first. It is implemented by using the FIFO queue. FCFS is a non-preemptive scheduling algorithm.
- *Shortest Job First (SJF)* - Process which have the shortest burst time are scheduled first.If two processes have the same bust time then FCFS is used to break the tie. It is a non-preemptive scheduling algorithm.
- *Longest Job First (LJF)* - It is similar to SJF scheduling algorithm. But, in this scheduling algorithm, we give priority to the process having the longest burst time. This is non-preemptive in nature i.e., when any process starts executing, can't be interrupted before complete execution.
- *Shortest Remaining Time First (SRTF)* - It is preemptive mode of SJF algorithm in which jobs are schedule according to shortest remaining time.
- *Longest Remaining Time First (LRTF)* - It is preemptive mode of LJF algorithm in which we give priority to the process having largest burst time remaining.

- *Round Robin Scheduling (RR)* - Each process is assigned a fixed time(Time Quantum/Time Slice) in cyclic way. It is designed especially for the time-sharing system. 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.

- *Priority Based scheduling (Non-Preemptive)* - In this scheduling, processes are scheduled according to their priorities, i.e., highest priority process is scheduled first. If priorities of two processes match, then schedule according to arrival time. Here starvation of process is possible.

- *Highest Response Ratio Next (HRRN)* - In this scheduling, processes with highest response ratio is scheduled. This algorithm avoids starvation.

  ```
  Response Ratio = (Waiting Time + Burst time) / Burst time
  ```

- *Multilevel Queue Scheduling* - According to the priority of process, processes are placed in the different queues. Generally high priority process are placed in the top level queue. Only after completion of processes from top level queue, lower level queued processes are scheduled. It can suuffer from starvation.

- *Multi level Feedback Queue Scheduling* - It allows the process to move in 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 is moved to a lower-priority queue.

Some useful facts about Scheduling Algorithms:

1. FCFS can cause long waiting times, especially when the first job takes too much CPU time.

2. Both SJF and Shortest Remaining time first algorithms may cause starvation. Consider a situation when the long process is there in the ready queue and shorter processes keep coming.

3. If time quantum for Round Robin scheduling is very large, then it behaves same as FCFS scheduling.

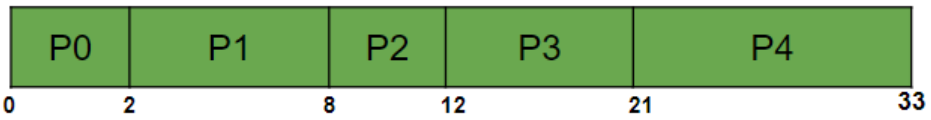
4. SJF is optimal in terms of average waiting time for a given set of processes,i.e., average waiting time is minimum with this scheduling, but problems are, how to know/predict the time of next job.

- FCFS Scheduling

FCFS (First-come-First-serve) follows the principle of FIFO (First-in-First-out). It is the simplest scheduling algorithm. FCFS simply queues processes in the order that they arrive in the ready queue. In this, the process which comes first will be executed first and the next process starts only after the previous gets fully executed. It is a non-preemptive algorithm. Let's look at this problem:

Process	Arrival Time	Burst Time
P0	0	2
P1	1	6
P2	2	4
P3	3	9
P4	4	12

Only the processes in the ready queue are to be considered. Since there is no preemption, once the job starts it is going to run until its completion. Now let's look at the Gantt chart:



Now, let's compute each type of time using concepts and formulae, that we studied in the previous lecture. Now the chart looks like this:

Process	Arrival Time	Burst Time	Completion Time	Turn-Around Time	Waiting Time
P0	0	2	2	2	0
P1	1	6	8	7	1
P2	2	4	12	10	6
P3	3	9	21	18	9
P4	4	12	33	29	17

So, Average waiting time = (0+1+6+9+17)/5 = 33/5
and Average turn-around time = (2+7+10+18+29)/5 = 74/5

Example:

FCFS (Example)

Process	Duration	Oder	Arrival Time
P1	24	1	0
P2	3	2	0
P3	4	3	0

Gantt Chart :

