|  |  |  |
| --- | --- | --- |
| **计算机操作系统课程设计**  **实践报告** | | |
|  | 题 目: | 仿真实现作业管理与虚页存储管理 |
|  | 姓 名: | 郭晗 |
|  | 学 院: | 人工智能学院 |
|  | 专 业: | 计算机科学与技术 |
|  | 班 级: | 计科191班 |
|  | 学 号: | 19219124 |
|  | 指导教师: | 姜海燕 职称: 教授 |
| 2022年3月 24 日 | | |

目录

[**仿真实现作业管理与虚页存储管理** 4](#_Toc99043700)

[**1 实践内容与目标** 5](#_Toc99043701)

[**1.1 实践内容** 5](#_Toc99043702)

[**1.2 实践目标** 5](#_Toc99043703)

[**1.2.1作业管理模块** 5](#_Toc99043704)

[**1.2.2虚页内存管理** 5](#_Toc99043705)

[**2 系统功能与 API 函数说明** 5](#_Toc99043706)

[**2.1硬件系统** 6](#_Toc99043707)

[**2.1.1内存Memory** 6](#_Toc99043708)

[**2.1.2外存Harddisk** 6](#_Toc99043709)

[**2.1.3CPU** 6](#_Toc99043710)

[**2.1.4MMU** 6](#_Toc99043711)

[**2.2作业管理** 7](#_Toc99043712)

[**2.2.1CreateJobs** 7](#_Toc99043713)

[**2.2.2ExtraJobs** 7](#_Toc99043714)

[**2.2.3JobsScheduling** 7](#_Toc99043715)

[**2.2.4ProcessScheduling** 7](#_Toc99043716)

[**2.4可视化** 8](#_Toc99043717)

[**3 硬件仿真设计** 8](#_Toc99043718)

[**3.1内存** 8](#_Toc99043719)

[**3.2外存** 9](#_Toc99043720)

[**3.3MMU** 9](#_Toc99043721)

[**4 数据结构与基础操作的抽象与设计** 9](#_Toc99043722)

[**4.1通用数据结构** 9](#_Toc99043723)

[**4.1.1指令Instruction** 9](#_Toc99043724)

[**4.1.2作业JCB** 9](#_Toc99043725)

[**4.1.3进程PCB** 10](#_Toc99043726)

[**4.1.4快表项TLB** 10](#_Toc99043727)

[**4.1.5页表项Page** 11](#_Toc99043728)

[**4.1.6逻辑地址LogicAddress** 11](#_Toc99043729)

[**4.2.基础操作** 11](#_Toc99043730)

[**4.2.1原语** 11](#_Toc99043731)

[**4.2.2地址变换** 12](#_Toc99043732)

[**4.2.3页面分配与回收** 13](#_Toc99043733)

[**5 程序结构及模块功能的实现** 14](#_Toc99043734)

[**5.1程序结构** 14](#_Toc99043735)

[**5.2模块功能** 14](#_Toc99043736)

[**5.2.1作业创建** 14](#_Toc99043737)

[**5.2.1硬件系统** 14](#_Toc99043738)

[**5.2.2作业管理** 15](#_Toc99043739)

[**5.3.3存储管理** 16](#_Toc99043740)

[**6 测试与分析** 17](#_Toc99043741)

[**6.1RunProcess测试。** 17](#_Toc99043742)

[**6.1.1测试用例** 17](#_Toc99043743)

[**6.1.2界面介绍** 19](#_Toc99043744)

[**6.1.3运行步骤** 21](#_Toc99043745)

[**6.2CreateJobs和ExtraJobs测试。** 23](#_Toc99043746)

[**6.2.1生成作业** 23](#_Toc99043747)

[**6.2.2额外生成作业** 24](#_Toc99043748)

[**6.2.3开始调度** 24](#_Toc99043749)

[**6.3测试完成功能** 24](#_Toc99043750)

[**6.3.1作业管理** 24](#_Toc99043751)

[**6.3.2虚页内存管理** 28](#_Toc99043752)

[**6.3.3可执行程序文件测试** 30](#_Toc99043753)

[**7 技术问题及解决方案** 33](#_Toc99043754)

[**7.1文件写入与读出** 33](#_Toc99043755)

[**7.2多线程控制** 34](#_Toc99043756)

[**7.3Java内部队列方法** 34](#_Toc99043757)

[**7.4缓冲区设计** 34](#_Toc99043758)

[**7.5参数问题** 35](#_Toc99043759)

[**7.6CPU关中断** 35](#_Toc99043760)

[**7.7可视化仿真界面表格** 36](#_Toc99043761)

[**8 实践心得** 36](#_Toc99043762)

[**参考文献** 37](#_Toc99043763)

[**附件 1：程序文件及结构说明** 37](#_Toc99043764)

[**附件 2：类图和顺序图说明** 39](#_Toc99043765)

[**附件 3：带注释的部分核心代码** 40](#_Toc99043766)

**仿真实现作业管理与虚页存储管理**

计科191 郭晗

19219124

**摘要：**操作系统是计算机的核心和灵魂，为了更深层次地理解操作系统基本概念和原理、资源管理技术及并发程序设计技术等，开展了此次操作系统课程设计。本次操作系统课程设计以操作系统原理为指导，利用面对对象程序设计技术完成了仿真实现作业管理与虚页存储管理。操作系统是一门理论、实践与应用都很强的课程，仅仅学习理论只是很难达到理想教学效果，仿真实验有助于我们对操作系统的原理与应用有更透彻的认识。

本次操作系统课程设计采用JAVA语言，采用面向对象思想仿真设计CPU、时钟中断、内存、MMU等裸机硬件部件，在硬件基础上完成了作业管理和虚页内存管理两个部分，并在最后采用了可视化的方法将最终结果展现出来。

作业管理部分，主要通过硬件系统、高级调度和低级调度来实现作业管理与进程状态转换，首先通过可视化界面生成作业按钮生成作业序列，作业首先进入系统在辅存上的后备作业队列等候，通过高级调度决定把后备队列作业中的哪些作业调入主存，为它们创建进程、分配资源，并将它们排在进程就绪队列外，使得这些作业的进程获得竞争处理机的权利，准备执行；通过低级调度决定就绪[队列](https://baike.baidu.com/item/%E9%98%9F%E5%88%97" \t "https://baike.baidu.com/item/%E4%BD%8E%E7%BA%A7%E8%B0%83%E5%BA%A6/_blank)中的哪个进程将获得[处理机](https://baike.baidu.com/item/%E5%A4%84%E7%90%86%E6%9C%BA)。

虚页存储管理部分，主要通过PageScheduling类仿真，每当进程被创建，会为之分配3块，并生成页表，当进程访问页面不在内存中，通过缺页中断与虚页调度LRU算法将页面调入内存。进程结束时则回收页面。

测试运行时，主要分成三部分，生成作业、调度运行并随时添加作业三部分。主要通过GUI类可视化仿真界面，和生成的ProcessResults.txt文件查看运行结果。过程中会呈现进程阻塞队列、就绪队列内存占用情况等运行结果表明实现了作业并发环境，多个进程可以正常调度。

**关键词**：操作系统；仿真；作业管理；虚页存储管理；

**1 实践内容与目标**

* 1. **实践内容**

此次课程设计选题为仿真实现作业管理与虚页存储管理，通过实现硬件系统、作业管理、虚页存储管理、可视化界面四部分，在多任务的作业并发环境下，仿真实现作业及进程并发环境、进程原语、作业调度过程及算法、MMU 地址变换、页表生成、缺页中断与虚页调度算法（LRU）、页面分配与回收算法等，并将实现原理过程通过可视化方式呈现工作过程。

* 1. **实践目标**

以下为可视化仿真实现作业管理与虚页内存管理中两模块的主要功能与实现函数介绍，具体实现函数介绍可查看第二部分系统功能与API函数说明。

**1.2.1作业管理模块**

（1）进程并发环境：操作系统具有程序并发性。通过采用多线程的方式仿真实现系统时钟中断。

1. 进程原语：按照教材步骤仿真实现原语函数，可以作为PCB类的方法。进程创建：实现函数CreatePrimitive(Memory memory,JCB jcb,Harddisk harddisk)，不考虑作业调度，假设有足够内存，有作业请求后自动创建；进程撤销：实现函数RevokePrimitive(Memory memory)，执行完成的进程调用撤销函数；五个进程阻塞：实现函数BlockingPrimitive(Memory memory)，CPU 模式切换、系统调用时调用；五个进程唤醒：实现函数WakeupPrimitive(Memory memory)，在阻塞队列的进程，根据每种指令的运行时间来决定每几秒唤醒一个阻塞进程。
2. 作业调度过程及算法：作业管理的核心部分，课设中采用两级作业调度。高级调度实现JobsScheduling类，采用线程方式时刻检测是否有新作业请求，采用先来先服务调度算法，采用按作业进入后备队列的先后次序来分配处理机（为其创建进程）；低级调度实现ProcessScheduling类中函数SystemRun(Memory memory,Harddisk harddisk,CPU cpu,**int** time,MMU mmu,ClockInterrupt c)，采用时间片轮转调度算法和静态优先级调度算法结合，不是每次发生时钟中断都要进行进程调度。

**1.2.2虚页内存管理**

（1）MMU地址变换：实现逻辑地址到物理地址的转换，及虚拟内存管理。通过实现Logic\_To\_Physical(LogicAddress logicaddress,PCB pcb)实现这部分功能。

1. 页表生成：生成页表并页表变化过程写入文件。通过实现Ptabel(PCB p)实现生成页表。
2. 缺页中断与虚页调度算法：当进程访问页面不在内存时，系统发生缺页中断现象，并通过LRU页面置换算法替换相应的页面。通过PageFault(Page\_Fault\_Interrupt Process\_PageList[],int page,PCB pcb,Memory memory,MMU mmu,Harddisk harddisk)和PageLRU(Page\_Fault\_Interrupt Process\_PageList[])函数来实现。
3. 页面分配与回收：进程被创建或撤销时，需要对进程进行页面分配或回收。通过实现页面分配函数AllocatedPage(Memory memory,PCB pcb)和页面回收RecycleProcessPage(Memory memory,PCB pcb)函数实现该功能。

**2 系统功能与 API 函数说明**

**2.1硬件系统**

硬件系统主要包括内存、外存、CPU、MMU，并通过Memory、Harddisk、CPU、MMU四个类对其进行抽象仿真。其中对硬件的具体仿真实现请查看第三部分硬件仿真设计。

**2.1.1内存Memory**

仿真内存单元。

（1）Memory()

功能：初始化内存单元。

**2.1.2外存Harddisk**

仿真计算机硬盘。

（1）Harddisk()

功能：初始化磁盘。

（2）void InBackupQueue(ClockInterrupt c)

功能：将作业加入后备队列。

（3）void SwapArea\_Visit(int j)

功能：修改对换区页面访问次数。

（4）boolean SwapArea\_Full()

功能：检查磁盘对换区是否已满，返回值返回对换区是否已满。

（5）void IN\_SwapArea(int j)

功能：将主存页面写入对换区。

（6）void SwapArea\_Update(int j)

功能：对换区替换算法，找到对换区中最久未访问的页面，将其写入外存，并将由内存换出的页面移入对换区。

（7）boolean SwapArea\_Hit(int j)

功能：对换区页面是否命中，返回值返回对换区是否已命中。

（8）void Submit\_Job()

功能：初始化Jcb，并将作业存入外存。

（9）void ReadJobs(String Filename,int array[])

功能：作业从文件读入数组。

**2.1.3CPU**

CPU的功能分为现场保护和现场恢复。当指令类型为4、5时，需要进行关中断，即需要现场保护和现场恢复。

（1）void protect(PCB pcb\_cpu)

功能：现场保护，把当前进程运行的信息保存在pcb里。

（2）void recover(PCB pcb\_cpu)

