

Unit II – Process Management

Process

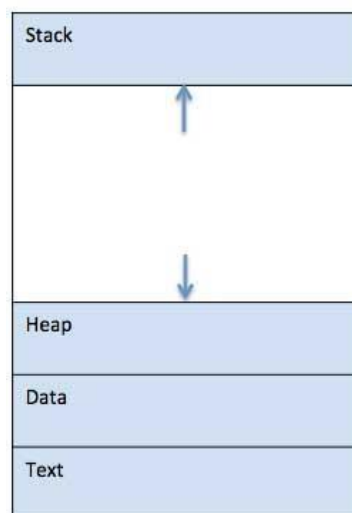
Definition:

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

- Process is not as same as program code but a lot more than it.
- 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.

Process memory is divided into four sections for efficient working :

- The **Text section** is made up of the compiled program code, read in from non-volatile storage when the program is launched.
- The **Data section** is made up the global and static variables, allocated and initialized prior to executing the main.
- The **Heap** is used for the dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
- The **Stack** is used for local variables. Space on the stack is reserved for local variables when they are declared.

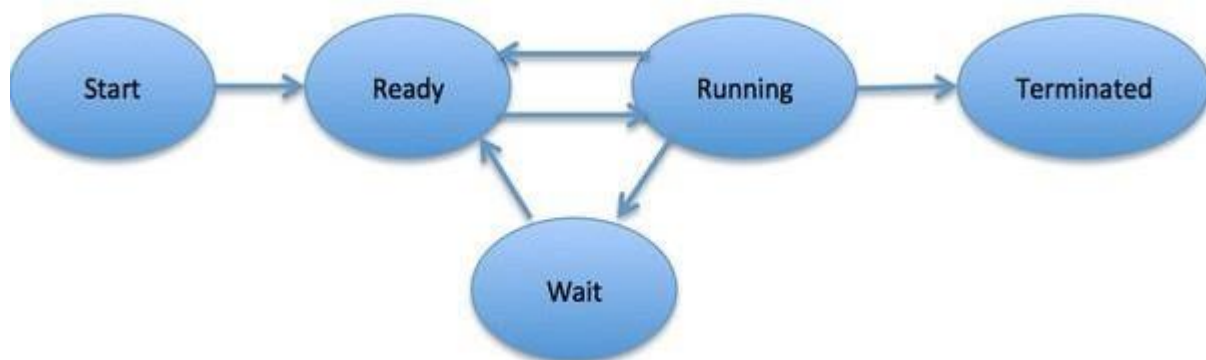


Process States:

When a process executes, it passes through different states. These stages may differ in different operating systems.

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

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



Process State Diagram

Process Control Block (PCB):

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

S.N.	Information & Description
1	Process State: The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	Process privileges: This is required to allow/disallow access to system resources.
3	Process ID: Unique identification for each of the process in the operating system.
4	Pointer: A pointer to parent process.
5	Program Counter: Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	CPU registers: Various CPU registers where process need to be stored for execution for running state.

7	CPU Scheduling Information: Process priority and other scheduling information which is required to schedule the process.
8	Memory management information: This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	Accounting information: This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	IO status information: This includes a list of I/O devices allocated to the process.

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



Process Control Block (PCB) Diagram

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

Process Scheduling

Definition

- The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process based on a particular strategy.
- Process scheduling is an essential part of Multiprogramming operating systems.
- Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

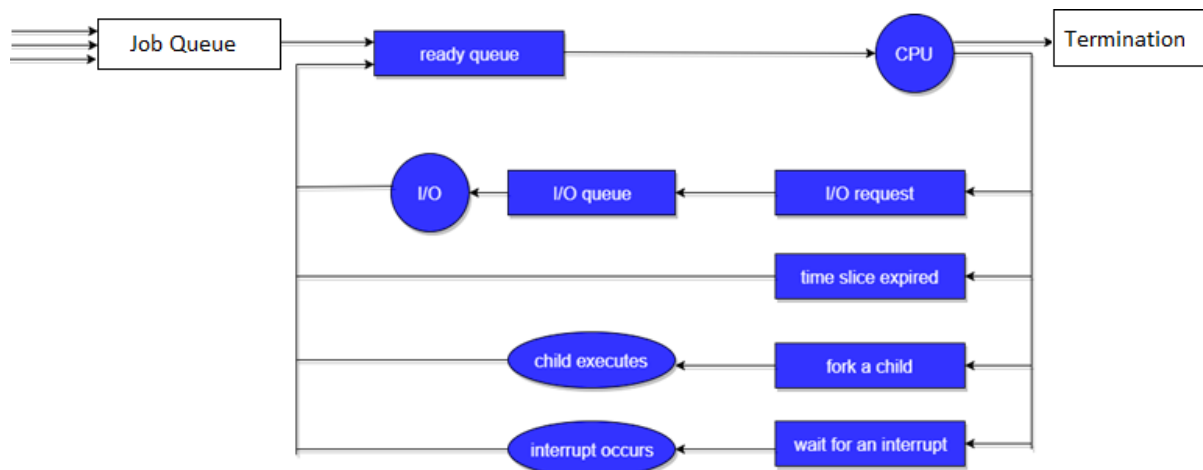
What are Scheduling Queues?

- All processes, upon entering into the system, are stored in the **Job Queue**.
- Processes in the Ready state are placed in the **Ready Queue**.
- Processes waiting for a device to become available are placed in **Device Queues**.

There are unique device queues available for each I/O device.

A new process is initially put in the **Ready queue**. It waits in the ready queue until it is selected for execution (or dispatched). Once the process is assigned to the CPU and is executing, one of the following several events can occur:

- The process could issue an I/O request, and then be placed in the **I/O queue**.
- The process could create a new sub-process and wait for its termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.



In the first two cases, the process eventually switches from the waiting state to the ready state and is then put back in the ready queue. A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.

Schedulers:

- Schedulers are special system software which handle process scheduling in various ways.
- Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types –
 - Long-Term Scheduler
 - Short-Term Scheduler
 - Medium-Term Scheduler

Long Term Scheduler

- It is also called a job scheduler.
- A long-term scheduler determines which programs are admitted to the system for processing.
- It selects processes from the queue and loads them into memory for execution.
- Process loads into the memory for CPU scheduling.
- The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound.