P1 waiting time : 0
P2 waiting time : 24
P3 waiting time : 27

The Average waiting time :

(0+24+27)/3 = 17

Implementation:

```
1-  Input the processes along with their burst time (bt).
2-  Find the waiting time (wt) for all processes.
3-  As the first process that comes need not to wait so
    waiting time for process 1 will be 0 i.e. wt[0] = 0.
4-  Find waiting time for all other processes i.e. for
     process i ->
       wt[i] = bt[i-1] + wt[i-1] .
5-  Find turnaround time = waiting_time + burst_time
    for all processes.
```

```
6-  Find average waiting time =
                total_waiting_time / no_of_processes.
7-  Similarly, find average turnaround time =
                total_turn_around_time / no_of_processes.
```

Important Points:

1. Simple and Easy to implement
2. Non-preemptive
3. Average Waiting Time is not optimal
4. Can not utilize resources in parallel: Results in Convoy effect. Convoy Effect is a phenomenon associated with the First Come First Serve (FCFS) algorithm, in which the whole Operating System slows down due to a few slow processes. Consider a situation when many IO-bound processes are there and one CPU bound process. The IO bound processes have to wait for CPU bound process when the CPU bound process acquires CPU. The IO-bound process could have taken CPU for some time, then used IO devices. The diagram below is a good analogy to the problem:

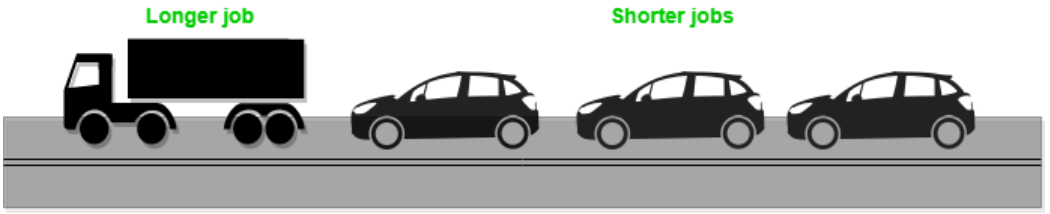


Figure - The Convey Effect, Visualized

- SJF Scheduling

The Shortest-Job-First (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. It is a non-preemptive algorithm. Being a non-preemptive algorithm, the first process assigned to the CPU is executed till completion, then the process whose burst time is minimum is assigned next to the CPU and hence it continues.

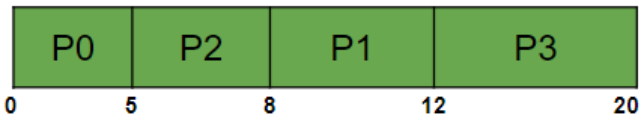
Algorithm:

```
1- Sort all the processes in increasing order
   according to burst time.
2- Then simply, apply FCFS.
```

Let's look at the following non preemptive SJFS problem and understand the stepwise execution:

Process	Arrival Time	Burst Time
P0	0	5
P1	1	4
P2	2	3
P3	3	8

Now lets draw the Gantt chart to the following problem:



How to compute below times in SJF using a program?

1. Completion Time: Time at which process completes its execution.
2. Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time - Arrival Time

