Parul® University
Vadodara, Gujarat

NAAC GRADE A++

Information and Communication Technology

Processes, Thread & Process Scheduling

Study Guide

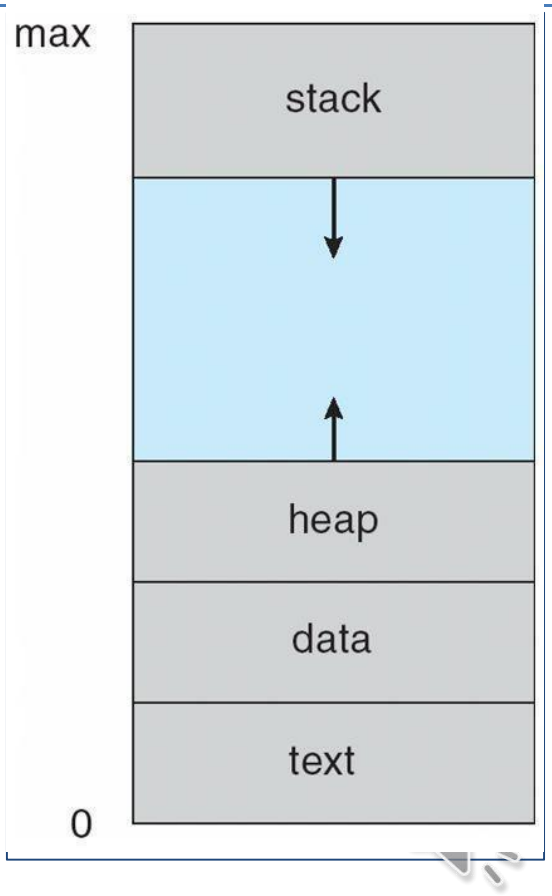
Assistant Prof. Sumersing Patil
CSE-AI&DS, PIET
Parul University

PROCESS

- A process is a program in execution.
- 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.
- A process is an 'active' entity as opposed to program which is considered to be a 'passive' entity. Attributes held by process include hardware state, memory, CPU etc.

What does a process look like in memory?

- Programs are executable(.exe) files generally stored in Secondary Memory.
- In order to execute program it must be loaded in main memory.
- When a program is loaded into the memory and it becomes process.
- Then process memory can be divided into four sections — stack, heap, text and data.



- The **Text section** is madeup of the compiled program code , read in from non-volatile storage when the program is launched.
- The **Data section** : made up the global and static variables, allocated and initialized prior to executing the main.
- The **Heap Section** : Dynamically allocated memory to process during its run time. , and is managed via calls to new, delete, malloc, free, etc.
- The **Stack** : contains temporary data, such as function parameters, returns addresses, and local variables.

@ Process Vs. Program

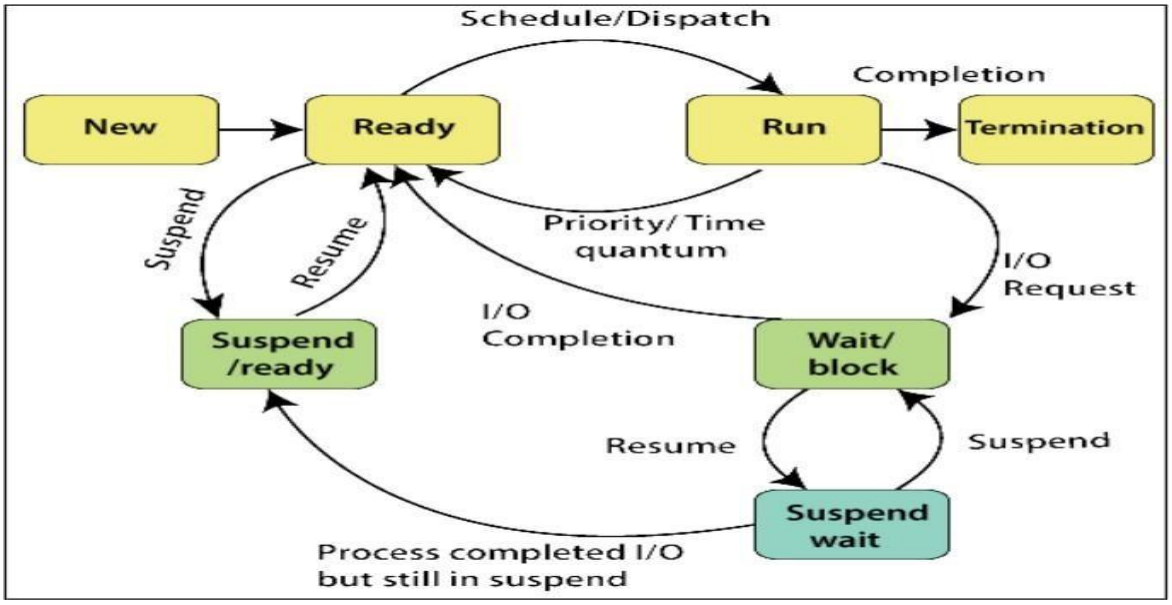
Process	Program
A process is program in execution.	A program is set of instructions.
A process is an active/ dynamic entity.	A program is a passive/ static entity.
A process has a limited life span. It is created when execution starts and terminated as execution is finished.	A program has a longer life span. It is stored on disk forever.
A process contains various resources like memory address, disk, printer etc... as per requirements.	A program is stored on disk in some file. It does not contain any other resource.
A process contains memory address which is called address space.	A program requires memory space on disk to store all instructions.

@ Operations on the Process

- **Creation:** If the process is created, then it will be ready to enter the ready queue and ready to run.
- **Scheduling :** There are so many processes which are present in the ready queue, the operating system selects one process from various processes and start to execute the process. The selection of process in a sequence for execution is called scheduling.
- **Execution :** After the scheduling is done on the process then the process starts executing. Sometimes there may be a situation that arises when the process enters into a blocked or wait state during the execution, then in such a case, the processor starts to execute another process.
- **Deletion/Killing :** When the objective of the process completes, the operating system kills the process. The process control block of the process is also deleted, and the operating system

terminates the process.

