Name : Ryshikesh Kazbhazi Palve
Roll No. : 31258
Subject :

Date : / /20 | Page No :



# Assignment No. 1

DOP : 27-07-2021 DOS : 12-08-2021

## Problem Definition :

Write a program in C/C++/Java/Python to simulate CPU scheduling Algorithms : FCFS, SJF (Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive).

## Learning objectives :

① To implement the simulation of CPU scheduling algorithms.
② To choose a rescheduling algorithm which will make the system efficient, fast and fair.
③ To analyze different scheduling algorithms.

## Learning Outcomes :

After completion of assignment, students will be able to -
① Implement CPU scheduling Algorithms.
② Evaluate performance of different scheduling algorithms.

## Concepts related theory :

### CPU scheduling in Operating Systems

Scheduling of process/work is done to finish the work on time.

**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.

$$\text{Turn Around Time} = \text{Completion Time} - \text{Arrival Time}$$

**Waiting Time :** Time Difference between turn around time and burst time.

$$\text{Waiting Time} = \text{Turn Around Time} - \text{Burst Time}$$

### Objectives of process scheduling :

1. Maximize CPU utilization.
2. Fair allocation of CPU.
3. Maximize throughput (Number of processes that complete their execution per unit time.)

④ Minimize Turn Around Time (Time taken by a process to finish execution).
⑤ Minimize Waiting Time (Time a process waits in ready queue).
⑥ Minimize response time (Time when a process produces first response).

## Different Scheduling Algorithms :-

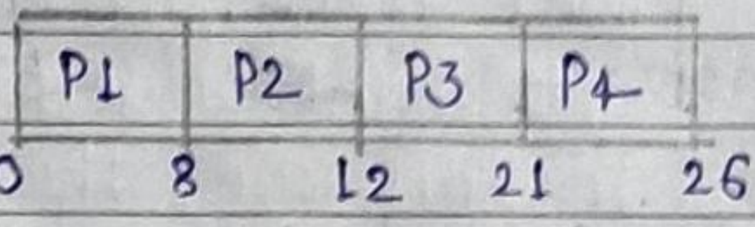
### 1. First Come First Serve (FCFS):

Simplest scheduling algorithm that schedules according to arrival times of processes. First come first serve scheduling algorithm states that the process that requests the CPU first is allocated the CPU first. It is implemented by using the FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue. FCFS is non-preemptive scheduling algorithm.

Example of FCFS scheduling Algorithm :

| Process Id | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time |
|---|---|---|---|---|---|
| 1 | 0 | 8 | 8 | 8 | 0 |
| 2 | 1 | 4 | 12 | 11 | 7 |
| 3 | 2 | 9 | 21 | 19 | 10 |
| 4 | 3 | 5 | 26 | 23 | 18 |

Gant chart :




Average Waiting Time = 8.75
Average Turn Around Time = 15.25

## 2.] Shortest Job First (SJF) :

### a) Non-Preemptive :

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.

### b) Preemptive :

In this scheduling, the scheduler always select a process for execution with the shortest expected remaining processing time.

⇒ A process is preempted after the time slab.

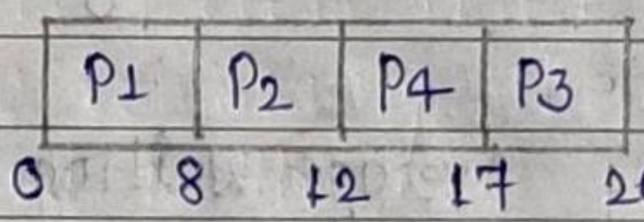
⇒ The next process to be scheduled by for execution is based on shortest expected remaining processing time.

## Example of SJF scheduling Algorithm :

### a) Non-Preemptive :

| Process Id | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time |
|---|---|---|---|---|---|
| 1 | 0 | 8 | 8 | 8 | 0 |
| 2 | 2 | 1 | 4 | 12 | 11 | 7 |
| 3 | 2 | 9 | 26 | 24 | 15 |
| 4 | 3 | 5 | 17 | 14 | 9 |

Gant chart:



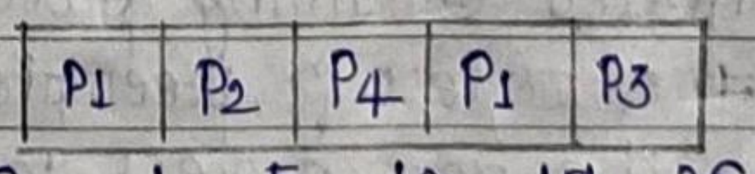

Average Waiting Time : 7.75

Average Turn Around Time : 14.25

### b) Preemptive :

| Process Id | Arrival Time | Burst Time | Completion Time | Turn Around Time | Waiting Time |
|---|---|---|---|---|---|
| 1 | 0 | 8 | 17 | 17 | 9 |
| 2 | 1 | 4 | 5 | 4 | 0 |
| 3 | 2 | 9 | 26 | 24 | 15 |
| 4 | 3 | 5 | 10 | 7 | 2 |

Gant chart:




Average Waiting Time : 6.5

Average Turn Around Time : 13.0

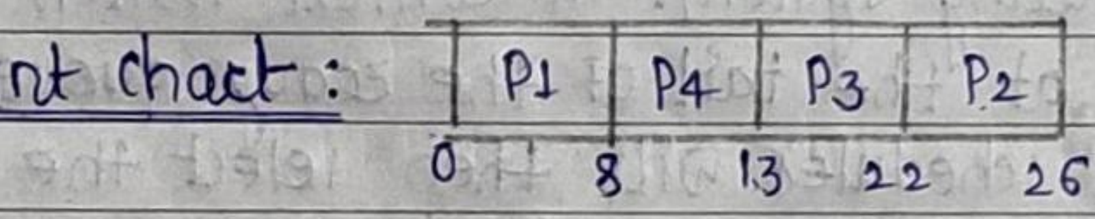
## 3.] Priority Based Scheduling (Non-preemptive):

In this scheduling, processes are scheduled according to their priorities, i.e. highest priority process is scheduled first. If priorities of two processes match, then schedule according to arrival time. Here starvation of process is possible.

