Experiment No 5
Aim: Implementation of FCFS and Round Robin CPU Scheduling.
Class: SE Comp
Year: 2020-21
Performed by: Danyl Fernandes, 72
Date: 24-03-2021

Preemptive Scheduling:

Preemptive Scheduling is a CPU scheduling technique that works by dividing time slots of CPU to a given process. The time slot given might be able to complete the whole process or might not be able to. When the burst time of the process is greater than CPU cycle, it is placed back into the ready queue and will execute in the next chance. This scheduling is used when the process switches to ready state.

Algorithms that are backed by preemptive Scheduling are round-robin (RR), priority, SRTF (shortest remaining time first).

Non-Preemptive Scheduling:

Non-preemptive Scheduling is a CPU scheduling technique the process takes the resource (CPU time) and holds it till the process gets terminated or is pushed to the waiting state. No process is interrupted until it is completed, and after that processor switches to another process.

Algorithms that are based on non-preemptive Scheduling are non-preemptive priority, and the shortest Job first.

Comparison:

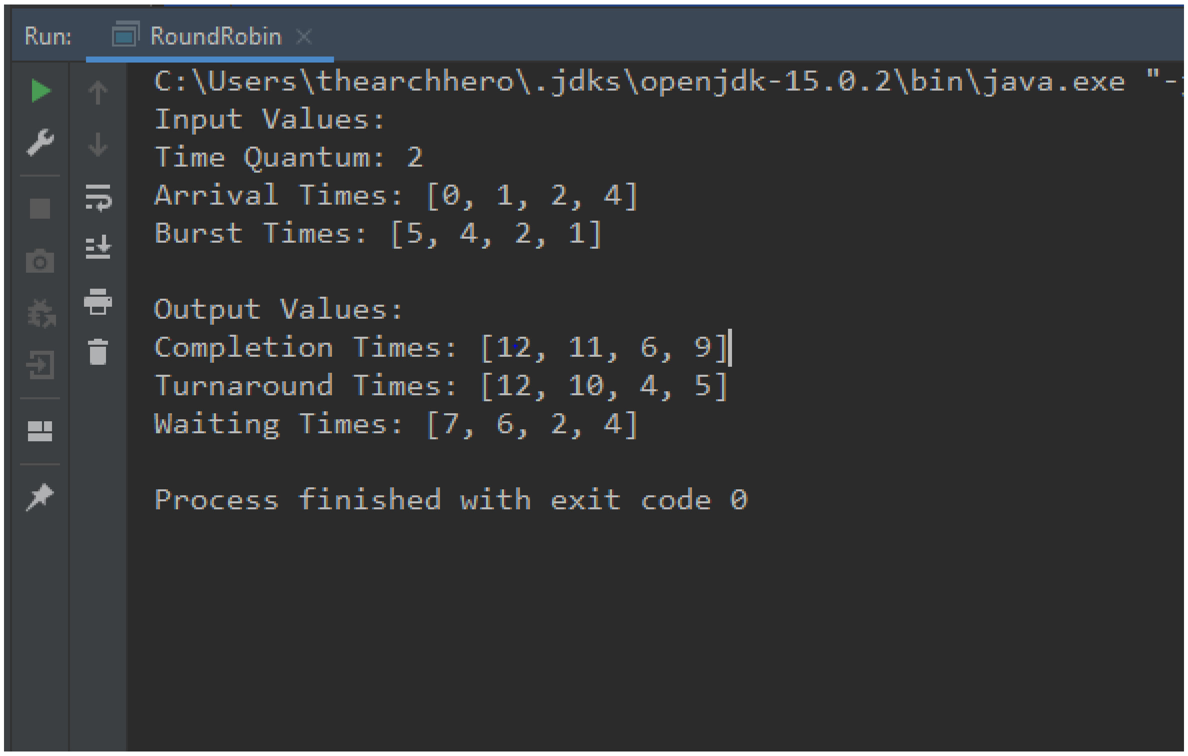
Preemptive Scheduling	Non-Preemptive Scheduling
Resources are allocated according to the cycles for a limited time.	Resources are used and then held by the process until it gets terminated.
The process can be interrupted, even before the completion.	The process is not interrupted until its life cycle is complete.
Starvation may be caused, due to the insertion of priority process in the queue.	Starvation can occur when a process with large burst time occupies the system.
Queue and remaining time overheads.	No such overheads are required.

