

**操作系统原理课程设计报告**

|  |  |
| --- | --- |
| **系（院）：** | 计算机科学学院 |
| **专业班级：** | 计科11701 |
| **姓 名：** | 杨华钟 |
| **学 号：** | 201703208 |
| **指导教师：** | 罗爱军 |
| **设计时间：** | 2020.5.25——2020.5.30 |
| **设计地点：** | 家里 |

[一、 课程设计目的 4](#_Toc41737074)

[二、课程设计的任务和要求 4](#_Toc41737075)

[三、模拟程序的描述： 5](#_Toc41737076)

[四、运行环境 7](#_Toc41737077)

[五、算法原理 8](#_Toc41737078)

[1）多级反馈队列调度算法 13](#_Toc41737079)

[2）优先权调度算法 14](#_Toc41737080)

[六、需求分析 16](#_Toc41737081)

[七、总体设计 17](#_Toc41737082)

[八、详细设计与实现[含代码和实现界面] 19](#_Toc41737083)

[九、主要代码分析： 26](#_Toc41737084)

[十、总结 44](#_Toc41737085)

# 课程设计目的

《操作系统原理》是计算机科学与技术专业的一门专业核心课程，也是研究生入学考试中计算机专业综合中所涉及的内容。该课程理论性强，纯粹的理论学习相对枯燥乏味，不易理解。通过课程设计，可加强学生对原理知识的理解。

# 二、课程设计的任务和要求

本次课程设计的题目是，时间片轮转调度算法的模拟实现。要求在充分理解时间片轮转调度算法原理的基础上，编写一个可视化的算法模拟程序。

具体任务如下：

1、根据需要，合理设计PCB结构，以适用于时间片轮转调度算法；

2、设计模拟指令格式，并以文件形式存储，程序能够读取文件并自动生成指令序列。

3、根据文件内容，建立模拟进程队列，并能采用时间片轮转调度算法对模拟进程进行调度。

# 三、模拟程序的描述：

模拟指令的格式：操作命令+操作时间

● C ： 表示在CPU上计算

● I ： 表示输入

● O ： 表示输出

● W ： 表示等待

● H ： 表示进程结束

操作时间代表该操作命令要执行多长时间。这里假设I/O设备的数量没有限制，I和O设备都只有一类。

I，O，W三条指令实际上是不占有CPU的，执行这三条指令就应该将进程放入对应的等待队列（输入等待队列，输出等待队列 ，其他等待队列）。

例如，有一虚拟程序文件prc.txt描述如下：

P1

C10

I20

C40

I30

C20

O30

H00

P2

I10

C50

O20

H00

P3

C10

I20

W20

C40

O10

H00................

# 四、运行环境

编程语言：java swing，编程软件idea，操作系统：Windows10。

# 五、算法原理

在早期的时间片轮转法中,系统将所有的就绪进程按先来先服务的原则,排成一个队列,每次调度时,把CPU分配给队首进程,并令其执行一个时间片.时间片的大小从几ms到几百ms.当执行的时间片用完时,由一个计时器发出时钟中断请求,调度程序便据此信号来停止该进程的执行,并将它送往就绪队列的末尾;然后,再把处理机分配给就绪队列中新的队首进程,同时也让它执行一个时间片.这样就可以保证就绪队列中的所有进程,在一给定的时间内,均能获得一时间片的处理机执行时间.这个算法时间片大小的确定对系统响应时间的要求，就绪队列中进程数目，系统的处理能力。时间片轮转（RR）调度算法是专门为分时系统设计的。它类似于 FCFS调度，但是增加了抢占以切换进程。

该算法中，将一个较小时间单元定义为时间量或时间片。时间片的大小通常为 10~100ms。就绪队列作为循环队列。CPU 调度程序循环整个就绪队列，为每个进程分配不超过一个时间片的 CPU。

为了实现 RR 调度，我们再次将就绪队列视为进程的 FIFO 队列。新进程添加到就绪队列的尾部。CPU 调度程序从就绪队列中选择第一个进程，将定时器设置在一个时间片后中断，最后分派这个进程。

接下来，有两种情况可能发生。进程可能只需少于时间片的 CPU 执行。对于这种情况，进程本身会自动释放 CPU。调度程序接着处理就绪队列的下一个进程。否则，如果当前运行进程的 CPU 执行大于一个时间片，那么定时器会中断，进而中断操作系统。然后，进行上下文切换，再将进程加到就绪队列的尾部，接着 CPU 调度程序会选择就绪队列内的下一个进程。

不过，采用 RR 策略的平均等待时间通常较长。假设有如下一组进程，它们在时间 0 到达，其 CPU 执行以 ms 计：



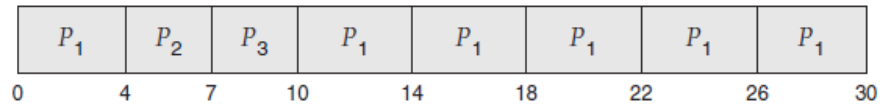
进程 执行时间

P1 24

P2 3

P3 3

如果使用 4ms 的时间片，那么 P1 会执行最初的 4ms。由于它还需要 20ms，所以在第一个时间片之后它会被抢占，而 CPU 就交给队列中的下一个进程。由于 P2 不需要 4ms，所以在其时间片用完之前就会退出。CPU 接着交给下一个进程，即进程 P3。在每个进程都得到了一个时间片之后，CPU 又交给了进程 P1 以便继续执行。因此，RR 调度结果如下：



时间片轮转调度结果

现在，我们计算这个调度的平均等待时间。P1 等待 10-4 = 6ms，P2 等待 4ms，而 P3 等待 7ms。因此，平均等待时间为 17/3 = 5.66ms。

在 RR 调度算法中，没有进程被连续分配超过一个时间片的 CPU（除非它是唯一可运行的进程）。如果进程的 CPU 执行超过一个时间片，那么该进程会被抢占，并被放回到就绪队列。因此，RR调度算法是抢占的。