### Example of priority Based scheduling Algorithm:

| Process Id | priority | Arrival Time | Burst Time | Completion Time | Turn Around Time | Waiting Time |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 8 | 8 | 8 | 0 |
| 2 | 2 | 1 | 4 | 26 | 25 | 21 |
| 3 | 3 | 2 | 9 | 22 | 20 | 11 |
| 4 | 4 | 3 | 5 | 13 | 10 | 5 |

Gant chart:




Average Waiting Time : ~~12.25~~ 9.25

Average Turn Around Time : ~~17.25~~ 15.75

## 4.] Round Robin scheduling Algorithm : (Preemptive)

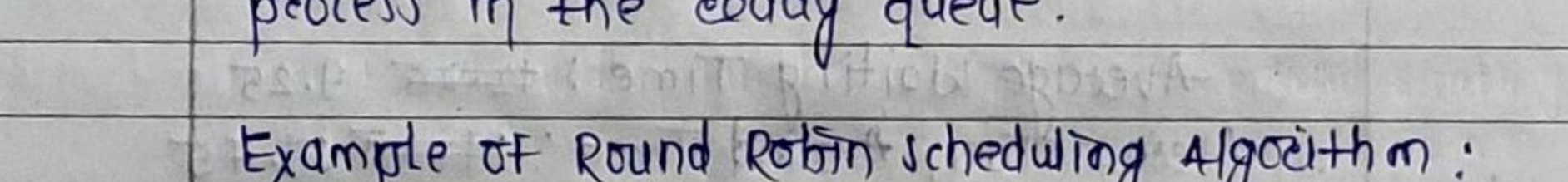
Each process is assigned a fixed time (Time Quantum / Time slice) in cyclic way. It is designed especially for the time-sharing system. The ready queue is treated as a circular queue. The cpu scheduler goes around the ready queue, allocating the cpu to each process for a time

Interval of up to 1-time quantum. To implement Round Robin scheduling, we keep the ready queue as a FIFO queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1-time quantum and dispatches the process. One of two things will then happen. The process may have a CPU burst of less than 1-time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running process is longer than 1-time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue.

Example of Round Robin scheduling Algorithm :

| Process Id | Arrival Time | Burst Time | completion Time | Turn Around Time | Waiting Time |
|---|---|---|---|---|---|
| 1 | 0 | 8 | 22 | 22 | 14 |
| 2 | 1 | 4 | 12 | 11 | 7 |
| 3 | 2 | 9 | 26 | 24 | 15 |
| 4 | 3 | 5 | 23 | 20 | 15 |

Gant chart :



Average Waiting Time : 12.25
Average Turn Around Time : 17.25

Algorithm :

1) FCFS Algorithm :

step 1 : start.
step 2 : Input the processes along with their Arrival Time and Burst Time.
step 3 : Find the waiting time for all processes
i.e. for a given process i :
wt[i] = (bt[0] + bt[1] + ..... + bt[i-1]) - at[i]
step 4 : Now Find Turn Around Time
= Waiting time + burst time for all processes
step 5 : Average Waiting Time
= total_waiting_time / no_of_processes
step 6 : Average Turn Around Time
= total_turn_around_time / no_of_processes
step 7 : stop.

2) SJF Algorithm (Non-Preemptive) :

step 1 : start
step 2 : Sort all the processes according to the

arrival time.

Step 2 : Then select that process which has minimum arrival time and Burst Time.

Step 3 : After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum burst time.

Step 4 : Find completion time of process of each process
Step 5 : Find Turn Around Time of each process.
Step 6 : Find Waiting time of each process.
Step 7 : Find average waiting time -
Step 8 : Find Average Turn Around Time.
Step 9 : Stop.

3) SJF Algorithm : (Preemptive)

Step 1 : Start.

Step 2 : Traverse until all process gets completely executed.
a) Find process with minimum remaining time at every single time lap.
b) Reduce its time by 1.
c) check if its remaining time becomes 0.
d) Increment the counter of process completion.
e) Completion time of current process = current_time + 1.
f) Calculate waiting time for each completed process.

: Date : 1/20 | Page No :



wt[i] = completion time − arrival_time − burst_time

g) Increment time lap by one.

Step 3 : Find Turn Around Time (waiting_time + burst_time).

Step 4 : Find Average Waiting Time.

Step 5 : Find Average Turn Around Time.

Step 6 : Stop.

4) Priority Based Scheduling Algorithm : (Non-Preemptive).

Step 1 : Start.

Step 2 : Read the input for processes with their arrival time, burst time and priority.