@ Process State



Process State & Description

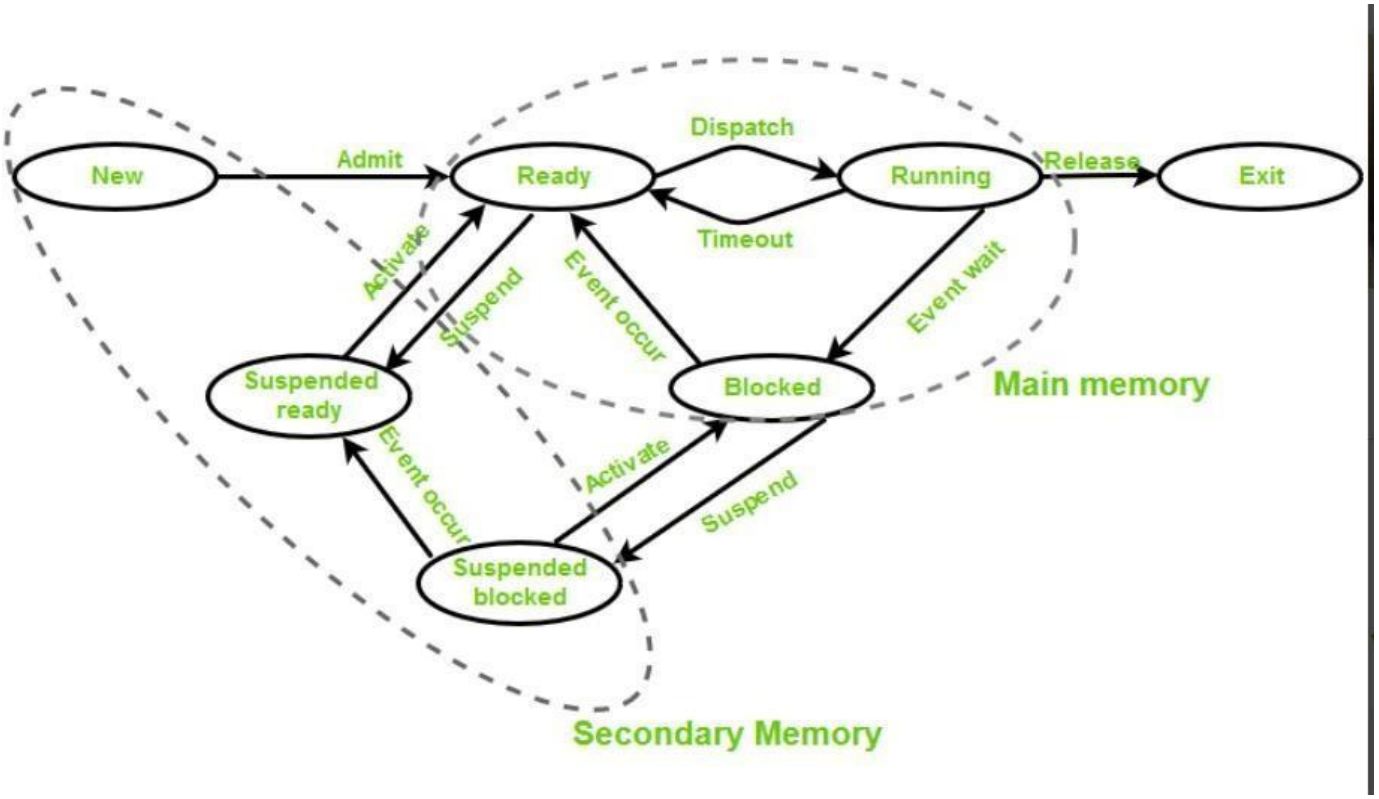
1. **New:** – The state in which a process is created. The New state is the newly created program which is stored in the secondary storage, and taken by the operating system at the time of process creation.
2. **Ready:** – After a process has been created, it enters into the ready state means the process is loaded into the memory. In this, the process is set to run and wait for its execution time to get the CPU. Processes that are ready for CPU execution are placed in a queue for a ready process.
3. **Run:** – In a run state, the CPU selects the process for execution and executes the instructions within the process.
4. **Blocked or Wait:** – If the process is in run state and the process needs some resources for execution, but the resource is held by some other process than the process enters into the blocked or waiting state.
5. **Terminated or Completed:** – If the execution of the process is completed, then the process enters into the terminated or completed state.

6. **Suspend ready**: – Sometimes, due to the minimum number of resources, some process which is in ready state transfer or moves to secondary storage from the primary storage, and this type of process that move in

the ready state are called suspend ready.

7. **Suspend wait or suspend blocked:** – Suspend-wait is like suspend blocked and uses the process which is performing input/output operation and due to minimum amount of main memory move them into secondary storage. It may go to suspend ready when work is finished.

@ Process State Transitions



Ready --> Running

When it is time, the dispatcher selects a new process to run

Running --> Ready

The running process has expired his time slot the running process gets interrupted because a higher priority process is in the ready state.

Running → wait/ Blocked

When a process requests something for which it must wait. a service that

the OS is not ready to perform an access to a resource not yet available initiates I/O and must wait for the result waiting for a process to provide input (IPC).

Wait/ Blocked --> Ready

When the event for which it was waiting occurs

Blocked --> Blocked_Suspend :

When none of the processes occupying the main memory is in a Ready state, OS swaps one of the blocked processes out onto to the Suspend queue.

Blocked_Suspend --> Ready_Suspend :

