B.C.A. (Sem – IV)

B.C.A. - 404

Operating System

Purushottam Singh

Unit- 2

Unit-2

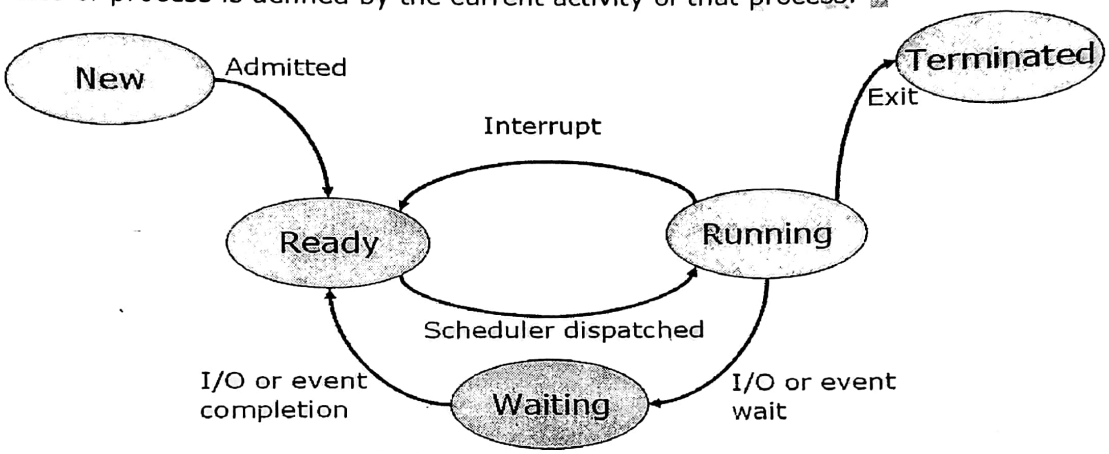
Process: -

Introduction of process: -

- Early computer systems allowed only one program to be executed at a time.
- This program had complete control of the system, and the program had access to all of the system's resources.
- Today's computer systems allow multiple programs to be loaded into memory and to be executed at same time.
- A process is the **unit of work** in modern time-sharing system.
- The jobs, user program, task and spooling is now call **process** in modern operating system.
- A process is a program in execution. The execution of a process must be done in sequential order.
- Process includes the current activity that is known as program counter, and the components of the processor's registers.
- Process is the active entity, with a program counter specifying the next instruction to execute and a set of associated resources.
- Two different processes may be associated with the same program. In this situation they are considered as two separate execution sequence.

Process state transaction diagram: -

- Consider the following process state transaction diagram.
- As a process executes, it change its state.
- The state of process is defined by the current activity of that process.



- Each process may be in one of the following state.
- **New:** The process is being created.
- **Running:** Instructions are being executed.
- **Waiting:** The process is waiting for some event to occur such as I/O completion or waiting for signals.
- **Ready:** The process is waiting to be assigned to a processor.
- **Terminated:** The process has finished his execution.
- These states are found in all systems.
- It is important to realize that only one process can be running on any processor at any instance.
- Many processes may be ready and waiting for execution.

Process control block (PCB): -

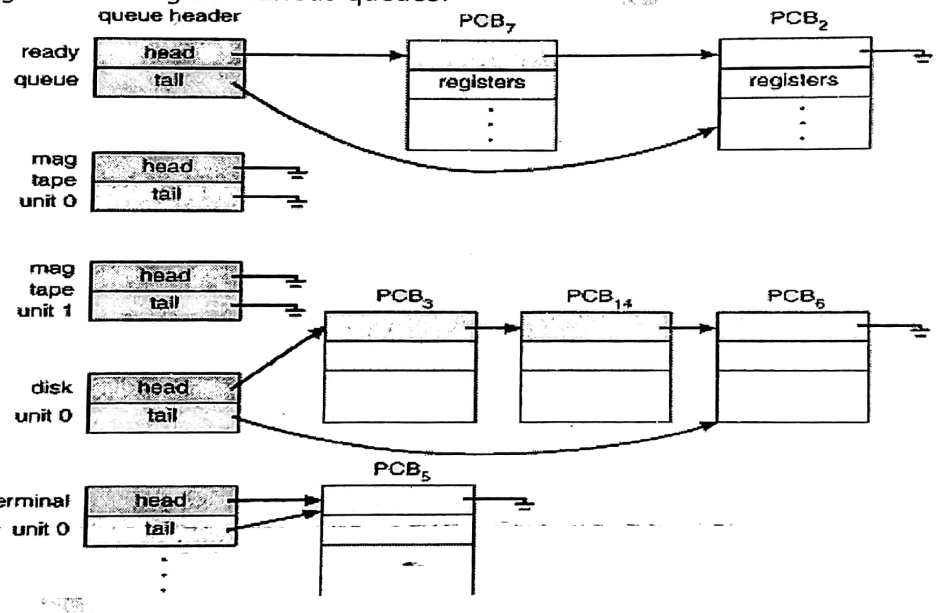
- Each process in the operating system represented by **process control block (PCB)**.
- Process control block is also known as **task control block**.
- **Pointer:** Pointer contains the physical address of process.
- **Process state:** The state may be new, ready, running, waiting or terminated.
- **Process number:** Sequence number of the process.
- **Program counter:** The counter indicates the address of the next instruction to be executed for this process.
- **CPU registers:** There are several Commands of registers number and Commands. That is depending on the computer architecture. They include index registers, stack pointer, etc... Along with the program counter, this state information must be saved when interrupt occurs, to allow the process to be executed after some time.
- **CPU scheduling information:** This information includes a process priority.
- **Memory management information:** This information includes the value of base registers, limit registers, etc...

