Alexander Khemai
ID: 1072075
CSCI 330-M02
1 December 2017

<u>How to run the code:</u>

- Use the command line
- Run in a Java IDE such as Eclipse

<u>What is CPU Scheduling?</u>

CPU scheduling is the basis of multiprogrammed operating systems. It is a task which allows a process to use the CPU while there is a process waiting to use the CPU, because of the CPU being occupied. The goal of CPU scheduling is to make the system fast, efficient and reliable. Also, the number of jobs and CPU utilization should be maximized, while turnaround time, waiting time and context switching should be minimized.

<u>How does Round Robin Work?</u>

Round Robin is one of the various CPU scheduling algorithms. Round Robin is a preemptive process scheduling algorithm. Each process gets a unit of CPU time, called a time quantum. After this time has elapsed, the process is preempted and added to the end of the ready queue.

<u>Program Implementation and Analysis</u>

The Round Robin scheduling algorithm was simulated in Java. There are a total of 4 classes: Test.java, CPU.java, Process.java, and RoundRobinSim.java. The CSV file containing process ID's, arrival times and burst times is read in the Test class and is treated as processes. The Process class handles the creation objects that are type Process and accepts three parameters that are process ID, burst time and arrival time. Those processes are then stored in an ArrayList and then is taken as a parameter for the RoundRobinSim class.

The RoundRobinSim class is where the actual Round Robin simulation occurs. The class takes in the processes arraylist and the time quantum as parameters. The arraylist is of type process, since it represents a group of processes. The processes are then put into the ready queue based on arrival time, which is also an arraylist. The CPU class acts as a holder of an object with the type process and also contains a counter that counts the time a process spends executing. Once the simulation has ended the scheduling, the program prints statistics about it and that includes the average waiting time, average turnaround time, CPU utilization represented as a percentage and throughput.

Performance Calculation

CPU Utilization: (total burst time - (number of context switches * context switch time) * 100/ total time

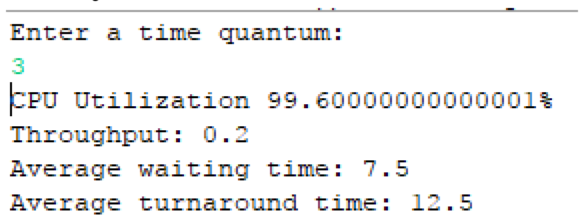
Throughput: number of processes / total time

Average waiting time: total waiting time of all processes / number of processes

Average turnaround time: total turnaround time / number of processes

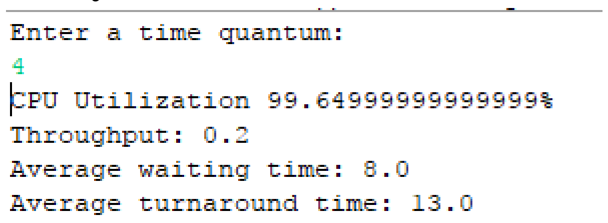
Output:

Time Quantum 3



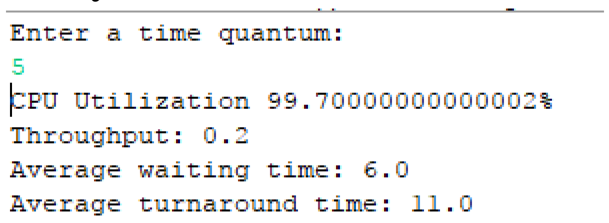
```
Enter a time quantum:
3
CPU Utilization 99.60000000000001%
Throughput: 0.2
Average waiting time: 7.5
Average turnaround time: 12.5
```

Time Quantum 4



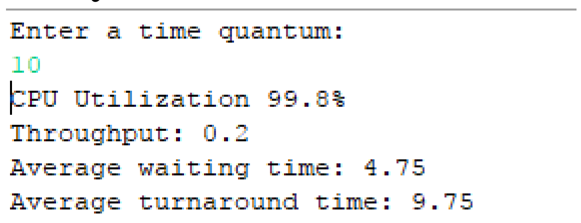
```
Enter a time quantum:
4
CPU Utilization 99.64999999999999%
Throughput: 0.2
Average waiting time: 8.0
Average turnaround time: 13.0
```

Time Quantum 5



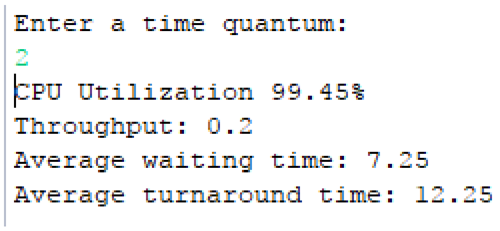
```
Enter a time quantum:
5
CPU Utilization 99.70000000000002%
Throughput: 0.2
Average waiting time: 6.0
Average turnaround time: 11.0
```

Time Quantum 10



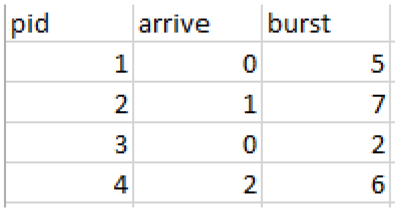
```
Enter a time quantum:
10
CPU Utilization 99.8%
Throughput: 0.2
Average waiting time: 4.75
Average turnaround time: 9.75
```

