Assignment No. 3

Name: **Sahil Jagadale** Division: TE-11 Batch: L-11 Roll No.: **33327**

Title: SJF & Round Robin CPU Scheduling

Aim: Implement C program for CPU scheduling algorithms: Shortest Job First (SJF) and

Round Robin with different arrival time.

THEORY:

1. **Shortest Job First (SJF)**scheduling works on the process with the shortest burst time or duration first.

This is the best approach to minimize waiting time. This is used in Batch Systems.

It is of two types:

1) Non Pre-emptive
2) Pre-emptive

To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time. This scheduling algorithm is optimal if all the jobs/processes are available at the same time.

2. **Round Robin (RR)** scheduling algorithm is mainly designed for time-sharing systems. This algorithm is like FCFS scheduling, but in Round Robin(RR) scheduling, preemption is added which enables the system to switch between processes. A fixed time is allotted to each process, called a quantum, for execution. Once a process is executed for the given time period that process is preempted and another process is executed for the given time period. Context switching is used to save states of preempted processes. This algorithm is simple and easy to implement and the most important is thing is this algorithm is starvation-free as all processes get a fair share of CPU

Some important characteristics of the Round Robin(RR) Algorithm are as follows:

- Round Robin Scheduling algorithm resides under the category of Preemptive Algorithms.
- This algorithm is one of the oldest, easiest, and fairest algorithm.
- In this algorithm, the time slice should be the minimum that is assigned to a specific task that
- needs to be processed. Though it may vary for different operating systems.
- This is a hybrid model and is clock-driven in nature.
- This is a widely used scheduling method in the traditional operating system.

Code:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

typedef struct
{
   int pid;
   int burst_time;
   int arrival_time;
   int waiting_time;
```

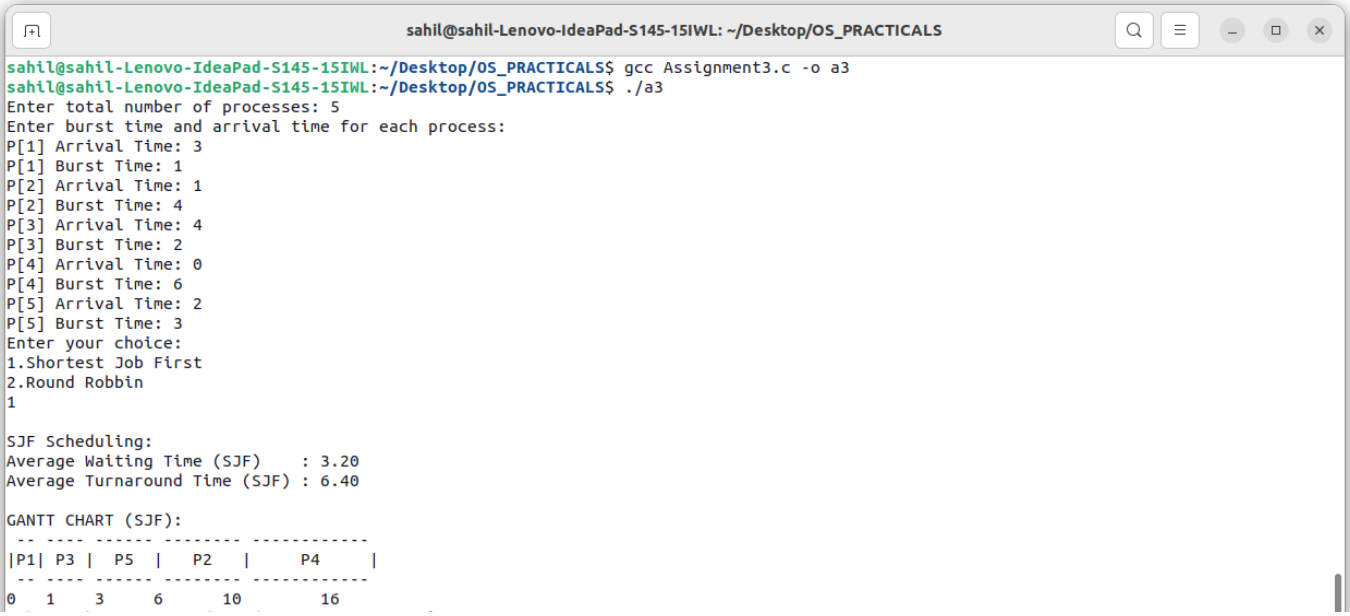
```
    int turnaround_time;
    int remaining_time;
} Process;

void print_gantt_chart(Process p[], int n);
void sjf(Process p[], int n);
void round_robin(Process p[], int n, int quantum);

int main()
{
    Process p[MAX];
    int i, j, n, quantum;
    int sum_waiting_time_sjf = 0, sum_turnaround_time_sjf = 0;
    int sum_waiting_time_rr = 0, sum_turnaround_time_rr = 0;

    printf("Enter total number of processes: ");
    scanf("%d", &n);
    printf("Enter burst time and arrival time for each process:\n");

