**一、需求分析**

明确陈述说明程序设计的任务，强调的是程序要做什么，主要包括：

**(1)输入的形式和输入值的范围；**

输入形式是键盘输入和鼠标输入，键盘输入值的范围是[0,10]的整数，鼠标输入的选择有确认、四种调度算法（时间片轮转调度、优先数调度、最短进程优先调度和最短剩余时间调度优先调度）、开始模拟、清空和退出。

**(2)输出的形式；**

输出的形式为java图形界面显示。根据输入模拟进程调度，输出由动态面板显示出来。

**(3)程序所能达到的功能；**

1、创建进程，模拟四种调度算法，通过图形界面动态显示过程与结果。

2、清空前一次的模拟结果。

**(4)测试数据：包括正确的输入及其输出结果和含有错误的输入及其输出结果。**

正确的输入：1、进程数n：0<=n<=10，确认。2、选择四种调度算法。3、开始模拟。

正确的输出结果：能按所选的调度算法正确地执行，并且将结果显示在图形界面窗口。

错误的输入：1、进程数n：n<0||n>10。2、缺少点击确认、四种调度算法、开始模拟等按钮。

错误的输出结果：1、进程数n非法输入时会弹出窗口提示。2、程序无法执行。

**二、概要设计**

说明本程序中用到的所有抽象数据类型的定义、主程序的流程以及各程序模块之间的层次(调用)关系。

**所有抽象数据类型的定义：**

Class Main //程序入口，给界面的位置、大小和名称赋值。

Class UI extends JFrame //构建图形界面。

Class Label implements Runnable //刷新界面的数据，形成动态界面。

Class Pcb //定义进程块。

Class comparePri implements Comparator<Pcb> //比较优先级的大小，进行排序。

Class compareTime implements Comparator<Pcb> //比较剩余时间的大小，进行排序。

Class SchedulingMethod //创建进程以及四种调度算法的实现。

**主程序的流程：**

构建图形界面->各线程启动，形成动态界面->键盘输入进程数->鼠标选择调度方法->开始模拟，数据刷新->清空->重复/退出

**各程序模块之间的层次(调用)关系：**

Label

Pcb

SchedulingMethod.XX(调度方法)

UI

Main

SchedulingMethod.createProcess

**三、详细设计**

实现概要设计中定义的所有数据类型，对每个操作只需要写出伪码算法；对主程序和其他模块也都需要写出伪码算法(伪码算法达到的详细程度应能够按照伪码算法在计算机键盘上直接输入高级程序设计语言程序)；画出函数的调用关系图。

public class Main {

public static void main(String[] args) {

UI window = new UI(str,x,y,w,h);

}

}

class Label implements Runnable{

public Label(JLabel[] id,JLabel[] name,JLabel[] status,JLabel[] pri, JLabel[] time,JLabel[] next) {

this.label1 = id;

this.label2 = name;

this.label3 = status;

this.label4 = pri;

this.label5 = time;

this.label6 = next;

}

@Override

public void run() {

while(true) {

i = pp.getId();

label2[i].setText(String.valueOf(pp.getName()));

label4[i].setText(String.valueOf(pp.getPri()));

label5[i].setText(String.valueOf(pp.getTime()));

label6[i].setText(String.valueOf(pp.getNext()));

if(pp.getStatus() == 0) {

label3[i].setText("就绪");

}else if(pp.getStatus() == 1) {

label3[i].setText("运行");

}else {

label3[i].setText("终止");

}

}

}

}

public class UI extends JFrame {

public UI(String s,int x,int y,int w,int h) {

btn01.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

new Thread(new Runnable() {

@Override

public void run() {

n = SchedulingMethod.getN();

switch(key) {

case 1:

SchedulingMethod.RR(SchedulingMethod.createProcess(n), n);

break;

case 2:

SchedulingMethod.PSA(SchedulingMethod.createProcess(n), n);

break;

case 3:

SchedulingMethod.SPN(SchedulingMethod.createProcess(n), n);

break;

case 4:

SchedulingMethod.SPT(SchedulingMethod.createProcess(n), n);

break;

}

}

}).start();

}

});

btn02.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

new Thread(new Runnable() {

@Override

public void run() {

for(int i = 0; i < 10; i++) {

Pcb p= new Pcb();

p.setId(i);

