Project Report: Round Robin CPU Scheduling Simulation

By Amir Amit & Nicholas Voronetskis

## **Table of Contents**

* Introduction
* CPU Scheduling Overview
* Round Robin Scheduling Algorithm
* Project Implementation
* Analyzing Program Output
* Conclusion
* References

## 

## 

## 

## 

## **1. Introduction**

The goal of this project is to use a Java application to mimic round-robin CPU scheduling. As input parameters, the application receives user-inputted data that includes process details and a time quantum. The round-robin method is then used to imitate the execution of processes, and performance measures including CPU utilization, throughput, average waiting time, and average turnaround time are computed.

How to Run Program:

First, we go into the mainRR class. Within this class, you have the flexibility to adjust the values of "processes," "burst," and "arrival time." These variables represent different aspects of the processes being simulated. Additionally, you can modify the "quantum" value to tailor the time slice for the Round Robin scheduling algorithm. After making any necessary adjustments, simply execute the code by pressing the "run" button to observe how the changes impact the program's behavior and output.

public class MainRR {

public static void main(String[] args) {

// Process IDs

int process[] = {1, 2, 3, 4};

// Arrival time

int arrivalTime[] = {0, 1, 0, 2};

// Burst time

int burstTimeOfProcess[] = {5, 7, 2, 6};

int sizeOfProcess = process.length;

int quantum\_valueOfProcess = 10;

RoundRobinScheduler.*findavgTimeOfProcess*(process, arrivalTime, sizeOfProcess, burstTimeOfProcess, quantum\_valueOfProcess);

}

}

## **2. CPU Scheduling Overview**

CPU scheduling is the process of determining which processes should be assigned to the CPU at any given time. The goal is to maximize CPU utilization, throughput, and responsiveness while minimizing waiting times and turnaround times. We accomplish this by allowing all processes to run for a set time, allowing for all processes to get a chance to run and repeat till all processes are completed.

## **3. Round Robin Scheduling Algorithm**

Round Robin is a preemptive CPU scheduling algorithm where each process is assigned a fixed time quantum. Processes are executed cyclically, with each process running for a time quantum and then being preempted to allow another process to run. The processes are set into a queue in order of their arrival, or when they are set out of the running state and still require more time to run before completion. This continues until all processes are completed.

## **4. Project Implementation**

The project is implemented in Java programming language. The main components of the implementation include

* Compile the program with any ide, with the Main class.
* Implementing a FIFO queue for process scheduling.
* Maintaining a clock to timestamp events.
* Simulating process execution using the round-robin algorithm.
* Calculating performance

## 

## **5. Analyzing Program Output**

The Process list from the code-

quantum\_valueOfProcess = 4;

public class MainRR {

public static void main(String[] args) {

// Process IDs

int process[] = {1, 2, 3, 4};

// Arrival time

int arrivalTime[] = {0, 1, 0, 2};

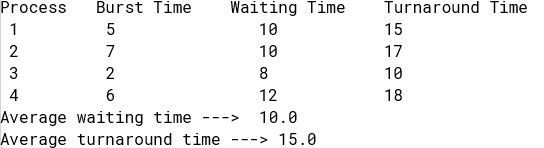
// Burst time

int burstTimeOfProcess[] = {5, 7, 2, 6};

int sizeOfProcess = process.length;

int quantum\_valueOfProcess = 4;

RoundRobinScheduler.*findavgTimeOfProcess*(process, arrivalTime, sizeOfProcess, burstTimeOfProcess, quantum\_valueOfProcess);

}

}

Test 2: quantum\_valueOfProcess = 2;

public class MainRR {

public static void main(String[] args) {

// Process IDs

int process[] = {1, 2, 3, 4};

// Arrival time

int arrivalTime[] = {0, 1, 0, 2};

// Burst time

int burstTimeOfProcess[] = {5, 7, 2, 6};