如果就绪队列有 n 个进程，并且时间片为 q，那么每个进程会得到 1/n 的 CPU 时间，而且每次分得的时间不超过 q 个时间单元。每个进程等待获得下一个 CPU 时间片的时间不会超过 (n-1)q 个时间单元。例如，如果有 5 个进程，并且时间片为 20ms，那么每个进程每 100ms 会得到不超过 20ms 的时间。

RR 算法的性能很大程度取决于时间片的大小。在一种极端情况下，如果时间片很大，那么 RR 算法与 FCFS 算法一样。相反，如果时间片很小（如 1ms），那么 RR 算法可以导致大量的上下文切换。

例如，假设我们只有一个需要 10 个时间单元的进程。如果时间片为 12 个时间单元，那么进程在一个时间片不到就能完成，而且没有额外开销。如果时间片为 6 个时间单元，那么进程需要 2 个时间片，并且还有一个上下文切换。如果时间片为 1 个时间单元，那么就会有 9 个上下文切换，相应地使进程执行更慢（图 1)。

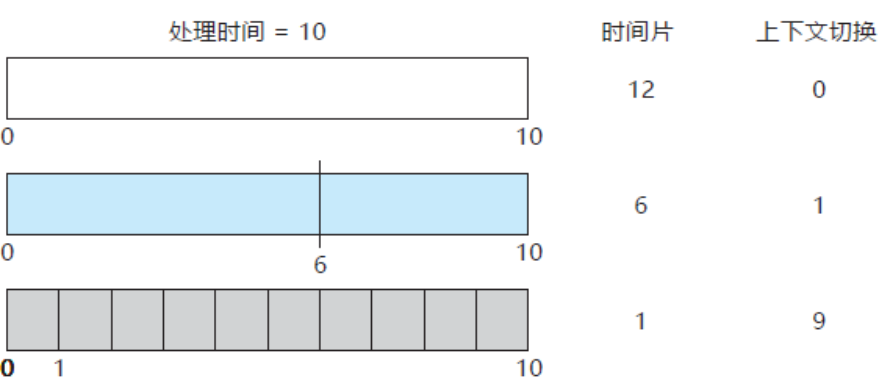


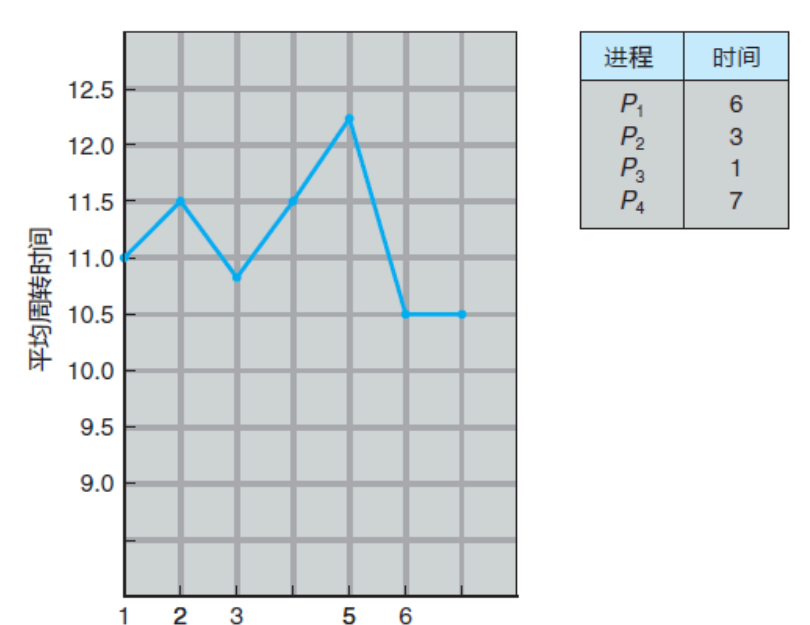
图1

更小时间片如何增加上下文切换

图 1 更小时间片如何增加上下文切换

因此，我们希望时间片远大于上下文切换时间。如果上下文切换时间约为时间片的 10%，那么约 10% 的 CPU 时间会浪费在上下文切换上。在实践中，大多数现代操作系统的时间片为 10~100ms，上下文切换的时间一般少于 10ms；因此，上下文切换的时间仅占时间片的一小部分。

周转时间也依赖于时间片大小。正如从图 2 中所看到的，随着时间片大小的增加，一组进程的平均周转时间不一定会改善。一般情况下，如果大多数进程能在一个时间片内完成，那么平均周转时间会改善。



周转时间如何随着时间片大小而改变

图 2 周转时间如何随着时间片大小而改变

例如，假设有三个进程，都需要 10 个时间单元。如果时间片为 1 个时间单元，那么平均周转时间为 29；如果时间片为 10，那么平均周转时间会降为 20；如果再考虑上下文切换时间，那么平均周转时间对于较小时间片会增加，这是因为需要更多的上下文切换。

尽管时间片应该比上下文切换时间要大，但也不能太大。如果时间片太大，那么 RR 调度就演变成了 FCFS 调度。根据经验，80% 的 CPU 执行应该小于时间片。

## 1）多级反馈队列调度算法

(1) 设置多个就绪队列,并为各个队列赋予不同的优先级. 第一个队列的优先级最高,第二个队列次之,其余各队列的优先权逐个降低.

该算法赋予各个队列中进程执行时间片的大小也各不相同:

在优先权愈高的队列中,为每个进程所规定的执行时间片就愈小.

例如:第二个队列的时间片要比第一个队列的时间片长一倍,……,第i+1个队列的时间片要比第i个队列的时间片长一倍.

