|  |  |  |
| --- | --- | --- |
| 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3月 15 日 | | |

**目录**

[**摘要** 1](#_Toc3554432)

[**一、实践目的与意义** 2](#_Toc3554433)

[**二、实践任务与合作** 2](#_Toc3554434)

[**三、程序结构说明** 2](#_Toc3554435)

[**四、裸机硬件仿真设计** 3](#_Toc3554436)

[**（一）CPU设计** 3](#_Toc3554437)

[**（二）内存设计** 3](#_Toc3554438)

[**（三）硬盘设计** 4](#_Toc3554439)

[**（四）地址线数据线** 4](#_Toc3554440)

[**五、通用数据结构设计** 4](#_Toc3554441)

[**（一）页表设计** 4](#_Toc3554442)

[**（二）PCB设计** 5](#_Toc3554443)

[**（三）JCB设计** 5](#_Toc3554444)

[**（四）系统全局变量** 6](#_Toc3554445)

[**六、模块设计说明** 7](#_Toc3554446)

[**（一）作业管理** 7](#_Toc3554447)

[**（二）进程管理** 8](#_Toc3554448)

[**（三）页面管理** 11](#_Toc3554449)

[**（四）调度算法** 13](#_Toc3554450)

[**（五）界面模块** 14](#_Toc3554451)

[**七、参考文献** 14](#_Toc3554452)

[**附件1：程序文件及结构说明** 15](#_Toc3554453)

[**附件2：类图说明** 16](#_Toc3554454)

**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、实践目的与意义**

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、实践任务与合作**

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。

为了实现以上任务目标，我们进行了如下分工：

**组长（陈扬）：**

系统整体框架的构思与搭建

程序设计规范的撰写

三级调度过程及算法

JCB、PCB的设计

死锁检测与撤销算法

页表生成

可视化方式呈现过程

进程同步互斥的实现

进程与进程原语的设计实现

**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

页面设计实现

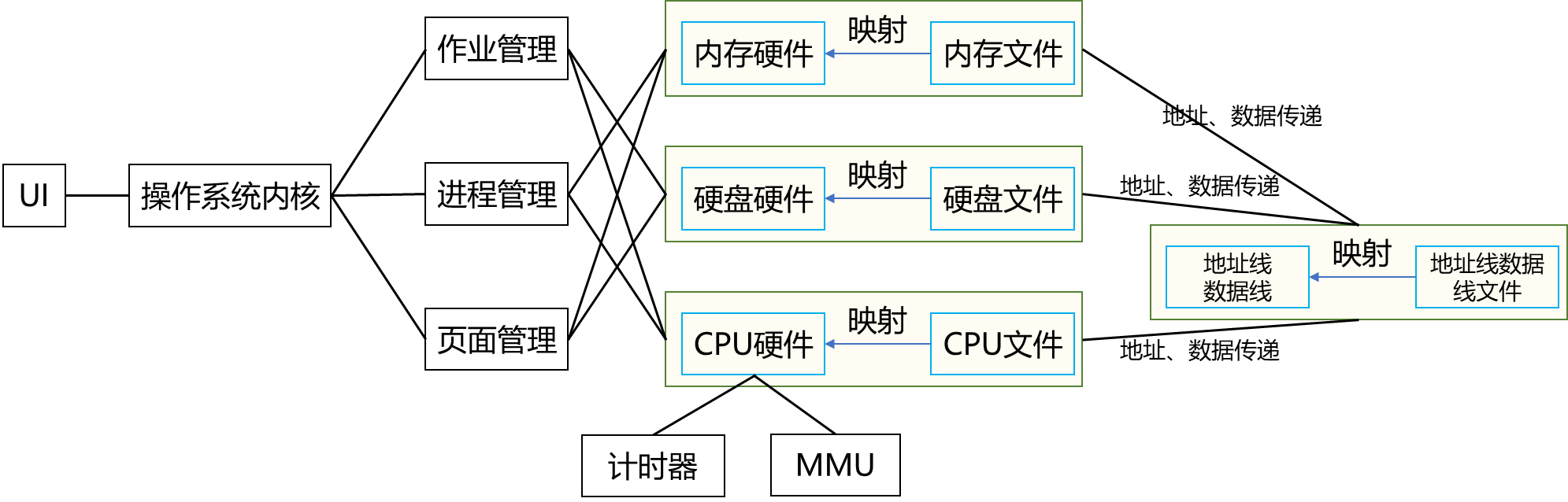
页面调度算法

页面分配与回收算法

可视化方式呈现过程

**三、程序结构说明**

本系统的结构设计参考了操作系统课本的设备管理章节，从最底层向上可分为：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。再向上，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。

