Amitoj Johar
CSCI 330
ID: 1069158
CPU Scheduling Project

What is CPU Scheduling?

The purpose of CPU Scheduling is to give all processes in the CPU a chance to run and not let only one process to use the CPU resources and run forever. CPU scheduling follows scheduling criteria: CPU utilization, throughput, turnaround time, waiting time and response time. The goal of CPU scheduling is to maximize CPU utilization, maximize throughput, minimize turnaround time, minimize waiting time and minimize response time. To achieve these goals, we have some CPU Scheduling algorithms such as

1. First Come First Serve
2. Shortest Time Left
3. Round Robin
4. Priority

How Round Robin Works

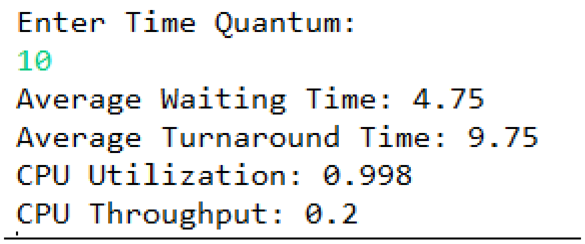
Round Robin is an algorithm where each process gets a small unit of time known as the time quantum and it is roughly 10-100 milliseconds. After a process has used the unit of time it was given that process is halted and added to the end of the ready queue so it can finish completing when it gains access to the CPU again. In this algorithm, the time quantum should be larger than the context switch time otherwise more time will be dedicated towards switching processes and less time will be used on CPU utilization. A timer is used schedule the next process after every time quantum.

Implementation

The program I developed is to show a simulation of Round Robin scheduling by demonstrating the time a number of processes would take to run on CPU. The program consists of 4 Java classes: Process, MainClass, CPU, and RoundRobin. The program is designed to read a list of processes in a CSV file and tell the user average waiting time, average turnaround time, CPU utilization, and CPU throughput. The Process class allows me to create Process objects that accept three parameters which are the process ID, burst time and arrival time. This allows me to accept my processes and then store them in an ArrayList so the arrival times of all processes can be compared and the ones with the least arrival time are put into the ready queue. The CPU class holds an object of type Process which will keep track of the process currently running in CPU. The timer variable in the CPU class is designed to increment while the Process object in CPU is being used. The processes that are completed are added to a new ArrayList so it can be used to retrieve information that can be used to calculate the timings we need. The program output first asks user to enter the value of their desired time quantum. The program takes the time quantum and calculates the average waiting time, average turnaround time, CPU utilization and CPU throughput. To determine the average waiting time, I summed the waiting times of all processes and divided it by the number of processes. Likewise, I summed the turnaround times of all processes and divided by the number of processes. For the CPU utilization, I took the total number of context switches with the amount of time a process will get to switch and subtracted it from the total burst time, and lastly I divided the result by the counter. To find the CPU throughput, the number of processes was divided by the counter.

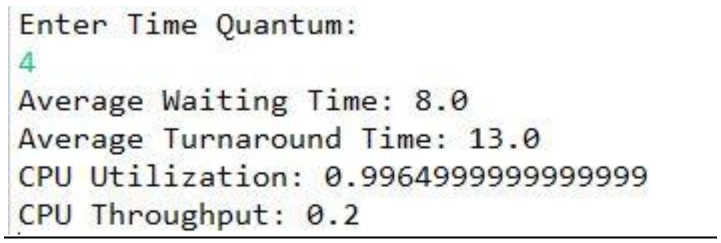
Resulting Output

1. Time Quantum is 2:



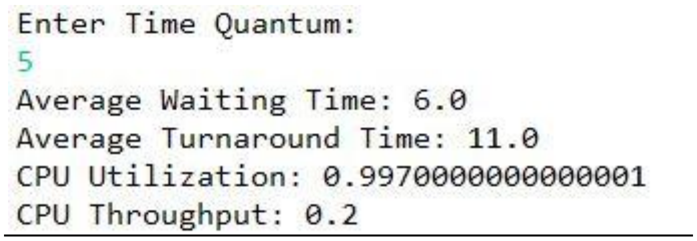
```
Enter Time Quantum:
10
Average Waiting Time: 4.75
Average Turnaround Time: 9.75
CPU Utilization: 0.998
CPU Throughput: 0.2
```

2. Time Quantum is 4:



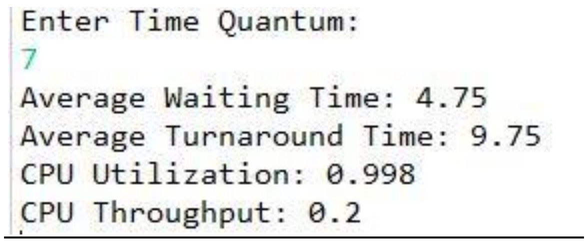
```
Enter Time Quantum:
4
Average Waiting Time: 8.0
Average Turnaround Time: 13.0
CPU Utilization: 0.9964999999999999
CPU Throughput: 0.2
```

3. Time Quantum is 5:



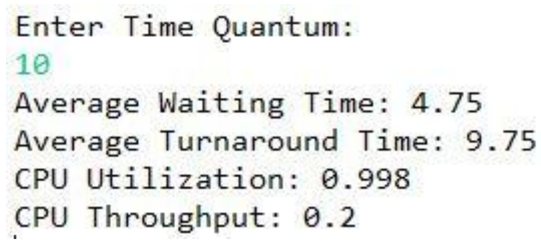
```
Enter Time Quantum:
5
Average Waiting Time: 6.0
Average Turnaround Time: 11.0
CPU Utilization: 0.9970000000000001
CPU Throughput: 0.2
```

4. Time Quantum is 7:



```
Enter Time Quantum:
7
Average Waiting Time: 4.75
Average Turnaround Time: 9.75
CPU Utilization: 0.998
CPU Throughput: 0.2
```

5. Time Quantum is 10:



```
Enter Time Quantum:
10
Average Waiting Time: 4.75
Average Turnaround Time: 9.75
CPU Utilization: 0.998
CPU Throughput: 0.2
```

<u>Instructions on how to run code</u>
To run code:
Use any Java compiler and run code to see output in the window.