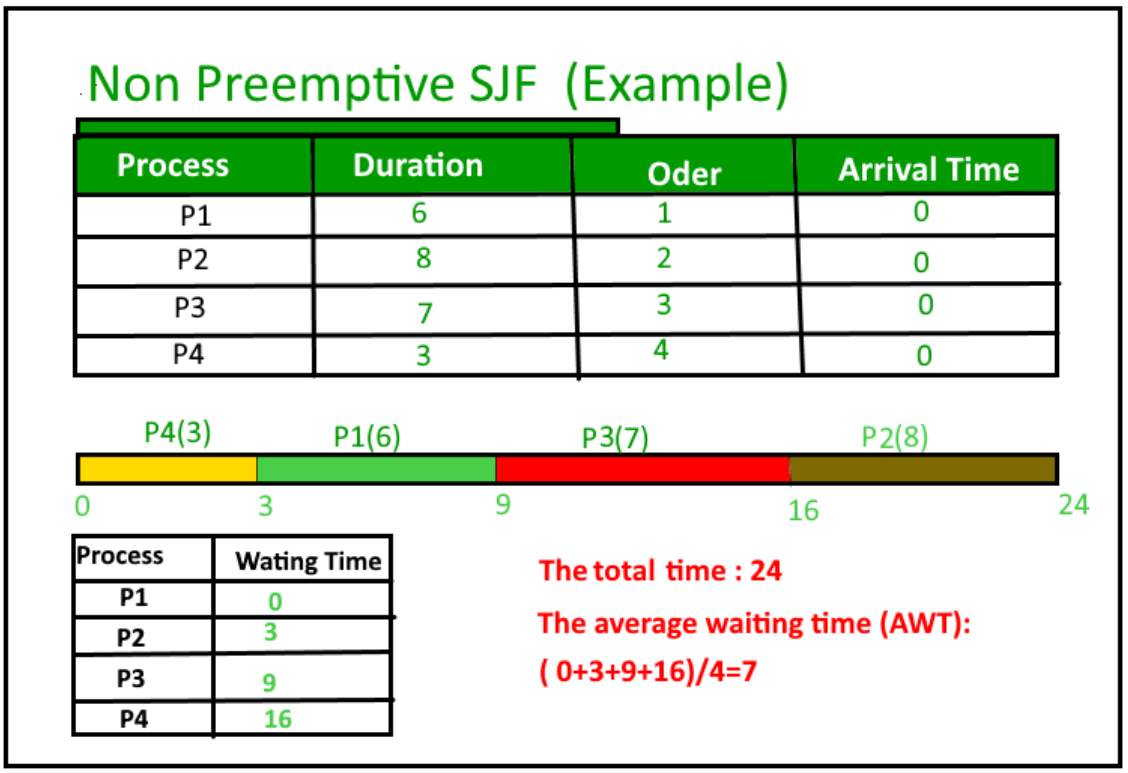
3. Waiting Time(W.T): Time Difference between turn around time and burst time.
   Waiting Time = Turn Around Time - Burst Time

Therefore the final chart showing all type of times are as follows:

Process	Arrival Time	Burst Time	Completion Time	Turn-Around Time	Waiting Time
P0	0	5	5	5	0
P1	1	4	12	11	7
P2	2	3	8	6	3
P3	3	8	20	17	9

So, Average turn-around time = (5+11+6+17)/4 = 39/4
and Average waiting time = (0+7+3+9)/4 = 19/4

Example:



Some of the key points are as follows:

- Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms.
- It is a Greedy Algorithm.
- It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing.
- It is practically infeasible as Operating System may not know burst time and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times. SJF can be used in specialized environments where accurate estimates of running time are available.

- SRTF Scheduling

In the previous post, we have discussed SJF which is a non-preemptive scheduling algorithm. In this post, we will discuss the preemptive version of SJF known as Shortest Remaining Time First (SRTF).

In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time.

Implementation:

```
1- Traverse until all process gets completely executed.
   a) Find process with minimum remaining time at every single time lap.
   b) Reduce its time by 1.
   c) Check if its remaining time becomes 0.
   d) Increment the counter of process completion.
   e) Completion time of current process = current_time +1;
   f) Calculate the waiting time for each completed process.
         wt[i]= Completion time - arrival_time-burst_time
   g) Increment time lap by one.
```

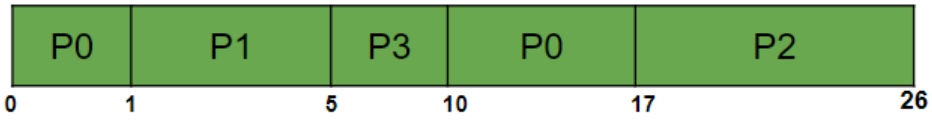
```
2- Find turnaround time (waiting_time+burst_time).

   Let's look at the below problem.
```

