



# Mawlana Bhashani Science and Technology University

## Lab-Report

Report No: 10

Course code: ICT- 3110

Course title: Operating Systems Lab

Date of Performance:

Date of Submission:

### Submitted by

Name: Sayma Akter Shumi

ID: IT-18032

3<sup>rd</sup> year 1<sup>st</sup> semester

Session: 2017-2018

Dept. of ICT

MBSTU.

### Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

**Reference:**Nusrat Jahan Jui(IT-18039)

**Experiment no : 10**

**Experiment Name** : Implementation of Round Robin Scheduling Algorithm.

**Theory:**

Round robin scheduling is the preemptive scheduling in which every process get executed in a cyclic way. In Round Robin Scheduling, CPU is assigned to the process on the basis of FCFS for a fixed amount of time. This fixed amount of time is called as time quantum or time slice. After the time quantum expires, the running process is preempted and sent to the ready queue. Then, the processor is assigned to the next arrived process. It is always preemptive in nature.

Implementation :

- Step-1: We first have a queue where the processes are arranged in first come first serve order.
- Step-2: A quantum value is allocated to execute each process.
- Step-3: The first process is executed until the end of the quantum value. After this, an interrupt is generated and the state is saved.
- Step-04: The CPU then moves to the next process and the same method is followed.
- Step-05: Same steps are repeated till all the processes are over.