- It also controls the degree of multiprogramming.
- If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.
- On some systems, the long-term scheduler may not be available or minimal.
- Time-sharing operating systems have no long term scheduler.
- When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler:

- It is also called as CPU scheduler.
- Its main objective is to increase system performance in accordance with the chosen set of criteria.
- It is the change of ready state to running state of the process.
- CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.
- Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

- Medium-term scheduling is a part of swapping.
- It removes the processes from the memory.
- It reduces the degree of multiprogramming.
- The medium-term scheduler is in-charge of handling the swapped out-processes.
- A running process may become suspended if it makes an I/O request.
- A suspended process cannot make any progress towards completion.
- In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage.
- This process is called swapping, and the process is said to be swapped out or rolled out.
- Swapping may be necessary to improve the process mix.

Comparison among Scheduler

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

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

Dispatcher

A dispatcher is a special program which comes into play after the scheduler. When the scheduler completes its job of selecting a process, it is the dispatcher which takes that process to the desired state/queue. The dispatcher is the module that gives a process control over the CPU after it has been selected by the short-term scheduler. This function involves the following:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program

Role of the dispatcher:

1. The dispatcher gives control of the CPU to the process selected by the short-term scheduler.
2. A dispatcher is a communications worker who receives and transmits information to coordinate operations of other personnel and vehicles carrying out a service.
3. A dispatcher is a special program which comes into play after the scheduler.
4. When the scheduler completes its job of selecting a process, it is the dispatcher which takes that process to the desired state/ queue.

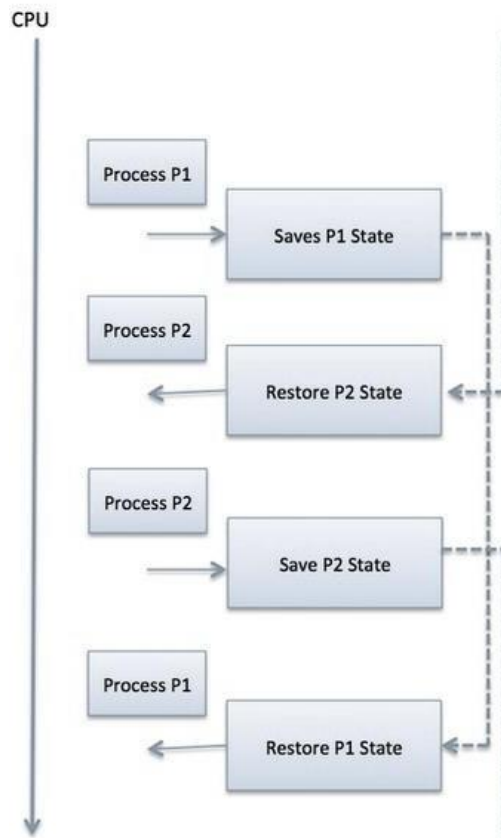
Difference Between Dispatcher and Scheduler

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 different 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 dependent on scheduler. Means dispatcher have to wait until scheduler selects a process.	Scheduler works independently. 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.

Properties	DISPATCHER	SCHEDULER
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.
Tasks	Dispatcher allocates the CPU to the process selected by the short-time scheduler.	Scheduler performs three task. Job scheduling (Long-term scheduler), <u>CPU scheduling</u> (Short-term scheduler) and swapping (Medium-term scheduler).
Purpose	To move the process from the ready queue to the CPU	To select the process and decide which process to run
Execution time	It takes a very short execution time	It takes longer execution time than dispatcher
Interaction	The dispatcher works with the CPU and the selected process	The scheduler works with the ready queue and the dispatcher.

Context Switch:

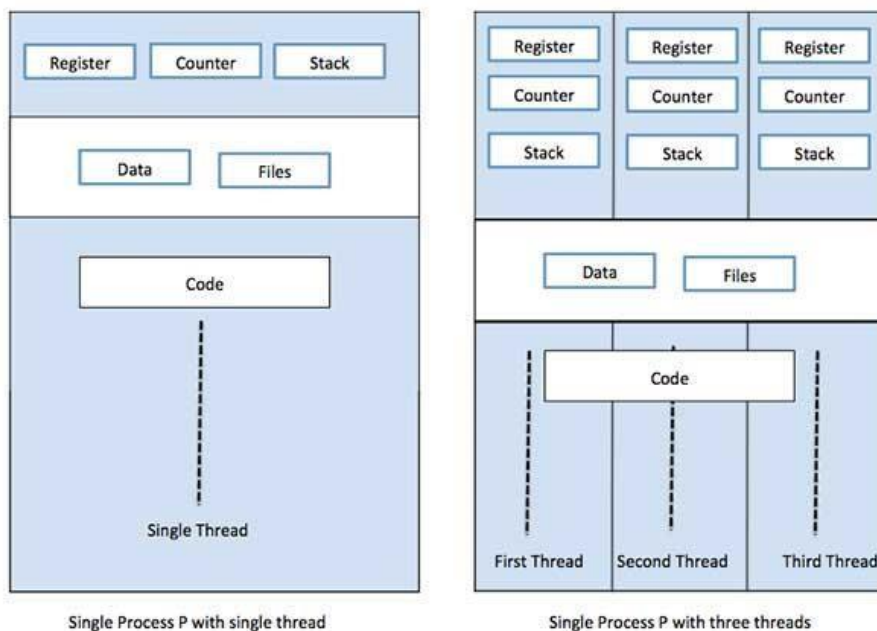
- A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time.
- Using this technique, a context switcher enables multiple processes to share a single CPU.
- Context switching is an essential part of a multitasking operating system features.
- When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block.
- After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc.
- At that point, the second process can start executing.



- Context switches are computationally intensive since register and memory state must be saved and restored.
- To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers.
- When the process is switched, the following information is stored for later use.
 - Program Counter
 - Scheduling information
 - Base and limit register value
 - Currently used register
 - Changed State
 - I/O State information
 - Accounting information

What is Thread?

- A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.
- A thread shares with its peer threads few information like code segment, data segment and open files.
- When one thread alters a code segment memory item, all other threads see that.
- A thread is also called a **lightweight process**.
- Threads provide a way to improve application performance through parallelism.
- Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.
- Each thread belongs to exactly one process and no thread can exist outside a process.
- Each thread represents a separate flow of control.
- Threads have been successfully used in implementing network servers and web server.
- They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors.
- The following figure shows the working of a single-threaded and a multithreaded process.



Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.

3	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.
---	---	--

4	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.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

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.