Process	Arrival Time	Burst Time
P0	0	8
P1	1	4
P2	2	9
P3	3	5

At first, the P0 is scheduled and it runs for 1 unit of time. P1 also around at 1 unit. So, now the remaining time of P0(i.e., 7) is compared to the burst time of P1(i.e., 4). The lesser is scheduled, here P1 is scheduled. Again it runs for 1 unit when P2 arrives. The same comparison is done. Since P1 has 3 remaining unit time which is less than P2, it is continued to be scheduled. At 3 P3 also arrives whose burst time is 5. Being more than the remaining time of P2 that is 2, P2 is continued to be scheduled until completion. Then P3 is scheduled whose burst time is 5. This runs till completion followed by P2. All the breakdowns can be seen in the below and the Gantt chart:

Process	Arrival Time	Burst Time
P0	0	~~8~~ 7
P1	1	~~4~~ ~~3~~ 2
P2	2	9
P3	3	5

Gantt chart for the process:



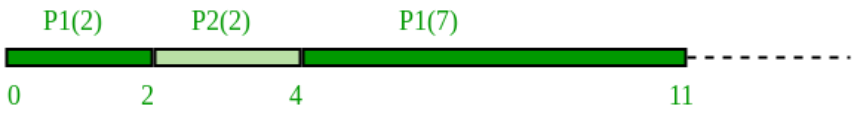
Now let's calculate all other time using the basic formulae.

Process	Arrival Time	Burst Time	Completion Time	Turn-Around Time	Waiting Time
P0	0	8	17	17	9
P1	1	4	5	4	0
P2	2	9	26	24	15
P3	3	5	10	7	2

So, Average turn around time = (17+4+24+7)/4 = 52/4
and Average waiting time = (9+0+15+2)/4 = 26/4

Example:

Process	Duration	Order	Arrival Time
P1	9	1	0
P2	2	2	2



P1 waiting time: 4-2 = 2
P2 waiting time: 0
The average waiting time(AWT): (0 + 2) / 2 = 1

Advantage:

1. Short processes are handled very quickly.
2. The system also requires very little overhead since it only makes a decision when a process completes or a new process is added.
3. When a new process is added the algorithm only needs to compare the currently executing process with the new process, ignoring all other processes currently waiting to execute.

Disadvantage:

1. Like shortest job first, it has the potential for process starvation.
2. Long processes may be held off indefinitely if short processes are continually added.
3. It is impractical since it is not possible to know the burst time of every process in ready queue in advance.

- LRTF Scheduling

LRTF (Longest-Remaining-Time-First) is the pre-emptive version of the LJF algorithm. In this scheduling algorithm, we find the process with the maximum remaining time and then process it. We check for the maximum remaining time after some interval of time (say 1 unit each) to check if another process having more Burst Time arrived up to that time.
The algorithm stands as:

```
1) Sort the processes in increasing order of their arrival time.
2) Choose the process having the least arrival-time but the most burst-time.
   Execute it for 1 unit/quantum.
   Check if any other process arrives upto that point of execution.
3) Repeat above steps until all processes have been executed.
```

As an example, consider the following 4 processes (P1, P2, P3, P4):

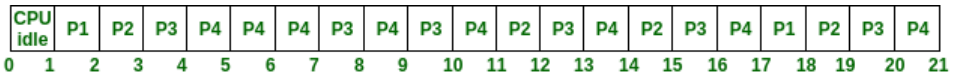
```
Process    Arrival time    Burst Time

P1             1 ms           2 ms

P2             2 ms           4 ms

P3             3 ms           6 ms

P4             4 ms           8 ms
```

Execution:

1. At t = 1, Available Process : P1. So, select P1 and execute 1 ms.
2. At t = 2, Available Process : P1, P2. So, select P2 and execute 1 ms (since BT(P1)=1 which is less than BT(P2) = 4)
3. At t = 3, Available Process : P1, P2, P3. So, select P3 and execute 1 ms (since, BT(P1) = 1 , BT(P2) = 3 , BT(P3) = 6).
4. Repeat the above steps until the execution of all processes.

