**华 南 农 业 大 学 信 息（软 件） 学 院**

《操作系统分析与设计实习》成绩单

开设时间：2021学年第一学期

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 小组成员、组内分工、工作量比例、各成员个人成 | | | | | | | | | | | |
| **学号** | 201925230110 | **姓名** | 赖鸿祥 | **分工** | 文件管理和用户接口 | | | **工作量比例** | 25% | **成绩** |  |
| **学号** | 201925230105 | **姓名** | 陈欣明 | **分工** | 存储管理 | | | **工作量比例** | 25% | **成绩** |  |
| **学号** | 201925230107 | **姓名** | 何广生 | **分工** | 设备管理 | | | **工作量比例** | 25% | **成绩** |  |
| **学号** | 201925230108 | **姓名** | 黄浩靖 | **分工** | 进程管理 | | | **工作量比例** | 25% | **成绩** |  |
| 实 验 题 目 | 模拟操作系统实现 | | | | | | | | | | |
| 自 我 评 价 | 本次实验的完成体现了高度协作性，小组成员一起分析问题、解决问题，遇到需要沟通、分工的点都会积极参与进来，分工明确和热情投入是本次实验得以高效、高质量完成的关键因素，通过本次实验的完成也反映了我们编程思想方面存在误区和不足，基础掌握不够牢固，总言这次实验获益良多。 | | | | | | | | | | |
| 教 师 评 语 | 评价指标：   * 题目内容和要求完成情况 优 □ 良 □ 中 □ 差 □ * 对算法原理的理解程度 优 □ 良 □ 中 □ 差 □ * 程序设计水平 优 □ 良 □ 中 □ 差 □ * 程序运行效果及正确性 优 □ 良 □ 中 □ 差 □ * 课程设计报告结构清晰 优 □ 良 □ 中 □ 差 □ * 报告中总结和分析详尽 优 □ 良 □ 中 □ 差 □ | | | | | | | | | | |
|  |  | | | | | **教师签名** |  | | | | |

1. **文件管理和用户接口**
2. **需求分析**
3. 文件的逻辑结构

文件的逻辑结构采用流式结构； 文件用文本文件； 假设系统中只有两种文件，一种是存放任意字符的文件，一种是可执行文件。 可执行文件的内容就是模拟系统内进程的程序体，手工输入即可。

1. 磁盘模拟

用一个文件 disk 模拟磁盘，设磁盘的每个盘块 64 字节，模拟磁盘共有 128 块。第 0、1 块 39 存放文件分配表，第 2 块存放根目录，其余存放子目录和文件。（所以你创建的目录和流式文件 不能太大太多，但至少要包含 5 个目录和 15 个文件。注意：文件对磁盘块是独占的，即一个文 件至少占据一个磁盘块，而一个磁盘块中不能同时存放两个文件的数据。）

1. 目录结构 、

目录结构采用树型目录结构。 (a)目录项内容： 每个目录项 8 个字节，其中： 目录名或文件名：3 个字节； 扩展名：1 个字节（可执行文件扩展名为 e，目录没有扩展名）； 目录、文件属性：1 字节； 起始盘号：1 字节； 文件长度：2 字节（目录没有长度）。 (b)根目录 根目录位置固定，为磁盘第 2 块，大小固定，共可包含 8 个目录项，占用模拟磁盘第 2 块； （c）子目录 位置不固定，大小不固定。

1. 磁盘的分配采用链接结构（显式链接）的分配方式。

系统采用文件分配表方式记录磁盘空间的 使用情况和链接结构的指针。（参考 MSDOS 的 FAT） 文件分配表中一项需要 1 字节，而磁盘有 128 块，因而 128 项，模拟磁盘空间中的第 0、1 块用来存放文件分配表。

1. 用户接口

用户接口提供用户命令接口，接收用户从键盘键入的命令，要求实现以下命令： (下面例子中的 $ 只是命令提示符而已，和 UNIX 无关 。 除 aa 外，其余均为文件名。你可 以根据自己实现方便或喜欢而自定义命令参数。） 需要实现的命令包括： 创建文件： create例如 $ create \aa\bb . e 删除文件： delete例如 $ delete \aa\yy 显示文件： type 例如 $ type \zz 拷贝文件： copy 例如 $ copy \xx \aa\yy 建立目录： mkdir例如 $ mkdir \dd

1. 屏幕显示

屏幕显示要求包括： 用户命令接口：用于系统运行时用户输入命令； 磁盘目录显示：要求显示磁盘的目录结构；（不需要画成“树状”） 磁盘使用情况：显示磁盘每一个磁盘块的空间是占用还是空闲。

