package os2;

import java.applet.Applet;

import java.awt.Color;

import java.awt.Component;

import java.awt.Frame;

import java.awt.Graphics;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import javax.swing.\*;

//import exp.FrameForPriorityScheduling;

//import exp.Main;

public class Main1

{

static int minimumArrivalTime,sumCPUBurstTime,minimumbursttime;

static int lengthOfEachBlock;

static final int SCREEN\_WIDTH=700,SCREEN\_HEIGHT=280;

static final int rectangleUpperPadding=50,rectangleHeight=100;

static int numberOfProcesses;

static int CPUBurstTime[],arrivalTime[];

static int priority;

static BufferedReader br;

static Main1 obj;

static int prior[];

Main1()

{

this.obj=this;

}

public static void main(String[] args) throws NumberFormatException, IOException

{

br=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the priority : ");

priority=Integer.parseInt(br.readLine());

for(int i=0;i<priority;i++)

{

System.out.println(" 1:FCFS 2:SJF 3:PS");

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

int ch=Integer.parseInt(br.readLine());

switch(ch)

{

case 1: drawGanttChartForFCFS();

break;

case 2:drawGanttChartForNonPreemptiveSJF();

break;

case 3:drawGanttChartForPriorityScheduling();

break;

default:System.out.println("Invalid Input");

}

drawGanttChart();

}

}

private static void PS() throws NumberFormatException, IOException {

br=new BufferedReader(new InputStreamReader(System.in));

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

numberOfProcesses=Integer.parseInt(br.readLine());

CPUBurstTime=new int[numberOfProcesses];

Main1.prior=new int[numberOfProcesses];

for(int i=0;i<numberOfProcesses;i++)

{

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("Enter the data for the process "+(i+1));

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.print("Enter the CPU Burst Time : ");

CPUBurstTime[i]=Integer.parseInt(br.readLine());

System.out.print("Enter the Priority : ");

prior[i]=Integer.parseInt(br.readLine());

}

// TODO Auto-generated method stub

}

private static void SJF() throws NumberFormatException, IOException {

br=new BufferedReader(new InputStreamReader(System.in));

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

numberOfProcesses=Integer.parseInt(br.readLine());

CPUBurstTime=new int[numberOfProcesses];

arrivalTime=new int[numberOfProcesses];

for(int i=0;i<numberOfProcesses;i++)

{

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("Enter the data for the process "+(i+1));

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.print("Enter the arrival Time : ");

arrivalTime[i]=Integer.parseInt(br.readLine());

System.out.print("Enter the CPU Burst Time : ");

CPUBurstTime[i]=Integer.parseInt(br.readLine());

}

minimumbursttime=Integer.MAX\_VALUE;

for(int i=0;i<numberOfProcesses;i++)

{

if(minimumbursttime>CPUBurstTime[i])

minimumbursttime=CPUBurstTime[i];

}

}

private static void FCFS() throws NumberFormatException, IOException {

br=new BufferedReader(new InputStreamReader(System.in));

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

numberOfProcesses=Integer.parseInt(br.readLine());

CPUBurstTime=new int[numberOfProcesses];

arrivalTime=new int[numberOfProcesses];

for(int i=0;i<numberOfProcesses;i++)

{

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("Enter the data for the process "+(i+1));

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.print("Enter the arrival Time : ");

arrivalTime[i]=Integer.parseInt(br.readLine());

System.out.print("Enter the CPU Burst Time : ");

CPUBurstTime[i]=Integer.parseInt(br.readLine());

}

minimumArrivalTime=Integer.MAX\_VALUE;

for(int i=0;i<numberOfProcesses;i++)

{

if(minimumArrivalTime>arrivalTime[i])

minimumArrivalTime=arrivalTime[i];

}

}

public static void drawGanttChart() throws NumberFormatException, IOException

{

sumCPUBurstTime=0;

/\* calculating the sum of all cpu burst time \*/

for(int i=0;i<numberOfProcesses;i++)

{

sumCPUBurstTime+=CPUBurstTime[i];

}

/\* now the total width of the screen is to be divided into sumCPUBurstTime equal parts \*/

lengthOfEachBlock=SCREEN\_WIDTH/sumCPUBurstTime;

// drawGanttChartForPriorityScheduling();

// drawGanttChartForNonPreemptiveSJF();

// drawGanttChartForFCFS();

}

public static void drawGanttChartForFCFS() throws NumberFormatException, IOException

{ FCFS();

new FrameForFCFS(obj);

}

public static void drawGanttChartForNonPreemptiveSJF() throws NumberFormatException, IOException

{ SJF();

new FrameNonPreemptiveSJF(obj);

}

public static void drawGanttChartForPriorityScheduling() throws NumberFormatException, IOException

{ PS();

new FrameForPriorityScheduling(obj);

}

}

class FrameForFCFS extends Frame

{

private static final long serialVersionUID = 1L;

int arrivalTime[];

Main1 obj;

FrameForFCFS(Main1 obj)

{

super("FCFS");

this.obj=obj;

this.setResizable(false);

this.setVisible(true);

this.setSize(obj.SCREEN\_WIDTH+100, obj.SCREEN\_HEIGHT);

arrivalTime=(int[]) obj.arrivalTime.clone();