(2) 当一个新进程进入内存后,首先将它放入第一队列的末尾,按FCFS原则排队等待调度.当轮到该进程执行时,如它能在该时间片内完成,便可准备撤离系统;如果它在一个时间片结束时尚未完成,调度程序便将该进程转入第二队列的末尾,再同样地按FCFS原则等待调度执行;如果它在第二队列中运行一个时间片后仍未完成,再依次将它放入第三队列,……,如此下去,当一个长作业(进程)从第一队列依次降到第n队列后,在第n队列中便采取按时间片轮转的方式运行.

(3) 仅当第一队列空闲时,调度程序才调度第二队列中的进程运行; 仅当第1~(i-1) 队列均空时,才会调度第i队列中的进程运行.如果处理机正在第i队列中为某进程服务时,又有新进程进入优先权较高的队列(第1~(i-1)中的任何一个队列),则此时新进程将抢占正在运行进程的处理机,即由调度程序把正在运行的进程放回到第i队列的末尾,把处理机分配给新到的高优先权进程.?

性能

(1)终端型作业用户

(2) 短批处理作业用户

(3) 长批处理作业用户

满足了多数用户的需求

## 2）优先权调度算法

1、优先权调度算法的类型

非抢占式优先权算法

在这种方式下,系统一旦把处理机分配给就绪队列中优先权最高的进程后,该进程便一直执行下去,直至完成; 或因发生某事件使该进程放弃处理机时,系统方可再将处理机重新分配给另一优先权最高的进程.这种调度算法主要用于批处理系统中;也可用于某些对实时性要求不严的实时系统中.

抢占式优先权调度算法

系统同样把处理机分配给优先权最高的进程,使之执行.但在其执行期间,只要又出现了另一个其优先权更高的进程,进程调度程序就立即停止当前进程(原优先权最高的进程)的执行,重新将处理机分配给新到的优先权最高的进程.

这种抢占式的优先权调度算法,能更好地满足紧迫作业的要求,常用于要求比较严格的实时系统中, 以及对性能要求较高的批处理和分时系统中.

2、优先权的类型

(1) 静态优先权

静态优先权是在创建进程时确定的,且在进程的整个运行期间保持不变.

一般地,优先权是利用某一范围内的一个整数来表示的,例如,0~7或0~255中的某一整数, 又把该整数称为优先数.只是具体用法各异:有的系统用"0"表示最高优先权,当数值愈大时,其优先权愈低;而有的系统恰恰相反.

确定进程优先权的依据有如下三个方面:

1.进程类型.(系统进程/用户进程)

2.进程对资源的需求.(需求量的大小)

3.用户要求.(用户进程紧迫程度)

(2) 动态优先权

动态优先权是指在创建进程时所赋予的优先权,可以随进程的推进或随其等待时间的增加而改变的,以便获得更好的调度性能.

例如,我们可以规定,在就绪队列中的进程,随其等待时间的增长,其优先权以速率a提高.若所有的进程都具有相同的优先权初值,则显然是最先进入就绪队列的进程,将因其动态优先权变得最高而优先获得处理机,此即FCFS算法.

优先权的变化规律可描述为:

由于等待时间与服务时间之和,就是系统对该作业的响应时间,故该优先权又相当于响应比RP.据此,又可表示为:

3,高响应比优先调度算法

由上面的式子可以得到以下结论:

(1) 如果作业的等待时间相同,则要求服务的时间愈短,其优先权愈高,因而该算法有利于短作业.

(2) 当要求服务的时间相同时,作业的优先权决定于其等待时间,等待时间愈长,其优先权愈高,因而它实现的是先来先服务.

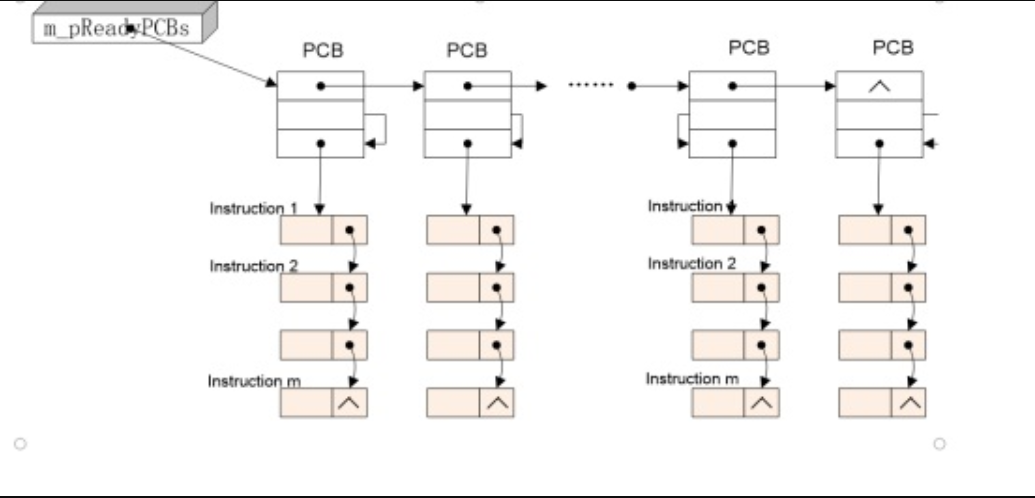
(3) 对于长作业,作业的优先级可以随等待时间的增加而提高,当其等待时间足够长时,其优先级便可升到很高, 从而也可获得处理机.