Label.setPP(p);

try {

Thread.sleep(1000);

}catch(InterruptedException e) {

e.printStackTrace();

}

}

}

}).start();

}

});

btn03.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

new Thread(new Runnable() {

@Override

public void run() {

System.exit(0);

}

}).start();

}

});

btn04.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

new Thread(new Runnable() {

@Override

public void run() {

error = false;

if((Pattern.compile("[0-9]\*")).matcher(text.getText()).matches())

if(Integer.parseInt(text.getText()) >= 0 && Integer.parseInt(text.getText()) <= 10)

SchedulingMethod.setN(Integer.parseInt(text.getText()));

else

error = true;

else

error = true;

if(error) {

JFrame newJFrame = new JFrame("非法输入");

JLabel label = new JLabel("n为0-10的正整数");

panel.add(label);

}

}

}).start();

}

});

btn05.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

new Thread(new Runnable() {

@Override

public void run() {

key = 1;

label1.setText("当前选择的是时间片轮转调度");

}

}).start();

}

});

btn06.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

new Thread(new Runnable() {

@Override

public void run() {

key = 2;

label1.setText("当前选择的是优先数调度");

}

}).start();

}

});

btn07.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

new Thread(new Runnable() {

@Override

public void run() {

key = 3;

label1.setText("当前选择的是最短进程优先调度");

}

}).start();

}

});

btn08.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

new Thread(new Runnable() {

@Override

public void run() {

key = 4;

label1.setText("当前选择的是最短剩余时间优先调度");

}

}).start();

}

});

for(int i = 0; i < 10; i++) {

idLabel[i] = new JLabel("进程"+i);

nameLabel[i] = new JLabel("0");

statusLabel[i] = new JLabel("就绪");

priLabel[i] = new JLabel("0");

timeLabel[i] = new JLabel("0");

nextLabel[i] = new JLabel("0");

}

new Thread(new Label(idLabel,nameLabel,statusLabel,priLabel,timeLabel,nextLabel)).start();

pane.setLayout(new BorderLayout());

pane.add(panel2,NORTH);

pane.add(panel,WEST);

pane.add(panel4,CENTER);

pane.add(panel5,EAST);

pane.add(panel6,SOUTH);

setVisible(true);

}

}

public class Pcb {

public Pcb() {

this.id = 0;

this.name = 0;

this.status = 0;

this.pri = 0;

this.time = 0;

this.next = 0;

}

public Pcb(int id,int name,int status,int pri,int time,int next) {

this.id = id;

this.name = name;

this.status = status;

this.pri = pri;

this.time = time;

this.next = next;

}

}

class comparePri implements Comparator<Pcb>{

public int compare(Pcb o1,Pcb o2) {

if(o1.getPri() > o2.getPri()) {

return -1;

}

else if(o1.getPri() < o2.getPri()) {

return 1;

}

else {

return 0;

}

}

}

class compareTime implements Comparator<Pcb>{

public int compare(Pcb o1,Pcb o2) {

if(o1.getTime() > o2.getTime()) {

return 1;

}

else if(o1.getTime() < o2.getTime()) {

return -1;

}

else {

return 0;

}

}

}

public class SchedulingMethod{

public SchedulingMethod() {

SchedulingMethod.n = 0;

}

public static Pcb[] createProcess(int n) {

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

pcb[i] = new Pcb();

pcb[i].setId(i);

pcb[i].setName((int)(Math.random()\*10)+i\*10);

pcb[i].setStatus(0);

pcb[i].setPri((int)(Math.random()\*10)+1);

pcb[i].setTime((int)(Math.random()\*10)+1);

if(n!=1)

pcb[i].setNext(i+1);

else

pcb[i].setNext(i);

if(i == n-1)

pcb[i].setNext(0);

}

return pcb;

}

public static void RR(Pcb[] pcb,int n) {

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

pcbList.add(pcb[i]);

}

while(!isFinish(pcbList)) {

p = pcbList.get(0);

p.setStatus(1);

if(pcbList.size()!=1)

p.setNext(pcbList.get(1).getId());

else

p.setNext(pcbList.get(0).getId());

Label.setPP(p);

try {

Thread.sleep(1000);