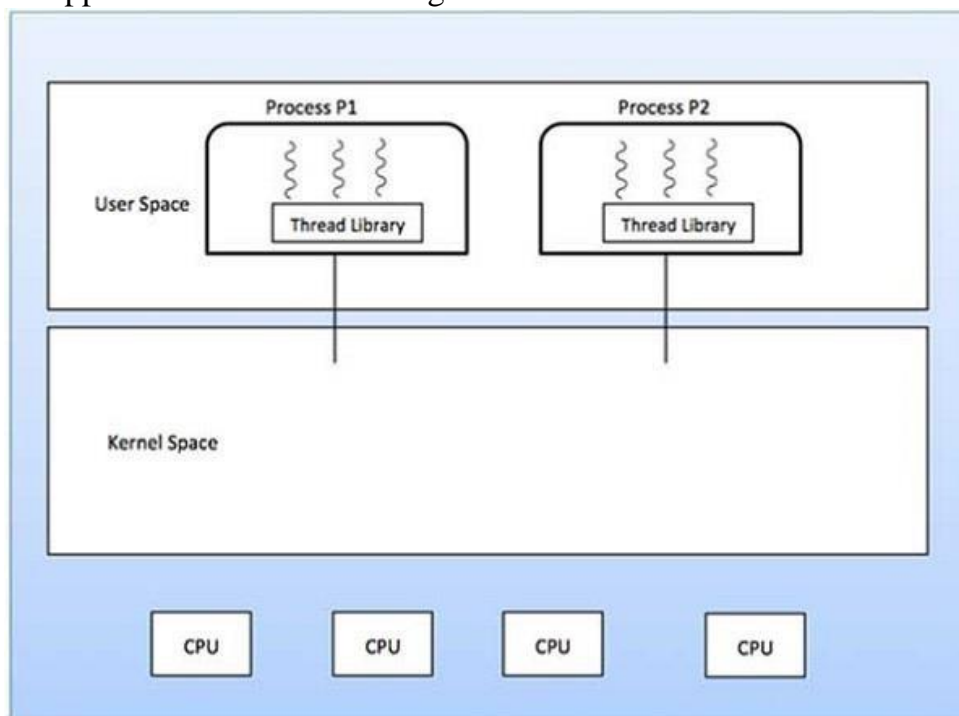
Types of Thread

Threads are implemented in following two ways –

- **User Level Threads** – User-managed threads.
- **Kernel Level Threads** – Operating System managed threads acting on kernel, an operating system core.

User Level Threads

- 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.



Advantages

- 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

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads

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. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individual threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

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

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.

Multithreading Models

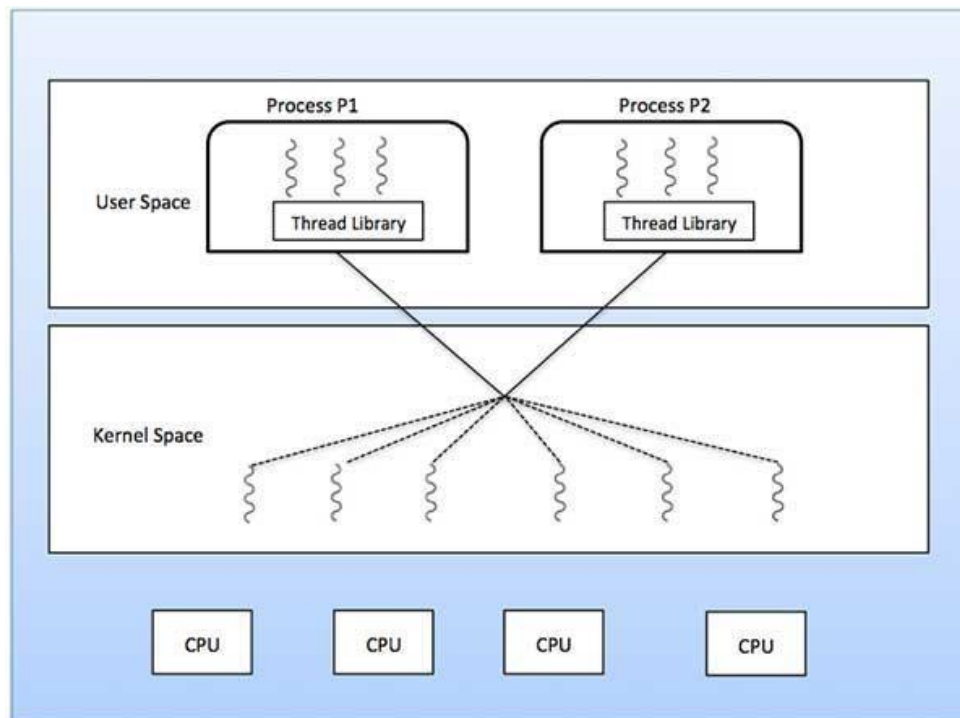
Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- Many to many relationship.
- Many to one relationship.
- One to one relationship.

Many to Many Model

- The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.
- The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads.
- In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine.
- This model provides the best accuracy on concurrency and when a thread performs

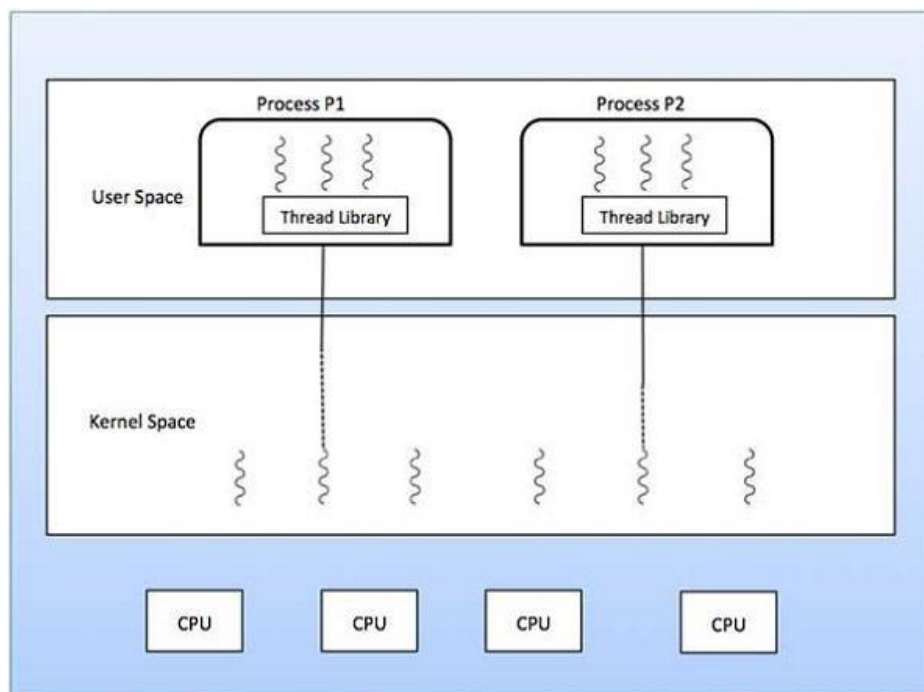
a blocking system call, the kernel can schedule another thread for execution.



