Experiment 5:- CPU Scheduling Algorithms

1.1 Objective:

Simulate the following CPU scheduling algorithms 1. FCFS 2. SJF 3. Priority 4. Round Robin. Calculate Average Waiting Time, Average Turn-Around Time, Average Response time for each algorithm.

1.2 Software Required:

- Operating System: UBUNTU/Linux
- Software Required: Terminal

1.3 Pre-Requisite:

- Basic C Programming,
- Concept of Unix System call commands
- Understanding of File I/O.

1.4 Program:

```
#include <stdio.h>
 void FCFS(int processes[], int n, int burst_time[])
 {
        int waiting_time[n], turnaround_time[n], total_waiting_time =
0,total_turnaround_time = 0;
        waiting_time[0] = 0;

        // Waiting time for first process is 0
        // Calculating waiting time for each process
        for (int i = 1; i < n; i++)

        {
                waiting_time[i] = burst_time[i - 1] + waiting_time[i - 1];
                total_waiting_time += waiting_time[i];
        }
```

```
  // Calculating turnaround time for each process
    for (int i = 0; i < n; i++)

    {
        turnaround_time[i] = burst_time[i] + waiting_time[i];
        total_turnaround_time += turnaround_time[i];
    }
    printf("First-Come, First-Served (FCFS) Scheduling Algorithm\n");
    printf("--------------------------------------------------\n");
    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
  // Printing process details
    for (int i = 0; i < n; i++)

    {
    printf("%d\t%d\t\t%d\t\t%d\n", processes[i], burst_time[i],waiting_time[i], turnaround_time[i]);
    }
    printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
    printf("Average Turnaround Time: %.2f\n",(float)total_turnaround_time / n);
    printf("\n");
}
void SJF(int processes[], int n, int burst_time[])

{

    int waiting_time[n], turnaround_time[n], completion_time[n],total_waiting_time = 0,

    total_turnaround_time = 0;
    for (int i = 0; i < n; i++)

    {

        int shortest_job_index = i;
       // Find the shortest job
        for (int j = i + 1; j < n; j++)

        {
            if (burst_time[j] < burst_time[shortest_job_index])shortest_job_index = j;
```

```
          }
        // Swap the shortest job with the current process
          int temp = burst_time[i];
          burst_time[i] = burst_time[shortest_job_index];
          burst_time[shortest_job_index] = temp;
          temp = processes[i];
          processes[i] = processes[shortest_job_index];
          processes[shortest_job_index] = temp;

}

waiting_time[0] = 0;

// Waiting time for first process is 0

// Calculating waiting time for each process

for (int i = 1; i < n; i++)

{

          waiting_time[i] = burst_time[i - 1] + waiting_time[i - 1];
          total_waiting_time += waiting_time[i];

}
 // Calculating turnaround time for each process
for (int i = 0; i < n; i++)

{

          turnaround_time[i] = burst_time[i] + waiting_time[i];
          total_turnaround_time += turnaround_time[i];
}
printf("Shortest Job First (SJF) Scheduling Algorithm\n");
printf("---------------------------------------------\n");
printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

// Printing process details

for (int i = 0; i < n; i++)
```