Step 3 : First process will schedule, which have the lowest arrival time, if two or more processes will have lowest arrival time, then whoever has higher priority will schedule first.

Step 4 : Once all the processes have been arrived, we can shedule them based on their priority.

Step 5 : Find Completion Time for each process.

Step 6: Find Turn Around Time for each process.

Step 7: Find Waiting Time for each process.

Step 8: Find Average Waiting Time.

Step 9: Find Average Turn Around Time.

Step 10: Stop.

5) Round Robin Algorithm: (Preemptive)

Step 1: Start.

Step 2: Create two arrays of burst time res_b[] and of arrival time res_a[] and copy the value of the b[] and a[] array to calculate the remaining time. (b[] is burst time, a[] is arrival time)

Step 3: Create an another array for wt[] to store waiting time.

Step 4: Initialize Time: t=0;

Step 5: Keep traversing the all process while all process are not done.
Do following for ith process if it is not done yet.
a - if res_a[i] <= q (quantum time :- q)
a. 1. if res_b[i] > q
a. t = t+q

b. res_b[i] -= q
c. q[i] += q;

2. else res_b[i] <= q (for last to execute)
a. t = t + b[i];
b. wt[i] = t - b[i] - q[i];
c. res_b[i] = 0;

b- else res_q[i] < q
1. Initialize j=0 to number of process
if q[j] < q[i] (compare is there any other process come before these process)

1. if res_b[j] > q
a. t = t + q
b. res_b[j] -= q;
c. q[j] += q;
2. else res_b[j] <= q
a. t = t + b[j];
b. wt[j] = t - b[j] - q[j];
c. res_b[j] = 0;

2. Now we executing the i'th process
1. if res_b[i] > q
a. t = t + q
b. res_b[i] -= q;
c. q[i] += q;
2. else res_b[i] <= q
a. t = t + b[i];
b. wt[i] = t - b[i] - q[i];
c. res_b[i] = 0;

## Test Cases :-

| Test case Id | steps to be executed | Expected Result | Actual Result | Test Case Pass/Fail |
|---|---|---|---|---|
| 1. | No. of processes = 4 | FCFS : | FCFS : | |

| At | Bt | Priority | Ct | TAt | Wt | Ct | TAt | Wt |
|---|---|---|---|---|---|---|---|---|
| 0 | 8 | 1 | 8 | 8 | 0 | 8 | 8 | 0 |
| 1 | 4 | 2 | 12 | 11 | 7 | 12 | 11 | 7 |
| 2 | 9 | 3 | 21 | 19 | 10 | 21 | 19 | 10 |
| 3 | 5 | 4 | 26 | 23 | 18 | 26 | 23 | 18 |
| | | | Avg. Wt = 8.75 | | | Avg. Wt = 8.75 | | |
| | | | Avg. TAt = 15.25 | | | Avg. TAt = 15.25 | | |

| SJF (Non-Preemptive) | | | SJF - (Non-Preemptive) | | |
|---|---|---|---|---|---|
| Ct | TAt | Wt | Ct | TAt | Wt |
| 8 | 8 | 0 | 8 | 8 | 0 |
| 12 | 11 | 7 | 12 | 11 | 7 |
| 26 | 24 | 15 | 26 | 24 | 15 |
| 17 | 14 | 9 | 17 | 14 | 9 |
| Avg. Wt = 7.75 | | | Avg. Wt = 7.75 | | |
| Avg. TAt = 14.25 | | | Avg. TAt = 14.25 | | |

| SJF (Preemptive) | | | SJF (Preemptive) | | |
|---|---|---|---|---|---|
| Ct | TAt | Wt | Ct | TAt | Wt |
| 17 | 17 | 9 | 17 | 17 | 9 |
| 5 | 4 | 0 | 5 | 4 | 0 |
| 26 | 24 | 15 | 26 | 24 | 15 |
| 10 | 7 | 2 | 10 | 7 | 2 |
| Avg. Wt = 6.5 | | | Avg. Wt = 6.5 | | |
| Avg. TAt = 13.0 | | | Avg. TAt = 13.0 | | |

| Test case Id | Steps to be executed | Expected Result | Actual Result | Test case Pass/Fail |
|---|---|---|---|---|

**Priority (Non-Preemptive)**

| Ct | TAt | Wt |
|---|---|---|
| 8 | 8 | 0 |
| 26 | 25 | 21 |
| 22 | 20 | 11 |
| 13 | 10 | 5 |

Avg. Wt. = 9.25
Avg. TAt. = 15.75

**Priority (Non-Preemptive)**

| Ct | TAt | Wt |
|---|---|---|
| 8 | 8 | 0 |
| 26 | 25 | 21 |
| 22 | 20 | 11 |
| 13 | 10 | 5 |

Avg Wt. = 9.25
Avg. TAT. = 15.75

**Round Robin (Preemptive)**

| Ct | TAt | Wt |
|---|---|---|
| 20 | 20 | 12 |
| 11 | 10 | 7 |
| 24 | 22 | 15 |
| 20 | 17 | 15 |

Avg. Wt. = 12.25
Avg. TAt = 17.25

**Round Robin (Preemptive)**

| Ct | TAt | Wt |
|---|---|---|
| 20 | 20 | 12 |
| 11 | 10 | 7 |
| 24 | 22 | 15 |
| 20 | 17 | 15 |

Avg. Wt = 12.25
Avg. TAt = 17.25

## Conclusion :-

We have analysed and implemented CPU scheduling Algorithms successfully. We done Evaluation of performances of different scheduling algorithms.

