Round Robin Scheduler Simulation

Operating System Concepts

CPE 409

Prepared for

Dr. Rukkaya Umar
Computer Engineering
Nile University of Nigeria

Prepared By

David Adeshina Arungbemi
191203012
Computer Engineering
Nile University of Nigeria

23 December 2022

TABLE OF CONTENTS

LIST OF FIGURES

AIM

To design an program to simulate Round Robin Scheduler Algorithm in real time with JAVA.

OBJECTIVES

1. Create a Round Robin class to hold most of backend.
2. Create a Process class to be called by Round Robin class to instantiate new processes.
3. Configure program to simulate in real time.
4. Create a GUI with JAVA Swing to interface with backend.

INTRODUCTION

Round Robin Scheduler Algorithm is one of the many CPU scheduling algorithm that exist. What makes it different from others is its time quantum. Time quantum's determine how long each process is allowed to run. Round robin scheduling algorithms can be very fast and efficient if the right quantum time is chosen. The program created simulates Round Robin Scheduling in real time. It also includes a GUI, allowing the user to input and observe the results.

THEORY

The CPU determines the manner and sequence in which processes should be carried out through a process known as **CPU scheduling**. Preemptive and non-preemptive CPU scheduling are the two categories. When "scheduling" these processes, the CPU takes into account the following factors: CPU utilisation, throughput, turnaround time, waiting time, and response time [1]. Throughput is a measure of how much can be processed in a given time, waiting time is the total time spent a process spends in the ready queue and response time is the time spent between ready state and first time getting to CPU [1].

CPU scheduling is of two types: Preemptive and Non-Preemptive Scheduling.

Preemptive scheduling takes into account that some processes could have a higher priority and as a result, must be carried out before processes with a lower priority. Preemptive scheduling allocates CPU resources to a process for a brief period of time before returning those resources to be used by another process (the next in execution) [1]. The process is returned to the ready state, where it will stay until it has another opportunity to execute, if it had not yet finished its execution [1].

Non-preemptive scheduling only allows for the execution of new processes after the current process has finished. Until it is terminated or moved to the process waiting state, the process retains CPU resources (CPU time) [1]. A process that the CPU is currently running is not stopped until it is finished. After the process has finished running, the CPU selects the following process from the ready queue (the queue in which all processes that are ready for execution are stored) [1].

Round Robin Algorithm is one of the preemptive CPU scheduling algorithm which gives each process a set amount of time quantum to complete the task at hand.

Every process in this algorithm is carried out in a cyclic manner, which means that processes that still have time left in their burst after the time quantum has expired are sent back to the ready state and must wait for their turn to carry out the execution until it ends. This processing is done in FIFO order, which denotes that operations are carried out according to the principle of first-come, first-served [2].

Round Robin Scheduling is of the following steps

1. Processes are added to ready queue in FIFO.
2. The process begins, and the queue's head is popped out to be executed.
3. The process keeps running until its quantum time.

4. If the process's burst time (the amount of time it takes to complete execution) is less than or equal to the process's quantum time, the process has completed and terminates.
5. If burst time exceeds quantum time, the process is halted and sent to the back of the ready queue with its remaining quantum time.
6. The next process is then popped out and the entire steps begins again.

Advantages of Round Robin include

1. It is starvation free.
2. Each process gets equal priority(in fixed time Round Robin, however there exists dynamic Round Robin with changing quantum time).
3. It can be easily implemented.

Disadvantages of Round Robin include

1. If quantum time is too high, it becomes First Come First Serve(a non preemptive algorithm. - each process will have to wait for the leading process to complete before it runs itself). It can lead to starvation.
2. Higher context switching due to too long quantum time.

