EXPERIMENT 6 SJF

#SJF(Non Preemptive)

import java.util.Arrays;

import java.util.Comparator;

import java.util.Scanner;

class Process {

int id;

int arrivalTime;

int burstTime;

int waitingTime;

int turnaroundTime;

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

this.id = id;

this.arrivalTime = arrivalTime;

this.burstTime = burstTime;

this.waitingTime = 0;

this.turnaroundTime = 0;

}

}

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();

Process[] processes = new Process[n];

// Input arrival time and burst time for each process

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 Process(i + 1, arrivalTime, burstTime);

}

// Sort processes by arrival time, and if equal, by burst time

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

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

int currentTime = 0;

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

// Find the next process with the shortest burst time that has arrived

int minIndex = -1;

int minBurstTime = Integer.MAX\_VALUE;

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

if (processes[j].arrivalTime <= currentTime && processes[j].burstTime < minBurstTime && processes[j].waitingTime == 0) {

minBurstTime = processes[j].burstTime;

minIndex = j;

}

}

if (minIndex == -1) { // If no process has arrived, move currentTime forward

currentTime++;

i--;

continue;

}

// Calculate waiting time and turnaround time for the selected process

Process process = processes[minIndex];

process.waitingTime = currentTime - process.arrivalTime;

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

// Increment current time by the burst time of the selected process

currentTime += process.burstTime;

// Mark process as completed

processes[minIndex].waitingTime = 1; // marking it completed

}

// Output results

System.out.println("\nProcess ID | Arrival Time | Burst Time | Waiting Time | Turnaround Time");

int totalWaitingTime = 0;

int totalTurnaroundTime = 0;

for (Process process : processes) {

System.out.printf(" %d | %d | %d | %d | %d\n",

process.id, process.arrivalTime, process.burstTime, process.waitingTime, process.turnaroundTime);

totalWaitingTime += process.waitingTime;

totalTurnaroundTime += process.turnaroundTime;

}

double averageWaitingTime = (double) totalWaitingTime / n;

double averageTurnaroundTime = (double) totalTurnaroundTime / n;

System.out.printf("\nAverage Waiting Time: %.2f", averageWaitingTime);

System.out.printf("\nAverage Turnaround Time: %.2f\n", averageTurnaroundTime);

scanner.close();

}

}