1. **概要设计**
2. 鼠标目录事件流程。



1. 用户接口流程



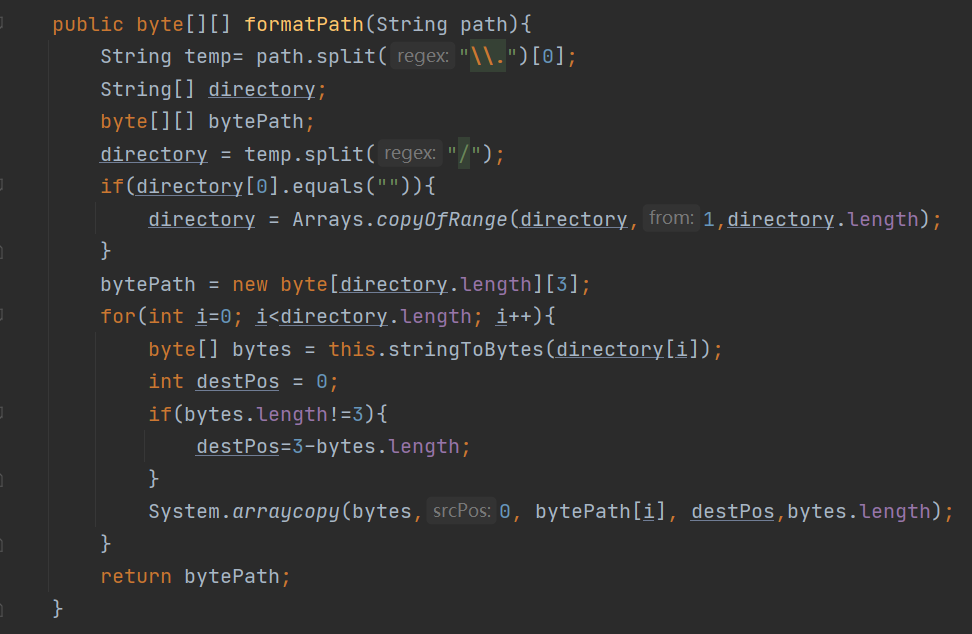
1. **详细设计**
2. Disk.java

Number和二字节整数的互相转换





将路径字符串格式化为字节型并返回一个二维字节数组，数组每行为文件或目录数据

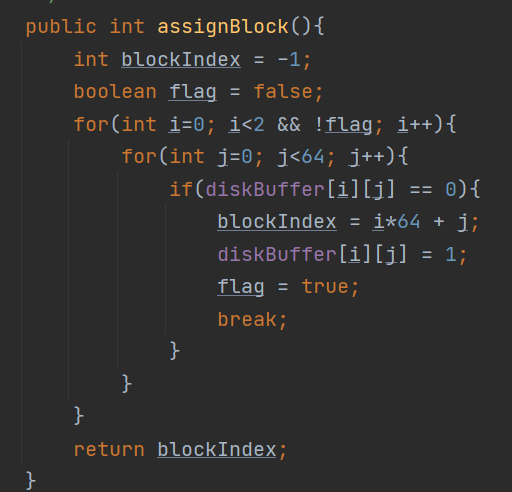


1. Fat.java

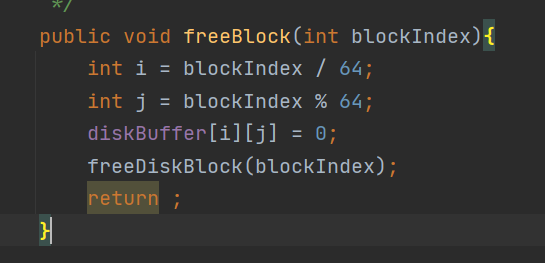
判断是否有num个空磁盘



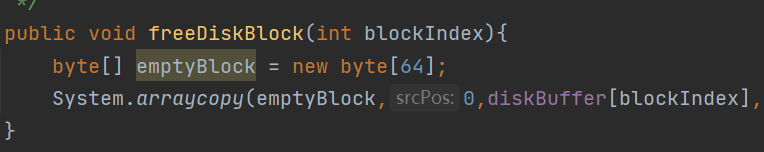
查找文件分配表并分配磁盘空盘块



将该盘块文件分配表中的位置数据置零,并将blockIndex指向盘块释放



将blockIndex盘块清空

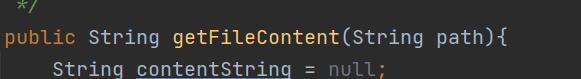


FiluUtil.java

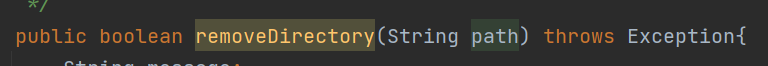
返回path文件或目录所占有的盘块



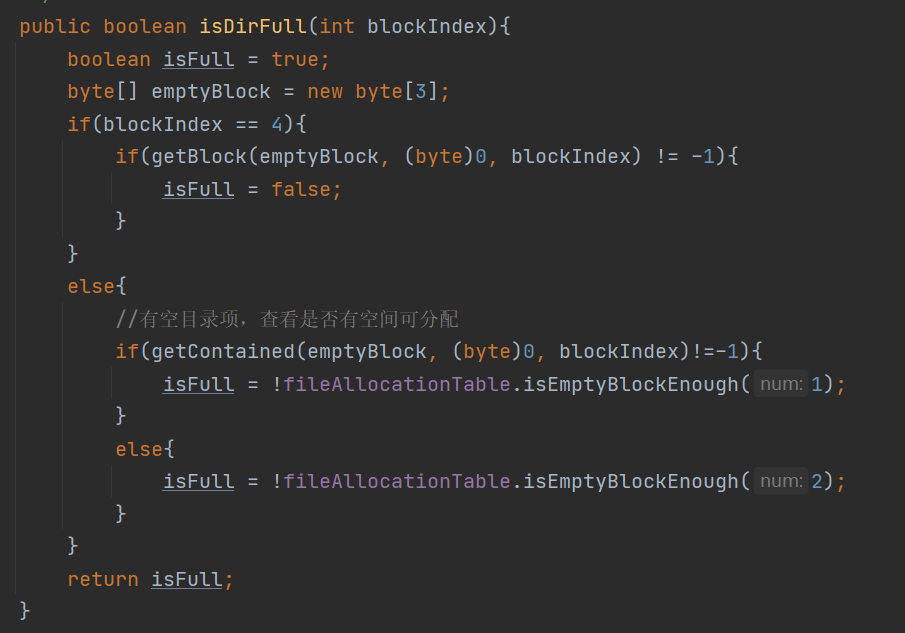
获取文件内容并返回字符串



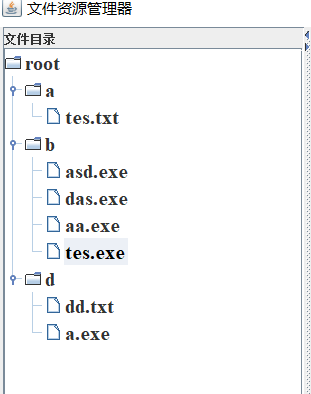
从磁盘中移除空目录

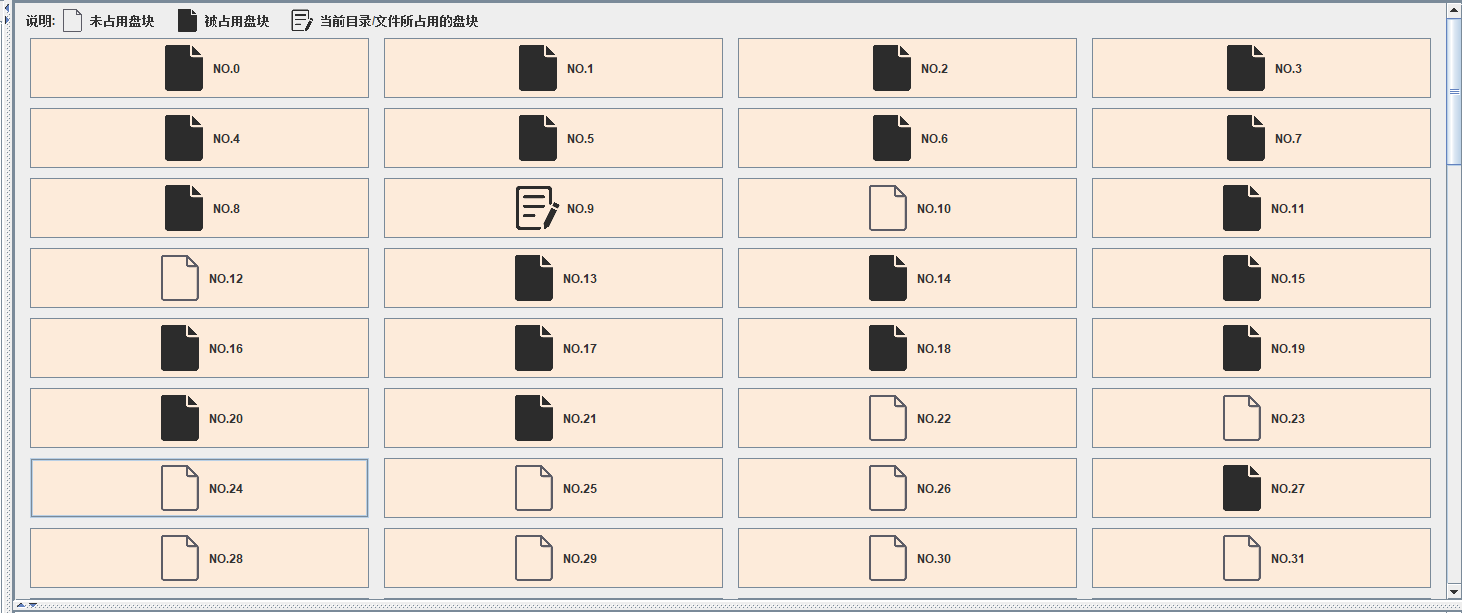


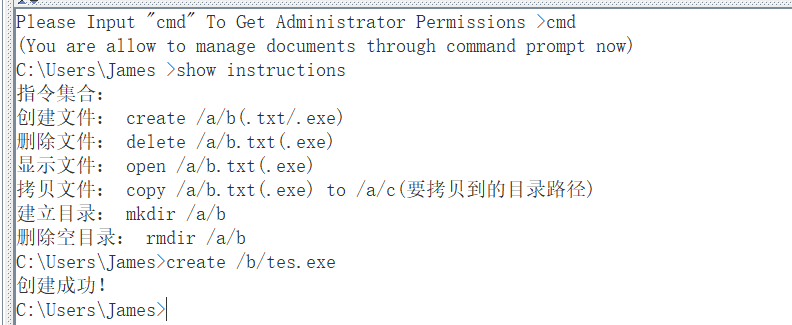
判断目录空间是否已满



1. **运行结果**







1. **心得体会**
2. 调试过程遇到的问题和解决方法

在调试过程中遇到了问题就是删除的时候，我只是清除了文件目录项和FAT表，并没有清除磁盘中的内容。可是这样会导致下次判断这个磁盘块是否被占用的时候判断异常。所以我每次删除的额时候都将磁盘内容还原成初始状态，保证不会出现上述异常。

1. 设计思路和设计想法
2. 分层设计

为了多人合作的协调性，整个项目结构分成4部分分别是文件管理和用户接口，内存，进程，设备管理等四部分。

1. 总结

由于使用了相对比较良好的框架，所以总体设计过程中比较少冲突和修改，也说明软件工程那一套理论确实存在很大的实用价值。唯一不足之处是本次代码没有使用git这类代码管理工具，使得每次都是使用压缩包进行传输。代码回滚也比较麻烦。