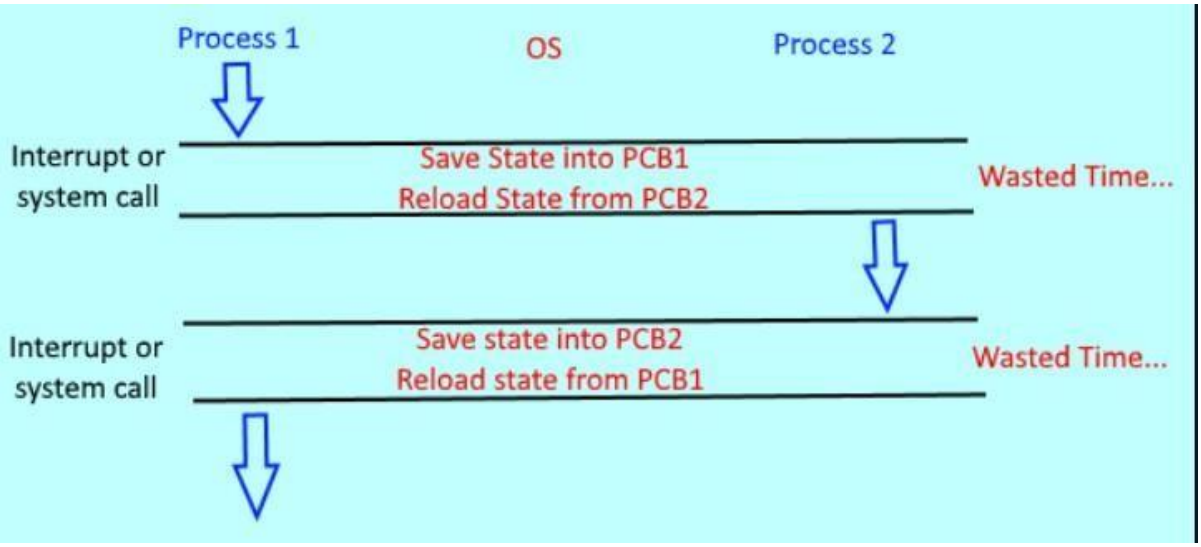
When a Suspended process is ready to run it moves into "Ready, Suspend" queue.

@ Process Control Block

A Process Control Block or simple PCB is a data structure that is used to store the information of a process that might be needed to manage the scheduling of a particular process.

1. **Process ID :** A process id is a unique identity of a process. Each process is identified with the help of the process id.
2. **Process State** : The current state of the process i.e., whether it is ready, running, waiting, or whatever.
3. **Pointer :** It is a stack pointer which is required to be saved when the process is switched from one state to another to retain the current position of the process.
4. **Process privileges** : This is required to allow/disallow access to system resources.
5. **Priority:** There is a priority associated with each process. Based on that priority the CPU finds which process is to be executed first. Higher priority process will be executed first.
6. **Program Counter:** The program counter, points to the next instruction that is to be executed by the CPU. It is used to find the next instruction that is to be executed.
7. **CPU registers** : Various CPU registers where process need to be stored for execution for running state.
8. **CPU Scheduling Information** : Process priority and other scheduling information which is required to schedule the process.
9. **Accounting information** :This includes the amount of CPU used for process execution, time limits, execution ID etc.
10. **IO status information** : list of I/O devices allocated to the process.

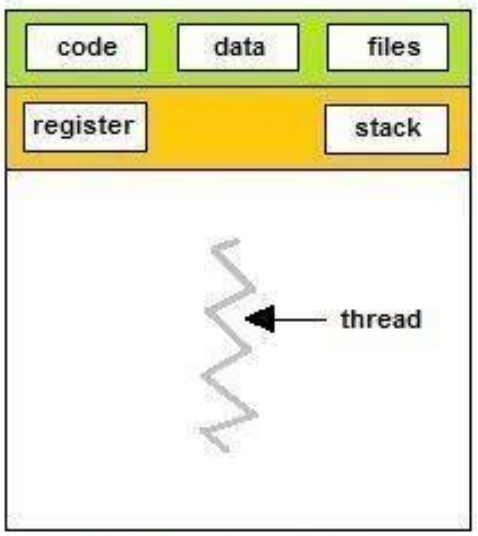
@ Context Switching



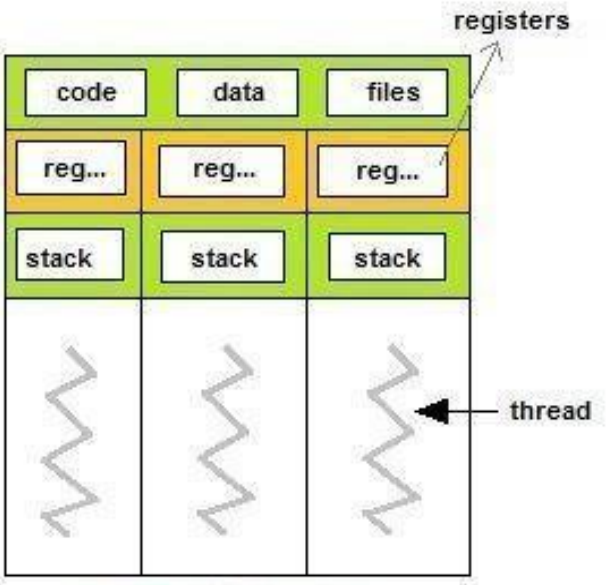
- Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a Context Switch.
- A context switch is a procedure that a computer's CPU (central processing unit) follows to **change from one task (or process) to another** while ensuring that the tasks do not conflict.
- Effective context switching is critical if a computer is to provide user-friendly multitasking.
- A context switch is the mechanism to **store and restore the state** or context of a CPU in Process Control block so that a **process execution can be resumed** from the same point at a later time.
- Using this technique, a context switcher enables **multiple processes to share a single CPU.**

@ Thread

- A thread is a single sequence stream within a process.
- They are sometimes called lightweight processes.
- In a process, threads allow multiple executions of streams.
- Thread is an execution unit which consists of its own program counter, a stack, and a set of registers.
- As each thread has its own independent resource for process execution, multiple processes can be executed parallel by increasing number of threads.
- Each thread belongs to exactly one process and no thread can exist outside a process.



single-threaded process



multithreaded process

Advantages of Thread

- Threads minimize the context switching time.

- Use of threads provides concurrency within a process.

- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

@ Process Vs. Thread

Process	Thread
Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

@ Types of Thread

1) User Level Thread:

- In this case, the thread management **kernel is not aware** of the existence of threads.
- The **thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts.**

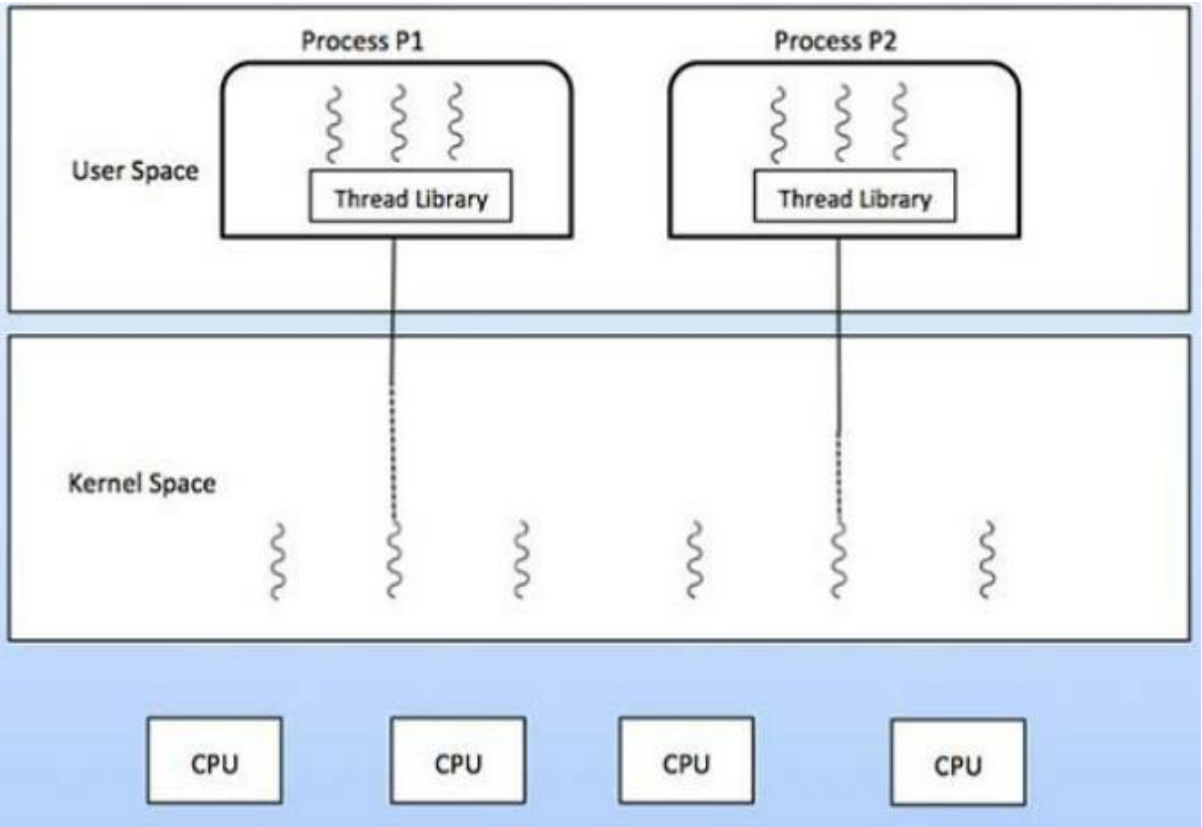
- The application starts with a single thread.

Advantag

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage

Disadvantages

Multithreaded application cannot take advantage of multiprocessing



2) Kernel Level Thread

- In this case, thread **management** is done by the **Kernel**.
- There is no thread management code in the application area.
- Kernel threads are **supported directly by the operating system**.
- The **Kernel performs thread creation, scheduling and management in Kernel space.**

Advantag

- Kernel can simultaneously schedule multiple threads from the sam e process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.

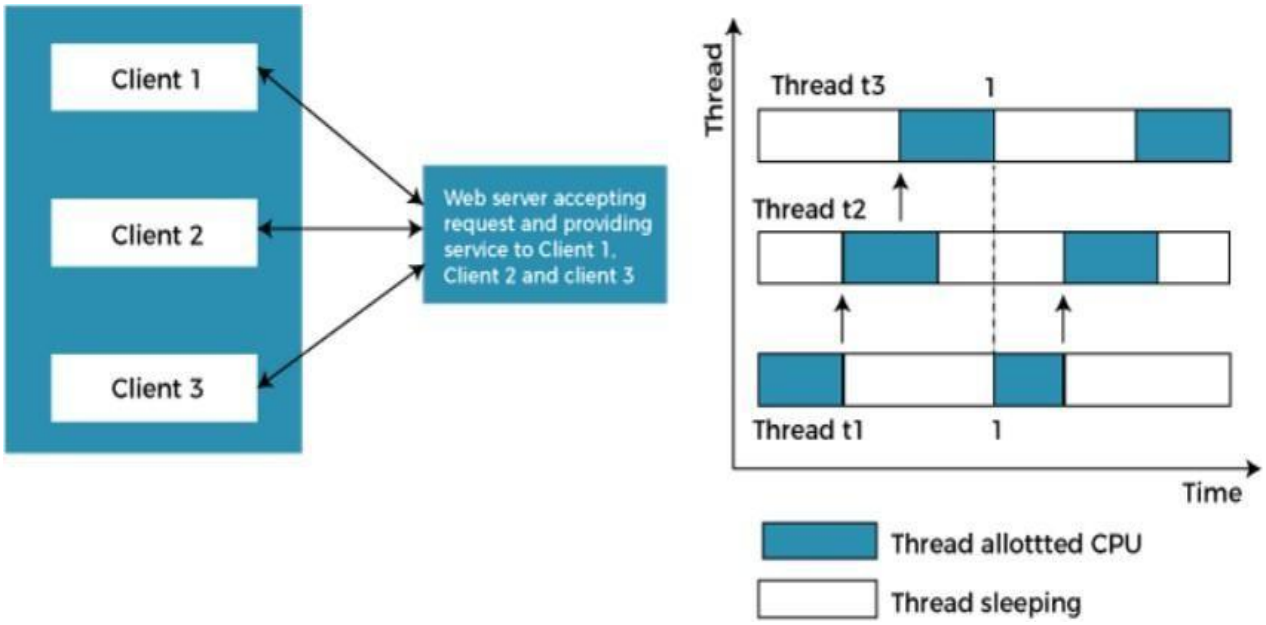
Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

User-Level Threads	Kernel-Level Thread
User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

@ Multithreading

- Multithreading allows the application to divide its task into individual threads. In multi-threads, the same process or task can be done by the number of threads.