功能：现场恢复，把pcb的信息保存在当前运行的进程里。

**2.1.4MMU**

（1）LogicAddress LAddress\_Divide(int LAddress)

功能：将逻辑地址分解为页号和偏移地址。

（2）Boolean TLB\_Full()

功能：检查快表是否已满，返回值返回快表是否已满。

（3）void TLB\_Replace(int p,PCB pcb)

功能：快表已满，选择最长时间未被访问的页面进行替换。

（4）int TLB\_HIT(LogicAddress logicaddress,PCB pcb)

功能：查看快表是否已满，返回值返回快表下标。

（5）int Logic\_To\_Physical(LogicAddress logicaddress,PCB pcb)

功能：将逻辑地址转换为物理地址，并更新快表，返回值返回指令物理地址。

**2.2作业管理**

作业管理部分主要是生成作业、随时生成作业、高级调度、低级调度，并将四部分分别抽象为CreateJobs类、ExtraJobs类、JobsScheduling类、ProcessScheduling类

**2.2.1CreateJobs**

为了保证仿真操作系统的随机性，需要随机生成作业序列。

（1）void CJobIns()

功能：随机生成5个备选作业，并作业信息以及相应的指令信息写入文件。

**2.2.2ExtraJobs**

功能：在已有的作业基础上，随机添加额外作业。

（1）void CExtraJobIns()

功能：随时生成作业，并将信息写入相应文件。

**2.2.3JobsScheduling**

采用线程类，发生5次时钟中断检测是否有新作业请求，采用按作业进入后备队列的先后次序来分配处理机（为其创建进程）。

（1）public void run()

功能：高级调度

**2.2.4ProcessScheduling**

低级调度实现采用时间片轮转调度算法和静态优先级调度算法结合。

（1）SystemRun(Memory memory,Harddisk harddisk,CPU cpu,int time,MMU mmu,ClockInterrupt c)

功能：低级调度，实现进程调度。

（2）进程原语部分：

CreatePrimitive(Memory memory,JCB jcb,Harddisk harddisk)

功能：进程创建，新建态到就绪态。

RevokePrimitive(Memory memory)

功能：进程撤销，运行态到终止态。

BlockingPrimitive(Memory memory)

功能：进程阻塞，运行态到阻塞态。

WakeupPrimitive(Memory memory)

功能：进程唤醒，阻塞态到就绪态。

（3）进程状态转换部分：

public void Ready\_To\_Running(Memory memory,CPU cpu,ClockInterrupt c)

功能：就绪态到运行态。

public void Running\_To\_Ready(Memory memory,CPU cpu,ClockInterrupt c)

功能：运行态到就绪态。

public void CheckFinishedProcess(PCB pcb,Memory memory,CPU cpu,ClockInterrupt c,Harddisk harddisk)

功能：检查指令是否执行完，即进程是否结束，运行态到终止态，阻塞态到终止态。

**2.3存储管理**

存储管理部分主要是虚存管理，主要通过PageScheduling类中一些方法实现。

1. boolean AllocatedPage(Memory memory,PCB pcb)

功能：为进程分配内存页面，返回值返回是否分配成功。

1. void LRU\_Visit(int j)

功能：修改对页面的访问次数。

1. void RecycleProcessPage(Memory memory,PCB pcb)

功能：回收进程页面。

1. int Page\_Hit(LogicAddress address)

功能：访问内存页面是否命中，返回值返回命中页面对应页表下标。

1. void GetInstruc(PCB pcb,MMU mmu,int Instruc\_ID,Memory memory,Harddisk harddisk,CPU cpu)

功能：根据进程低级调度传来的指令地址取指令。

1. void Ptabel（PCB p）

功能：生成页表并页表变化过程写入文件。

**2.4可视化**

最终将结果和运行过程通过可视化界面呈现，即实现线程类GUI，随过程变化实时更新界面。

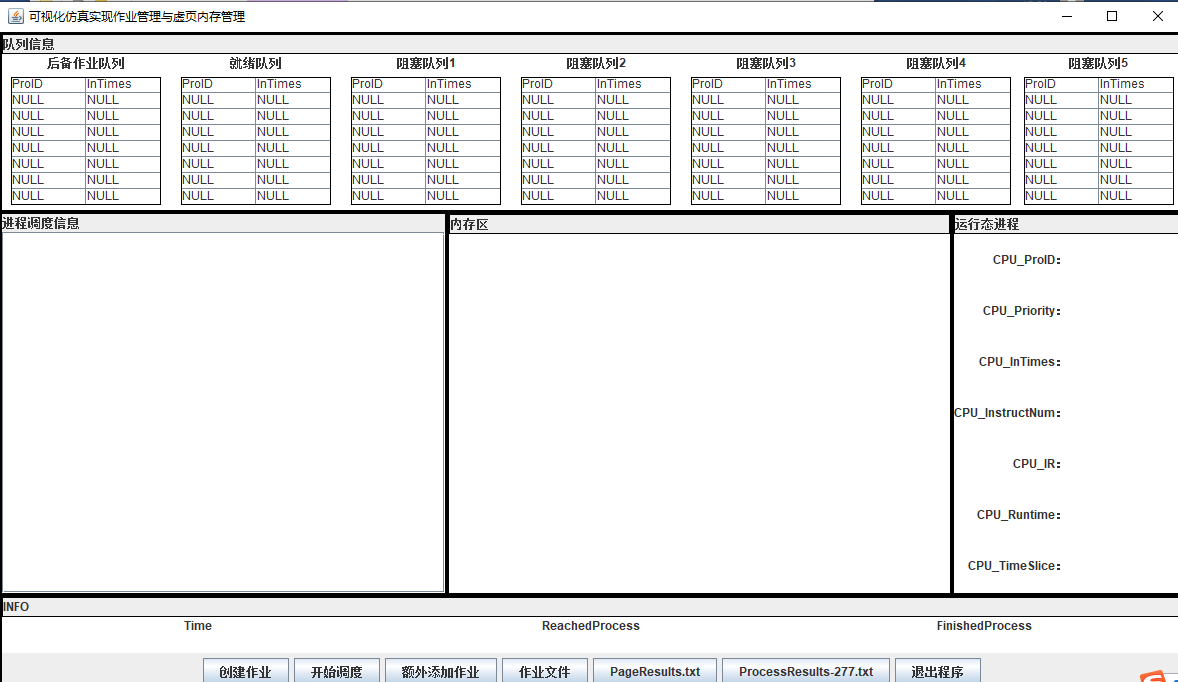


图2.1

**3 硬件仿真设计**

**3.1内存**

内存共32KB，每个物理块大小1024B，即总共32个物理块，每个物理块假设为1页帧。内存中设置了就绪队列、阻塞队列1、2、3、4、5、运行队列以及PCB表。以下是我的内存分配：

|  |  |  |
| --- | --- | --- |
| 页号 | 0-23 | 24-31 |
| 分配区域 | 用户区 | 系统区 |
| 存放项目 | 用户进程映像 | 单缓冲区 |

表3.1

**3.2外存**

外存中，10个柱面，1个柱面中有32个磁道，1个磁道中有64个扇区。可以假设1个扇区为1个物理块，每个物理块大小1024B。在外存中设置了后备队列。由于一个柱面足够大，所以我将都放在0号柱面，共2^11个物理块，磁盘交换区大小每个用户进程 3 页（块），即24块。以下是我的外存分配：

|  |  |  |
| --- | --- | --- |
| 磁道号 | 0-30 | 31 |
| 分配区域 | 文件区 | 交换区 |
| 存放项目 | 作业指令信息 | 内存换出进程 |

表3.2

**3.3MMU**

MMU位于CPU内部，可以假想为一个进程的所需要的资源都放在虚拟地址空间里面，而CPU在取指令时，机器指令中的地址码部分为虚拟地址（线性地址），需要经过MMU转换成为内存地址，才能进行取指令。MMU实现了包括从逻辑地址到虚拟地址（线性地址）再到内存地址的变换过程。

**4 数据结构与基础操作的抽象与设计**

**4.1通用数据结构**

**4.1.1指令Instruction**

系统指令类型有多种，由操作系统设计要求，我们设计指令Instruction类如下：

class Instruction {

int Instruc\_ID;//指令段编号

int Instruc\_State;//指令类型

int L\_Address;//指令访问的逻辑地址

int InRunTimes;//指令运行时间

（1）指令类型：0-正常调度1s 4-磁盘文件读操作

1-正常调度2s 5-磁盘文件写操作

2-键盘输入操作 6-打印操作

3-屏幕显示输出操作

（2）时钟计算：每1秒发生一次时钟中断。

（3）物理地址为 16 位，数据存储的单位为双字节。

**4.1.2作业JCB**

当有新作业请求，生成相应的JCB，按照先来先服务的作业调度算法将作业调入内存。系统需要随机生成作业，设计作业JCB类如下：

class JCB {

int ProID;//作业序号

int Priority;//作业优先级

int InTimes;//作业请求时间

int InstrucNum;//作业包含的程序指令数目

int HardDiskAddress[]=new int[2];//磁道扇区

public Instruction Instruc[]=new Instruction[30];暂定指令数30

}

**4.1.3进程PCB**

PCB(process control block)，进程控制块，是我们学习操作系统后遇到的第一个数据结构描述，它是对系统的进程进行管理的重要依据，和进程管理相关的操作无一不用到PCB中的内容。将进程抽象为PCB类，一个进程最大占3个页面，用户进程最高并发度为 8。PCB包含的成员如下：

public class PCB {

int ProID;//进程编号

int Priority;//进程优先数 优先数越小 优先级越大

int InTimes;//进程创建时间

int EndTimes;//进程结束时间

int PSW;//进程状态 //0新建态,1就绪态,2运行态,3阻塞态 4终止态

int RunTimes;//进程运行时间列表

int TurnTimes;//进程周转时间统计

int InstrucNum;//进程包含的指令数目

int PC;//下一条指令的编号

int IR;//正在执行的指令编号

int mem;

int HardDiskAddress[][]=new int[2][];//外存地址

Instruction Instruc[]=new Instruction[128];//一页最多存128条指令

PageScheduling page;//进程的页面管理

}

其中，PCB指令数目用[20-30]以内的随机整数产生；PCB的PSW：0新建态,1就绪态,2运行态,3阻塞态 4终止态来表示进程在运行过程中的状态。系统一个进程最大为3个页面。系统PCB表采用链表的方式实现。

**4.1.4快表项TLB**

（1）快表是一种特殊的高速缓冲存储器（Cache），内容是页表中的一部分或全部内容。在操作系统中引入快表是为了加快地址映射速度。在虚拟页式存储管理中设置了快表，作为当前进程页表的Cache。通常快表处于MMU中。

（2）快表中地址转换流程如下：

按照逻辑地址中的页号查快表；

若该页已存在快表中，则由页号和单元号形成绝对地址；

若该页不在快表中，则再查主存页表，与单元号形成绝对地址，同时将该页登记到快表中；

当快表填满后，又要登记新页时，则需要按照一定替换策略淘汰一个旧的登记项。

public class TLB {

int Pagenum;//页号

int Blocknum;//物理块号

int State;//状态

int Visit;//访问字段，用于记录本页最近访问次数

}

**4.1.5****页表项Page**

每一个[进程](https://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B/382503" \t "https://baike.baidu.com/item/_blank)都拥有一个自己的页表。页表结构如下：