1. **存储管理**

## 需求分析

存储管理部分主要实现内存空间的分配和回收、存储保护。用数组模拟内存空间。

（1）模拟系统中，主存部分分为两部分，一部分是系统区，这里只存放进程控制块和内存分 配表，一部分是用户区，这里主要是对用户区的管理。 系统区包括 PCB 区域、内存空间分配表；用户区用数组模拟，大小为 512 字节，存储管理采用动态分区存储管理方式并能实现 “可合并碎片”

（2）用户界面显示内存使用情况示意图，以不同的颜色表示哪些区域已分配或未分配。

## 概要设计

### 2.1数据类型定义

1.内存空间的分配和回收采用动态分区存储管理方式，采用最佳适应算法。

2.定义Hole类模拟内存空间中的内存块，具有起始地址、内存块大小、是否分配等属性。创建链表对象holes模拟内存空间分配表，链表元素为模拟内存块的类hole。

3.定义ProcessAddress类记录进程id和所分配的内存块

### 2.2首次适配算法流程图



### 2.3释放内存算法流程图



## 详细设计

### 3.1伪码算法：

申请内存：

*//申请内存*public void getMemory(int size,int location,String id,Hole hole) { *//size为申请大小 location为分配分区的下标  
 // hole为location对应的分区，newhole为分配完后剩余的空闲分区* if (hole.getSize() - size >= *MIN\_SIZE*) {*//如果当前分区大小满足申请空间* Hole newHole = new Hole(null,hole.getHead() + size, hole.getSize() - size);*//创建一个新的空内存块* holes.add(location + 1, newHole);*//更新内存信息* hole.setSize(size);*//分配空间* hole.setUid(id);*//分配进程标识符* }  
 pcbs.add(new ProcessAddress(id, hole));*//更新内存表* hole.setFree(false);*//将该内存块状态设为非空* System.*out*.println("成功分配大小为" + size + "的内存");*//输出分配结果*}

