# Mawlana Bhashani Science and Technology University

# Lab-Report

Report No: 10

Course code: ICT-3110

Course title:Operating Systems Lab

Date of Performance:14-09-2020

Date of Submission:19-09-2020

## Submitted by

Name: Sabikun Nahar Piya

ID:IT-18020

3rd year 1st semester

Session: 2017-2018

Dept. of ICT

MBSTU.

## Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

## Experiment No:10

## Experiment Name:Implementation of Rund Robin scheduling program

## Theory:

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. It is the oldest, simplest scheduling algorithm, which is mostly used for multitasking.In Round-robin scheduling, each ready task runs turn by turn only in a cyclic queue for a limited time slice. This algorithm also offers starvation free execution of processes.

## Implementation:

step 1: Declare the array size.

step 2: Get the number of elements to be inserted.

step 3: Get the value.

step 4: Set the time sharing system with preemption.

step 5: Define quantum is defined from 10 to 100ms.

step 6: Declare the queue as a circular.

step 7: Make the CPU scheduler goes around the ready queue allocating CPU to eachprocess for the time interval specified.

step 8: Make the CPU scheduler picks the first process and sets time to interrupt afterquantum expired dispatches the process.

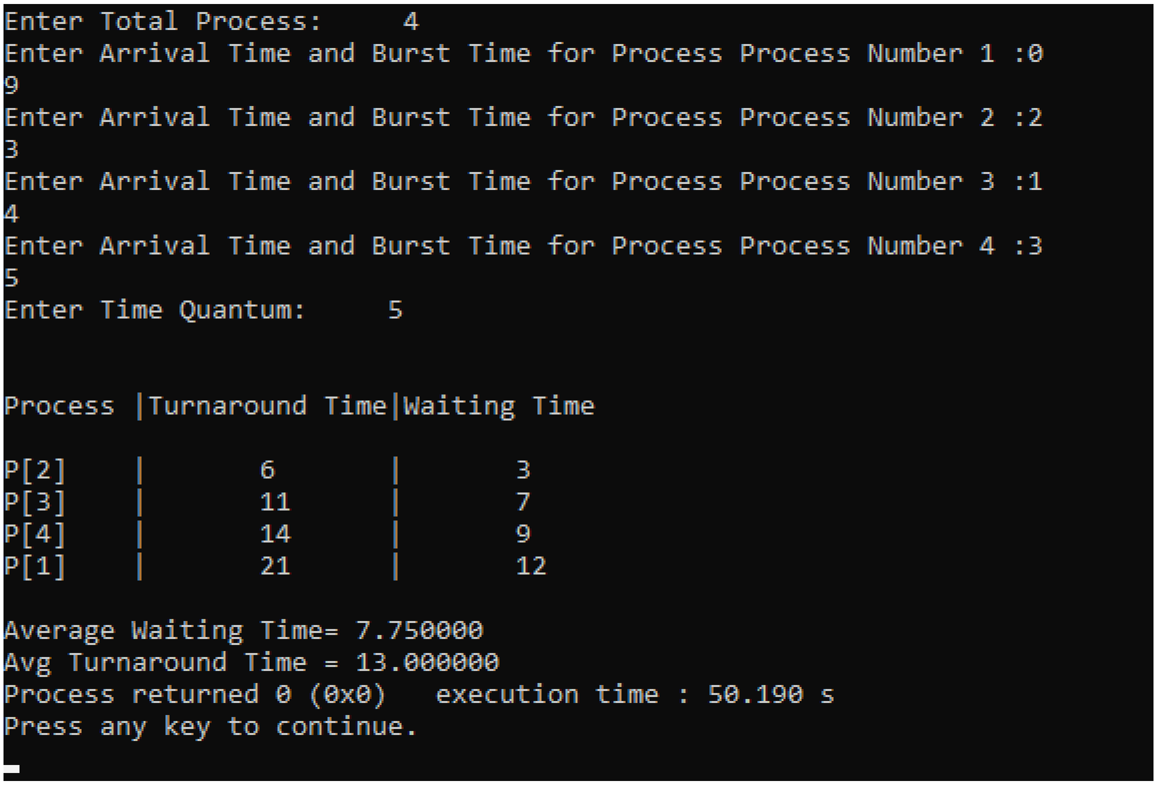
step 9: If the process has burst less than the time quantum than the process releasesthe CPU

## Working process:

Code for Rund Robin scheduling algorithm:

```
#include<stdio.h>
int main()
{
 int count,j,n,time,remain,flag=0,time_quantum;
 int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
 printf("Enter Total Process:\t ");
 scanf("%d",&n);
 remain=n;
 for(count=0;count<n;count++)
 {
  printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
  scanf("%d",&at[count]);
  scanf("%d",&bt[count]);
  rt[count]=bt[count];
 }
 printf("Enter Time Quantum:\t");
 scanf("%d",&time_quantum);
 printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
 for(time=0,count=0;remain!=0;)
 {
  if(rt[count]<=time_quantum && rt[count]>0)
  {
   time+=rt[count];
   rt[count]=0;
   flag=1;
  }
  else if(rt[count]>0)
  {
   rt[count]-=time_quantum;
   time+=time_quantum;
  }
  if(rt[count]==0 && flag==1)
  {
   remain--;
   printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
   wait_time+=time-at[count]-bt[count];
   turnaround_time+=time-at[count];
   flag=0;
  }
  if(count==n-1)
   count=0;
   else if(at[count+1]<=time)
     count++;
   else
     count=0;
  }
  printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
  printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);
  return 0;
}
```

## Output:



```
Enter Total Process:     4
Enter Arrival Time and Burst Time for Process Process Number 1 :0
9
Enter Arrival Time and Burst Time for Process Process Number 2 :2
3
Enter Arrival Time and Burst Time for Process Process Number 3 :1
4
Enter Arrival Time and Burst Time for Process Process Number 4 :3
5
Enter Time Quantum:     5


Process |Turnaround Time|Waiting Time

P[2]    |       6       |       3
P[3]    |       11      |       7
P[4]    |       14      |       9
P[1]    |       21      |       12

Average Waiting Time= 7.750000
Avg Turnaround Time = 13.000000
Process returned 0 (0x0)   execution time : 50.190 s
Press any key to continue.
```

## Discussion:

It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.One of the most commonly used technique in CPU scheduling as a core.It is preemptive as processes are assigned CPU only for a fixed slice of time at most.The disadvantage of it is more overhead of context switching.