|  |  |  |
| --- | --- | --- |
| 南农大  **计算机操作系统课程设计**  **实践报告**  XH2 | | |
|  | 题 目: | 可视化仿真实现Linux2.6进程管理与内存管理 |
|  | 姓 名: | 陈扬 梁嘉文 |
|  | 学 院: | 信息科技学院 |
|  | 专 业: | 计算机科学技术系 |
|  | 班 级: | 网工161 计科161 |
|  | 学 号: | 19316117 19216126 |
|  | 指导教师: | 姜海燕 职称: 教授 |
| 2019年2月 24 日 | | |

**目录**

[**摘要** 1](#_Toc2016479)

[**一、实践目的与任务** 2](#_Toc2016480)

[**（一）实践目的** 2](#_Toc2016481)

[**（二）实践任务** 2](#_Toc2016482)

[**（三）分工合作** 2](#_Toc2016483)

[**二、系统功能与API函数说明** 2](#_Toc2016484)

[**（一）仿真实现作业及进程并发环境** 2](#_Toc2016485)

[**（二）MMU地址变换** 3](#_Toc2016486)

[**（三）进程原语** 3](#_Toc2016487)

[**（四）页表生成与页面调度算法** 3](#_Toc2016488)

[**（五）三级作业调度过程及算法** 3](#_Toc2016489)

[**（六）页面分配与回收算法** 4](#_Toc2016490)

[**（七）进程同步互斥** 4](#_Toc2016491)

[**（八）进程死锁检测与撤销算法** 4](#_Toc2016492)

[**三、硬件仿真设计** 4](#_Toc2016493)

[**（一）CPU设计** 4](#_Toc2016494)

[**（二）内存设计** 5](#_Toc2016495)

[**（三）硬盘设计** 5](#_Toc2016496)

[**（四）地址线数据线** 5](#_Toc2016497)

[**四、基础数据结构与操作的抽象与设计** 6](#_Toc2016498)

[**（一）硬件设计** 6](#_Toc2016499)

[**（二）页表设计** 6](#_Toc2016500)

[**（三）JCB/PCB设计** 6](#_Toc2016501)

[**（四）系统全局变量** 7](#_Toc2016502)

[**五、程序结构及模块的设计与实现** 8](#_Toc2016503)

[**（一）程序设计结构** 8](#_Toc2016504)

[**（二）作业管理** 8](#_Toc2016505)

[**（三）进程管理** 9](#_Toc2016506)

[**（四）页面管理** 9](#_Toc2016507)

[**六、测试与分析** 10](#_Toc2016508)

[**（一）测试环境** 10](#_Toc2016509)

[**（二）硬件测试** 10](#_Toc2016510)

[**1、内存功能测试** 11](#_Toc2016511)

[**2、硬盘功能测试** 11](#_Toc2016512)

[**（三）功能测试** 12](#_Toc2016513)

[**七、技术问题及解决方案** 12](#_Toc2016514)

[**（一）框架设计** 12](#_Toc2016515)

[**（二）进程管理** 13](#_Toc2016516)

[**八、实践心得** 13](#_Toc2016517)

[**（一）陈扬的实验心得** 13](#_Toc2016518)

[**（二）梁嘉文的实验心得** 13](#_Toc2016519)

[**参考文献** 14](#_Toc2016520)

[**附件1：程序文件及结构说明** 14](#_Toc2016521)

[**附件2：类图说明** 15](#_Toc2016522)

[**附件3：带注释的部分核心代码** 15](#_Toc2016523)

**可视化仿真实现Linux2.6进程管理与内存管理**

**摘要**

为了检验自己的操作系统课程的学习情况与掌握程度，以及将该课程知识用编程语言描述的技能，本次小组选择“可视化仿真实现Linux2.6进程管理与内存管理”的题目进行操作系统课程设计。该课程设计对于理解操作系统中进程管理与内存管理的知识有着重要作用，同时，管理手段采用Linux2.6内核的规则，可借此机会检验自己的编程水平与Linux核心代码的阅读水平。

系统模拟仿真了Linux2.6系统，并实现了作业及进程并发环境、MMU 地址变换、进程原语、页表生成与页面调度算法、三级作业调度过程及算法（需作业调度到指令集，至少实现三态转换）、页面分配与回收算法、进程同步互斥、进程死锁检测与撤销算法，并将实现原理过程通过可视化方式呈现。

按照计算机理论，在程序框架上，将系统分为硬件、驱动程序、系统管理模块、UI界面四个模块。

在硬件层面，系统根据题目需要，设计了CPU、内存、外存（硬盘）、地址线数据线四个硬件，CPU中还包含计时器与MMU。CPU负责进程指令的执行与数据的传递，计时器负责发出中断与系统时间的计算，MMU负责系统中地址的变换。内存与外存是系统中的存储设备，所有作业、进程以及页面的管理都以这两个硬件为基础进行设计。

系统管理模块分为作业管理、进程管理与页面管理。作业管理模块是基于CPU与硬盘硬件的系统模块之一。该模块的功能是为作业的创建、存入、删除以及作业的调入检测提供相关的支持。作业管理模块在java工程中写在JobModule.java文件中，同时该类被Control.java调用。

进程管理模块是基于CPU与内存的系统模块之一。该模块的功能是为作业调入后转换成的进程提供管理功能。进程管理是整个系统设计中最为复杂的部分，其包括低级调度管理、中级调度管理、高级调度管理、进程链表等功能。进程管理模块在java工程中写在ProcessModule.java文件中，同时该类被Control.java调用。

