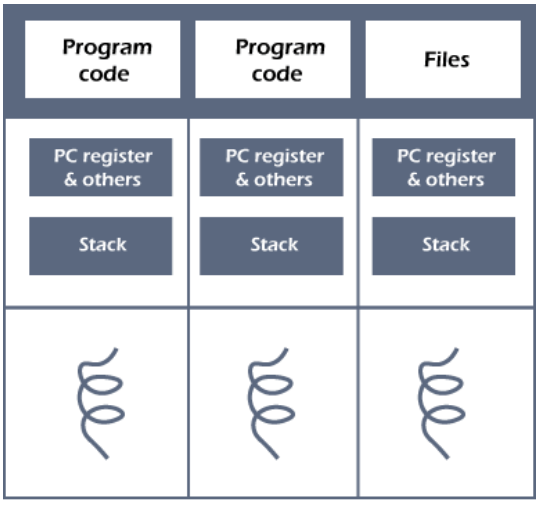
## Unit 2
## Multithreaded Programming:

### Threads in Operating System (OS)

- A thread is a single sequential flow of execution of tasks of a process so itis also known as thread of execution or thread of control.
- There is a way of thread execution inside the process of any operating system.
- Apart from this, there can be more than one thread inside a process.
- Each thread of the same process makes use of a separate program counter and a stack of activation records and control blocks.
- Thread is often referred to as a lightweight process.




Three threads of same process

- The process can be split down into so many threads.
- **For example**, in a browser, many tabs can be viewed as threads. MS Word uses many threads - formatting text from one thread, processing input from another thread, etc.

### Need of Thread:

- It takes far less time to create a new thread in an existing process than to create a new process.
- Threads can share the common data, they do not need to use Inter- Process communication.
- Context switching is faster when working with threads.
- It takes less time to terminate a thread than a process.

## Types of Threads

In the operating system there are two types of threads.

1. Kernel level thread.
2. User-level thread.

## User-level thread

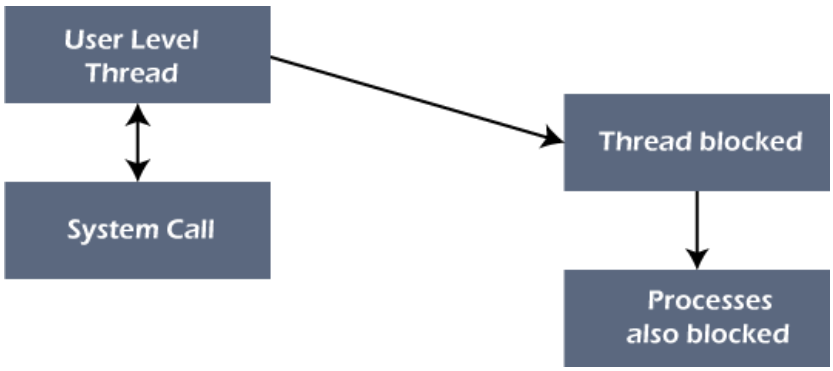
- The OS does not recognize the user-level thread.
- User threads can be easily implemented and it is implemented by the user.
- If a user performs a user-level thread blocking operation, the wholeprocess is blocked.
- The kernel level thread does not know nothing about the user levelthread.
- The kernel-level thread manages user-level threads as if they are single-threaded processes?
- Examples: Java thread, POSIX threads, etc.

## Advantages of User-level threads

1. The user threads can be easily implemented than the kernel thread.
2. User-level threads can be applied to such types of operating systems thatdo not support threads at the kernel-level.
3. It is faster and efficient.
4. Context switch time is shorter than the kernel-level threads.
5. It does not require modifications of the operating system.
6. User-level threads representation is very simple. The register, PC, stack, and mini thread control blocks are stored in the address space of the user-level process.
7. It is simple to create, switch, and synchronize threads without the intervention of the process.

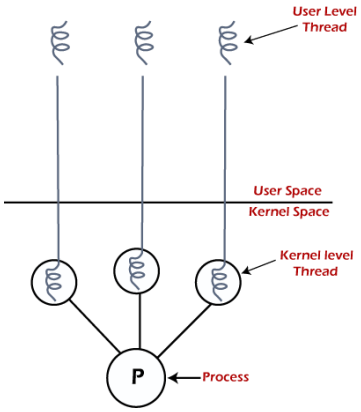
## Disadvantages of User-level threads

1. User-level threads lack coordination between the thread and the kernel.
2. If a thread causes a page fault, the entire process is blocked.



**Kernel level thread**

- The kernel thread recognizes the operating system.
- There is a thread control block and process control block in the systemfor each thread and process in the kernel-level thread.
- The kernel-level thread is implemented by the operating system.
- The kernel knows about all the threads and manages them.
- The kernel-level thread offers a system call to create and manage the threads from user-space.
- The implementation of kernel threads is more difficult than the userthread.
- Context switch time is longer in the kernel thread.
- If a kernel thread performs a blocking operation, the Banky threadexecution can continue. Example: Window Solaris.




**Advantages of Kernel-level threads**

1. The kernel-level thread is fully aware of all threads.
2. The scheduler may decide to spend more CPU time in the process of threads being large numerical.
3. The kernel-level thread is good for those applications that block the frequency.

**Disadvantages of Kernel-level threads**

1. The kernel thread manages and schedules all threads.
2. The implementation of kernel threads is difficult than the user thread.
3. The kernel-level thread is slower than user-level threads.

**Components of Threads**

Any thread has the following components.