- **Accounting information:** This information includes the amount of time limits, job or process number, etc...
- **I/O status information:** This information includes the list of I/O devices which is allocated to the process, list of open files and so on.

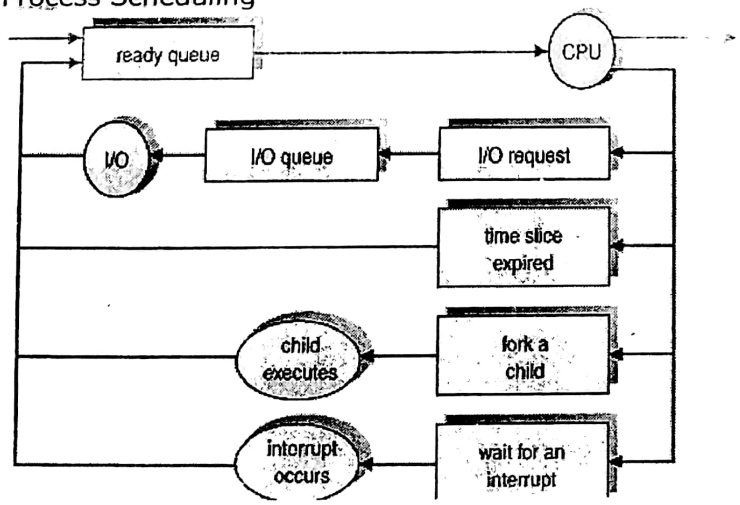
Pointer	Process state
Process number	
Program counter	
Registers	
Memory limits	
List of open files	

Process scheduling queue: -

- Job queue: Set of all processes in the system.
- Ready queue: Set of all processes residing in main memory, ready and waiting to execute.
- Device queues: Set of processes waiting for an I/O device.
- Process migrates among the various queues.