页面管理是负责系统中对于页面存入、读取、对换、换入换出功能的管理模块。在页面管理中，该模块与其他模块的信息交换全部都通过Page类来进行。当有页面换入、换出请求发出时，该模块先将页面信息写入Page类的对象，再将该对象传出，当其他模块收到该对象时，也可以对该对象进行操作，以减少操作的复杂度。页面管理模块在java工程中写在PageModule.java文件中，同时该类被Control.java调用。

系统的UI界面为java程序直接提供给用户进行操作的界面，通过该界面，用户可以方便快捷的使用所有系统功能并知晓系统所有功能模块以及硬件设备的实时信息。

综上所述，该系统很好地实现了课程设计的所有要求，同时，还提供生动形象的UI界面，方便用户进行操作管理。

关键字：Linux2.6、系统仿真、进程管理、内存管理、JAVA编程

**可视化仿真实现Linux2.6进程管理与内存管理**

**一、实践目的与任务**

**（一）实践目的**

为了检验自己一学期的操作系统课程的学习情况与掌握程度，以及将该课程知识用编程语言描述的技能，所以本次选择“可视化仿真实现Linux2.6进程管理与内存管理”的题目进行操作系统课程设计。

该课程设计对于理解操作系统中进程管理与内存管理的知识有着重要作用，同时，管理手段采用Linux2.6内核的规则，可借此机会检验自己的编程水平与Linux核心代码的阅读水平。

**（二）实践任务**

根据Linux2.6 进程管理与内存管理原理，仿真实现作业及进程并发环境、MMU 地址变换、进程原语、页表生成与页面调度算法、三级作业调度过程及算法（需作业调度到指令集，至少实现三态转换）、页面分配与回收算法、进程同步互斥、进程死锁检测与撤销算法，并将实现原理过程通过可视化方式呈现。

**（三）分工合作**

**组长（陈扬）：**

三级作业调度过程及算法

进程死锁检测与撤销算法

可视化方式呈现过程

进程同步互斥的实现

系统整体框架的构思与搭建

程序设计规范的撰写

**组员（梁嘉文）：**

CPU部件的仿真

内存空间的仿真实现

MMU地址变换

进程与进程原语的设计实现

页表设计实现

页面调度算法

页面分配与回收算法

进程同步互斥实现

**二、系统功能与API函数说明**

**（一）仿真实现作业及进程并发环境**

public void JobToProcess(JCB d)

功能：从硬盘中读入的作业，为其分配资源成为进程

public void ToReady(short num)

功能：将num序号（pid）的进程放到就绪队列

public void ToRunning(short num)

功能：将num序号（pid）的进程放到运行队列

public void ToWaitting(short num)

功能：将num序号（pid）的进程放到等待队列

public void ProcessWait(short num,int time)

功能：使得进程等待time时间（ms）

**（二）MMU地址变换**

public void InitMMU()

功能：从CPU文件中读取mmu的信息

public void CloseMMU()

功能：将MMU的信息保存到CPU文件中去

**（三）进程原语**

public void ProcessCreate()

功能：进程原语：进程创建

public void ProcessCancel()

功能：进程原语：进程撤销

public void ProcessWait()

功能：进程原语：进程阻塞

public void ProcessWake()

功能：进程原语：进程唤醒

**（四）页表生成与页面调度算法**

public void SetPageNum(int num)

功能：分配页表序号

public void SetPageData(int add,short data)

功能：写入页表数据

public short GetPageData(int add) 返回值为页表中的数据

功能：获取页表数据

public void RefreshPageUseInfo()

功能：更新页表使用情况（命中置0，未命中+1）

public short GetLRUPageNum()

功能：对所有页表使用情况排序后返回最久未使用页表

public void ExchangePage(short num1,short num2)

功能：将num1序号的页面与num2序号的页面对换

public short ChangeInPage(short num) 返回值为被唤出的内存页号

功能：将num序号的页面（该页面位于外存的虚存区）换入到内存区，返回被换出的内存页号

**（五）三级作业调度过程及算法**

public short GetJobNum() 返回值为作业总数量的值

功能：获得当前磁盘中的作业总数量

public void SaveJobToHardDisk(JCB d1,ArrayList<Integer> d2)

功能：将作业保存到外存，d1是JCB数据块，d2是包含所有指令的ArrayList，每条指令占8字节

public JCB GetJobFromHardDisk(ArrayList<Integer> instructions) 返回值为获取的作业的数据

功能：按照FIFO方法，读取磁盘中下一个将被调度进来的JCB，同时，将该作业所有的指令存入instructions中

public short GetJobNeedPages() 返回值为作业所需要的页框数

功能：返回下一个需要被调入的所需要占用的页框数

**（六）页面分配与回收算法**

public void GetFreePageNum()

功能：返回当前物理内存+虚存中可用的页框数

public boolean IfCouldGetPage(short num) 返回值为允许或不允许

功能：是否可以申请num个页面（不要求连续）

public void RecyclePage(short num)

功能：回收页面，参数num为页面号（num从0开始编号）

public void WritePage(short num,Page data)

