中南大學
CENTRAL SOUTH UNIVERSITY

操作系统原理
实验报告

学生姓名	杨志轩
学　　号	8208180813
专业班级	计科 1804
指导教师	李玺
学　　院	计算机学院
完成时间	2020.6.20

目录

一．实验概述 ……3

(一)实验目的……2

(二)实验内容及要求…… 2

二．需求分析.....4

三、总体设计

(一)总体设计思路工作流程图……6

四、详细设计与实现

(一)数据结构设计与实现……7

(二)模块设计和实现……14

五、实验运行结果…… 15

一． 实验概述

(一). 实验目的

多道系统中，进程与进程之间存在同步与互斥关系。当就绪进程数大于处理机数时，需按照某种策略决定哪些进程先占用处理机。在可变分区管理方式下，采用首次适应算法实现主存空间的分配和回收。 本实验模拟实现处理机调度及内存分配及回收机制，以对处理机调度的工作原理以及内存管理的工作过程进行更深入的了解。

(二).实验内容及要求

1. 实验内容

（1）选择一个调度算法，实现处理机调度；

（2）结合（1）实现主存储器空间的分配和回收。

2. 实验具体要求 4

（1）设计一个抢占式优先权调度算法实现多处理机调度的程序，并且实现在可变分区管理方式下，采用首次适应算法实现主存空间的分配和回收。

（2）PCB 内容包括：进程名/PID；要求运行时间（单位时间）；优先权；状态；进程属性：独立进程、同步进程（前趋、后继）。

（3）可以随机输入若干进程，可随时添加进程，并按优先权排序；

（4）从就绪队首选进程运行：优先权-1；要求运行时间-1；要求运行时间为 0 时，撤销该进程；一个时间片结束后重新排序，进行下轮调度；

（5）考虑两个处理机，考虑同步进程的处理机分配问题，每次调度后，显示各进程状态，运行进程要显示在哪个处理机上执行。

（6）规定道数，设置后备队列和挂起状态。若内存中进程少于规定道数，可自动从后备队列调度一作业进入。被挂起进程入挂起队列，设置解挂功能用于将制定挂起进程解挂入就绪队列。

（7）结合实验一 PCB 增加所需主存大小，主存起始位置；采用首次适应算法分配主存空间。

（8）自行假设主存空间大小，预设操作系统所占大小并构造未分分区表。表目内容：起址、长度、状态（未分/空表目）。

（9）进程完成后，回收主存，并与相邻空闲分区合并。

（10）最好采用图形界面；

二．需求分析

设计一个抢占式优先权调度算法实现多处理机调度的程序，并且实现在可变分区管理方式下，采用首次适应算法实现主存空间的分配和回收。 PCB 内容包括：进程名/PID；要求运行时间（单位时间）；优先权；状态；进程属性：独立进程、同步进程（前趋、后继）。 可以随机输入若干进程，可随时添加进程，并按优先权排序； 从就绪队首选进程运行：优先权-1；要求运行时间-1；要求运行时间为 0 时，撤销该进程；一个时间片结束后重新排序，进行下轮调度；
考虑两个处理机，考虑同步进程的处理机分配问题，每次调度后，显示各进程状态，运行进程要显示在哪个处理机上执行。 规定道数，设置后备队列和挂起状态。若内存中进程少于规定道数，可自动从后备队列调度一作业进入。被挂起进程入挂起队列，设置解挂功能用于将制定挂起进程解挂入就绪队列。结合实验一 PCB 增加所需主存大小，主存起始位置；采用首次适应算法分配主存空间。 自行假设主存空间大小，预设操作系统所占大小并构造未分分区表。表目内容：起址、长度、状态（未分/空表目）。 进程完成后，回收主存，并与相邻空闲分区合并。最好采用图形界面；