Note that CPU will be idle for 0 to 1 unit time since there is no process available in the given interval.

Gantt chart will be as following below,



Waiting Time (WT)	Turn Around Time (TAT)	Completion Time (CT)	P.No.	Arrival Time (AT)	Burst Time (BT)
15	17	18	P1	1	2
13	17	19	P2	2	4
11	17	20	P3	3	6
9	17	21	P4	4	8

Output:

```
Total Turn Around Time = 68 ms

So, Average Turn Around Time = 68/4 = 17.00 ms



And, Total Waiting Time = 48 ms

So Average Waiting Time = 48/4 = 12.00 ms
```

- Priority Scheduling

Non-Preemptive Priority scheduling: It is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned a priority. The process with the highest priority is to be executed first and so on.
Processes with the same priority are executed on the FCFS scheme. Priority can be decided based on memory requirements, time requirements or any other resource requirement.
Algorithm:

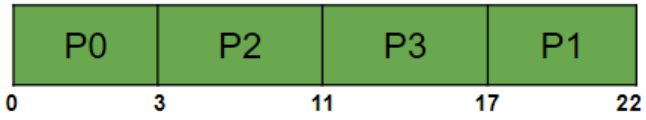
```
1- First input the processes with their burst time
   and priority.
2- Sort the processes, burst time and priority
   according to the priority.
3- Now simply apply FCFS algorithm.
```

Let's look at the following problem with P2 set as the highest priority.

Process	Arrival Time	Priority	Burst Time
P0	0	5	3
P1	1	3	5
P2	2	15	8
P3	3	12	6

At first, P0 enters into scheduling since it arrives first. After its completion, the other 3 processes arrive. Now, priority comes into the picture. The process with the highest priority is scheduled. Here it is P2, followed by P3 and P1.

Now let's look at the Gantt chart:

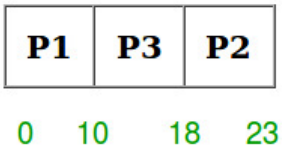


The final chart looks like this:

Process	Arrival Time	Priority	Burst Time	Turn Around Time	Waiting Time
P0	0	5	3	3	0
P1	1	3	5	21	16
P2	2	15	8	9	1
P3	3	12	6	14	8

Example:

Process	Burst Time	Priority
P1	10	2
P2	5	0
P3	8	1



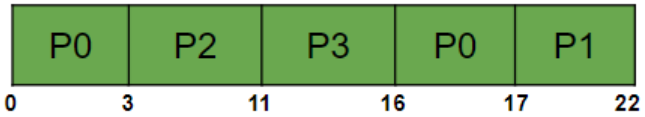
Pre-emptive Priority Scheduling: This follows preemption where one process can be pre-empted and replaced by another process. Let's look at a similar problem as taken above and understand the working of this algorithm:

Process	Arrival Time	Priority	Burst Time
P0	0	5	3
P1	1	3	5
P2	2	15	8
P3	3	12	6

At first, P0 is scheduled and it runs for 1 unit followed by the arrival of P1. Since P1 has less priority than P0, P0 continues for 1 more unit, Now P2 arrives which has a greater priority and hence is scheduled. It runs until completion and then P3 follows which has a greater priority among the present processes. It completes and then the priority of the remaining processes are compared i.e., P0 and P1. P0 having greater priority finishes its remaining 1 unit of time and then P1 follows.

Process	Arrival Time	Priority	Burst Time
P0	0	5	~~3~~ 1
P1	1	3	5
P2	2	15	8
P3	3	12	6

Below is the Gantt chart of the following process:



Here is the final chart showing all the time:

Process	Arrival Time	Priority	Burst Time	Turn Around Time	Waiting Time
P0	0	5	3	17	14
P1	1	3	5	21	16
P2	2	15	8	8	0
P3	3	12	6	13	7

Note: A major problem with priority scheduling is indefinite blocking or **Starvation**. A solution to the problem of indefinite blockage of the low-priority process is ageing. **Aging** is a technique of gradually increasing the priority of processes that wait in the system for a long period of time.