Round Robin

```java
import java.util.*;

/**
*
* @author Danyl Fernandes
*
* A program demonstrating the Round-Robin CPU Scheduling algorithm
* */

public class RoundRobin {

   public void compute(int n, int timeQuantum, int[] arrivalTime, int[] burstTime) {

       Queue<Integer> readyQueue = new LinkedList<>();
       Queue<Integer> contextSwitchQueue = new LinkedList<>();

       int[] remainingTime = new int[n];

       // Copying burstTime into remainingTime
       if (n >= 0) System.arraycopy(burstTime, 0, remainingTime, 0, n);

       //int maxArrivalTime = Collections.max(Arrays.asList(arrivalTime));

       int[] completionTime = new int[n];
       int[] turnaroundTime = new int[n];
       int[] waitingTime = new int[n];

       int previousElapsedTime = 0;
       int elapsedTime = 0;

       while (true) {
           boolean tasksComplete = true;

           for (int process = 0; process < n; process++) {

               if (remainingTime[process] > 0) {
                   tasksComplete = false;

                   // If it's the first process, add it to the readyQueue
                   if (elapsedTime == 0) {
                       readyQueue.add(0);
                   }
```

```
                    // If new processes came in, add them to the readyQueue
                    // based on their arrivalTime
                    if (!(elapsedTime > n)) {
                        for (int i = 0; i < n; i++) {
                            if (arrivalTime[i] > previousElapsedTime && arrivalTime[i] <=
elapsedTime) {
                                readyQueue.add(i);
                            }
                        }
                    }

                    // Remove one process from the contextSwitchQueue
                    // add it to the readyQueue
                    if (!contextSwitchQueue.isEmpty()) {
                        readyQueue.add(contextSwitchQueue.remove());
                    }

                    // Execute one process from the ready queue
                    int processInReadyQueue = readyQueue.remove();

                    // If the process' burst time is greater than the timeQuantum, subtract
from the burstTime
                    // If the process' burst time is lesser than the timeQuantum, set its
burstTime1 to 0
                    if (remainingTime[processInReadyQueue] > timeQuantum) {

                        remainingTime[processInReadyQueue] -= timeQuantum;
                        previousElapsedTime = elapsedTime;
                        elapsedTime += timeQuantum;
                    } else {

                        previousElapsedTime = elapsedTime;
                        elapsedTime += remainingTime[processInReadyQueue];
                        remainingTime[processInReadyQueue] = 0;
                    }

                    // Was it complete?
                    // If yes, we got it's completion time
                    // If not, add it to the contextSwitchQueue
                    if (remainingTime[processInReadyQueue] == 0) {
                        completionTime[processInReadyQueue] = elapsedTime;
                    } else {
                        contextSwitchQueue.add(processInReadyQueue);
```

```java
                }
            }
        }

        if (tasksComplete) {
            break;
        }
    }

    // Compute and print TAT and WT
    for (int process = 0; process < n; process++) {
        turnaroundTime[process] = completionTime[process] - arrivalTime[process];
        waitingTime[process] = turnaroundTime[process] - burstTime[process];
    }

    System.out.println("Input Values:");
    System.out.println("Time Quantum: " + timeQuantum);
    System.out.println("Arrival Times: " + Arrays.toString(arrivalTime));
    System.out.println("Burst Times: " + Arrays.toString(burstTime) + "\n");

    System.out.println("Output Values:");
    System.out.println("Completion Times: " + Arrays.toString(completionTime));
    System.out.println("Turnaround Times: " + Arrays.toString(turnaroundTime));
    System.out.println("Waiting Times: " + Arrays.toString(waitingTime));

  }

  public static void main(String[] args) {
      new RoundRobin().compute(4, 2, new int[]{0, 1, 2, 4}, new int[]{5, 4, 2, 1});
  }
}
```

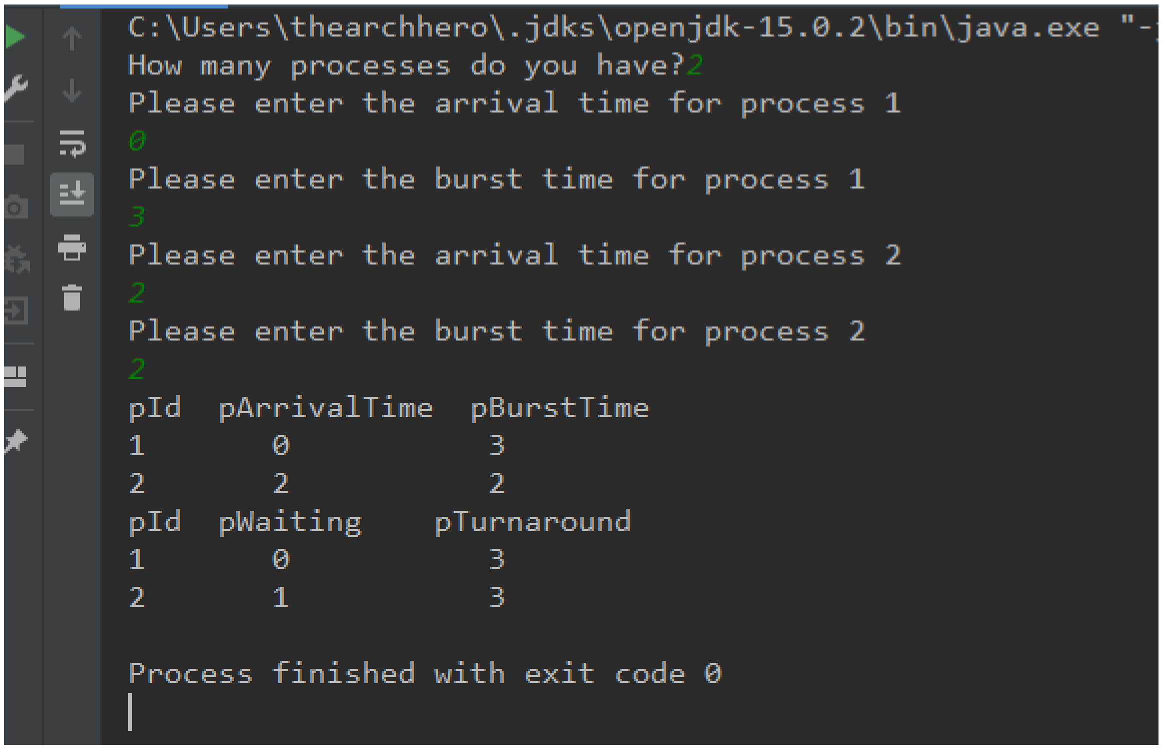


```
C:\Users\thearchhero\.jdks\openjdk-15.0.2\bin\java.exe "-
Input Values:
Time Quantum: 2
Arrival Times: [0, 1, 2, 4]
Burst Times: [5, 4, 2, 1]

Output Values:
Completion Times: [12, 11, 6, 9]
Turnaround Times: [12, 10, 4, 5]
Waiting Times: [7, 6, 2, 4]

Process finished with exit code 0
```