    for (i = 0; i < n; i++)
    {
        p[i].pid = i + 1;
        printf("P[%d] Arrival Time: ", i + 1);
        scanf("%d", &p[i].arrival_time);
        printf("P[%d] Burst Time: ", i + 1);
        scanf("%d", &p[i].burst_time);
        p[i].waiting_time = p[i].turnaround_time = 0;
        p[i].remaining_time = p[i].burst_time;
    }
    int choice;
    printf("Enter your choice: ");
    printf("\n1.Shortest Job First\n");
    printf("2.Round Robbin\n");
    scanf("%d",&choice);
    switch(choice){
    case 1:{
      sjf(p, n);
      for (i = 0; i < n; i++)
      {
          sum_waiting_time_sjf += p[i].waiting_time;
          sum_turnaround_time_sjf += p[i].turnaround_time;
      }
      puts("");
      printf("SJF Scheduling:\n");
      printf("Average Waiting Time (SJF)    : %-2.2lf\n", (double)sum_waiting_time_sjf / (double)n);
      printf("Average Turnaround Time (SJF) : %-2.2lf\n", (double)sum_turnaround_time_sjf / (double)n);

      // Print Gantt charts
      puts(""); // Empty line
      printf("GANTT CHART (SJF):\n");
      print_gantt_chart(p, n);
      break;
  }
    case 2:{
      printf("\nEnter time quantum for Round Robin: ");
      scanf("%d", &quantum);
```

```
      round_robin(p, n, quantum);

      for (i = 0; i < n; i++)
      {
   sum_waiting_time_rr += p[i].waiting_time;
   sum_turnaround_time_rr += p[i].turnaround_time;
      }

      puts("");
      printf("Round Robin Scheduling:\n");
      printf("Average Waiting Time (RR)    : %-2.2lf\n", (double)sum_waiting_time_rr / (double)n);
      printf("Average Turnaround Time (RR) : %-2.2lf\n", (double)sum_turnaround_time_rr / (double)n);

      puts(""); // Empty line
      printf("GANTT CHART (RR):\n");
      print_gantt_chart(p, n);
      break;
}
default :{
break;
}
   }
   return 0;
}

void sjf(Process p[], int n)
{
   // SJF scheduling logic here
   int i, j;
   for (i = 0; i < n; i++)
   {
      for (j = i + 1; j < n; j++)
      {
         if (p[i].burst_time > p[j].burst_time)
         {
            // Swap processes
            Process temp = p[i];
            p[i] = p[j];
            p[j] = temp;
         }
      }
   }

   int current_time = 0;

   for (i = 0; i < n; i++)
   {
      p[i].waiting_time = current_time - p[i].arrival_time;
      if (p[i].waiting_time < 0)
      {
         p[i].waiting_time = 0;
      }
      p[i].turnaround_time = p[i].waiting_time + p[i].burst_time;
      current_time += p[i].burst_time;
   }
```

```
}

void round_robin(Process p[], int n, int quantum)
{
    // Round Robin scheduling logic here
    int i, j;
    int remaining_processes = n;
    int current_time = 0;

    while (remaining_processes > 0)
    {
        for (i = 0; i < n; i++)
        {
            if (p[i].remaining_time > 0)
            {
                if (p[i].remaining_time <= quantum)
                {
                    current_time += p[i].remaining_time;
                    p[i].remaining_time = 0;
                }
                else
                {
                    current_time += quantum;
                    p[i].remaining_time -= quantum;
                }

                p[i].waiting_time += current_time - p[i].arrival_time - p[i].burst_time;

                if (p[i].remaining_time == 0)
                {
                    remaining_processes--;
                    p[i].turnaround_time = current_time - p[i].arrival_time;
                }
            }
        }
    }
}


void print_gantt_chart(Process p[], int n)
{
    // Print Gantt chart logic here
    int i, j;
    printf(" ");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < p[i].burst_time; j++)
            printf("--");
        printf(" ");
    }
    printf("\n|");

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < p[i].burst_time - 1; j++)
            printf(" ");
```

```
        printf("P%d", p[i].pid);
        for (j = 0; j < p[i].burst_time - 1; j++)
            printf(" ");
        printf("|");
    }
    printf("\n ");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < p[i].burst_time; j++)
            printf("--");
        printf(" ");
    }
    printf("\n");

    int current_time = 0;
    printf("%d", current_time);
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < p[i].burst_time; j++)
        {
            current_time++;
            printf(" ");
        }
        printf("  %d", current_time);
    }
    printf("\n");
}
```

Output:




sahil@sahil-Lenovo-IdeaPad-S145-15IWL: ~/Desktop/OS_PRACTICALS
sahil@sahil-Lenovo-IdeaPad-S145-15IWL:~/Desktop/OS_PRACTICALS$ ./a3
Enter total number of processes: 5
Enter burst time and arrival time for each process:
P[1] Arrival Time: 0
P[1] Burst Time: 7
P[2] Arrival Time: 1
P[2] Burst Time: 5
P[3] Arrival Time: 2
P[3] Burst Time: 3
P[4] Arrival Time: 3
P[4] Burst Time: 1
P[5] Arrival Time: 4
P[5] Burst Time: 2
Enter your choice:
1.Shortest Job First
2.Round Robbin
2
Enter time quantum for Round Robin: 2
Round Robin Scheduling:
Average Waiting Time (RR) : 10.20
Average Turnaround Time (RR) : 11.00
GANTT CHART (RR):
| P1 | P2 | P3 |P4| P5 |
0 7 12 15 16 18
sahil@sahil-Lenovo-IdeaPad-S145-15IWL:~/Desktop/OS_PRACTICALS$