回收内存：

*// 内存回收*public void releaseMemory(String id){*//传入进程id* ProcessAddress pcb = null;  
 boolean flag = false;*//记录是否找到相对应的内存分区* for(int i =0; i < pcbs.size(); i++){*//遍历内存表* if(id.equals(pcbs.get(i).getId())== true){*//如果找到对应的内存* pcb = pcbs.get(i);*//获取内存信息* flag = true;*//记录是否找到分区* break;  
 }  
 }  
 if(flag == false){*//如果没找到* System.*out*.println("无【"+id+"】分区");  
 }  
 if (pcb != null) {*//如果找到了* for (int i = 0; i < holes.size(); i++) {  
 Hole hole = holes.get(i);*//获取对应的内存块  
 //地址和空间相同，找到目标位置* if ((pcb.getHole().getSize() == hole.getSize()) && (pcb.getHole().getHead() == hole.getHead())) {  
 id1 = i;*//记录位置* break;  
 }  
 }  
 }  
 Hole hole = holes.get(id1);  
 hole.setUid(null);*//id置为null即删除原有的进程标识符* if(hole.isFree()){*//如果当前内存块为空* System.*out*.println("【" + id + "】此空间空闲，无需释放");  
 }  
 for (int i = 0; i < pcbs.size(); i++){  
 ProcessAddress pcb2 = pcbs.get(i);  
 *//判断条件：起始地址相同并且内存块大小相同* if((pcb2.getHole().getHead()==hole.getHead())&&(pcb2.getHole().getSize()==hole.getSize())){  
 pcbs.remove(i);*//删除进程控制块* break;  
 }  
 }  
 *//判断后面分区是否为true，合并分区* if (id1 < holes.size()-1&&holes.get(id1+1).isFree()){  
 Hole nextHole = holes.get(id1 + 1);*//id1记录当前所删除的分区* hole.setSize(hole.getSize()+nextHole.getSize());*//合并空间* holes.remove(nextHole);*//从内存中删除多余分区* }  
 *//判断前面分区是否为true，合并分区* if (id1 >0 &&holes.get(id1-1).isFree()){*//id1记录当前所删除的分区，如果前一个分区为空* Hole lastHole = holes.get(id1 - 1);*//获得前面一个分区* lastHole.setSize(hole.getSize()+lastHole.getSize());*//和前面分区合并空间* holes.remove(id1);*//删除前面分区* id1--;*//删除后位置往前* }  
 holes.get(id1).setFree(true);*//设置为空* System.*out*.println("回收内存成功");  
}

最佳适应算法：

int findIndex = -1; *//最佳分区的下标*int min = this.getSize();*//首先给min赋一个最大值*for (int i = 0; i < this.getHoles().size();i++){*//遍历所有内存块*  
 Hole hole = this.getHoles().get(i);*//获取一个内存块* if(hole.isFree() && hole.getSize()>=size){*//如果当前内存块空闲并且容量大于新建进程所需要的大小  
 //hole.getsize()-size表示空闲区分配之后所剩余的空间* if(min >hole.getSize() - size){*//找到满足条件的空闲分区中找到其中最小的空闲分区* min = hole.getSize();*//更新最小分区大小*  
 findIndex = i ; *//记录当前最小分区下标*  
 }  
 }  
}  
if (findIndex != -1) { *//若存在合适分区* this.getMemory(size, findIndex, id,this.getHoles().get(findIndex));  
 return true;  
}  
else {*//没有合适的分区* System.*err*.println("内存不足！");  
 return false;  
}

在用户窗口显示内存使用情况：

public void printScreen4(LinkedList<Hole> holes){*/\*在一个Label里面添加button组件，ss用于记录横坐标位置，\*/* jLabel1MainMemory.removeAll();  
 int ss = 26;  
 for (int i = 0; i < holes.size(); i++){  
 string = null;  
 *//获取进程id* Hole hole = holes.get(i);  
 if(hole.getUid() != null)  
 {  
 string = hole.getUid().substring(9,13);  
 }  
 Button button = new Button(string);  
 *//空闲区颜色设为黑色，占用区颜色设置为绿色* if (hole.isFree()){  
 button.setBackground(Color.*BLACK*);  
 }  
 else {  
 button.setBackground(Color.*green*);  
 }  
 jLabel1MainMemory.add(button);  
 button.setBounds(ss,30,hole.getSize(),50);  
*// jLabel1MainMemory.add(button);* ss += hole.getSize();  
 }  
 }

### 3.2函数调用关系图



## 调试分析

### (1) 遇到的问题及解决方法

在进程运行结束进行内存回收时候，我在合并内存分区时遇到了问题。最开始设计考虑时，只想到了和后面的空闲区合并，但是忽略了前面的空闲区，导致部分内存区空间较小不足以分配新的进程，造成了空间浪费。后来我加多了一个判断，如果所释放的位置前方的内存区也为空，则一起合并。

### (2) 算法时间复杂性和改进设想；

最佳适应算法BestFit()：

最佳适应算法时间复杂度为O（n）。可以将所有空闲区按其大小排序后，以递增顺序形成一个空白链，这样子在查找时找到的第一个满足所需空间的空闲区即是最优的空闲区。这样能很好的改进时间复杂度。

