



# 操作系统课程设计报告

两道批处理系统的作业调度-2

学 院 计算机学院

专 业 计算机科学与技术

姓 名 黄钰竣

学 号 3117004568

指导老师 申建芳

2019年12月

# 课程设计选题介绍

## （一）题目

两道批处理系统的两级调度-2

## （二）主要内容

本课程设计要求模拟实现一个的两道批处理系统的两级调度。通过具体的作业调度、进程调度等功能的实现，加深对批处理系统的两级调度模型和实现过程的理解。

## （三）题目要求

**内存任何时候最多只允许有两道作业。**要求作业从进入系统到最后完成，要经历两级调度：作业调度和进程调度。作业调度是高级调度，它的主要功能是根据一定的算法，为它们建立初始状态为就绪的作业进程。进程调度是低级调度，它的主要功能是根据一定的算法将CPU分派给就绪队列中的一个进程。

1. 作业调度分别采用先来先服务算法，进程调度采用可抢占的优先级调度算法。

（2）假定“预输入”程序已经把一批作业的信息存放在输入井了，并为它们建立了相应作业表。测试数据如下：

**作业 进入时间 估计运行时间 优先数**

**JOB1 10：00 40分钟 5**

**JOB2 10：20 30分钟 3**

**JOB3 10：30 50分钟 4**

**JOB4 10：50 20分钟 6**

（3）优先数越小的优先级别越高。分别在不同算法控制下运行设计的程序，依次显示被选中作业、内存空闲区和磁带机的情况。比较不同算法作业的选中次序及作业平均周转时间。

选用程序设计语言：C、C＋＋等。

# 设计思路

## （一）数据对象结构设计

### 1.控制块存储结构设计

由于是模拟两道批处理作业调度，所以处理机处理的作业经过了从外存到内存位置的变化。就存储控制块而言，涉及JCB（作业控制块，Job Control Block）和PCB（进程控制块，Process Control Block），两者分别存在于外存和内存，一个负责为作业调度提供管理信息，一个则负责为进程调度提供管理信息。两者虽然作用范围以及名字上有区别，但实质为作业调度和进程调度算法提供的管理信息相似，应该具有相似的存储结构。所以在本模拟程序中，JCB与PCB会使用两个不同的类作为区别，并且存在一个从JCB到PCB的转换方法，但两者的数据结构是相似的。

JCB的存储结构为：

{[作业名],[到达时间],[需运行时间],[优先级数]}

PCB的存储结构为：

{[进程名],[到达时间],[需运行时间],[进程状态],[优先级数],[已运行时间],[完成时间]}

此外，作为进程的模拟对象。JCB应该具有“运行”方法，用以模拟进程获得处理机后执行。

### 2.作业/进程（就绪、完成）队列结构

在面向对象语言中，一般都封装好了可以存储数据对象的队列，所以本程序直接使用语言基础包中提供的实现类用以存储控制块。

值得指出的是，为了便于实现的各个调度算法，队列中应该实现根据到达时间、优先数、等控制信息，对作业/进程进行排序的操作，用以为取出调度对象并执行后续操作做准备。

### 3.时钟模拟

为了模拟时间的变化，应该设置一个时钟对象，用以记录时间。根据题目中给出的模拟时间数据，时钟的存储结构应该包括小时和分钟。

## （二）UI界面设计

为了更好的显示出作业在计算机中执行的变化，应该设计一种能够既能够显示进程状态信息，又能够动态展现作业在各个队列里移动的执行过程界面。综合考虑上述的因素，以及参考老师提供的课程设计样例，我借鉴使用了以共6张表格形式展现的“外存队列”、“内存队列”、“完成队列”以及“正在执行进程”图形界面设计方案。该方案能够通过不断更新更新各表格信息，实现对作业状态的监控。

同时，为了能够使用户能够更加仔细的查看各进程的运行信息，本程序为进程的运行增设了“暂停运行”以及“继续运行”功能，用以暂停进程的执行，方便用户停下来查看当前时刻各进程的信息。