## CODE :-

```
/*
 * Problem Statement :-
 *  Write a program in C/C++/Java/Python to simulate CPU Scheduling Algorithms:
 *  FCFS, SJF(Preemptive), Priority(Non-Preemptive) and Round Robin (Preemptive).
 */
```

## FCFS Scheduling Algorithm :-

```
package assignmentNo_1;

public class FCFS_Scheduling
{
    void display(Process p1[])
    {
        float avgWt = 0, avgTat = 0;
        int n = p1.length;

        Process p[] = new Process[n];
        GanttChart g[] = new GanttChart[n];
        for(int i=0; i<n; i++)
        {
            g[i] = new GanttChart();
            p[i] = new Process();
            p[i].processId = p1[i].processId;
            p[i].arrivalTime = p1[i].arrivalTime;
            p[i].burstTime = p1[i].burstTime;
        }

        for(int i=0; i<n; i++)
        {
            int min_idx = i;
            for(int j = i+1; j<n; j++)
            {
                if(p[j].arrivalTime < p[min_idx].arrivalTime)
                {
                    min_idx = j;
                }
            }
            Process temp = p[i];
            p[i] = p[min_idx];
            p[min_idx] = temp;
```

```
        }

        for(int i=0; i<n; i++)
        {
            if(i==0)
            {
                p[i].completionTime = p[i].arrivalTime + p[i].burstTime;
                g[i].pStartTime = p[i].arrivalTime;
            }
            else
            {
                if(p[i].arrivalTime > p[i-1].completionTime)
                {
                    p[i].completionTime = p[i].arrivalTime + p[i].burstTime;
                    g[i].pStartTime = p[i].arrivalTime;
                }
                else
                {
                    p[i].completionTime = p[i-1].completionTime + p[i].burstTime;
                    g[i].pStartTime = p[i-1].completionTime;
                }
            }

            p[i].turnAroundTime = p[i].completionTime - p[i].arrivalTime;
            p[i].waitingTime = p[i].turnAroundTime - p[i].burstTime;

            avgWt  += p[i].waitingTime;
            avgTat += p[i].turnAroundTime;
            g[i].pId = i;
            g[i].pEndTime = p[i].completionTime;
        }

        System.out.println("\nProcess_\tArrival_\tBurst_  \tCompletion_  \tTurnAround_  \tWaiti
ng_");
        System.out.println("   Id    \tTime    \tTime     \tTime    \tTime    \tTime");

        for(int  i = 0 ; i<n;  i++)
        {
            System.out.println("   "+p[i].processId+"  \t\t"+p[i].arrivalTime+
                "  \t\t"+p[i].burstTime+"  \t\t"+p[i].completionTime+"  \t\t"
                +p[i].turnAroundTime + "  \t\t"+p[i].waitingTime);
        }

        System.out.print("\n\n\t\t Gantt Chart : ");
        System.out.print("\t 0");
        int prev = 0;
```

```
        for(int i=0; i<n;i++)
        {
            if(g[i].pStartTime == prev)
            {
                System.out.print("| P"+ (g[i].pId+1) +"| "+g[i].pEndTime);
            }
            else
            {
                System.out.print("| "+ g[i].pStartTime +"| P"+ (g[i].pId+1) +"| "+g[i].pEndTime
);
            }
            prev = g[i].pEndTime;
        }

        System.out.println("\n\n\t\t Average Waiting Time     : "+ (avgWt/n));
        System.out.println("\t\t Average Turn Around Time : "+(avgTat/n));
    }
}
```

## SJF Scheduling Algorithm (Non-Preemptive):-

```
package assignmentNo_1;

public class SJF_Non_Preemptive_Scheduling
{
    void display(Process p1[])
    {
        float avgWt = 0, avgTat = 0;
        int n = p1.length, tot = 0, st = 0;

        Process p[] = new Process[n];
        GanttChart g[] = new GanttChart[n];
        for(int i=0; i<n; i++)
        {
            g[i] = new GanttChart();
            p[i] = new Process();
            p[i].processId = p1[i].processId;
            p[i].arrivalTime = p1[i].arrivalTime;
            p[i].burstTime = p1[i].burstTime;
        }
```

```
        boolean completed[] = new boolean[n];
        for(int i=0; i<n; i++)
        {
            completed[i] = false;
        }

        while(true)
        {
            int min = 9999, idx = n;
            if(tot == n)
            {
                break;
            }

            for(int i=0; i<n; i++)
            {
                if( (p[i].arrivalTime <= st) && (completed[i] == false) && (p[i].burstTime < mi
n) )
                {
                    min = p[i].burstTime;
                    idx = i;
                }
            }

            if (idx == n)
            {
                st++;
            }
            else
            {
                p[idx].completionTime = st + p[idx].burstTime;
                g[tot].pId = idx;
                g[tot].pStartTime = st;
                g[tot].pEndTime = p[idx].completionTime;
                st += p[idx].burstTime;
                completed[idx] = true;
                p[idx].turnAroundTime = p[idx].completionTime - p[idx].arrivalTime;
                p[idx].waitingTime = p[idx].turnAroundTime - p[idx].burstTime;
                tot += 1;
            }

        }


        System.out.println("\nProcess_\tArrival_\tBurst_  \tCompletion_  \tTurnAround_  \tWaiti
ng_");
```

```
        System.out.println("   Id    \tTime    \tTime     \tTime    \tTime    \tTime");

        for(int  i = 0 ; i<n;  i++)
        {
            System.out.println("   "+p[i].processId+"  \t\t"+p[i].arrivalTime+
                "  \t\t"+p[i].burstTime+"  \t\t"+p[i].completionTime+"  \t\t"
                +p[i].turnAroundTime + "  \t\t"+p[i].waitingTime);

            avgWt  += p[i].waitingTime;
            avgTat += p[i].turnAroundTime;
        }

        System.out.print("\n\n\t\t Gantt Chart:");
        System.out.print("\t 0");
        int prev = 0;
        for(int i=0; i<n;i++)
        {
            if(g[i].pStartTime == prev)
            {
                System.out.print("| P"+ (g[i].pId+1) +"| "+g[i].pEndTime);
            }
            else
            {
                System.out.print("| "+ g[i].pStartTime +"| P"+ (g[i].pId+1) +"| "+g[i].pEndTime
);
            }
            prev = g[i].pEndTime;
        }

        System.out.println("\n\n\t\t Average Waiting Time     : "+ (avgWt/n));
        System.out.println("\t\t Average Turn Around Time : "+(avgTat/n));
    }
}
```