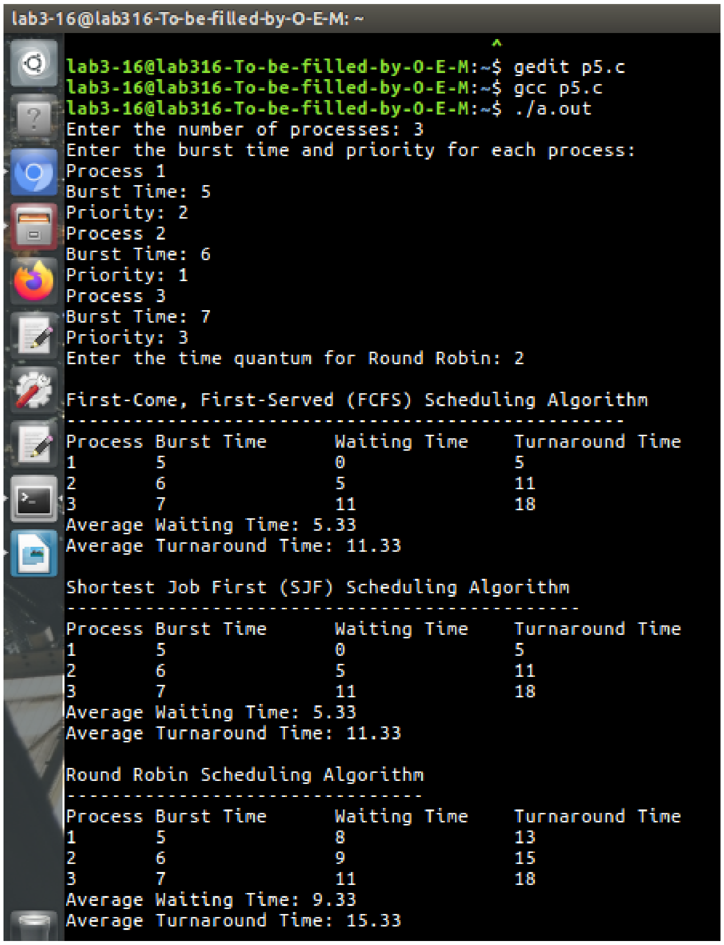
```
        {
        printf("%d\t%d\t\t%d\t\t%d\n", processes[i], burst_time[i],waiting_time[i], turnaround_time[i]);
        }
         printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
         printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
         printf("\n");
}
void RoundRobin(int processes[], int n, int burst_time[], int quantum)
{
        int remaining_time[n], waiting_time[n], turnaround_time[n],total_waiting_time = 0,
        total_turnaround_time = 0;
        // Copying burst time into remaining time array
        for (int i = 0; i < n; i++)
        {
                remaining_time[i] = burst_time[i];
        }
        int time = 0; // Current time
        // Run the round robin algorithm
        while (1)
        {
                int all_processes_completed = 1;
                // Traverse all processes
                for (int i = 0; i < n; i++)
                {
                        if (remaining_time[i] > 0)
                        {
                                all_processes_completed = 0;
```

```
            // There is still a pending process
            if (remaining_time[i] > quantum)
            {
                time += quantum;
                remaining_time[i] -= quantum;
            }
            else
            {
                time += remaining_time[i];
                waiting_time[i] = time - burst_time[i];
                remaining_time[i] = 0;
            }
        }
    }
    if (all_processes_completed)
    {
        break;
    }
}
 // Calculating turnaround time for each process
 for (int i = 0; i < n; i++)
{
    turnaround_time[i] = burst_time[i] + waiting_time[i];
    total_waiting_time += waiting_time[i];
    total_turnaround_time += turnaround_time[i];
}
printf("Round Robin Scheduling Algorithm\n");
```