功能：将传入的页面data写入第num个页面中

public Page GetPage(short num) 返回值为第num个页面的数据

功能：获得第num个页面

public String ApplyPage(short page\_num) 返回值为页面分配情况

功能：申请page\_num个页面，返回一个String类型的值，String格式的数据说明：从左到右编号为0-127，共128位，每一位的值为0/1，1代表该页面分配给该进程使用。在写入程序区时，必须按照分配的页面顺序从小到大写入

**（七）进程同步互斥**

public void POperation(short num,short instruction)

功能：进程的P操作处理

public void VOperation(short num,short instruction)

功能：进程的V操作处理

**（八）进程死锁检测与撤销算法**

public void ProcessXXCheck()

功能：进程死锁检测

public void ProcessXXRelease()

功能：进程死锁释放

**三、硬件仿真设计**

**（一）CPU设计**

CPU硬件中包含计时器与MMU。在程序设计时，需要将两者分开，设计三个类，CPU类“cpu”，计时器类“timer”，MMU硬件类“mmu”。timer与MMU只在cpu类中存有唯一一个实例，即可体现出计时器与MMU包含于CPU的概念。任何对计时器与MMU的调用，都需要通过cpu类中的实例来进行，而不能够直接访问。

CPU类的定义为：

public class CPU

{

public static CPU cpu=new CPU();

public Timer ti;

public MMU mm;

}

CPU的结构及内容：

1、地址寄存器PC

2、PSW程序状态寄存器

3、IR（指令寄存器）

4、页基址寄存器CR3

5、在时间片结束时执行完的指令数量already\_run

**（二）内存设计**

内存大小为32KB，故可根据该要求设计memory类，该类有一个总大小为32KB的对象数组，为short类型，因为系统要求地址线与数据线为16位，所以在该系统的设计时，统一使用short类型数据。需要注意：内存memory类为文件的映射，因此，在该类被实例化之前，其构造函数需要进行从文件到对象的映射，即读取文件内容，并根据文件内容初始化该实例；在结束程序时需将更改后的内存文件还原至文件中。以便下次打开时进行映射操作。内存硬件类memory只是简单地对硬件进行模拟仿真，不需要过多复杂的操作，只需要提供两个基本操作即可，即数据的存入与取出。但是，数据的存入与取出都需要通过地址线与数据线。

内存类的定义为：

public class Memory

{

public static Memory memory=new Memory();

private byte []data=new byte[32\*1024]; //32KB=32768B

}

内存总大小为32KB，规定：前16KB为内核区，后16KB为用户区

内核区存储的内容：核心栈+系统内核，进程所有PCB信息

综上所述，内存的组成结构为：

核心栈+系统内核（1页）、PCB池（31页）、用户区（32页）

**（三）硬盘设计**

与内存硬件的设计相似，硬盘类只提供简单的数据存入取出。但是，数据的存入与取出也都需要通过地址线与数据线。硬盘硬件只负责最基本的硬件仿真。硬盘harddisk类为文件的映射，因此，在该类被实例化之前，其构造函数需要进行从文件到对象的映射，即读取文件内容，并根据文件内容初始化该实例；在结束程序时需将更改后的内存文件还原至文件中。以便下次打开时进行映射操作。

硬盘类定义为：

public class HardDisk

{

public static HardDisk harddisk=new HardDisk();

private byte [][][]data=new byte[32][64][512];

}

硬盘总大小为1MB，规定：

前48KB为虚存区（与内存的16KB用户区组合，加起来共64KB，也是16位地址线的最大寻址空间）

再16KB为系统文件区（用来模拟开机后需要被加载入内存的系统文件）

剩下的960KB都为文件区（用来存储作业）

综上所述，硬盘的组成结构为：

虚存区（96页）+系统文件区（32页）+文件区（1920页）

**（四）地址线数据线**

地址线数据线为计算机硬件中存在但是会被经常忽略的部件。在该系统的设计中，有必要对地址线与数据线进行仿真。

根据计算机组成原理与系统结构的知识，地址线与数据线为在物理上连接CPU与其他各个带有存储功能部件的物理结构。在设计时，若对每一个部件都进行地址线与数据线的存入取出处理，必然会增加系统设计的复杂程度。故地址线与数据线的设计可考虑进行适当地简化。

故将设计思想简化为：只有当硬件需要进行存取操作时，才开始调用地址线与数据线。

**四、基础数据结构与操作的抽象与设计**

**（一）硬件设计**

系统中共包含三个硬件：CPU、内存与硬盘。而CPU类中又包含有MMU类和Timer类的两个实例对象，用来表示CPU对MMU与Timer两个组件的包含关系。

针对MMU类，设计有从文件加载与保存到文件的功能；Timer类设计有计时器相关的开始、结束、中断等功能。

在CPU类中，可以完成上述功能的使用及对CPU的加载和转存。

内存类与硬盘类的设计中主要考虑了作为存储器的本质功能，即存储数据与读取数据，从而将管理工作交给了更高一层的功能模块进行处理，只留有对特定地址进行读写操作的功能。

**（二）页表设计**

Page类设计如下：

{

private int page\_num; //页号

private short[] data=new short[512/2]; //每页大小=512B=256个short类型

}

由于地址线和数据线均为16位，故每页中单位大小设置为16位，采用short类型进行描述。采用统一的页大小设计后，在进行页面调度时可以值关注调度算法，避免数据类型的影响。