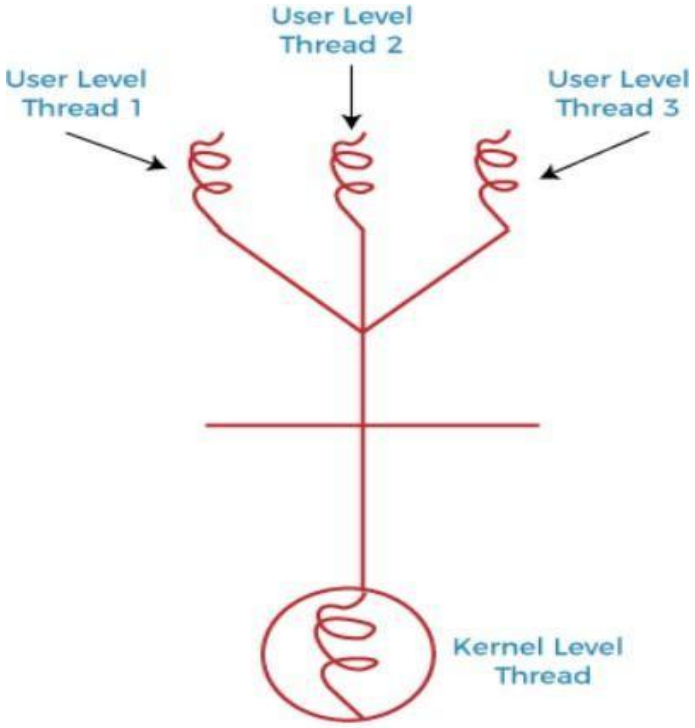
- In the above example, client1, client2, and client3 are accessing the web server without any waiting. In multithreading, several tasks can run at the same time.
- **In an operating system**, threads are divided into the user-level thread and the Kernel-level thread.

Multithreading Models

1) Many to One Model
2) One to One Model
3) Many to Many Model

Many to One Model

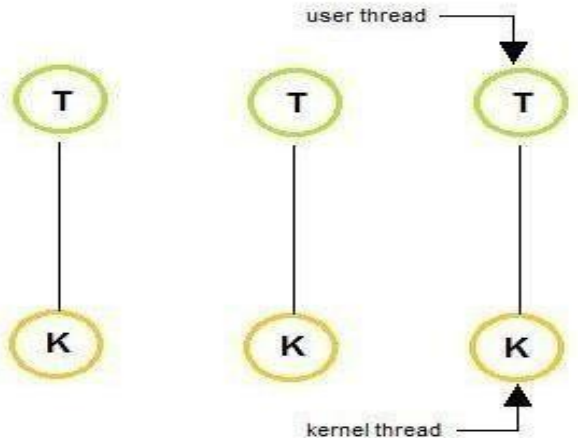
- The many to one model maps many user levels threads to one kernel thread. This type of relationship facilitates an effective context-switching environment.



Many-to-one model

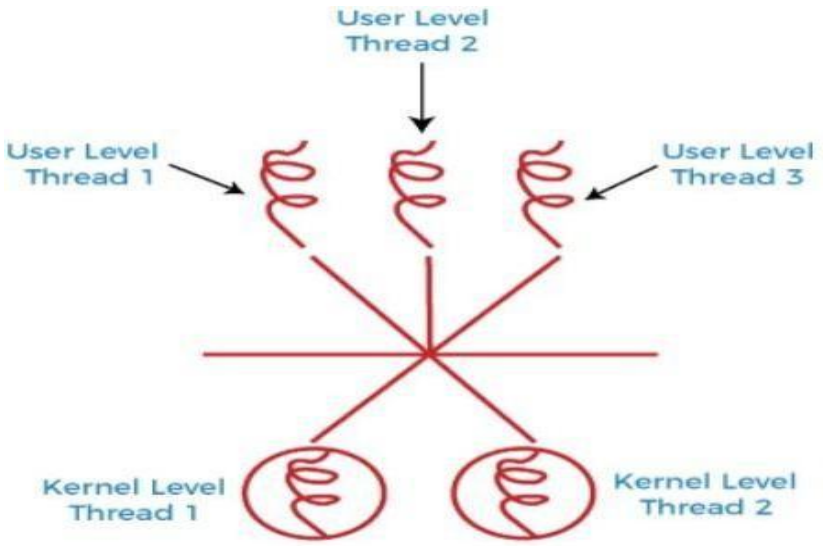
One to one multithreading model

- The one to one model creates a separate kernel thread to handle each and every user thread.
- Most implementations of this model place a limit on how many threads can be created.
- Linux and Windows from 95 to XP implement the one-to-one model for threads.



Many to Many Model multithreading model

- The many to many model multiplexes any number of user threads onto an equal or smaller number of kernel threads, combining the best features of the one-to-one and many-to-one models.
- Users can create any number of the threads.
- Processes can be split across multiple processors



Many-to-Many model

Benefits of Multithreading

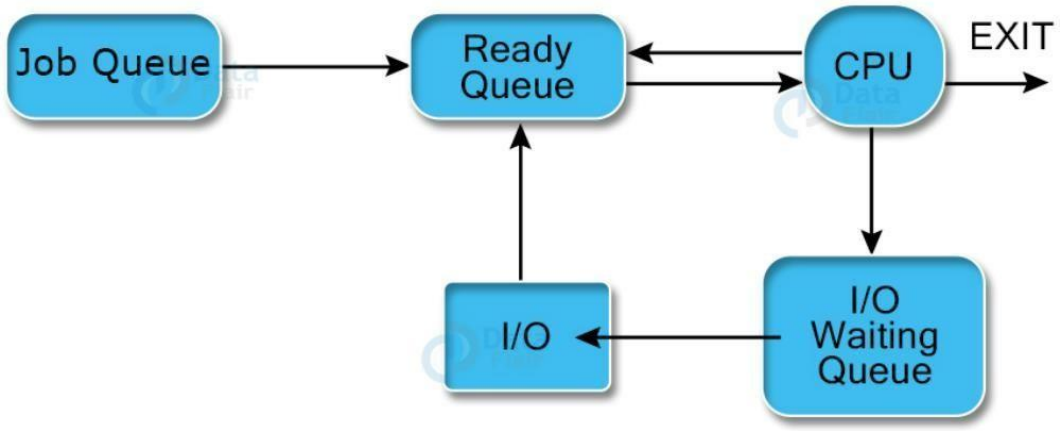
- Responsiveness
- Resource sharing, hence allowing better utilization of resources.
- Economy: Creating and managing threads becomes easier.
- Scalability: One thread runs on one CPU. In Multithreaded processes, threads can be distributed over a series of processors to scale.
- Context Switching is smooth. Context switching refers to the procedure followed by CPU to change from one task to another.