Many to One Model

Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

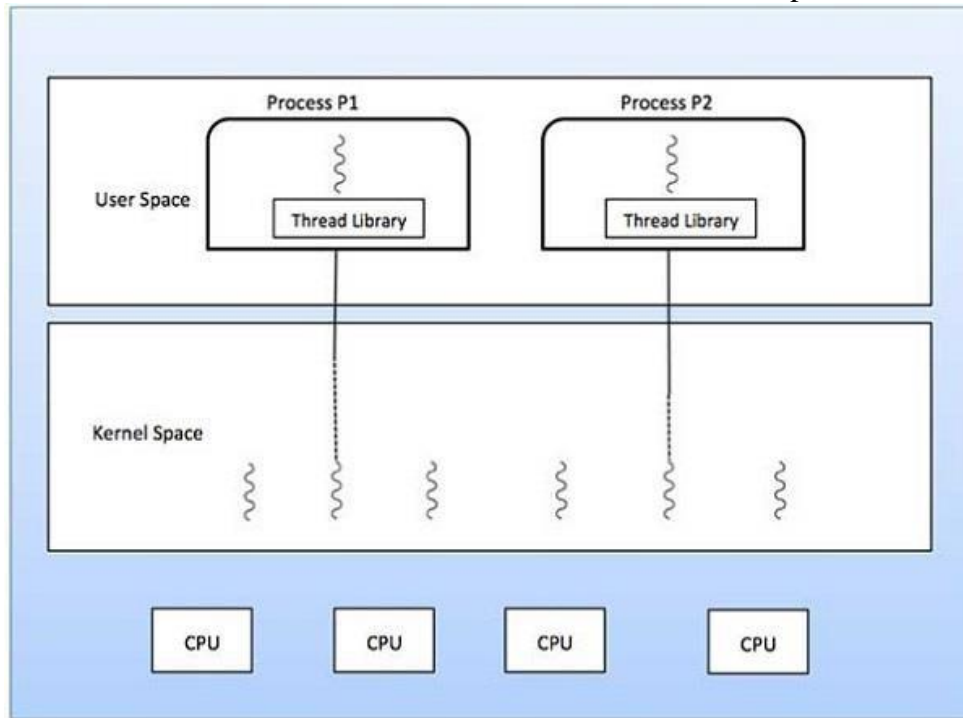
If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.



One to One Model

There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.



Difference between User-Level & Kernel-Level Thread

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

CPU Scheduling Algorithms

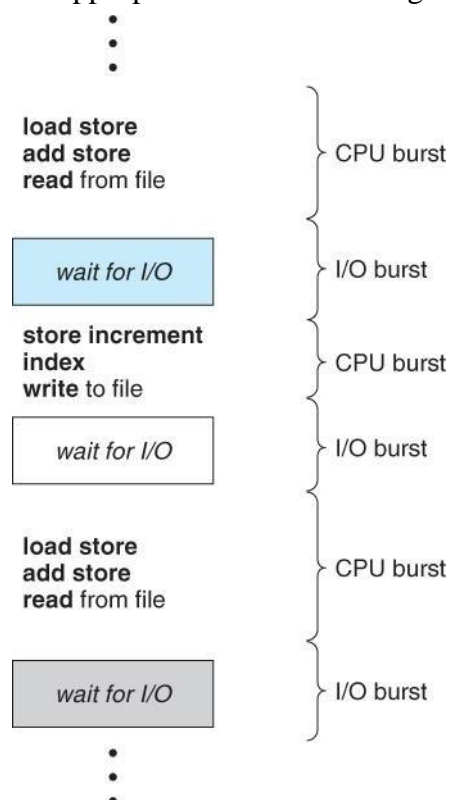
Scheduling types

Scheduling Objectives

- Be Fair while allocating resources to the processes
- Maximize the throughput of the system
- Maximize the number of users receiving acceptable response times.
- Be predictable
- Balance resource use
- Avoid indefinite postponement
- Enforce Priorities
- Give preference to processes holding key resources
- Give better service to processes that have desirable behavior patterns

CPU and I/O Burst Cycle:

- Process execution consists of a cycle of CPU execution and I/O wait.
- Processes alternate between these two states.
- Process execution begins with a CPU burst, followed by an I/O burst, then another CPU burst ... etc
- The last CPU burst will end with a system request to terminate execution rather than with another I/O burst.
- The duration of these CPU bursts has been measured.
- An I/O-bound program would typically have many short CPU bursts, A CPU-bound program might have a few very long CPU bursts.
- This can help to select an appropriate CPU-scheduling algorithm.



Preemptive Scheduling:

- Preemptive scheduling is used when a process switches from running state to ready state or from waiting state to ready state.
- The resources (mainly CPU cycles) are allocated to the process for the limited amount of time and then is taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining.
- That process stays in ready queue till it gets next chance to execute.

Non-Preemptive Scheduling:

- Non-preemptive Scheduling is used when a process terminates, or a process switches from running to waiting state.
- In this scheduling, once the resources (CPU cycles) is allocated to a process, the process holds the CPU till it gets terminated or it reaches a waiting state.
- In case of non-preemptive scheduling does not interrupt a process running CPU in middle of the execution.
- Instead, it waits till the process complete its CPU burst time and then it can allocate the CPU to another process.

Basis for Comparison	Preemptive Scheduling	Non Preemptive Scheduling
Basic	The resources are allocated to a process for a limited time.	Once resources are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
Interrupt	Process can be interrupted in between.	Process can not be interrupted till it terminates or switches to waiting state.
Starvation	If a high priority process frequently arrives in the ready queue, low priority process may starve.	If a process with long burst time is running CPU, then another process with less CPU burst time may starve.
Overhead	Preemptive scheduling has overheads of scheduling the processes.	Non-preemptive scheduling does not have overheads.
Flexibility	Preemptive scheduling is flexible.	Non-preemptive scheduling is rigid.
Cost	Preemptive scheduling is cost associated.	Non-preemptive scheduling is not cost associative.

Scheduling Criteria

- There are several different criteria to consider when trying to select the "best" scheduling algorithm for a particular situation and environment, including:
 - **CPU utilization** - Ideally the CPU would be busy 100% of the time, so as to waste 0 CPU cycles. On a real system CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)
 - **Throughput** - Number of processes completed per unit time. May range from 10 / second to 1 / hour depending on the specific processes.

