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: 13/9/2020

Submitted by

Name: Mahfuza Talukdar

ID:IT-18009

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 most popular scheduling algorithm which can actually be implemented in most of the operating systems. This is the preemptive version of first come first serve scheduling. The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a cyclic way. A certain time slice is defined in the system which is called time quantum**.** Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will terminate else the process will go back to the ready queue and waits for the next turn to complete the execution.

1. It can be actually implementable in the system because it is not depending on the burst time.
2. It doesn't suffer from the problem of starvation or convoy effect.
3. All the jobs get a fare allocation of CPU.

Corresponding Code :

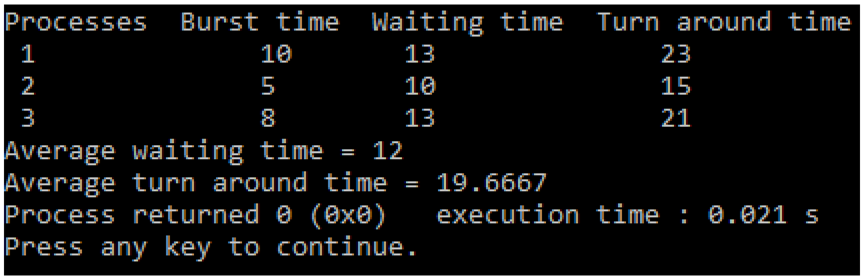
```
// C++ program for implementation of RR scheduling

#include<iostream>

using namespace std;


// Function to find the waiting time for all

// processes

void findWaitingTime(int processes[], int n,

                        int bt[], int wt[], int quantum)

{

    // Make a copy of burst times bt[] to store remaining

    // burst times.

    int rem_bt[n];
```

```
for (int i = 0 ; i < n ; i++)
        rem_bt[i] = bt[i];

int t = 0; // Current time

// Keep traversing processes in round robin manner
// until all of them are not done.
while (1)
{
        bool done = true;

        // Traverse all processes one by one repeatedly
        for (int i = 0 ; i < n; i++)
        {
                // If burst time of a process is greater than 0
                // then only need to process further
                if (rem_bt[i] > 0)
                {
                        done = false; // There is a pending process

                        if (rem_bt[i] > quantum)
                        {
                                // Increase the value of t i.e. shows
                                // how much time a process has been processed
                                t += quantum;

                                // Decrease the burst_time of current process
```

```
                    // by quantum

                    rem_bt[i] -= quantum;

                }

                // If burst time is smaller than or equal to

                // quantum. Last cycle for this process

                else

                {

                    // Increase the value of t i.e. shows

                    // how much time a process has been processed

                    t = t + rem_bt[i];

                    // Waiting time is current time minus time

                    // used by this process

                    wt[i] = t - bt[i];

                    // As the process gets fully executed

                    // make its remaining burst time = 0

                    rem_bt[i] = 0;

                }

            }

        }

        // If all processes are done

        if (done == true)

        break;

    }
```

```
}

// Function to calculate turn around time
void findTurnAroundTime(int processes[], int n,
                                                        int bt[], int wt[], int tat[])
{
        // calculating turnaround time by adding
        // bt[i] + wt[i]
        for (int i = 0; i < n ; i++)
                tat[i] = bt[i] + wt[i];
}

// Function to calculate average time
void findavgTime(int processes[], int n, int bt[],
                                                                        int quantum)
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;

        // Function to find waiting time of all processes
        findWaitingTime(processes, n, bt, wt, quantum);

        // Function to find turn around time for all processes
        findTurnAroundTime(processes, n, bt, wt, tat);

        // Display processes along with all details
        cout << "Processes "<< " Burst time "
                << " Waiting time " << " Turn around time\n";
        // Calculate total waiting time and total turn
        // around time
        for (int i=0; i<n; i++)
        {
                total_wt = total_wt + wt[i];
                total_tat = total_tat + tat[i];
                cout << " " << i+1 << "\t\t" << bt[i] <<"\t "
                        << wt[i] <<"\t\t " << tat[i] <<endl;
        }

        cout << "Average waiting time = "
                << (float)total_wt / (float)n;
        cout << "\nAverage turn around time = "
                << (float)total_tat / (float)n;
}

// Driver code
int main()
{
        // process id's
        int processes[] = { 1, 2, 3};
        int n = sizeof processes / sizeof processes[0];

        // Burst time of all processes
        int burst_time[] = {10, 5, 8};
        // Time quantum

        int quantum = 2;

        findavgTime(processes, n, burst_time, quantum);

        return 0;

}
```

Output :



```
Processes  Burst time  Waiting time  Turn around time
 1              10       13              23
 2              5        10              15
 3              8        13              21
Average waiting time = 12
Average turn around time = 19.6667
Process returned 0 (0x0)   execution time : 0.021 s
Press any key to continue.
```

Discussion :

From this lab, we have learnt that how to implement Round Robin scheduling algorithm using C language. We can solve any problem of this algorithm in future.