1. Program counter
2. Register set
3. Stack space

**Benefits of Threads**

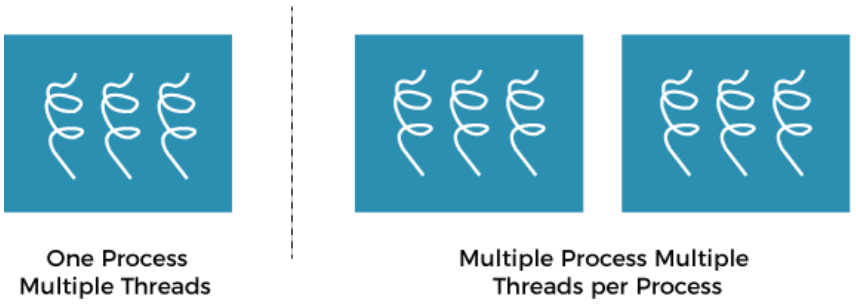
- **Enhanced throughput of the system:**
  - When the process is split into many threads and each thread is treated as a job, the number of jobs done in the unit time increases.
  - That is why the throughput of the system also increases.
- **Effective Utilization of Multiprocessor system:**
  - When you have more than one thread in one process, you canschedule more than one thread in more than one processor.
- **Faster context switch:**
  - The context switching period between threads is less than the process context switching. The process context switch means moreoverhead for the CPU.
- **Responsiveness:**
  - When the process is split into several threads, and when a thread completes its execution, that process can be responded to as soon as possible.
- **Communication:**
  - Multiple-thread communication is simple because the threads share the same address space, while in process; we adopt just a few exclusive communication strategies for communicationbetween two processes.
- **Resource sharing:**
  - Resources can be shared between all threads within a process, such as code, data, and files. Note: The stack and register cannot be shared between threads. There is a stack and register for each thread.

**Multithreading Definition:**

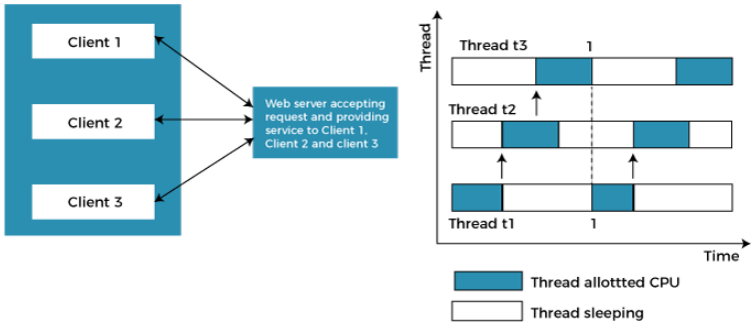
- Multithreading is the ability of a program or an operating system to enable more than one user at a time without requiring multiple copies of the program running on the computer.
- Multithreading can also handle multiple requests from the same user.

**Multithreading Models in Operating system**

- 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, or we can say that there is more than one thread to perform the task in multithreading.
- With the use of multithreading, multitasking can be achieved.



- The main drawback of single threading systems is that only one task can be performed at a time, so to overcome the drawback of this single threading, there is multithreading that allows multiple tasks to be performed.



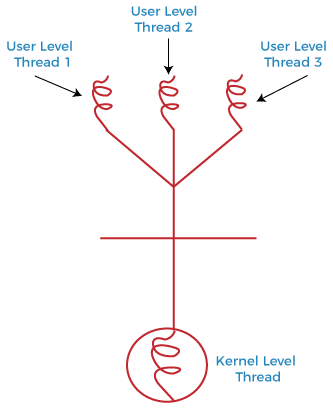

- In the above example, client1, client2, and client3 are accessing the web server without any waiting.
- In multithreading, several tasks can run at the same time.
- In an operating system threads are divided into the user-level thread and the Kernel-level thread.
- User-level threads handled independent form above the kernel and thereby managed without any kernel support.
- On the opposite hand, the operating system directly manages the kernel-level threads.
- Nevertheless, there must be a form of relationship between user-level and kernel-level threads.

**There exists three established multithreading models classifying theserelationships are:**

- Many to one multithreading model
- One to one multithreading model
- Many to Many multithreading models

**Many to one multithreading 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, easily implemented even on the simple kernel with no thread support.
- The disadvantage of this model is that since there is only one kernel-level thread schedule at any given time, this model cannot take advantage of the hardware acceleration offered by multithreaded processes or multi- processor systems.
- In this, all the thread management is done in the user space.
- If blocking comes, this model blocks the whole system.

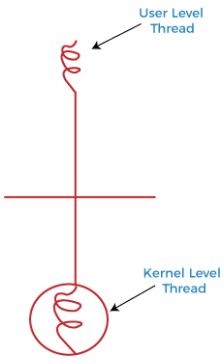


Many-to-one model

- In the above figure, the many to one model associates all user-level threads to single kernel-level threads.

**One to one multithreading model**

- The one-to-one model maps a single user-level thread to a single kernel-level thread.
- This type of relationship facilitates the running of multiple threads inparallel.
- However, this benefit comes with its drawback.
- The generation of every new user thread must include creating a corresponding kernel threads causing an overhead, which can hinder the performance of the parent process.
- Windows series and Linux operating systems try to tackle this problem by limiting the growth of the thread count.



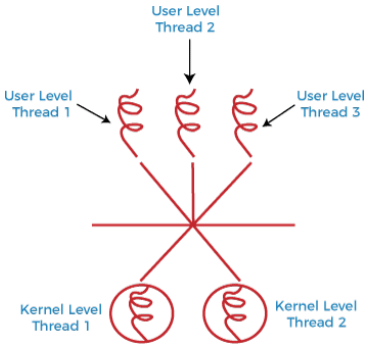
One-to-one model

- In the above figure, one model associates that one user-level thread to a single kernel-level thread.

**Many to Many Model multithreading model**

- In this type of model, there are several user-level threads and several kernel-level threads.

- The number of kernel threads created depends upon a particular application.
- The developer can create as many threads at both levels but may not bethe same.
- The many to many models are a compromise between the other twomodels.
- In this model, if any thread makes a blocking system call, the kernel can schedule another thread for execution.
- Also, with the introduction of multiple threads, complexity is not present as in the previous models.
- Though this model allows the creation of multiple kernel threads, true concurrency cannot be achieved by this model.
- This is because the kernel can schedule only one process at a time.

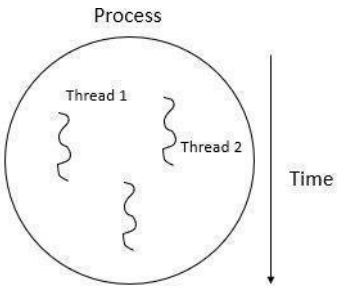



Many-to-Many model

- Many to many versions of the multithreading model associate several user-level threads to the same or much less variety of kernel-level threads in the above figure.

