Round Robin CPU Scheduling Project

By: Nuran Ghoneim

Course: CSCI 330 - M01

CPU Scheduling:

CPU scheduling helps maximize CPU utilization by allowing one process to use CPU while all other processes are on hold. CPU scheduling maintains order by going through the ready queue and selecting a process to be executed. All other processes must wait until the CPU is free to be rescheduled. When a process is on hold, it is in the waiting queue where it's waiting for an event to occur. When a process is in the ready queue, it is waiting to be chosen by the CPU scheduler so it can be executed. The dispatcher also has a role in the CPU scheduling process. The dispatcher is a module that gives control of the CPU's core to the process that was selected by the CPU scheduler. There are different algorithms that the CPU scheduler can use to select a process:

- First Come First Serve(FCFS) Scheduling
- Shortest-Job-First(SJF) Scheduling
- Priority Scheduling
- Round Robin(RR) Scheduling
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

Round Robin:

Every process is given a specific time frame to execute. When new processes are requested, they are added to the ready queue. The scheduler picks a process from the queue that either has a higher priority or based on the first come first serve algorithm. When a process is selected to be executed, a timer is set to define the time this process has to complete its task. If the process is not completed within the given time frame, it will be preempted and brought back to the end of the queue. Round robin helps minimize waiting time and turnaround time and maximizes throughput and CPU utilization.

Program Implementation:

I have 3 classes: Main, Process, and Scheduler.

Main Class: Has the scanner for the user to input the name of the file and the time quantum. It also scans for the file once the user inputs it to make sure it's there, then executes the code.

Process Class: This class just initializes all the variables I'm going to be using.

Scheduler Class: This is the round-robin processing class. This class has all the calculations for the functions that are being used such as Turnaround Time and Waiting Time. It will run through and make sure all the parameters are defined in the excel file then run so all the processes are completed.

```java
//Nuran Ghoneim

package cpu_scheduler;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

public class Main {
    public static void main(String[] arg) throws IOException {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter filename: ");
        String fileName = sc.nextLine();
        System.out.print("Enter Time Quantum: ");
        int timequantum = sc.nextInt();
        Scanner input = new Scanner(new File(fileName));
        input.useDelimiter("(,|\\n)");  // seperating each output into another line so each process prints out on its own line
        input.nextLine();
        ArrayList<Process> csvFile = new ArrayList<>();
        while (input.hasNext()) {
            Process process = new Process(input.nextInt(), input.nextInt(), input.nextInt());
            csvFile.add(process);
        }

        Scheduler myScheduler = new Scheduler(csvFile, timequantum);
        myScheduler.rrfuntions();
        sc.close();
        input.close();
    }
}
```

```java
//Nuran Ghoneim
package cpu_scheduler;

public class Process {
    public int processId;
    public int arrivalTime;
    public int serviceTime;
    public int burstTime;
    public int completionTime;

    public Process(int processId, int arrivalTime, int burstTime) {
        this.processId = processId;
        this.arrivalTime = arrivalTime;
        this.burstTime = burstTime;
        this.serviceTime = 0;
    }

}
```

```java
//Nuran Ghoneim
package cpu_scheduler;

import java.util.*;
import java.util.*;

public class Scheduler {
    int timer;
    ArrayList<Process> listOfJobs;
    ArrayList<Process> readyQ;
    ArrayList<Process> endOfProcesses;
    int timeQuantum;
    int contextSwitch;

    Process cpu;
    int counter;

    public Scheduler(ArrayList<Process> listOfJobs, int timeQuantum) {
        this.timeQuantum = timeQuantum;
        this.listOfJobs = listOfJobs;
    }

    public void rrfuntions() {
        timer = 0;
        contextSwitch = 0;
        cpu = null;
        readyQ = new ArrayList<>();
        endOfProcesses = new ArrayList<>();

        while (!readyQ.isEmpty() || !listOfJobs.isEmpty() || cpu != null) {

            for (int i = 0; i < listOfJobs.size(); i++) {   // adding to readyQueue
                if (listOfJobs.get(i).arrivalTime == timer) {
                    readyQ.add(listOfJobs.remove(i));
                }
            }

            if (cpu == null) {    // adding to CPU
                cpu = readyQ.remove(0);
            }

            counter++;
            cpu.serviceTime++;
```

```
            if (cpu.burstTime == cpu.serviceTime) { // process is complete
                cpu.completionTime = timer; // completion time is set
                endOfProcesses.add(cpu);
                cpu = null;
                contextSwitch++;
                counter = 0;
            } else if (counter == timeQuantum) { // if it exceeds the timeQuantum
                readyQ.add(cpu);
                cpu = null;
                contextSwitch++;
                counter = 0;
            }
            timer++;

        }

        double turnAroundTimeTotal = 0.0;
        double waitingTimeTotal = 0.0;
        double utilizationTotal = 0.0;

    //calculating the functions below
        for (int i = 0; i < endOfProcesses.size(); i++) {
            turnAroundTimeTotal += endOfProcesses.get(i).completionTime - endOfProcesses.get(i).arrivalTime;
            waitingTimeTotal += (endOfProcesses.get(i).completionTime - endOfProcesses.get(i).arrivalTime)
                    - endOfProcesses.get(i).burstTime;
            utilizationTotal += endOfProcesses.get(i).burstTime;
        }
        double avgTurnAroundTime = turnAroundTimeTotal / endOfProcesses.size();
        double avgWaitingTime = waitingTimeTotal / endOfProcesses.size();
        double cpuUtilization = (utilizationTotal - (contextSwitch * .01)) / timer;
        double throughput = (double) endOfProcesses.size() / timer;

        System.out.println("");
        System.out.println("CPU Utilization: " + cpuUtilization);
        System.out.println("Throughput: " + throughput);
        System.out.println("Average Waiting Time: " + avgWaitingTime);
        System.out.println("Average Turnaround Time: " + avgTurnAroundTime);

    }

}
```

Output:

```
Enter filename: ExcelFile.csv
Enter Time Quantum: 1

CPU Utilization: 0.99
Throughput: 0.2
Average Waiting Time: 7.5
Average Turnaround Time: 12.5
```

```
Enter filename: ExcelFile.csv
Enter Time Quantum: 2

CPU Utilization: 0.9945
Throughput: 0.2
Average Waiting Time: 7.25
Average Turnaround Time: 12.25
```

```
Enter filename: ExcelFile.csv
Enter Time Quantum: 3

CPU Utilization: 0.9960000000000001
Throughput: 0.2
Average Waiting Time: 7.5
Average Turnaround Time: 12.5
```

```
Enter filename: ExcelFile.csv
Enter Time Quantum: 4

CPU Utilization: 0.9964999999999999
Throughput: 0.2
Average Waiting Time: 8.0
Average Turnaround Time: 13.0
```

```
Enter filename: ExcelFile.csv
Enter Time Quantum: 5

CPU Utilization: 0.9970000000000001
Throughput: 0.2
Average Waiting Time: 6.0
Average Turnaround Time: 11.0
```

Instructions on how to run my code:

1. On Eclipse create a new Project and a new package called cpu_scheduler.
2. Import the files into the source folder.
3. Download the CSV onto your desktop and drag it into your source folder but outside of the package.
4. Run the program.