图4.1

public class Page {

//页表项，共4+5+3+11=23位，需要16+16位，即4B

public int Pagenum;//页号，10-20的数，4位

public int Blocknum;//物理块号24，5位

public int State;//状态位，用于指示是否调入内存,0,未调入，1已调入，1位

public int Visit;//访问字段，用于记录本页最近已有多长时间未被访问，1位

public int Modify;//修改位，标识该页在调入内存后是否被修改，1位

public int Address[]=new int[2];//外存地址，用于指出该页在外存上的地址，供调入该页时参考，5+6=11位

**4.1.6****逻辑地址LogicAddress**

public class LogicAddress

{

int block;//页号

int offset;//页内位移

}

**4.2.基础操作**

**4.2.1原语**

计算机进程的控制通常由原语完成。所谓原语，一般是指由若干条指令组成的程序段，用来实现某个特定功能，在执行过程中不可被中断。在操作系统中，某些被进程调用的操作，如队列操作、对信号量的操作、检查启动外设操作等，一旦开始执行，就不能被中断，否则就会出现操作错误，造成系统混乱。所以，这些操作都要用原语来实现，原语是操作系统核心（不是由进程，而是由一组程序模块组成）的一个组成部分，并且常驻内存，通常在管态下执行。原语一旦开始执行，就要连续执行完，不允许中断。以下为进程控制的几个基本原语。

1. 进程创建原语

public void CreatePrimitive(Memory memory,JCB jcb,Harddisk harddisk)

{

//初始化PCB内成员的值

//分配内存

//修改进程状态，加入就绪队列

}

1. 进程唤醒

public void WakeupPrimitive(Memory memory)

{

//阻塞队列非空

//出阻塞队列

//进就绪队列

//阻塞队列为空

//输出阻塞队列为空

}

1. 进程阻塞

public void BlockingPrimitive(Memory memory)

{

//运行队列非空

//出运行队列

//进阻塞队列

//运行队列为空

//输出运行队列为空

}

1. 进程撤销

public void RevokePrimitive(Memory memory)

{

//出运行队列，进程撤销

//从PCB表删除

}

**4.2.2****地址变换**

请求分页系统中的地址变换机构，是在分页系统地址变换机构的基础上，再为实现虚拟存储器而添加了某些功能而形成的，如产生和处理缺页中断，以及从内存中换出一页的功能等等。下图4.2.3表示出了请求分页系统中的地址变换过程。在进行地址变换时，首先去检索快表，试图从中找出所要访问的页。若找到，便改动页表项中的访问位，然后利用页表项中给出的物理块号和页内地址形成物理地址。地址变换过程到此结束。

void GetInstruc(PCB pcb,MMU mmu,int Instruc\_ID,Memory memory,Harddisk harddisk,CPU cpu) {

//获取逻辑地址

//逻辑地址分解为页号和页内位移

//将逻辑地址转换为物理地址，此功能由MMU实现

//返回值为-1：产生越界中断

//返回值为-2：产生缺页中断

//其余快表或页表命中，成功取指令

}



图4.2

**4.2.3页面分配与回收**

页面分配与回收是虚页存储管理的一部分，当进程创建时，需要给进程分配页框，仿真的存储管理算法是请求分页式系统，页面分配时，采取固定分配局部置换，每个进程分配3个物理块；当进程结束时，则需要回收页面。这两部分主要通过PageScheduling类中public boolean AllocatedPage(Memory memory,PCB pcb)和public void RecycleProcessPage(Memory memory,PCB pcb)实现。

（1）页面分配，固定分配局部置换，每个进程分配3 个物理块

public boolean AllocatedPage(Memory memory,PCB pcb)

{

//内存空间足够大，则为进程分配内存

//当内存物理块未被占用，修改页表

//否则，输出内存不够，分配失败

}

（2）页面回收

public void RecycleProcessPage(Memory memory,PCB pcb)

{

//输出进程已执行完毕，回收页面，释放内存

//回收页表

}

**5 程序结构及模块功能的实现**

**5.1程序结构**

本次操作系统的程序主要分为两部分，第一部分是创建作业，第二部分是仿真实现操作系统作业管理与虚页内存管理，以硬件结构为基础，层层向上扩展，此次的操作系统课程设计程序结构通过下图来表示：



图5.1

**5.2模块功能**

**5.2.1作业创建**

本次操作系统课程设计需要随机生成组序列与相应的指令序列。

|  |  |
| --- | --- |
| 类名 | 功能 |
| CreateJobs | 随机创建5个作业，并将作业信息写入19219124-jobs-input.txt，将其指令信息写入1.txt、2.txt、3.txt、4.txt、5.txt |
| ExtraJobs | 操作系统具有不确定性，在系统运行过程中可以额外创建作业 |

表5.1

**5.2.1硬件系统**

操作系统最基础的模块就是硬件系统，硬件系统主要包括CPU、MMU、内存和外存的仿真实现。这四部分的主要函数在报告第二部分系统功能与 API 函数说明中已经列出。

|  |  |
| --- | --- |
| 类名 | 功能 |
| CPU | CPU的功能分为现场保护和现场恢复。当指令类型为4、5时，需要进行关中断，即需要现场保护和现场恢复。 |
| MMU | MMU实际存在于CPU内部，但将MMU抽象为一个单独的类，主要实现逻辑地址到物理地址的转化，同时将快表更新这一部分也设置在MMU中 |
| Memory | Memory类主要仿真实现内存，内存共32KB，每个物理块大小1024B，即总共32个物理块，每个物理块假设为1页帧。内存中设置了就绪队列、阻塞队列1、2、3、4、5、运行队列以及PCB表 |
| Harddisk | Harddisk类主要仿真实现外存，外存中，1个柱面中有32个磁道，1个磁道中有64个扇区。可以假设1个扇区为1个物理块，每个物理块大小1024B。在外存中设置了后备队列。 |

表5.2

**5.2.2作业管理**

作业管理部分主要是将作业提交到外存、高级调度、低级调度，分别抽象为

Submit\_Job()函数、Submit\_ExtraJob()函数、JobsScheduling类、ProcessScheduling类，将作业保存到磁盘上，然后在时钟中断类ClockInterrupt的控制下，讲作业调入内存，为作业创建进程，并依次执行程序的指令。

|  |  |
| --- | --- |
| Submit\_Job() | 该方法设置在Harddisk类中，当开始调度时，自动通过Submit\_Job()函数生成JCB并送入外存 |
| Submit\_ExtraJob() | 该方法同样设置在Harddisk类中，为运行过程中随时提交作业，实现方法与Submit\_Job()函数类似，他将随时添加的作业信息从19219124-jobs-input.txt读入，指令信息从6.txt、7.txt...初始化JCB，送入外存 |
| JobsScheduling类 | 高级调度，设置为一个线程，检测后备作业队列是否有新作业生成，采用先来先服务算法，并把它调入内存就绪队列 |
| ProcessScheduling类 | 低级调度，当时间片到了或是进程阻塞让出CPU，就从就绪队列里挑出优先级最高的进程到运行队列。 |

表5.3

5.2.2.1创建JCB

由于Submit\_ExtraJob()函数与Submit\_Job()函数实现方法类似，此部分只描述Submit\_Job()部分。

void Sunmit\_Job()

{

//从文件读入作业信息

//创建JCB

//写入磁盘

//从文件读入指令信息

//修改JCB的指令信息

}

5.2.2.2低级调度

无论是批处理系统、分时系统还是实时系统，用户进程数目一般都多余处理机数，这将直接导致用户进程相互争夺处理机。另外，系统进程也同样需要使用处理机。进程调度虽然是系统内部的低级调度，但进程调度算法的优劣直接影响作业调度的性能,因此一个好的调度策略对处理机的处理速度是至关重要的。此设计主要提供了时间片轮转法和静态优先级法结合的算法。

系统设置时间片为2s，当时间片到了或是进程阻塞让出CPU，会从就绪队列里挑出优先级最高的进程到运行队列。

低级调度主要是进程状态转换，这次操作系统课程设计中，每个进程分得的时间片为2s，完成了进程状态的转换，主要通过该类中函数public int SystemRun(Memory memory,Harddisk harddisk,CPU cpu,int time,MMU mmu,ClockInterrupt c)中体现出来。

public int SystemRun(Memory memory,Harddisk harddisk,CPU cpu,int time,MMU mmu,ClockInterrupt c)

{

//cpu非空闲

//取指令执行指令

//检查进程是否结束

//cpu空闲

//按照优先级，将优先级高的进程从就绪队列->运行队列

//时间片结束，运行->就绪

//检查是否所有进程全部执行完毕

}

**5.3.3存储管理**

存储管理部分主要是虚存管理，本次课程设计主要采取的管理方式是请求分页式存储管理，请求分页系统是建立在基本分页系统的基础上，为了能支持虚拟存储器功能而添加了请求调页功能和页面置换功能。在请求分页系统中所须要的主要数据结构是页表。其基本作用仍然是将用户地址空间中的逻辑地址变换为内存空间中的物理地址。因为仅仅将应用程序的一部分调入内存，另一部分仍在盘上，故须在页表中再添加若干项，供程序(数据)在换进、换出时参考。在请求分页系统中的每个页表项例如图5.3.3所看到的：



图5.2

以下主要通过PageScheduling类中一些方法，结合CPU中缺页中断处理程序实现。这部分主要通过描述PageScheduling来实现虚页存储管理。

|  |  |
| --- | --- |
| boolean AllocatedPage(Memory memory,PCB pcb) | 页面分配功能，固定分配局部置换，请求分页系统中，为每个进程分配3个物理块 |
| void RecycleProcessPage(Memory memory,PCB pcb) | 当进程结束的时候，需要回收页面，清空页表等 |
| void Ptabel(PCB p) | 生成页表，将页表变化信息写入文件ProcessResults.txt |
| void GetInstruc(PCB pcb,MMU mmu,int Instruc\_ID,Memory memory,Harddisk harddisk,CPU cpu) | 进程执行指令的时候，会传递逻辑地址，GetInstrc函数实现取指令的功能 |