创建 pcb 类和内存类，pcb 派生出后备队列，就绪队列和挂起队列。这是基本的数据结构。

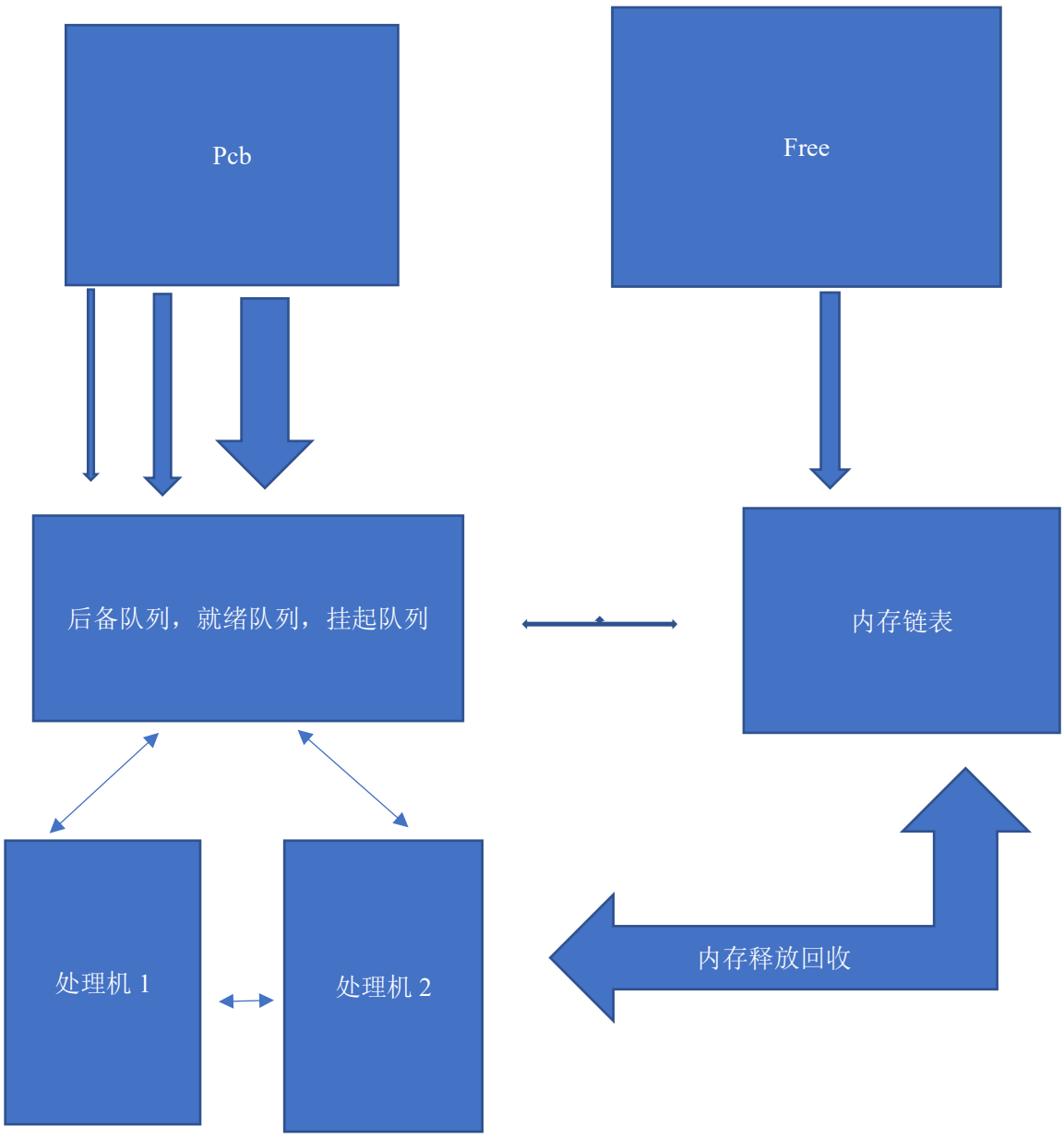
实验用到的算法，内存分配首次适应算法，内存回收算法，队列的优先级排序算法和队列转移算法。