**Thread libraries:**

- A thread is a lightweight of process and is a basic unit of CPU utilization which consists of a program counter, a stack, and a set of registers.
- Given below is the structure of thread in a process –




- A process has a single thread of control where one program can counter and one sequence of instructions is carried out at any given time.

- Dividing an application or a program into multiple sequential threadsthat run in quasi-parallel, the programming model becomes simpler.
- Thread has the ability to share an address space and all of its data among themselves.
- This ability is essential for some specific applications.

- Threads are lighter weight than processes, but they are faster to createand destroy than processes.

**Thread Library**

- A thread library provides the programmer with an Application programinterface for creating and managing thread.

**Ways of implementing thread library**

- There are two primary ways of implementing thread library, which are as follows
  - The first approach is to provide a library entirely in user space with kernel support. All code and data structures for the library exist in a local function call in user space and not in a system call.
  - The second approach is to implement a kernel level library supported directly by the operating system. In this case the code and data structures for the library exist in kernel space.
- Invoking a function in the application program interface for the library typically results in a system call to the kernel.

The main thread libraries which are used are given below −

- **POSIX threads** –
  - Pthreads, the threads extension of the POSIX standard, may be provided as either a user level or a kernel level library.
- **WIN 32 thread** –
  - The windows thread library is a kernel level library available onwindows systems.
- **JAVA thread** –
  - The JAVA thread API allows threads to be created and managed directly as JAVA programs.

**Threading Issues:**

- The thread issues are as
  - The fork() and exec() system calls
  - Signal Handling
  - Cancellation
  - Thread Polls

**The fork () and exec () system calls**

- The fork() is used to create a duplicate process. The meaning of the fork()and exec() system calls change in a multithreaded program.
- If one thread in a program which calls fork(), does the new processduplicate all threads, or is the new process single-threaded?
- If we take, some UNIX systems have chosen to have two versions of fork(),

one that duplicates all threads and another that duplicates only the thread that invoked the fork() system call.
- If a thread calls the exec() system call, the program specified in the parameter to exec() will replace the entire process which includes all threads.

**Signal Handling**

- Generally, signal is used in UNIX systems to notify a process that a particular event has occurred.
- A signal received either synchronously or asynchronously, based on the source of and the reason for the event being signaled.
- All signals, whether synchronous or asynchronous, follow the same pattern as given below –
    - A signal is generated by the occurrence of a particular event.
    - The signal is delivered to a process.
    - Once delivered, the signal must be handled.

**Cancellation**

- Thread cancellation is the task of terminating a thread before it hascompleted.
- **For example** − If multiple database threads are concurrently searching through a database and one thread returns the result the remaining threads might be cancelled.
- A target thread is a thread that is to be cancelled, cancellation of target thread may occur in two different scenarios –
    - **Asynchronous cancellation** –
        - One thread immediately terminates the target thread.
    - **Deferred cancellation –**
        - The target thread periodically checks whether it should terminate, allowing it an opportunity to terminate itself in an ordinary fashion.

**Thread polls**

- Multithreading in a web server, whenever the server receives a request it creates a separate thread to service the request.
- Some of the problems that arise in creating a thread are as follows –
    - The amount of time required to create the thread prior to servingthe request together with the fact that this thread will be discarded once it has completed its work.
    - If all concurrent requests are allowed to be serviced in a new thread, there is no bound on the number of threads concurrently active in the system.
    - Unlimited thread could exhaust system resources like CPU time or memory.
    - A thread pool is to create a number of threads at process start-up and place them into a pool, where they sit and wait for work.

## CPU Scheduling

### CPU Scheduling

- The CPU Scheduling is the process by which a process is executed by the using the resources of the CPU.
- The process also can wait due to the absence or unavailability of the resources.
- These processes make the complete use of Central Processing Unit. The operating system must choose one of the processes in the list of ready-to-launch processes whenever the CPU gets idle.
- A temporary (CPU) scheduler does the selection.
- The Scheduler chooses one of the ready-to-start memory processes to getthe CPU.

### CPU Scheduling Criteria

- Different CPU scheduling algorithms have different properties and thechoice of a particular algorithm depends on various factors.
- Many criteria have been suggested for comparing CPU scheduling algorithms.
- The criteria include the following:

1. **CPU utilization:**
    a. The main objective of any CPU scheduling algorithm is to keepthe CPU as busy as possible.
    b. Theoretically, CPU utilization can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the load upon the system.
2. **Throughput:**
    a. A measure of the work done by the CPU is the number of processes being executed and completed per unit of time.
    b. This is called throughput.
    c. The throughput may vary depending on the length or duration of the processes.
3. **Turnaround time:**
    a. For a particular process, an important criterion is how long it takes to execute that process.
    b. The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time.
    c. Turn-around time is the sum of times spent waiting to get into memory, waiting in the ready queue, executing in CPU, and waiting for I/O.
    d. The formula to calculate Turnaround Time = Compilation Time – Arrival Time.
4. **Waiting time:**
    a. A scheduling algorithm does not affect the time required to complete the process once it starts execution.
    b. It only affects the waiting time of a process i.e. time spent by a process waiting in the ready queue.
    c. The formula for calculating Waiting Time = Turnaround Time – Burst Time.

5. **Response time:**
    a. In an interactive system, turn-around time is not the best criterion.
    b. A process may produce some output fairly early and continue computing new results while previous results are being output to the user.
    c. Thus another criterion is the time taken from submission of the process of the request until the first response is produced.
    d. This measure is called response time.
    e. The formula to calculate Response Time = CPU Allocation Time(when the CPU was allocated for the first) – Arrival Time
6. **Completion time:**
    a. The completion time is the time when the process stops executing, which means that the process has completed its burst time and is completely executed.
7. **Priority:**
    a. If the operating system assigns priorities to processes, the scheduling mechanism should favor the higher-priority processes.
8. **Predictability:**
    a. A given process always should run in about the same amount of time under a similar system load.