Source Code:

RoundRobin.java

```
import java.util.*;
public class RoundRobin
{
	int timeQuantum;
	int counter;
	int contextSwitchTime = 0;
	ArrayList<Process> readyqueue = new ArrayList<>();
	ArrayList<Process>rrprocesses;
	CPU cpu1 = new CPU();
	ArrayList<Process>finishedProcesses = new ArrayList<>();
	double waitingTime = 0;
	double turnAroundTime = 0;
	double totalBurstTime = 0;
	public RoundRobin(int timeQuantum, ArrayList<Process>rrprocesses)
	{
		this.timeQuantum = timeQuantum;
		this.rrprocesses = rrprocesses;
	}
	public void algorithm()
	{
		counter = 0;
		while(!readyqueue.isEmpty() || !rrprocesses.isEmpty() || 
cpu1.inCPU != null)
		{
			//System.out.println(counter);
			for(int i = 0; i < rrprocesses.size(); i++)
			{

	//System.out.println(rrprocesses.get(i).processArrivalTime);
				if(counter == 
rrprocesses.get(i).processArrivalTime)
					readyqueue.add(rrprocesses.remove(i));
			}
			//System.out.println("  " + rrprocesses.size());
			//System.out.println("  " + readyqueue.size());

			// Make A CPU
			//Check if CPU is null
			if(cpu1.inCPU == null){
				cpu1.inCPU = readyqueue.remove(0);
			}

			cpu1.timeSpent++;
```

```
                cpu1.inCPU.processServiceTime++;
                // Process 0 = readyqueue.remove(0)
                // Use this to CPU
                //what if the process finishes
                //empty out CPU
                if(cpu1.inCPU.burstTime ==
cpu1.inCPU.processServiceTime){
                     finishedProcesses.add(cpu1.inCPU);
                     cpu1.inCPU.completionTime = counter;
                     cpu1.inCPU= null;
                     contextSwitchTime++;
                     cpu1.timeSpent = 0;
                }
                //else if time quantum equals time spent
                //if(timeQuantum == timeSpent)
                //kick out the process, wait again at the back of the
readyqueue

                else if(cpu1.timeSpent == timeQuantum){
                     readyqueue.add(cpu1.inCPU);
                     cpu1.inCPU = null;
                     contextSwitchTime++;
                     cpu1.timeSpent = 0;
                }
                counter++;
           }

           for(int i =0; i<finishedProcesses.size(); i++)
           {
                turnAroundTime +=
finishedProcesses.get(i).completionTime -
finishedProcesses.get(i).processArrivalTime;
                waitingTime +=
(finishedProcesses.get(i).completionTime -
finishedProcesses.get(i).processArrivalTime) -
finishedProcesses.get(i).burstTime;
                totalBurstTime += finishedProcesses.get(i).burstTime;
           }
           System.out.println("Average Waiting Time: " +
waitingTime/finishedProcesses.size());
           System.out.println("Average Turnaround Time: " +
turnAroundTime/finishedProcesses.size());
           System.out.println("CPU Utilization: " + ((totalBurstTime -
(contextSwitchTime*0.01))/counter));
           System.out.println("CPU Throughput: " +
((double)finishedProcesses.size()/counter));
```

```
	}

}
```
CPU.java
```
public class CPU
{
	public CPU()
	{
		inCPU = null;
	}
	public Process inCPU;
	public int timeSpent;
}
```

Process.java
```
public class Process
{
	public String processID;
	public int processArrivalTime;
	public int processServiceTime;
	public int completionTime;
	public int burstTime;
	public Process(String processID, int processArrivalTime , int
burstTime)
	{
		this.processArrivalTime = processArrivalTime;
		this.processID = processID;
		this.burstTime = burstTime;
		processServiceTime = 0;
	}
}
```

MainClass.java

```
import java.io.*;
import java.util.*;

public class MainClass
{
      public static void main(String [] args)
      {
            int timeQuantum;
            ArrayList<Process> processes = new ArrayList<>();
            System.out.println("Enter Time Quantum: ");
            Scanner user = new Scanner(System.in);
            timeQuantum = user.nextInt();
            try{
                  FileReader reader = new FileReader("processes.csv");
                  Scanner read = new Scanner(reader);
                  read.nextLine();
                  while (read.hasNextLine())
                  {
                        String line = read.nextLine();
                        String[] numbers = line.split(",");
                        processes.add(new Process(numbers[0],
Integer.parseInt(numbers[1]), Integer.parseInt(numbers[2])));
                  }
                  RoundRobin round = new RoundRobin(timeQuantum,
processes);
                  round.algorithm();
                  read.close();
                  user.close();
            } catch (IOException E)
            {
                  System.out.print("An exception was thrown!");
            }
      }
}
```