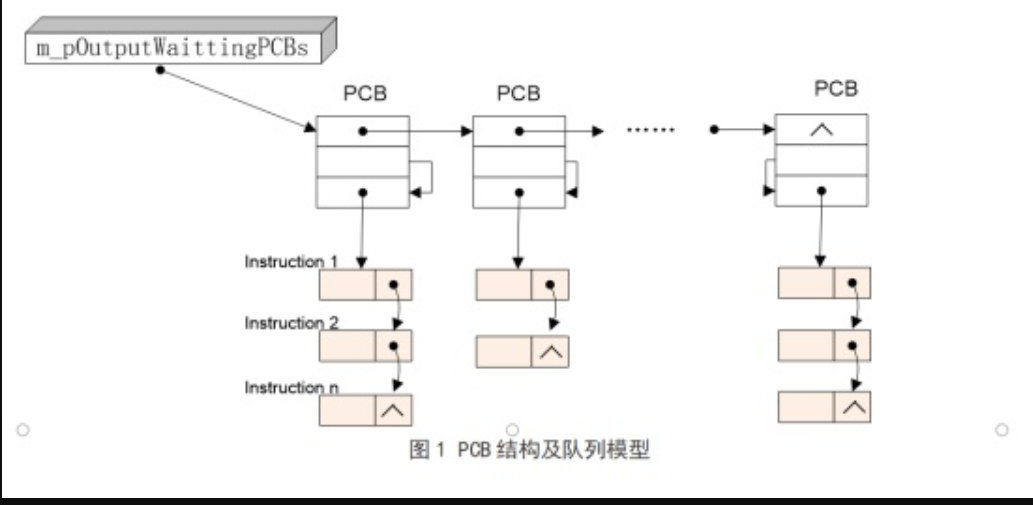
该算法照顾了短作业,且不会使长作业长期得不到服务

# 六、需求分析

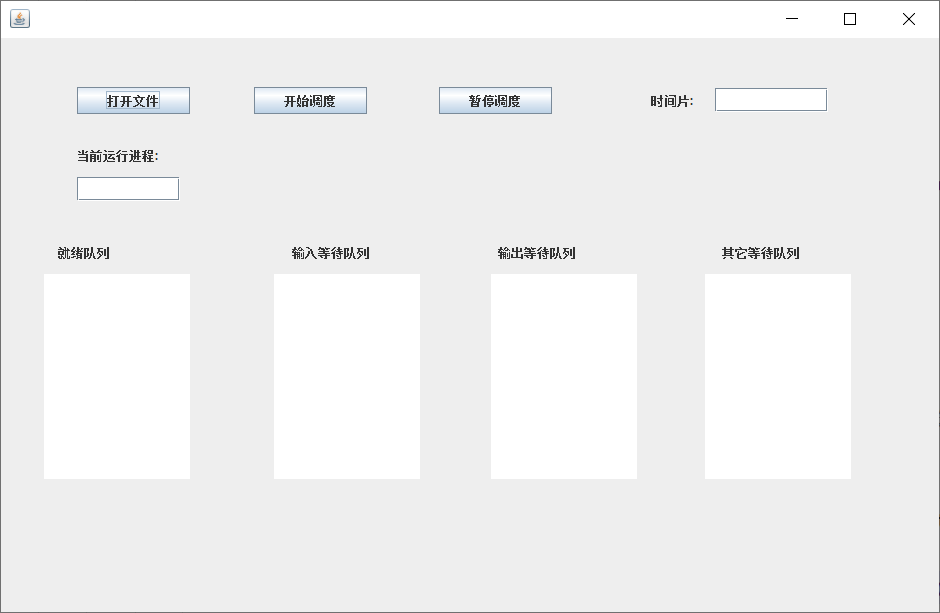
在计算机系统中，可能同时有数百个批处理作业存放在磁盘的作业队列中，或者有数百个终端与主机相连接，这样一来内存和处理器等资源便供不应求。如何从这些作业中挑选作业进入主存运行、如何在进程之间分配处理器时间，无疑是操作系统资源管理中的一个重要问题。因此引入处理器调度用来完成涉及处理器分配的工作，而其中的低级调度，执行分配CPU 的程序即分派程序(dispatcher)，它是操作系统最为核心的部分，执行十分频繁，低级调度策略优劣直接影响到整个系统的性能。  
引入的目的是按照某种原则决定就绪队列中的哪个进程或内核级线程能获得处理器，并将处理器出让给它进行工作。

