Mawlana Bhashani Science and Technology University

Lab -Report

Report No:10
Course code: ICT-3110
Course title: Operating System Lab
Date of Performance:16-09-2020
Date of Submission:23-09-2020

Submitted by

Name: Ruku Shikder
ID: IT-18057
3th 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

- What is Round Robin Scheduling Algorithm?
- How to Implement in c?

Round Robin Scheduling Algorithm:

Round Robin Scheduling is a scheduling algorithm used by the system to schedule CPU utilization. This is a preemptive algorithm. There exist a fixed time slice associated with each request called the quantum. The job scheduler saves the progress of the job that is being executed currently and moves to the next job present in the queue when a particular process is executed for a given time quantum.

No process will hold the CPU for a long time. The switching is called a context switch. It is probably one of the best scheduling algorithms. The efficiency of this algorithm depends on the quantum value.

- The queue structure in ready queue is of First In First Out (FIFO) type.
- A fixed time is allotted to every process that arrives in the queue. This fixed time is known as time slice or time quantum.
- 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.
- 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.
- 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.
- The same steps are repeated until all the process are finished.

Algorithm:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue and time quantum (or) time slice

Step 3: For each process in the ready Q, assign the process id and accept the

CPU burst time

Step 4: Calculate the no. of time slices for each process where

No. of time slice for process(n) = burst time process(n)/time slice

Step 5: If the burst time is less than the time slice then the no. of time slices =1.

Step 6: Consider the ready queue is a circular Q, calculate

Waiting time for process(n) = waiting time of process(n-1)+ burst time of process(n-1) + the time difference in getting the CPU from process(n-1)

Turnaround time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

Step 7: Calculate

Average waiting time = Total waiting Time / Number of process

Average Turnaround time = Total Turnaround Time / Number of process Step 8: Stop the process

Implementation in c:

Code:

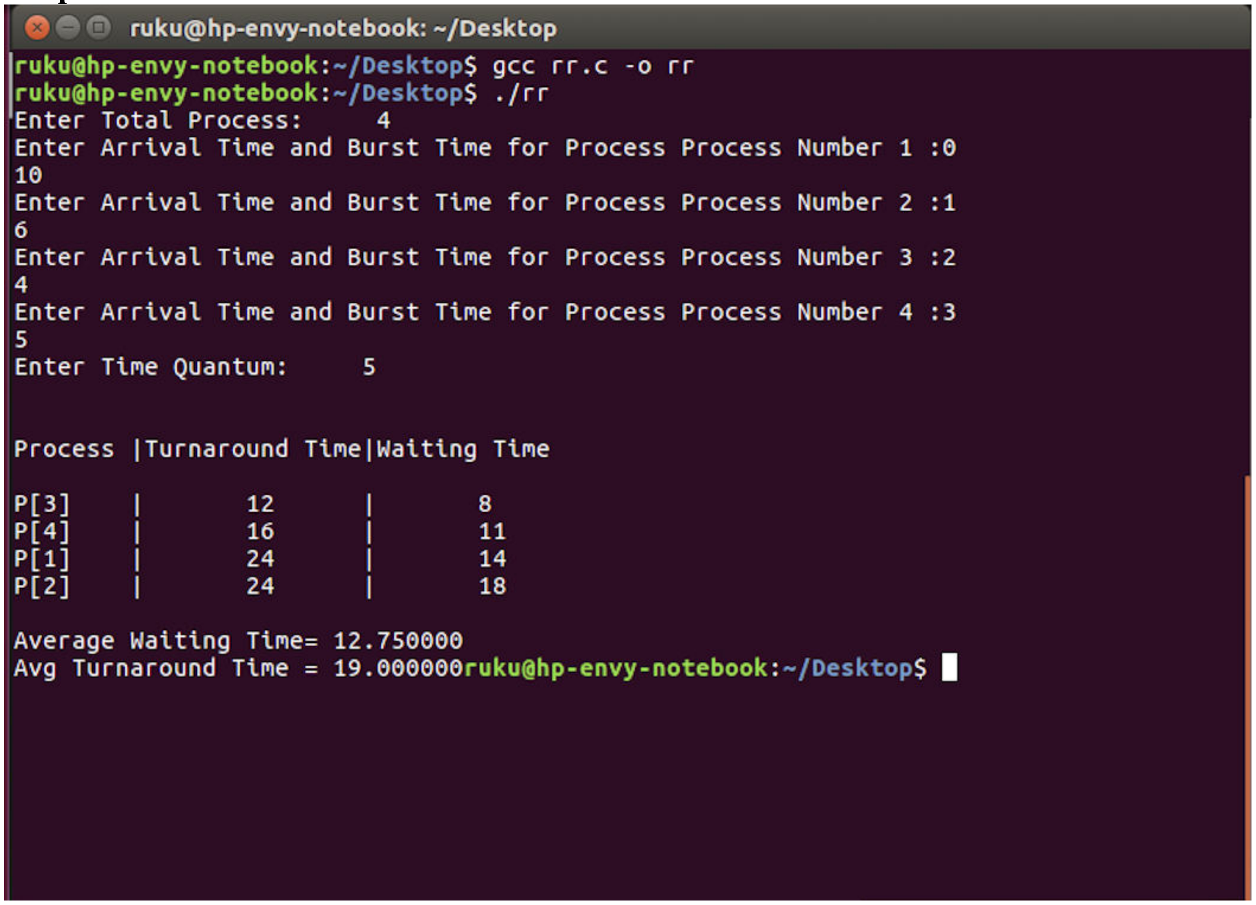
```
#include<stdio.h>

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:



```
ruku@hp-envy-notebook: ~/Desktop
ruku@hp-envy-notebook:~/Desktop$ gcc rr.c -o rr
ruku@hp-envy-notebook:~/Desktop$ ./rr
Enter Total Process:     4
Enter Arrival Time and Burst Time for Process Process Number 1 :0
10
Enter Arrival Time and Burst Time for Process Process Number 2 :1
6
Enter Arrival Time and Burst Time for Process Process Number 3 :2
4
Enter Arrival Time and Burst Time for Process Process Number 4 :3
5
Enter Time Quantum:     5


Process |Turnaround Time|Waiting Time

P[3]    |       12      |       8
P[4]    |       16      |       11
P[1]    |       24      |       14
P[2]    |       24      |       18

Average Waiting Time= 12.750000
Avg Turnaround Time = 19.000000ruku@hp-envy-notebook:~/Desktop$ 
```

Discussion:

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 t0 the waiting time. This is how the round-robin scheduling takes plaLow overhead for decision making.Unlike other algorithms, it gives equal priority to all processes.Starvation rarely occurs in this process.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.