- Representation of Process Scheduling



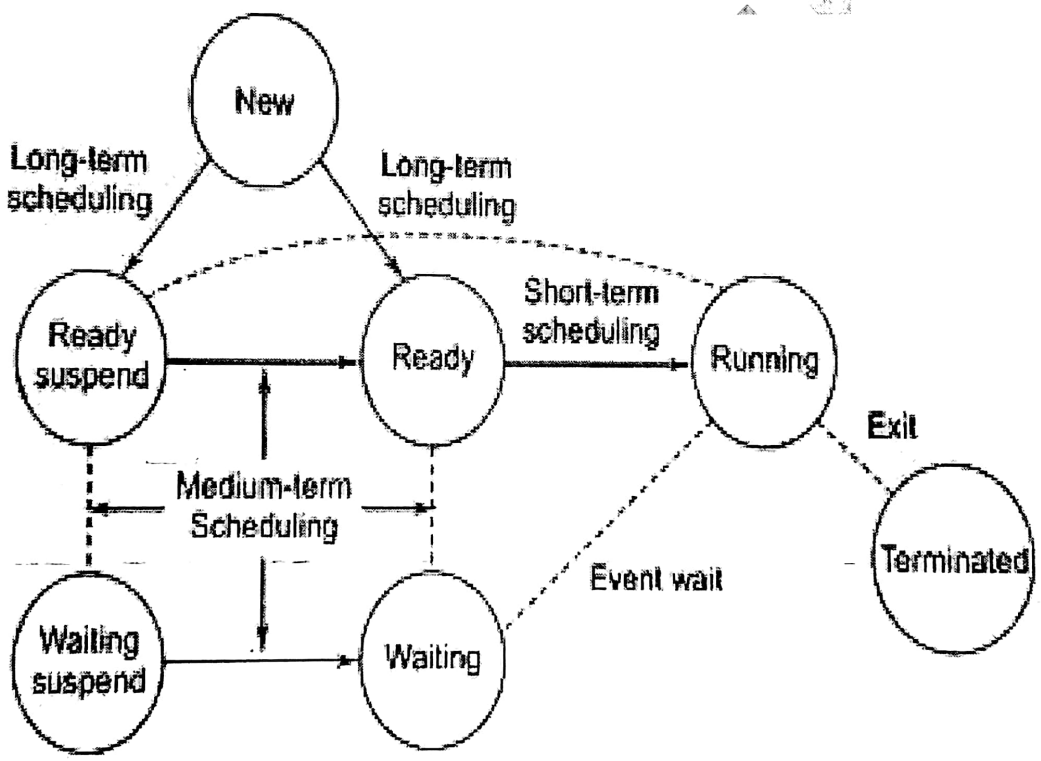
- Multiple processes are ready to execute, among them one process is selected and load into memory it is provide to CPU for execution.

- CPU scheduling decision may take place when a process:
  1. Switches from running to wait state.
  2. Switches from running to ready state.
  3. Switches from waiting to ready.
  4. Terminates.

Schedulers: -

Long term scheduler (Job scheduler) (Admission scheduler): -

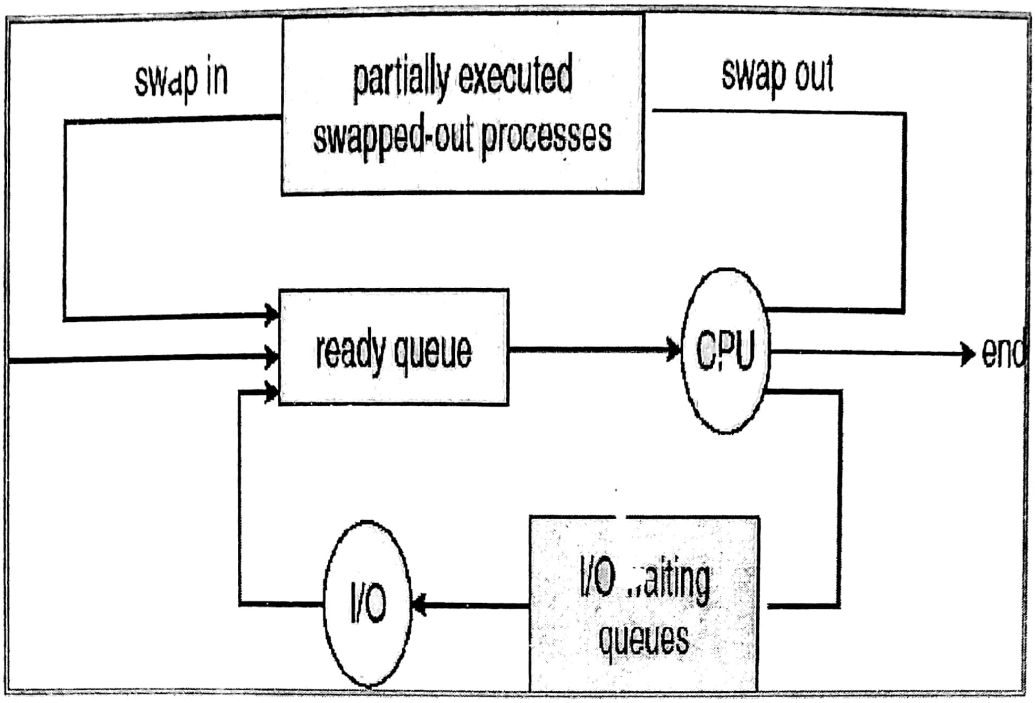
- The long term scheduler affects the processes which are in **new-state and exit-state.**
- The long-term decides which processes are to be admitted to the ready queue (in the Main Memory).
- Long term scheduling obviously controls the degree of multiprogramming in multitasking systems.
- Certain policies to decide, if more than one job is submitted, which of them should be selected.
- Once a new process is accepted (and partially created) it may enter the scheduling queues in one of two places:
  1. If all resources (such as memory requirements) are initially fully available to the new process, it may be admitted to the tail of the Ready queue.
  2. If not all resources are immediately available, the new process may be instantiate l in the Blocked-Suspended queue until those resources are provided.
- The long-term scheduler is executed relatively infrequently.
- Alternatively, when the CPU becomes sufficiently idle, the long-term scheduler may be permitted to introduce a new process.