@ Process Scheduling

- The act of determining which process is in the ready state, and should be moved to the running state is known as Process Scheduling".
- Process scheduling is the activity of the process manager that handles suspension of another process on the basis of a particular strategy.
- The part of os that makes the choice is called scheduler.
- The algorithm used by this scheduler is called scheduling algorithm.

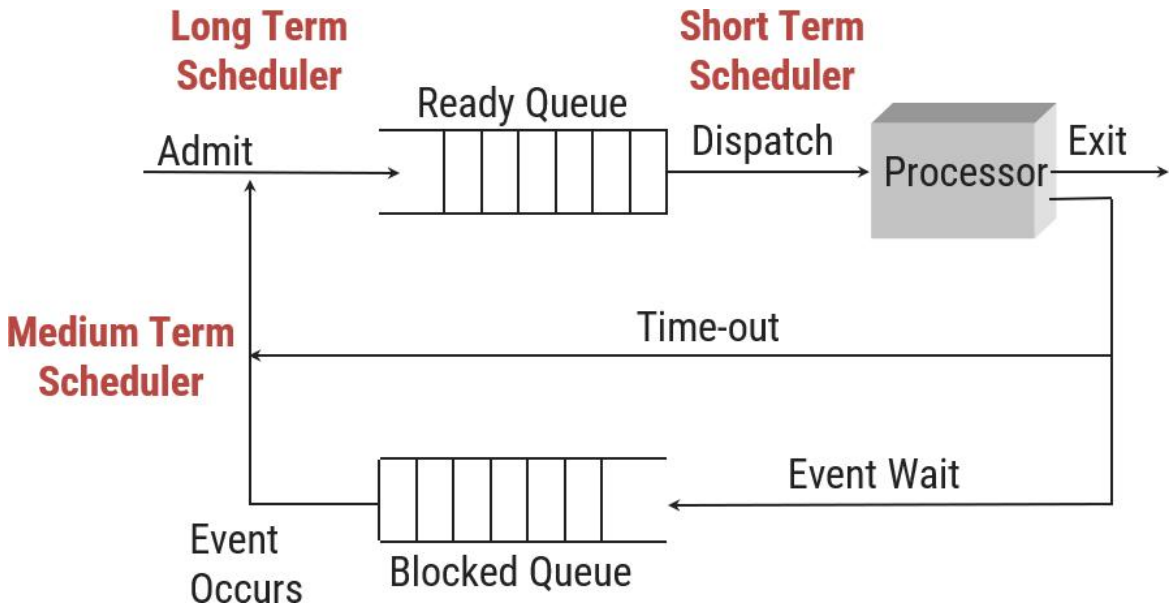
Process Scheduling Queues

- **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.



@ Types of Schedulers

Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
It is a job scheduler.	It is a CPU scheduler.	It is a process swapping scheduler.
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.
Speed is lesser than short term scheduler.	Speed is fastest among other two schedulers.	Speed is in between both short and long term scheduler.
It is almost absent or minimal in time sharing system.	It is also minimal in time sharing system.	It is a part of time sharing systems.



Scheduling Criteria

- **Fairness**: giving each process a fair share of the CPU.
- **Balance**: keeping all the parts of the system busy (Maximize).
- **Throughput**: no of processes that are completed per time unit (Maximize).
- **Turnaround time**: time to execute a process from submission to completion (Minimize).
  - ↪ **Turnaround time** = Process finish time – Process arrival time
- **CPU utilization**: percent of time that the CPU is busy in executing a process.
  - ↪ keep CPU as busy as possible (Maximized).
- **Response time**: time between issuing a command and getting the result (Minimized).
- **Waiting time**: amount of time a process has been waiting in the ready queue (Minimize).
  - ↪ **Waiting time** = Turnaround time – Actual execution time

1) **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.
2) **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.)

3) **Arrival Time(AT):**
   It is the time When process is arrived into ready queue is known arrival time. Ex: train is arrived at 7 am.

4) **Response time(RT) :**
   When process is getting a CPU that is called as response time. Amount of time it takes from when a request was submitted until the first response is produced. Ex. Train will departure at 7.10 am.

5) **Burst time(BT) :**
   Process Requires CPU for what amount of time , that time is called as burst time. Ex. Train will reach at 9 am.(2 hours)

6) **Completion time(CT):**
   Completion time of process. (9 am)

7) **Turnaround time(TAT) :**
   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).
   From Arrival time to Completion time.
   TAT = CT – AT or TAT = BT + WT

8) **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.

- Process is getting response time when process will get CPU & start execution, till then process have to wait .

  WT = TAT – BT

@ 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

1. First Come First Serve(FCFS) 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.
- 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, betterthe scheduling algorithm

Mode: Non-preemptive (can't stop execution in between)

Criteria : Arrival time

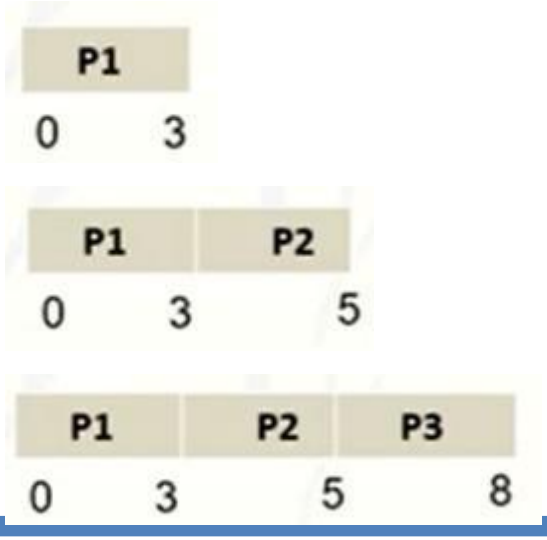
Example: given data table- Find average Turn around time and Average Waiting time.