实验核心围绕着两大处理机运行，运行前检测就绪队列优先级排序，运行后也要检测优先级排序，以实现动态优先级，时间片以人工操作为准，一次操作代表一个时间片，

处理机的设计最为复杂，我们选定一次操作为一个时间片，需要判断，内存道数是否已满，和后备队列是否为空，如果满足条件，需要从就绪队列取出第一个元素，时间和优先级动态规划，第二个需要判断 cpu 中正在执行的进程和就绪队列进程的优先级，如果 cpu 优先级低就需要挂起队列，本实验设计的挂起只能手动解卦，否则就执行 cpu 中的进程。第三个判断 cpu 中的进程是否执行完毕

三. 总体设计

(一).总体设计思路工作流程图



四．详细设计与实现

(一). 数据结构设计与实现
(二). 模块设计与实现

数据结构

```
public class PCB {

    int id;// 进程号
    int time;// 运行时间
    int priority; // 优先级。 数字越小，优先级越高
    int cometime; // 到达时间
    int length; // 进程大小

    public PCB(int id, int cometime, int time, int priority) {
        super();
        this.id = id;
        this.time = time;
        this.priority = priority;
        this.cometime = cometime;
    }

    public PCB(int id, int time, int priority, int cometime, int length) {
        super();
        this.id = id;
        this.time = time;
        this.priority = priority;
        this.cometime = cometime;
        this.length = length;
    }

    public PCB() {
        super();
    }

    @Override
    public String toString() {
        return id + "_" + time + "_" + priority + "_" + cometime + "_" + length;
    }

}
```

内存回收算法

```
public static void release(PCB pcb) {
        Collections.sort(LinkedFree, new SortByStart());
        int id = pcb.id;

        Iterator<Free> iterator = LinkedFree.iterator();// 迭代器
        int index = 0; // 要移除的分区号 下标
        int index2 = -1; // 要合并的分区号

        while (iterator.hasNext()) {
            Free p = iterator.next();
            index = LinkedFree.indexOf(p);
            if (p.id == id) { // 找到此分区
                Free free1 = null, free3 = null;
                if (index - 1 >= 0) { // 前面有分区
                    if (LinkedFree.get(index - 1).status == 0) { // 前 是空闲
分区
                        free1 = LinkedFree.get(index - 1);
                    }
                }
                if (index + 1 < LinkedFree.size()) { // 后面有分区
                    if (LinkedFree.get(index + 1).status == 0) { // 后 是空闲
分区
                        free3 = LinkedFree.get(index + 1);
                    }
                }

                if (free1 != null && free3 != null) { // 合并前后分区
                    free1.capacity += p.capacity + free3.capacity;
                    index2 = index;
                    iterator.remove();

                } else if (free1 != null && free3 == null) { // 合并前面分区
                    free1.capacity += p.capacity;
                    iterator.remove();
                } else if (free1 == null && free3 != null) { // 合并后面分区
                    free3.capacity += p.capacity;
                    free3.start -= p.capacity;
                    iterator.remove();
                } else { // 不需要合并
                    p.status = 0; // 忙碌 变为空闲

                }
            }
        }
```

```
        if (index2 != -1) {
            LinkedFree.remove(index2); // 删除合并的内存块信息
        }

    }
```

首次适应算法

```
public static void allocation(PCB pcb) {

        int applySize = pcb.length;
        Free temp = null; // 要分配的分区
        Collections.sort(LinkedFree, new SortByStart()); // 根据起始地址排序

        for (int i = 0; i < LinkedFree.size(); i++) {
            Free point = LinkedFree.get(i);
            if (point.capacity >= applySize && point.status == 0) { // 内存
充裕
                temp = new Free(pcb.id, point.start, applySize);
                temp.status = 1;
                point.capacity -= applySize;
                point.start += applySize;
                LinkedFree.add(temp);
                if (point.capacity == 0) { // 空间全部分配
                    LinkedFree.remove(point);
                }
                break;
            }

        }
        Collections.sort(LinkedFree, new SortByStart()); // 根据起始地址排序
    }
```