- **Turnaround time** - Time required for a particular process to complete, from submission time to completion.
- **Waiting time** - How much time processes spend in the ready queue waiting their turn to get on the CPU.
- **Response time** - The time taken in an interactive program from the issuance of a command to the *commence* of a response to that command.

In brief:

Arrival Time: Time at which the process arrives in the ready queue.

Completion Time: Time at which process completes its execution.

Burst Time: Time required by a process for CPU execution.

Turn Around Time: Time Difference between completion time and arrival time.

Turn Around Time = Completion Time – Arrival Time

Waiting Time(W.T): Time Difference between turnaround time and burst time.

Waiting Time = Turn Around Time – Burst Time

Types of Scheduling Algorithm

(a) First Come First Serve (FCFS)

In FCFS Scheduling

- The process which arrives first in the ready queue is firstly assigned the CPU.
- In case of a tie, process with smaller process id is executed first.
- It is always non-preemptive in nature.
- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Advantages-

- It is simple and easy to understand.
- It can be easily implemented using queue data structure.
- It does not lead to starvation.

Disadvantages-

- It does not consider the priority or burst time of the processes.
- It suffers from convoy effect i.e. processes with higher burst time arrived before the processes with smaller burst time.

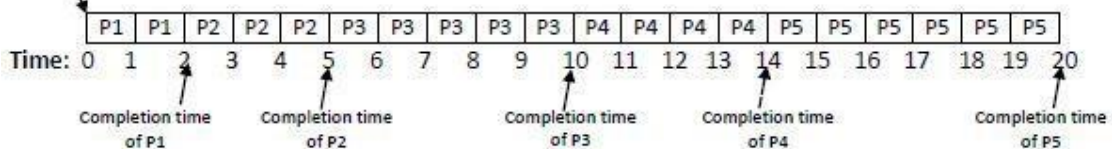
Example 1:

Q. Consider the following processes with burst time (CPU Execution time). Calculate the average waiting time and average turnaround time?

Process id	Arrival time	Burst time/CPU execution time
P1	0	2
P2	1	3
P3	2	5
P4	3	4
P5	4	6

Sol.

Gantt chart



Turnaround time = Completion time – Arrival time

Waiting time = Turnaround time – Burst time

Process id	Arrival time	Burst time	Completion time	Turnaround time	Waiting time
P1	0	2	2	2-0=2	2-2=0
P2	1	3	5	5-1=4	4-3=1
P3	2	5	10	10-2=8	8-5=3
P4	3	4	14	14-3=11	11-4=7
P5	4	6	20	20-4=16	16-6=10

Average turnaround time = $\sum_{i=0}^n \text{Turnaround time}(i)/n$

where, n= no. of process

Average waiting time = $\sum_{i=0}^n \text{Waiting time}(i)/n$

where, n= no. of process

Average turnaround time = $2+4+8+11+16/5 = 41/5 = 8.2$

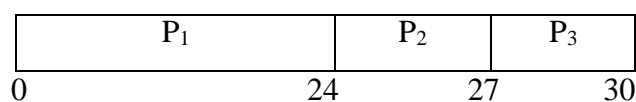
Average waiting time = $0+1+3+7+10/5 = 21/5 = 4.2$

Example 2:

Consider the processes P1, P2, P3 given in the below table, arrives for execution in the same order, with Arrival Time 0, and given Burst Time,

PROCESS	ARRIVAL TIME	BURST TIME
P1	0	24
P2	0	3
P3	0	3

Gantt chart



PROCESS	WAIT TIME	TURN AROUND TIME
P1	0	24
P2	24	27
P3	27	30

Total Wait Time = $0 + 24 + 27 = 51$ ms

Average Waiting Time = (Total Wait Time) / (Total number of processes) = $51/3 = 17$ ms

Total Turn Around Time: $24 + 27 + 30 = 81$ ms

Average Turn Around time = (Total Turn Around Time) / (Total number of processes)
 $= 81 / 3 = 27$ ms

Throughput = $3 \text{ jobs}/30 \text{ sec} = 0.1 \text{ jobs/sec}$

Example 3:

Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with given Arrival Time and Burst Time.

PROCESS	ARRIVAL TIME	BURST TIME
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Gantt chart

P ₁	P ₂	P ₃	P ₄	
0	8	12	21	26

PROCESS	WAIT TIME	TURN AROUND TIME
P1	0	$8 - 0 = 8$
P2	$8 - 1 = 7$	$12 - 1 = 11$
P3	$12 - 2 = 10$	$21 - 2 = 19$
P4	$21 - 3 = 18$	$26 - 3 = 23$

Total Wait Time:= $0 + 7 + 10 + 18 = 35$ ms

Average Waiting Time = (Total Wait Time) / (Total number of processes)= $35/4 = 8.75$ ms

Total Turn Around Time: $8 + 11 + 19 + 23 = 61$ ms

Average Turn Around time = (Total Turn Around Time) / (Total number of processes)
 $61/4 = 15.25$ ms

Throughput: $4 \text{ jobs}/26 \text{ sec} = 0.15385 \text{ jobs/sec}$

(b) Shortest Job First (SJF)

- Process which have the shortest burst time are scheduled first.
- If two processes have the same burst time, then FCFS is used to break the tie.
- This is a non-pre-emptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processor should know in advance how much time process will take.
- Pre-emptive mode of Shortest Job First is called as Shortest Remaining Time First (SRTF).

Advantages-

- SRTF is optimal and guarantees the minimum average waiting time.
- It provides a standard for other algorithms since no other algorithm performs better than it.

Disadvantages-

- It can not be implemented practically since burst time of the processes can not be known in advance.
- It leads to starvation for processes with larger burst time.
- Priorities can not be set for the processes.
- Processes with larger burst time have poor response time.

Example-01:

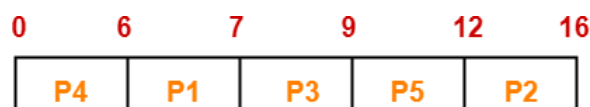
Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

Solution-

If the CPU scheduling policy is SJF non-preemptive, calculate the average waiting time and average turnaround time.