### (3) 设计过程的经验和体会；

在本次实验中我主要负责存储管理部分以及部分用户界面设计与实现。在实验开发过程之中设计到了很多操作系统方面的知识，例如进程的创建和内存的分配等，以这些知识为基础才能在开发过程中事半功倍，但在开发的过程之中发现我的知识储备并不充足，有些知识已经被淡忘，然后回过头再去复习课本和以前上课的课件。在操作系统中内存起着非常重要的作用，内存分配设计好了才能获得较好的运行效率。在分配内存时我选择使用的是最佳适应算法，该算法能很好的提高内存使用率。在释放内存时候没有考虑周全，导致合并分区时出现较多bug，一边设计一边制作一边解决bug。在设计用户界面显示内存使用情况时并不知道如何入手，然后在网上查阅资料后，找到一个很巧妙的方法，分配给进程的内存都有一个内存地址和大小，那么我们可以在一个label横着摆放多个button组件，所有button组件的height属性和纵坐标是固定的，我们以内存地址作为横坐标，内存大小作为长度，那么就可以在指定位置输出内存块了，当多个内存块连接起来，那就能很好的模拟出内存。同时使用不同的颜色分别表示被占用的内存区已经空闲区。在这过程当中很感谢队友们对我的悉心指导和帮助。存储管理部分和其它模块有着较深的联系，例如进程创建的时候会分配内存等，在开发过程中避免不了和队友们的沟通和交流。而在开发过程中我们团结协作，共同进退，不仅提高了效率，也加深了我们之间的友谊。这次实验提高了我的代码编写水平，提高了自己的自信心，从最开始看到题目的无从下手到完成实验之后的那份自豪，这段时间的努力有了结果。此外也对操作系统的原理更加清晰，也明白了自己还有很多不足，需要不断学习，不断提高自己。今后还需要更加努力！

### (4) 实现过程中出现的主要问题及解决方法。

出现的问题：如何高效率管理内存的分配。

解决方法：

我们采用动态分区管理存储方式中的最佳适应算法。该算法能把满足进程要求又是当前最小的空闲分区分配给进程。所以每次分配都是最合适该文件大小的分区，提高了内存的利用率。

## 关健界面



1. **设备管理**

### 需求分析

1. 模拟操作系统中有A、B、C三种独占型设备，A设备1个，B设备2个，C设备2个。（同一种设备的实体被认为是相同的）。
2. 采用死锁预防方法来处理申请独占设备可能造成的死锁。
3. 屏幕显示：每个设备是否被使用。哪个设备在使用该设备，哪些进程在等待使用该设备。
4. 设置设备分配表和等待队列

### 概要设计

1. 抽象出了 Device.java类
2. 抽象出了DeviceTable.java类
3. 抽象出了 DeviceWaitQueue.java类





### 详细设计

设备类

getDevice( )

@申请设备

@param Uid

@param time

@param size

@return int 设备号

伪代码：

在设备表中查找要申请的设备。

改变设备的状态，分配设备。

removeDevice( )

@释放设备

@param Uid

@return int[]

伪代码：

在设备表中查找要释放的设备。

释放设备后，从等待队列中取出一个节点。

getFirstNode( )

@获取等待队列中的节点

@return int

伪代码：

在对应设备中的等待队列中取出对应的节点。

**public class** Device {  
 */\* A、B、C三种独占型设备，A设备1个，B设备2个，C设备2个。 \*/* **private int** returnIndexA1 = 1;  
 **private int** returnIndexB1 = 2;  
 **private int** returnIndexB2 = 3;  
 **private int** returnIndexC1 = 4;  
 **private int** returnIndexC2 = 5;  
 **private int** c;  
 */\* 同一设备的实体被认为是相同的 \*/* **private** LinkedList<DeviceWaitQueue> blockA ;  
 **private** LinkedList<DeviceWaitQueue> blockB ;  
 **private** LinkedList<DeviceWaitQueue> blockC ;  
 **private** DeviceTable deviceTable;  
 DeviceWaitQueue deviceWaitQueueA;  
 DeviceWaitQueue deviceWaitQueueB;  
 DeviceWaitQueue deviceWaitQueueC;  
 **public** Device() {  
 **this**.blockA = **new** LinkedList<DeviceWaitQueue>();  
 **this**.blockB = **new** LinkedList<DeviceWaitQueue>();  
 **this**.blockC = **new** LinkedList<DeviceWaitQueue>();  
 **this**.deviceTable = **new** DeviceTable();  
 }  
 */\*\*  
 \** ***@Description*** *申请A设备  
 \** ***@param*** *Uid  
 \** ***@param*** *time  
 \** ***@param*** *size  
 \*/* **public int** getDeviceA(String Uid, **int** time, **int** size ){  
 deviceWaitQueueA =**new** DeviceWaitQueue(Uid,size,time);  
 **if** (deviceTable.getA1().equals(**"设备空闲"**) ){  
 deviceTable.setA1(Uid);  
 c = returnIndexA1;  
 returnIndexA1 = 9;  
 **return** c;  
 }  
 **else** {  
 blockA.add(deviceWaitQueueA);  
 **return** 9;  
 }  
  
 }  
 */\*\*  
 \** ***@Description*** *释放A设备  
 \** ***@param*** *Uid  
 \*/* **public int**[] removeDeviceA(String Uid){  
 **int**[] re =**new int**[2];  
 **if** (Uid.equals(deviceTable.getA1())){  
 deviceTable.setA1(**"设备空闲"**);  
 **if**(!blockA.isEmpty()) {  
 re[0] = 1;  
 re[1] = gerFirstNodeA();  
 **return** re;  
 }  
 returnIndexA1 = 1;  
 **return null**;  
 }**else  
 return null**;  
 }  
  