处理机算法

```
JButton btnNewButton_1 = new JButton("Step");
        btnNewButton_1.addMouseListener(new MouseAdapter() {

            public void mouseClicked(MouseEvent e) {
                try {

                    lblNewLabel_3.setText("t1=" + t1);
```

```
                    label_6.setText("到达时间:" + t);
                    if (i < daoshu && houzhi.size() != 0) {
                        Collections.sort(houzhi, new SortByPriority()); //
后置队列按优先级排序
                        Move(houzhi.get(0), houzhi, jiuxu);
                        i++;
                    }
                    if(!jiuxu.isEmpty())
                    {Collections.sort(jiuxu, new SortByPriority());} // 就绪
队列按优先级排序

                    if (!jiuxu.isEmpty()&&cpu1 == false) {// cpu1上没有进程
                        PCB pcb1 = jiuxu.get(0);
                        jiuxu.remove(0);

                        cpuPcb1 = pcb1;
                        if (cpuPcb1.time == 0) {
                            textArea_4.append("cpu1上的进程" + cpuPcb1.id + "
已完成。    t=" + t + "\n");
                            textField_9.setText("");

                            release(cpuPcb1);
                            i--;
                            cpu1 = false;

                        }
                        cpuPcb1.time -= 1;
                        cpuPcb1.priority+=1;
                        cpu1 = true;
                        textArea_4.append("进程" + cpuPcb1.id + "进入CPU1执
行。    t=" + t + "\n");
                        textField_9.setText("进程" + cpuPcb1.id + "正在执行
");
                    } else if ( !jiuxu.isEmpty()&&cpuPcb1.priority >
jiuxu.get(0).priority ) // cpu1上有进程，进行优先级比较
                    {
                        Move1(cpuPcb1, guaqi);  //将进程置入挂起队列

                        textArea_4.append("进程" + cpuPcb1.id + "被进程" +
jiuxu.get(0).id + "抢占，进程" + jiuxu.get(0).id
                                + "进入CPU1执行。    t=" + t + "\n");
                        release(cpuPcb1);
                        cpuPcb1 = jiuxu.get(0);
                        jiuxu.remove(0);
```

```
                    textField_9.setText("进程" + cpuPcb1.id + "正在执行
");
                    cpuPcb1.time -= 1;
                    cpuPcb1.priority+=1;

                } else if(!jiuxu.isEmpty()&&cpuPcb1.priority <=
jiuxu.get(0).priority||cpu1==true) {

                    cpuPcb1.time -= 1;
                    cpuPcb1.priority+=1;
                    textArea_4.append("进程" + cpuPcb1.id + "进入CPU1执
行。    t=" + t + "\n");
                    textField_9.setText("进程" + cpuPcb1.id + "正在执行
");

                }
                if(cpuPcb1==null) {
                 textArea_4.append("cpu1为空\n");
                }
                else if (cpuPcb1.time == 0) {
                    textArea_4.append("cpu1上的进程" + cpuPcb1.id + "已完
成。    t=" + t + "\n");
                    textField_9.setText("");

                    release(cpuPcb1);
                    cpuPcb1=null;
                    i--;
                    cpu1 = false;

                }

                //处理机2
                if (!jiuxu.isEmpty()&&cpu2 == false) {// cpu2上没有进程
                    PCB pcb1 = jiuxu.get(0);
                    jiuxu.remove(0);cpuPcb2 = pcb1;
                    if (cpuPcb2.time == 0) {
                        textArea_4.append("cpu2上的进程" + cpuPcb2.id + "
已完成。    t=" + t + "\n");
                        textField_9.setText("");

                        release(cpuPcb2);
                        i--;
                        cpu2 = false;
```

```
                    }
                    cpuPcb2.time -= 1;
                    cpuPcb2.priority+=1;
                    cpu2 = true;
                    textArea_4.append("进程" + cpuPcb2.id + "进入CPU2执
行。    t=" + t + "\n");
                    textField_9.setText("进程" + cpuPcb2.id + "正在执行
");
                } else if (!jiuxu.isEmpty()&&cpuPcb2.priority >
jiuxu.get(0).priority ) // cpu2上有进程，进行优先级比较
                {
                    Move1(cpuPcb2,  guaqi);  //将进程由就绪队列置入挂起队列

                    textArea_4.append("进程" + cpuPcb2.id + "被进程" +
jiuxu.get(0).id + "抢占，进程" + jiuxu.get(0).id
                            + "进入CPU2执行。    t=" + t + "\n");
                    release(cpuPcb2);
                    cpuPcb2 = jiuxu.get(0);
                    jiuxu.remove(0);
                    textField_9.setText("进程" + cpuPcb2.id + "正在执行
");
                    cpuPcb2.time -= 1;
                    cpuPcb2.priority+=1;

                } else if(!jiuxu.isEmpty()&&cpuPcb2.priority <=
jiuxu.get(0).priority||cpu2==true){
                    cpuPcb2.time -= 1;
                    cpuPcb2.priority+=1;
                    textArea_4.append("进程" + cpuPcb2.id + "进入CPU2执
行。    t=" + t + "\n");
                    textField_9.setText("进程" + cpuPcb2.id + "正在执行
");

                }



                 if(cpuPcb2==null) {
                 textArea_4.append("cpu2为空\n");
                }
                 else if (cpuPcb2.time == 0) {
                    textArea_4.append("CPU2上的进程" + cpuPcb2.id + "已完
成。    t=" + t + "\n");
                    textField_9.setText("");
```

```
                    release(cpuPcb2);
                    cpuPcb2=null;
                    i--;
                    cpu2 = false;

                }

                t++;

                System.out.println(jiuxu.toString());
                textArea_3.setText("");
                textArea_2.setText("");
                textArea_1.setText("");
                textArea.setText("");

                for (PCB pcb : jiuxu)// 打印就绪队列
                    textArea_2.append(pcb.toString() + "\n");
                for (Free free : LinkedFree)// 打印空闲分区表
                    textArea_1.append(free.toString() + "\n");
                for (PCB pcb : guaqi)// 打印挂起队列
                    textArea_3.append(pcb.toString() + "\n");
                for (PCB pcb : houzhi)// 打印后置队列
                    textArea.append(pcb.toString() + "\n");

            }
            catch (Exception e2) {
                lblNewLabel_3.setText("总用时: " + (--t));
                textField_9.setText("");
            }

        }
```