Gantt Chart-



Gantt Chart

Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	7	$7 - 3 = 4$	$4 - 1 = 3$
P2	16	$16 - 1 = 15$	$15 - 4 = 11$
P3	9	$9 - 4 = 5$	$5 - 2 = 3$
P4	6	$6 - 0 = 6$	$6 - 6 = 0$
P5	12	$12 - 2 = 10$	$10 - 3 = 7$

Now,

- Average Turn Around time = $(4 + 15 + 5 + 6 + 10) / 5 = 40 / 5 = 8$ unit
- Average waiting time = $(3 + 11 + 3 + 0 + 7) / 5 = 24 / 5 = 4.8$ unit

Example-02:

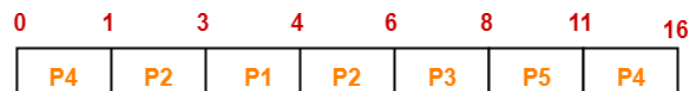
Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

If the CPU scheduling policy is SJF pre-emptive, calculate the average waiting time and average turnaround time.

Solution-

Gantt Chart-



Gantt Chart

Process Id	Exit time	Turn Around time	Waiting time
P1	4	$4 - 3 = 1$	$1 - 1 = 0$
P2	6	$6 - 1 = 5$	$5 - 4 = 1$
P3	8	$8 - 4 = 4$	$4 - 2 = 2$
P4	16	$16 - 0 = 16$	$16 - 6 = 10$
P5	11	$11 - 2 = 9$	$9 - 3 = 6$

Now,

- Average Turn Around time = $(1 + 5 + 4 + 16 + 9) / 5 = 35 / 5 = 7$ unit
- Average waiting time = $(0 + 1 + 2 + 10 + 6) / 5 = 19 / 5 = 3.8$ unit

Example-03:

Consider the set of 6 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	7
P2	1	5
P3	2	3
P4	3	1
P5	4	2
P6	5	1

If the CPU scheduling policy is shortest remaining time first, calculate the average waiting time and average turnaround time.

Solution-
Gantt Chart-


Gantt Chart

Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	19	$19 - 0 = 19$	$19 - 7 = 12$
P2	13	$13 - 1 = 12$	$12 - 5 = 7$
P3	6	$6 - 2 = 4$	$4 - 3 = 1$
P4	4	$4 - 3 = 1$	$1 - 1 = 0$
P5	9	$9 - 4 = 5$	$5 - 2 = 3$
P6	7	$7 - 5 = 2$	$2 - 1 = 1$

Now,

- Average Turn Around time = $(19 + 12 + 4 + 1 + 5 + 2) / 6 = 43 / 6 = 7.17$ unit
- Average waiting time = $(12 + 7 + 1 + 0 + 3 + 1) / 6 = 24 / 6 = 4$ unit

Example -04:

Consider the set of 3 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	9
P2	1	4
P3	2	9

If the CPU scheduling policy is SRTF, calculate the average waiting time and average turn around time.

Solution-
Gantt Chart-


Gantt Chart

Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	13	$13 - 0 = 13$	$13 - 9 = 4$
P2	5	$5 - 1 = 4$	$4 - 4 = 0$
P3	22	$22 - 2 = 20$	$20 - 9 = 11$

Now,

- Average Turn Around time = $(13 + 4 + 20) / 3 = 37 / 3 = 12.33$ unit
- Average waiting time = $(4 + 0 + 11) / 3 = 15 / 3 = 5$ unit

Example-05:

Consider the set of 4 processes whose arrival time and burst time are given below-

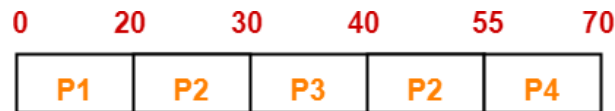
Process Id	Arrival time	Burst time
P1	0	20
P2	15	25
P3	30	10

P4	45	15
----	----	----

If the CPU scheduling policy is SRTF, calculate the waiting time of process P2.

Solution-

Gantt Chart-



Gantt Chart

Now, we know-

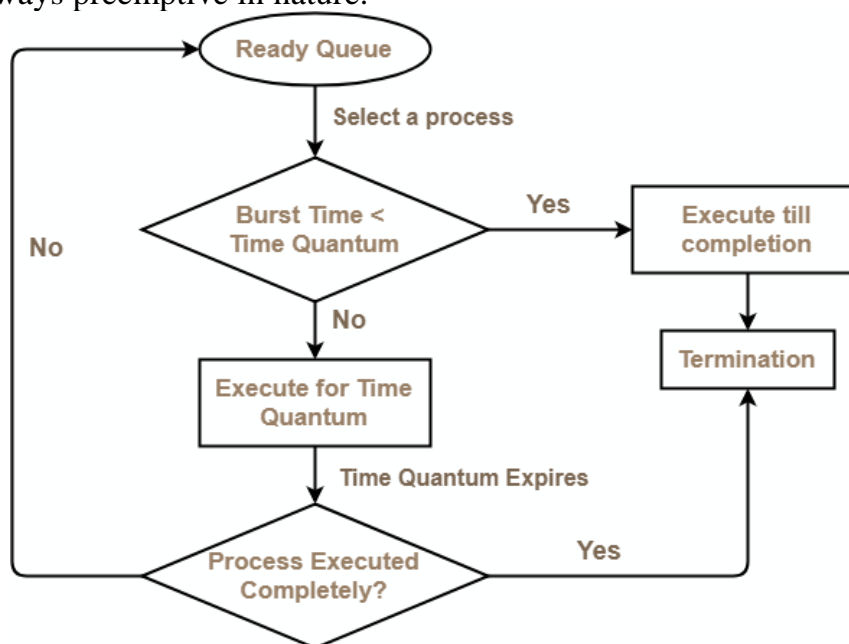
- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Thus,

- Turn Around Time of process P2 = 55 – 15 = 40 unit
- Waiting time of process P2 = 40 – 25 = 15 unit

(c) **Round Robin Scheduling**

- CPU is assigned to the process on the basis of FCFS for a fixed amount of time.
- This fixed amount of time is called as time quantum or time slice.
- After the time quantum expires, the running process is preempted and sent to the ready queue.
- Then, the processor is assigned to the next arrived process.
- It is always preemptive in nature.



Round Robin Scheduling

Advantages-

- It gives the best performance in terms of average response time.
- It is best suited for time sharing system, client server architecture and interactive system.

Disadvantages-

- It leads to starvation for processes with larger burst time as they have to repeat the cycle many times.
- Its performance heavily depends on time quantum.
- Priorities can not be set for the processes.

With decreasing value of time quantum,

- Number of context switch increases
- Response time decreases
- Chances of starvation decreases

Thus, smaller value of time quantum is better in terms of response time.