**（三）JCB/PCB设计**

PCB类中定义了如下变量，用以描述进程控制块的信息：

private short pid; //进程标识符

private short state; //进程状态。就绪态、等待态、运行态

private short priority; //进程优先级

private int job\_intime; //作业创建时间

private int process\_intime; //进程创建时间

private int end\_time; //作业/进程结束时间

private short timeslice; //时间片长度

private int runtime; //进程已经运行时间

private int counter; //该进程处于运行状态下的时间片余额

private byte PSW; //程序状态字。管态、目态

private short current\_instruction\_no; //当前运行到的指令编号

private short instruction\_num; //该进程总共包含的指令数目

private short pages\_num; //该作业/进程所占用的页面数目

private short [][]page\_table=new short[62][2];

//页表，page\_table[i][0]为进程的页号，从0开始编号；page\_table[i][1]为对应的物理页号

private short already\_run\_instruction\_num; //在时间片结束时执行完的指令数量

private ArrayList<Integer> instructions=new ArrayList<Integer>(); //该进程所有的指令

同时基于这些变量分别设置了用来读取和修改的接口函数进行操作。

JCB类中定义了如下变量，用以描述作业的信息：

private short priority; //进程优先级

private int job\_intime; //作业进入时间

private short instruction\_num; //作业包含的指令数目

private short pages\_num; //作业所占用的页面数目

同样分别设置了用来读取和修改的接口

**（四）系统全局变量**

kernel类中定义了系统环境中的一些常量的值，以及一些变量不同值所对应的状态描述

/\*系统基本信息\*/

public static int MEMORY\_SIZE=32\*1024;

意义：内存大小，32KB

public static int MEMORY\_KERNEL\_SPACE\_SIZE=16\*1024;

意义：内存内核空间大小，16KB

public static int MEMORY\_KERNEL\_CORESTACKANDOSKERNEL\_

SIZE=512;

意义：核心栈+系统内核大小，1页

public static int MEMORY\_KERNEL\_PCBS\_SIZE=31\*512;

意义：PCB池大小，31页

public static int MEMORY\_USER\_SPACE\_SIZE=16\*1024;

意义：内存用户空间大小（页表、页框使用），16KB

public static int HARDDISK\_SIZE=1\*1024\*1024;

意义：硬盘空间大小，1MB

public static int HARDDISK\_VIRTUAL\_STORAGE\_SIZE=96\*512;

意义：虚存区大小，96页

public static int HARDDISK\_SYSTEMFILE\_SIZE=32\*512;

意义：系统文件大小，32页

public static int HARDDISK\_FILE\_SPACE\_SIZE=1920\*512;

意义：文件区大小，1920页

public static int INTERRUPTION\_INTERVAL=10;

意义：系统发生中断的间隔

/\*系统基本信息\*/

/\*Process State 进程状态参数\*/

public final static int PROCESS\_READY = 0;

意义：就绪态

public final static int PROCESS\_WAITING = 1;

意义：等待态

public final static int PROCESS\_RUNNING = 2;

意义：运行态

public final static int PROCESS\_SUSPENSION = 3;

意义：挂起态

/\*Process State 进程状态参数\*/

/\*Process PSW 程序状态字\*/

public final static int PSW\_KERNEL\_STATE=0;

意义：管态

public final static int PSW\_USER\_STATE=1;