- Round-Robin Scheduling

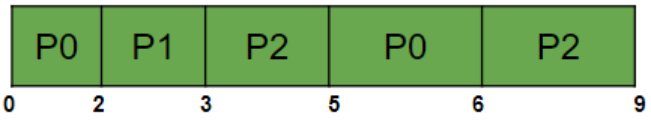
Round-Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot (also called quantum). Once that chunk of time is completed, the process is context-switched with the next in the queue. It is the most common and practically usable scheduling algorithm, as it doesn't require the system to estimate the burst-time of a process. Calculation of burst-time is practically not possible as no one can predict how long a process will take to execute. It although has a minor disadvantage of the overhead of context-switching (some time gets wasted in the process). Let's try to understand the working using the following problem, with given Time Quantum of 2 units:

Process	Arrival Time	Burst Time
P0	0	3
P1	1	1
P2	1	5

Each process runs for 2 unit of time, followed by the next process. The whole process runs in a circular manner.

Process	Arrival Time	Burst Time
P0	0	~~3~~ 1
P1	1	~~4~~ 0
P2	1	~~5~~ 3

Here is the Gantt chart of the above problem:



Here is the final chart showing all type of times:

Process	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
P0	0	3	6	3	0
P1	1	1	2	1	1
P2	1	5	8	3	2

Round Robin Example:

Process	Duration	Order	Arrival Time
P1	3	1	0
P2	4	2	0
P3	3	3	0

Suppose time quantum is 1 unit.

P1	P2	P3	P1	P2	P3	P1	P2	P3	P2

0 10

P1 waiting time : 4

P2 waiting time: 6

P3 waiting time: 6

The average waiting time(AWT) : (4+6+6)/3=5.33

The algorithm to calculate the various time statistics are as follows:

```
1) Create an array rem_bt[] to keep track of remaining
   burst time of processes. This array is initially a
```

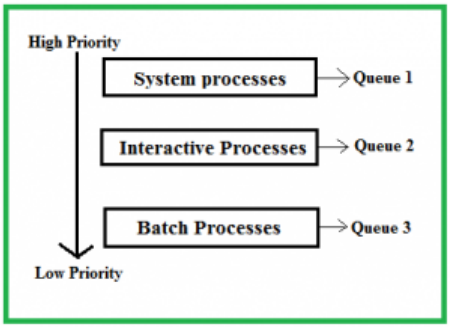
```
   copy of bt[] (burst times array)
2) Create another array wt[] to store waiting times
   of processes. Initialize this array as 0.
3) Initialize time : t = 0
4) Keep traversing all processes while all processes
   are not done. Do following for the ith process if it is not done yet.
    a- If rem_bt[i] > quantum
       (i)  t = t + quantum
       (ii) bt_rem[i] -= quantum;
    c- Else // Last cycle for this process
       (i)  t = t + bt_rem[i];
       (ii) wt[i] = t - bt[i]
       (ii) bt_rem[i] = 0; // This process is over
```

Once we have waiting times, we can compute turn around time tat[i] of a process as sum of waiting and burst times, i.e., wt[i] + bt[i]

- Multilevel Queue Scheduling

Multi-level Queue Scheduling is required when we group processes into some specific categories. e.g. **Foreground (interactive)** and **Background (batch)** processes. These two classes have different scheduling needs. *System* processes are of the highest priority (extremely critical), then comes *Interactive* processes followed by *Batch* and *Student* processes.

Ready Queue is divided into separate queues for each class of processes. For example, let us take three different types of process System processes, Interactive processes, and Batch Processes. All three process has its own queue. Now, look at the below figure.



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

Scheduling among the queues : What will happen if all the queues have some processes? Which process should get the CPU? To determine this Scheduling among the queues is necessary. There are two ways to do so -

1. **Fixed priority preemptive scheduling method -** Each queue has absolute priority over lower priority queue. Let us consider following priority order **queue 1 > queue 2 > queue 3**.According to this algorithm no process in the batch queue(queue 3) can run unless queue 1 and 2 are empty. If any batch process (queue 3) is running and any system (queue 1) or Interactive process(queue 2) entered the ready queue the batch process is preempted.