With increasing value of time quantum,

- Number of context switch decreases
- Response time increases
- Chances of starvation increases

Thus, higher value of time quantum is better in terms of number of context switch.

- With increasing value of time quantum, Round Robin Scheduling tends to become FCFS Scheduling.
- When time quantum tends to infinity, Round Robin Scheduling becomes FCFS Scheduling.
- The performance of Round Robin scheduling heavily depends on the value of time quantum.
- The value of time quantum should be such that it is neither too big nor too small.

Example-01:

Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2

P5	4	3
----	---	---

If the CPU scheduling policy is Round Robin with time quantum = 2 unit, calculate the average waiting time and average turnaround time.

Solution-

Ready Queue- _____ P5, P1, P2, P5, P4, P1, P3, P2, P1

Gantt Chart-



Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	13	$13 - 0 = 13$	$13 - 5 = 8$
P2	12	$12 - 1 = 11$	$11 - 3 = 8$
P3	5	$5 - 2 = 3$	$3 - 1 = 2$
P4	9	$9 - 3 = 6$	$6 - 2 = 4$
P5	14	$14 - 4 = 10$	$10 - 3 = 7$

Now,

- Average Turn Around time = $(13 + 11 + 3 + 6 + 10) / 5 = 43 / 5 = 8.6$ unit
- Average waiting time = $(8 + 8 + 2 + 4 + 7) / 5 = 29 / 5 = 5.8$ unit

Problem-02:

Consider the set of 6 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	4
P2	1	5
P3	2	2
P4	3	1
P5	4	6
P6	6	3

If the CPU scheduling policy is Round Robin with time quantum = 2, calculate the *average waiting time* and *average turnaround time*.

Solution-

Ready Queue- P5, P6, P2, P5, P6, P2, P5, P4, P1, P3, P2, P1

Gantt chart-



Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	8	$8 - 0 = 8$	$8 - 4 = 4$
P2	18	$18 - 1 = 17$	$17 - 5 = 12$
P3	6	$6 - 2 = 4$	$4 - 2 = 2$
P4	9	$9 - 3 = 6$	$6 - 1 = 5$
P5	21	$21 - 4 = 17$	$17 - 6 = 11$
P6	19	$19 - 6 = 13$	$13 - 3 = 10$

Now,

- Average Turn Around time = $(8 + 17 + 4 + 6 + 17 + 13) / 6 = 65 / 6 = 10.84$ unit
- Average waiting time = $(4 + 12 + 2 + 5 + 11 + 10) / 6 = 44 / 6 = 7.33$ unit

Problem-03: Consider the set of 6 processes whose arrival time and burst time are given below-

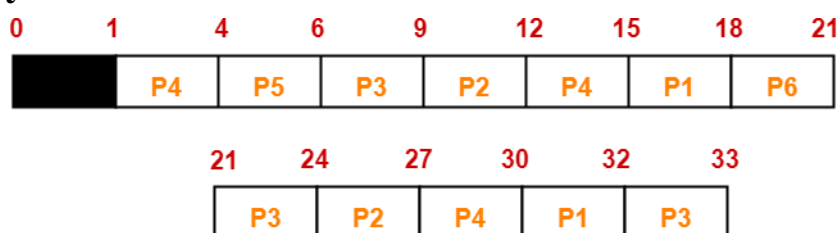
Process Id	Arrival time	Burst time
P1	5	5
P2	4	6
P3	3	7
P4	1	9
P5	2	2
P6	6	3

If the CPU scheduling policy is Round Robin with time quantum = 3, calculate the average waiting time and average turnaround time.

Solution-

Ready Queue- P3, P1, P4, P2, P3, P6, P1, P4, P2, P3, P5, P4

Gantt chart-



Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

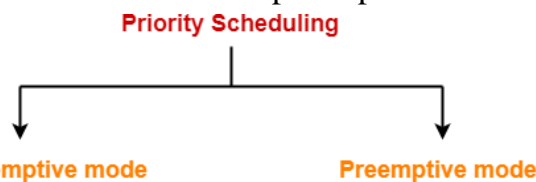
Process Id	Exit time	Turn Around time	Waiting time
P1	32	$32 - 5 = 27$	$27 - 5 = 22$
P2	27	$27 - 4 = 23$	$23 - 6 = 17$
P3	33	$33 - 3 = 30$	$30 - 7 = 23$
P4	30	$30 - 1 = 29$	$29 - 9 = 20$
P5	6	$6 - 2 = 4$	$4 - 2 = 2$
P6	21	$21 - 6 = 15$	$15 - 3 = 12$

Now,

- Average Turn Around time = $(27 + 23 + 30 + 29 + 4 + 15) / 6 = 128 / 6 = 21.33$ unit
- Average waiting time = $(22 + 17 + 23 + 20 + 2 + 12) / 6 = 96 / 6 = 16$ unit

(d) Priority Scheduling

- Out of all the available processes, CPU is assigned to the process having the highest priority.
- In case of a tie, it is broken by **FCFS Scheduling**.
- Priority Scheduling can be used in both preemptive and non-preemptive mode.



- The waiting time for the process having the highest priority will always be zero in preemptive mode.
- The waiting time for the process having the highest priority may not be zero in non-preemptive mode.

Priority scheduling in preemptive and non-preemptive mode behaves exactly same under following conditions-

- The arrival time of all the processes is same
- All the processes become available

Advantages-

- It considers the priority of the processes and allows the important processes to run first.
- Priority scheduling in pre-emptive mode is best suited for real time operating system.

Disadvantages-

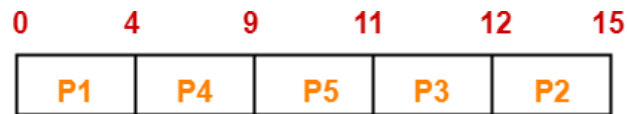
- Processes with lesser priority may starve for CPU.
- There is no idea of response time and waiting time.