表5.4

**6 测试与分析**

**6.1RunProcess测试。**

**6.1.1测试用例**

给定的测试用例：图6.1-6.6

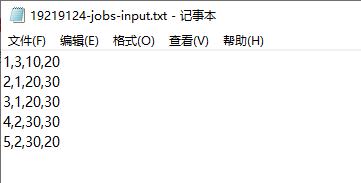


图6.1

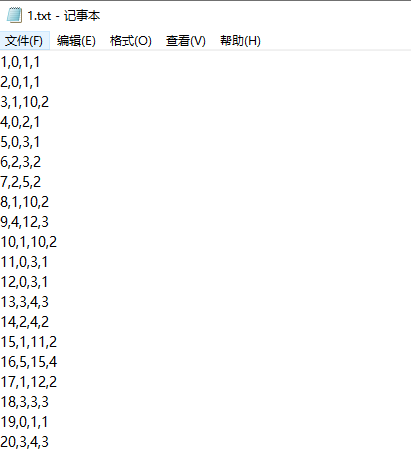


图6.2

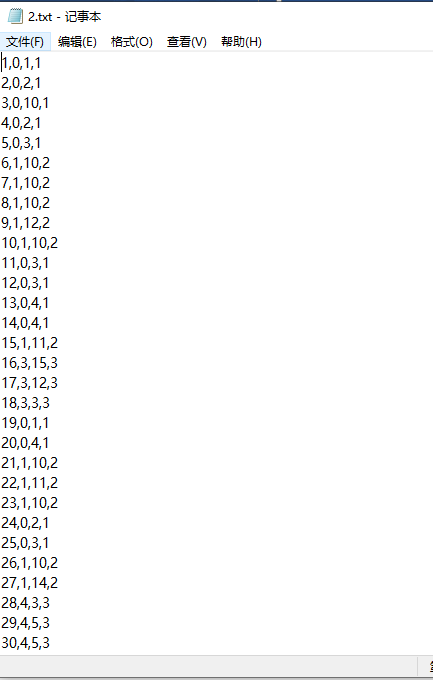


图6.3

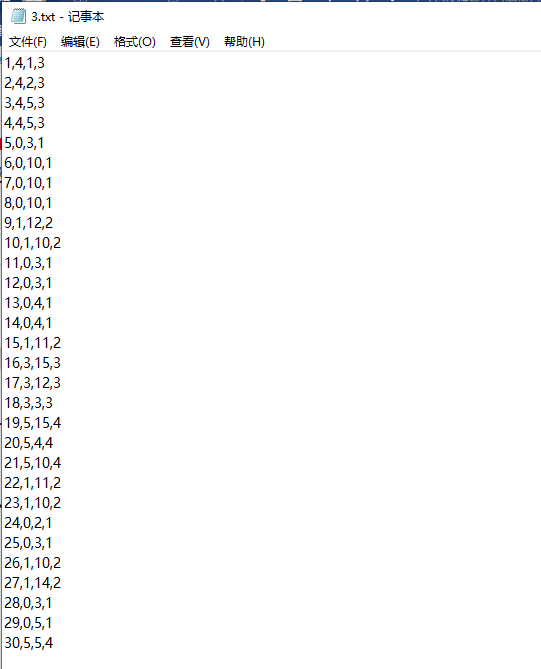


图6.4

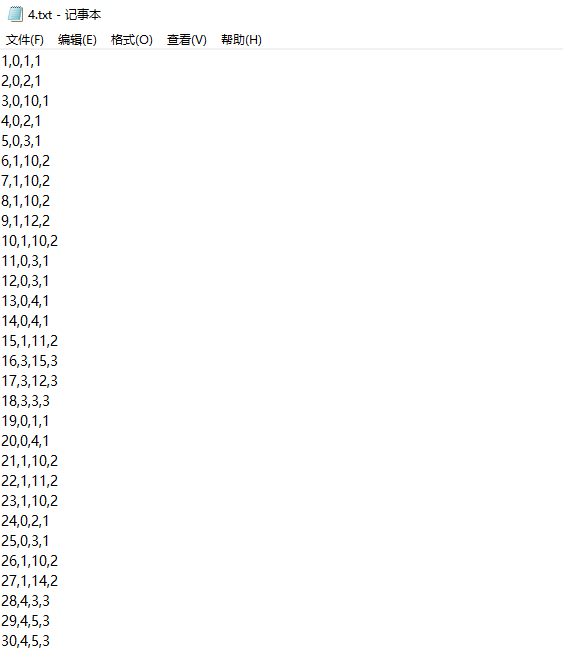


图6.5

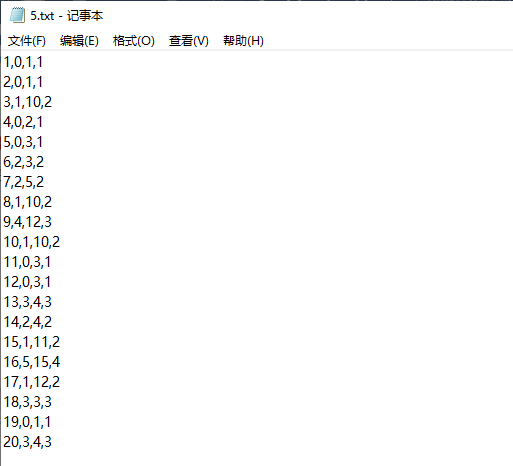


图6.6

**6.1.2界面介绍**

界面共分六部分：队列信息、进程调度信息、内存信息、运行态进程、系统信息和按钮功能区。

（1）队列信息部分显示了当时就绪队列以及五个阻塞队列内的进程信息如图6.7。

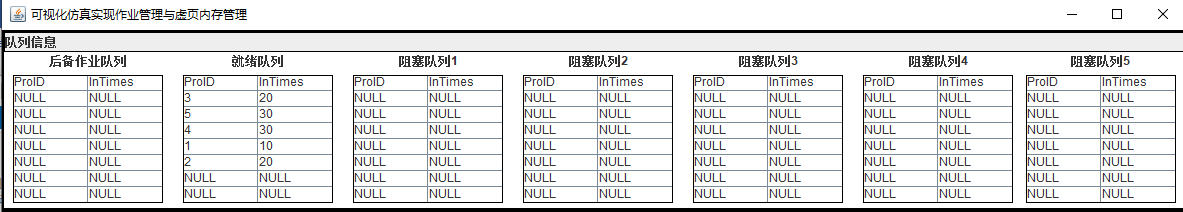


图6.7

（2）进程调度信息部分显示了每一秒发生作业或进程调度事件、缓冲区处理事件、输入/输出井事件以及就绪、阻塞队列等的状态信息如图6.8。

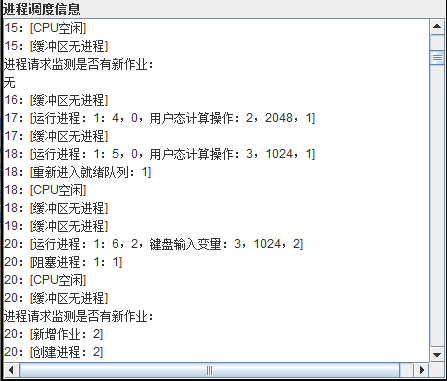


图6.8

（3）内存信息显示了当前内存块的存储情况，用位示图来表示，如图6.9。

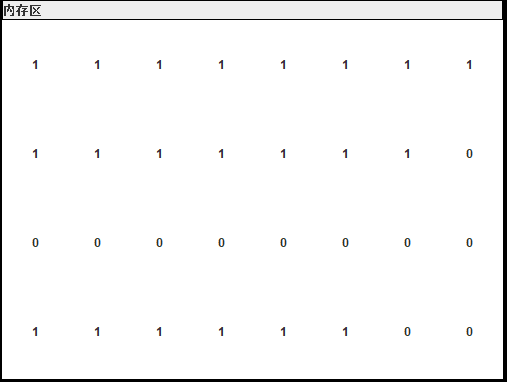


图6.9

（4）运行态进程显示了当前CPU运行的进程信息，如图6.10。

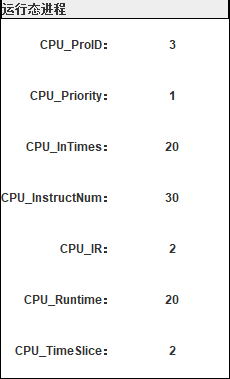


图6.10

（5）系统信息显示了时间、到达进程数和完成进程数，如图6.11。



图6.11

（6）按钮功能区可以提供创建作业、开始调度、额外添加作业、作业文件、查看页面结果信息、查看进程调度信息和退出程序的功能，如图6.12。



图6.12

**6.1.3运行步骤**

1、点击运行test文件夹下19219124-郭晗.exe，即可出现图6.13界面。确保当前目录下19219124-jobs-input.txt以及1-5.txt。若没有，手动创建。

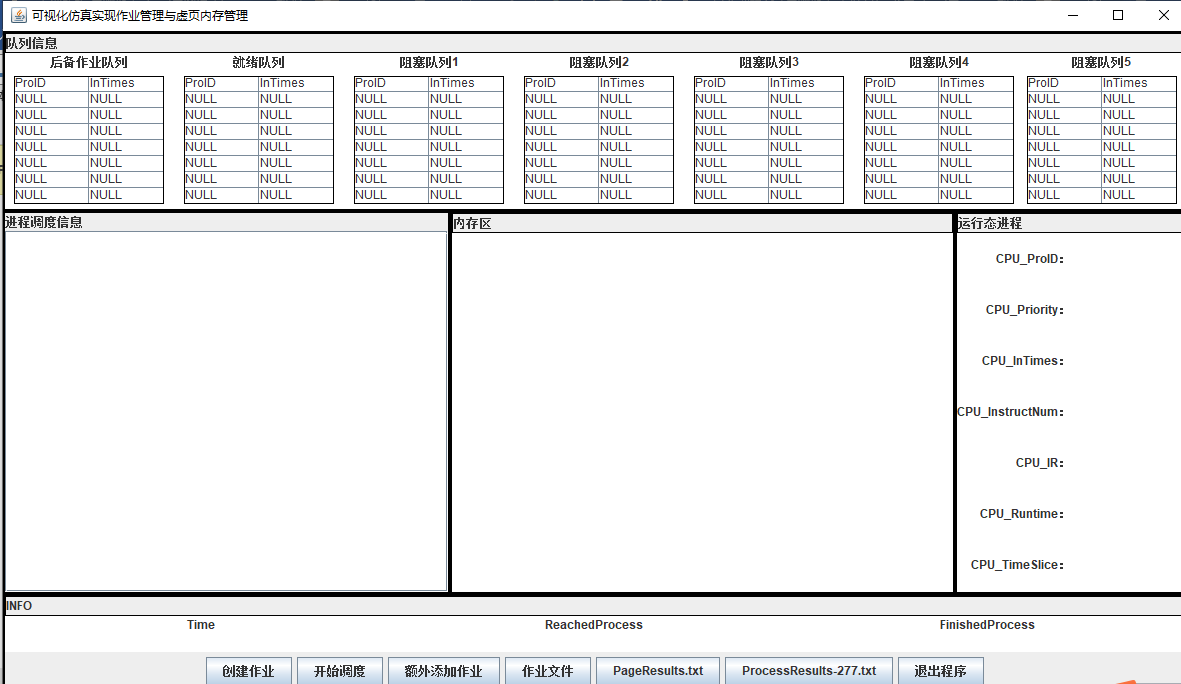


图6.13

2、点击开始调度按钮后界面如图6.14：

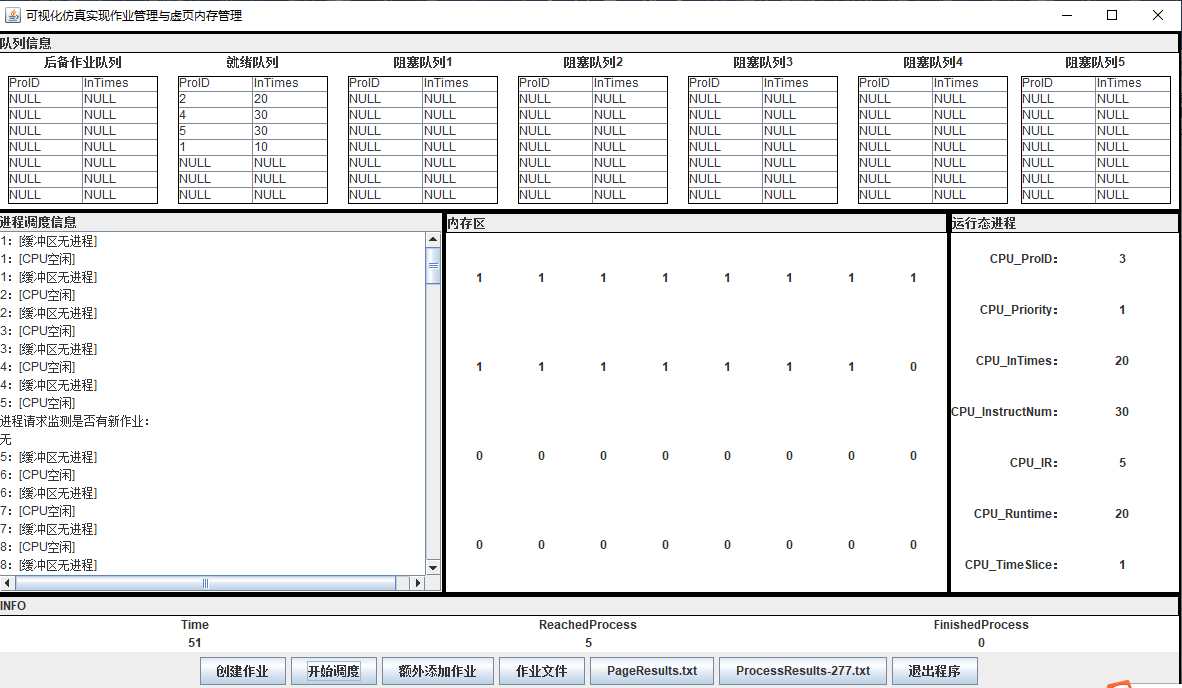


图6.14

3、运行结束后可观察生成文件PageResults.txt和ProcessResults-277.txt（图6.15），也可通过点击界面上PageResults.txt和ProcessResults-277.txt两个按钮来查看相应内容，其中PageResults.txt（图6.16）文件存储页表变化和取指令（地址变换）的过程，ProcessResults-277.txt文件存储作业调度和进程调度信息、页面分配与回收等问题。