Medium term scheduler: -

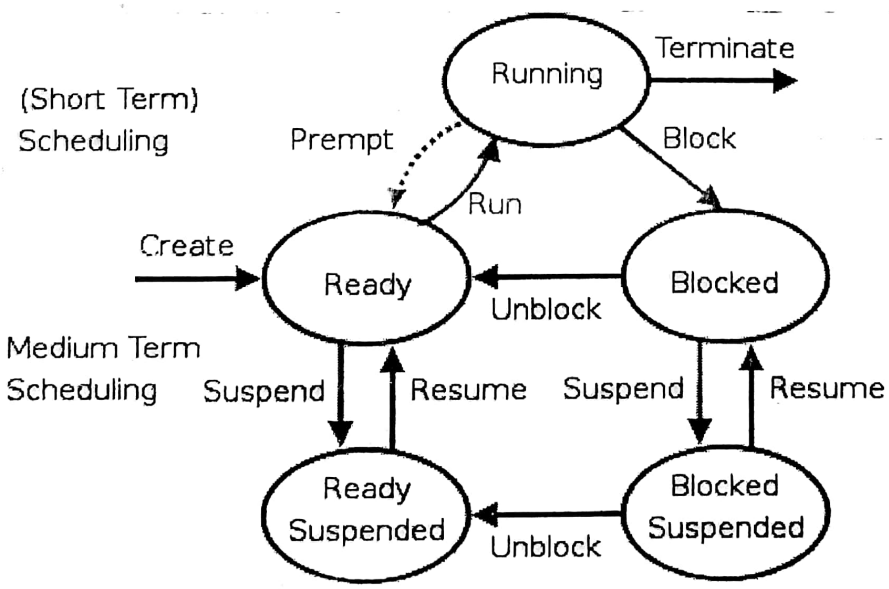
- The medium term scheduler affects the processes which are in **ready-suspended and blocked-suspended.**
- The medium-term scheduler temporarily removes processes from main memory and places them on secondary memory (such as a disk drive) or vice versa.
- This is commonly referred to as "swapping out" or "swapping in" (also incorrectly as "paging out" or "paging in").
- The medium-term scheduler may decide to swap out a process which has one the following condition:
  1. Process which has not been active for some time.
  2. Process which has a low priority.
  3. Process which page is faulting frequently.
  4. Process which is taking up a large amount of memory in order to free up main memory for other processes.
  5. Process has been unblocked and is no longer waiting for a resource.

- In many systems today the medium-term scheduler may actually perform the role of the long-term scheduler, by treating binaries as "swapped out processes" upon their execution.



