Problem Statement 8 FCFS and ROUND ROBIN

package C;

import java.util.Scanner;

class Process {

int pid;

int waitingTime;

int arrivalTime;

int burstTime;

int turnAroundTime;

int timeToComplete;

int completionTime = 0;

Process(int pid, int arrival, int burst) {

this.pid = pid;

this.arrivalTime = arrival;

this.burstTime = burst;

this.timeToComplete = burstTime;

}

}

public class Scheduler {

static Scanner s = new Scanner(System.in);

public static void main(String[] args) {

System.out.println("ADITYA KOKATE");

System.out.println("22147");

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

int n = s.nextInt();

Process[] myProcess = new Process[n];

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

System.out.print("Enter Arrival time and Burst Time: ");

int arrival = s.nextInt();

int burst = s.nextInt();

myProcess[i] = new Process(i + 1, arrival, burst);

}

System.out.println("Select the type of scheduler to be used:");

System.out.println("1. FCFS");

System.out.println("2. Round Robin");

System.out.print("Enter your choice: ");

int choice = s.nextInt();

switch (choice) {

case 1:

FCFS(myProcess);

break;

case 2:

RoundRobin(myProcess);

break;

default:

System.out.println("Incorrect Choice");

break;

}

s.close();

}

static void FCFS(Process[] myProcess) {

int currentTime = 0;

// Sort processes based on their arrival time

for (int i = 0; i < myProcess.length; i++) {

for (int j = i + 1; j < myProcess.length; j++) {

if (myProcess[i].arrivalTime > myProcess[j].arrivalTime) {

Process temp = myProcess[i];

myProcess[i] = myProcess[j];

myProcess[j] = temp;

}

}

}

// Calculating waiting time and turnaround time

for (Process process : myProcess) {

if (currentTime < process.arrivalTime) {

currentTime = process.arrivalTime;

}

process.completionTime = currentTime + process.burstTime;

process.turnAroundTime = process.completionTime - process.arrivalTime;

process.waitingTime = process.turnAroundTime - process.burstTime;

currentTime += process.burstTime;

System.out.println("Process " + process.pid + ":");

System.out.println("Turnaround Time: " + process.turnAroundTime + ", Completion Time: " + process.completionTime + ", Waiting Time: " + process.waitingTime);

}

}

static void RoundRobin(Process[] myProcess) {

System.out.print("Specify time quantum: ");

int quantum = s.nextInt();

int currentTime = 0;

int completedProcesses = 0;

// Creating a copy of the timeToComplete array

int[] remainingTime = new int[myProcess.length];

for (int i = 0; i < myProcess.length; i++) {

remainingTime[i] = myProcess[i].burstTime;

}

// Traverse processes in Round Robin manner

while (completedProcesses < myProcess.length) {

boolean didProcess = false;

for (int i = 0; i < myProcess.length; i++) {

if (remainingTime[i] > 0) {

didProcess = true;

if (remainingTime[i] > quantum) {

currentTime += quantum;

remainingTime[i] -= quantum;

} else {

currentTime += remainingTime[i];

myProcess[i].completionTime = currentTime;

myProcess[i].turnAroundTime = myProcess[i].completionTime - myProcess[i].arrivalTime;

myProcess[i].waitingTime = myProcess[i].turnAroundTime - myProcess[i].burstTime;

remainingTime[i] = 0;

completedProcesses++;

}

}

}

if (!didProcess) break; // Break if all processes are done

}

for (Process process : myProcess) {

System.out.println("Process " + process.pid + ":");

System.out.println("Turnaround Time: " + process.turnAroundTime + ", Completion Time: " + process.completionTime + ", Waiting Time: " + process.waitingTime);

}

}

}