Program to implement CPU scheduling for shortest job first (Preemptive and Non-Preemptive)

* SJF (Non-Preemptive)

CODE:-

**import java.util.Arrays;**

**import java.util.Comparator;**

**import java.util.Scanner;**

**class ProcessSJF {**

**int id;**

**int arrivalTime;**

**int burstTime;**

**int waitingTime;**

**int turnaroundTime;**

**public ProcessSJF(int id, int arrivalTime, int burstTime) {**

**this.id = id;**

**this.arrivalTime = arrivalTime;**

**this.burstTime = burstTime;**

**}**

**}**

**public class SJFNonPreemptive {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter the number of processes: ");**

**int n = scanner.nextInt();**

**ProcessSJF[] processes = new ProcessSJF[n];**

**for (int i = 0; i < n; i++) {**

**System.out.print("Enter arrival time for process " + (i+1) + ": ");**

**int arrivalTime = scanner.nextInt();**

**System.out.print("Enter burst time for process " + (i+1) + ": ");**

**int burstTime = scanner.nextInt();**

**processes[i] = new ProcessSJF(i+1, arrivalTime, burstTime);**

**}**

**// Sort by arrival time, then burst time**

**Arrays.sort(processes, Comparator.comparingInt((ProcessSJF p) -> p.arrivalTime)**

**.thenComparingInt(p -> p.burstTime));**

**int currentTime = 0;**

**for (ProcessSJF process : processes) {**

**if (currentTime < process.arrivalTime) {**

**currentTime = process.arrivalTime;**

**}**

**process.waitingTime = currentTime - process.arrivalTime;**

**process.turnaroundTime = process.waitingTime + process.burstTime;**

**currentTime += process.burstTime;**

**}**

**System.out.println("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time");**

**for (ProcessSJF process : processes) {**

**System.out.println("P" + process.id + "\t\t" + process.arrivalTime + "\t\t" + process.burstTime + "\t\t" +**

**process.waitingTime + "\t\t" + process.turnaroundTime);**

**}**

**double avgWaitingTime = Arrays.stream(processes).mapToInt(p -> p.waitingTime).average().orElse(0);**

**double avgTurnaroundTime = Arrays.stream(processes).mapToInt(p -> p.turnaroundTime).average().orElse(0);**

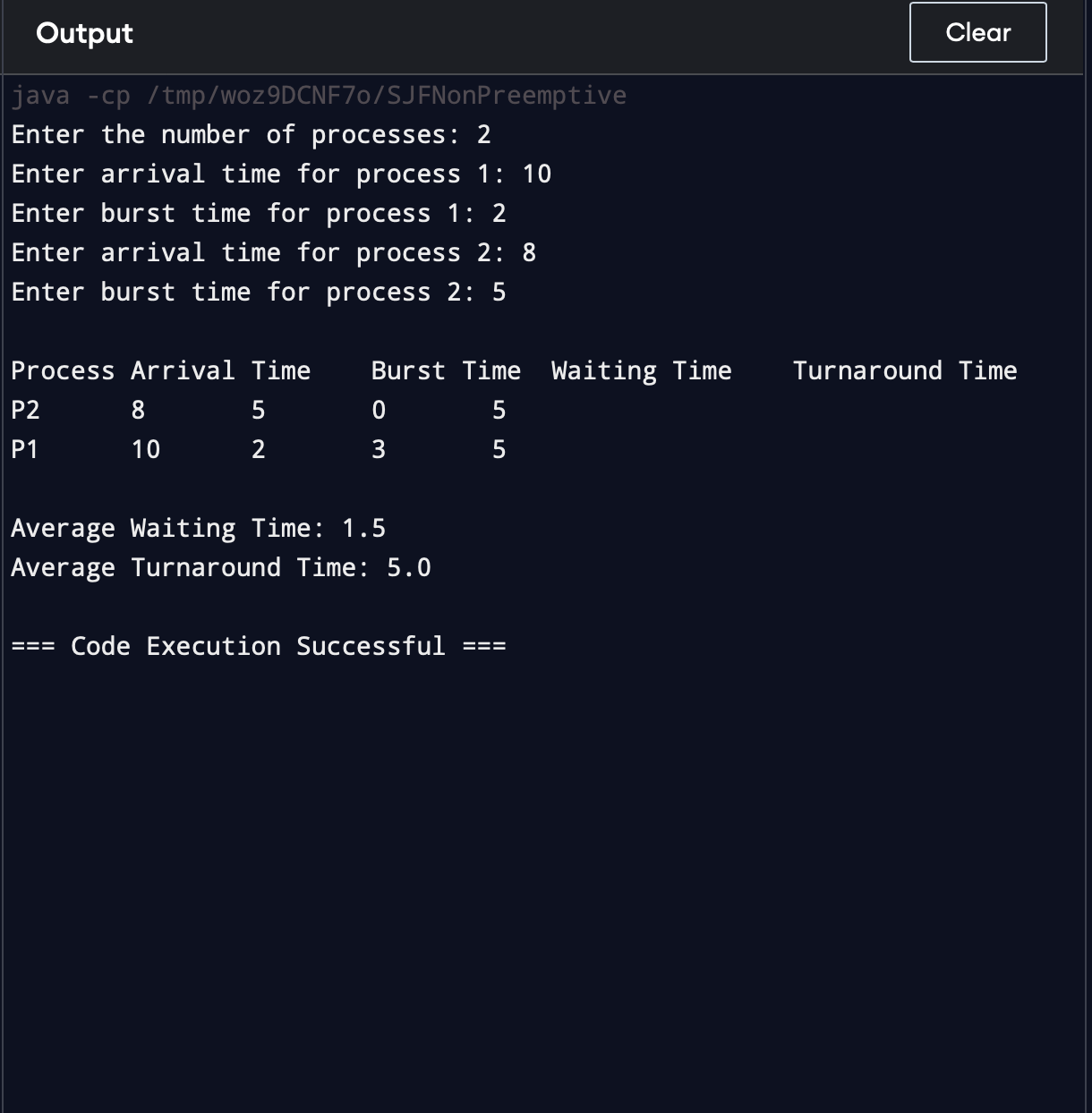
**System.out.println("\nAverage Waiting Time: " + avgWaitingTime);**

