**成绩：**



**操作系统设计**

**院 系**

**姓 名**

**指导教师**

**二O一九 年 十一 月 二十 日**

操作系统课程设计报告

南京信息工程大学软件工程系，南京 210044

**摘要：**操作系统是计算机学科相关专业的重要课程。本课程报告包含课程设计概要，算法和算例介绍，以及课程的心得体会。

**关键词：**操作系统；多级反馈队列；LRU算法

1. 系统概要设计

本操作系统模拟程序同时体现了进程管理和存储管理。进程管理实现并发执行多个进程（理论上不限数量），体现进程切换，进程调度算法使用非抢占式多级反馈队列。存储管理实现进程的页面访问请求，单个进程分配内存物理块访问多个页面（理论上不限数量），并体现页面置换，请求分页的页面置换算法使用 LRU（最近最少使用）算法。

系统输入：

进程到达时间，进程服务时间，页面请求序列。

系统输出：

进程调度和页面请求信息。

1. 详细设计

2.1进程调度设计

2.1.1 进程数据结构

将单个进程封装成实体类，基本属性包括：进程号，进程状态标识符，进程到达时间，进程服务时间，进程已运行时间。数据结构相关属性包括：页面请求序列，物理块分配表（表示内存分配），页面置换队列。进程号为具体进程的唯一标识符，进程状态标识符标注当前进程的状态（包含等待和运行中两种状态），进程到达时间、服务时间和已运行时间表示进程的生命周期并和状态一起体现进程的切换。其中页面请求序列，物理块分配表，页面置换队列是为进程的页面实现LRU置换算法设计而定义。

2.1.2 进程调度算法（多级反馈队列）

将单个队列封装成实体类，包含队列优先级属性和链表，优先级标识了当前进程队列的优先级，链表是当前进程队列的数据结构。

在程序中声明4个队列，并设置其等级为1、2、3、4，时间片依次增大。再依次输入进程相关的信息，入队列开始进行页面的请求。

2.2物理块和页面置换算法设计

2.2.1物理块设计

在主函数中声明静态成员链式散列表，键、值分别为地址和分配状态，依此扮演内存的角色。每个进程分配的最大物理块数量固定，以分配连续地址的方式分配内存。当当前进程执行完成时释放物理块地址。

2.2.2页面置换算法（LRU）

页面置换LRU算法参照书本上使用队列（或栈）的方式实现，算法图例如下：



算法在每次访问页面时调用，使最近使用的页面总是置顶。而底部则是页面淘汰的根据。

设置队列的最大容纳为分配的最大物理块数，将算法封装成函数，每单位时间调用该函数以实现页面的访问，中断和置换。

1. 核心数据结构与算法实现

3.1进程控制块（PCB）实体类

核心代码如下：

private int pid;//进程标识符  
private String status;//进程状态标识  
private int priority;//进程优先级  
private int life;//进程服务时间  
private int alive;//进程已运行时间  
private int arrival;//进程到达时间  
private ArrayList<Integer> pageMap = new ArrayList<Integer>();//进程请求页序列  
//进程分配的物理块  
private LinkedList<Page> pageBlock = new LinkedList<Page>();  
private LinkedList<Page> lruSerial = new LinkedList<Page>();  
public PCB() {}  
public PCB(int pid, String status, int priority, int life, int arrival) {

……  
 for (int i = 0; i < life; i++) {  
 Random random = new Random();  
 int randomPage = random.nextInt(10);  
 this.pageMap.add(randomPage);  
 }  
}  
public PCB(int pid, String status, int priority, int life, int arrival, int pageNum[]) {……}

3.2进程队列（PCBsQueue）实体类

核心代码如下：

private int priority; //队列优先级  
private LinkedList<PCB> queue = new LinkedList<PCB>();  
public PCBsQueue(int priority) {this.priority = priority;}  
public int getPriority() {return priority;}  
public void setPriority(int priority) {this.priority = priority;}  
public LinkedList<PCB> getQueue() {return queue;}  
public void setQueue(LinkedList<PCB> queue) {this.queue = queue;}

3.3页面置换算法（LRU）

核心代码如下：

public static PCB runLru(PCB pcb) {  
 ArrayList<Integer> arr = pcb.getPageMap();  
 LinkedList<Page> pageBlock = pcb.getPageBlock();  
 LinkedList<Page> pageStack = pcb.getLruSerial();  
 int pageId = arr.get(0);  
 for (int i = 0; i < pageStack.size(); i++) {  
 if (pageStack.get(i).getNum() == pageId) {  
 Page newPage = new Page();  
 newPage.setNum(pageId);  
 pageStack.remove(i);  
 pageStack.add(0, newPage);  
 arr.remove(0);  
 pcb.setPageMap(arr);  
 pcb.setLruSerial(pageStack);  
 return pcb;  
 }  
 }  
 Page p = new Page(pageId);  
 pageStack.add(0, p);  
 if (pageStack.size() > 5) {  
 for (int j = 0; j < pageBlock.size(); j++) {  
 if (pageBlock.get(j).getNum() == pageStack.getLast().getNum()) {  
 pageBlock.set(j, p);  
 break;  
 }  
 }  
 pageStack.remove(5);  
 } else {  
 pageBlock.add(0, p);  
 }  
 arr.remove(0);  
 pcb.setPageMap(arr);  
 pcb.setLruSerial(pageStack);  
 pcb.setPageBlock(pageBlock);  
 return pcb;  
}

3.4进程调度算法（多级反馈队列）

核心代码如下：

while (currentPCBsNum != 0 && !isStopScheduling) {

boolean isEmptyFlag = false;

if (!tmpQueue.isEmpty()&&tmpQueue.getFirst().getArrival() == currentTime) {

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

if (tmpQueue.get(i).getArrival() == currentTime) {

PCB newPcb = tmpQueue.getFirst();

PCBsQueues[PCBsQueues.length - 1].getQueue().offer(newPcb);

tmpQueue.remove(i);

i--;

}

}

}

for (int i = PCBsQueues.length - 1; i >= 0; i--) {

LinkedList<PCB> queue = PCBsQueues[i].getQueue();

if (queue.size() > 0) {

isEmptyFlag = true;

//读取该队列首个PCB

PCB pcb = queue.element();

pcb.setStatus("Running");

int pid = pcb.getPid();

int priority = pcb.getPriority();

int life = pcb.getLife();

int curTimeSlice = PCBsQueuesTimeSlice[i];

ArrayList<Integer> arr = pcb.getPageMap();

while (curTimeSlice > 0) {

while (isPauseScheduling) {

try {

Thread.sleep((int) (1000));

} catch (InterruptedException e) {

e.printStackTrace();

}

}

showPCBQueues(PCBsQueues);

//显示物理块状态

if (pcb.getPageBlock() != null) {

showBlock(pcb);

}

//修改pcb属性

life = life - 1;

curTimeSlice--;

timeLbl.setText("已执行时间： "+ currentTime);

currentTime++;

if (!tmpQueue.isEmpty()&&tmpQueue.getFirst().getArrival() == currentTime) {

/\* PCB newPcb = tmpQueue.getFirst();

PCBsQueues[PCBsQueues.length - 1].getQueue().offer(newPcb);

tmpQueue.remove(0);\*/

for (int j = 0; j < tmpQueue.size(); j++) {

if (tmpQueue.get(j).getArrival() == currentTime) {

PCB newPcb = tmpQueue.getFirst();

PCBsQueues[PCBsQueues.length - 1].getQueue().offer(newPcb);

tmpQueue.remove(j);

j--;

}

}

}

pcb.setLife(life);

pcb.setAlive(pcb.getAlive() + 1);

runLru(pcb);

//通过延时一个时间片来模拟该进程的执行

try {

Thread.sleep((int) (2500));

} catch (InterruptedException e) {

e.printStackTrace();

}

if (life <= 0) {

break;

}

}

//若该进程执行完成

if (life <= 0) {

//移除该队列的首个PCB

queue.poll();

pidsUsed[pid] = 0;

currentPCBsNum--;

}//改变其PCB的相关参数,并插入其优先级所对应的队列尾部

else {

//移除该队列的首个PCB

queue.poll();

priority = priority - 1;

pcb.setPriority(priority);

pcb.setLife(life);

pcb.setStatus("Ready");

LinkedList<PCB> nextQueue = PCBsQueues[priority].getQueue();

nextQueue.offer(pcb);

PCBsQueues[priority].setQueue(nextQueue);

}

break;