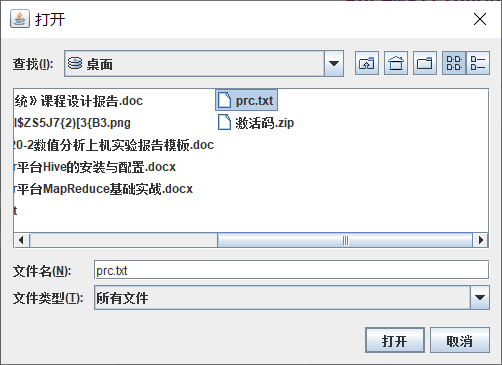
# 七、总体设计

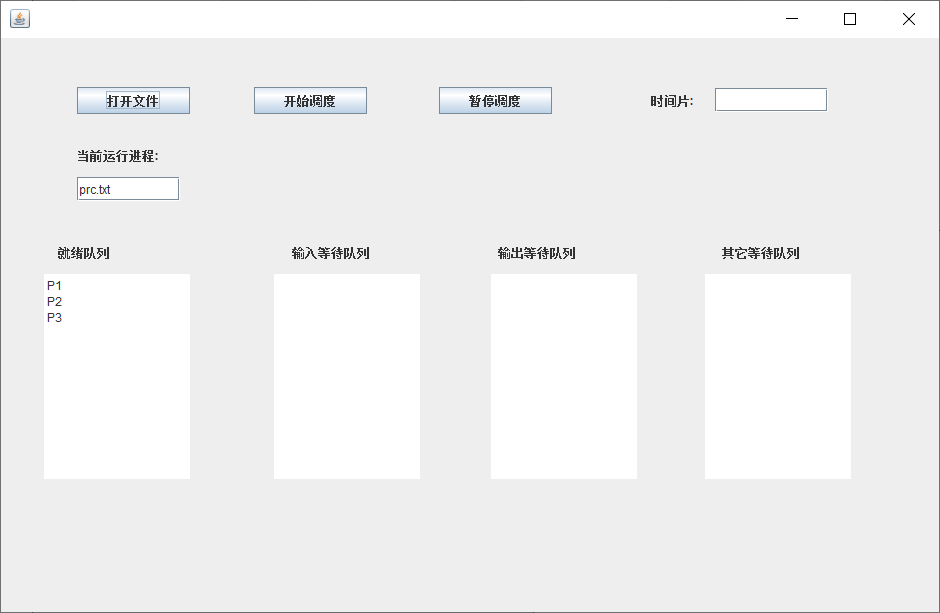




# 八、详细设计与实现[含代码和实现界面]







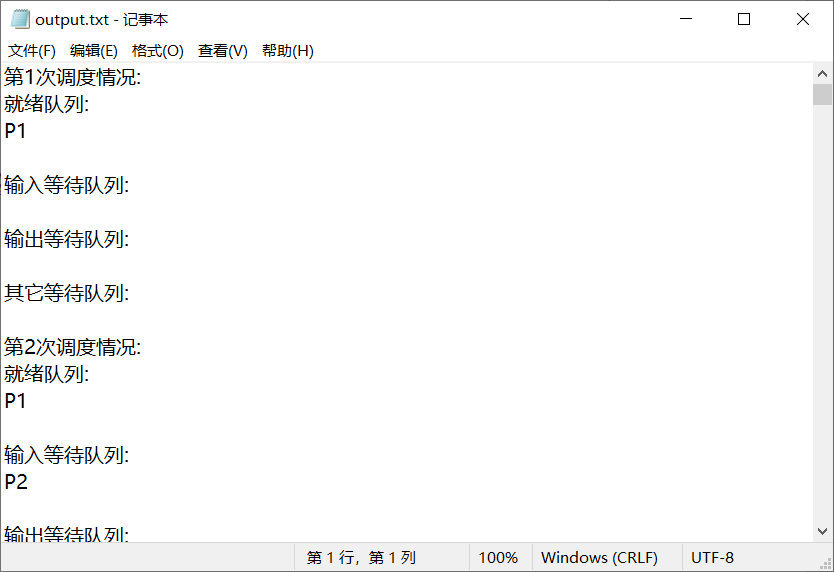
这里必须先把时间片输入进去否则的话就不能进行调度

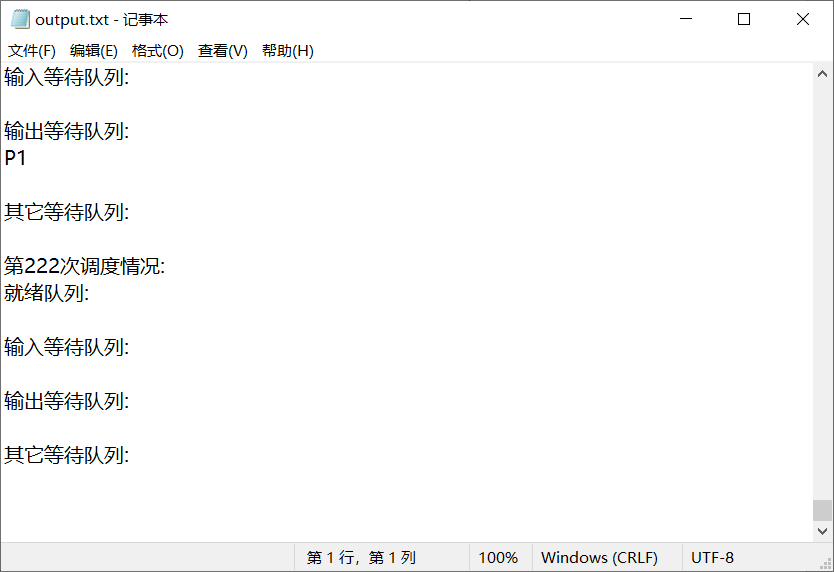


开始调度：



调度完成后就在相应位置上有[输出的文件](https://github.com/Futureword123456/JAVA_Study/commit/fa6e402d656ddae74953ac7949b8d085be8a732d)





# 九、主要代码分析：

(1)、进程控制块(Processing Control Block),是操作系统核心中一种数据结构，主要表示进程状态，其作用是使一个程序成为一个能够独立运行的基本单位，并且可以并发执行的进程。或者说，OS是根据PCB来对并发执行的进程进行控制和管理。PCB通常是占用系统内存中一块连续的内存空间，存放着操作系统用于描述进程情况及控制进程运行的全部信息。定义一个PCB控制块

class PCB {

public String pname;//进程名称

public List<Instruction> pInstructions;//进程中指针列表

public int CurrentInstruction;//当前运行指令索引

PCB() {

this.pInstructions = new ArrayList<Instruction>();

}

}

(2)、定义指令类型

class Instruction {

public char IName;//指令类型

public double IRemainTime;//指令运行时间

public double IRuntime;////指令剩余运行时间

Instruction() {

}

}

创建一个java类：kexhe.Java

先定义这个javaswing组件，这个可视化的界面

JTextField text\_timeslice = new JTextField();

JTextField textRunningProcess = new JTextField();

JTextPane text\_ready\_queue = new JTextPane();

JTextPane text\_iwait\_queue = new JTextPane();

JTextPane text\_other\_queue = new JTextPane();

JTextPane text\_owait\_queue = new JTextPane();

JTextField text\_count = new JTextField();

//创建各种队列，用ArrayList实现

List<PCB> AllQueue = new ArrayList<PCB>();

List<PCB> ReadyQueue = new ArrayList<PCB>();

List<PCB> InputWaitingQueue = new ArrayList<PCB>();

List<PCB> OutputWaitingQueue = new ArrayList<PCB>();

List<PCB> PureWaitingQueue = new ArrayList<PCB>();

private JPanel contentPane;

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
setBounds(100, 100, 956, 621);  
**contentPane** = **new** JPanel();  
**contentPane**.setBorder(**new** EmptyBorder(5, 5, 5, 5));  
setContentPane(**contentPane**);  
**contentPane**.setLayout(**null**);  
JLabel label = **new** JLabel(**"\u65F6\u95F4\u7247:"**);  
label.setBounds(650, 53, 60, 18);  
**contentPane**.add(label);  
  