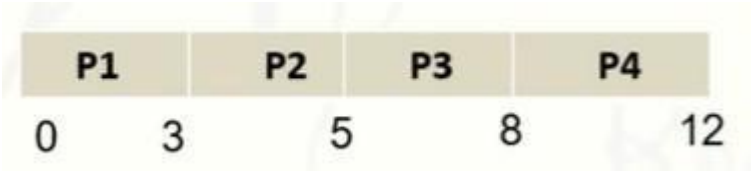
P_id	Arrival time(AT)	Burst Time (BT)
P1	0	3
P2	1	2
P3	2	3
P4	3	4
P5	4	5

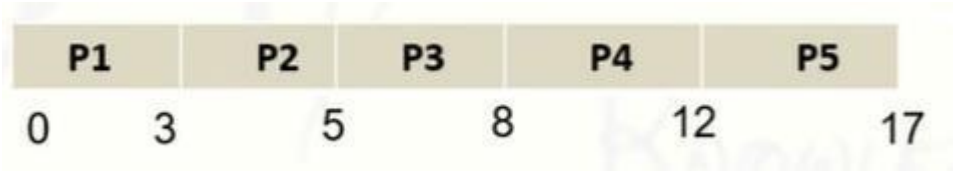
Solution:

P_id	Arrival time(AT)	Burst Time (BT)	Completion Time(CT)	Turnaround Time (TAT) (CT - AT)	Waiting Time(WT) (TAT – BT)
P1	0	3	3	3	0
P2	1	2	5	4	2
P3	2	3	8	6	3
P4	3	4	12	9	5
P5	4	5	17	13	8

Gantt chart







So

Total Turnaround time (TAT) = 3 + 4 + 6 + 9 + 13

= 35 Average TAT = 35/5 = 7 ms

Total Waiting time (WT) = 0 + 2 + 3 + 5 + 8 = 18

Average WT = 18/5 = 3.6 ms

Do your self

Example 1 : Consider Arrival time as 0 for each process if not given.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

Example 2:

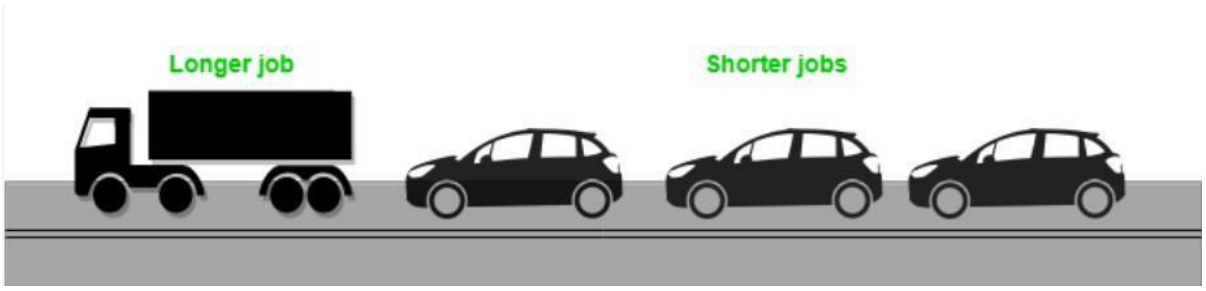
PROCESS	ARRIVAL TIME	BURST TIME
P1	0	2
P2	1	2
P3	5	3
P4	6	4

Problems in FCFS

- It is Non Pre-emptive algorithm, which means the process priority doesn't matter.
- Not optimal Average Waiting Time.
- Resources utilization in parallel is not possible, which leads to Convoy Effect, and hencepoor resource(CPU, I/O etc) utilization.

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.



- If the CPU gets the processes of the higher burst time at the front end of the ready queue then the processes of lower burst time may get blocked which means they may never get the CPU if the

job in the execution has a

very high burst time. This is called convoy effect or starvation.

2. Shortest-Job-First(SJF) Scheduling

- Shortest Job First scheduling works on the process with the shortest burst time or duration first.
- 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)
- **It is of two types:**
  - Non Pre-emptive
  - Pre-emptive

Mode: Non-preemptive (can't stop execution in between)

Criteria : Burst time

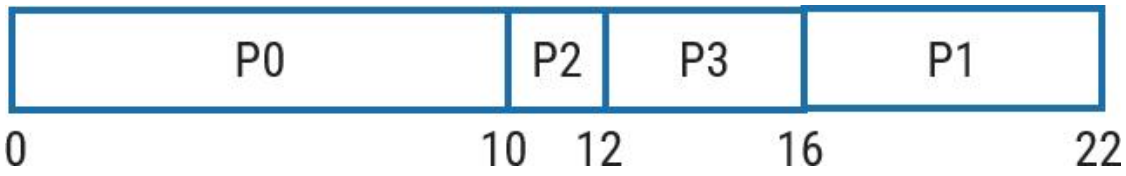
Example : find Average TAT and Average WT.

P_id	Arrival time(AT)	Burst Time (BT)
P0	0	10
P1	1	6
P2	3	2
P3	5	4

Solution:

P_id	Arrival time(AT)	Burst Time (BT)	Completion Time(CT)	Turnaround Time (TAT) (CT - AT)	Waiting Time(WT) (TAT – BT)
P0	0	10	10	10	0
P1	1	6	22	21	15
P2	3	2	12	9	7
P3	5	4	16	11	7

Gantt chart



Average TAT = 10+21+9+11 / 4 = 12.75 ms

Average WT = 0+15+7+7 /4 = 7.25 ms

Do your self:

Ex.1:

P_id	Arrival time(AT)	Burst Time (BT)
P1	1	7
P2	2	5
P3	3	2
P4	4	4
P5	5	8

Example 2 : Consider Arrival time as 0 for each process if not given.

PROCESS	BURST TIME
P1	21
P2	3
P3	6

P	2

Limitation of SJF

- SJF can't be implemented for CPU scheduling for the short time as we can't predict the length of upcoming CPU burst.
- It is difficult to estimate time required to complete execution.
- Starvation is possible for long process. Long process may wait forever. (Starvation is the problem that occur when high priority processes keep executing and low priority processes get blocked for indefinite time.)

3. Priority Scheduling

- Priority is a method of scheduling processes that is based on priority.
- In this algorithm, scheduler selects the tasks to work as per the priority.
- The processes with higher priority should be carried first, where as jobs with equal priorities are carried out on FCFS basis.
- Priority depends upon memory requirements, time requirements etc.