}

}

if(!isEmptyFlag){

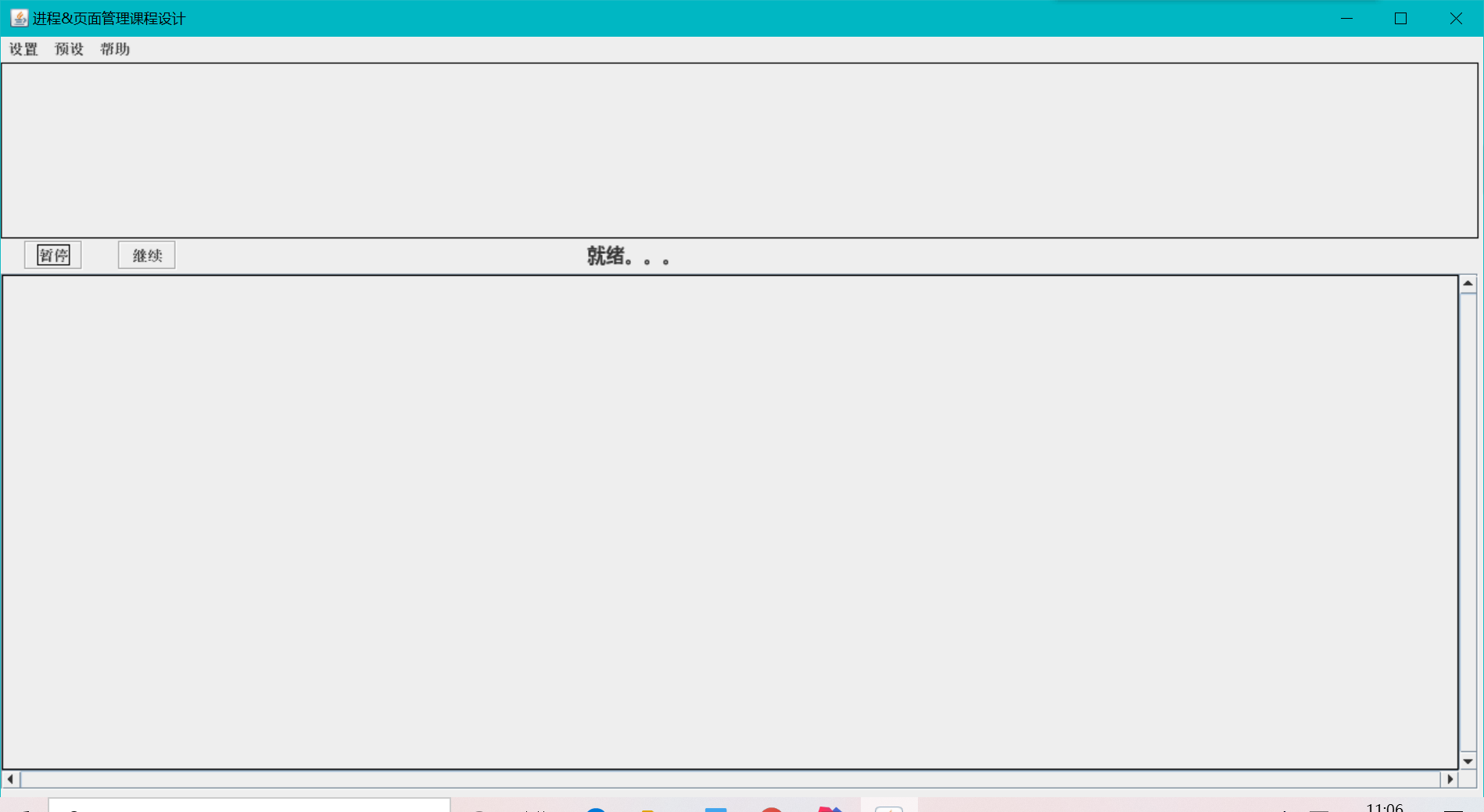
currentTime++;

}

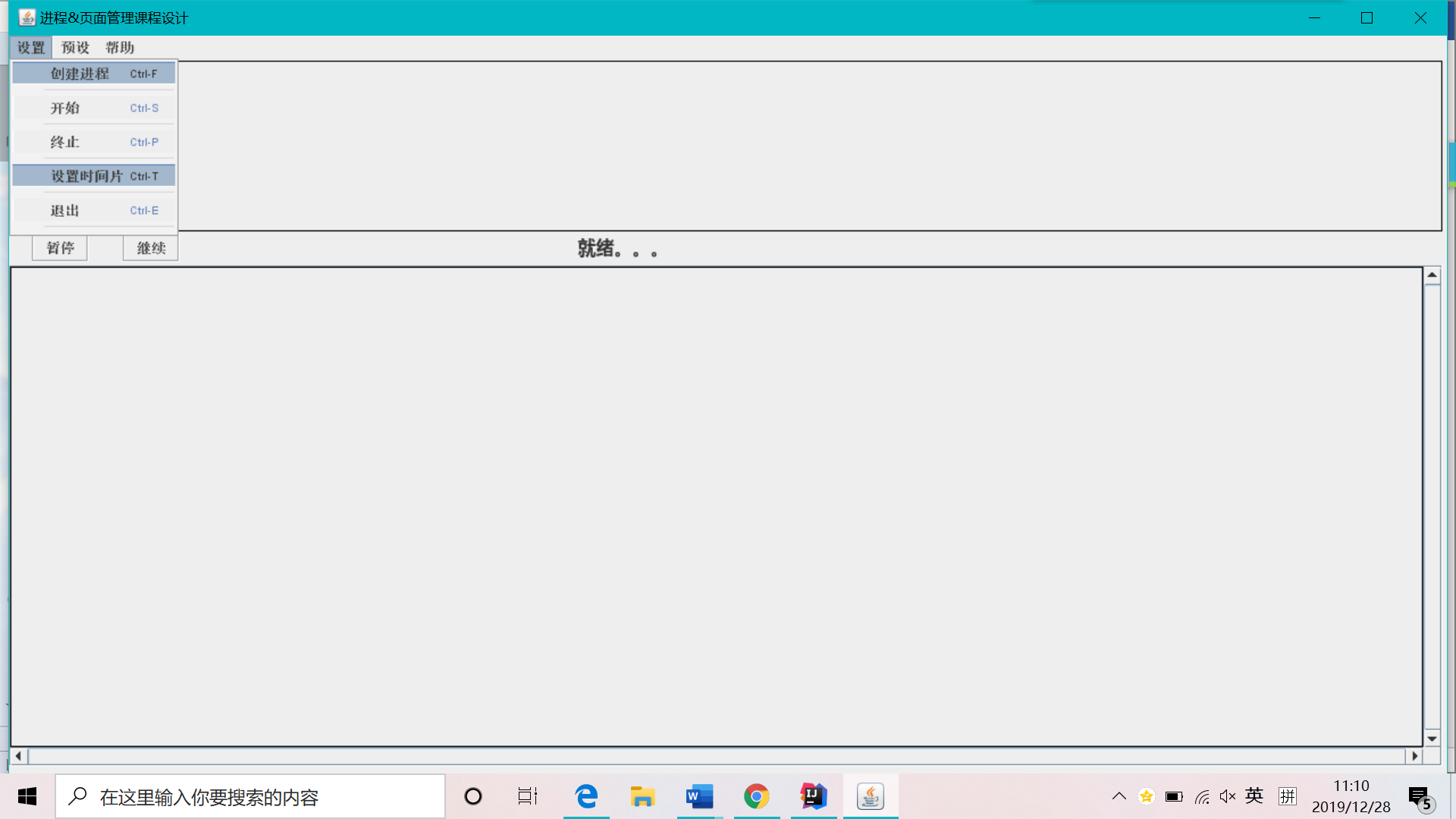
}

1. 系统演示

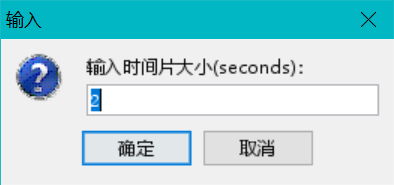
1、程序主界面：



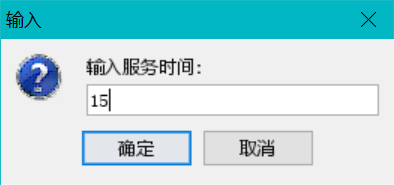
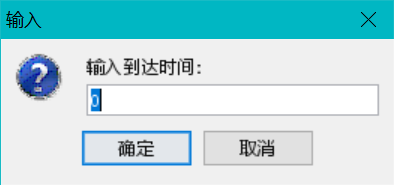
2、主要功能有创建新进程，设置时间片，开始执行和中断执行（并非暂停）。