JButton btnOpenFile = **new** JButton(**"\u6253\u5F00\u6587\u4EF6"**);  
btnOpenFile.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent arg0) {  
 JFileChooser chooser = **new** JFileChooser();  
 **if** (chooser.showOpenDialog(btnOpenFile) == JFileChooser.***APPROVE\_OPTION***) {*//* File file = chooser.getSelectedFile();  
 **textRunningProcess**.setText(file.getName());  
 readFile(file);  
 }  
 ;  
 }  
});  
btnOpenFile.setBounds(76, 49, 113, 27);  
**contentPane**.add(btnOpenFile);  
*//开始调度按钮事件*JButton btnStartSchedule = **new** JButton(**"\u5F00\u59CB\u8C03\u5EA6"**);  
btnStartSchedule.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
 **if** (**text\_timeslice**.getText().equals(**""**)) {  
 JOptionPane.*showMessageDialog*(**null**, **"请输入时间片大小!"**);  
 } **else** {  
 TimerTask task = **new** TimerTask() {  
 **public void** run() {  
 RunOneTime();  
  
 }  
 };  
 btnStartSchedule.setEnabled(**false**);  
 **time**.schedule(task, **new** Date(), Long.*parseLong*(**text\_timeslice**.getText()));  
 }  
 }  
});  
btnStartSchedule.setBounds(253, 49, 113, 27);  
**contentPane**.add(btnStartSchedule);  
*//暂停调度按钮事件*JButton btnPauseSchedule = **new** JButton(**"\u6682\u505C\u8C03\u5EA6"**);  
btnPauseSchedule.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
 **time**.cancel();  
 }  
});  
btnPauseSchedule.setBounds(438, 49, 113, 27);  
**contentPane**.add(btnPauseSchedule);  
**text\_timeslice**.setBounds(714, 50, 113, 24);  
**contentPane**.add(**text\_timeslice**);  
**text\_timeslice**.setColumns(10);  
JLabel label\_1 = **new** JLabel(**"\u5F53\u524D\u8FD0\u884C\u8FDB\u7A0B:"**);  
label\_1.setBounds(76, 108, 103, 18);  
**contentPane**.add(label\_1);  
**textRunningProcess**.setBounds(76, 139, 103, 24);  
**contentPane**.add(**textRunningProcess**);  
**textRunningProcess**.setColumns(10);  
JLabel label\_2 = **new** JLabel(**"\u5C31\u7EEA\u961F\u5217"**);  
label\_2.setBounds(57, 205, 72, 18);  
**contentPane**.add(label\_2);  
JLabel label\_3 = **new** JLabel(**"\u8F93\u5165\u7B49\u5F85\u961F\u5217"**);  
label\_3.setBounds(291, 205, 96, 18);  
**contentPane**.add(label\_3);  
JLabel label\_4 = **new** JLabel(**"\u8F93\u51FA\u7B49\u5F85\u961F\u5217"**);  
label\_4.setBounds(497, 205, 96, 18);  
**contentPane**.add(label\_4)  
JLabel label\_5 = **new** JLabel(**"\u5176\u5B83\u7B49\u5F85\u961F\u5217"**);  
label\_5.setBounds(721, 205, 96, 18);  
**contentPane**.add(label\_5);  
**text\_ready\_queue**.setBounds(43, 236, 146, 205);  
**contentPane**.add(**text\_ready\_queue**);  
**text\_iwait\_queue**.setBounds(273, 236, 146, 205);  
**contentPane**.add(**text\_iwait\_queue**);  
**text\_owait\_queue**.setBounds(490, 236, 146, 205);  
**contentPane**.add(**text\_owait\_queue**);  
**text\_other\_queue**.setBounds(704, 236, 146, 205);  
**contentPane**.add(**text\_other\_queue**);

到这里主界面就设计好了，接下来就是读取文件

**public void** readFile(File file) {*//读文件* BufferedReader bReader;  
 **try** {  
 bReader = **new** BufferedReader(**new** FileReader(file));  
 StringBuffer sBuffer = **new** StringBuffer();  
 String str;  
 **while** ((str = bReader.readLine()) != **null**) {  
 sBuffer.append(str + **'\n'**);  
 **switch** (str.charAt(0)) {  
 **case 'P'**:  
 PCB p = **new** PCB();  
 p.**pname** = str;  
 **AllQueue**.add(p);  
 **break**;  
 **case 'C'**:  
 Instruction ins1 = **new** Instruction();  
 ins1.**IRemainTime** = Integer.*parseInt*(str.substring(1, 3));  
 ins1.**IName** = **'C'**;  
 **AllQueue**.get(**AllQueue**.size() - 1).**pInstructions**.add(ins1);  
 **break**;  
 **case 'I'**:  
 Instruction ins2 = **new** Instruction();  
 ins2.**IRemainTime** = Integer.*parseInt*(str.substring(1, 3));  
 ins2.**IName** = **'I'**;  
 **AllQueue**.get(**AllQueue**.size() - 1).**pInstructions**.add(ins2);  
 **break**;  
 **case 'O'**:  
 Instruction ins3 = **new** Instruction();  
 ins3.**IRemainTime** = Integer.*parseInt*(str.substring(1, 3));  
 ins3.**IName** = **'O'**;  
 **AllQueue**.get(**AllQueue**.size() - 1).**pInstructions**.add(ins3);  
 **break**;  
 **case 'W'**:  
 Instruction ins4 = **new** Instruction();  
 ins4.**IRemainTime** = Integer.*parseInt*(str.substring(1, 3));  
 ins4.**IName** = **'W'**;  
 **AllQueue**.get(**AllQueue**.size() - 1).**pInstructions**.add(ins4);  
 **break**;  
 **case 'H'**:  
 Instruction ins5 = **new** Instruction();  
 ins5.**IRemainTime** = Integer.*parseInt*(str.substring(1, 3));  
 ins5.**IName** = **'H'**;  
 **AllQueue**.get(**AllQueue**.size() - 1).**pInstructions**.add(ins5);  
 **break**;  
 }*//switch* }*//while* String s1 = **""**;  
 **for** (PCB p : **AllQueue**) {  
 s1 = s1 + p.**pname** + **"\r\n"**;  
 }  
 **text\_ready\_queue**.setText(s1);  
} **catch** (Exception e) {  
 *//* ***TODO: handle exception*** }  
 }