**四、裸机硬件仿真设计**

**（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。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。

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。

**五、通用数据结构设计**

**（一）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只关注调度算法，避免数据类型的影响。

**（二）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、挂起态

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; //每次运行时，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

public short [][]page\_table = new short

[(kernel.MEMORY\_USER\_SPACE\_SIZE)/kernel.SINGLE\_PAGE\_SIZE][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 ArrayList<Integer> instructions=new ArrayList<Integer>(); //该进程所有的指令

private short in\_page\_num=0; //该PCB所在的页号

private short pool\_location=-1; //该PCB在PCB池的位置

private int total\_runtime=0; //总共运行的时间

public int ins\_runtime=0; //指令的执行总时间

public boolean if\_in\_p=false; //是否处于P状态

public boolean if\_p\_success=true; //P是否成功

分别设置了用来读取和修改数据成员的接口。

**（三）JCB设计**

private short job\_id; //作业ID

private short priority; //作业/进程的优先级

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

private ArrayList<Short> all\_instructions=new ArrayList<Short>(); //所有指令的链表

private short in\_page\_num=0; //该JCB所在的页号

分别设置了用来读取和修改数据成员的接口。

**（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 SINGLE\_PAGE\_SIZE=512; //每一页/块的大小

public static int MEMORY\_SIZE=32\*1024; //内存大小，32KB

public static int MEMORY\_KERNEL\_SPACE\_SIZE=16\*1024; //内存内核空间大小，16KB

public static int MEMORY\_KERNEL\_CORESTACKANDOSKERNEL\_SIZE=512; //核心栈+系统内核大小，1页

public static int MEMORY\_KERNEL\_PCB\_POOL\_SIZE=31\*512; //PCB池大小，31页

public static int MEMORY\_USER\_SPACE\_SIZE=16\*1024; //内存用户空间大小（页表、页框使用），16KB

public static int HARDDISK\_SIZE=1\*1024\*1024; //硬盘空间大小，1MB

public static int HARDDISK\_VIRTUAL\_MEMORY\_SIZE=64\*1024; //虚存区大小，128页，64KB

public static int HARDDISK\_SYSTEMFILE\_SIZE=16\*1024; //系统文件大小，32页，16KB

public static int HARDDISK\_FILE\_SPACE\_SIZE=944\*1024; //文件区大小，1888页，944KB

public static int HARDDISK\_CYLINDER\_NUM=32; //磁盘磁道数

public static int HARDDISK\_SECTOR\_NUM=64; //磁盘扇区数

public static int HARDDISK\_PAGE\_SIZE=512; //磁盘每页/块大小

public static int SINGLE\_INSTRUCTION\_SIZE=8; //单条指令的大小

public static int INSTRUCTIONS\_PER\_PAGE=SINGLE\_PAGE\_SIZE/SINGLE\_INSTRUCTION\_SIZE; //每一页的指令数目

public static int INTERRUPTION\_INTERVAL=10; //系统发生中断的间隔

public static int SYSTEM\_TIME=0; //系统内时间

public static void SystemTimeAdd() {kernel.SYSTEM\_TIME+=kernel.INTERRUPTION\_INTERVAL;} //系统时间自增

public static int TLB\_LENGTH=kernel.MEMORY\_USER\_SPACE\_SIZE/kernel.SINGLE\_PAGE\_SIZE/2; //TLB快表的长度，16

/\*系统基本信息\*/

/\*Process State 进程状态参数\*/

public final static short PROCESS\_READY = 0; //就绪态

public final static short PROCESS\_WAITING = 1; //等待态

public final static short PROCESS\_RUNNING = 2; //运行态

public final static short PROCESS\_SUSPENSION = 3; //挂起态

/\*Process State 进程状态参数\*/

/\*Process PSW 程序状态字\*/

public final static byte PSW\_KERNEL\_STATE=0; //管态

public final static byte PSW\_USER\_STATE=1; //目态

/\*Process PSW 程序状态字\*/

/\*硬件初始化需要变量\*/

public static String MEMORYFILE\_PATHNAME="./input/memory.dat"; //内存文件地址

public static String HARDDISKFILE\_PATHNAME="./input/harddisk.dat"; //硬盘文件地址

public static String CPUFILE\_PATHNAME="./input/cpu.dat"; //CPU文件地址

/\*硬件初始化需要变量\*/

/\*60条指令\*/

public static int GetInstructionType(int instruction){

//获取指令类型

if(instruction>=0&&instruction<=9)return 1;

if(instruction>=10&&instruction<=19)return 2;

if(instruction>=20&&instruction<=29)return 3;

if(instruction>=30&&instruction<=39)return 4;

if(instruction>=40&&instruction<=49)return 5;

if(instruction>=50&&instruction<=59)return 6;

return -1;}

public static int GetInstructionTime(int instruction)