Short term scheduler (CPU scheduler): -

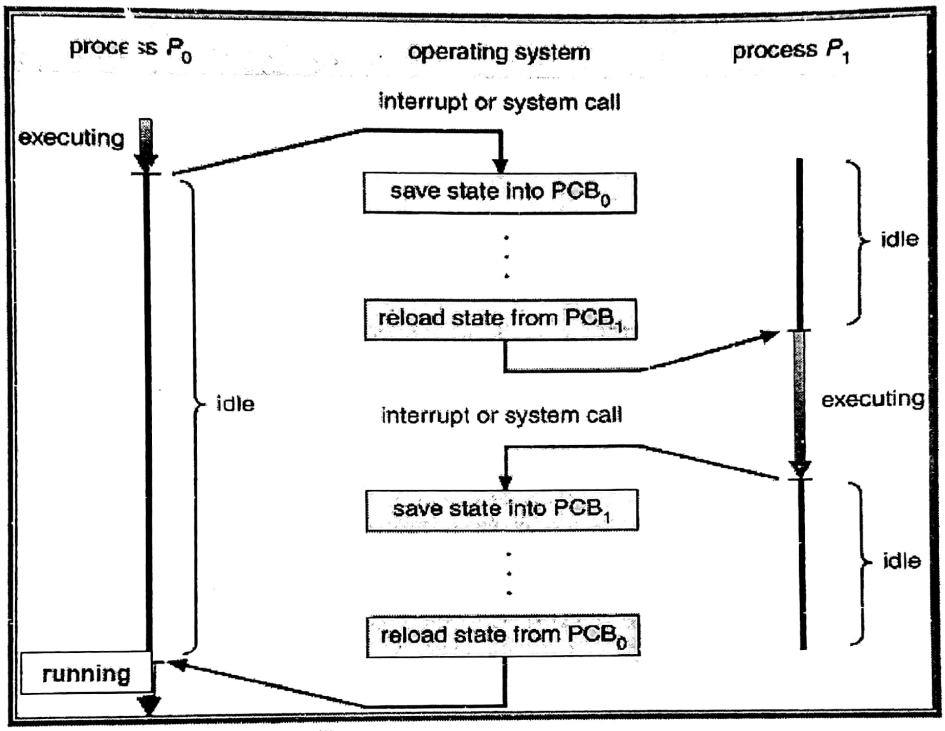
- The short term scheduler selects which process should be executed next and allocates to the CPU.
- The short-term scheduler is invoked whenever an event occurs which provides the opportunity, or requires, the interruption of the current process and the new (or continued) execution of another process.
- Thus the short-term scheduler makes scheduling decisions much more frequently than the long-term or mid-term schedulers.
- A scheduling decision will at a minimum have to be made after every time slice and these are very short.
- This scheduler can be preemptive, implying that it is capable of forcefully removing processes from a CPU when it decides to allocate that CPU to another process.
- The non-preemptive (also known as "voluntary" or "co-operative"), in which case the scheduler is unable to "force" processes off the CPU.
- In most cases short-term scheduler is written in assembly because it is a critical part of the operating system.



Unblock is done by another task (a.k.a. wakeup, release, V)
Block is a.k.a. sleep, request, P)

Context switch: -

- A context switch (also sometimes referred to as a process switch or a task switch) is the switching of the CPU from one process to another.
- A process (also sometimes referred to as a task) is an executing (i.e., running) instance of a program.
- The context switching is performed by scheduler (Dispatcher).
- Context switching performed following task.
  1. Save PCB state of the old process.
  2. Load PCB state of the new process.
  3. Flush memory cache.
  4. Change memory mapping (TLB).
- Context switching time is pure overhead because the system is not doing much useful work while switching.
- The context switching time is very expensive while switching the process. It will take 1 to 1000 microseconds.
- Context switching times are highly dependent on hardware support.



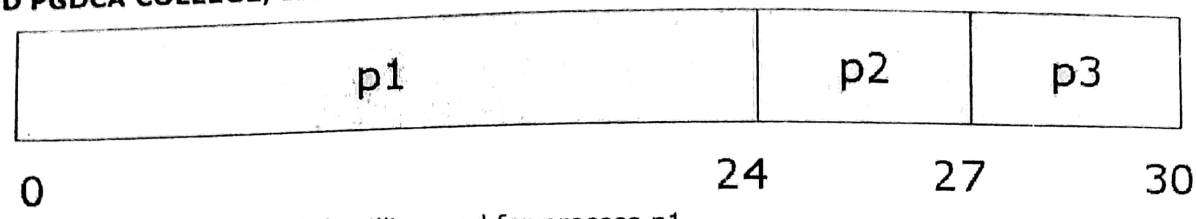
Process scheduling algorithms: -

First Come First Served (FCFS) Algorithm: -

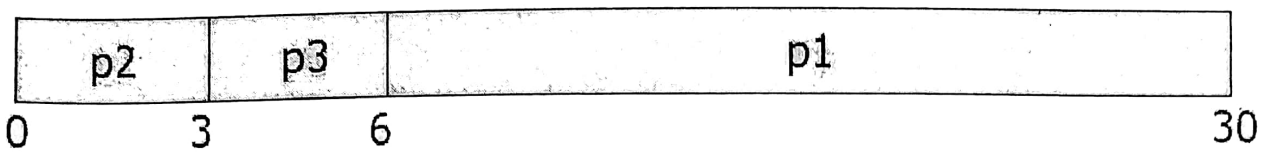
- The simplest CPU scheduling algorithm is the first-come, first-served (FCFS) scheduling algorithm.
- The FCFS scheduling is non-preemptive.
- With this scheme, the process that requests first for the CPU that process is allocated to the CPU first.
- The implementation FCFS policy is managed by using FIFO condition.
- When the process is entered in to ready queue, its PCB is linked onto tail side of queue.
- When the CPU is free, it is allocated to the process at the head of the queue.
- After execution of that process it is removed from queue.
- The average waiting time under the FCFS policy is quite long.
- Consider the following set of process that arrives at time 0, with the length of the CPU-burst time given in milliseconds.