FCFS Scheduling:

```
import java.util.ArrayList;
import java.util.Scanner;

/**
 * @author Danyl Fernandes
 * */
class FCFS {

    public static void main(String[] args) {
        ArrayList<Process> processList = new ArrayList<Process>();

        System.out.print("How many processes do you have?");
        Scanner scanner = new Scanner(System.in);

        int totalNoOfProcesses = scanner.nextInt();

        //taking in data
        for (int i = 0; i < totalNoOfProcesses; i++) {

            System.out.println("Please enter the arrival time for process " + (i + 1));
            int processArrivalTime = scanner.nextInt();

            System.out.println("Please enter the burst time for process " + (i + 1));
            int processBurstTime = scanner.nextInt();

            processList.add(new Process((i + 1), processArrivalTime, processBurstTime));
        }

        // printing data that was input
        System.out.print("pId\t pArrivalTime  pBurstTime\n");
        for (Process process : processList) {
            process.printProcessData();
        }

        // calculating waiting times
        for (int i = 0; i < processList.size(); i++) {
            Process currentProcess = processList.get(i);
            int previousProcessBurstTime, previousProcessArrivalTime,
previousProcessWaitingTime;

            if (i == 0) {
                currentProcess.setWaitingTime(0);
                currentProcess.setTurnAroundTime(currentProcess.getBurstTime() +
currentProcess.getWaitingTime());
            } else {
                Process previousProcess = processList.get(i - 1);

                previousProcessArrivalTime = previousProcess.getArrivalTime();
                previousProcessWaitingTime = previousProcess.getWaitingTime();
                previousProcessBurstTime = previousProcess.getBurstTime();

                int previousProcessTotalTime = previousProcessArrivalTime +
previousProcessBurstTime + previousProcessWaitingTime;

                currentProcess.setWaitingTime(previousProcessTotalTime -
currentProcess.getArrivalTime());
                currentProcess.setTurnAroundTime(currentProcess.getBurstTime() +
currentProcess.getWaitingTime());
            }


        }

        System.out.print("pId\t pWaiting\t pTurnaround\n");
        for (Process process : processList) {
            process.printWaitData();
        }
    }
}

class Process {
    private final int id;
    private final int arrivalTime;
    private final int burstTime;
    private int waitingTime;
    private int turnAroundTime;

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

    public int getId() {
        return id;
    }
```

```java
    public int getArrivalTime() {
        return arrivalTime;
    }

    public int getBurstTime() {
        return burstTime;
    }

    public int getWaitingTime() {
        return waitingTime;
    }


    public void setWaitingTime(int waitingTime) {
        this.waitingTime = waitingTime;
    }

    public void setTurnAroundTime(int turnAroundTime) {
        this.turnAroundTime = turnAroundTime;
    }

    public void printProcessData() {
        System.out.print(id + "\t\t" + arrivalTime + "\t\t\t" + burstTime + "\n");
    }

    public void printWaitData() {
        System.out.print(id + "\t\t" + waitingTime + "\t\t\t" + turnAroundTime + "\n");
    }
}
```



```
C:\Users\thearchhero\.jdks\openjdk-15.0.2\bin\java.exe "-
How many processes do you have?2
Please enter the arrival time for process 1
0
Please enter the burst time for process 1
3
Please enter the arrival time for process 2
2
Please enter the burst time for process 2
2
pId  pArrivalTime  pBurstTime
1       0           3
2       2           2
pId  pWaiting    pTurnaround
1       0           3
2       1           3

Process finished with exit code 0
```