{/\*获取指令执行所需要时间\*/return ((int)(instruction%10))\*10+20;}

public static int[] MUTEX= {-4,-3,-2,-1,0,1,2,3,4,5}; //临界区信号量,10个

public static int[] SYSTEM\_RESOURCE= {0,1,2,3,4,5,6,7,8,9}; //系统资源量，10个

public static int GetUseResourceNum(int instruction)

{/\*获取该指令所申请、释放的PV、资源所在的数组序号\*/return instruction%10;}

/\*60条指令\*/

}

**六、模块设计说明**

**（一）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()

//获得当前磁盘中的作业总数量

public int GetCurrentCreateJobID()

//获取当前要创建的作业的编号

public void SaveJobToHardDisk(JCB jcb)

//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 void GetJCBFromFile(File f)

//从文件中读取所有的JCB

public ArrayList<JCB> GetAllJCB()

//获取所有的JCB

public void NextJob()

//已经读取完一个作业，进入到下一个作业

public int GetNextJobNum()

//获取下一个将要被创建的JCB的序号

public boolean IsAllJobToProcess()

//检测是否还有作业没有变成进程

public void RefreshJobList()

//刷新作业后备队列

public boolean IsJobListEmpty()

//查看作业后备队列是否为空

}

**（二）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：

ProcessModule()//构造函数

public PCB TurnJCBToPCB(JCB jcb)

//将JCB变换为PCB

public void TransferJobCodeToSwapArea(JCB jcb,String apply)

//将指定作业的程序段存入虚存中

//jcb为作业控制块，apply为申请到的虚存空间分配字符串

public void WriteProcessPageTable(PCB pcb,String apply)

//将申请到的虚存页面写入到进程的页表中

public short GetFreePCBNumInPool()

//获取PCB池中可用的PCB数量

public void DeletePCBInPool(PCB pcb)

//将某一个PCB从PCB池中删除

public short ApplyOnePCBInPool()

//在PCB池中申请一个PCB

public void AddToPCBPool(PCB pcb)

//将PCB加入到PCB池中

public void TransferProcessToRunningQueue(PCB pcb)

//将进程移入运行队列

//遍历就绪队列、等待队列、挂起队列，将进程移出，只加入到运行队列

//移入运行队列

public void TransferProcessToReadyQueue(PCB pcb,boolean if\_active)

//将进程移入就绪队列

//遍历运行队列、等待队列、挂起队列，将进程移出，只加入到就绪队列

//移入就绪队列

public void TransferProcessToWaitQueue(PCB pcb)

//将进程移入等待队列

//遍历运行队列、就绪队列、挂起队列，将进程移出，只加入到等待队列

//移入等待队列

public void TransferProcessToSuspendQueue(PCB pcb)

//将进程移入挂起队列

//遍历运行队列、就绪队列、等待队列，将进程移出，只加入到挂起队列

//移入挂起队列

public void TransferProcessToEndQueue(PCB pcb)

//将进程移入完成队列

//遍历运行队列、就绪队列、等待队列、挂起队列，将进程移出，加入到完成队列

//移入完成队列

public void RefreshReadyQueueBitmap()

//刷新就绪队列的bitmap

public void RefreshActiveExpired()

//刷新active和expired指针

public boolean IfPageInMemory(short page\_num)

//检测需要的页是否在内存中

//检测是否发生缺页中断

public void ChangePageTable(short ori\_page\_num,short changed\_page\_num)

//将持有原来页的进程的页表更新

public void SolveMissingPage(PCB pcb,short need\_page\_num)

//缺页中断的处理，need\_page\_num为需要的在外存中的页的页号

public boolean IfRunOver(PCB pcb)

//检测某进程是否运行完毕

public boolean IfTimeSliceOver(PCB pcb)

//检测时间片是否用完

public void AddToEndQueue(PCB pcb)

//将作业加入到结束队列

public boolean IsRunningQueueEmpty()

//检测运行队列是否为空

public boolean IsReadyQueueEmpty()