3、首先设置时间片，若不设置默认为2。演示使用默认值2。



4、创建新进程。依此输入每个进程的到达时间和服务时间。共创建5个进程。



5、进程按照到达顺序在调用序列队列排列好。在执行过程中仍可以以合法输入（到达时间不早于当前总执行时间）不断加入新进程到调用序列。

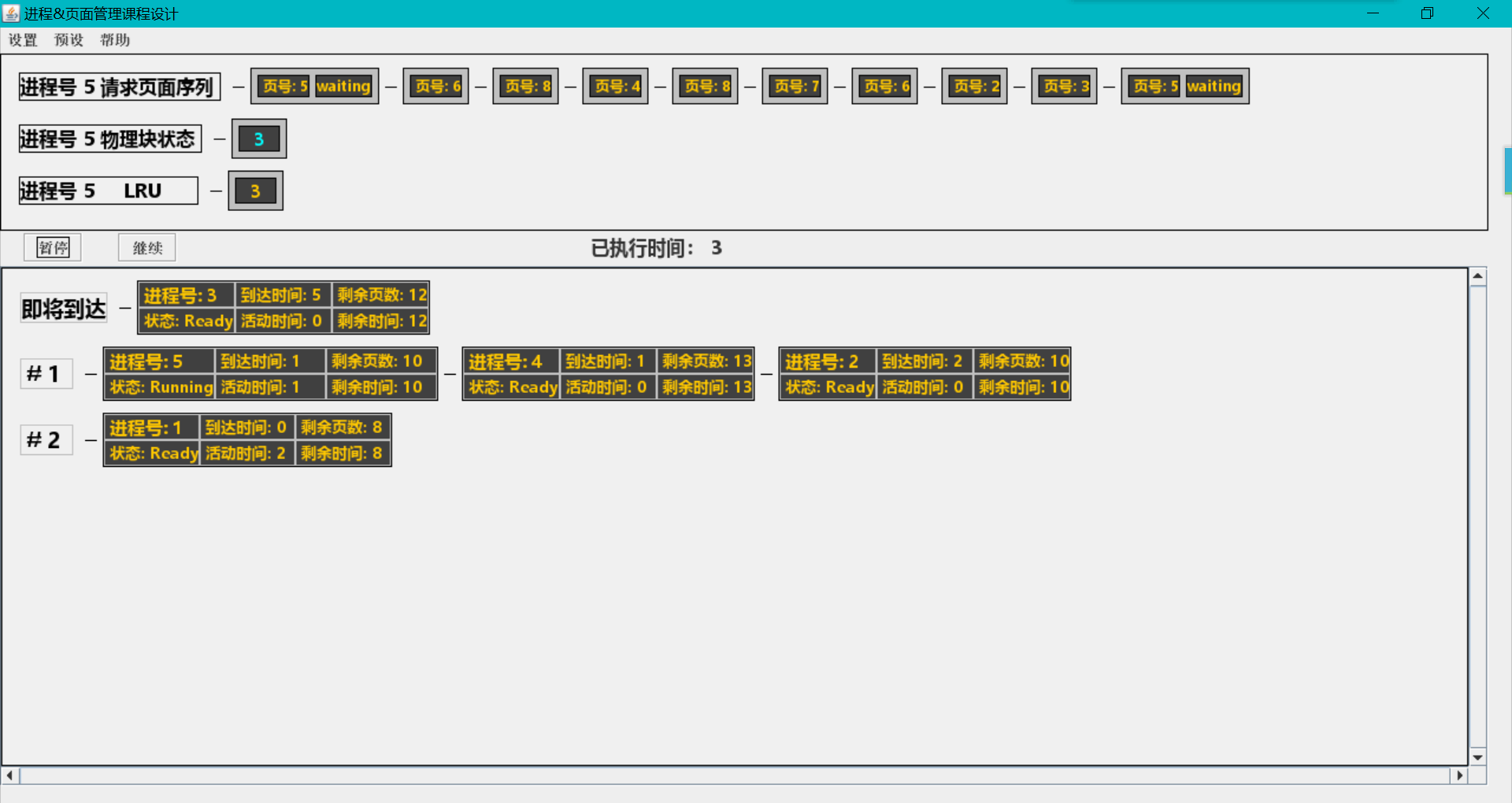


6、进程按到达时间进入1级进程队列。演示以5号进程为例，跟踪其完整的执行过程。

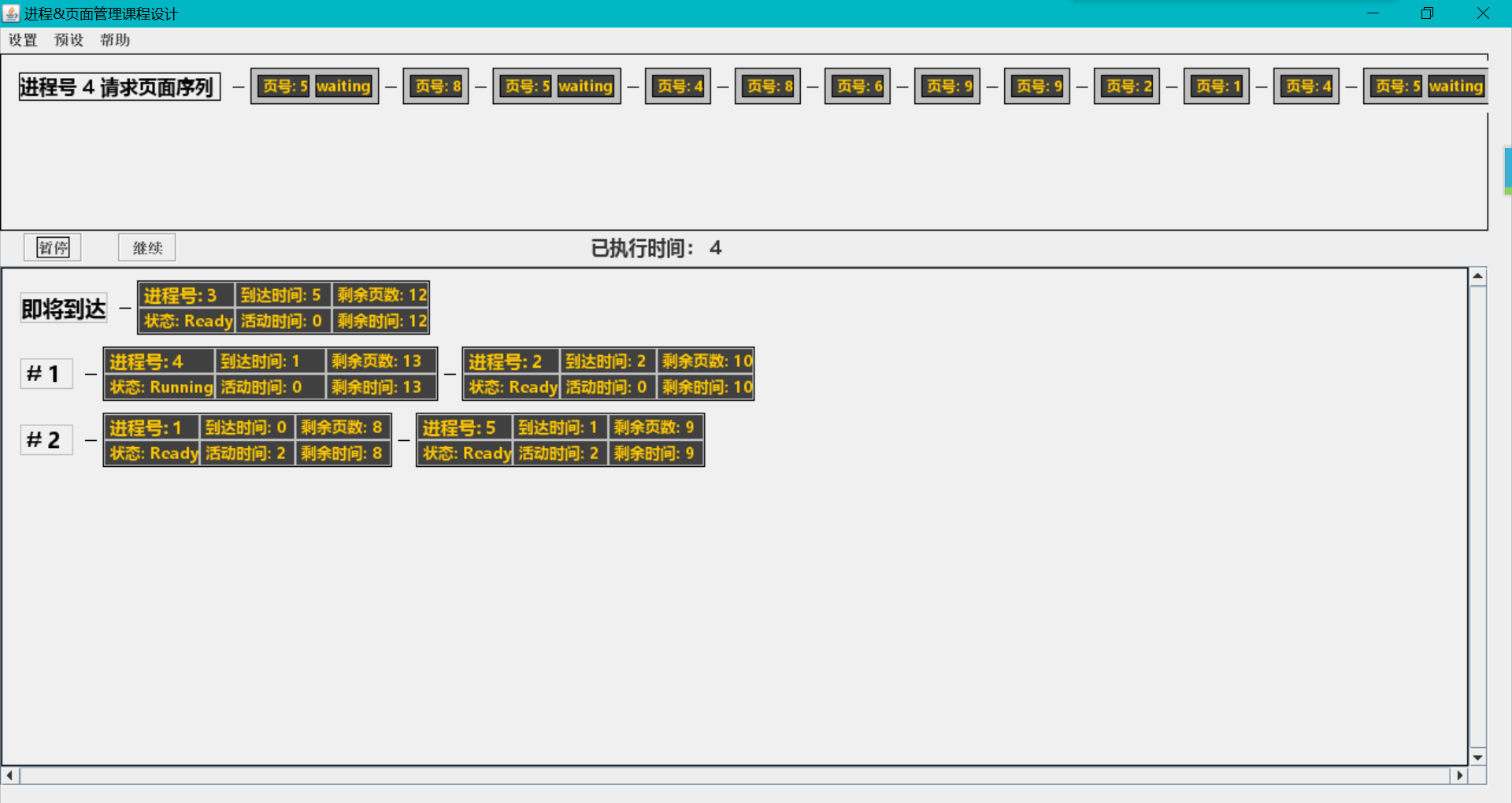
6.1、5号进程到达时间为1，时间1时进入1级队列队尾。另外，4号进程的到达时间也是1且服务时间更长，所以也同时进入队列并排在5号后面。