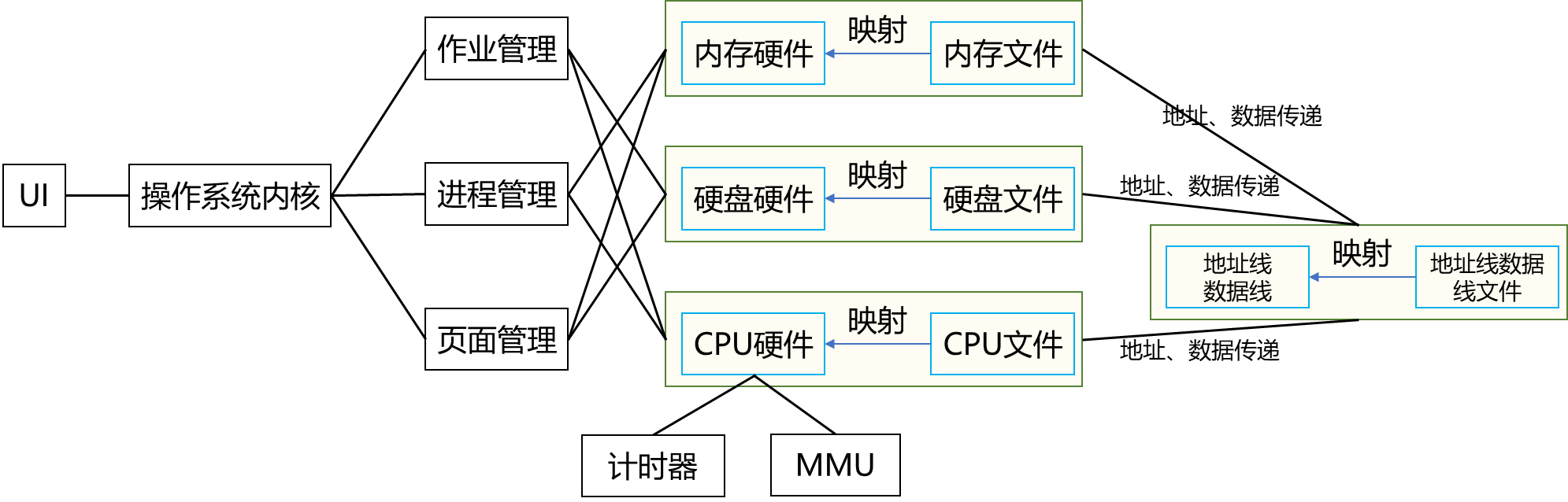
意义：目态

/\*Process PSW 程序状态字\*/

**五、程序结构及模块的设计与实现**

**（一）程序设计结构**

基于程序的硬件设计与基础数据结构设计，在此基础上便可进行程序的结构设计。本系统的结构设计参考了操作系统课本的设备管理章节，从最底层网上可分为：硬件、硬件驱动、系统管理模块、系统内核、UI界面。同时，由于对于硬件部分的仿真本身就包含了存入、读取的功能，所以将硬件与硬件驱动进行合并。在系统管理模块部分，分为：作业管理、进程管理、页面管理三部分。在系统内核部分，系统提供了系统的全局变量与系统的操作类，用来将其下的三个模块进行整合。再网上，即JAVA所提供的UI交互界面，可以方便用户使用该系统。

 程序的总体结构设计如图所示。

**（二）作业管理**

作业管理模块是基于CPU与硬盘硬件的系统模块之一。该模块的功能是为作业的创建、存入、删除以及作业的调入检测提供相关的支持。

作业管理模块在java工程中写在JobModule.java文件中，同时该类被Control.java调用。

与作业管理有关的操作有：

public short GetJobNum() 返回值为作业总数量的值

功能：获得当前磁盘中的作业总数量

public void SaveJobToHardDisk(JCB d1,ArrayList<Integer> d2)

功能：将作业保存到外存，d1是JCB数据块，d2是包含所有指令的ArrayList，每条指令占8字节

public JCB GetJobFromHardDisk(ArrayList<Integer> instructions) 返回值为获取的作业的数据

功能：按照FIFO方法，读取磁盘中下一个将被调度进来的JCB，同时，将该作业所有的指令存入instructions中

public short GetJobNeedPages() 返回值为作业所需要的页框数

功能：返回下一个需要被调入的所需要占用的页框数

**（三）进程管理**

进程管理模块是基于CPU与内存的系统模块之一。该模块的功能是为作业调入后转换成的进程提供管理功能。

进程管理是整个系统设计中最为复杂的部分，其包括低级调度管理、中级调度管理、高级调度管理、进程链表等功能。

作业管理模块在java工程中写在ProcessModule.java文件中，同时该类被Control.java调用。

与进程管理有关的操作有：

public void ToReady(short num)

功能：将num序号（pid）的进程放到就绪队列

public void ToRunning(short num)

功能：将num序号（pid）的进程放到运行队列

public void ToWaitting(short num)

功能：将num序号（pid）的进程放到等待队列

public void ProcessWait(short num,int time)

功能：使得进程等待time时间（ms）

public void ProcessCreate()

功能：进程原语：进程创建

public void ProcessCancel()

功能：进程原语：进程撤销

public void ProcessWait()

功能：进程原语：进程阻塞

public void ProcessWake()

功能：进程原语：进程唤醒

**（四）页面管理**

页面管理是负责系统中对于页面存入、读取、对换、换入换出功能的管理模块。在页面管理中，该模块与其他模块的信息交换全部都通过Page类来进行。当有页面换入、换出请求发出时，该模块先将页面信息写入Page类的对象，再将该对象传出，当其他模块收到该对象时，也可以对该对象进行操作，以减少操作的复杂度。

作业管理模块在java工程中写在PageModule.java文件中，同时该类被Control.java调用。

与页面管理相关的操作有：

public void SetPageNum(int num)

功能：分配页表序号

public void SetPageData(int add,short data)

功能：写入页表数据

public short GetPageData(int add) 返回值为页表中的数据

功能：获取页表数据

public void RefreshPageUseInfo()

功能：更新页表使用情况（命中置0，未命中+1

public short GetLRUPageNum()

功能：对所有页表使用情况排序后返回最久未使用页表

public void ExchangePage(short num1,short num2)

功能：将num1序号的页面与num2序号的页面对换

public short ChangeInPage(short num) 返回值为被唤出的内存页号

功能：将num序号的页面（该页面位于外存的虚存区）换入到内存区，返回被换出的内存页号

public void GetFreePageNum()

功能：返回当前物理内存+虚存中可用的页框数

public boolean IfCouldGetPage(short num) 返回值为允许或不允许

功能：是否可以申请num个页面（不要求连续）

public void RecyclePage(short num)

功能：回收页面，参数num为页面号（num从0开始编号）

public void WritePage(short num,Page data)

功能：将传入的页面data写入第num个页面中

public Page GetPage(short num) 返回值为第num个页面的数据

功能：获得第num个页面

public String ApplyPage(short page\_num) 返回值为页面分配情况

功能：申请page\_num个页面，返回一个String类型的值，String格式的数据说明：从左到右编号为0-127，共128位，每一位的值为0/1，1代表该页面分配给该进程使用。在写入程序区时，必须按照分配的页面顺序从小到大写入

**六、测试与分析**

**（一）测试环境**

系统：Windows 10 1809(17763.316)

CPU：Intel Core i7-6700HQ

内存：16GB

显卡：NVIDIA GTX 965M

硬盘：500GB 7200rpm

JAVA版本：

java 11.0.2 2019-01-15 LTS

Java(TM) SE Runtime Environment 18.9 (build 11.0.2+9-LTS)