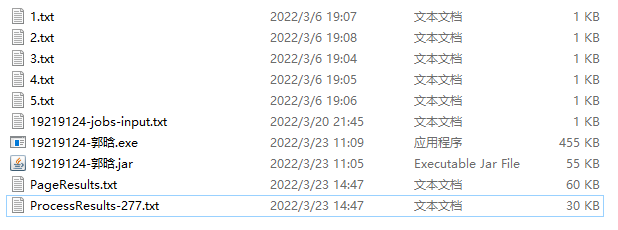


图6.15



图6.16

**6.2CreateJobs和ExtraJobs测试。**

**6.2.1生成作业**

测试：运行CreateJobs包内的CreateJobs.java文件

结果分析：可查看生成以下文件，其中作业信息和指令信息也在图6.17-6.19给出。

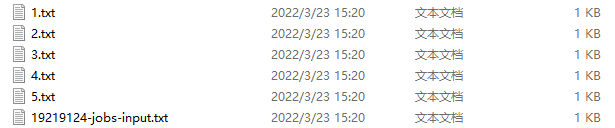


图6.17

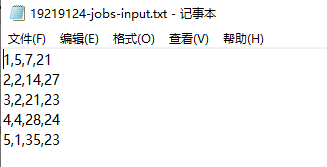


图6.18

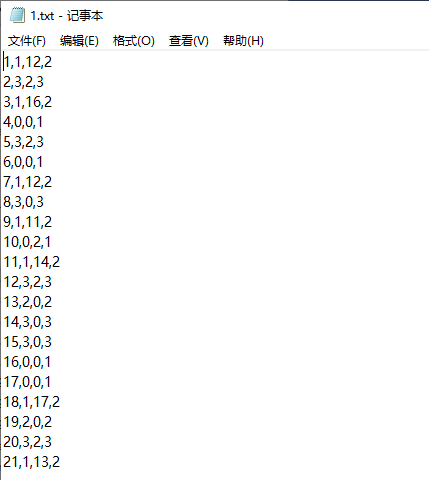


图6.19

**6.2.2额外生成作业**

测试：运行CreateJobs包内的ExtraJobs.java文件

结果分析：可查看生成了额外的作业以及作业指令，可查看以下文件，其中作业信息和指令信息也在图6.20-6.21给出。可在原来的作业文件基础上直接增加作业信息。

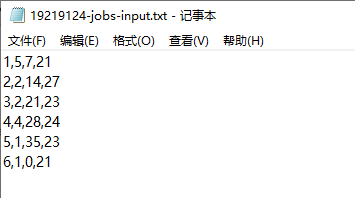


图6.20

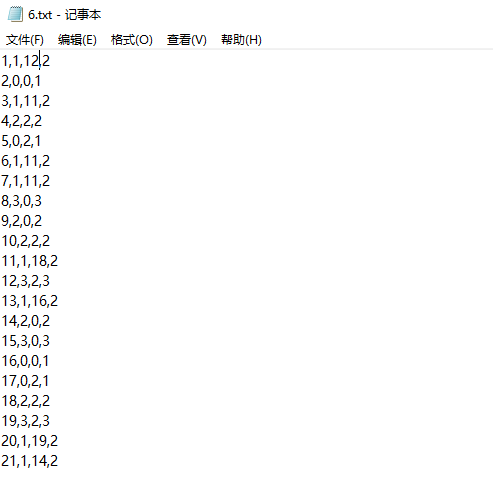


图6.21

**6.2.3开始调度**

运行步骤以及过程同6.1.1给定的测试用例。

**6.3测试完成功能**

**6.3.1作业管理**

一、检测作业调度

采用test文件夹中test1文件夹内的数据。直接将这些测试数据复制粘贴到OS文件夹内即可运行，运行时不需要创建作业，直接点击开始调度。

作业调度采用先来先服务算法，为每5s检测一次后备队列，若没有新作业，则继续，又有新作业生成，则为作业创建进程，分配空间并加入就绪队列。

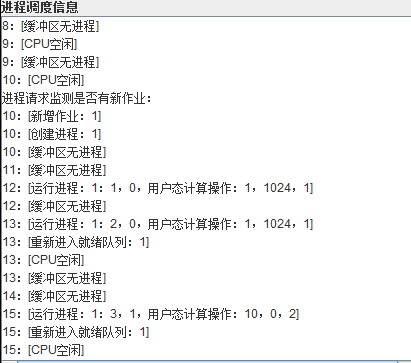


图6.22

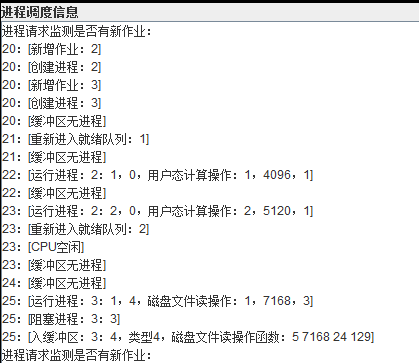


图6.23

（1）测试结果图6.22结果分析:

第1个作业到达时间为第10s，因此第5s检测时没有新作业请求，但在第10s检测到第一个作业请求，此时作业1出后备队列，新增作业1、创建进程1。

（2）测试结果图6.23结果分析：

第2、3个作业到达时间均为20s，则可在第20s同时检测到两个新作业请求，并同时为其创建进程。

二、测试随时添加额外的作业

采用test文件夹中test1文件夹内的数据。直接将这些测试数据复制粘贴到OS文件夹内即可运行，运行时不需要创建作业，直接点击开始调度。随时添加作业也体现了操作系统的不确定性，在运行过程中也可点击额外添加作业按钮：



图6.24

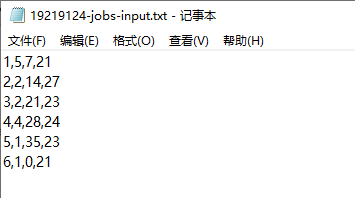


图6.25

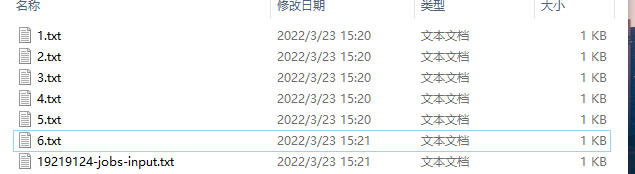


图6.26

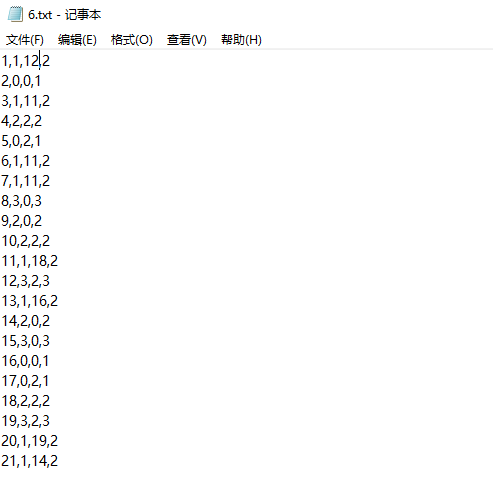


图6.27

（1）测试结果图6.24结果分析：

稍等片刻可观察界面系统信息一栏中ReachedProcess为6，相对于最初的5个作业增加了一个。也可观察到19219124-input-jobs.txt中新增了作业信息，创建了6.txt指令文件（图6.25-6.26）。这说明额外添加作业添加成功。

三、检测进程调度算法，时间片轮转法+静态优先级调度算法

采用test文件夹中test2文件夹内的数据。直接将这些测试数据复制粘贴到OS文件夹内即可运行，运行时不需要创建作业，直接点击开始调度。进程调度算法将CPU时间分为每个时间片2s，假如就绪队列有多个进程，则将优先级高的进程加入运行队列，占用CPU：

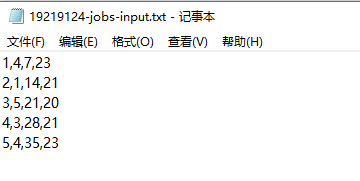


图6.28

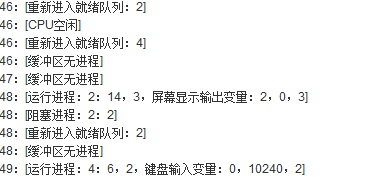


图6.29

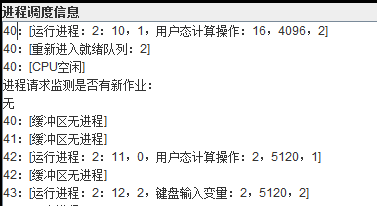


图6.30

（1）测试结果图6.28结果分析:

图6.28中进程二和进程四优先级分别为1、3，即进程二的优先级高。进程2、4同时进入就绪队列后，就绪队列里有进程二和进程四，可见进程二先占用CPU。

（2）测试结果图6.29-6.30结果分析：

图6.29中出现了进程二从40s占用CPU,到42s时间片用完，从运行队列到就绪队列（图6.30）。

**6.3.2虚页内存管理**

一、生成页表

采用test文件夹中test3文件夹内的数据。直接将这些测试数据复制粘贴到OS文件夹内即可运行，运行时不需要创建作业，直接点击开始调度。当程序运行完，才可以查看生成的两个结果文件，页表变化保存在PageResults.txt文件中：



图6.31



图6.32

（1）测试结果图6.31结果分析:

进程一第0条指令访问页面12，页面12不在内存中，发生缺页中断。

（2）测试结果图6.32结果分析：

测试LRU算法，图6.32中出现了进程二的的三个页框对应的页面分别是12、1、2，之后进程再访问页面0的时候，会发生缺页中断，此时页表已满，需要根据LRU算法进行页面置换，查看得知，页面1最长时间未被访问，则页面0替换页面1。

二、地址变换

采用test文件夹中test4文件夹内的数据。直接将这些测试数据复制粘贴到OS文件夹内即可运行，运行时不需要创建作业，直接点击开始调度。当程序运行完，才可以查看生成的两个结果文件，取指令过程保存在PageResults.txt文件中，也可查看：

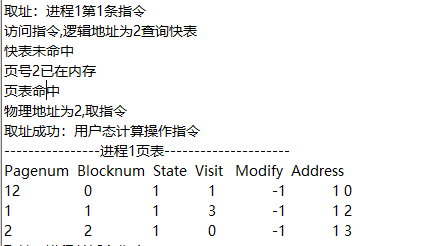


图6.33

（1）测试结果图6.33结果分析:

进程一第1条指令访问页面2，页面2的逻辑地址为2，由MMU的将逻辑地址分解为页号，可计算得出页号=2。对应的物理块号为2，则物理地址为2。即地址转换正确。

三、缺页中断与虚页调度算法

采用test文件夹中test4文件夹内的数据。直接将这些测试数据复制粘贴到OS文件夹内即可运行，运行时不需要创建作业，直接点击开始调度。当程序运行完，才可以查看生成的两个结果文件，取指过程保存在PageResults.txt文件中：



图6.34

（1）测试结果图6.34结果分析:

测试LRU算法，图6.34中出现了进程一的三个页框对应的页面分别是12、1、2，之后进程再访问进程一的时候，会发生缺页中断，此时页表已满，需要根据LRU算法进行页面置换，查看得知，页面1最长时间未被访问，则页面4替换页面1，访问位置0。

四、页面分配与回收

采用test文件夹中test4文件夹内的数据。直接将这些测试数据复制粘贴到OS文件夹内即可运行，运行时不需要创建作业，直接点击开始调度。页面分配与回收的功能可以体现在可视化界面上：