*每隔一个时间间隔运行一次*

**public void** RunOneTime() {  
 **int** size = **AllQueue**.size();  
 **if** (**num** < size) {  
 PCB p = **AllQueue**.get(**num**);  
 **switch** (p.**pInstructions**.get(0).**IName**) {  
 **case 'C'**:  
 **ReadyQueue**.add(p);  
 **break**;  
 **case 'I'**:  
 **InputWaitingQueue**.add(p);  
 **break**;  
 **case 'O'**:  
 **OutputWaitingQueue**.add(p);  
 **break**;  
 **case 'W'**:  
 **PureWaitingQueue**.add(p);  
 **break**;  
 **case 'H'**:  
 **break**;  
 }  
 **num**++;  
 }  
 *//只要有队列不为空就一直调度* **if** (**ReadyQueue**.size() != 0 || **InputWaitingQueue**.size() != 0 || **OutputWaitingQueue**.size() != 0 || **PureWaitingQueue**.size() != 0) {  
  
 *//只有一个CPU，所以只能对就绪队列的第一个PCB的运行指令时间进行减一操作* **if** (**ReadyQueue**.size() != 0) {  
 **if** (**ReadyQueue**.get(0).**pInstructions**.get(**ReadyQueue**.get(0).**CurrentInstruction**).**IRemainTime** == 0) {  
 **ReadyQueue**.get(0).**CurrentInstruction**++;  
 **switch** (**ReadyQueue**.get(0).**pInstructions**.get(**ReadyQueue**.get(0).**CurrentInstruction**).**IName**) {  
 **case 'C'**:  
 **ReadyQueue**.add(**ReadyQueue**.get(0));*//先加再删，ArrayList的好处，动态添加和删除，[1,2,3]把1删了再加，就是[2,,3,1]了* **ReadyQueue**.remove(**ReadyQueue**.get(0));  
 **break**;  
 **case 'I'**:  
 **InputWaitingQueue**.add(**ReadyQueue**.get(0));  
 **ReadyQueue**.remove(**ReadyQueue**.get(0));  
 **break**;  
 **case 'O'**:  
 **OutputWaitingQueue**.add(**ReadyQueue**.get(0));  
 **ReadyQueue**.remove(**ReadyQueue**.get(0));  
 **break**;  
 **case 'W'**:  
 **PureWaitingQueue**.add(**ReadyQueue**.get(0));  
 **ReadyQueue**.remove(**ReadyQueue**.get(0));  
 **break**;  
 **case 'H'**:  
 **ReadyQueue**.remove(**ReadyQueue**.get(0));  
 **break**;  
 }  
 } **else** {  
 **ReadyQueue**.get(0).**pInstructions**.get(**ReadyQueue**.get(0).**CurrentInstruction**).**IRemainTime**--;  
 **textRunningProcess**.setText(**ReadyQueue**.get(0).**pname**);  
 **ReadyQueue**.add(**ReadyQueue**.get(0));  
 **ReadyQueue**.remove(**ReadyQueue**.get(0));  
 }  
 }  
 *//输入等待队列和输出等待队列以及其它等待队列之所以采用for循环是因为任务书上说I/O设备不限，所以可以对这几个队列的每个PCB进行减一* **if** (**InputWaitingQueue**.size() != 0) {  
 **for** (**int** i = 0; i <= **InputWaitingQueue**.size() - 1; i++) {  
 **if** (**InputWaitingQueue**.get(i).**pInstructions**.get(**InputWaitingQueue**.get(i).**CurrentInstruction**).**IRemainTime** == 0) {  
 **InputWaitingQueue**.get(i).**CurrentInstruction**++;  
 **switch** (**InputWaitingQueue**.get(i).**pInstructions**.get(**InputWaitingQueue**.get(i).**CurrentInstruction**).**IName**) {  
 **case 'C'**:  
 **ReadyQueue**.add(**InputWaitingQueue**.get(i));  
 **InputWaitingQueue**.remove(**InputWaitingQueue**.get(i));  
 **break**;  
 **case 'I'**:  
 **InputWaitingQueue**.add(**InputWaitingQueue**.get(i));  
 **InputWaitingQueue**.remove(**InputWaitingQueue**.get(i));  
 **break**;  
 **case 'O'**:  
 **OutputWaitingQueue**.add(**InputWaitingQueue**.get(i));  
 **InputWaitingQueue**.remove(**InputWaitingQueue**.get(i));  
 **break**;  
 **case 'W'**:  
 **PureWaitingQueue**.add(**InputWaitingQueue**.get(i));  
 **InputWaitingQueue**.remove(**InputWaitingQueue**.get(i));  
 **break**;  
 **case 'H'**:  
 **InputWaitingQueue**.remove(**InputWaitingQueue**.get(i));  
 **break**;  
 }  
 } **else** {  
 **InputWaitingQueue**.get(i).**pInstructions**.get(**InputWaitingQueue**.get(i).**CurrentInstruction**).**IRemainTime**--;  
 }  
 }  
 }  
  