**（二）硬件测试**

本系统在结构上完全模拟仿真计算机裸机硬件的设计结构。在最底层的硬件设计上，根据系统的功能需求，设计了CPU、内存、外存（硬盘）三个主要硬件。同时，地址线、数据线也有设计，其功能为联系CPU与其他两个硬件之间的地址、数据传递，因此在本文档中并不把地址线、数据线作为单独的硬件进行测试。

CPU硬件中同时包含计时器与MMU，无法进行单独测试，需要与具体的进程管理与内存管理结合使用。在此处，只进行内存与外存硬件的功能测试。

**1、内存功能测试**

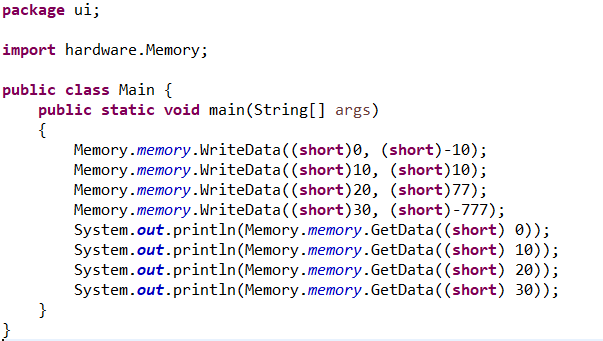
在系统的硬件设计中，内存与外存（硬盘）是负责存储功能的硬件。内存负责存储与进程相关的数据与系统内核区。

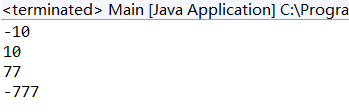
内存基本功能是否准确决定了系统是否能够正常运行，因此，对于内存硬件的测试是整个系统测试的基础。

在内存设计中，小组充分考虑了地址线与数据线的作用，所有的地址信息与数据信息的传入与取出都必须通过地址线数据线。

内存的测试方法：在main函数中实例化内存Memory类的实例，并通过此实例调用存入、读取功能，进行多次测试，以检验该模块设计的准确性。

如图，存入数据：-10、10、77、-777数据并取出，结果正确，即说明该硬件类的设计正确。





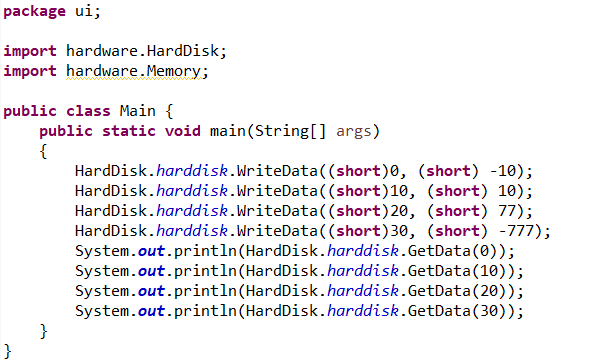
**2、硬盘功能测试**

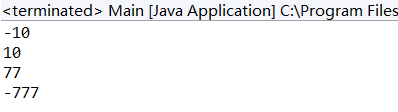
硬盘的功能为存储作业文件与提供虚存区。硬盘基本功能是否准确决定了系统是否能够正常的调用虚存功能、使用页面管理功能与页面调度。

在硬盘的设计中，小组充分考虑了地址线与数据线的作用，所有的地址信息与数据信息的传入与取出都必须通过地址线数据线。

硬盘的测试方法：在main函数中实例化内存HardDisk类的实例，并通过此实例调用存入、读取功能，进行多次测试，以检验该模块设计的准确性。

如图，存入数据：-10、10、77、-777数据并取出，结果正确，即说明该硬件类的设计正确。





**（三）功能测试**

目前该模块的功能尚未全部完成，功能测试尚不可用。

**七、技术问题及解决方案**

**（一）框架设计**

1、JAVA中全局变量的定义及使用

问题描述：与C/C++编程语言不同，JAVA编程语言是一种强制面向对象的语言，没有“全局变量”的说法。但是，在实际编程中，很多地方需要使用到全局变量。

解决方案：JAVA提供了静态变量的设计方法，由此受到启发，可以使用JAVA中“静态对象”的方法来进行设计。比如，内存在整个系统设计时，只能够有一个对象，且需要其他模块对这个对象都进行共享，使用同一个数据。此时，便可以在内存类中直接使用public static关键词对该类进行实例化。在其他模块调用时，不必使用地址传递，直接使用***Memory.memory***对象即可调用。

2、硬件模拟仿真

问题描述：在实际的Linux系统中，硬件都以文件的形式进行表示。在本系统设计中，无法从界面上体现出“文件”的概念，所以需要另外设计方法来体现出Linux系统从硬件到文件的映射关系。

解决方案：小组自行设计每一个硬件的存储结构。同时，在JAVA代码编写时，在内存、硬盘、CPU类的构造函数中，直接调用文件读取函数，从文件将各个硬件的信息读入，完成从文件到类的映射。系统的硬件的抽象表示为类和对象，实际的表示为文件。通过此方法，可以在一定程度上模拟仿真Linux系统对硬件的描述。