**Working Process :**

Code for Round Robin Scheduling Algorithm—

```
#include<stdio.h>
int main()
{
    int i, limit, total = 0, x, counter = 0, time_quantum;
    int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10];
```

```

float average_wait_time, average_turnaround_time;

printf("Enter Total Number of Processes:");
scanf("%d", &limit);
x = limit;
for(i = 0; i < limit; i++)
{
    printf("Enter Details of Process[%d]\n", i + 1);
    printf("Arrival Time:");
    scanf("%d", &arrival_time[i]);
    printf("Burst Time:");
    scanf("%d", &burst_time[i]);
    temp[i] = burst_time[i];
}
printf("Enter Time Quantum:");
scanf("%d", &time_quantum);
printf("Process ID\tBurst Time\tTurnaround Time\t Waiting Time\n");
for(total = 0, i = 0; x != 0;)
{
    if(temp[i] <= time_quantum && temp[i] > 0)
    {
        total = total + temp[i];
        temp[i] = 0;
        counter = 1;
    }
    else if(temp[i] > 0)
    {
        temp[i] = temp[i] - time_quantum;
        total = total + time_quantum;
    }
    if(temp[i] == 0 && counter == 1)

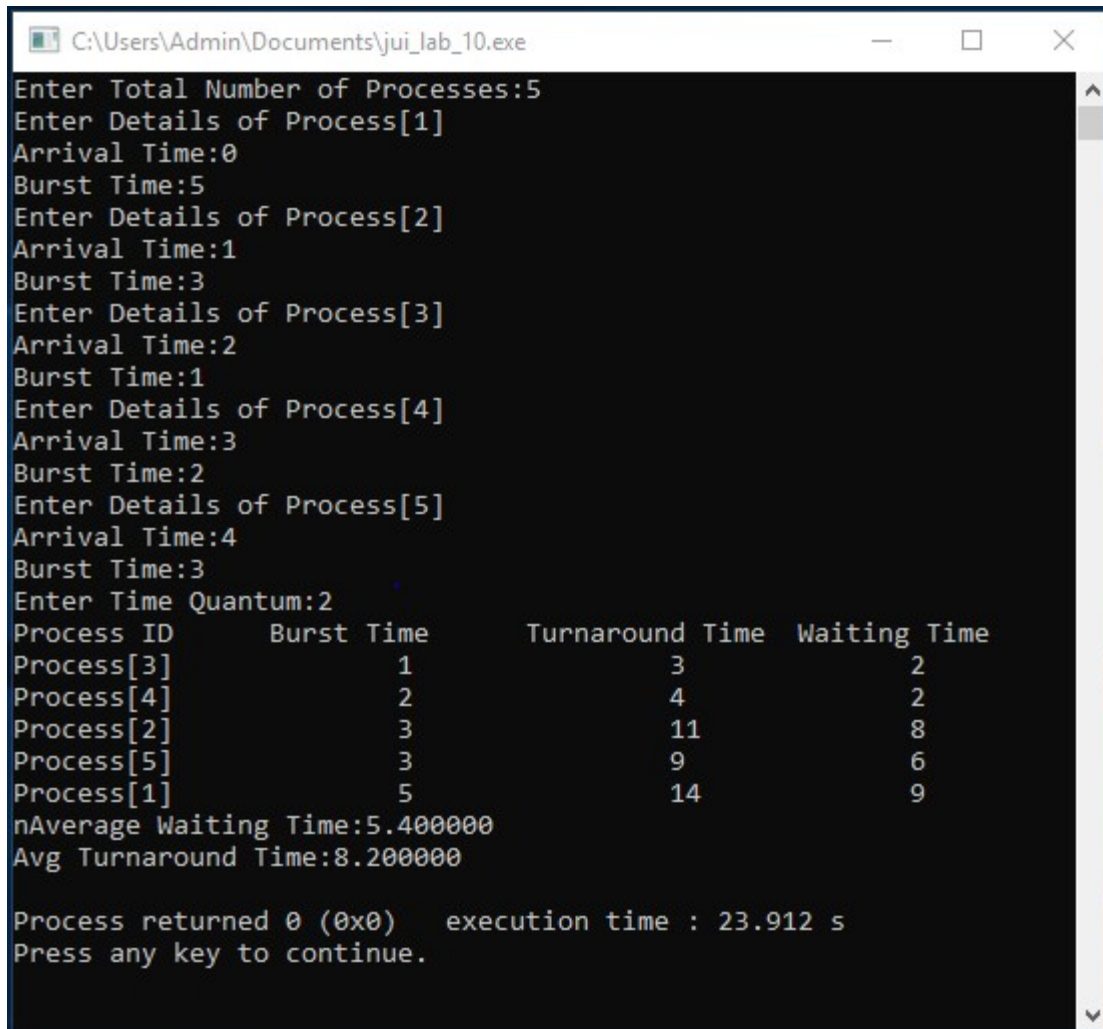
```

```

    {
        x--;
        printf("Process[%d]\t\t%d\t\t %d \t\t%d\n", i + 1, burst_time[i], total -
arrival_time[i], total - arrival_time[i] - burst_time[i]);
        wait_time = wait_time + total - arrival_time[i] - burst_time[i];
        turnaround_time = turnaround_time + total - arrival_time[i];
        counter = 0;
    }
    if(i == limit - 1)
    {
        i = 0;
    }
    else if(arrival_time[i + 1] <= total)
    {
        i++;
    }
    else
    {
        i = 0;
    }
}
average_wait_time = wait_time * 1.0 / limit;
average_turnaround_time = turnaround_time * 1.0 / limit;
printf("\nAverage Waiting Time:%f\n", average_wait_time);
printf("Avg Turnaround Time:%f\n", average_turnaround_time);
return 0;
}

```

### Output :



```
C:\Users\Admin\Documents\jui_lab_10.exe
Enter Total Number of Processes:5
Enter Details of Process[1]
Arrival Time:0
Burst Time:5
Enter Details of Process[2]
Arrival Time:1
Burst Time:3
Enter Details of Process[3]
Arrival Time:2
Burst Time:1
Enter Details of Process[4]
Arrival Time:3
Burst Time:2
Enter Details of Process[5]
Arrival Time:4
Burst Time:3
Enter Time Quantum:2
Process ID      Burst Time      Turnaround Time  Waiting Time
Process[3]      1              3                2
Process[4]      2              4                2
Process[2]      3              11               8
Process[5]      3              9                6
Process[1]      5              14               9
nAverage Waiting Time:5.400000
Avg Turnaround Time:8.200000

Process returned 0 (0x0)   execution time : 23.912 s
Press any key to continue.
```

### Discussion :

The above algorithm has been implemented using C language. In the above code, we ask the user to enter the number of processes and arrival time and burst time for each process. We then calculate the waiting time and the turn around time using the round-robin algorithm. The main part here is calculating the turn around time and the waiting time. Turn around time is calculated by adding the total time taken and subtracting the arrival time. The waiting time is calculated by subtracting the arrival time and burst time from the total and adding it to the waiting time. This is how the round-robin scheduling takes place.