设计的图形用户界面如下：



图1 UI界面设计效果图

# 三、程序流程图

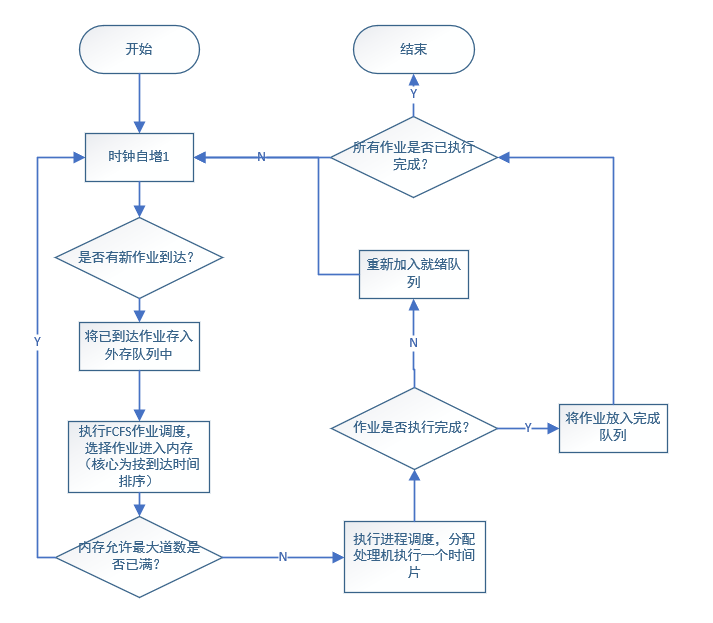


图2 模拟两道批处理程序流程图

# 四、程序执行结果

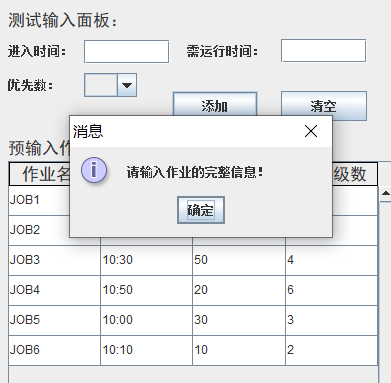


图3 未输入完整信息

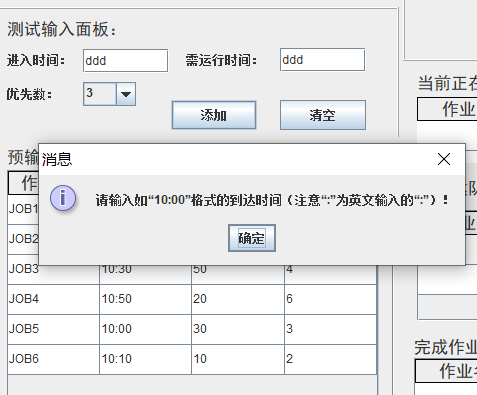


图4 未按格式输入“进入时间”信息

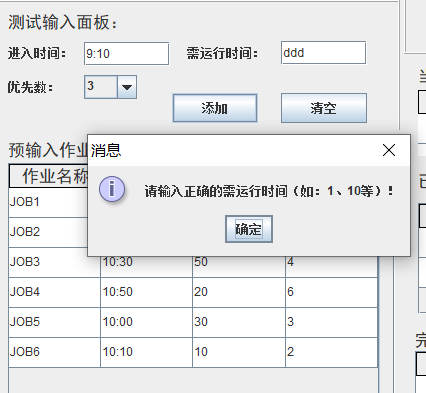


图5 未按格式输入“需运行时间”



图6运行时程序界面图



图7 运行暂停结果图



图8 运行结束图

# 五、关键性代码

此代码按照上面介绍到的程序流程执行，模拟了从作业到达，到进入停留外存，到经先来先服务作业调度算法调入内存（实现PCB到JCB的转换），再到优先级优先调度算法调度分配处理机执行，最后进入作业完成队列的过程，为程序的核心部分。

**public** **void** runSystem() {

storeList.sortByTime(); // 对输入队列按到达时间排序，排序后就无需再比较所有元素

**int** num = storeList.size();

JCB firstJob = storeList.pop();

Time clock = firstJob.getArrivalTime();

pcbWList.add(**new** PCB(firstJob));

**while** (pcbFList.size() < num && !isStop) {

**if** (isStop) {

**break**;

}

// 模拟作业按照时间到达

**for** (**int** i = 0; i < storeList.size(); i++) {

**if** (clock.compareTo(storeList.getFirstTime()) == 0) {

jcbWList.add(storeList.pop());

} **else** {

// 不做无用的比较

**break**;

}

}

**if** (pcbWList.size() < MAXPROCESS) {

// 进行作业调度，采用先来先服务调度算法

jcbWList.sortByTime();

**while** (pcbWList.size() < MAXPROCESS) {

// 从外存队列中获取元素添加到就绪队列中，补足允许的最大道数数量

**if** (jcbWList.size() > 0) {

pcbWList.add(**new** PCB(jcbWList.pop()));

} **else** {

**break**;

}

}

}

**if** (pcbWList.size() > 0) {

// 进行进程调度，采用可抢占的优先级调度

pcbWList.sortByPriority();

// 集中更新界面信息

**new** Thread(**new** Runnable() {

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

changeTables(pcbWList.getFirst());

pcbWList.updateTable();

setClockOutput(clock);

}

}).start();

**try** {

Thread.*sleep*(DELAY);