**Modes in CPU Scheduling Algorithms**

- There are two modes in CPU Scheduling Algorithms. They are:
    - Non Pre-emptive Approach
    - Pre-emptive Approach

- In Non-Preemptive-Approach the process once stars its execution, then the CPU is not allotted to the same process until the completion of process. There is a shift of Processes by the Central Processing Unit. The complete CPU is allocated to the Process when only certain required conditions are achieved and there will be change of CPU allocation if there is break or false occurrence in the required conditions.

- In Preemptive-Approach the process once starts its execution then the CPU is allotted to the same process until the completion of process. There would be no shift of Processes by the Central Processing Unit. The complete CPU is allocated to the Process and there would be no change of CPU allocation until the process is complete.

**1. Non Pre-emptive Approach:**

I. FCFS(First Come First Serve)
II. SJF (Shortest Job First)
III. Priority

## 1. FCFS (First Come First Serve) Scheduling Algorithm:

**FCFS process and examples**

The process that requests the CPU first is allocated the CPU first

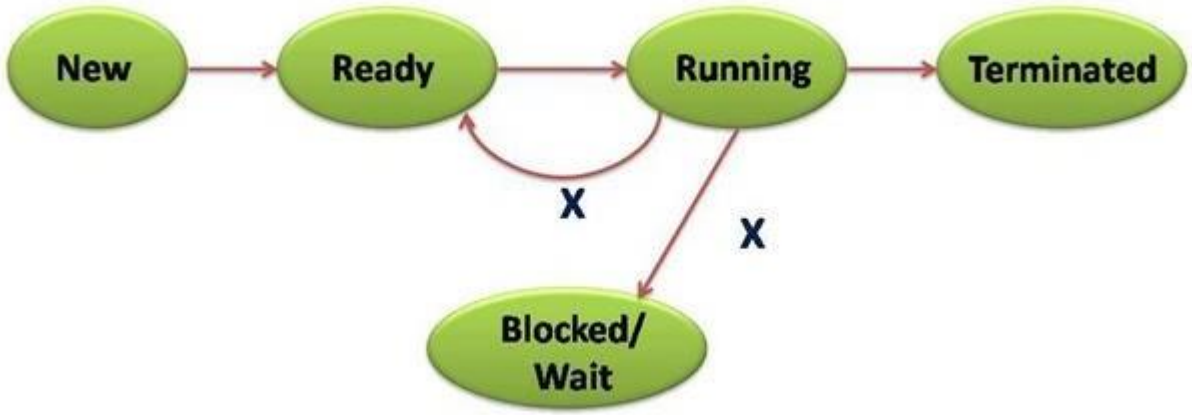
**Criteria: Arrival Time**

**Decision mode: Non-preemptive**

**Data-Structure: Queue**

**TAT = CT – AT**

**WT = TAT - BT**




**Example:**

| Process No. | Arrival Time (AT) | Burst Time (BT) |
|---|---|---|
| 1 | 0 | 4 |
| 2 | 1 | 3 |
| 3 | 2 | 1 |
| 4 | 3 | 2 |
| 5 | 4 | 5 |

First we design a **Gantt Chart** according to arrival time and burst time to follow FCFS.


Now calculate **Completion Time (CT)** from Gantt Chart

**Turn Around Time (TAT)= Completion Time (CT) – Arrival Time (AT)**

**TAT**: $P_1 = 4 - 0 = 4, P_2 = 7 - 1 = 6, P_3 = 8 - 2 = 6$

$P_4 = 10 - 3 = 7, P_5 = 15 - 4 = 11$

We can also calculate **Wait Time (WT)** using the formula:

**WT = TAT - BT**

$P_1 = 4 - 4 = 0, \quad P_2 = 6 - 3 = 3, \quad P_3 = 6 - 1 = 5$

$P_4 = 7 - 2 = 5, P_5 = 11 - 5 = 6$

| Process No. | AT | BT | CT | TAT | WT | Response Time (RT) |
|---|---|---|---|---|---|---|
| 1 | 0 | 4 | 4 | 4 | 0 | 0 |
| 2 | 1 | 3 | 7 | 6 | 3 | 3 |
| 3 | 2 | 1 | 8 | 6 | 5 | 5 |
| 4 | 3 | 2 | 10 | 7 | 5 | 5 |
| 5 | 4 | 5 | 15 | 11 | 6 | 6 |

**In the case of Non-Preemptive waiting time and first Response Time (RT) is same.**

**AVG Turn Around Time (ATAT) = $\frac{4+6+6+7+11}{5}$ = 34/5 = 6.8 m/s**

**AVG Waiting Time (AWT) = $\frac{0+3+5+5+6}{5}$ = 19/5 = 3.16 m/s**

## 2. SJF (Shortest Job First) Scheduling Algorithm:

**SJF process and examples:**

This algorithm associate with each process the length of the process's next CPU burst. When the CPU is available, it is assigned to the process that has the **smallest next CPU burst**. Thus, a request remains pending until all shorter requests have been serviced. If the next CPU bursts of two processes are the same, **FCFS scheduling** is used to break the tie.

**Criteria - Burst time**

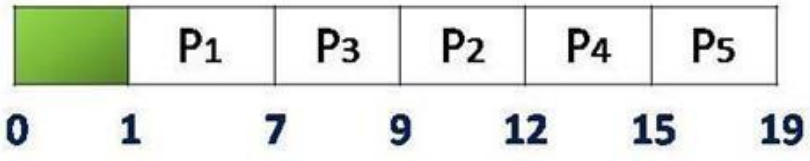
**Mode - Non-preemptive**

**Data-structure - Min Heap** is the efficient data structure of SJF.

**Example:**

| Process No. | Arrival Time (AT) | Burst Time (BT) |
|---|---|---|
| 1 | 1 | 6 |
| 2 | 2 | 3 |
| 3 | 3 | 2 |
| 4 | 4 | 3 |
| 5 | 5 | 4 |

## Gantt chart:




**TAT = CT - AT and WT = TAT - BT**

| Process No. | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|
| 1 | 1 | 6 | 7 | (7 – 1) = 6 | (6 – 6) = 0 |
| 2 | 2 | 3 | 12 | (12 – 2) = 10 | (10 – 3) = 7 |
| 3 | 3 | 2 | 9 | (9 – 3) = 6 | (6 – 2) = 4 |
| 4 | 4 | 3 | 15 | (15 – 4) = 11 | (11 – 3) = 8 |
| 5 | 5 | 4 | 19 | (19 – 5) = 14 | (14 – 4) = 10 |

SJF also suffers from the **convoy effect** i.e. if the larger process comes earlier then **average waiting time** will be increased.

