EXPERIMENT 6 SJTF

import java.util.\*;

class Process {

int id;

int arrivalTime;

int burstTime;

int remainingTime;

int waitingTime;

int turnaroundTime;

boolean isCompleted;

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

this.id = id;

this.arrivalTime = arrivalTime;

this.burstTime = burstTime;

this.remainingTime = burstTime;

this.waitingTime = 0;

this.turnaroundTime = 0;

this.isCompleted = false;

}

}

public class SRTFScheduling {

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);

}

int currentTime = 0;

int completed = 0;

int shortestProcessIndex = -1;

int minRemainingTime = Integer.MAX\_VALUE;

while (completed < n) {

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

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

if (processes[i].arrivalTime <= currentTime && !processes[i].isCompleted && processes[i].remainingTime < minRemainingTime) {

shortestProcessIndex = i;

minRemainingTime = processes[i].remainingTime;

}

}

if (shortestProcessIndex == -1) { // If no process has arrived, increment time

currentTime++;

continue;

}

// Process the selected process for 1 time unit

processes[shortestProcessIndex].remainingTime--;

minRemainingTime = processes[shortestProcessIndex].remainingTime;

if (processes[shortestProcessIndex].remainingTime == 0) { // If process is completed

processes[shortestProcessIndex].isCompleted = true;

completed++;

int finishTime = currentTime + 1;

processes[shortestProcessIndex].turnaroundTime = finishTime - processes[shortestProcessIndex].arrivalTime;

processes[shortestProcessIndex].waitingTime = processes[shortestProcessIndex].turnaroundTime - processes[shortestProcessIndex].burstTime;

// Reset minimum remaining time and index for next selection

minRemainingTime = Integer.MAX\_VALUE;

shortestProcessIndex = -1;

}

// Increment time after each unit of processing

currentTime++;

}

// 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();

}

}