 **if** (**OutputWaitingQueue**.size() != 0) {  
 **for** (**int** i = 0; i <= **OutputWaitingQueue**.size() - 1; i++) {  
 **if** (**OutputWaitingQueue**.get(i).**pInstructions**.get(**OutputWaitingQueue**.get(i).**CurrentInstruction**).**IRemainTime** == 0) {  
 **OutputWaitingQueue**.get(i).**CurrentInstruction**++;  
 **switch** (**OutputWaitingQueue**.get(i).**pInstructions**.get(**OutputWaitingQueue**.get(i).**CurrentInstruction**).**IName**) {  
 **case 'C'**:  
 **ReadyQueue**.add(**OutputWaitingQueue**.get(i));  
 **OutputWaitingQueue**.remove(**OutputWaitingQueue**.get(i));  
 **break**;  
 **case 'I'**:  
 **InputWaitingQueue**.add(**OutputWaitingQueue**.get(i));  
 **OutputWaitingQueue**.remove(**OutputWaitingQueue**.get(i));  
 **break**;  
 **case 'O'**:  
 **OutputWaitingQueue**.add(**OutputWaitingQueue**.get(i));  
 **OutputWaitingQueue**.remove(**OutputWaitingQueue**.get(i));  
 **break**;  
 **case 'W'**:  
 **PureWaitingQueue**.add(**OutputWaitingQueue**.get(i));  
 **OutputWaitingQueue**.remove(**OutputWaitingQueue**.get(i));  
 **break**;  
 **case 'H'**:  
 **OutputWaitingQueue**.remove(**OutputWaitingQueue**.get(i));  
 **break**;  
 }  
 } **else** { **OutputWaitingQueue**.get(i).**pInstructions**.get(**OutputWaitingQueue**.get(i).**CurrentInstruction**).**IRemainTime**--;  
 }  
 }  
 }  
 **if** (**PureWaitingQueue**.size() != 0) {  
 **for** (**int** i = 0; i <= **PureWaitingQueue**.size() - 1; i++) {  
 **if** (**PureWaitingQueue**.get(i).**pInstructions**.get(**PureWaitingQueue**.get(i).**CurrentInstruction**).**IRemainTime** == 0) {  
 **PureWaitingQueue**.get(i).**CurrentInstruction**++;  
 **switch** (**PureWaitingQueue**.get(i).**pInstructions**.get(**PureWaitingQueue**.get(i).**CurrentInstruction**).**IName**) {  
 **case 'C'**:  
 **ReadyQueue**.add(**PureWaitingQueue**.get(i));  
 **PureWaitingQueue**.remove(**PureWaitingQueue**.get(i));  
 **break**;  
 **case 'I'**:  
 **InputWaitingQueue**.add(**PureWaitingQueue**.get(i));  
 **PureWaitingQueue**.remove(**PureWaitingQueue**.get(i));  
 **break**;  
 **case 'O'**:  
 **OutputWaitingQueue**.add(**PureWaitingQueue**.get(i));  
 **PureWaitingQueue**.remove(**PureWaitingQueue**.get(i));  
 **break**;  
 **case 'W'**:  
 **PureWaitingQueue**.add(**PureWaitingQueue**.get(i));  
 **PureWaitingQueue**.remove(**PureWaitingQueue**.get(i));  
 **break**;  
 **case 'H'**:  
 **PureWaitingQueue**.remove(**PureWaitingQueue**.get(i));  
 **break**;  
 }  
 } **else** {  
 **PureWaitingQueue**.get(i).**pInstructions**.get(**PureWaitingQueue**.get(i).**CurrentInstruction**).**IRemainTime**--;  
 }  
 }  
 }  
 *//所有用到CPU资源的PCB都没了就清空这个控件* **if** (**ReadyQueue**.size() == 0) {  
 **textRunningProcess**.setText(**""**);  
 }  
 *//每进行一次调度就更新一次控件的信息* String str1 = **""**;  
 **for** (**int** i = 0; i < **ReadyQueue**.size(); i++) {  
 str1 = str1 + **ReadyQueue**.get(i).**pname** + **"\r\n"**;  
 }  
 **text\_ready\_queue**.setText(str1);  
 String str2 = **""**;  
 **for** (**int** i = 0; i < **InputWaitingQueue**.size(); i++) {  
 str2 = str2 + **InputWaitingQueue**.get(i).**pname** + **"\r\n"**;  
 }  
 **text\_iwait\_queue**.setText(str2);  
 String str3 = **""**;  
 **for** (**int** i = 0; i < **OutputWaitingQueue**.size(); i++) {  
 str3 = str3 + **OutputWaitingQueue**.get(i).**pname** + **"\r\n"**;  
 }  
 **text\_owait\_queue**.setText(str3);  
 String str4 = **""**;  
 **for** (**int** i = 0; i < **PureWaitingQueue**.size(); i++) {  
 str4 = str4 + **PureWaitingQueue**.get(i).**pname** + **"\r\n"**;  
 }  
 **text\_other\_queue**.setText(str4);  
 **count**++;  
 String schedule = **"第"** + String.*valueOf*(**count**) + **"次调度情况:\r\n"** + **"就绪队列:\r\n"** + str1 + **"\r\n"** + **"输入等待队列:\r\n"** + str2 + **"\r\n"** + **"输出等待队列:\r\n"** + str3 + **"\r\n"** + **"其它等待队列:\r\n"** + str4 + **"\r\n"**;  
 *writeFile*(schedule);  
 } **else** {  
 JOptionPane.*showMessageDialog*(**null**, **"调度结束！"**);  
 **time**.cancel();  
 }  
}

到这里这个程序就完成了，所有的代码在[Github](https://github.com/Futureword123456/JAVA_Study/tree/master/src/com/yang/day27)上

# 十、总结

通过本次课程设计，我学到了很多的实用性知识。除了更深的了解这几个算法，而且对C++语言进行了复习，回忆、查询、巩固了操作系统与java相关的知识。 看懂了**算法思想（理论）**是一回事，用**代码实现（实践）**又是一回事。

本次课设虽然完成了，还是有许多不足之处，希望后期的努力让自己的作品更加优秀。在整个课程设计中，老师对我们耐心的指导与帮助才使我们可以顺利完成此次课程设计，再次衷心感谢老师。

指导老师意见：

成绩: 教师签名：

年 月 日