} **catch** (Exception e) {

System.*exit*(0);// 退出程序

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 暂停操作 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**while** (isPause) {

**try** {

Thread.*sleep*(1000);

} **catch** (Exception e) {

System.*exit*(0);// 退出程序

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 点击重置发生终止操作 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**if** (isStop) {

**break**;

}

PCB pcb = pcbWList.pop();

// 进程运行的时候会停留1秒，便于观察

// 运行进程

**if** (pcb.run(TIMESLICE) == Status.***WAIT***) {

// 运行完一个时间片后，作业仍未完成

pcbWList.add(pcb);

} **else** {

// 作业已完成

pcb.setFinishedTime(clock);

pcbFList.add(pcb);

}

}

System.***out***.println("In " + clock.toString());

System.***out***.println("存储队列：");

storeList.displayList();

System.***out***.println("外存队列：");

jcbWList.displayList();

System.***out***.println("就绪队列：");

pcbWList.displayList();

System.***out***.println("完成队列：");

pcbFList.displayList();

System.***out***.println();

clock.increase(TIMESLICE);

}

// 集中更新界面信息

**new** Thread(**new** Runnable() {

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

changeTables(**null**);

}

}).start();

}

// 更新正在运行列表

**private** **void** changeTables(PCB pcb) {

**if** (pcb == **null**) {

**for** (**int** i = 0; i < 5; i++) {

table\_1.setValueAt("", 0, i);

}

} **else** {

// 更新正在执行作业列表

table\_1.setValueAt(pcb.getName(), 0, 0);

table\_1.setValueAt(pcb.getArrivalTimeString() + "", 0, 1);

table\_1.setValueAt(pcb.getRunedTime() + "", 0, 2);

table\_1.setValueAt(pcb.getJobSize() + "", 0, 3);

table\_1.setValueAt(pcb.getPriority() + "", 0, 4);

}

// 更新内存就绪、完成作业、外存就绪列表

pcbWList.updateTable();

pcbFList.updateTable();

jcbWList.updateTable(5);

// //更新界面

// panel\_3.validate();

// panel\_4.validate();

// panel\_6.validate();

// panel\_7.validate();

}

# 六、开发过程遇到的问题及解决方案

## 界面更新类问题

在使用Swing实现用户图形界面的时候，如何使得“当前时钟面板”正确显示并及时更新时间，以及动态更新各个进程状态表（存储队列、外存队列、内存队列、完成队列）花费了我最多的调试时间。

我在实际中遇到的情况，主要是在对进程模拟执行完毕后，界面才更新显示，显示的仅是所有进程运行完毕的状态信息。通过在网络上查找相关的Swing界面动态更新的信息，我初步对问题产生的原因有了几种猜测，即：

第一种，未调用界面更新函数以重画界面。

第二种，更新只能由UI线程实现。即问题产生的原因是，程序界面仅在某个组件的函数方法执行完成后更新。由于编写处理程序是在按钮的响应方法中调用，且程序按顺序处理步骤较多，在组件返回前进行了多次的界面修改，最终使得仅剩最后一次界面修改得以显示（其他修改则丢失）。

针对上述的猜想，我逐一的进行了排查和调试，最终发现了问题产生的真正原因，对界面的更新产生的新的认识。

程序在进行界面更新的，执行的步骤为：首先，由主函数产生一个初始化图形界面线程，该线程负责在程序执行初期绘制界面，但其并不会绘制完界面后马上结束，而是会用于处理界面产生的图形组件的触发操作（如点击、滑动等），这类线程称为UI线程。我在设计程序时由于未考虑到线程模拟程序所用线程与UI线程之间的关系，所以将模拟线程的处理运算工作，都放在了UI线程上，这就造成了UI线程的负担过重，导致其无法响应界面的更新。

问题的解决方法是，在“开始执行”按钮触发后，不是直接调用进程模拟函数，而是先创建一个新线程，将模拟函数的操作交由新线程完成，从而保证UI线程的正常执行。同时，为了让“时钟”、“队列状态”部分界面更新的更及时，我也单独开辟了一个新线程用于处理此类更新。实际上程序在运行的过程中并不存在“界面仅能由UI线程进行更新”这一说，UI的主要工作为界面初始化和组件的响应操作，界面上的一些元素还是可以通过其他逻辑处理线程动态的改变。

# 七、文件附录

项目文件夹为OSDesign

源码文件夹的路径为：\课程设计\OSDesign\src\OSDesign\

项目主类文件：OSDesign.java

可执行文件路径：\课程设计\两道批处理2（黄钰竣）.exe

其中源代码目录下：

OSDesign.java为界面设计文件，JCB.java、JCBList.java、PCB.java、PCBList.java、Status.java、Time.java为功能模块文件