**AVG Turn Around Time (ATAT) = $\frac{6+10+6+11+14}{5}$ = 47/5 = 9.4 m/s**

**AVG Waiting Time (AWT) = $\frac{0+7+4+8+10}{5}$ = 29/5 = 5.8 m/s**

## 3. Priority Scheduling Algorithm:

A **priority** is associated with each process, and the CPU is allocated to the process with the highest priority.
Priority may be static or dynamic.

**Static priority:** It doesn't change priority throughout the execution of the process
**Dynamic priority:** In this dynamic priority, priority can be changed by the scheduler at a regular interval of time.

An **SJF algorithm** is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst, the larger the CPU burst, the lower the priority, and vice versa.

Some systems use low numbers to represent low priority; others use low numbers for high priority. This difference can lead to confusion but **here we assume that low numbers represent low priority**.

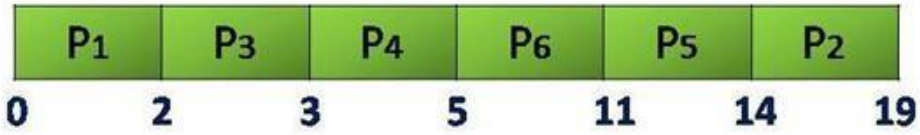
**Criteria: Priority**

**Mode: Non preemptive**

**Example 1: (Non-preemptive priority scheduling): Lowest number has low priority**

| Process No. | Priority | Arrival Time (AT) | Burst Time (BT) |
|---|---|---|---|
| $P_1$ | 2 | 0 | 2 |
| $P_2$ | 4 | 1 | 5 |
| $P_3$ | 6 | 2 | 1 |
| $P_4$ | 10 | 3 | 2 |
| $P_5$ | 8 | 4 | 3 |
| $P_6$ | 12 | 5 | 6 |

**Gantt Chart:**




Now we calculate **Turn Around Time (TAT)** and **Waiting Time (WT)** using the following formula:

**TAT = CT - AT, WT = TAT – BT**

**And**

**Response Time (RT) = FR (First Response) - AR (Arrival Time)**

| P_No. | Priority | AT | BT | CT | TAT | WT | First Response (FR) | RT |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 2 | 2 | (2 – 0) = 2 | (2 – 2) = 0 | 0 | (0 – 0) = 0 |
| 2 | 4 | 1 | 5 | 19 | (19 – 1) = 18 | (18 – 5) = 13 | 14 | (14 – 1) = 13 |
| 3 | 6 | 2 | 1 | 3 | (3 – 2) = 1 | (1 – 1) = 0 | 2 | (2 – 2) = 0 |
| 4 | 10 | 3 | 2 | 5 | (5 – 3) = 2 | (2 – 2) = 0 | 3 | (3 – 3) = 0 |
| 5 | 8 | 4 | 3 | 14 | (14 – 4) = 10 | (10 – 3) = 7 | 11 | (11 – 4) = 7 |
| 6 | 12 | 5 | 6 | 11 | (11 – 5) = 6 | (6 – 6) = 0 | 5 | (5 – 5) = 0 |

**Note: Here waiting time and response time same because it is non-preemptive. For every non-preemptive scheduling waiting time = response time.**

**Avg Turn Around Time (ATAT)** $= (2 + 18 + 1 + 2 + 10 + 6)/6$
$= 6.5$

**Average Waiting Time (AWT)** $= (13 + 7)/6 = 3.33$

## 2. Pre-emptive Approach:

1 SJF/SRTF (Shortest Reaming Time First)
2 Round Robin
3 Priority

### 1. SJF/SRTF (Shortest Reaming Time First) Scheduling Algorithm:

**SRTF process and examples:**

The shortest remaining time First (SRTF) algorithm is **preemptive** version of **SJF**. In this algorithm, the scheduler always chooses the processes that have the shortest expected remaining processing time. When a new process joins the ready queue it may in fact have a shorter remaining time than the currently running process. Accordingly, the scheduler may preempt the current process when a new process (with a shorter burst time) becomes ready.
This feature helps to improve the **turnaround times** and **weighted turnarounds of processes**.

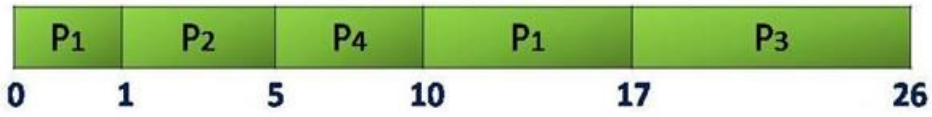
After every one unit of processing current process, scheduler can check any processes available at ready queue with shorter burst time.

**Criteria: BT (Burst Time) + AT (Arrival Time)**
**Mode: Preemptive**

**Example:**

| Process No. | Arrival Time (AT) | Burst Time (BT) |
|---|---|---|
| $P_1$ | 0 | 8 |
| $P_2$ | 1 | 4 |
| $P_3$ | 2 | 9 |
| $P_4$ | 3 | 5 |

**Gantt Chart**




Process P1 is started at time 0, since it is the only process in the queue. Process P2 arrives at time 1. The remaining time for process P1 (7 milliseconds) is larger than the time required by process P2 (4 milliseconds), so process P1 is preempted, and process P2 is scheduled.