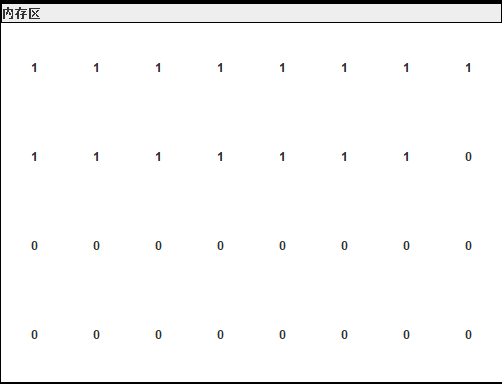


图6.35

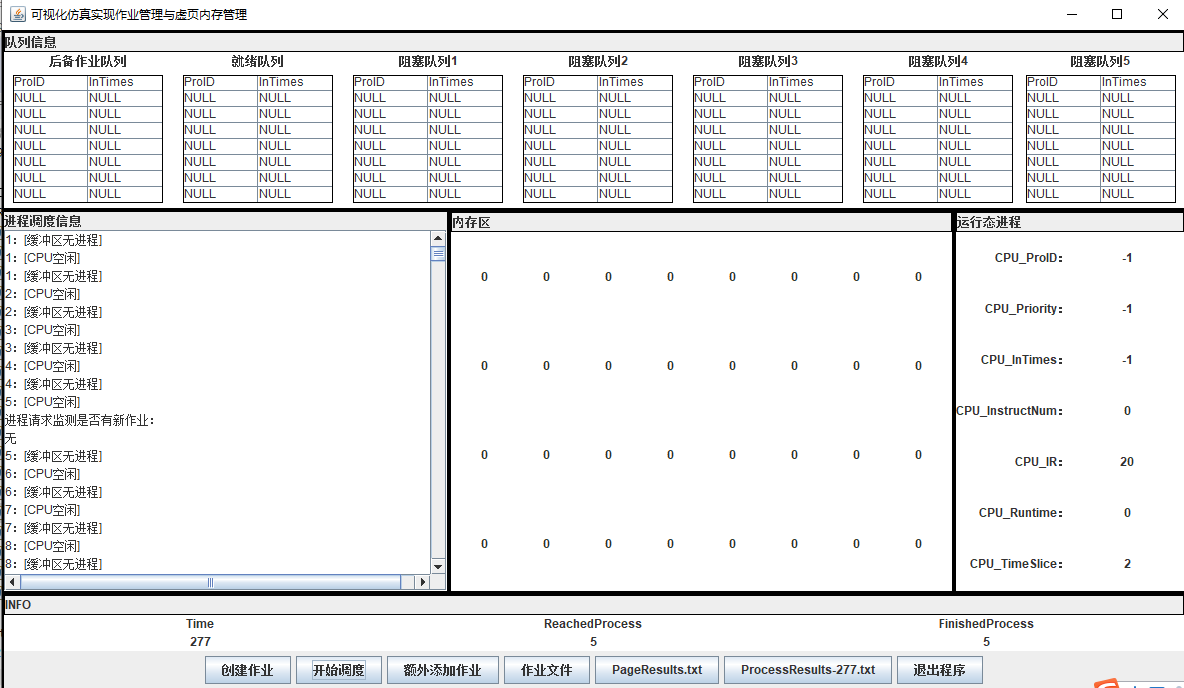


图6.36

（1）测试结果图6.35-6.36结果分析:

此测试数据共有5个作业序列，一个进程分配3个页面，最后一行8块为缓冲区，缓冲区按需分配，如例1，最终所有进程执行完毕，收回所有进程，可以看到，内存区标志位全是0，即未被占用。

**6.3.3可执行程序文件测试**

本次操作系统课设还生成了可执行文件，本程序在test文件夹提交了19219124-郭晗.jar可执行文件和19219124-郭晗.exe可执行文件，这两个是一样的，但经过多次测试，发现由于每个人eclipse版本不同，19219124-郭晗.jar在有的电脑上运行不了，因此又生成了19219124-郭晗.exe可执行程序。

在自我测试报告和视频里，为了讲解方便，源程序代码直接在本人eclipse软件中运行测试的，但最后经过测试，直接使用可执行程序文件也可以进行测试。以下举例将测试可执行文件19219124-郭晗.exe的运行，并通过可执行程序文件测试具体功能，这里以测试test4文件夹内的测试用例。

step1：找到test文件夹下的19219124-郭晗.exe，双击运行出现以下界面（图6.37）：

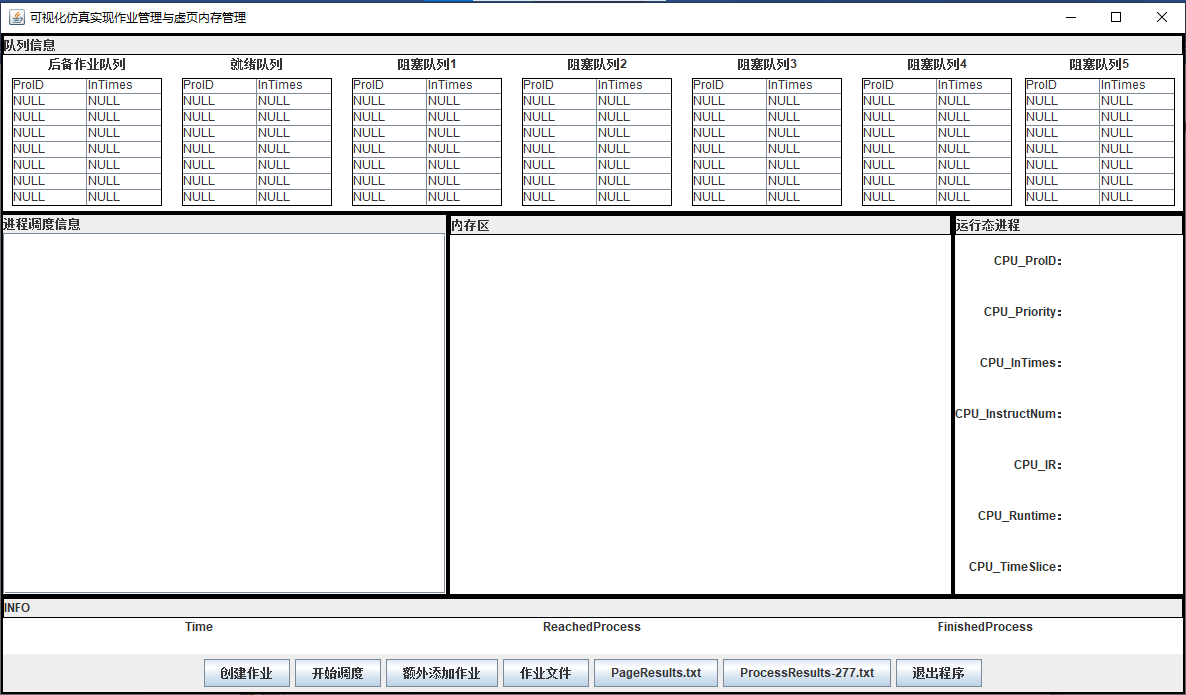


图6.37

step2：点击创建作业按钮，可观察到在本文件夹下出现创建的作业文件和指令文件（图6.38）：

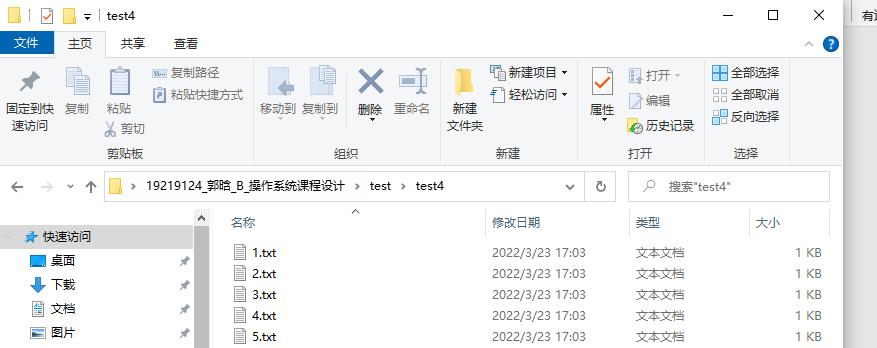


图6.38

step3：以下步骤类似于二、RunProcess测试1中根据给定的19219124-jobs-input.txt以及1-5.txt进行测试。仍可随时添加作业等，运行结果、运行结束界面如下图（图6.39-6.40）：