}catch(InterruptedException e) {

e.printStackTrace();

}

p.setTime(p.getTime()-1);

if (p.getTime() == 0) {

p.setStatus(-1);

p.setNext(0);

pcbList.remove(0);

}

else {

if(pcbList.size()!=1)

p.setStatus(0);

pcbList.add(pcbList.get(0));

pcbList.remove(0);

}

Label.setPP(p);

try {

Thread.sleep(1000);

}catch(InterruptedException e) {

e.printStackTrace();

}

}

}

public static void PSA(Pcb[] pcb,int n) {

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

pcbList.add(pcb[i]);

}

while(!isFinish(pcbList)) {

Collections.sort(pcbList,new comparePri());

p = pcbList.get(0);

p.setStatus(1);

if(pcbList.size()!=1)

p.setNext(pcbList.get(1).getId());

else

p.setNext(pcbList.get(0).getId());

Label.setPP(p);

try {

Thread.sleep(1000);

}catch(InterruptedException e) {

e.printStackTrace();

}

p.setPri(p.getPri()-1);

p.setTime(p.getTime()-1);

if (p.getTime() == 0) {

p.setStatus(-1);

p.setNext(0);

pcbList.remove(0);

}else {

if(pcbList.size()!=1)

p.setStatus(0);

}

Label.setPP(p);

try {

Thread.sleep(1000);

}catch(InterruptedException e) {

e.printStackTrace();

}

}

}

public static void SPN(Pcb[] pcb,int n) {

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

pcbList.add(pcb[i]);

}

Collections.sort(pcbList,new compareTime());

while(!isFinish(pcbList)) {

p = pcbList.get(0);

p.setStatus(1);

if(pcbList.size()!=1)

p.setNext(pcbList.get(1).getId());

else

p.setNext(pcbList.get(0).getId());

Label.setPP(p);

try {

Thread.sleep(1000);

}catch(InterruptedException e) {

e.printStackTrace();

}

p.setTime(p.getTime()-1);

if (p.getTime() == 0) {

p.setStatus(-1);

p.setNext(0);

pcbList.remove(0);

}else {

if(pcbList.size()!=1)

p.setStatus(0);

}

Label.setPP(p);

try {

Thread.sleep(1000);

}catch(InterruptedException e) {

e.printStackTrace();

}

}

}

public static void SPT(Pcb[] pcb,int n) {

ArrayList<Pcb> pcbList = new ArrayList<Pcb>();

Pcb p;

int rand;

int i=1;

int count = 0;

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

rand = (int)(Math.random()\*100);

if(rand == 77)

pcbList.add(pcb[i]);

}\*/

pcbList.add(pcb[0]);

while(!isFinish(pcbList)) {

while(i<n && count != 2) {

rand = (int)(Math.random()\*100);

if(rand == 77) {

pcbList.add(pcb[i]);

i++;

count++;

}

}

count = 0;

p = pcbList.get(0);

p.setStatus(1);

if(pcbList.size()!=1)

p.setNext(pcbList.get(1).getId());

else

p.setNext(pcbList.get(0).getId());

Label.setPP(p);

try {

Thread.sleep(1000);

}catch(InterruptedException e) {

e.printStackTrace();

}

p.setTime(p.getTime()-1);

if (p.getTime() == 0) {

p.setStatus(-1);

p.setNext(0);

pcbList.remove(0);

}else {

if(pcbList.size()!=1)

p.setStatus(0);

}

Label.setPP(p);

try {

Thread.sleep(1000);

}catch(InterruptedException e) {

e.printStackTrace();

}

Collections.sort(pcbList,new compareTime());

}

}

public static boolean isFinish(ArrayList<Pcb> pcbList) {

flag = true;

for(Pcb p:pcbList) {

if(p.getTime() != 0) {

flag = false;

break;

}

}

return flag;

}

}

main()

UI()

SchedulingMethod.*setN*

SchedulingMethod.*SPN*

SchedulingMethod.g*etN*

SchedulingMethod.*PSA*

SchedulingMethod.RR

SchedulingMethod.*SPT*

Pcb()

Label()

comparePri

compareTime

isFinish()

SchedulingMethod.createProcess

Pcb()