// this.setVisible(false);

}

public void paint(Graphics g)

{

super.paint(g);

//this.getContentPane().setBackground(Color.white);

int currentTime=obj.minimumArrivalTime;

arrivalTime=(int[]) obj.arrivalTime.clone();

int i,j,min,mini = 0;

int leftStart=50;

// g=this.getContentPane().getGraphics();

g.drawString(""+obj.minimumArrivalTime,leftStart,obj.rectangleUpperPadding+obj.rectangleHeight+20);

for(j=0;j<obj.numberOfProcesses;j++)

{

min=Integer.MAX\_VALUE;

for(i=0;i<obj.numberOfProcesses;i++)

{

if(min>arrivalTime[i])

{

min=arrivalTime[i];

mini=i;

}

}

arrivalTime[mini]=Integer.MAX\_VALUE;

//g=this.getContentPane().getGraphics();

g.drawRect(leftStart,obj.rectangleUpperPadding,obj.lengthOfEachBlock\*obj.CPUBurstTime[mini],obj.rectangleHeight);

g.drawString("P"+(mini+1),leftStart+5,obj.rectangleUpperPadding+50);

leftStart+=obj.lengthOfEachBlock\*obj.CPUBurstTime[mini];

currentTime+=Main1.CPUBurstTime[mini];

g.drawString(""+currentTime,leftStart,obj.rectangleUpperPadding+obj.rectangleHeight+20);

}

}

}

class FrameNonPreemptiveSJF extends Frame{

private static final long serialVersionUID = 1L;

int CPUBurstTime[];

Main1 obj;

FrameNonPreemptiveSJF(Main1 obj)

{

super("Non Preemptive SJF");

this.obj=obj;

this.setResizable(false);

this.setVisible(true);

this.setSize(obj.SCREEN\_WIDTH+100, obj.SCREEN\_HEIGHT);

CPUBurstTime=(int[]) obj.CPUBurstTime.clone();

}

public void paint(Graphics g)

{

super.paint(g);

// this.getContentPane().setBackground(Color.white);

int currentTime=obj.minimumbursttime;

CPUBurstTime=(int[]) obj.CPUBurstTime.clone();

int i,j,min,mini = 0;

int leftStart=50;

// g=this.getContentPane().getGraphics();

g.drawString(""+obj.minimumArrivalTime,leftStart,obj.rectangleUpperPadding+obj.rectangleHeight+20);

for(j=0;j<obj.numberOfProcesses;j++)

{

min=Integer.MAX\_VALUE;

for(i=0;i<obj.numberOfProcesses;i++)

{

if(min>CPUBurstTime[i] && obj.arrivalTime[i]<=currentTime)

{

min=CPUBurstTime[i];

mini=i;

}

}

// g=this.getContentPane().getGraphics();

g.drawRect(leftStart,obj.rectangleUpperPadding,obj.lengthOfEachBlock\*obj.CPUBurstTime[mini],obj.rectangleHeight);

g.drawString("P"+(mini+1),leftStart+5,obj.rectangleUpperPadding+50);

leftStart+=obj.lengthOfEachBlock\*obj.CPUBurstTime[mini];

currentTime+=obj.CPUBurstTime[mini];

g.drawString(""+currentTime,leftStart,obj.rectangleUpperPadding+obj.rectangleHeight+20);

CPUBurstTime[mini]=Integer.MAX\_VALUE;

}

}

}

class FrameForPriorityScheduling extends Frame

{

private static final long serialVersionUID = 1L;

Main1 obj;

int prior[];

FrameForPriorityScheduling(Main1 obj)

{

super("Priority Scheduling");

this.obj=obj;

this.setVisible(true);

this.setSize(obj.SCREEN\_WIDTH+100, obj.SCREEN\_HEIGHT);

repaint();

}

public void paint(Graphics g)

{

super.paint(g);

prior=(int[]) obj.prior.clone();

// this.getContentPane().setBackground(Color.white);

int currentTime=0;

int leftStart=50;

int min,mini = 0;

g.drawString(""+currentTime,leftStart,obj.rectangleUpperPadding+obj.rectangleHeight+20);

for(int j=0;j<obj.numberOfProcesses;j++)

{

min=Integer.MAX\_VALUE;

for(int i=0;i<obj.numberOfProcesses;i++)

{

if(min>prior[i])

{

min=prior[i];

mini=i;

}

}

prior[mini]=Integer.MAX\_VALUE;

// g=this.getContentPane().getGraphics();

g.drawRect(leftStart,obj.rectangleUpperPadding,obj.lengthOfEachBlock\*obj.CPUBurstTime[mini],obj.rectangleHeight);

g.drawString("P"+(mini+1),leftStart+5,obj.rectangleUpperPadding+50);

leftStart+=obj.lengthOfEachBlock\*obj.CPUBurstTime[mini];

currentTime+=obj.CPUBurstTime[mini];

g.drawString(""+currentTime,leftStart,obj.rectangleUpperPadding+obj.rectangleHeight+20);

}

}

}