**System.out.println("Average Turnaround Time: " + avgTurnaroundTime);**

**}**

**}**

**OUTPUT:-**

****

* 2) SJTF (shortest remaining time first -Preemptive SJF)

CODE:-

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

class ProcessSJTF {

int id;

int arrivalTime;

int burstTime;

int remainingTime;

int waitingTime;

int turnaroundTime;

boolean isCompleted = false;

public ProcessSJTF(int id, int arrivalTime, int burstTime) {

this.id = id;

this.arrivalTime = arrivalTime;

this.burstTime = burstTime;

this.remainingTime = burstTime;

}

}

public class SJTFPreemptive {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of processes: ");

int n = scanner.nextInt();

List<ProcessSJTF> processes = new ArrayList<>();

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

System.out.print("Enter arrival time for process " + (i+1) + ": ");

int arrivalTime = scanner.nextInt();

System.out.print("Enter burst time for process " + (i+1) + ": ");

int burstTime = scanner.nextInt();

processes.add(new ProcessSJTF(i+1, arrivalTime, burstTime));

}

int currentTime = 0;

int completedProcesses = 0;

while (completedProcesses < n) {

// Find the process with the shortest remaining time that has arrived

ProcessSJTF currentProcess = null;

for (ProcessSJTF process : processes) {

if (!process.isCompleted && process.arrivalTime <= currentTime) {

if (currentProcess == null || process.remainingTime < currentProcess.remainingTime) {

currentProcess = process;

}

}

}

if (currentProcess != null) {

// Process the selected process for one unit of time

currentProcess.remainingTime--;

currentTime++;

// If process is completed

if (currentProcess.remainingTime == 0) {

currentProcess.isCompleted = true;

completedProcesses++;

currentProcess.turnaroundTime = currentTime - currentProcess.arrivalTime;

currentProcess.waitingTime = currentProcess.turnaroundTime - currentProcess.burstTime;

}

} else {

// No process is ready to execute, so move forward in time

currentTime++;

}

}

System.out.println("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time");

for (ProcessSJTF process : processes) {

System.out.println("P" + process.id + "\t\t" + process.arrivalTime + "\t\t" + process.burstTime + "\t\t" +

process.waitingTime + "\t\t" + process.turnaroundTime);

}

double avgWaitingTime = processes.stream().mapToInt(p -> p.waitingTime).average().orElse(0);

double avgTurnaroundTime = processes.stream().mapToInt(p -> p.turnaroundTime).average().orElse(0);

System.out.println("\nAverage Waiting Time: " + avgWaitingTime);

System.out.println("Average Turnaround Time: " + avgTurnaroundTime);

}

}

OUTPUT:-