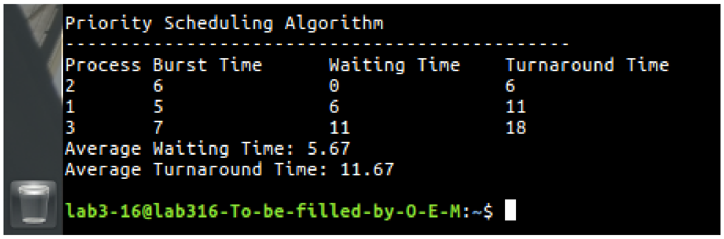
```
        printf("--------------------------------\n");

        printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
        // Printing process details
        for (int i = 0; i < n; i++)

        {

        printf("%d\t%d\t\t%d\t\t%d\n", processes[i], burst_time[i],waiting_time[i], turnaround_time[i]);
        }
        printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);

        printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);

        printf("\n");

}
void Priority(int processes[], int n, int burst_time[], int priority[])

{

        int waiting_time[n], turnaround_time[n], total_waiting_time = 0,total_turnaround_time = 0;
        for (int i = 0; i < n; i++)

        {
                int highest_priority_index = i;
                // Find the highest priority job
                 for (int j = i + 1; j < n; j++)

                {
                        if (priority[j] < priority[highest_priority_index])highest_priority_index = j;
                }
                // Swap the highest priority job with the current process
                int temp = burst_time[i];
                burst_time[i] = burst_time[highest_priority_index];
                burst_time[highest_priority_index] = temp;
                temp = processes[i];
                processes[i] = processes[highest_priority_index];
        processes[highest_priority_index] = temp;
        temp = priority[i];
        priority[i] = priority[highest_priority_index];
        priority[highest_priority_index] = temp;
}
waiting_time[0] = 0; // Waiting time for first process is 0
// Calculating waiting time for each process
for (int i = 1; i < n; i++)
{
        waiting_time[i] = burst_time[i - 1] + waiting_time[i - 1];
        total_waiting_time += waiting_time[i];
}
// Calculating turnaround time for each process
for (int i = 0; i < n; i++)
{
        turnaround_time[i] = burst_time[i] + waiting_time[i];
        total_turnaround_time += turnaround_time[i];
}
printf("Priority Scheduling Algorithm\n");
printf("---------------------------------------------\n");
printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
 // Printing process details
for (int i = 0; i < n; i++)
{
printf("%d\t%d\t\t%d\t\t%d\n", processes[i], burst_time[i],waiting_time[i], turnaround_time[i]);
}
```

```
        printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);

        printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);

        printf("\n");

}
int main()

{

        int n;

        printf("Enter the number of processes: ");

        scanf("%d", &n);

        int processes[n], burst_time[n], priority[n];

        printf("Enter the burst time and priority for each process:\n");

        for (int i = 0; i < n; i++)

        {

                printf("Process %d\n", i + 1);
                printf("Burst Time: ");
                scanf("%d", &burst_time[i]);
                printf("Priority: ");
                scanf("%d", &priority[i]);
                processes[i] = i + 1;

        }

        int quantum;

        printf("Enter the time quantum for Round Robin: ");

        scanf("%d", &quantum);

        printf("\n");
        FCFS(processes, n, burst_time);
        SJF(processes, n, burst_time);
        RoundRobin(processes, n, burst_time, quantum);
        Priority(processes, n, burst_time, priority);
        return 0;
}
```

1.5 Results:



```
lab3-16@lab316-To-be-filled-by-O-E-M:~$ gedit p5.c
lab3-16@lab316-To-be-filled-by-O-E-M:~$ gcc p5.c
lab3-16@lab316-To-be-filled-by-O-E-M:~$ ./a.out
Enter the number of processes: 3
Enter the burst time and priority for each process:
Process 1
Burst Time: 5
Priority: 2
Process 2
Burst Time: 6
Priority: 1
Process 3
Burst Time: 7
Priority: 3
Enter the time quantum for Round Robin: 2

First-Come, First-Served (FCFS) Scheduling Algorithm
--------------------------------------------------
Process Burst Time      Waiting Time    Turnaround Time
1       5               0               5
2       6               5               11
3       7               11              18
Average Waiting Time: 5.33
Average Turnaround Time: 11.33

Shortest Job First (SJF) Scheduling Algorithm
---------------------------------------------
Process Burst Time      Waiting Time    Turnaround Time
1       5               0               5
2       6               5               11
3       7               11              18
Average Waiting Time: 5.33
Average Turnaround Time: 11.33

Round Robin Scheduling Algorithm
--------------------------------
Process Burst Time      Waiting Time    Turnaround Time
1       5               8               13
2       6               9               15
3       7               11              18
Average Waiting Time: 9.33
Average Turnaround Time: 15.33
```



```
Priority Scheduling Algorithm
---------------------------------------------
Process Burst Time      Waiting Time    Turnaround Time
2       6               0               6
1       5               6               11
3       7               11              18
Average Waiting Time: 5.67
Average Turnaround Time: 11.67

lab3-16@lab316-To-be-filled-by-O-E-M:~$ █
```