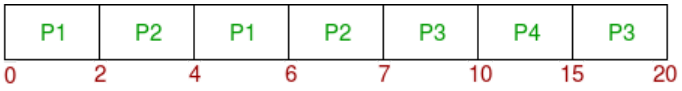
2. **Time slicing** - In this method each queue gets certain portion of CPU time and can use it to schedule its own processes.For instance, queue 1 takes 50 percent of CPU time queue 2 takes 30 percent and queue 3 gets 20 percent of CPU time.

Example Consider the below table of four processes under Multilevel queue scheduling. Queue number denotes the queue of the process.

Process	Arrival Time	CPU Burst Time	Queue Number
P1	0	4	1
P2	0	3	1
P3	0	8	2
P4	10	5	1

Priority of queue 1 is greater than queue 2. queue 1 uses Round Robin (Time Quantum = 2) and queue 2 uses FCFS.

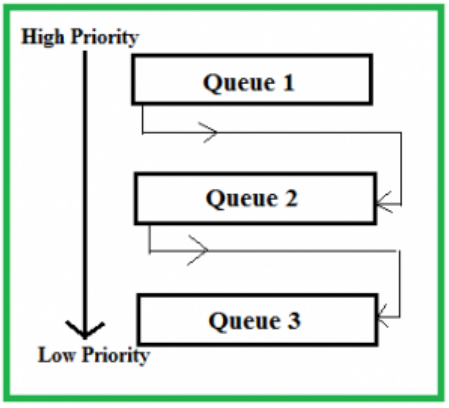
Below is the **gantt chart** of the problem :



At starting both queues have process so process in queue 1 (P1, P2) runs first (because of higher priority) in the round robin fashion and completes after 7 units then process in queue 2 (P3) starts running (as there is no process in queue 1) but while it is running P4 comes in queue 1 and interrupts P3 and start running for 5 second and after its completion P3 takes the CPU and completes its execution.

- Multilevel Feedback Queue Scheduling

MLFQ (Multilevel-Feedback-Queue) scheduling is a modification to the MLQ (Multilevel-Queue) which allows processes to move between queues. It keeps analyzing the behavior (time of execution) of processes, according to which it changes its priority.



Now let us suppose that queues 1 and 2 follow round robin with time quantum 4 and 8 respectively and queue 3 follow FCFS. One implementation of MFQS can be:

1. When a process starts executing then it first enters queue 1.
2. In queue 1 process executes for 4 unit and if it completes in this 4 unit or it gives CPU for I/O operation in this 4 unit than the priority of this process does not change and if it again comes in the ready queue than it again starts its execution in Queue 1.
3. If a process in queue 1 does not complete in 4 unit then its priority gets reduced and it shifted to queue 2.
4. Above points 2 and 3 are also true for queue 2 processes but the time quantum is 8 unit. In a general case if a process does not complete in a time quantum than it is shifted to the lower priority queue.
5. In the last queue, processes are scheduled in FCFS manner.
6. A process in lower priority queue can only execute only when higher priority queues are empty.
7. A process running in the lower priority queue is interrupted by a process arriving in the higher priority queue.

NOTE: A process in the lower priority queue can suffer from starvation due to some short processes taking all the CPU time. A simple solution can be to boost the priority of all the process after regular intervals and place them all in the highest priority queue.

What is the need for such a complex scheduling algorithm?

- Firstly, it is more flexible than the multilevel queue scheduling.
- To optimize turnaround time algorithms like SJF is needed which require the running time of processes to schedule them. But the running time of the process is not known in advance. MFQS runs a process for a time quantum and then it can change its priority(if it is a long process). Thus it learns from past behavior of the process and then predicts its future behavior. This way it tries to run a shorter process first thus optimizing turnaround time.
- MFQS also reduces the response time.