 */\*\*  
 \** ***@Description*** *获取等待队列第一个节点  
 \*/* **public int** gerFirstNodeA(){  
 deviceWaitQueueA = blockA.removeFirst();  
 getDeviceA(deviceWaitQueueA.getUid(),deviceWaitQueueA.getTime(),deviceWaitQueueA.getSize());  
 **return** deviceWaitQueueA.getTime();  
 }  
 */\*\*  
 \** ***@Description*** *申请B设备  
 \** ***@param*** *Uid  
 \** ***@param*** *time  
 \** ***@param*** *size  
 \*/* **public int** getDeviceB(String Uid, **int** time, **int** size ){  
 deviceWaitQueueB =**new** DeviceWaitQueue(Uid,size,time);  
 **if** (deviceTable.getB1().equals(**"设备空闲"**) ){  
 deviceTable.setB1(Uid);  
 c = returnIndexB1;  
 returnIndexB1 = 9;  
 **return** c;  
 }  
 **else if**(deviceTable.getB2().equals(**"设备空闲"**) ) {  
 deviceTable.setB2(Uid);  
 c = returnIndexB2;  
 returnIndexB2 = 9;  
 **return** c;  
 }  
 **else**{  
 blockB.add(deviceWaitQueueB);  
 **return** 9;  
 }  
 }  
 */\*\*  
 \** ***@Description*** *释放B设备  
 \** ***@param*** *Uid  
 \*/* **public int**[] removeDeviceB(String Uid){  
 **int**[] re =**new int**[2];  
 **if** (Uid.equals(deviceTable.getB1())){  
 deviceTable.setB1(**"设备空闲"**);  
 **if**(!blockB.isEmpty()) {  
 re[0] = 3;  
 re[1] = gerFirstNodeB();;  
 **return** re;  
 }  
 returnIndexB1 = 3;  
 **return null**;  
 }**else if** (Uid.equals(deviceTable.getB2())){  
 deviceTable.setB2(**"设备空闲"**);  
 **if**(!blockB.isEmpty()) {  
 re[0] = 4;  
 re[1] = gerFirstNodeB();;  
 **return** re;  
 }  
 returnIndexB2 = 4;  
 **return null**;  
 }  
 **else  
 return null**;  
 }  
 */\*\*  
 \** ***@Description*** *获取B设备等待队列第一个节点  
 \*/* **public int** gerFirstNodeB(){  
 deviceWaitQueueB = blockB.removeFirst();  
 getDeviceB(deviceWaitQueueB.getUid(),deviceWaitQueueB.getTime(),deviceWaitQueueB.getSize());  
 **return** deviceWaitQueueB.getTime();  
 }  
 */\*\*  
 \** ***@Description*** *申请C设备 两个c设备  
 \** ***@param*** *Uid  
 \** ***@param*** *time  
 \** ***@param*** *size  
 \*/* **public int** getDeviceC(String Uid, **int** time, **int** size ){  
 deviceWaitQueueC = **new** DeviceWaitQueue(Uid,size,time);  
 **if** ( deviceTable.getC1().equals(**"设备空闲"**) ){  
 deviceTable.setC1(Uid);  
 c = returnIndexC1;  
 returnIndexC1 = 9;  
 **return** c;  
 }  
 **else if**( deviceTable.getC2().equals(**"设备空闲"**) ){  
 deviceTable.setC2(Uid);  
 c = returnIndexC2;  
 returnIndexC2 = 9;  
 **return** c;  
 }  
 **else**{  
 blockC.add(deviceWaitQueueC);  
 **return** 9;  
 }  
 }  
 */\*\*  
 \** ***@Description*** *释放C设备  
 \** ***@param*** *Uid  
 \*/* **public int**[] removeDeviceC(String Uid){  
 **int**[] re = **new int**[2];  
 **if** (Uid.equals(deviceTable.getC1())){  
 deviceTable.setC1(**"设备空闲"**);  
 **if**(!blockC.isEmpty()) {  
 re[0] = 6;  
 re[1] = gerFirstNodeC();;  
 **return** re;  
 }  
 returnIndexC1 = 6;  
 **return null**;  
 }**else if** (Uid.equals(deviceTable.getC2())){  
 deviceTable.setC2(**"设备空闲"**);  
 **if**(!blockC.isEmpty()) {  
 re[0] = 7;  
 re[1] = gerFirstNodeC();;  
 **return** re;  
 }  
 returnIndexC2 = 7;  
 **return null**;  
 }  
 **else  
 return null**;  
 }  
 */\*\*  
 \** ***@Description*** *获取B设备等待队列第一个节点  
 \*/* **public int** gerFirstNodeC(){  
 deviceWaitQueueC = blockB.removeFirst();  
 getDeviceC(deviceWaitQueueC.getUid(),deviceWaitQueueC.getTime(),deviceWaitQueueC.getSize());  
 **return** deviceWaitQueueC.getTime();  
 }  
  
 **public** DeviceTable getDeviceTable() {  
 **return** deviceTable;  
 }  
  
 **public void** setDeviceTable(DeviceTable deviceTable) {  
 **this**.deviceTable = deviceTable;  
 }  
  
 **public** LinkedList<DeviceWaitQueue> getBlockA() {  
 **return** blockA;  
 }  
  
 **public void** setBlockA(LinkedList<DeviceWaitQueue> blockA) {  
 **this**.blockA = blockA;  
 }  
  
 **public** LinkedList<DeviceWaitQueue> getBlockB() {  
 **return** blockB;  
 }  
  
 **public void** setBlockB(LinkedList<DeviceWaitQueue> blockB) {  
 **this**.blockB = blockB;  
 }  
  
 **public** LinkedList<DeviceWaitQueue> getBlockC() {  
 **return** blockC;  
 }  
  
 **public void** setBlockC(LinkedList<DeviceWaitQueue> blockC) {  
 **this**.blockC = blockC;  
 }  
  
 **public** DeviceWaitQueue getDeviceWaitQueueA() {  
 **return** deviceWaitQueueA;  
 }  
  
 **public void** setDeviceWaitQueueA(DeviceWaitQueue deviceWaitQueueA) {  
 **this**.deviceWaitQueueA = deviceWaitQueueA;  
 }  
  