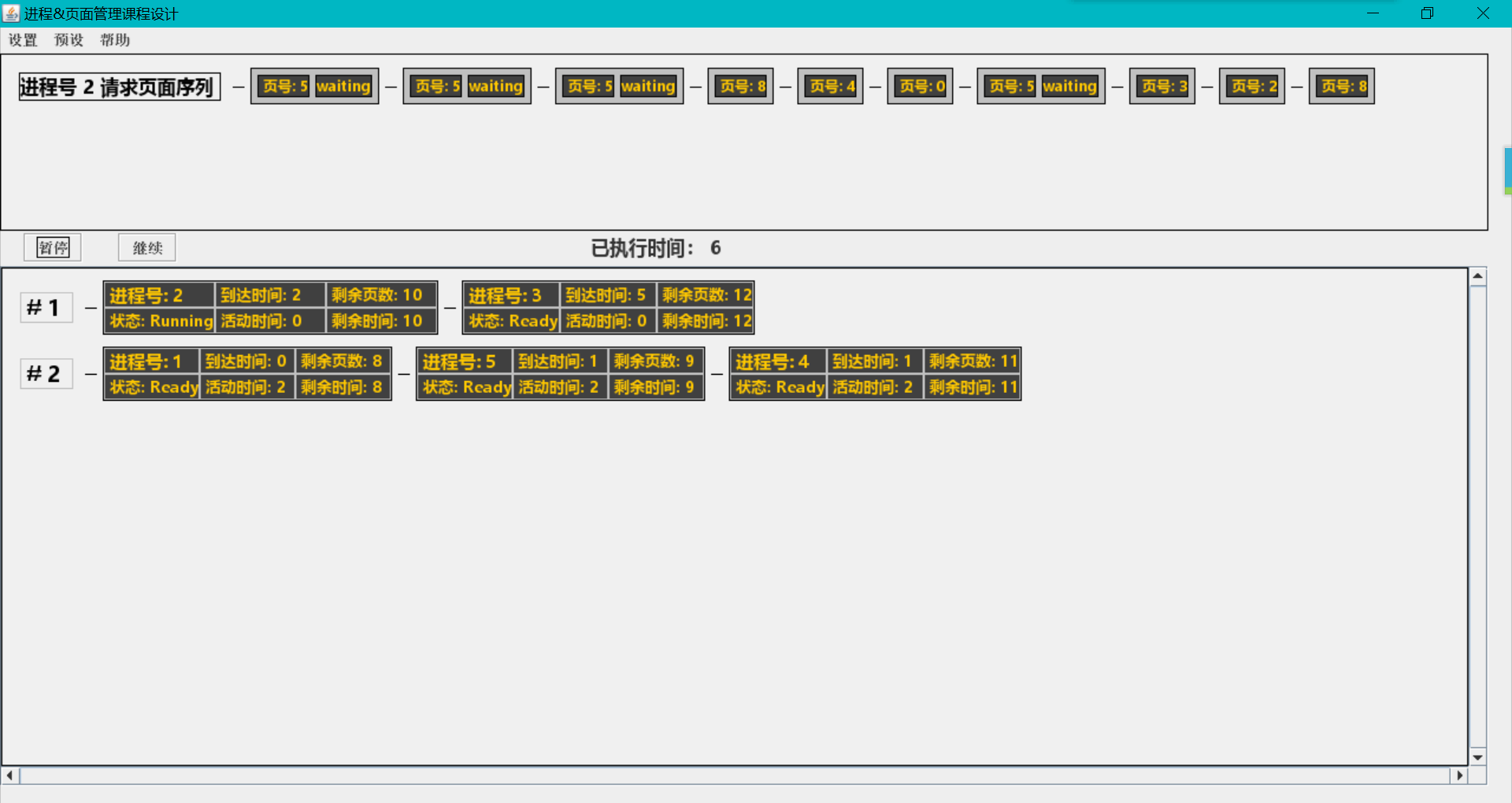
6.2、5号进程前的进程全部完成一个时间片并进入2级队列，此时5号进程开始执行。请求的第一个页面为3，置入物理块，并执行LRU算法。