Process	Burst Time
P1	24
P2	03
P3	03

- If the process arrives in the order p1, p2, p3 and are served in FCFS order, we get the result shown in the following.



- The waiting time is zero (0) millisecond for process p1.
- The waiting time is zero (24) millisecond for process p2.
- The waiting time is zero (27) millisecond for process p3.
- Thus, the average waiting time is (0 + 24 + 27) / 3 = 17 milliseconds.
- If the process arrives in the order p2, p3, p1 the result will be as shown in the following.



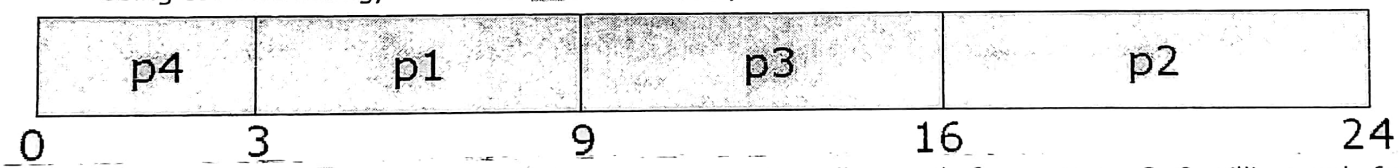
- The average waiting time is now (6 + 0 + 3) / 3 = 3 milliseconds.
- Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU either by terminating or by requesting I/O.

Non preemptive Shortest Job First (SJF) scheduling algorithm: -

- A different approach to CPU scheduling is the shortest-job-first (SJF) algorithm.
- This algorithm is associates with each process.
- When the CPU is available, it is assigned to the process that has smallest CPU burst time.
- If two processes have same burst time, then FCFS scheduling is used to break the tie.
- SJF scheduling is done by examining by the length of the next CPU burst time of the process, rather then its total length.
- Consider the following example.

Process	Burst Time
P1	06
P2	08
P3	07
P4	03

- Using SJF scheduling, we would schedule these processes according to the following.



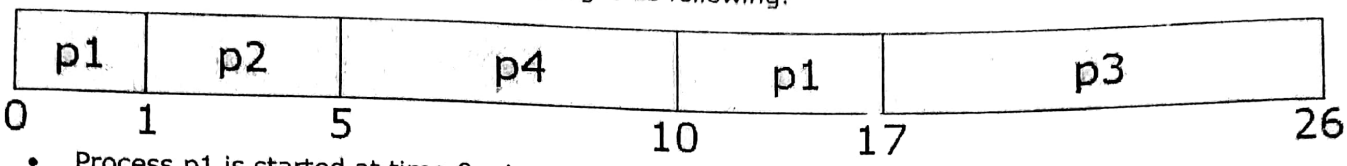
- The waiting time is 3 milliseconds for process p1, 16 milliseconds for process p2, 9 milliseconds for process p3, and 0 milliseconds for process p4.
- Thus, the average waiting time is (3 + 16 + 9 + 0) / 4 = 7 milliseconds.
- If we are using FCFS scheduling scheme, then the average waiting time is 10.25 milliseconds.
- SJF scheduling is used frequently in long-term scheduling.
- It can not be implemented at the level of short-term CPU scheduling.

Preemptive Shortest Job First (SJF) scheduling algorithm: -

- The SJF algorithm may be either preemptive or non preemptive.
- A preemptive SJF algorithm will preempt the currently executing process, whereas non preemptive SJF algorithm will allow the currently running process to finish CPU burst.
- Preemptive SJF scheduling algorithm is some times called **shortest-remaining-time-first** scheduling.
- Consider the following example.

Process	Arrival Time	Burst Time
P1	00	08
P2	01	04
P3	02	09
P4	03	05

- The result of preemptive SJF scheduling is as following.