 **public** DeviceWaitQueue getDeviceWaitQueueB() {  
 **return** deviceWaitQueueB;  
 }  
  
 **public void** setDeviceWaitQueueB(DeviceWaitQueue deviceWaitQueueB) {  
 **this**.deviceWaitQueueB = deviceWaitQueueB;  
 }  
  
 **public** DeviceWaitQueue getDeviceWaitQueueC() {  
 **return** deviceWaitQueueC;  
 }  
  
 **public void** setDeviceWaitQueueC(DeviceWaitQueue deviceWaitQueueC) {  
 **this**.deviceWaitQueueC = deviceWaitQueueC;  
 }  
}

设备表

get( )

@获取设备的状态

@return string

伪代码：

返回设备的状态

set( )

@设置设备的状态

伪代码：

设置设备的状态

**public class** DeviceTable {  
 **private** String A1;  
 **private** String B1;  
 **private** String B2;  
 **private** String C1;  
 **private** String C2;  
  
 **public** DeviceTable() {  
 A1 = **"设备空闲"**;  
 B1 = **"设备空闲"**;  
 B2 = **"设备空闲"**;  
 C1 = **"设备空闲"**;  
 C2 = **"设备空闲"**;  
 }  
  
 **public** String getA1() {  
 **return** A1;  
 }  
  
 **public void** setA1(String a1) {  
 A1 = a1;  
 }  
  
 **public** String getB1() {  
 **return** B1;  
 }  
  
 **public void** setB1(String b1) {  
 B1 = b1;  
 }  
  
 **public** String getB2() {  
 **return** B2;  
 }  
  
 **public void** setB2(String b2) {  
 B2 = b2;  
 }  
  
 **public** String getC1() {  
 **return** C1;  
 }  
  
 **public void** setC1(String c1) {  
 C1 = c1;  
 }  
  
 **public** String getC2() {  
 **return** C2;  
 }  
  
 **public void** setC2(String c2) {  
 C2 = c2;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "DeviceTable:\n"** +  
 **"A='"** + A1 + **'\''** +  
 **"\nB='"** + B1 + **'\''** +  
 **"\nB='"** + B2 + **'\''** +  
 **"\nC='"** + C1 + **'\''** +  
 **"\nC='"** + C2 + **'\''** +  
 }  
}

等待队列

getUid( )

@获取等待队列的状态

@return string

伪代码：

返回等待队列的状态

setUid( )

@设置等待队列的状态

伪代码：

设置等待队列的状态

getSize( )

@获取等待队列的大小

@return int

伪代码：

返回等待队列的大小

setSize( )

@设置等待队列的大小

伪代码：

设置等待队列的大小

getTime( )

@获取等待队列的时间

@return int

伪代码：

返回等待队列的时间

setUid( )

@设置等待队列的状态

伪代码：

设置等待队列的状态

**public class** DeviceWaitQueue {  
 **private** String uid;  
 **private int** size;  
 **private int** time;  
  
 **public** DeviceWaitQueue(String Uid, **int** size, **int** time ){  
 **this**.uid = Uid;  
 **this**.size = size;  
 **this**.time = time;  
 }  
  
 **public** String getUid() {  
 **return** uid;  
 }  
  
 **public void** setUid(String uid) {  
 **this**.uid = uid;  
 }  
  
 **public int** getSize() {  
 **return** size;  
 }  
  
 **public void** setSize(**int** size) {  
 **this**.size = size;  
 }  
  
 **public int** getTime() {  
 **return** time;  
 }  
  
 **public void** setTime(**int** time) {  
 **this**.time = time;  
 }  
}

### 运行效果



### 个人感悟

在这一个学期的操作系统课程设计中我主要做设备管理的实现，收获了很多也学会了很多：

1. 代码能力，学习了很多关于java.swing里的知识，对处理图形界面有了初步的认识。
2. 对于这次负责设备管理，让我复习了os课程里传授的知识，在实战中运用所学，巩固知识。
3. 这次团队的分工，很好的解耦，每个人各自分工，合理使得团队效益最大化。

## 进程管理

### 需求分析

1. CPU：模拟中央处理器，用于解释执行“可执行文件”里面的指令。

x=? 给x赋值

x++ x加1

x-- x减1

!A? 占用A设备？时间。

寄存器模拟：

PSW，IR，PC，DR

中断模拟：

①进程结束②时间片到 ③I/O中断

1. PCB：

包括进程标识符、主要寄存器、进程状态、阻塞原因。三个队列：空白PCB队列，就绪队列和阻塞队列。

1. 进程调度：
   1. 采用时间片轮转法，时间片为5
   2. 将正在运行进程的现场（寄存器组）保存在该进程的 PCB 中；从就绪队列中选择一个进程；将这个进程的 PCB 中记录的各寄存器内容恢复到 CPU 各个寄存器内。（恢复现场）
   3. 闲逛进程
2. 进程控制：
   1. 创建进程
   2. 进程销毁
   3. 进程阻塞
   4. 进程唤醒

### 概要设计

CPU流程图



进程运行流程



### 详细设计

寄存器



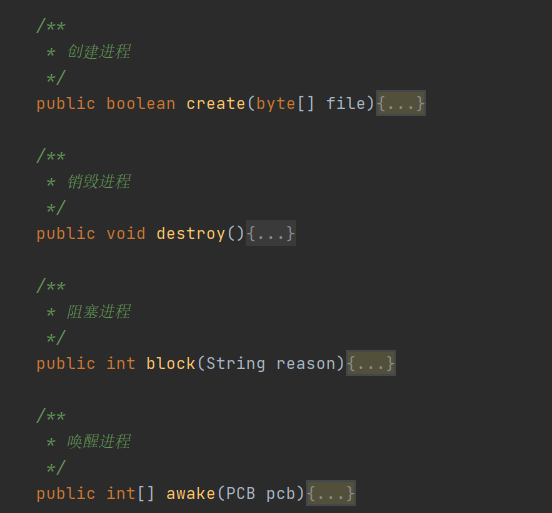
Cpu类





ProcessScheduling类

主要方法：



1. create()