Types of Priority:

1. Non-pre-emptive priority

- The process , that has higher priority, is served first.
- Priority can be higher number higher priority/lower number higher priority.

Mode: Non-preemptive (can't stop execution in between)

Criteria : Priority and Arrival time

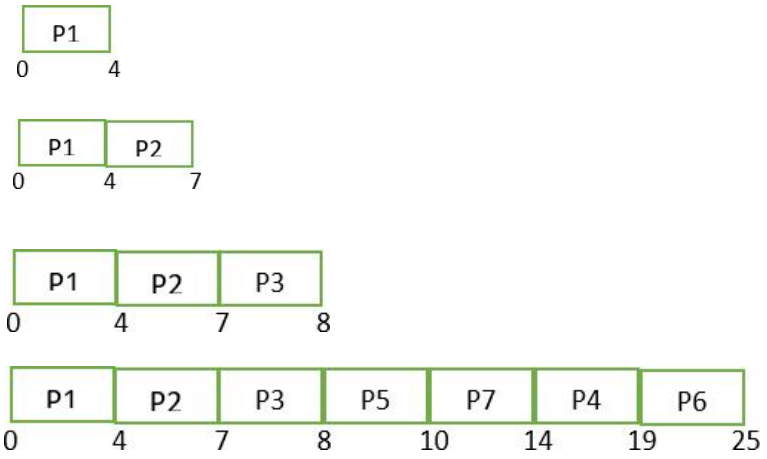
P_id	Priority	Arrival time(AT)	Burst Time (BT)
P1	1	0	4
P2	3	1	3
P3	5	2	1
P4	9	3	5
P5	7	4	2
P6	11	5	6
P7	8	6	4

Solution:

Priority considered as: lower value higher priority

P_id	Priority	Arrival time(AT)	Burst Time (BT)	CT	TAT(CT - AT)	WT(TAT-BT)
P1	1	0	4	4	4	0
P2	3	1	3	7	6	3
P3	5	2	1	8	6	5
P4	9	3	5	19	16	11
P5	7	4	2	10	6	4
P6	11	5	6	25	20	14
P7	8	6	4	14	8	4

Gantt chart



Average TAT = 4+6+6+16+6+20+8/7 = 9.43

Average WT = 0+3+5+11+4+14+4/7 = 5.86

Do your self:

Ex:1

P_id	Priority	Burst Time (BT)
P1	2	21
P2	1	3
P3	4	6
P4	3	2

Ex. 2

P_id	Priority	Arrival time	Burst Time (BT)
P0	5	0	10
P1	4	1	6
P2	2	3	2
P3	0	5	4

Advantages

- Priority is considered so critical processes can get even better response time.

Disadvantages

Starvation is possible for low priority processes. It can be overcome by using technique called 'Aging'.

Aging: gradually increases the priority of processes that wait in the system for a long time.

2. Pre-emptive priority

- **Preemptive:** When a new process arrives, its priority is compared with current process priority.
- If the new process has higher priority than the current, the current process is suspended and new job is started.
- This strategy can also be implemented by using **sorted FIFO queue**.
- All processes in a queue are sorted based on their priority with highest

priority process at front end.

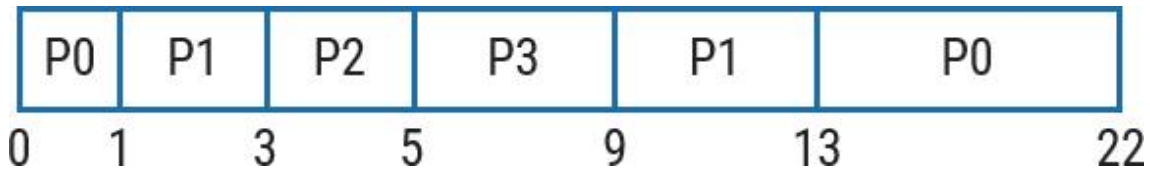
- When CPU becomes free, a process from the first position in a queue is selected to run.

Process	Arrival Time (T0)	Burst Time (ΔT)	Priority
P0	0	10	5
P1	1	6	4
P2	3	2	2
P3	5	4	0

Solution:

Process	Arrival Time (T0)	Burst Time (ΔT)	Priority	Finish Time (T1)	Turnaround Time (TAT = T1 - T0)	Waiting Time (WT = TAT - ΔT)
P0	0	10	5	22	22	12
P1	1	6	4	13	12	6
P2	3	2	2	5	2	0
P3	5	4	0	9	4	0

Gantt chart



Avg. Turnaround Time: 10

ms Avg. Waiting Time:

4.5

ms

Advantages

- Priority is considered so critical processes can get even better response time.

Disadvantages

- Starvation is possible for low priority processes. It can be overcome by using technique called 'Aging'.
- **Aging:** gradually increases the priority of processes that wait in the system for a long time.
- Context switch overhead is there.

Do your self : solve same problem of non-preemptive .

4. Round Robin (RR)

- Each selected process is assigned a time interval, called time quantum or time slice.
- Process is allowed to run only for this time interval.
- Here, two things are possible:
- First, process is either blocked or terminated before the quantum has elapsed. In this case the CPU switching is done and another process is scheduled to run.
- Second, process needs CPU burst longer than time quantum. In this case, process is running at the end of the time quantum.
- Here, length of time quantum is critical to determine.

Mode : Preemptive

Selection of new job is **as per FCFS** scheduling

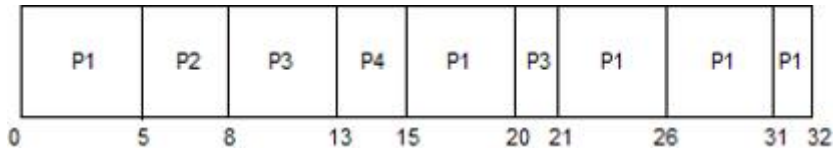
algorithm. Ex: Time quantum (q) : 5

P_id	Burst Time (BT)
P1	21
P2	3
P3	6
P4	2

Solution:

P_id	Burst Time (BT)	CT	TAT	WT
P1	21	32	32	11
P2	3	8	8	5
P3	6	20	20	14
P4	2	15	15	13

Gantt chart



Average TAT :18.75 ms

Average WT: 10.75

Do yourself: do above example using time quantum 3 and 4 .