PROCEDURE

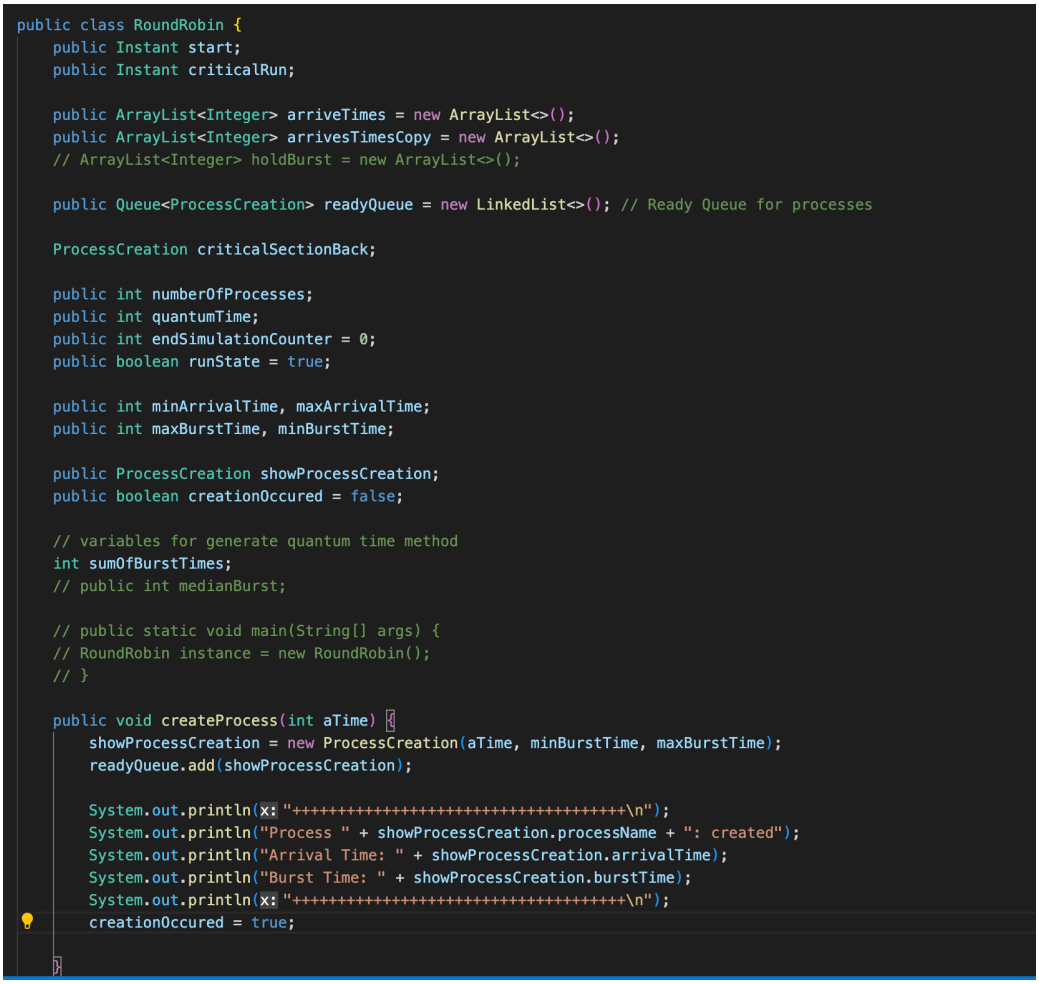
1. Since JAVA is OOP based, the program was divided into three classes, the **GUI** class, **RoundRobin** class, **ProcessCreation** class.
2. **RoundRobin** class was where most of the backend was setup, it consisted of methods such as a create process method-which created a process, it did this by creating an instance of **ProcessCreation** class storing the newly created object in the ready queue.
3. **ProcessCreation** class consisted of attributes that described a process such as arrival time, burst time, time quantum, etc.
4. There was also the quantum time generation method built to create a quantum time for each process by finding the average of all burst times in

the ready queue. As long as the process was in the ready queue its quantum time could change depending on processes added or removed.

5. Next, the determine arrival time method was built to generate arrival times and check queue method to decide when to activate the create process method based on generated arrival times.
6. To create a real time simulation, the **Instant** class that came with JAVA was made use of. By spacing instants in time, it could be determined when each process was to be complete using if and else statements. The difference between time instants was calculated with imported **Duration** class.

```
class ProcessCreation { // built for creating new processes
    public int burstTime;
    public int arrivalTime;
    public int quantumNumber;
    public int originalBurstTime;

    public int waitingTime;
    public int turnAroundTime;
    public int exitTime;
    static int nameUpdate = 0;
    public int processName = 0; // E.g Process 1

    public ProcessCreation(int arrivalTime, int minBurst, int maxBurst) {
        this.burstTime = ThreadLocalRandom.current().nextInt(minBurst, maxBurst + 1); // sets the burstTime to a random
                                                                                      // time in
        // 1 to 5 seconds. variable is subject to change

        this.originalBurstTime = this.burstTime; // keeps the original burst time, does not change
        // this.arrivalTime = ThreadLocalRandom.current().nextInt(1, 6);
        this.arrivalTime = arrivalTime;
        nameUpdate += 1;
        this.processName = nameUpdate;

    }
}
```