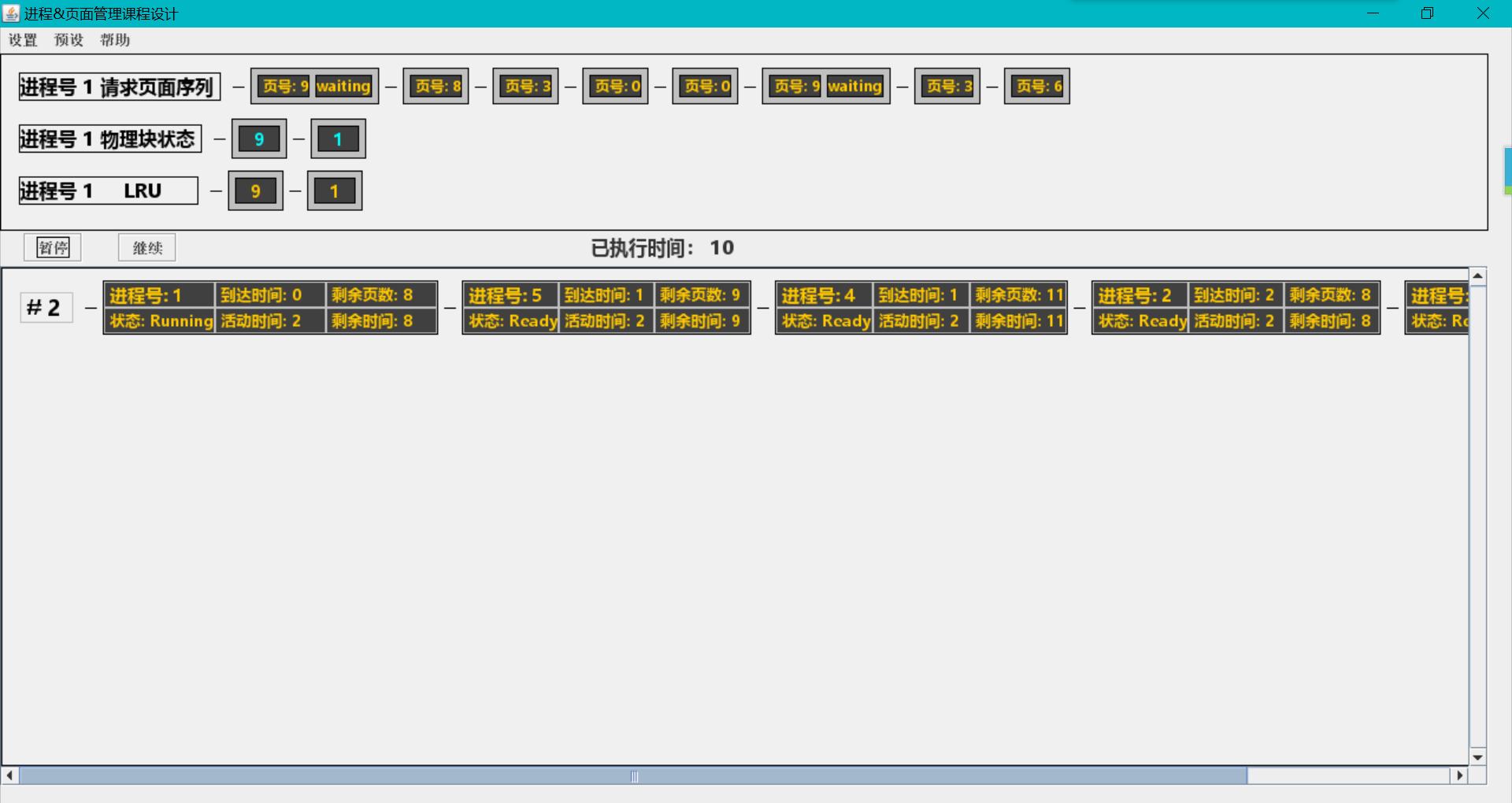
6.3、5号进程执行完一个时间片后进入2级队列的队尾。



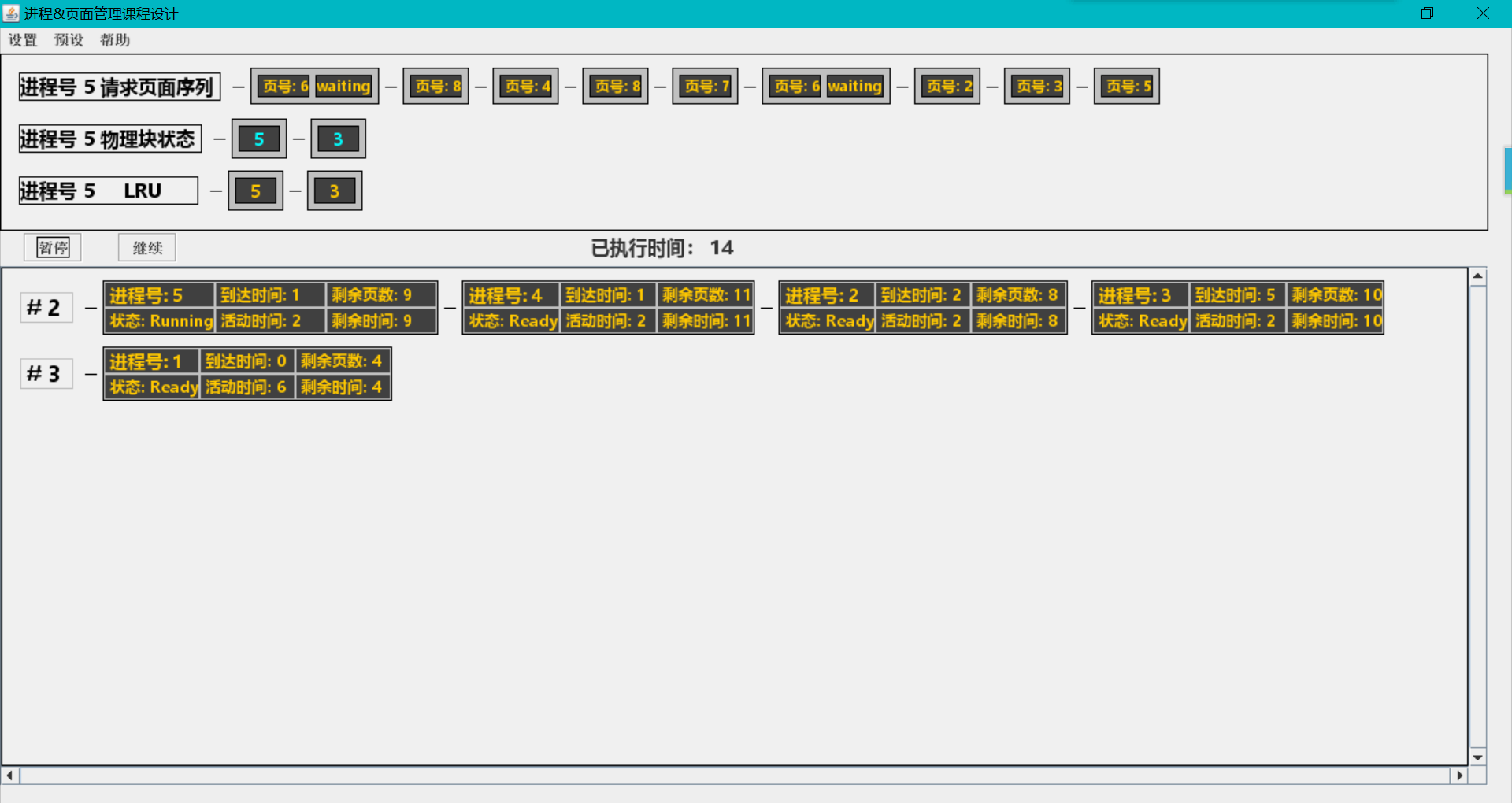
6.4、此时所有进程都已经进入多级反馈队列。

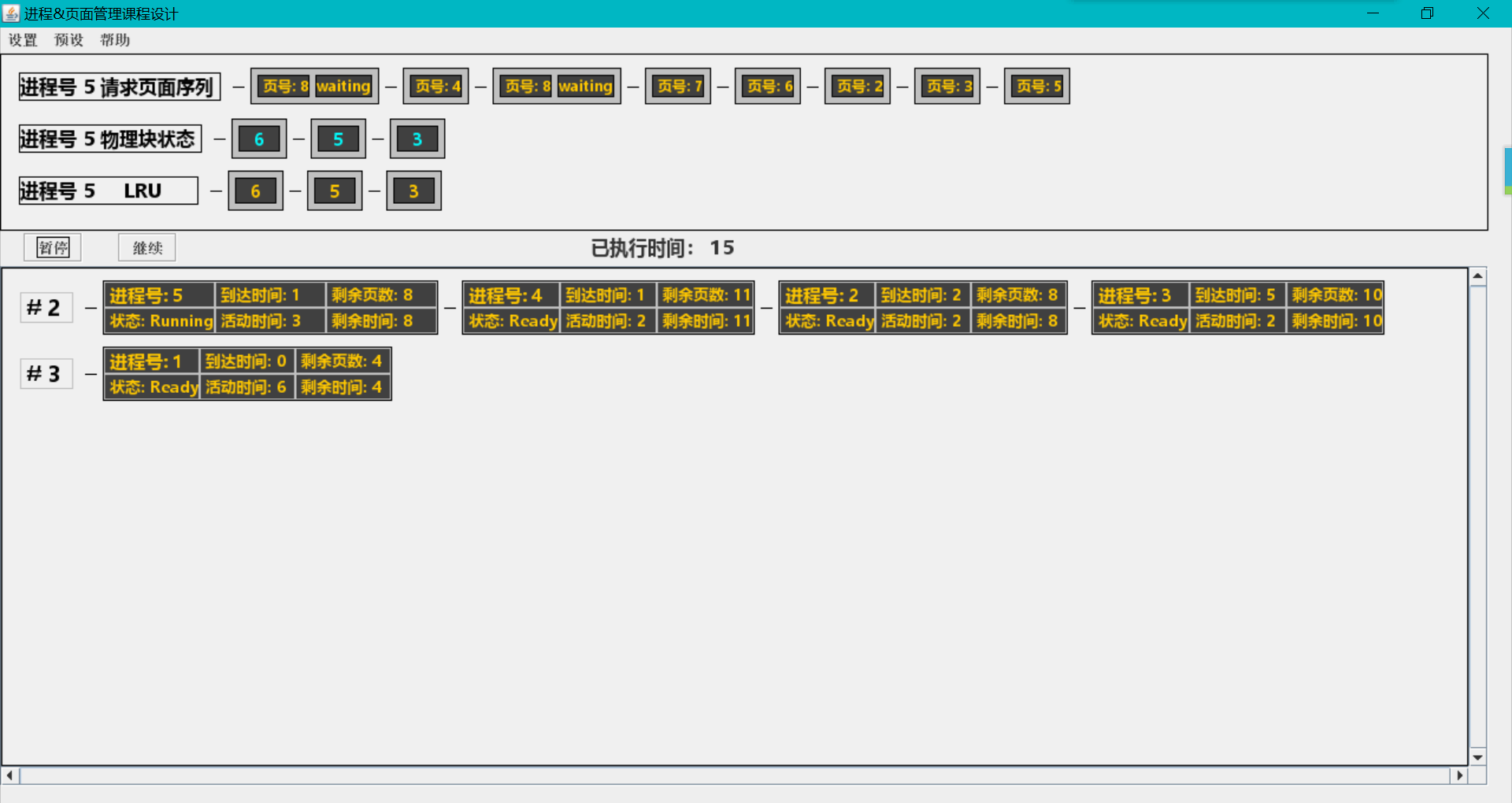


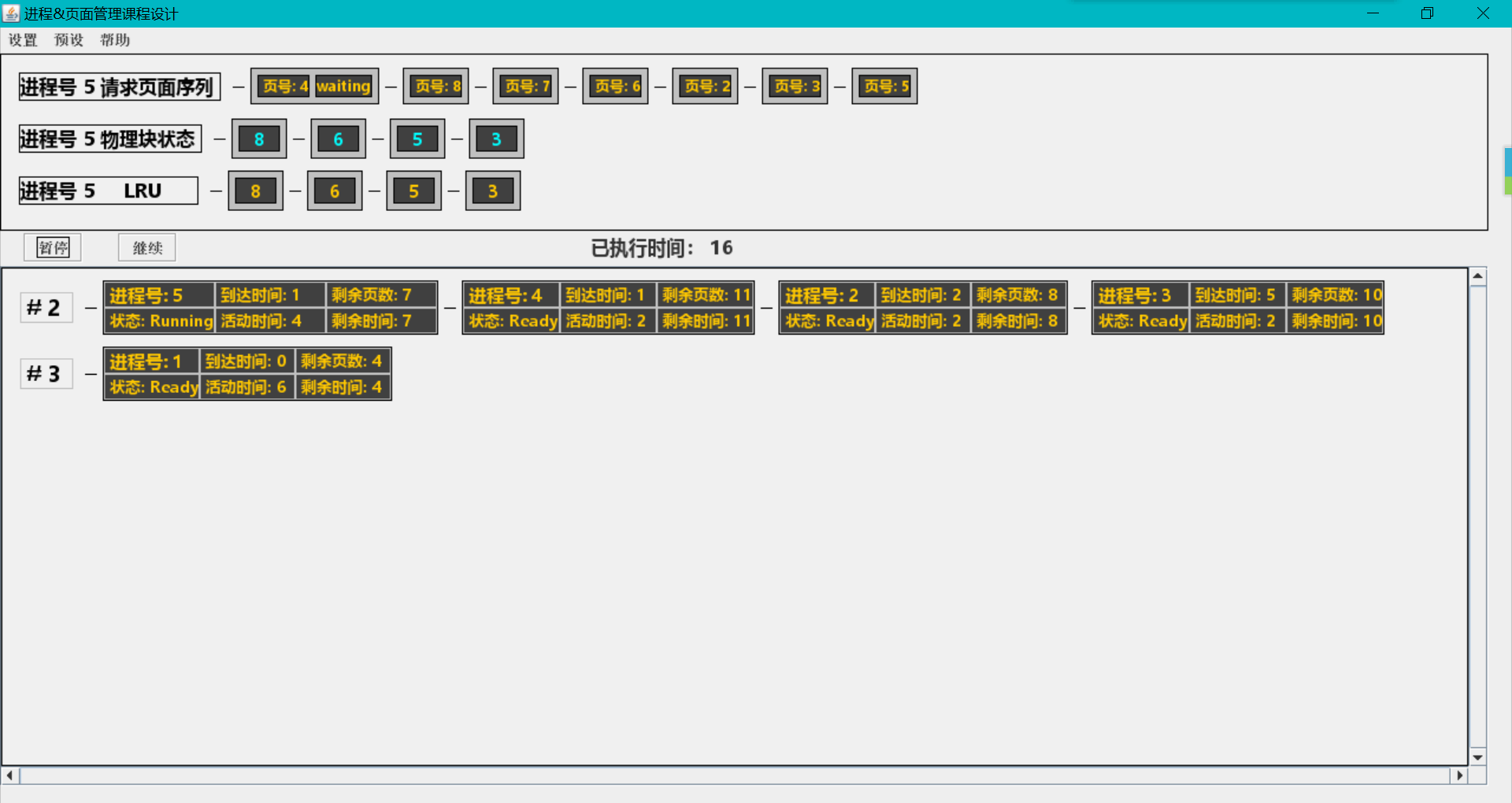
6.5、1级队列的所有进程都执行完成，开始执行2级队列的进程。

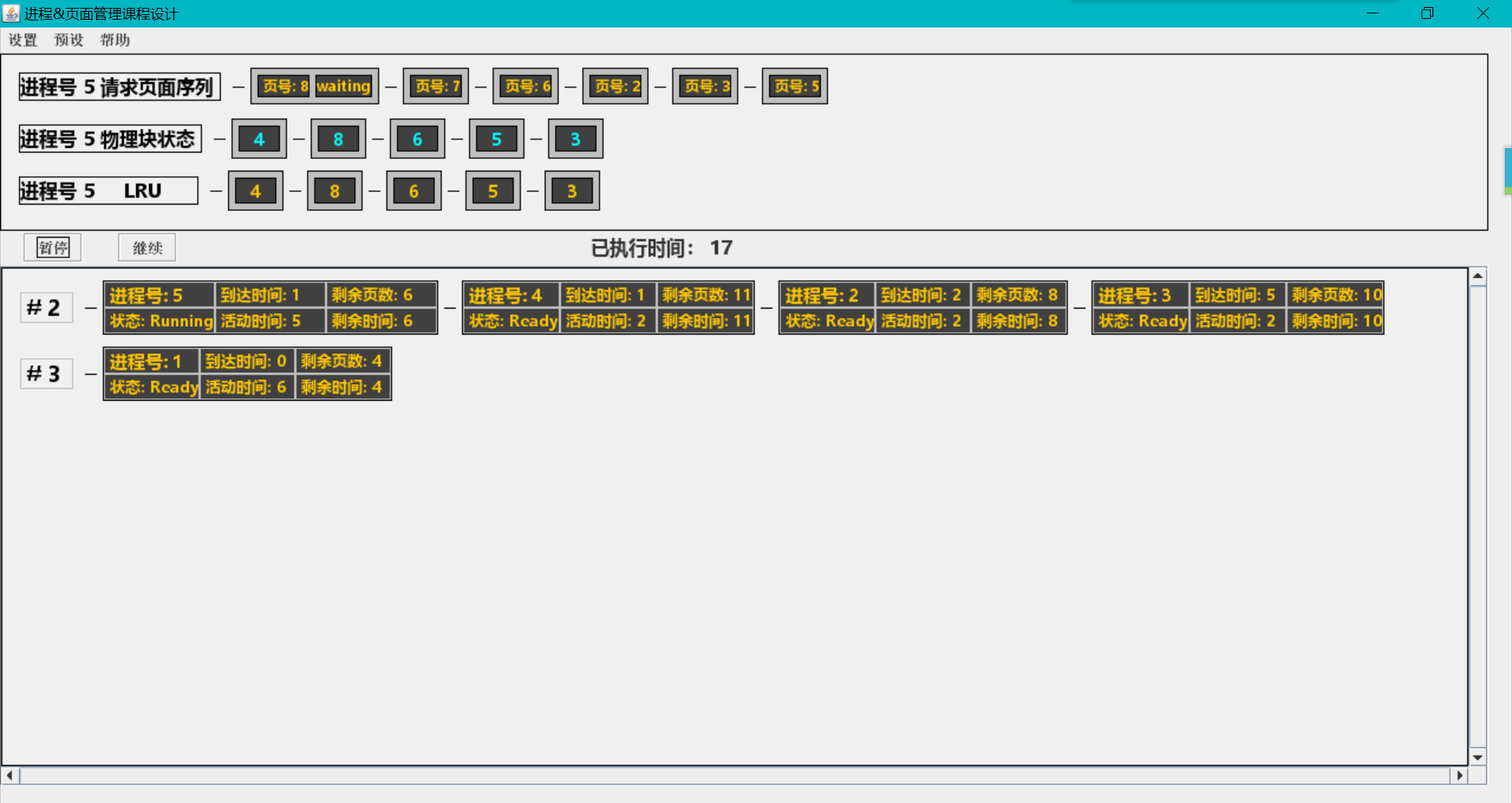


6.6、轮到5号进程，继续执行并持续请求页面，由于物理块限制分配5个，在请求第6个页面时才可能产生中断和置换。