**TAT = CT - AT, WT = TAT – BT**
**And**
**Response Time (RT) = FR (First Response) – AR (Arrival Time)**

| P_No. | AT | BT | CT | TAT | WT | First Response (FR) | RT |
|---|---|---|---|---|---|---|---|
| P1 | 0 | 8 | 17 | (17 – 0) = 17 | (17 – 8) = 9 | 0 | (0 – 0) = 0 |
| P2 | 1 | 4 | 5 | (5 – 1) = 4 | (4 – 4) = 0 | 1 | (1 – 1) = 0 |
| P3 | 2 | 9 | 26 | (26 – 2) = 24 | (24 – 9) = 15 | 17 | (17 – 2) = 15 |
| P4 | 3 | 5 | 10 | (10 – 3) = 7 | (7 – 5) = 2 | 5 | (5 – 3) = 2 |

$$\textbf{AVG Turn Around Time (ATAT)} = \frac{17 + 4 + 24 + 7}{4} = 13$$

$$\textbf{AVG Waiting Time (AWT)} = \frac{9 + 0 + 15 + 2}{4} = 6.5$$

## 2. Round Robin Scheduling Algorithm:

The round-robin (RR) scheduling algorithm is designed especially for timesharing systems and Interactive systems. It is similar to FCFS scheduling, but preemption is added to enable the system to switch between processes. A small unit of time, called a time quantum or time slice, is defined.

A time quantum is generally from 10 to 100 milliseconds in length. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval.

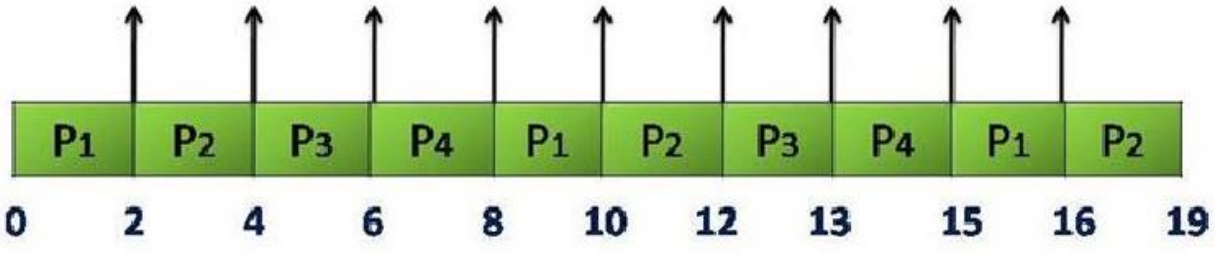
Round Robin Scheduling is very much practical and there is no starvation (no convoy effect) because every process gets CPU for a certain amount of time unit/quantum.

Criteria: TQ (Time Quantum) + AT (Arrival Time)
Mode: Preemptive
Example 1: TQ = 2 (Here)

| Process No. | Arrival Time (AT) | Burst Time (BT) |
|---|---|---|
| P1 | 0 | 5 |
| P2 | 1 | 7 |
| P3 | 2 | 3 |
| P4 | 3 | 4 |

## Gantt Chart

**Context Switching**




Number of Context Switching = **09**

Now we calculate **Turn Around Time (TAT)** and **Waiting Time (WT)** using the following formula:

**TAT = CT - AT, WT = TAT – BT**

**And**

**Response Time (RT) = FR (First Response) - AR (Arrival Time)**

| P_No. | AT | BT | CT | TAT | WT | First Response (FR) | Response Time (RT) |
|---|---|---|---|---|---|---|---|
| P1 | 0 | 5 3 1 | 16 | (16 – 0) = 16 | (16 – 5) = 11 | 0 | (0 – 0) = 0 |
| P2 | 1 | 7 5 3 | 19 | (19 – 1) = 18 | (18 – 7) = 11 | 2 | (2 – 1) = 1 |
| P3 | 2 | 3 2 1 | 13 | (13 – 2) = 11 | (11 – 3) = 8 | 4 | (4 – 2) = 2 |
| P4 | 3 | 4 2 2 | 15 | (15 – 3) = 12 | (12 – 4) = 8 | 6 | (6 – 3) = 3 |

**AVG Turn Around Time (ATAT) = $\frac{16+18+11+12}{4}$ = 57/4 = 14.25m/s**

**AVG Waiting Time (AWT) = $\frac{11+11+8+8}{4}$ = 38/4 =14.25m/s**

## 3. Priority Scheduling Algorithm:

A **priority** is associated with each process, and the CPU is allocated to the process with the highest priority.

Priority may be static or dynamic.

**Static priority:** It doesn't change priority throughout the execution of the process

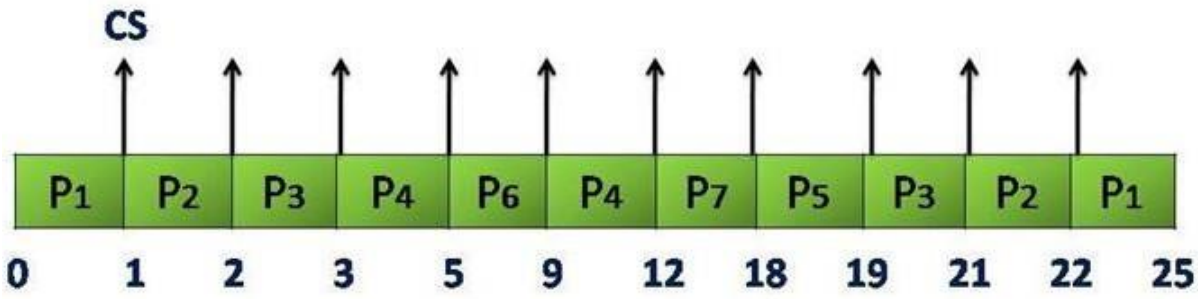
**Dynamic priority:** In this dynamic priority, priority can be changed by the scheduler at a regular interval of time.

An **SJF algorithm** is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst, the larger the CPU burst, the lower the priority, and vice versa.