Fig. 1.1 - Entire ProcessCreation Class



```
public class RoundRobin {
    public Instant start;
    public Instant criticalRun;

    public ArrayList<Integer> arriveTimes = new ArrayList<>();
    public ArrayList<Integer> arrivesTimesCopy = new ArrayList<>();
    // ArrayList<Integer> holdBurst = new ArrayList<>();

    public Queue<ProcessCreation> readyQueue = new LinkedList<>(); // Ready Queue for processes

    ProcessCreation criticalSectionBack;

    public int numberOfProcesses;
    public int quantumTime;
    public int endSimulationCounter = 0;
    public boolean runState = true;

    public int minArrivalTime, maxArrivalTime;
    public int maxBurstTime, minBurstTime;

    public ProcessCreation showProcessCreation;
    public boolean creationOccured = false;

    // variables for generate quantum time method
    int sumOfBurstTimes;
    // public int medianBurst;

    // public static void main(String[] args) {
    // RoundRobin instance = new RoundRobin();
    // }

    public void createProcess(int aTime) {
        showProcessCreation = new ProcessCreation(aTime, minBurstTime, maxBurstTime);
        readyQueue.add(showProcessCreation);

        System.out.println(x: "++++++++++++++++++++++++++++++++++\n");
        System.out.println("Process " + showProcessCreation.processName + ": created");
        System.out.println("Arrival Time: " + showProcessCreation.arrivalTime);
        System.out.println("Burst Time: " + showProcessCreation.burstTime);
        System.out.println(x: "++++++++++++++++++++++++++++++++++\n");
        creationOccured = true;

    }
```

Fig 1.2 - A section of RoundRobin Class

7. The GUI class consisted mostly of GUI components, the result shown below in figure 4.3. The upper section consisted of a configuration section to set generate a number of processes, range of burst times and range of arrival times all set by the user. The generated arrival time section showed the expected arrival time(Note: All arrival times, burst times, etc were in seconds).
8. The middle section, consisted of smaller windows: a time quantum window: to show changing quantum times in the ready queue as new processes were added or moved, the critical section: to show process running at the moment, Logs: describing all things related to processes being run, process create section: to show the latest process created and

a process exited section: to show the latest process that had been completed.

9. Finally, in the lower section was the summary section to show a summary of the simulation(average arrival time and average turnaround time).

