SCHEDULER

package C1;

import java.io.Closeable;

import java.util.Scanner;

class Process{

int pid;

int waitingTime;

int arrivalTime;

int burstTime;

int turnAroundTime;

int timeToComplete;

int completionTime = 0;

int priority;

Process(int pid, int sub,int bur){

this.pid = pid;

this.arrivalTime = sub;

this.burstTime = bur;

this.timeToComplete = burstTime;

}

Process(int pid, int sub,int bur, int priority){

this.pid = pid;

this.arrivalTime = sub;

this.burstTime = bur;

this.priority = priority;

this.timeToComplete = burstTime;

}

}

public class Scheduler{

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

public static void main(String[] args){

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

int n = s.nextInt();

Process[] myProcess = new Process[n];

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

System.out.println("Enter Arrival time, Burst Time, and Priority: ");

int sub = s.nextInt();

int bur = s.nextInt();

int priority = s.nextInt();

myProcess[i] = new Process(i+1,sub,bur,priority);

}

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

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

System.out.println("2. SJF (Preemptive)");

System.out.println("3. Priority (Non-preemptive");

System.out.println("4. Round Robin");

System.out.println("5. Exit");

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

int choice = s.nextInt();

switch(choice){

case 1:

FCFS(myProcess);

break;

case 2:

SJF(myProcess);

break;

case 3:

PriorityScheduling(myProcess);

break;

case 4:

RoundRobin(myProcess);

break;

case 5:

s.close();

System.exit(1);

break;

default:

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

break;

}

s.close();

}

static void FCFS(Process myProcess[]){

int x=0;

//Arrange processes according to their arrival time in the ascending order

Process temp;

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

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

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

temp = myProcess[j];

myProcess[j] = myProcess[i];

myProcess[i] = temp;

}

}

}

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

x = x+myProcess[i].burstTime;

myProcess[i].completionTime = x;

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

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

System.out.println("Process "+ myProcess[i].pid +":");

System.out.println("turnAroundTime\tCompletion\twaitingTimeing");

System.out.println(myProcess[i].turnAroundTime+"\t\t\t"+myProcess[i].completionTime+"\t\t"+myProcess[i].waitingTime);

}

}

static void SJF(Process myProcess[]){

int curTimeInterval = 0, completedProcesses = 0;

Process curProcess;

//Traverse until all process gets completely executed.

curProcess = myProcess[0];

while(completedProcesses < myProcess.length){

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

if(myProcess[i].timeToComplete > 0){

curProcess = myProcess[i];

break;

}

}

System.out.println("Current Time Interval = " + curTimeInterval);

System.out.println("No of Processes Completed = " + completedProcesses);

//Find process with minimum remaining time at every single time lap

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

if(myProcess[i].arrivalTime > curTimeInterval || myProcess[i].timeToComplete == 0 ){

continue;

}

if(myProcess[i].timeToComplete < curProcess.timeToComplete){

curProcess = myProcess[i];

}

}

//Reduce its time by 1

curProcess.timeToComplete -= 1;

//Check if its remaining time becomes 0

if(curProcess.timeToComplete == 0){

//Increment the counter of process completion.

completedProcesses++;

//Completion time of current process = current\_time +1;

curProcess.completionTime = curTimeInterval +1;

}

curTimeInterval++;

}

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

//Calculate waitingTimeing time for each process

//waitingTimeing Time = Completion time - arrival\_time - burst\_time

myProcess[i].waitingTime = myProcess[i].completionTime - myProcess[i].arrivalTime - myProcess[i].burstTime;

//Find turnAroundTime time (waitingTimeing\_time+burst\_time)

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

System.out.println("Process "+ myProcess[i].pid +":");

System.out.println("turnAroundTime\tCompletion\twaitingTimeing");

System.out.println(myProcess[i].turnAroundTime+"\t\t\t"+myProcess[i].completionTime+"\t\t"+myProcess[i].waitingTime);

}

}

static void PriorityScheduling(Process myProcess[]){

//Arrange processes according to their priority in the descending order

Process temp;

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

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

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

temp = myProcess[j];

myProcess[j] = myProcess[i];

myProcess[i] = temp;

}

}

}

int x = 0;

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

x = x+myProcess[i].burstTime;

myProcess[i].completionTime = x;

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

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

System.out.println("Process "+ myProcess[i].pid +":");

System.out.println("turnAroundTime\tCompletion\twaitingTimeing");

System.out.println(myProcess[i].turnAroundTime+"\t\t\t"+myProcess[i].completionTime+"\t\t"+myProcess[i].waitingTime);

}

}

static void RoundRobin(Process myProcess[]){

int curTimeInterval = 0, completedProcesses = 0;

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

int quantum = s.nextInt();

// int quantum = 4;

//Keep traversing the all processes while all processes

//are not done. Do following for i'th process if it is

//not done yet.

while(completedProcesses < myProcess.length){

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

if(myProcess[i].timeToComplete > 0 && myProcess[i].timeToComplete > quantum){

//Execute the process for the time quantum

curTimeInterval += quantum;

myProcess[i].timeToComplete -= quantum;

}

else{

if(myProcess[i].timeToComplete > 0){

//Execute last cycle for the process

curTimeInterval += myProcess[i].timeToComplete;

myProcess[i].timeToComplete = 0;

myProcess[i].completionTime = curTimeInterval;

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

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

completedProcesses++;

}

}

}

}

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

System.out.println("Process "+ myProcess[i].pid +":");

System.out.println("turnAroundTime\tCompletion\twaitingTimeing");

System.out.println(myProcess[i].turnAroundTime+"\t\t\t"+myProcess[i].completionTime+"\t\t"+myProcess[i].waitingTime);

}

}

}