## SJF Scheduling Algorithm (Preemptive):-

```
package assignmentNo_1;

public class SJF_Preemptive_Scheduling
{
    void display(Process p1[])
    {
```

```
        float avgWt = 0, avgTat = 0;
        int n = p1.length, tot = 0, st = 0, gIdx = 0;

        Process p[] = new Process[n];
        for(int i=0; i<n; i++)
        {
            p[i] = new Process();
            p[i].processId = p1[i].processId;
            p[i].arrivalTime = p1[i].arrivalTime;
            p[i].burstTime = p1[i].burstTime;
        }

        GanttChart g[] = new GanttChart[50];
        for(int i=0; i<50; i++)
        {
            g[i] = new GanttChart();
        }

        boolean completed[] = new boolean[n];
        int temp[] = new int[n];

        for(int i=0; i<n; i++)
        {
            temp[i] = p[i].burstTime;
            completed[i] = false;
        }

        while(true)
        {
            int min = 9999, idx = n;
            if(tot == n)
            {
                break;
            }

            for(int i=0; i<n; i++)
            {
                if( (p[i].arrivalTime <= st) && (completed[i] == false) && (p[i].burstTime < mi
n) )
                {
                    min = p[i].burstTime;
                    idx = i;
                }
            }

            if (idx == n)
```

```
            {
                st++;
            }
            else
            {
                g[gIdx].pId = idx;
                g[gIdx].pStartTime = st;
                p[idx].burstTime--;
                st++;
                g[gIdx++].pEndTime = st;
                if( p[idx].burstTime == 0)
                {
                    p[idx].completionTime = st;
                    completed[idx] = true;
                    tot += 1;
                }
            }

        }

        for(int i=0; i<n; i++)
        {
            p[i].turnAroundTime = p[i].completionTime - p[i].arrivalTime;
            p[i].waitingTime = p[i].turnAroundTime - temp[i];
            avgWt  += p[i].waitingTime;
            avgTat += p[i].turnAroundTime;
        }

        System.out.println("\nProcess_\tArrival_\tBurst_  \tCompletion_  \tTurnAround_  \tWaiti
ng_");
        System.out.println("   Id    \tTime    \tTime     \tTime    \tTime    \tTime");

        for(int  i = 0 ; i<n;  i++)
        {
            System.out.println("   "+p[i].processId+"  \t\t"+p[i].arrivalTime+
                "  \t\t"+temp[i]+"  \t\t"+p[i].completionTime+"  \t\t"
                +p[i].turnAroundTime + "  \t\t"+p[i].waitingTime);
        }

        System.out.print("\n\n\t\t Gantt Chart:");
        System.out.print("\t 0");
        int prev = 0;
        for(int i=0; i<50;i++)
        {
            if(g[i].pEndTime == 0) break;
            if(g[i].pStartTime == prev)
```

```
            {
                if(g[i].pId != g[i+1].pId)  System.out.print("| P"+ (g[i].pId+1) +"| "+g[i].pEn
dTime);
            }
            else
            {
                if(g[i].pId != g[i+1].pId)  System.out.print("| "+ g[i].pStartTime +"| P"+ (g[i
].pId+1) +"| "+g[i].pEndTime);
            }
            prev = g[i].pEndTime;
        }

        System.out.println("\n\n\t\t Average Waiting Time     : "+ (avgWt/n));
        System.out.println("\t\t Average Turn Around Time : "+(avgTat/n));
    }
}
```