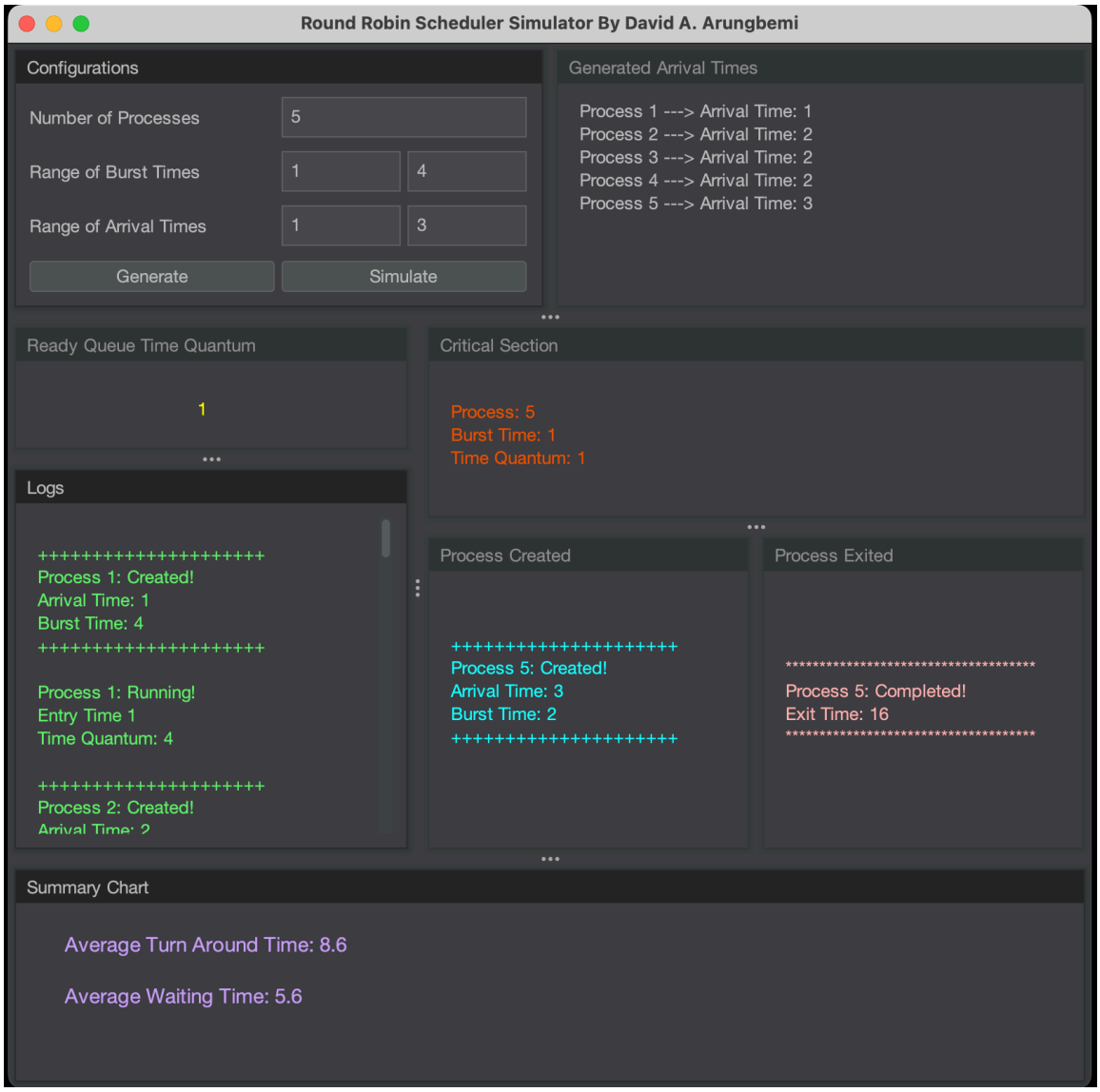


Fig. 1.3 - GUI Interface for Round Robin Simulator

10. The GUI section was easy to use and different text colours were added to identify each section.
11. The code is provided in the folder(Code is too large to be added in report).

DISCUSSION

The program would run from the GUI class by calling “new GUI()” in the main method of class GUI(main class). The GUI would then appear. User would enter required inputs and press generate. Assuming wrong inputs were entered, an popup displaying an error would appear. If no errors, the arrival times would generated automatically. Next, the simulate button would be entered to begin simulation. However, most of the simulation will be run in GUI class due to issues with getting results from the other class(if simulation was running from backend class(RoundRobin class), there would be no way to get results from GUI while it was running).

During the creation of the program, the following challenges were met: specifying time for each process in seconds(problem was solved with imported Instant and Duration class). Showing results of GUI in real time, while backend was running, the GUI would freeze and not display anything until the simulation was complete. After simulation, results would then appear. The following was tried to solve the problem: redirecting output of terminal to GUI directly, etc but no success. Issue seems to be with threads but there lack of sufficient knowledge to fix the problem. At the end, it was realised that the issue had to do with threads.

After in-depth research into threads involving JAVA and JAVA swing, JAVA SwingWorker class was introduced. The imported class, allowed for concurrency by keeping the background tasks(bulk of simulation) in a background thread called the Worker thread and Swing components on Event Dispatch thread. Having different threads for each part made it possible to interact with and update GUI in real time.

Finally, there was a plan for a gantt chart but could not find a proper way to express it in GUI, hence it became the summary section providing average turn around time and average waiting time as shown in figure 4.3 above.

CONCLUSION

At the end of this project, I was successfully able to design and create a program capable of running a round robin scheduler in real time.

REFERENCE(S)

[1] T. Mathur, “CPU scheduling,” *Scaler Topics*, 28-Apr-2022. [Online]. Available: https://www.scaler.com/topics/operating-system/cpu-scheduling/. [Accessed: 23-Dec-2022].

[2] V. Sharma, “What is Round Robin Scheduling in OS?,” *Scaler Topics*, 29-Aug-2022. [Online]. Available: https://www.scaler.com/topics/round-robin-scheduling-in-os/. [Accessed: 23-Dec-2022].