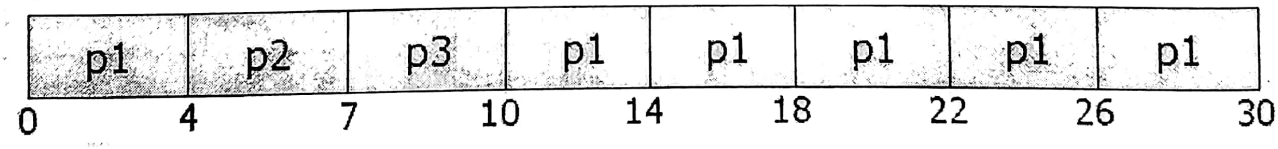
- Process p1 is started at time 0, since it is only one process in a queue.
- Process p2 arrives at time 1. Thus remaining time of process p1 (7 milliseconds) is larger then the time required by process p2 (4 milliseconds), so process p1 is preempted and process p2 is scheduled.
- The average waiting time of this example is ((10-1) + (1-1) + (17-2) + (5-3)) / 4 = **26/4** = 6.5 milliseconds.
- In non preemptive SJF scheduling the average waiting time of this example would be 7.75 milliseconds.

Round-Robin (RR) scheduling algorithm: -

- The Round-Robin (RR) scheduling algorithm is designed especially for time-sharing systems.
- It is similar to the FCFS scheduling, but preemption is added to switch between processes.
- A small unit of time is known as *quantum* or *time-slice*.
- A time quantum is generally from 10 to 100 milliseconds.
- New processes are added to the tailing side of ready queue.
- The CPU scheduler picks the first process from ready queue, sets a timer to create interrupt after 1 time quantum and dispatches the process.
- The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU. The scheduler then goes to the next process in ready queue.
- If the CPU burst of currently running process is longer than 1 time quantum, the timer will go off and generate interrupt to the operating system. A context switch will be executed, and the process will be put at tailing side of ready queue.
- Average waiting time of RR scheduling policy is quite long.
- The performance of RR scheduling algorithm depends on the size of time quantum.
- If time quantum is very large, the RR policy is become as same as FCFS police.
- If time quantum is very small (1 millisecond), the RR policy is become processer sharing policy.
- Consider the following set of processes that arrive at time 0, with length of CPU burst given in milliseconds.

Process	Burst Time
P1	24
P2	03
P3	03

- If we use a time quantum of 4 milliseconds, then process p1 gets first 4 milliseconds.
- Since process p1 required another 20 milliseconds, but it is preempted after first time quantum.
- The CPU given to the next process p2. Here process p2 does not require 4 milliseconds.
- Process p2 is completed before time.
- The CPU given to the next process p3. Here process p3 does not require 4 milliseconds.
- Process p3 is completed before time.
- Now CPU is returned to process p1 with additional 2 time quantum.
- The result of RR scheduling is as under.



- The average waiting time is [(30-24)+4+7]/3 = 17/3 =5.66 milliseconds.
- In RR scheduling algorithm, no process is allocated to the CPU for more than 1 time quantum.
- If a process' CPU burst exceeds 1 time quantum, that process is preempted and put back in ready queue.
- The RR scheduling algorithm is preemptive.

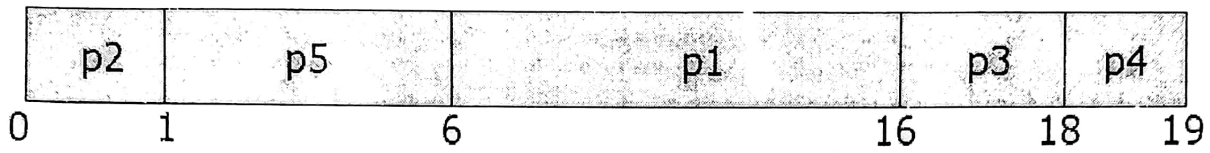
Priority based scheduling / Event driven (ED) scheduling algorithm: -

- The priority scheduling is a special case of general priority scheduling algorithm.
- A priority is associated with each process, and the CPU is allocated to the process with highest priority.

- If two processes have equal priority then they are scheduled in FCFS order.
- Note that we are discussing the scheduling in term of *high* and *low* priority.
- Priorities are consist of some fixed range of numbers, such as 0 – 9.
- There is no general agreement on whether 0 is the highest or lowest priority.
- Some systems use low numbers to represent low priority; others use low numbers for high priority.
- In this, we assume that low numbers represent high priority.
- As an example, consider the following set of processes, which is arrived at time 0, in the order p1, p2, p3, p4, p5 with the length of CPU burst time given in milliseconds.

