Mawlana Bhashani Science and Technology University

Lab-Report

Report No:

Course code: ICT-3110

Course title: Operating Systems Lab

Date of Performance:

Date of Submission:

Submitted by

Name: Hasibul Islam Imon

ID:IT-18047

3th year 1ndsemester

Session: 2017-18

Dept. of ICT

MBSTU.

Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

.

Lab report name:Implementation of Round Robin Scheduling Algorithm

Objectives:

- Round robin is a pre-emptive algorithm
- The CPU is shifted to the next process after fixed interval time, which is called time quantum/time slice.
- The process that is preempted is added to the end of the queue.
- Round robin is a hybrid model which is clock-driven
- Time slice should be minimum, which is assigned for a specific task that needs to be processed. However, it may differ OS to OS.
- It is a real time algorithm which responds to the event within a specific time limit.
- Round robin is one of the oldest, fairest, and easiest algorithm.
- Widely used scheduling method in traditional OS.

Question No.1 What is Round Robin Scheduling Algorithm?

Answer: The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turns. 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.

Question No.2 How to Implement in c?

Answer:

```
// 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
```

Conclusion:

Round robin is a CPU scheduling algorithm that is designed especially for time sharing systems. It is more like a FCFS scheduling algorithm with one change that in Round Robin processes are bounded with a quantum time

size. A small unit of time is known as Time Quantum or Time Slice. Time quantum can range from 10 to 100 milliseconds. CPU treat ready queue as a circular queue for executing the processes with given time slice. It follows preemptive approach because fixed time are allocated to processes. The only disadvantage of it is overhead of context switching.