Label.*setPP*

Pcb的set、get方法

**四、调试分析**

内容包括：

**(1) 调试过程中遇到的问题是如何解决的以及对设计与实现的讨论和分析；**

一开始没有使用线程做动态面板，导致无法实时更新进程的信息属性，监听器和面板刷新加入线程后就解决了问题。

图像界面显示与设计的界面基本一致，实验要求的功能也都实现了。不过在某些功能上还有点瑕疵，比如说只设计了显示当前进程的信息属性，不能一次性观察全部进程的信息属性；“清空”不能一次性清空。

**(2) 算法的时间复杂性(包括基本操作和其他算法的时间复杂性的分析)和改进设想；**

算法的时间复杂性不高，只有一个循环，为O(n)。四种调度算法的代码重复率较高，每个改动的地方不多，可以找出相同的地方，整合到一个函数，这样代码就更精简了。

**(3) 设计过程的经验和体会；**

这次实验我是先想好图形界面并将其实现才开始写调度算法，然后把操作流程每一步想好，一步一步进行，这样一来图形界面的按钮、文本框和标签需要的功能就能有序的写出来了，不像先在控制台写好调度算法，实现之后连接图形界面的时候需要改大量代码，并且容易出错。虽然过程中遇到不少问题，但是通过自己的思考、在网络翻阅大量帖子寻找答案及其同学的帮助，最终还是把问题解决了。本次实验让我又发现了不少知识盲区，通过做实验的方式加深了对这些问题的印象，还是挺有意思的。

**(4) 实现过程中出现的主要问题及解决方法。**

一开始做的图形界面并不满意，接触到绝对布局后就得到了改善。在调试调度方法时，图形界面的数据无法实时刷新，翻阅资料后了解到需要使用多线程才能实现动态界面，于是加了线程给监听器和标签，让数据实时更新。解决这一问题后又遇到了，延迟时间的改变会导致动态界面的数据出现错误显示，后来经过多次的调试，找到了正确的延迟时间。

**五、用户使用说明**

**说明如何使用你编写的程序，详细列出每一步的操作步骤。**

1.在文本框里输入进程数n（0<=n<=10）,并点击确认。

2.选择调度算法（四选一，有提示当前所选择的调度算法）。

3.点击“开始模拟”。开始模拟进程调度，此时可以观察进程的信息属性的变化情况。

4.点击“清空”。数据全部清空初始化。

5.此时可以选择重新模拟，即重复上述操作；也可以选择退出程序，即点击“退出”，窗口关闭，程序结束运行。

**六、测试与运行结果**

列出你的测试结果和运行情况（即运行时的关键画面），包括输入和输出。这里的测试数

据应该完整和严格，最好多于需求分析中所列。

值得注意的是，报告的各种文档资料，要在程序开发的过程中逐渐充实形成，而不是最后补

写。必要时可在报告中附部分关键源代码，但不需要附全部源代码。

**初始界面：**



**非法输入如下图：**







**代码如下：**

boolean error = false;

if((Pattern.compile("[0-9]\*")).matcher(text.getText()).matches()) //检查输入的进程数是否为整型

if(Integer.parseInt(text.getText()) >= 0 && Integer.parseInt(text.getText()) <= 10)

SchedulingMethod.setN(Integer.parseInt(text.getText()));

else

error = true;

else

error = true;

if(error)

JFrame newJFrame = new JFrame("非法输入");

**正确输入时不弹出窗口提示。**

**时间片轮转调度如下图所示：**



**开始模拟：**



**进程全部运行结束：**



**清空：**



**优先数调度如下图所示：**

**开始模拟：**





**进程全部运行完：**



**清空：**



**最短进程优先调度如下图所示：**

**开始模拟：**



**进程全部运行完：**



**清空：**



**最短剩余时间优先调度如下图所示：**

**开始模拟：**



**进程全部运行完：**



**清空：**



**点击退出，程序便结束运行。**

**代码如下：**

btn03 = new JButton("退出");

btn03.setLocation(50,350);

btn03.setSize(200, 100);

btn03.setFont(font);

btn03.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// TODO 自动生成的方法存根

new Thread(new Runnable() {

@Override

public void run() {

// TODO 自动生成的方法存根

System.exit(0);

}

}).start();

}

});

panel.add(btn03);