**（二）进程管理**

1、调度算法线程之间的相互制约

问题描述：在系统中，实现了三级调度。每一级调度的发生时间与频次各不相同，在某些特殊时刻，会同时发生多个调度，如何协调好这些调度的次序是一个相当麻烦的问题。

解决方案：小组提供两种解决方案，一种为：将三级调度所使用的函数统一归纳，写在一个函数中，每次调度时，都按照低级调度、中级调度、高级调度的顺序进行，这样，便可以保证每一级调度的时间先后顺序，即使在三个调度同时发生的时刻。另一种解决方案是，将三个调度方法分别运行于三个不同的线程，每一个线程必须保证自己所操作的数据是独占的，同时，需要通过多个信号量来协调不同的线程之间的关系。比较两种方法，前者更容易实现，但是不能够很明显的体现出并发的特性；后者相反，但是实现起来比较复杂，且会根据JAVA解释器的不同得出不同的结果。

**八、实践心得**

**（一）陈扬的实验心得**

经过一个寒假的努力，与队友付出了艰苦卓绝的努力，现在的系统终于初具规模。虽然系统的一些方面仍然有待改进，但是现在我对于操作系统整体概念与进程管理、内存管理的理解愈加深刻。

在实验过程中，令我印象最为深刻的是对于操作系统理论的理解与对于程序框架的搭建。

个人认为，在本次课程设计实验中，为了设计出一个符合要求，能够模拟仿真Linux2.6的系统，最为重要的是对于操作系统的整体概念的理解掌握。有了这些基础知识，才能够理清楚系统中各个部分、各个功能的含义以及他们与其他模块的联合调用关系，不然，在设计系统时将会举步维艰，寸步难行。

有了基础知识，其次便是程序结构的设计。一个好的程序结构，应该充分考虑系统的各方面特性，如系统与Linux2.6进程管理与内存管理的仿真如何实现，系统的模块划分是否合理，系统的内聚与耦合程度如何……在经过了艰苦卓绝的程序框架构思与设计后，小组终于设计出了一个很满意的程序框架。在此框架上编写的程序代码，相信能够有很好的表现。

**在此次的课程设计中，我主要负责：三级作业调度过程及算法、进程死锁检测与撤销算法、可视化方式呈现过程、进程同步互斥的实现、系统整体框架的构思与搭建、程序设计规范的撰写。**

目前，小组的课程设计进展良好，但是由于中间遇到过许多理论上与技术上的难题，当前的进展与原计划相比略有滞后。我们会抓紧时间，精心设计，认真调试，在最后给出一个令人满意的结果。

**（二）梁嘉文的实验心得**

在经过一个寒假之后，虽然系统的部分模块仍然有待完善，但我对于操作系统的整体概念，以及进程管理、内存管理等内容已有了更深的理解。

在本次课程设计实验中，为了设计出模拟仿真Linux2.6的系统，最为重要的是对于操作系统的整体概念的深入理解。没有深刻地理解这些基础知识，也就没有办法在更高的角度上整合这些模块，实现系统的仿真。

**我主要负责：****CPU部件的仿真、内存空间的仿真实现、MMU地址变换、进程与进程原语的设计实现、页表设计实现、页面调度算法、页面分配与回收算法、进程同步互斥实现**等几个方面内容的撰写，以及对系统进行调试和修改等工作。

虽然我所负责的大都是实现系统所需要的基础内容，更多是一些底层API函数和功能的实现，相较于更高层面的管理调度模块，难度稍低，但更贴近课程所学内容，需要在调度时直接提供相应的算法支持，这更能让我深刻理解课程中所学习的算法的工作原理和设计思路。

课程设计目前还在进行之中，在剩下的时间里我会抓紧时间继续学习，积极完成系统搭建和调试工作，认真完成课程设计任务

**参考文献**

[1]费翔林,骆斌.操作系统教程(第五版)[M].北京:高等教育出版社,2014

[2] Daniel P. Bovet,Marco Cesati.深入理解linux内核(第三版)[M].北京:中国电力出版社,2008

[3] Linux的内存管理[EB/OL].

https://www.cnblogs.com/xelatex/p/3491301.html

[4] Linux2.6 内核进程调度分析[EB/OL].

https://blog.csdn.net/dlutbrucezhang/article/details/8694793

[5] Linux2.6内核--进程调度理论[EB/OL].

https://www.cnblogs.com/joey-hua/p/5707780.html

[6] Linux 2.6 调度系统分析[EB/OL].

https://blog.csdn.net/hzrandd/article/details/51034488

[7] Linux内核中几个比较有意思的解释(进程调度算法，页面调度算法，非线性工作集) [EB/OL].

https://blog.51cto.com/dog250/1698404

**附件1：程序文件及结构说明**

├─hardware

│ AddressLine.java

│ CPU.java

│ DataLine.java

│ HardDisk.java

│ Memory.java

│ MMU.java

│ Timer.java

│

├─os

│ Control.java

│ JCB.java

│ JobModule.java

│ kernel.java

│ Page.java

│ PageModule.java

│ PCB.java

└─ ProcessModule.java

**附件2：类图说明**

由于目前程序的功能尚未完成，所以无法提供系统整体的类图。系统的类图将在最终的实验报告中予以体现。

**附件3：带注释的部分核心代码**

**文件：HardDisk.java**

package hardware;

public class HardDisk

{

public static HardDisk harddisk=new HardDisk();

private byte [][][]data=new byte[32][64][512];

public HardDisk()

{

InitHardDisk();

}

//将byte组装成short类型数据

private synchronized short ByteToShort(short cylinder,short sector,short offset)

{

//cylinder磁道、sector扇区、offset偏移

//地址结构：偏移——扇区——磁道

//占用位数：偏移0-8 扇区9-14 磁道15-19

short temp=0;

if(offset%2==0)