Time Quantum 2



```
Enter a time quantum:
2
CPU Utilization 99.45%
Throughput: 0.2
Average waiting time: 7.25
Average turnaround time: 12.25
```

CSV file containing processes



pid	arrive	burst
1	0	5
2	1	7
3	0	2
4	2	6

Source code:

```
public class Process
{
	public int processID, arrivalTime, burstTime, serviceTime,
completionTime;

	public Process(int processID, int arrivalTime , int
burstTime)
	{
		this.processID = processID;
		this.burstTime = burstTime;
		this.arrivalTime = arrivalTime;

	}
	public void setBurstTime(int burstTime)
	{
		this.burstTime = burstTime;
	}
	public int getBurstTime()
	{
		return burstTime;
	}
	public void setArrivalTime(int arrivalTime)
	{
		this.arrivalTime = arrivalTime;
	}
	public int getArrivalTime()
	{
```

```
          return arrivalTime;
     }
}
//cpu class to hold processes
public class CPU
{
     public Process running;
     public int timeSpent; //counter

     public CPU()
     {
          running = null;
     }

}

//Round robin
import java.util.*;

public class RoundRobinSim
{
     public ArrayList<Process> readyqueue;
     public ArrayList<Process> processes;
     public ArrayList<Process> info; //arraylist with the info
of process after it has finished
     public int timeQuantum;
     public int currentTime;
     public double avgWaitTime; //end print
     public double avgTurnAroundTime; //end print
     public double throughput;
     public int contextSwitchTime;
     public double totalBurstTime;


     public RoundRobinSim(ArrayList<Process> processes, int
timeQuantum)
     {
          this.processes = processes;
          this.timeQuantum = timeQuantum;
     }
     public void Scheduling()
     {

          CPU cp = new CPU(); //new cpu
          readyqueue = new ArrayList<>(); //readyqueue created
          info = new ArrayList<>();
          currentTime = 0;
          avgTurnAroundTime = 0;
          contextSwitchTime = 0;
          totalBurstTime = 0;
          while(!readyqueue.isEmpty() || !processes.isEmpty() ||
cp.running != null){ //adding processes to readyqueue
               for(int i=0; i < processes.size(); i++) //for
loop to gather processes and add it to ready queue
               {
                    if(processes.get(i).arrivalTime ==
currentTime)
                         readyqueue.add(processes.remove(i));
               }
               if(cp.running == null)
                    cp.running = readyqueue.remove(0); //return
the first element of ready queue

               cp.timeSpent++; //increment timeSpent
               cp.running.serviceTime++; //increment serviceTime

               if(cp.running.burstTime ==
cp.running.serviceTime)  //if burstTime is the same as
serviceTime
               {
                    info.add(cp.running);
                    cp.running.completionTime = currentTime;
                    cp.running = null;
                    cp.timeSpent = 0;
                    contextSwitchTime++;
               }
               else if(timeQuantum == cp.timeSpent) //if time
quantum has expired
               {
                    readyqueue.add(cp.running);
                    cp.running = null;
                    cp.timeSpent = 0;
                    contextSwitchTime++;
                }
                currentTime++;
            }

            for(int i = 0; i < info.size(); i++)
            {
                avgTurnAroundTime += info.get(i).completionTime -
info.get(i).arrivalTime;
                avgWaitTime += (info.get(i).completionTime -
info.get(i).arrivalTime) - info.get(i).burstTime;
                totalBurstTime += info.get(i).burstTime;
            }
            avgTurnAroundTime = avgTurnAroundTime / info.size();
            avgWaitTime = avgWaitTime / info.size();
            throughput = (double)info.size() / currentTime;

            System.out.println("CPU Utilization " +
((totalBurstTime - (contextSwitchTime * 0.01)) / currentTime) *
100 + "%");
            System.out.println("Throughput: " + (throughput));
            System.out.println("Average waiting time: " +
(avgWaitTime));
            System.out.println("Average turnaround time: " +
(avgTurnAroundTime));

        }
}

//main class
import java.util.*;
import java.io.*;

public class Test
{
        public int timeQuantum;
        public String csv;
        public BufferedReader br;
        public String line = "";
        static Scanner kb = new Scanner(System.in);
        static int tq;
```

```
	public ArrayList<Process> getArraylist() throws IOException
	{
		ArrayList<Process> proc = new ArrayList();
		br = new BufferedReader(new FileReader(csv)); //read
csv file
		br.readLine();
		while ((line = br.readLine()) != null)
		{
			String[] arr = line.split(","); //split commas
			Process p = new Process(Integer.parseInt(arr[0]),
Integer.parseInt(arr[1]) ,Integer.parseInt(arr[2])); //convert
string to int using the index of array
			proc.add(p);

		}

		return proc;
	}
	public static void main(String[] args) throws IOException
	{
		System.out.println("Enter a time quantum: ");
		tq = kb.nextInt();
		Test test = new Test ("input.csv", tq); //take csv
file and time quantum
		RoundRobinSim rr = new
RoundRobinSim(test.getArraylist(), test.timeQuantum);
		rr.Scheduling();
	}
	public Test (String csv, int timeQuantum)
	{
		this.csv = csv;
		this.timeQuantum = timeQuantum;
	}

}
```