Some systems use low numbers to represent low priority; others use low numbers for high priority. This difference can lead to confusion but **here we assume that low numbers represent low priority**.

| Process No. | Priority | Arrival Time (AT) | Burst Time (BT) |
|---|---|---|---|
| $P_1$ | 2 | 0 | 4 |
| $P_2$ | 4 | 1 | 2 |
| $P_3$ | 6 | 2 | 3 |
| $P_4$ | 10 | 3 | 5 |
| $P_5$ | 8 | 4 | 1 |
| $P_6$ | 12 | 5 | 4 |
| $P_7$ | 9 | 6 | 6 |

## Gantt Chart:




**Number of Context Switching (CS) = 10**

Now we calculate **Turn Around Time (TAT)** and **Waiting Time (WT)** using the following formula:

TAT = CT - AT, WT = TAT – BT

And

Response Time (RT) = FR (First Response) - AR (Arrival Time)

| P_No. | Priority | AT | BT | CT | TAT | WT | First Response (FR) | RT |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 4 | 25 | (25 – 0) = 25 | (25 – 4) = 21 | 0 | (0 – 0) = 0 |
| 2 | 4 | 1 | 2 | 22 | (22 – 1) = 21 | (21 – 2) = 19 | 1 | (1 – 1) = 0 |
| 3 | 6 | 2 | 3 | 21 | (21 – 2) = 19 | (19 – 3) = 16 | 2 | (2 – 2) = 0 |
| 4 | 10 | 3 | 5 | 12 | (12 – 3) = 9 | (9 – 5) = 4 | 3 | (3 – 3) = 0 |
| 5 | 8 | 4 | 1 | 19 | (19 – 4) = 15 | (15 – 1) = 14 | 18 | (18 – 4) = 14 |
| 6 | 12 | 5 | 4 | 9 | (9 – 5) = 4 | (4 – 4) = 0 | 5 | (5 - 5) = 0 |
| 7 | 9 | 6 | 6 | 18 | (18 – 6) = 12 | (12 – 6) = 6 | 12 | (12 – 6) = 6 |

**Avg Turn Around Time (ATAT)**

$$= (25 + 21 + 19 + 9 + 15 + 4 + 12)/7 = 105/7 = 15$$

**Avg Waiting Time (AWT)** $= (21 + 19 + 16 + 4 + 14 + 0 + 6)/7$

$$= 80/7 = 11.428$$

**Avg Response Time (ART)** $= (14 + 6)/7 = 20/7 = 2.85$

# Multiple Processors Scheduling/ Multiprocessor Scheduling

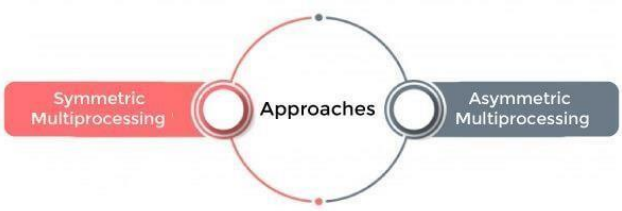
- **Multiple processor scheduling** or multiprocessor scheduling focuses on designing the system's scheduling function, which consists of more than one processor.
- Multiple CPUs share the load (load sharing) in multiprocessor scheduling so that various processes run simultaneously.
- In general, multiprocessor scheduling is complex as compared to single processor scheduling.
- In the multiprocessor scheduling, there are many processors, and they are identical, and we can run any process at any time.
- The multiple CPUs in the system are in close communication, which shares a common bus, memory, and other peripheral devices.
- So we can say that the system is tightly coupled.
- These systems are used when we want to process a bulk amount of data, and

these systems are mainly used in satellite, weather forecasting, etc.

- There are cases when the processors are identical, i.e., homogenous, in terms of their functionality in multiple-processor scheduling. We can use any processor available to run any process in the queue.
- Multiprocessor systems may be ***heterogeneous*** (different kinds of CPUs) or ***homogenous*** (the same CPU).
- There may be special scheduling constraints, such as devices connected via a private bus to only one CPU.
- There is no policy or rule which can be declared as the best scheduling solution to a system with a single processor. Similarly, there is no best scheduling solution for a system with multiple processors as well.

**Approaches to Multiple Processor Scheduling**

- There are two approaches to multiple processors scheduling in the operating system: Symmetric Multiprocessing and AsymmetricMultiprocessing.




1. **Symmetric Multiprocessing:**
   a. It is used where each processor is ***self-scheduling***.
   b. All processes may be in a common ready queue, or each processor may have its private queue for ready processes.
   c. The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.
2. **Asymmetric Multiprocessing:**
   a. It is used when all the scheduling decisions and I/O processing are handled by a single processor called the ***Master Server***.
   b. The other processors execute only the ***user code***.
   c. This is simple and reduces the need for data sharing, and thisentire scenario is called Asymmetric Multiprocessing.

## Real-Time CPU Scheduling

- CPU scheduling for real-time operating systems involves special issues. In general, we can distinguish between *soft real-time* systems and *hard real-time* systems.
- Soft real-time systems provide no guarantee as to when a critical real- time process will be scheduled.
- They guarantee only that the process will be given preference over noncritical processes.
- Hard real-time systems have stricter requirements. A task must be serviced by its deadline; service after the deadline has expired is the same as no service at all.
- **Minimizing Latency**

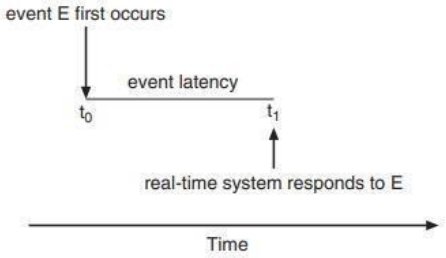



**Figure 5.17** Event latency.

- Consider the event-driven nature of a real-time system.
- The system is typically waiting for an event in real time to occur.
- Events may arise in software - as when a timer expires - or in hardware - as when a remote-controlled vehicle detects that it is approaching an obstruction.
- When an event occurs, the system must respond to and service it as quickly as possible.
- We refer to event latency as the amount of time that elapses from whenan event occurs to when it is serviced
- Usually, different events have different latency requirements.
- For example, the latency requirement for an antilock brake system might be 3 to 5 milliseconds.
- That is, from the time a wheel first detects that it is sliding, the system controlling the antilock brakes has 3 to 5 milliseconds to respond to and control the situation.
- Any response that takes longer might result in the automobile's veering out of control.
- In contrast, embedded system controlling radar in an airliner mighttolerate a latency period of several seconds.