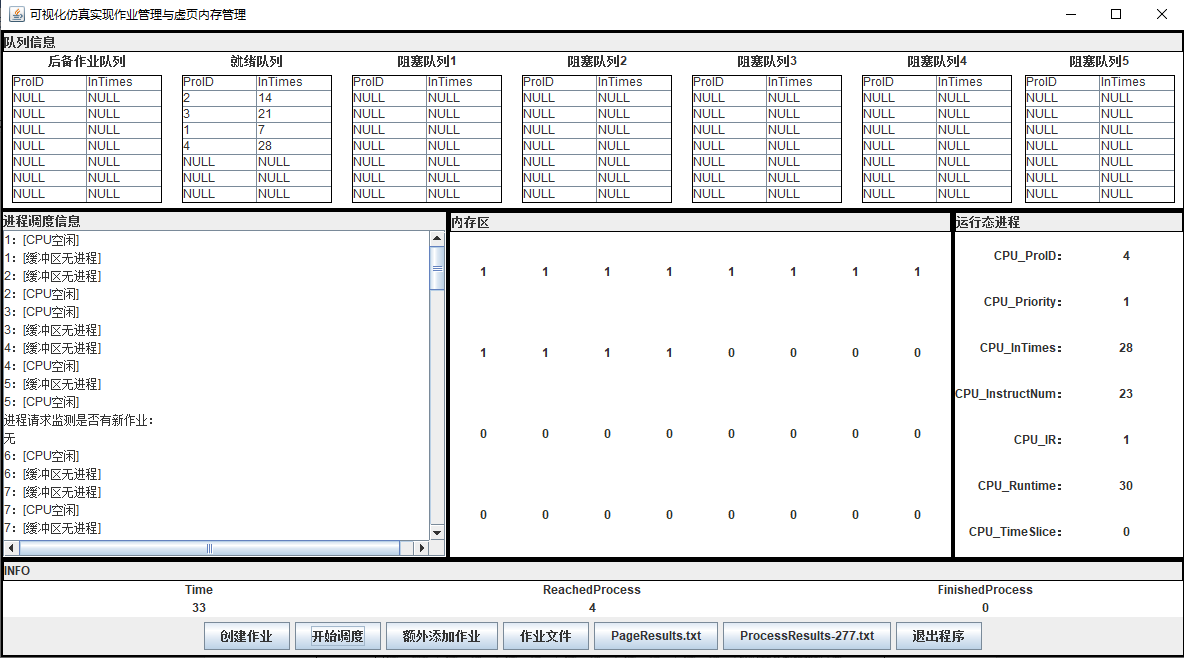


图6.39

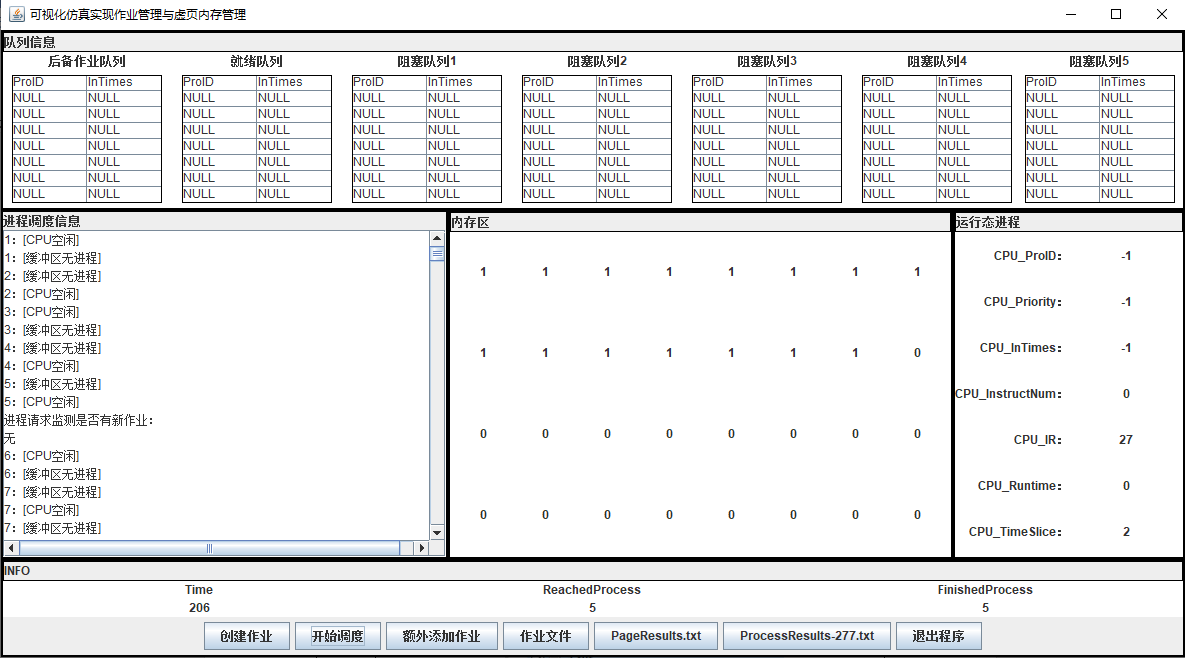


图6.40

采用test文件夹中test4文件夹内的数据。运行时不需要创建作业，直接点击开始调度。运行结果如下图（图6.41）：



图6.41

**7 技术问题及解决方案**

**7.1文件写入与读出**

（1）问题描述：以上学期操作系统第一次必修实验为基础，这次同样需要实现作业并发请求序列自动生成，即将1号进程的程序指令保存在 1.txt 文件中，2号、3号进程指令保存在2.txt、3.txt文件中等，JAVA语言文件的写入与读出方法很多，此次我定义了FileReader和BufferedReader类对象对文件进行写入与读出。如此可见，倘若每一次写入一个文件都要使用一次写入函数，那么实现写入多个名字相似的文件就要以相同的代码循环多次，造成代码冗余。

（2）解决方案：因此最终采取了定义了FileReader和BufferedReader类对象数组来对文件写入与读出，如图7.1，最终测试结果如图7.2：

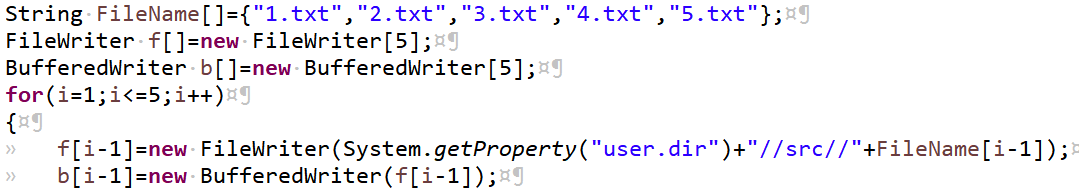


图7.1



图7.2

**7.2多线程控制**

（1）问题描述：不同于其他大多数编程语言，Java本身内置了多线程的支持。使用多线程，可以帮助我们编写出cpu最大利用率的高效程序，使得空闲时间降到最低。操作系统课设需要设置多个线程，如时钟、唤醒进程、检查后备队列等，刚开始不知道如何控制多个线程之间的关系，因为线程具有不确定性，因此运行的时候总是异常停止。

（2）解决方案：原因是线程运行太快，因此采用静态休眠方法（Sleep），实现延时效果，令每个线程主动让出CPU时间给其他线程。

**7.3Java内部队列方法**

（1）问题描述：就绪队列、运行队列、阻塞队列、PCB表等都定义为JAVA内部的Queue类，但是由于低级调度选用时间片轮转和静态优先级优先的调度算法，即从就绪队列到运行队列选用队列里某一个进程而非队头元素，但Queue类中并没有该方法；为了时刻检查各队列内元素，也需要将队列元素输出一遍，但Queue中页未有该方法。

（2）解决方案：因此需要补充Queue类的遍历以及将指定元素移除队列等方法。

**7.4缓冲区设计**

（1）问题描述：运行指令4、5 时会读文件内容到缓冲区或写文件数据到缓冲区，需要用到内存缓冲区，将内存后8块即24-31块设为内存缓冲区，作为单缓冲，需要实现“读者-写者”问题，也就是说在缓冲区没有被写满是不可以读缓冲区的，同理，缓冲区没有被读完也是不可以写的。

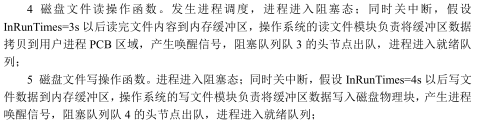


图7.3

（2）解决方案：通过控制缓冲区的大小来判断缓冲区试用情况，假设一条指令占缓冲区一块大小，如果这条指令进入缓冲区，则这个缓冲区块就满了，继续分配缓冲区从下一块开始。通过判断缓冲区此时的大小来确定缓冲区是否全满，全满需要进行清空操作才可继续运行。

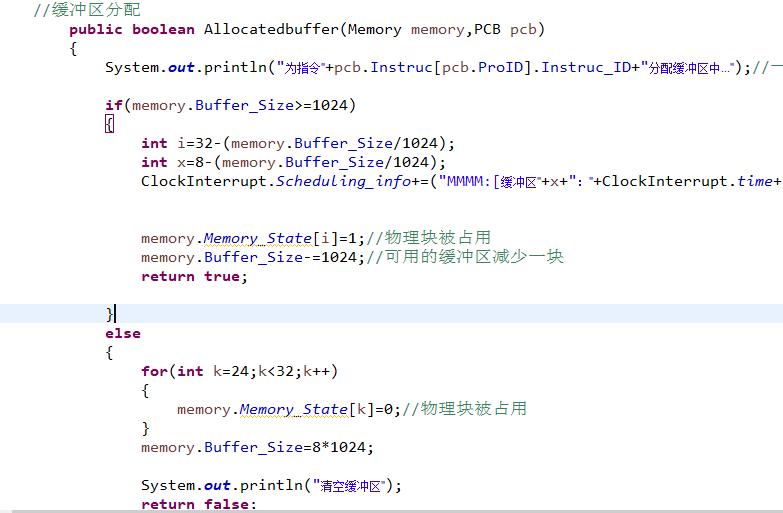


图7.4

**7.5参数问题**

（1）问题描述：操作系统课程设计中涉及到类文件比较多，因此每个类里的方法有时候需要多个类的成员，因此可能会造成重复定义类的对象，导致结果不正确的问题。

（2）解决方案：最终所有的类的对象实例都定义在ClockInterrupt时钟类里，由时钟类传递实例参数，防止类的实例对象不唯一即保证系统硬件唯一。

**7.6CPU关中断**

（1）问题描述：在遇到指令4、5的时候，进程进入阻塞态，需要进行关中断，在相对应的InRunTimes时间后进行读文件到缓冲区或写文件到缓冲区等等操作，也就是说在关中断时，CPU是非空闲状态，不能运行任何进程的。但我之前所设计的始终不能达到完全的关中断，总是会在执行完4、5指令1s后继续运行其他进程。

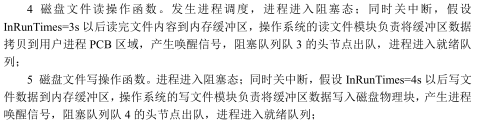


图7.5

（2）解决方案：在唤醒队列的线程里，判断阻塞队列是否空闲，然后判断阻塞时间与InRunTimes相等的情况下，将CPU的标志位置为0，在此时CPU的状态为空闲，也就是说CPU在完成关中断之后才可以去执行其它进程。解决了上述问题。

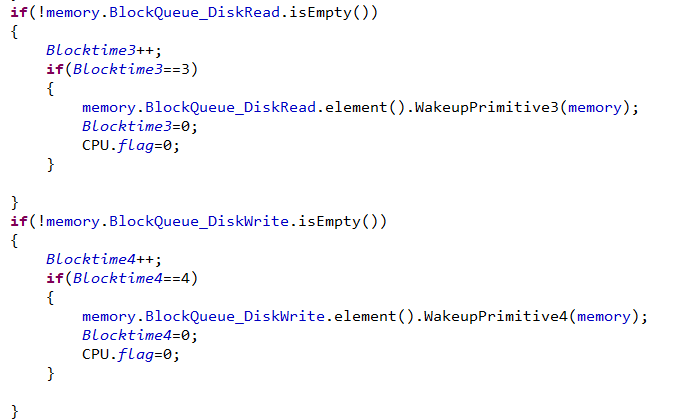


图7.6

**7.7可视化仿真界面表格**

1. 问题描述：实现可视化仿真界面时，最初以文本域显示队列信息，但系统运行太快，看不清队列元素变化。
2. 解决方案：最终采取了表格的方式来呈现一些队列，如就绪队列、阻塞队列、后备队列等，这样可以更清晰地将进程状态改变表示出来，从网上获取资料得知JTabel和TabelModel可以实现，其中JTabel为表格，可以设置行数与列数，TabelModel是表模型，存放表格相应的队列数据。前后两种方法呈现出来的效果如图7.3：

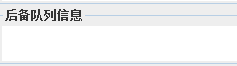
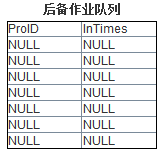
 

图7.7

**8 实践心得**

操作系统是一门理论、实践与应用都很强的课程，仅仅学习理论只是很难达到理想教学效果，而通过本次操作系统课程设计，我知道要想比较好的仿真操作系统运行过程，必须有强大的理论基础知识支撑，并且具有一定的编程能力。所以在进行编程设计之前，我又仔细学习了一遍操作系统理论知识。

对于此次课程设计，涉及到的代码量比较大，起初并没有什么思路，只能先尝试着先把生成作业序列程序和硬件仿真出来，然后通过研究课本理论知识，逐步对代码进行添加与修改。这是以上学期第一次必修实验为基础，最终将两个可执行程序集中在一个可视化界面上实现。虽然这次的课程设计以来，过程艰辛，也不止一次地想要放弃，问题不断地出现，这些问题有大有小，可能仅仅只是文件读入、也可能是多线程控制等，但正是这些困难让我明白了理论课上学习的知识并不能代表能真正地理解操作系统的内在运行过程，也帮助我提高了解决问题的能力，而此次实验也让我对操作系统如何实现它的功能的过程有了一定的深入了解。

本次课设我做了很多的无用功，都是因为我对多线程的理解不过深刻，导致我在实现大部分功能的时候发现最初的体系结构有很大的错误，必须将已经写好的代码推翻重写，浪费了很多的时间。这也是警示我以后要在看明白课设要求之后再行动，以后做课设之前还是需要巩固书本和课堂的理论知识再进行，否则只能不停返工，耗费时间和精力。

本次课设可以说是目前为止遇到过最复杂最耗时课设了，既对概念的理解提出了很高的要求，也对编程能力提出了挑战。起初毫无思路，到做完之后的思路清晰，以及将结果呈现在图形化界面以及生成了exe文件都是十分具有成就感的。

虽然我现在还没有完全实现预期的功能，在输出方面和界面设计方面还有一定的欠缺，但我会在接下来的时间内向着预期的功能努力。

**参考文献**

[1]Losileeya.JAVA线程与多线程https://blog.csdn.net/u013278099/article/details

/52317355，2016-8-25.

[2]王道论坛.王道考研.电子工业出版社,2018-04-00.