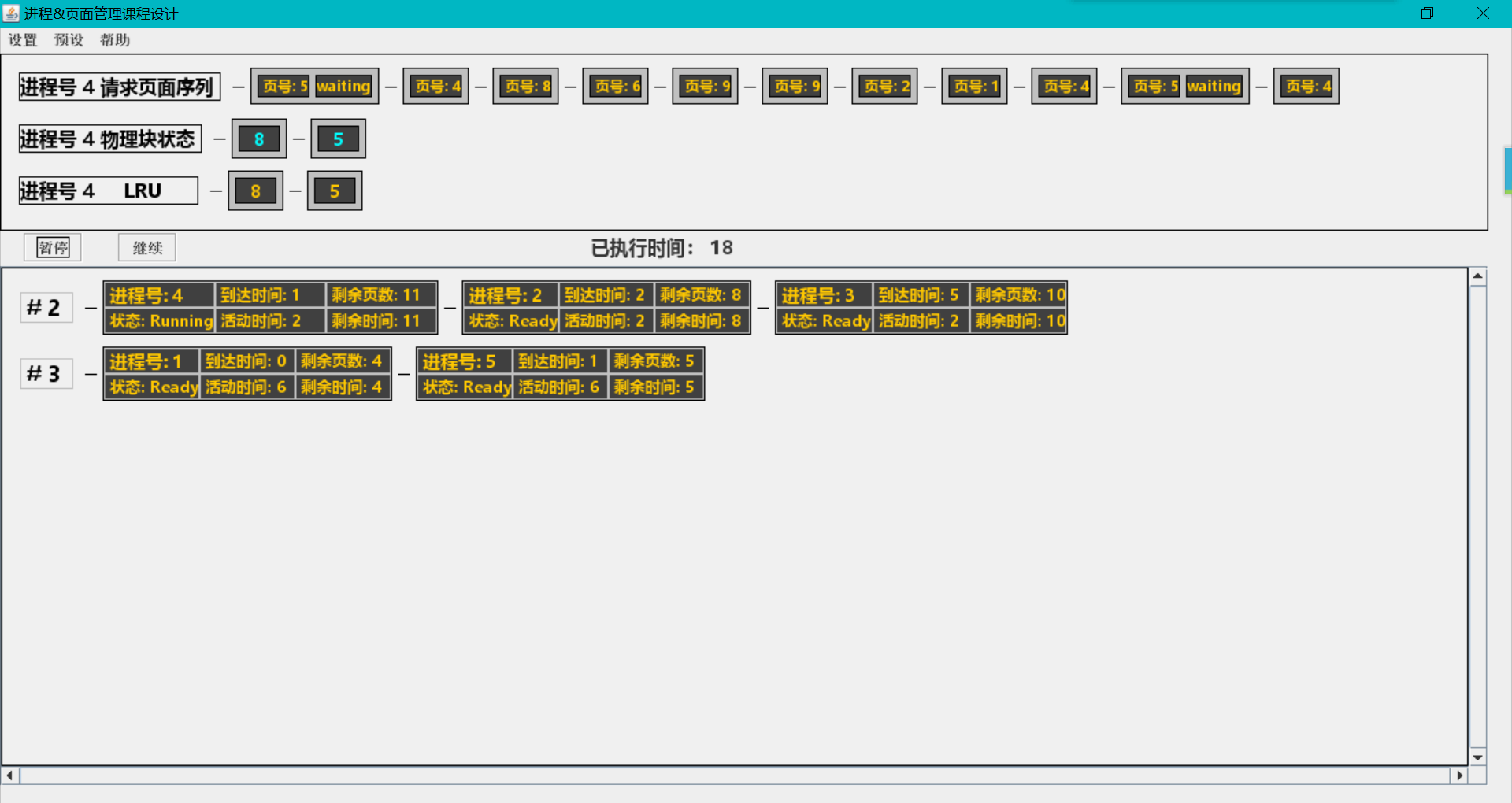




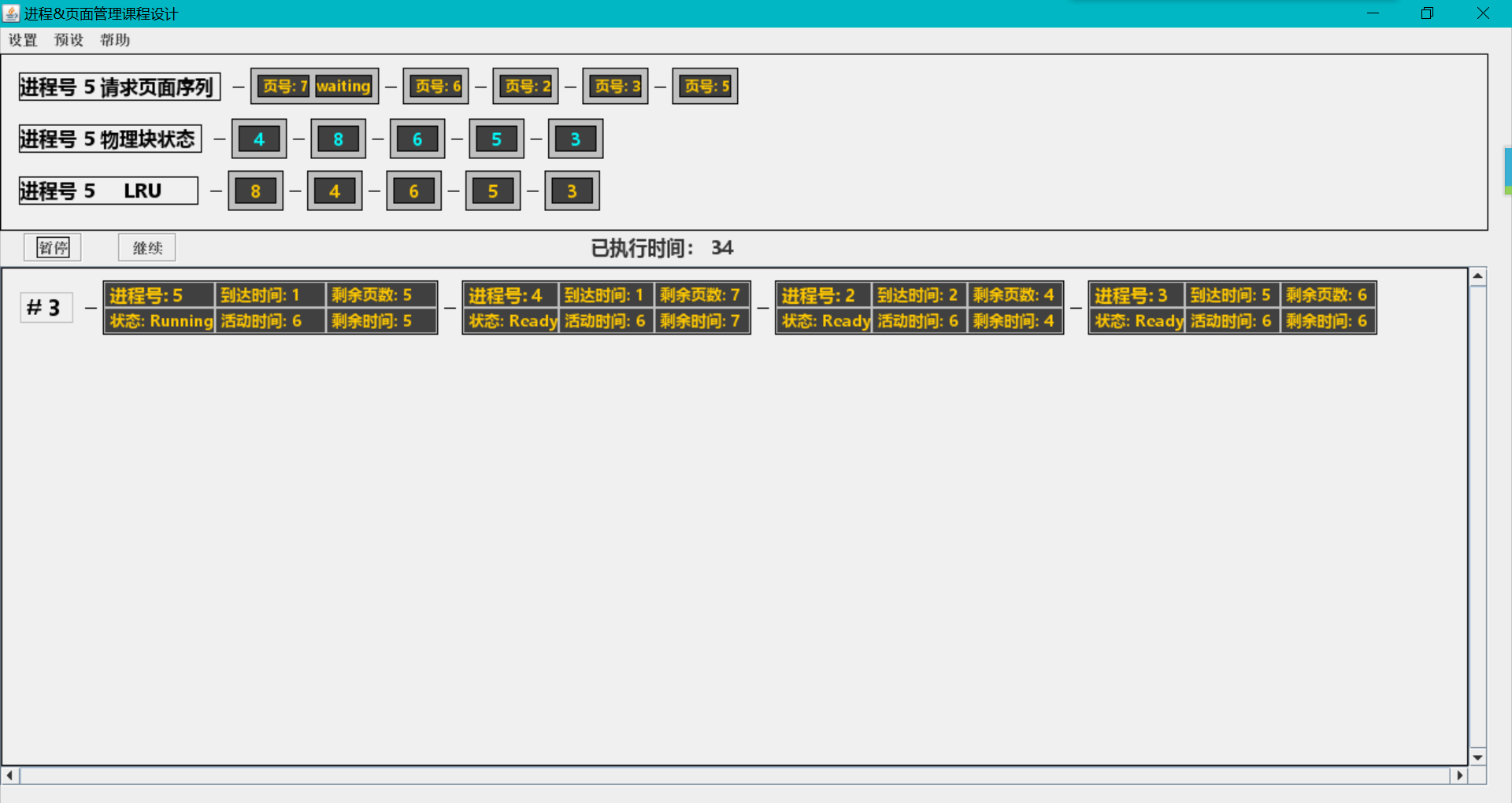


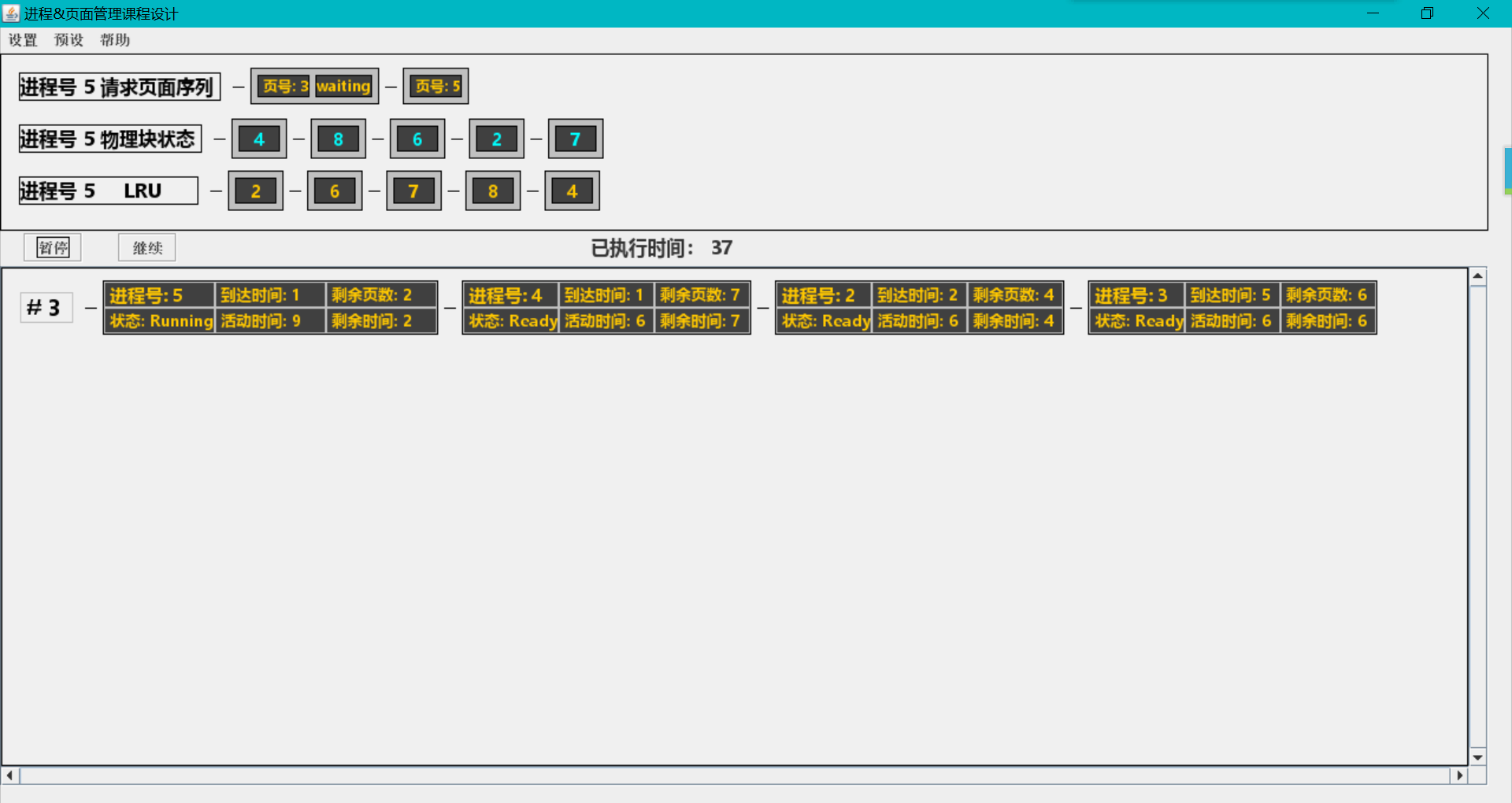


6.7、进入3级队列。

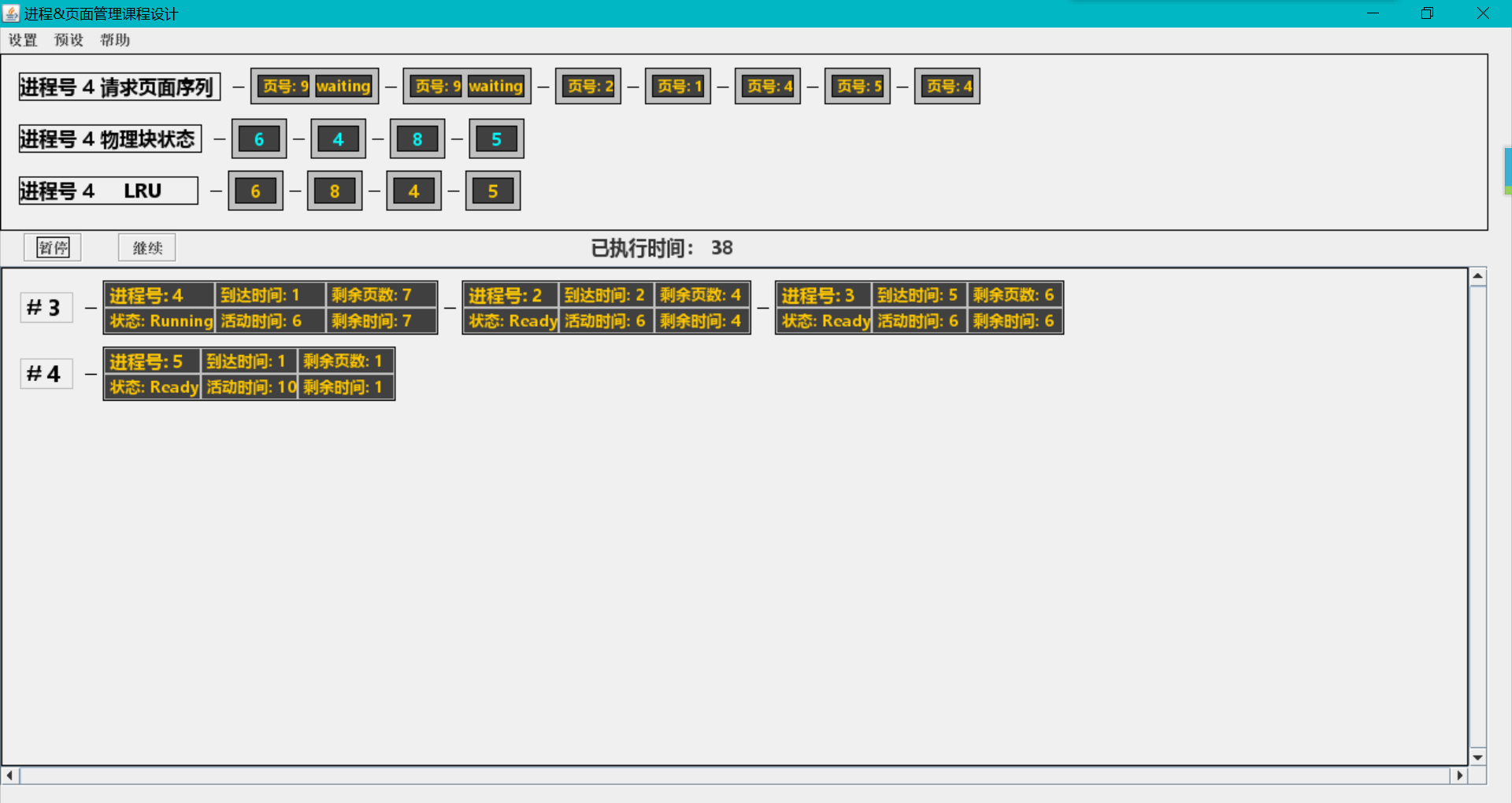


6.8、再次轮到5号进程，此时物理块已满，将产生中断和置换。





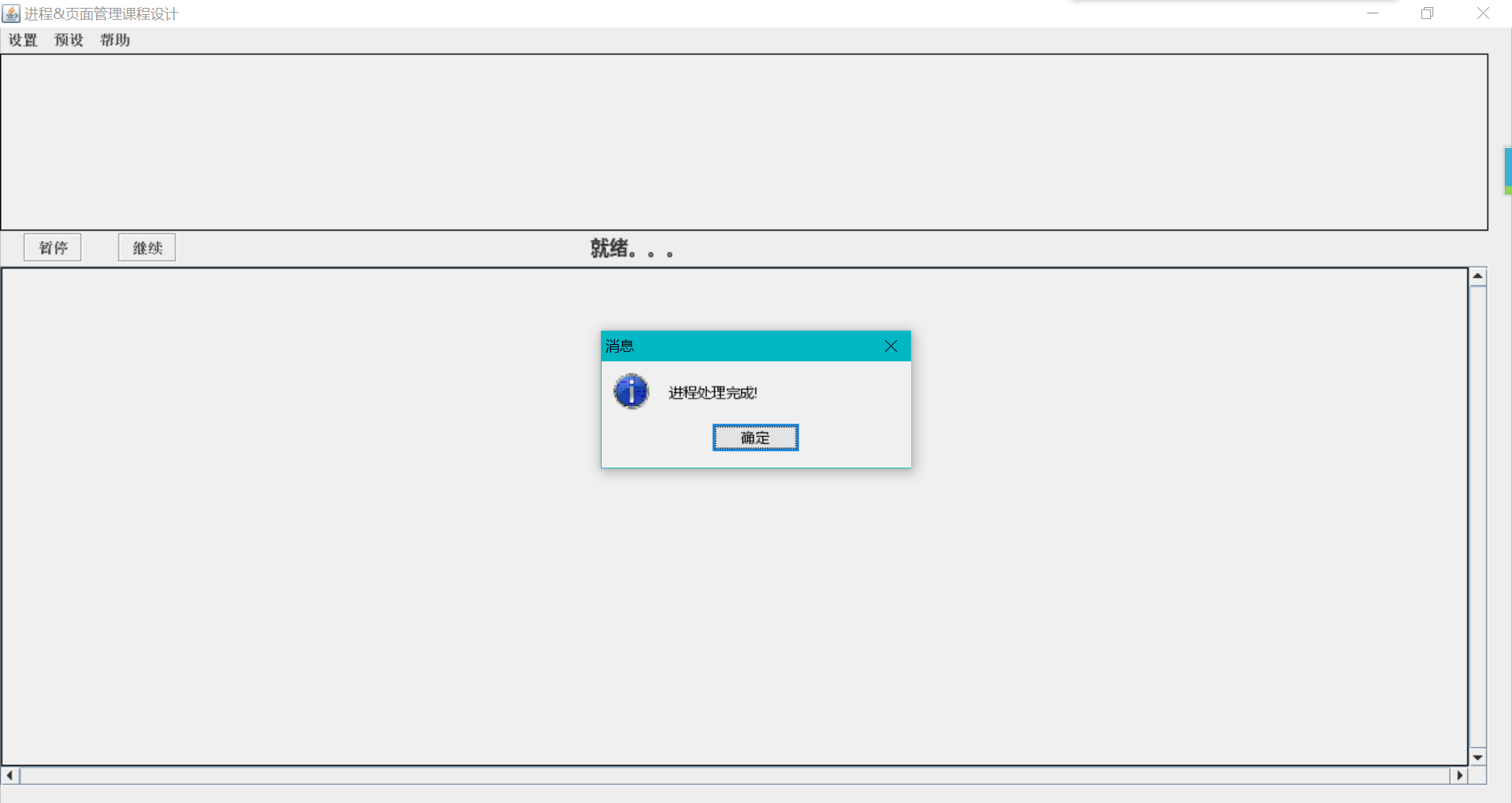
6.9、进入4级队列。

6.10、5号进程完全执行。





7、所有进程完成执行。



