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: 15/09/2020

Submitted by

Name: Md Ahadul Haque

ID:IT-18045

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 Round Robin Scheduling Algorithm.

Theory :

Round Robin Scheduling Algorithm is one of the simplest scheduling algorithm used in various operating systems for process scheduling and networks. The key idea is to allocate CPU to all processes in the same order for the same amount of time.It is also a preemptive scheduling algorithm famous for CPU Scheduling and used in various Operating Systems. It is better than other approaches like Shortest Job First algorithms considering that there is a guarantee that all processes will be completed at the cost of overall performance but it is better than brute force approach.

Implementation:

1. A fixed time is allotted to every process that arrives in the queue. This fixed time is known as time slice or time quantum.
2. The first process that arrives is selected and sent to the processor for execution. If it is not able to complete its execution within the time quantum provided, then an interrupt is generated using an automated timer.
3. The process is then stopped and is sent back at the end of the queue. However, the state is saved and context is thereby stored in memory. This helps the process to resume from the point where it was interrupted.
4. The scheduler selects another process from the ready queue and dispatches it to the processor for its execution. It is executed until the time Quantum does not exceed.
5. The same steps are repeated until all the process are finished.

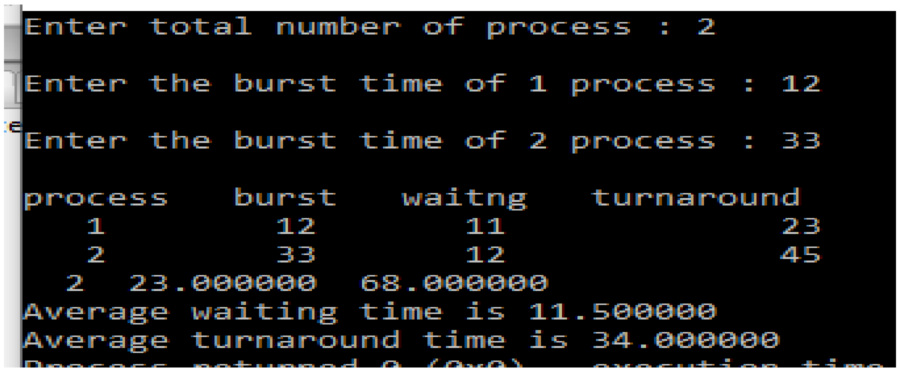
Working Process :

Code for Round Robin Scheduling Algorithm –

```
#include<stdio.h>
int main()
{
 int bursttime[100],waitingtime[100],turnaroundtime[100],b[100];
 int i,n,time,count=0;
 float totalwt=0,totalTT=0,avgwt,avgtt;

 printf("Enter total number of process : ");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  printf("\nEnter the burst time of %d process : ",i+1);
  scanf("%d",&bursttime[i]);
  b[i] = bursttime[i];
 }
 i=0;
 for(time=0;count!=n;time++)
 {
   while(bursttime[i] == 0)
   {
     i=(i+1)%n;
   }
   bursttime[i]--;
   if(bursttime[i]==0)
   {
     turnaroundtime[i] = time+1;
     count++;
   }
   i = (i+1)%n;
 }
 printf("\nprocess   burst   waitng   turnaround   ");
 for(i=0;i<n;i++)
 {  waitingtime[i] = turnaroundtime[i] - b[i];
    printf("\n   %d \t    %d \t     %d \t    %d",i+1,b[i],waitingtime[i],turnaroundtime[i]);
    totalwt = totalwt + waitingtime[i];
    totalTT = totalTT + turnaroundtime[i];
 }
 printf("\n  %d  %f  %f",n,totalwt,totalTT);
 avgwt = totalwt / n;
 avgtt = totalTT / n;
 printf("\nAverage waiting time is %f",avgwt);
 printf("\nAverage turnaround time is %f ",avgtt);
 return 0;
}
```

Output:



```
Enter total number of process : 2

Enter the burst time of 1 process : 12

Enter the burst time of 2 process : 33

process    burst    waitng    turnaround
   1         12        11              23
   2         33        12              45
  2  23.000000  68.000000
Average waiting time is 11.500000
Average turnaround time is 34.000000
```

Discussion :

The efficiency of the system is decreased if the quantum value is low as frequent switching takes place.The system may become unresponsive if the quantum value is high. Starvation rarely occurs in this process. Unlike other algorithms, it gives equal priority to all processes.