[3]费翔林，骆斌.操作系统教程[M].5 版.北京：高等教育出版社，2014-2.

[4]宁宁jiayou.TableModel 与AbstractTableModel.https://blog.csdn.net/u012343209

/article/details/18708533,2014-1-23.

[5]AnderwS.Tanenbaum.现代操作系统[M].(3版).北京:机械工业出版社,2009.

[6]操作系统概念：JAVA实现[M]. (7版).北京:高等教育出版社,2010.

**附件 1：程序文件及结构说明**

1. **程序文件结构说明**

一级目录：code

二级目录：.settings文件夹

bin文件夹

src问价

.classpath

.project

三级目录：src：CreateJobs文件夹

RunProcess文件夹

四级目录：CreateJobs：CreateJobs.java

ExtraJobs.java

Instruction.java

RubProcess: CheckWakeup

ClockInterrupt.java

Cpu.java

Disk\_Read.java

Disk\_Write.java

GUI.java

HardDisk.java

JCB.java

JobsScheduling.java

Keyboard.java

LogicAddress.java

Memory.java

MMU.java

Page\_Fault\_Interrupt.java

PageScheduling.java

PCB.java

ProcessScheduling.java

Screen.java

TLB.java

**源程序文件说明**

CreateJobs：

CreateJobs.java：创建作业

ExtraJobs.java：随时创建作业

Instruction.java：指令数据结构类

RunProcess:

CheckWakeup.java：检查唤醒时钟类

ClockInterrupt.java：时钟类

Cpu.java：仿真CPU

Disk\_Read.java：磁盘读操作类

Disk\_Write.java：磁盘写操作类

GUI.java：可视化界面

HardDisk.java：磁盘类

JCB.java：作业数据结构类

JobsScheduling.java：高级调度

Keyboard.java：键盘输入类

LogicAddress.java：逻辑地址数据结构类

Memory.java：内存

MMU.java：内存管理单元

Page\_Fault\_Interrupt.java：缺页中断类

PageScheduling.java：页面管理

PCB.java：进程数据结构类

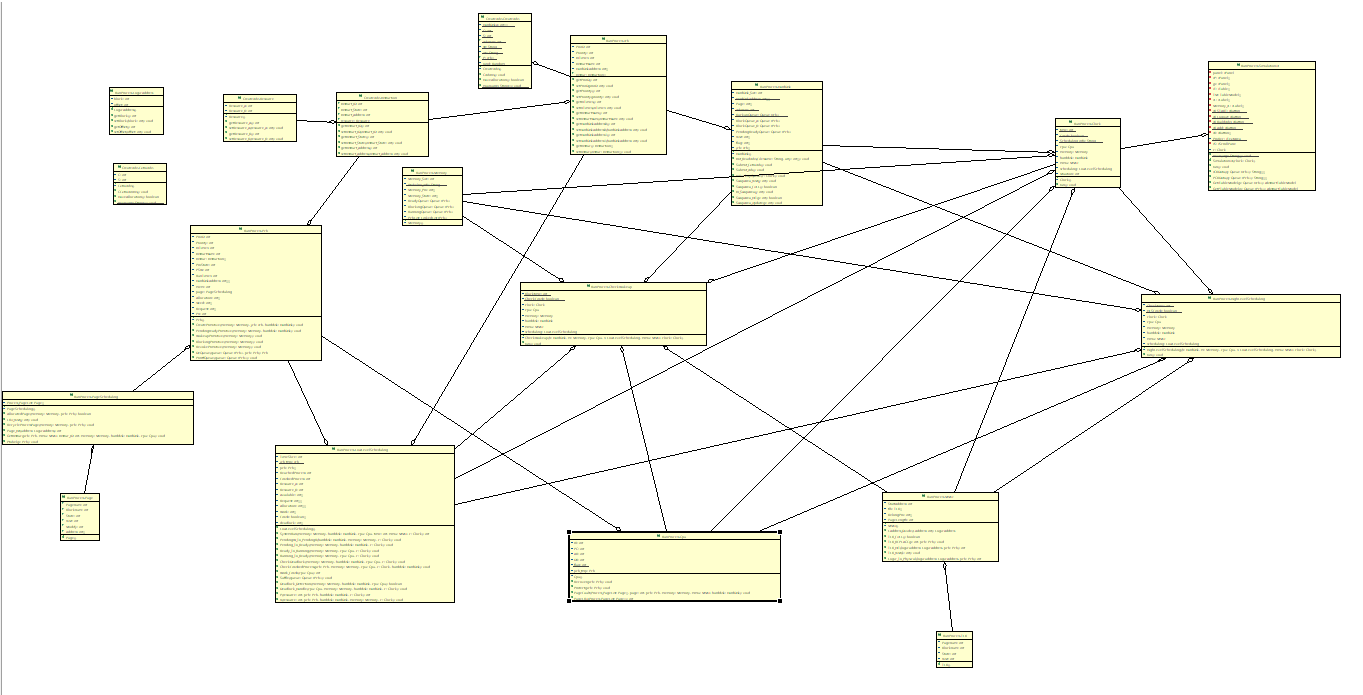
ProcessScheduling.java：低级调度

Screen.java：显示输出类

TLB.java：快表项数据结构类

**附件 2：类图和顺序图说明**

**（一）类图**



附件2-类图

**（二）顺序图**



附件2-顺序图

**附件 3：带注释的部分核心代码**

1. **Page\_Fault\_Interrupt类：**

//缺页中断处理程序

public void PageFault(Page Process\_PageList[],int page,Pcb pcb,Memory memory,MMU mmu,HardDisk harddisk)

{

//对换区查找页面

if(harddisk.SwapArea\_HIT(page))

{

Memory.PtabelIns\_info+=("从对换区调入页面"+page+"\n");

System.out.println("从对换区调入页面"+page);

SimulationUI.ProRec.append("从对换区调入页面"+page+"\n");

}

else

{

System.out.println("该页面不在对换区，从外存调入");

SimulationUI.ProRec.append("该页面不在对换区，从外存调入"+"\n");

Memory.PtabelIns\_info+=("该页面不在对换区，从外存调入"+"\n");

}

//查找被替换的页面

int Number;

Number=PageLRU(Process\_PageList);

//页面置换，修改页表

if(Process\_PageList[Number].Modify==1)//该页面被修改,则需要送入兑换区

{

//送入对换区

if(!harddisk.SwapArea\_FULL())

{

harddisk.In\_SwapArea(Process\_PageList[Number].Pagenum);

}

else

{

harddisk.SwapArea\_Update(Process\_PageList[Number].Pagenum);

}

Memory.PtabelIns\_info+=("进入对换区"+"\n");

System.out.println("进入对换区");

SimulationUI.ProRec.append("进入对换区"+"\n");

Process\_PageList[Number].Modify=-1;

}

else//页面未被修改

{

pcb.page.Process\_PageList[Number].Pagenum=page;//修改进程的页表该页号

pcb.page.Process\_PageList[Number].Visit=0;//修改进程页表的访问字段

pcb.page.Process\_PageList[Number].Address[0]=pcb.HardDiskAddress[0][0];

pcb.page.Process\_PageList[Number].Address[1]=pcb.HardDiskAddress[1][page];

}

for(int k=0;k<4;k++)

{

if(Process\_PageList[k].State==1&&k!=Number)

Process\_PageList[k].Visit++;

}

//修改快表

if(!mmu.TLB\_FULL())//快表未满

{

for(int j=0;j<5;j++)

{

if(mmu.tlb[j].Pagenum<0)

{

mmu.tlb[j].Pagenum=pcb.page.Process\_PageList[Number].Pagenum;

mmu.tlb[j].Blocknum=pcb.page.Process\_PageList[Number].Blocknum;

mmu.tlb[j].State=pcb.page.Process\_PageList[Number].State;

mmu.tlb[j].Visit=pcb.page.Process\_PageList[Number].Visit;

mmu.BelongPro[j]=pcb.ProID;

mmu.TLB\_Visit(j);

break;

}

}

}

else//快表已满

{

mmu.TLB\_REPLACE(Number,pcb);

}

}

public int PageLRU(Page Process\_PageList[])

{

int Number=-1;

int i,j;

for(i=0;i<4;i++)

{

if(Process\_PageList[i].State==1)

{

if(Number<0)

{

Number=i;

}

else if(Process\_PageList[i].Visit>Process\_PageList[Number].Visit)

{

Number=i;

}

}

}

Memory.PtabelIns\_info+=("被替换的页面是"+Process\_PageList[Number].Pagenum+"\n");

System.out.println("被替换的页面是"+Process\_PageList[Number].Pagenum);

SimulationUI.ProRec.append("被替换的页面是"+Process\_PageList[Number].Pagenum+"\n");

return Number;

}

**（二）PageScheduling类**

//取指令

public void GetInstruc(Pcb pcb,MMU mmu,int Instruc\_ID,Memory memory,HardDisk harddisk,Cpu cpu)

{

int logicaddress;

int physicaladdress;

LogicAddress laddress=new LogicAddress();

logicaddress=pcb.Instruc[Instruc\_ID].Instruct\_Address;//取指令地址

Memory.PtabelIns\_info+=("访问指令,逻辑地址为"+logicaddress);

System.out.print("访问指令,逻辑地址为"+logicaddress);

SimulationUI.ProRec.append("访问指令,逻辑地址为"+logicaddress);

laddress=mmu.LAddress\_Divide(logicaddress);

Memory.PtabelIns\_info+=(",页号为"+laddress.block+",页内位移为"+laddress.offset+"\n");

System.out.println(",页号为"+laddress.block+",页内位移为"+laddress.offset);

SimulationUI.ProRec.append(",页号为"+laddress.block+",页内位移为"+laddress.offset+"\n");

physicaladdress=mmu.Logic\_To\_Physical(laddress, pcb);

switch(physicaladdress)

{

case -1:Memory.PtabelIns\_info+=("越界中断"+"\n");

System.out.println("越界中断");

SimulationUI.ProRec.append("越界中断"+"\n");break;

case -2:Memory.PtabelIns\_info+=("页表未命中,缺页中断"+"\n");

System.out.println("页表未命中,缺页中断");

SimulationUI.ProRec.append("页表未命中,缺页中断"+"\n");

cpu.PageFault(pcb.page.Process\_PageList,laddress.block, pcb, memory, mmu,harddisk);

pcb.page.GetInstruc(pcb, mmu, Instruc\_ID, memory,harddisk,cpu);

break;

default://转化为物理地址

Memory.PtabelIns\_info+=("物理地址为"+physicaladdress+",取指令"+"\n");

System.out.println("物理地址为"+physicaladdress+",取指令");

SimulationUI.ProRec.append("物理地址为"+physicaladdress+",取指令"+"\n");

if(pcb.Instruc[Instruc\_ID].Instruct\_State==0)

{

Memory.PtabelIns\_info+=("取址成功:系统调用指令"+"\n");

System.out.println("取址成功:系统调用指令");

SimulationUI.ProRec.append("取址成功:系统调用指令"+"\n");

}

if(pcb.Instruc[Instruc\_ID].Instruct\_State==1)

{

Memory.PtabelIns\_info+=("取址成功:申请资源指令"+"\n");

System.out.println("取址成功:申请资源指令");

SimulationUI.ProRec.append("取址成功:申请资源指令"+"\n");

}

if(pcb.Instruc[Instruc\_ID].Instruct\_State==2)

{

Memory.PtabelIns\_info+=("取址成功:释放资源指令"+"\n");

System.out.println("取址成功:释放资源指令");

SimulationUI.ProRec.append("取址成功:释放资源指令"+"\n");

}

if(pcb.Instruc[Instruc\_ID].Instruct\_State==3)

{

Memory.PtabelIns\_info+=("取址成功:普通指令"+"\n");

System.out.println("取址成功:普通指令");

SimulationUI.ProRec.append("取址成功:普通指令"+"\n");

}

}

}