## Priority Scheduling Algorithm (Non-Preemptive):-

```
package assignmentNo_1;

import java.util.Scanner;

public class Priority_Scheduling
{
    void display(Process p1[])
    {
        int n = p1.length, tot = 0, st = 0;
        float avgWt = 0, avgTat = 0;

        int priority[] = new int[n];
        @SuppressWarnings("resource")
        Scanner in = new Scanner(System.in);
        for(int i=0; i<n; i++)
        {
            System.out.print("\n\n\t Enter Priority of P"+ (i+1) + ": ");
            priority[i] = in.nextInt();
        }

        Process p[] = new Process[n];
        GanttChart g[] = new GanttChart[n];
        for(int i=0; i<n; i++)
```

```
        {
            g[i] = new GanttChart();
            p[i] = new Process();
            p[i].processId = p1[i].processId;
            p[i].arrivalTime = p1[i].arrivalTime;
            p[i].burstTime = p1[i].burstTime;
        }

        boolean completed[] = new boolean[n];
        for(int i=0; i<n; i++)
        {
            completed[i] = false;
        }

        while(true)
        {
            int max = -9999, idx = n;
            if(tot == n)
            {
                break;
            }

            for(int i=0; i<n; i++)
            {
                if( (p[i].arrivalTime <= st) && (completed[i] == false) && (priority[i] > max)
)
                {
                    max = priority[i];
                    idx = i;
                }
            }

            if (idx == n)
            {
                st++;
            }
            else
            {
                p[idx].completionTime = st + p[idx].burstTime;
                g[tot].pId = idx;
                g[tot].pStartTime = st;
                g[tot].pEndTime = p[idx].completionTime;
                st += p[idx].burstTime;
                completed[idx] = true;
                p[idx].turnAroundTime = p[idx].completionTime - p[idx].arrivalTime;
                p[idx].waitingTime = p[idx].turnAroundTime - p[idx].burstTime;
```

```
                tot += 1;
            }

        }


        System.out.println("\nProcess_\tPriority \tArrival_\tBurst_  \tCompletion_  \tTurnAroun
d_  \tWaiting_");
        System.out.println("   Id    \t   \t\tTime    \tTime     \tTime    \tTime    \tTime");

        for(int  i = 0 ; i<n;  i++)
        {
            System.out.println("   "+p[i].processId+"\t\t " +priority[i] +"\t\t"+p[i].arrivalTi
me+
                "  \t\t"+p[i].burstTime+"  \t\t"+p[i].completionTime+"  \t\t"
                +p[i].turnAroundTime + "  \t\t"+p[i].waitingTime);

            avgWt  += p[i].waitingTime;
            avgTat += p[i].turnAroundTime;
        }

        System.out.print("\n\n\t\t Gantt Chart:");
        System.out.print("\t 0");
        int prev = 0;
        for(int i=0; i<n;i++)
        {
            if(g[i].pStartTime == prev)
            {
                System.out.print("| P"+ (g[i].pId+1) +"| "+g[i].pEndTime);
            }
            else
            {
                System.out.print("| "+ g[i].pStartTime +"| P"+ (g[i].pId+1) +"| "+g[i].pEndTime
);
            }
            prev = g[i].pEndTime;
        }

        System.out.println("\n\n\t\t Average Waiting Time     : "+ (avgWt/n));
        System.out.println("\t\t Average Turn Around Time : "+(avgTat/n));

    }

}
```

## Round Robin Scheduling Algorithm (Preemptive):-

```
package assignmentNo_1;

public class Round_Robin_Scheduling
{
    void display(Process p1[], int q)
    {
        Process p[]= new Process[p1.length];

        for(int i=0; i<p1.length;i++)
        {
            p[i] = new Process();
            p[i].processId = p1[i].processId;
            p[i].arrivalTime=p1[i].arrivalTime;
            p[i].burstTime=p1[i].burstTime;
            p[i].completionTime=p1[i].completionTime;
            p[i].turnAroundTime=p1[i].turnAroundTime;
            p[i].waitingTime=p1[i].waitingTime;
        }

        float avgwt = 0f, avgta = 0f;

        int res_b[] = new int[p.length];
        int res_a[] = new int[p.length];
        String seq = new String();

        for(int i=0; i<p.length; i++)
        {
            res_b[i] = p[i].burstTime;
            res_a[i] = p[i].arrivalTime;
        }

        int t = 0;

        while (true) {
            boolean flag = true;
            for (int i = 0; i < p.length; i++) {

                if (res_a[i] <= t) {
                    if (res_a[i] <= q) {
                        if (res_b[i] > 0) {
                            flag = false;
                            if (res_b[i] > q) {
                                t = t + q;
                                res_b[i] = res_b[i] - q;
                                res_a[i] = res_a[i] + q;
```

```
                            seq += "P" + p[i].processId + "->";
                        }
                        else {
                            t = t + res_b[i];
                            p[i].completionTime = t - p[i].arrivalTime;
                            p[i].waitingTime = t - p[i].burstTime - p[i].arrivalTime;
                            res_b[i] = 0;
                            seq += "P" + p[i].processId + "->";
                        }
                    }
                }
                else if (res_a[i] > q) {
                    for (int j = 0; j < p.length; j++) {
                        if (res_a[j] < res_a[i]) {
                            if (res_b[j] > 0) {
                                flag = false;
                                if (res_b[j] > q) {
                                    t = t + q;
                                    res_b[j] = res_b[j] - q;
                                    res_a[j] = res_a[j] + q;

                                    seq += "P" + p[j].processId + "->";
                                }
                                else {
                                    t = t + res_b[j];
                                    p[j].completionTime = t - p[j].arrivalTime;
                                    p[j].waitingTime = t -
p[j].burstTime - p[j].arrivalTime;
                                    res_b[j] = 0;
                                    seq += "P" + p[j].processId + "->";
                                }
                            }
                        }
                    }

                    if (res_b[i] > 0) {
                        flag = false;

                        if (res_b[i] > q) {
                            t = t + q;
                            res_b[i] = res_b[i] - q;
                            res_a[i] = res_a[i] + q;
                            seq += "P" + p[i].processId + "->";
                        }
                        else {
                            t = t + res_b[i];
```

```
                            p[i].completionTime = t - p[i].arrivalTime;
                            p[i].waitingTime = t - p[i].burstTime - p[i].arrivalTime;
                            res_b[i] = 0;
                            seq += "P" + p[i].processId + "->";
                        }
                    }
                }
            }

            else if (res_a[i] > t) {
                t++;
                i--;
            }
        }
        if (flag) {
            break;
        }
    }

    for (int i = 0; i < p.length; i++)
    {
        p[i].turnAroundTime = p[i].completionTime - p[i].arrivalTime;
    }

    System.out.println("\nProcess_\tArrival_\tBurst_  \tCompletion_  \tTurnAround_   \tWait
ing_");
    System.out.println("   Id    \tTime    \tTime     \tTime    \tTime    \tTime");
    for(int  i = 0 ; i< p.length;  i++)
    {
        System.out.println("   " + p[i].processId + "  \t\t" + p[i].arrivalTime + "  \t\t"
+ p1[i].burstTime + "  \t\t" + p[i].completionTime + "  \t\t" + p[i].turnAroundTime + "  \t\t"
 + p[i].waitingTime ) ;
        avgwt += p[i].waitingTime;
        avgta += p[i].turnAroundTime;
    }

    System.out.println("\n\n\t Sequence : " + seq);

    System.out.println("\n\n\t\t Average Waiting Time     : "+ (avgwt/p.length));
    System.out.println("\t\t Average Turn Around Time : "+ (avgta/p.length));

}

}
```