Two types of latencies affect the performance of real-time systems:
1. Interrupt latency
2. Dispatch latency

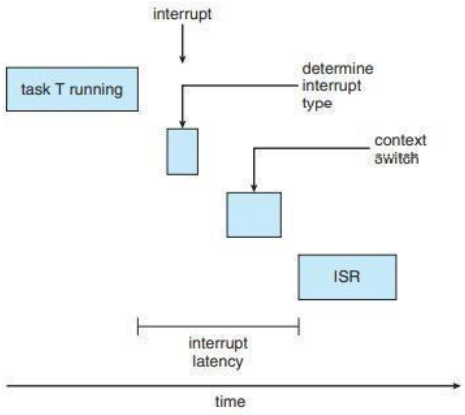



**Figure 5.18** Interrupt latency.

- Interrupt latency refers to the period of time from the arrival of an interrupt at the CPU to the start of the routine that services the interrupt.
- When an interrupt occurs, the operating system must first complete the instruction it is executing and determine the type of interrupt that occurred.
- It must then save the state of the current process before servicing the interrupt using the specific interrupt service routine (ISR).
- The total time required to perform these tasks is the interrupt latency (Figure 5.18).
- Obviously, it is crucial for real-time operating systems to minimize interrupt latency to ensure that real-time tasks receive immediate attention.
- Indeed, for hard real-time systems, interrupt latency must not simply be minimized; it must be bounded to meet the strict requirements of these systems.
- One important factor contributing to interrupt latency is the amount of time interrupts may be disabled while kernel data structures are being updated.
- Real-time operating systems require that interrupts be disabled for onlyvery short periods of time.
- The amount of time required for the scheduling dispatcher to stop one process and start another is known as dispatch latency.
- Providing real-time tasks with immediate access to the CPU mandates that real-time operating systems minimize this latency as well.
- The most effective technique for keeping dispatch latency low is to provide preemptive kernels.
- For hard real-time systems, dispatch latency is typically measured in several microseconds.

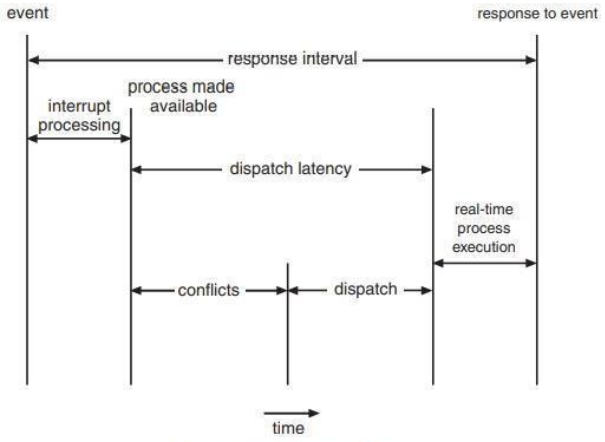



**Figure 5.19** Dispatch latency.

In Figure 5.19, we diagram the makeup of dispatch latency. The *conflictphase* of dispatch latency has two components:

1. Preemption of any process running in the kernel
2. Release by low-priority processes of resources needed by a high-priority process

Following the conflict phase, the dispatch phase schedules the high-priority process onto an available CPU.

# Assignment Questions

## 2 Marks:

1. What is thread? Difference between user level and kernel level thread.
2. Define turnaround time.
3. Define waiting time.
4. Define burst time.
5. What is response time?
6. Define schedulers list its type.
7. Define multithreading.
8. List the thread libraries.
9. What is CPU scheduling and list its type.
10. What is multi-processor? List its approaches.
11. What is real time CPU Scheduling?
12. What is thread scheduling?
13. List the thread issues.

## 5 Marks:

1. List out differentiate between process and thread.
2. What are the two types of thread and list the difference between both of them.
3. What is multithreading? List its advantages and explain.
4. Explain multithreading models
5. Explain CPU scheduling criteria.
6. What is multi-processor system and explain.
7. Explain real time CPU scheduling.
8. Explain thread scheduling.

## 10 Marks:

**1. Consider the following set of processes with the length of burst time given in milliseconds**

| Process | burst time |
|---|---|
| P1 | 3 |
| P2 | 1 |
| P3 | 3 |
| P4 | 4 |
| P5 | 2 |

i) Draw Gantt charts illustrating the execution of these processes Using FCFS And Round Robin(Quantum time=1 milliseconds) scheduling method.

ii) Calculate the average waiting time and average turnaround time for both of these scheduling method.

**2. Consider the following set of processes with the length of burst time & arrival time given in milliseconds**

| Process | Arrival time | Burst time | Priority |
|---|---|---|---|
| P1 | 0 | 8 | 4 |
| P2 | 0 | 9 | 1 |
| P3 | 1 | 4 | 2 |
| P4 | 2 | 3 | 3 |

**(Note: Lower number indicates higher priority)**

i) Draw Gantt charts illustrating the execution of these processes Using pre-emptive SJF And Non - preemptive Priority scheduling method.

ii) Calculate the average waiting time and average turnaround time for both of these scheduling method.

**3. Consider the following set of processes with the length of burst time & arrival time given in milliseconds**

| Process | Arrival time | Burst time | Priority | **(Note: Higher number indicates higher priority)** |
|---|---|---|---|---|
| P1 | 0 | 10 | 3 | |
| P2 | 0 | 1 | 1 | |
| P3 | 1 | 2 | 4 | |
| P4 | 2 | 1 | 5 | |
| P5 | 3 | 5 | 2 | |

i) Draw Gantt charts illustrating the execution of these processes Using non pre-emptive SJF And pre-emptive Priority scheduling method.

ii) Calculate the average waiting time and average turnaround time for both of these scheduling method.

# Thank You

# *****All the Best*****