{

temp=(short)((short)(this.data[cylinder][sector][offset])<<8);

temp|=this.data[cylinder][sector][offset+1]&0x00FF;

}

else

{

temp=(short)((short)(this.data[cylinder][sector][offset-1])<<8);

temp|=this.data[cylinder][sector][offset]&0x00FF;

}

return temp;

}

public synchronized void WriteData(int address,short data)

{

short address\_0\_15=0;

short address\_16\_31=0;

AddressLine.address\_line.WriteAddress((short) (address&0x0000FFFF));

address\_0\_15=AddressLine.address\_line.GetAddress();

AddressLine.address\_line.WriteAddress((short) ((address>>16)&0x0000FFFF));

address\_16\_31=AddressLine.address\_line.GetAddress();

//cylinder磁道、sector扇区、offset偏移

//地址结构：偏移——扇区——磁道

//占用位数：偏移0-8 扇区9-14 磁道15-19

short offset=(short) (address\_0\_15&0x01FF);

short sector=(short) ((address\_0\_15>>9)&0x03F);

short cylinder=(short) ((short) ((address\_0\_15&0x08000)>>11)|(short) (address\_16\_31&0x0F));

if(offset%2==0)

{

DataLine.data\_line.WriteData(data);

short data\_temp=DataLine.data\_line.GetData();

this.data[cylinder][sector][offset]=(byte) ((data\_temp>>8)&0x00FF);

this.data[cylinder][sector][offset+1]=(byte) (data\_temp&0x00FF);

}

else

{

DataLine.data\_line.WriteData(data);

short data\_temp=DataLine.data\_line.GetData();

this.data[cylinder][sector][offset-1]=(byte) ((data\_temp>>8)&0x00FF);

this.data[cylinder][sector][offset]=(byte) (data\_temp&0x00FF);

}

}

public synchronized short GetData(int address)

{

short address\_0\_15=0;

short address\_16\_31=0;

AddressLine.address\_line.WriteAddress((short) (address&0x0000FFFF));

address\_0\_15=AddressLine.address\_line.GetAddress();

AddressLine.address\_line.WriteAddress((short) ((address>>16)&0x0000FFFF));

address\_16\_31=AddressLine.address\_line.GetAddress();

//cylinder磁道、sector扇区、offset偏移

//地址结构：偏移——扇区——磁道

//占用位数：偏移0-8 扇区9-14 磁道15-19

short offset=(short) (address\_0\_15&0x01FF);

short sector=(short) ((address\_0\_15>>9)&0x03F);

short cylinder=(short) ((short) ((address\_0\_15&0x08000)>>11)|(short) (address\_16\_31&0x0F));

DataLine.data\_line.WriteData(ByteToShort(cylinder,sector,offset));

return DataLine.data\_line.GetData();

}

}

**文件：Memory.java**

package hardware;

public class Memory

{

public static Memory memory=new Memory();

private byte []data=new byte[32\*1024]; //32KB=32768B

public Memory()

{

InitMemory();

}

//将byte组装成short类型数据

private synchronized short ByteToShort(short address)

{

short temp=0;

if(address%2==0)

{

temp=(short) ((short)(this.data[address])<<8);

temp|=this.data[address+1]&0x00FF;

}

else

{

temp=(short) ((short)(this.data[address-1])<<8);

temp|=this.data[address]&0x00FF;

}

return temp;

}

public synchronized void WriteData(short address,short data)

{

AddressLine.address\_line.WriteAddress(address); //向地址线中写入地址信息

DataLine.data\_line.WriteData(data); //向数据线中写入数据信息

short address\_temp=AddressLine.address\_line.GetAddress();

short data\_temp=DataLine.data\_line.GetData();

if(address\_temp%2==0)

{

this.data[address\_temp]=(byte) (data\_temp>>8); //取高8位

this.data[address\_temp+1]=(byte) (data\_temp); //取低8位

}

else

{

this.data[address\_temp-1]=(byte) (data\_temp>>8); //取高8位

this.data[address\_temp]=(byte) (data\_temp); //取低8位

}

//将数据线中的信息写入到地址线指示的内存地址

}

public synchronized short GetData(short address)

{

AddressLine.address\_line.WriteAddress(address); //向地址线中写入地址信息

DataLine.data\_line.WriteData(ByteToShort(AddressLine.address\_line.GetAddress()));

return DataLine.data\_line.GetData();

//根据地址线中的地址信息取出内存中相应的数据

}

}

**文件：Timer.java**

package hardware;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.Date;

public class Timer extends Thread

{

private boolean if\_interrupt=false; //是否发生中断的标志位

private String current\_time=""; //存储当前时间的字符串

SimpleDateFormat sdf=new SimpleDateFormat("HH:mm:ss");

Calendar calendar;

Date date;

public short interval=1000; //计时器的中断发生间隔

public Timer()

{

InitTimer();

}

public void run()

{

while(true)

{

try

{

Thread.sleep(interval);

if\_interrupt=true;

} catch (InterruptedException e)

{

e.printStackTrace();

}

}

}

public boolean GetIfInterrupt()

{

return if\_interrupt;

}

public void ResetIfInterrupt()

{

if\_interrupt=false;

}

public String GetCurrentTime()

{

calendar = Calendar.getInstance();

date = calendar.getTime();

current\_time = sdf.format(date);

return current\_time;

}

}