## AssignmentNo1.java

```
package assignmentNo_1;

import java.util.Scanner;

class Process
{
    int processId, arrivalTime, burstTime, completionTime, turnAroundTime, waitingTime;

    Process()
    {
        processId = arrivalTime = burstTime = completionTime = turnAroundTime = waitingTime = 0
;
    }
}


class GanttChart
{
    int pId, pStartTime, pEndTime;

    GanttChart()
    {
        pId = pStartTime = pEndTime = 0;
    }
}


public class AssignmentNo1
{
    public static void main(String[] args)
    {
        System.out.println("\n\n\t\t\t === SCHEDULING ALGORITHMS ===");

        Scanner sc = new Scanner(System.in);
        boolean flag = true;

        System.out.print("\n\t Enter total number of processes : ");
        int total = sc.nextInt();

        Process p[] = new Process[total];

        for(int i=0; i<total; i++)
        {
            p[i] = new Process();
            System.out.println("\n\t Process No. "+ (i+1));
```

```
            p[i].processId = i+1;
            System.out.print("\n\t\t Enter Arrival Time : ");
            p[i].arrivalTime = sc.nextInt();
            System.out.print("\n\t\t Enter Burst Time : ");
            p[i].burstTime = sc.nextInt();
        }

        while(flag)
        {
            System.out.println("\n ==== Main-
Menu ==== \n\t 1. FCFS Scheduling \n\t 2. SJF Scheduling (Non Preemptive)"
                    + "\n\t 3. SJF Scheduling (Preemptive) \n\t 4. Priority Sheduling (Non Pree
mptive)\n\t 5. Round Robin Sheduling (Preemptive) \n\t 6. EXIT...");

            System.out.print("\n\t Enter choice : ");
            int choice1 = sc.nextInt();

            switch(choice1)
            {
                case 1:
                    System.out.println("\n\t 1.] \t'First-Come-First-
Served' (FCFS) Scheduling");
                    FCFS_Scheduling obj1 = new FCFS_Scheduling();
                    obj1.display(p);
                    System.out.println("\n\t\t ========================================");
                    break;

                case 2:
                    System.out.println("\n\n\t 2.] \t'Shortest-Jod-
First' (SJF) Scheduling (Non-Preemptive)");
                    SJF_Non_Preemptive_Scheduling obj2 = new SJF_Non_Preemptive_Scheduling();
                    obj2.display(p);
                    System.out.println("\n\t\t ========================================");
                    break;

                case 3:
                    System.out.println("\n\n\t 3.] \t'Shortest-Jod-
First' (SJF) Scheduling (Preemptive)");
                    SJF_Preemptive_Scheduling obj3 = new SJF_Preemptive_Scheduling();
                    obj3.display(p);
                    System.out.println("\n\t\t ========================================");
                    break;

                case 4:
                    System.out.println("\n\n\t 4.] \t'Priority Scheduling (Non-Preemptive)");
                    Priority_Scheduling obj4 = new Priority_Scheduling();
```

```
                    obj4.display(p);
                    System.out.println("\n\t\t ========================================");
                    break;

                case 5:
                    System.out.println("\n\n\t 5.] \t'Round Robin' Scheduling (Preemptive)");
                    Round_Robin_Scheduling obj5 = new Round_Robin_Scheduling();
                    System.out.print("\n\t Enter Time Quantum :  ");
                    int q = sc.nextInt();
                    obj5.display(p, q);
                    System.out.println("\n\t\t ========================================");
                    break;

                case 6:
                    flag = false;
                    System.out.print("\n\t\t\t\t Thank You ...!!");
                    System.out.println("\n\n\t\t\t\t *****=====*****");
                    break;

                default:
                    System.out.print("\n\t\t Invalid Choice ...!!");
            }
        }
        sc.close();
    }
}
```