Problem-01:

Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time	Priority
P1	0	4	2
P2	1	3	3
P3	2	1	4
P4	3	5	5
P5	4	2	5

If the CPU scheduling policy is priority non-preemptive, calculate the average waiting time and average turnaround time. *(Higher number represents higher priority)*

**Solution-
Gantt Chart-**


Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	4	$4 - 0 = 4$	$4 - 4 = 0$
P2	15	$15 - 1 = 14$	$14 - 3 = 11$
P3	12	$12 - 2 = 10$	$10 - 1 = 9$
P4	9	$9 - 3 = 6$	$6 - 5 = 1$
P5	11	$11 - 4 = 7$	$7 - 2 = 5$

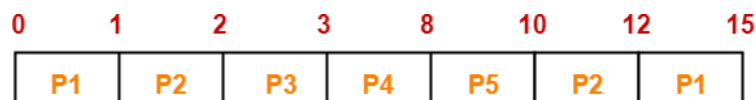
Now,

- Average Turn Around time = $(4 + 14 + 10 + 6 + 7) / 5 = 41 / 5 = 8.2$ unit
- Average waiting time = $(0 + 11 + 9 + 1 + 5) / 5 = 26 / 5 = 5.2$ unit

Problem-02: Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time	Priority
P1	0	4	2
P2	1	3	3
P3	2	1	4
P4	3	5	5
P5	4	2	5

If the CPU scheduling policy is priority preemptive, calculate the average waiting time and average turn around time. (Higher number represents higher priority).

**Solution-
Gantt Chart-**


Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	15	$15 - 0 = 15$	$15 - 4 = 11$
P2	12	$12 - 1 = 11$	$11 - 3 = 8$
P3	3	$3 - 2 = 1$	$1 - 1 = 0$
P4	8	$8 - 3 = 5$	$5 - 5 = 0$
P5	10	$10 - 4 = 6$	$6 - 2 = 4$

Now,

- Average Turn Around time = $(15 + 11 + 1 + 5 + 6) / 5 = 38 / 5 = 7.6$ unit
- Average waiting time = $(11 + 8 + 0 + 0 + 4) / 5 = 23 / 5 = 4.6$ unit

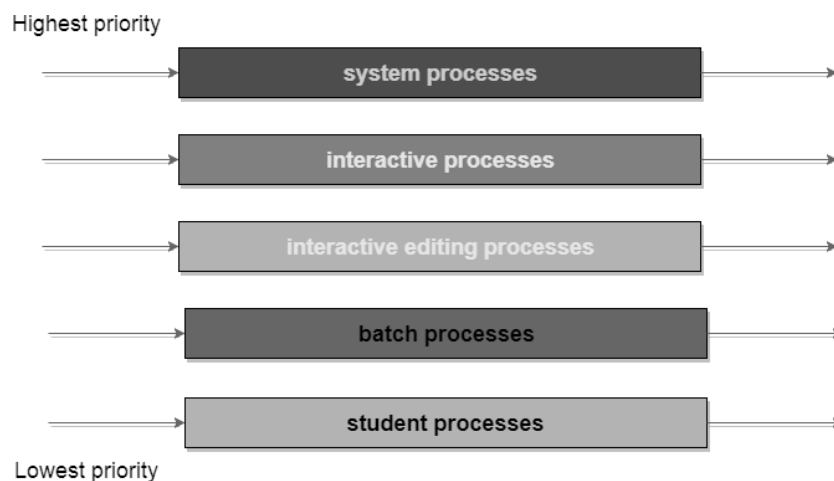
e. Multilevel Queue Scheduling

A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.

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

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

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



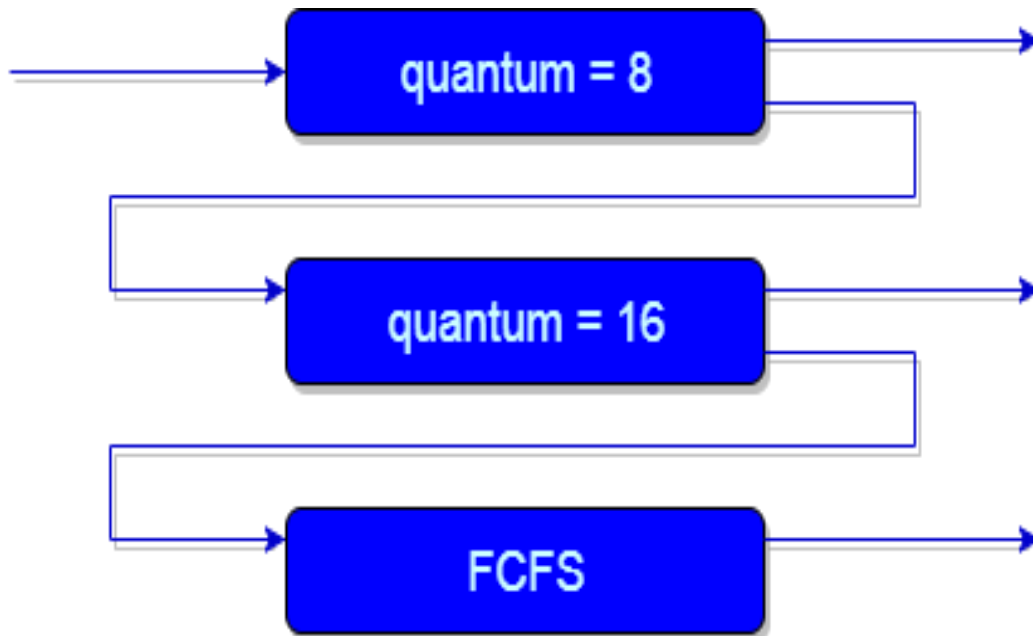
f. Multilevel Feedback Queue scheduling algorithm:

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. In general, a multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues.
- The scheduling algorithm for each queue.
- The method used to determine when to upgrade a process to a higher-priority queue.
- The method used to determine when to demote a process to a lower-priority

queue.

- The method used to determine which queue a process will enter when that process needs service.



Example of Multilevel Feedback Queues

- Three queues:
 - Q_0 – time quantum 8 milliseconds
 - Q_1 – time quantum 16 milliseconds
 - Q_2 – FCFS
- Scheduling
 - A new job enters queue Q_0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1 .
 - At Q_1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2 .

Need of Multilevel Feedback Queues

- This scheduling is more flexible than Multilevel queue scheduling.
- This algorithm helps in reducing the response time.
- In order to optimize the turnaround time, the SJF algorithm is needed which basically requires the running time of processes in order to schedule them. As we know that the running time of processes is not known in advance. Also, this scheduling mainly runs a process for a time quantum and after that, it can change the priority of the process if the process is long. Thus this scheduling algorithm mainly learns from the past behavior of the processes and then it can predict the future

behavior of the processes.

Advantages of Multilevel Feedback Queue Scheduling:

- It is more flexible.
- It allows different processes to move between different queues.
- It prevents starvation by moving a process that waits too long for the lower priority queue to the higher priority queue.

Disadvantages of Multilevel Feedback Queue Scheduling:

- The selection of the best scheduler, it requires some other means to select the values.
- It produces more CPU overheads.
- It is the most complex algorithm.