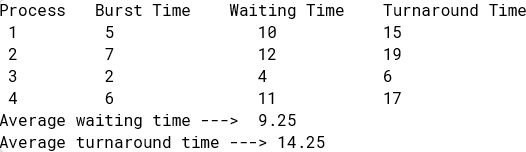
int sizeOfProcess = process.length;

int quantum\_valueOfProcess = 2;

RoundRobinScheduler.*findavgTimeOfProcess*(process, arrivalTime, sizeOfProcess, burstTimeOfProcess, quantum\_valueOfProcess);

}

}



Test 3: quantum\_valueOfProcess = 6;

public class MainRR {

public static void main(String[] args) {

// Process IDs

int process[] = {1, 2, 3, 4};

// Arrival time

int arrivalTime[] = {0, 1, 0, 2};

// Burst time

int burstTimeOfProcess[] = {5, 7, 2, 6};

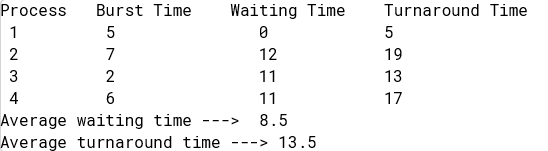
int sizeOfProcess = process.length;

int quantum\_valueOfProcess = 6;

RoundRobinScheduler.*findavgTimeOfProcess*(process, arrivalTime, sizeOfProcess, burstTimeOfProcess, quantum\_valueOfProcess);

}

}



Test 4: quantum\_valueOfProcess = 8;

public class MainRR {

public static void main(String[] args) {

// Process IDs

int process[] = {1, 2, 3, 4};

// Arrival time

int arrivalTime[] = {0, 1, 0, 2};

// Burst time

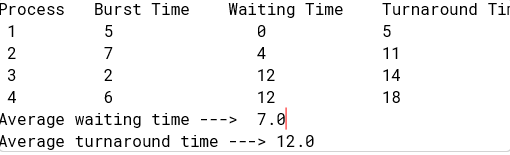
int burstTimeOfProcess[] = {5, 7, 2, 6};

int sizeOfProcess = process.length;

int quantum\_valueOfProcess = 8;

RoundRobinScheduler.*findavgTimeOfProcess*(process, arrivalTime, sizeOfProcess, burstTimeOfProcess, quantum\_valueOfProcess);

}



Test 5: quantum\_valueOfProcess = 10;

public class MainRR {

public static void main(String[] args) {

// Process IDs

int process[] = {1, 2, 3, 4};

// Arrival time

int arrivalTime[] = {0, 1, 0, 2};

// Burst time

int burstTimeOfProcess[] = {5, 7, 2, 6};

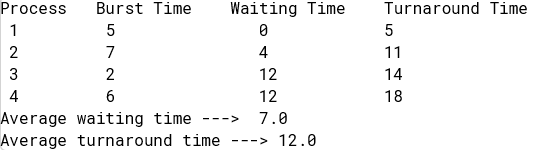
int sizeOfProcess = process.length;

int quantum\_valueOfProcess = 10;

RoundRobinScheduler.*findavgTimeOfProcess*(process, arrivalTime, sizeOfProcess, burstTimeOfProcess, quantum\_valueOfProcess);

}

}



From What is shown for these 5 tests, increasing the quantum value in round-robin scheduling tends to increase turnaround time unless it significantly exceeds the burst time of all processes. As we increased the quantum value from 1 to 2 to 4, the turnaround time gradually started increasing. Once we made the value 6, all the processes' burst times were smaller than that of the quantum value which caused improved responsiveness but tended to have higher context switches. While larger values may reduce overhead but result in poorer responsiveness. Interestingly, increasing the quantum value has been shown to decrease average wait time.

## **7. Conclusion**

In conclusion, the Round Robin CPU scheduling simulation provides insights into the performance of the algorithm under different time quantum values. By analyzing the program output, we can understand how adjusting the time quantum impacts CPU utilization, throughput, and other performance metrics.