**OUTPUT :-**

```
                    === SCHEDULING ALGORITHMS ===
        Enter total number of processes : 4
        Process No. 1
               Enter Arrival Time : 0
               Enter Burst Time : 8
        Process No. 2
               Enter Arrival Time : 1
               Enter Burst Time : 4
```

```
        Process No. 3

               Enter Arrival Time : 2

               Enter Burst Time : 9

        Process No. 4

               Enter Arrival Time : 3

               Enter Burst Time : 5

 ==== Main-Menu ====
        1. FCFS Scheduling
        2. SJF Scheduling (Non Preemptive)
        3. SJF Scheduling (Preemptive)
        4. Priority Sheduling (Non Preemptive)
        5. Round Robin Sheduling (Preemptive)
        6. EXIT...

        Enter choice : 1

        1.]  'First-Come-First-Served' (FCFS) Scheduling

Process_     Arrival_     Burst_       Completion_  TurnAround_  Waiting_
   Id        Time         Time         Time         Time         Time
   1         0            8            8            8            0
   2         1            4            12           11           7
   3         2            9            21           19           10
   4         3            5            26           23           18


               Gantt Chart :       0| P1| 8| P2| 12| P3| 21| P4| 26

               Average Waiting Time     : 8.75
               Average Turn Around Time : 15.25

               ========================================

 ==== Main-Menu ====
        1. FCFS Scheduling
        2. SJF Scheduling (Non Preemptive)
        3. SJF Scheduling (Preemptive)
        4. Priority Sheduling (Non Preemptive)
        5. Round Robin Sheduling (Preemptive)
        6. EXIT...

        Enter choice : 2


        2.]  'Shortest-Jod-First' (SJF) Scheduling (Non-Preemptive)

Process_     Arrival_     Burst_       Completion_  TurnAround_  Waiting_
   Id        Time         Time         Time         Time         Time
   1         0            8            8            8            0
   2         1            4            12           11           7
   3         2            9            26           24           15
   4         3            5            17           14           9


               Gantt Chart: 0| P1| 8| P2| 12| P4| 17| P3| 26

               Average Waiting Time     : 7.75
               Average Turn Around Time : 14.25
```

```
              ========================================

 ==== Main-Menu ====
       1. FCFS Scheduling
       2. SJF Scheduling (Non Preemptive)
       3. SJF Scheduling (Preemptive)
       4. Priority Sheduling (Non Preemptive)
       5. Round Robin Sheduling (Preemptive)
       6. EXIT...

       Enter choice : 3


       3.]  'Shortest-Jod-First' (SJF) Scheduling (Preemptive)

Process_     Arrival_     Burst_       Completion_  TurnAround_  Waiting_
   Id        Time         Time         Time         Time         Time
   1         0            8            17           17           9
   2         1            4            5            4            0
   3         2            9            26           24           15
   4         3            5            10           7            2


              Gantt Chart: 0| P1| 1| P2| 5| P4| 10| P1| 17| P3| 26

              Average Waiting Time     : 6.5
              Average Turn Around Time : 13.0

              ========================================

 ==== Main-Menu ====
       1. FCFS Scheduling
       2. SJF Scheduling (Non Preemptive)
       3. SJF Scheduling (Preemptive)
       4. Priority Sheduling (Non Preemptive)
       5. Round Robin Sheduling (Preemptive)
       6. EXIT...

       Enter choice : 4


       4.]  'Priority Scheduling (Non-Preemptive)


       Enter Priority of P1: 1


       Enter Priority of P2: 2


       Enter Priority of P3: 3


       Enter Priority of P4: 4

Process_     Priority     Arrival_     Burst_       Completion_  TurnAround_  Waiting_
   Id                     Time         Time         Time         Time         Time
   1          1          0            8            8            8            0
   2          2          1            4            26           25           21
   3          3          2            9            22           20           11
   4          4          3            5            13           10           5


              Gantt Chart: 0| P1| 8| P4| 13| P3| 22| P2| 26
```

```
              Average Waiting Time     : 9.25
              Average Turn Around Time : 15.75

              ========================================

 ==== Main-Menu ====
       1. FCFS Scheduling
       2. SJF Scheduling (Non Preemptive)
       3. SJF Scheduling (Preemptive)
       4. Priority Sheduling (Non Preemptive)
       5. Round Robin Sheduling (Preemptive)
       6. EXIT...

       Enter choice : 5


       5.]   'Round Robin' Scheduling (Preemptive)

       Enter Time Quantum :  2

Process_     Arrival_     Burst_       Completion_  TurnAround_         Waiting_
   Id        Time         Time         Time         Time         Time
   1         0            8            20           20           12
   2         1            4            11           10           7
   3         2            9            24           22           15
   4         3            5            20           17           15


       Sequence : P1->P2->P3->P1->P4->P2->P1->P3->P4->P1->P3->P4->P3->P3->


              Average Waiting Time     : 12.25
              Average Turn Around Time : 17.25

              ========================================

 ==== Main-Menu ====
       1. FCFS Scheduling
       2. SJF Scheduling (Non Preemptive)
       3. SJF Scheduling (Preemptive)
       4. Priority Sheduling (Non Preemptive)
       5. Round Robin Sheduling (Preemptive)
       6. EXIT...

       Enter choice : 6

                           Thank You ...!!

                           *****=====*****
```