队列优先级排序

```
class SortByPriority implements Comparator<PCB> { // 按优先级

    @Override
    public int compare(PCB p1, PCB p2) {

        if (p1.priority > p2.priority)
            return 1;
        else if (p1.priority == p2.priority && p1.cometime > p2.cometime)
// 若优先级相同，则先来先服务
```

```
            return 1;

        return -1;
    }

}
```

Java内存链表排序

```
class SortByStart implements Comparator<Free> { // 实现起点比较

    @Override
    public int compare(Free f1, Free f2) {

        if (f1.start > f2.start)
            return 1;

        return -1;
    }

}
```

五．实验运行结果


后备队列：
进程名 运行时间 优先级 到达时间 大小
(1)内存大小为1000K，OS占前50K
(2)道数取为3
(3)总时间设为t
空闲分区表：
分区号 分区大小 分区始址 状态
0_950_50_空闲
CPU状态：
就绪队列：
挂起队列：
3_2_3_0_0
PCB信息：
进程号：
3
运行时间：
4
优先级：
1
到达时间:4
大小：
0
挂起
解挂
添加
重置
状态：
进程3被进程2抢占，进程2进入CPU1执行。 t=3
cpu1上的进程2已完成。 t=3
进程1进入CPU2执行。 t=3
CPU2上的进程1已完成。 t=3
cpu1为空
cpu2为空
Step
重置系统
t1=0