Process	Burst Time	Priority
P1	10	03
P2	01	01
P3	02	03
P4	01	04
P5	05	02

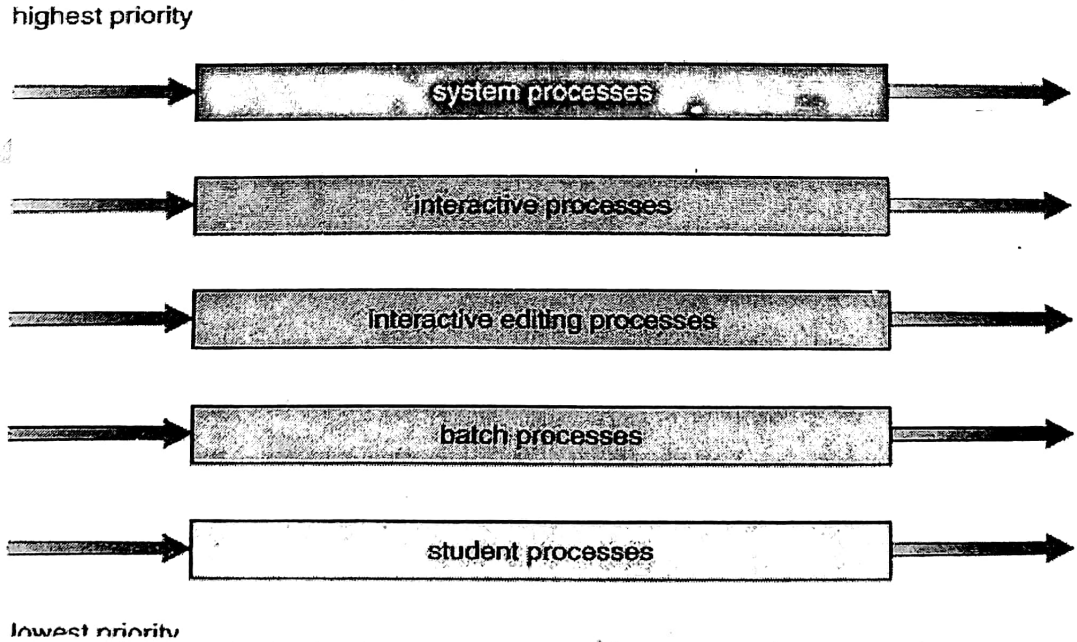
- Using priority scheduling, we would schedule these processes according to the following.



- Thus, the average waiting time is (6 + 0 + 16 + 28 + 1) / 5 = 8.2 milliseconds.
- Priority can be defined either internally or externally.
- Internally defined priorities use some measurable quantity to compute the priority of process.
- External priorities are set by operating system by using certain criteria like importance of process, Command of process etc...
- Priority scheduling can be either preemptive or non-preemptive.
- **Preemptive priority scheduling: -** When a process arrives at ready queue, its priority is compared with currently running process. A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of currently running process.
- **Non-preemptive priority scheduling: -** When a process arrives at ready queue, its priority is compared with currently running process. A non-preemptive priority scheduling algorithm simply put the new process at the head of ready queue.

Multi-Level queue (MLQ): -

- When processes can be categorized, then multiple separate queues can be established.
- Ready queue is partitioned into separate queues: Example, a queue for foreground (interactive) and another for background (batch) processes.



- Processes can be classified into different groups and permanently assigned to one queue.
- Each queue may have has its own scheduling algorithm like Round Robin, FCFS, and SJF etc...

- In addition, scheduling must be done between the queues.
- Fixed priority scheduling (i.e. serve first the queue with highest priority).
- Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; for example, 50% of CPU time is used by the highest priority queue, 20% of CPU time to the second queue, and so on...
- Also, need to specify which queue a process will be put to when it arrives to the system and/or when it starts a new CPU burst.

Measuring performance of computer system: -

- **CPU utilization:** We want to keep the CPU as busy as possible. The range of CPU utilization may from 0% to 100 %. For a lightly used system the range should be 40%. For a heavily used system the range should be 90%.
- **Throughput:** The number of processes that are completed in particular time period is known as throughput. For the long process this rate may be one process per hour or for short processes may be 10 processes per seconds.
- **Turnaround time:** The time duration from the time of submission of a process to the time of completion is known as turnaround time.
- **Waiting time:** Waiting time is the sum of the periods spent waiting in ready queue.
- **Response time:** The time from submission of a request unit the first response is produced. This is known as response time. Response time is the amount of time it takes to start responding.
- **CPU burst:** A process can use the CPU several times before complete the job. But the CPU burst time is an amount of time that a process uses the CPU for a single time.