输入值：file

输出值：boolean是否申请成功

操作变量：PCB进程及内存data

伪代码：申请内存空间

将进程的数据从磁盘读入内存

初始化pcb

将pcb保存入进程管理器的就绪队列

1. block()

输入值：设备号

输出值：无

操作变量：想要阻塞的PCB进程

伪代码：将进程从运行态转换成阻塞态

将进程pcb加入阻塞队列

保存现场

1. awake()

输入值：PCB

输出值：无

操作变量：想要唤醒的PCB进程

伪代码：将进程从阻塞态转换成就绪态

将进程的pcb加入就绪队列

1. destory()

输入值：无

输出值：无

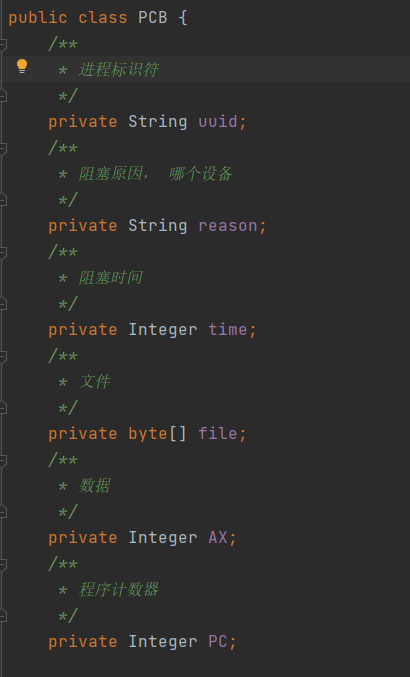
操作变量：想要销毁的PCB进程

伪代码：将进程从阻塞队列，就绪队列中删除

并且将pcb销毁

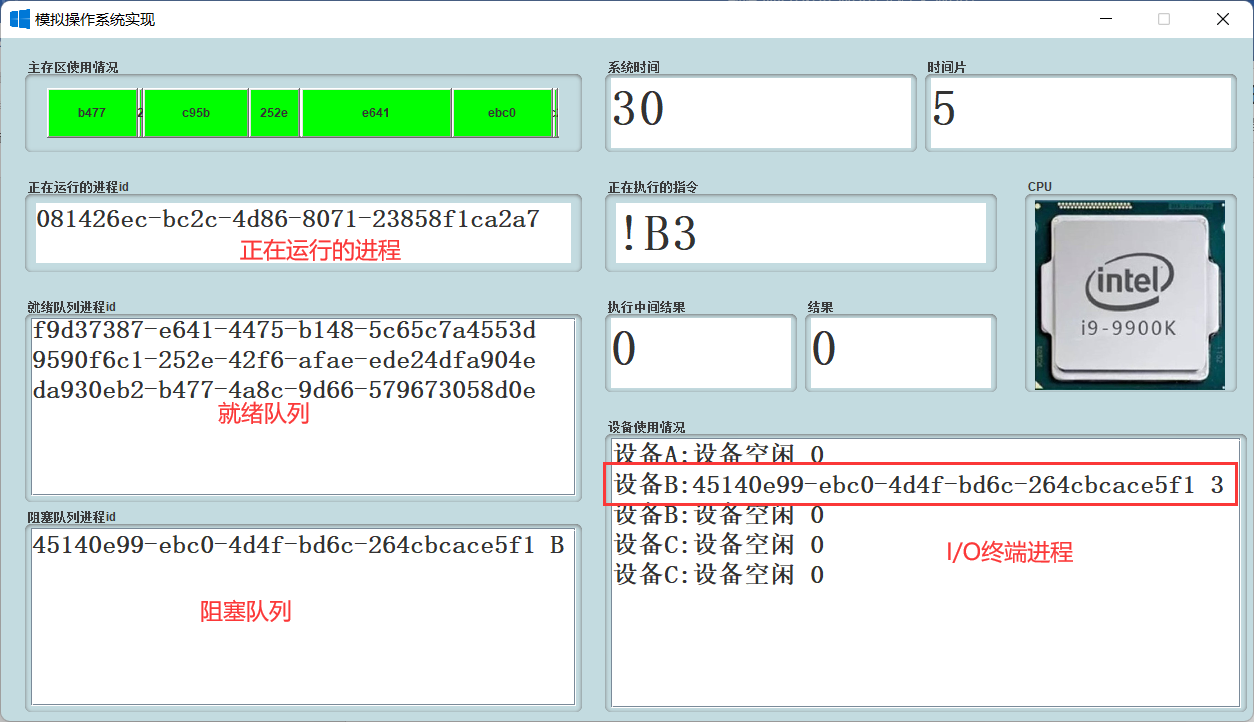
执行内存回收

PCB类



### 运行效果





### 个人感悟

在这一个学期的操作系统课程设计中我主要做cpu和进程管理的实现，收获了很多也学会了很多：

1. 代码能力，尝试了很多之前没有接触过的类和软件，极大提高了开发效率，也在其中学到了遇到问题解决问题的能力。
2. 对于操作系统有了更深层的了解和理解，在我遇到实现逻辑的问题时，回去翻课本和回顾老师之前所讲的内容，根据这些理论，很完美的解决我在实现逻辑上遇到的种种难题。
3. 其他课程与会应用其中，让我更深的理解了计算机及其架构。我在实现cpu时应用到了我在数据结构上学的知识还有计算机组成原理的知识，多门课相互融合，让我更加明白了计算机系统
4. 团队的重要性，当遇到难题解决不了时，团队提供了很多帮助，我们一起实现了分工完成，无缝连接，把课设分块，大家协力完成，比一个人完成效率高很多，也对以后团队合作提供了不少经验。

这是一次不错的课设体验。