1. 总结

通过本次课程设计，我熟悉了多种进程调度算法，其中对多级反馈队列算法有了更加深入的理解；熟悉了多种请求分页的页面置换算法，对LRU算法进行了深入的理解，对进程的数据结构有了更加清晰的认识。在计算机中，操作系统是其最基本也是最为重要的基础性系统软件。从计算机用户的角度来说，计算机操作系统体现在其提供的各项服务；从程序员的角度来说，其主要是指用户登录的界面或者接口；如果从设计人员的角度来说，就是指各式各样模块和单元之间的联系。事实上，全新操作系统的设计和改良的关键工作就是对体系结构的设计，经过几十年以来的发展，计算机操作系统已经由一开始的简单控制循环体发展成为较为复杂的分布式操作系统，再加上计算机用户需求的愈发多样化，计算机操作系统已经成为既复杂而又庞大的计算机软件系统之一。操作系统是计算机系统的灵魂，它管理计算机系统的资源，提供友善的人机互动，对于每一位计算机用户，尤其是计算机专业的学生，认知和理解操作系统非常重要。

**参考文献**

[1] 汤小丹、 梁红兵、 哲凤屏 、汤子瀛等.《计算机操作系统（第四版）》. 西安电子科技大学出版社，2014.5.

[2] 严蔚敏.《数据结构(C语言版)》.清华大学出版社，2007.3.

[3] [美] Cay S.Horstmann, Gary Cornell.《Java核心技术》. 机械工业出版社，2016.9.

[4] [美] Jon Kleinberg / Éva Tardos.《算法设计》.清华大学出版社，2007.