//检测就绪队列是否为空

public boolean IsWaitQueueEmpty()

//检测等待队列是否为空

public boolean IsSuspendQueueEmpty()

//检测挂起队列是否为空

public int GetActivePoint()

//获取active指针

public int GetExpiredPoint()

//获取expired指针

public PCB GetPCBWithID(short id)

//根据进程ID获取PCB

public void RunType1(PCB pcb)

//类型1指令的处理

public void RunType2(PCB pcb)

//类型2指令的处理

public void RunType3(PCB pcb)

//指令类型3的处理

public void RunType4(PCB pcb)

//指令类型4的处理

public void RunType5(PCB pcb)

//指令类型5的处理

public void RunType6(PCB pcb)

//类型6指令的处理

}

**（三）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：

PageModule()//构造函数

{

InitSwapAreaUsage(); //初始化交换区页面被占用状态

for(int i=0;i<6;i++) //实例化伙伴算法的空闲链表

this.free\_area[i]=new ArrayList<Short>();

InitFreeAreaList(); //初始化空闲链表

RefreshBitmap(); //刷新伙伴算法的Bitmap

}

private void InitSwapAreaUsage()

//初始化交换区页的使用情况

private void InitFreeAreaList()

//初始化空闲链表

public void RefreshBitmap()

//刷新bitmap

private int SetOneBit(int num,int loca,int bit)

//修改bitmap中的某一位信息

private int GetOneBit(int num,int loca)

//获得bitmap中的某一位信息

public int GetFreePageNumInMemory()

//返回当前物理内存中可用的页框数

public int GetFreePageNumInDisk()

//返回当前虚存中可用的页框数

public Page GetPage(short num)

//获得某一个页面

public boolean IfCouldApplyPageInDisk(short num)

//检测在虚存中是否可以申请num个页面

public int CalculateCloest2Num(short num)

//计算出与该数字最接近的2的次幂数的幂

private void SetBlockState(int list,int no,int set\_num,int state)

//在伙伴算法的链表中，在第list级别的第no块设置连续的set\_num块为state状态

private void RefreshBlockList()

//从底往上刷新块链表

public String ApplyPageInMemory(short num)

//在内存中，向伙伴算法申请num个页面

public String ApplyPageInDisk(short page\_num)

//在虚存中申请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0-191，共192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

//同时，在记录数组中记录这些申请的页框，将他们设置为已用状态

public void FreePageInMemory(short page\_num)

//在内存中，利用伙伴算法释放某一页

public void FreePageInDisk(short page\_num)

//在外存中，释放某一页

public void RecyclePage(short page\_num)

//回收页面，参数num为页面号（num从0开始编号）

//将该页的内容全部清空，并在记录中使得该页表示为未被占用

public void MoveToMemory(short memory\_page\_num,short swap\_page\_num)

//将指定的虚存页移动到指定的内存页中

public void MoveToDisk(short memory\_page\_num,short swap\_page\_num)

//将指定的内存页移动到指定的虚存页中

public void ExchangePage(short memory\_page\_num,short swap\_page\_num)

//交换两个页面的内容

public void CopyPage(short src\_page\_num,short des\_page\_num)

//将序号src\_page\_num的页面复制到序号为des\_page\_num的页面中

public short GetOneFreePageInMemory()

//在物理内存中找到一个空闲的页面，并返回该页面序号

public short GetOneFreePageInDisk()

//在虚存中找到一个空闲的页面，并返回该页面序号

public void LRUVisitOnePage(int page\_num)

//LRU访问某一页

public short LRUGetLastPageNum()

//获得应该调出的页面号

public boolean isBlockUsing(int i,int j)

//获取内存中某一个页的使用情况

public boolean isPageUsing(int i)

//获取虚存中某一页的使用情况

}

**（四）调度算法**

调度算法是实现并发环境模拟的核心功能模块之一，分为高级调度、中级调度和低级调度三个层次，从不同方面保证了系统的稳定高效运行，为多任务并发执行创造条件。

public void run()

//线程执行函数，负责进行调度

public void UIRefresh()

//不同UI的刷新

public void SuspendProcessWithPageNum(short num)

//检测某一页所关联的进程，并将该进程加入到挂起态

public void HighLevelScheduling()

//高级调度

public void MiddleLevelScheduling()

//中级调度

public void LowLevelScheduling()

//低级调度

}

**（五）界面模块**

在程序执行之后，采用不同界面对系统的各个方面的信息进行展示，在监测系统状态的同时查看模拟系统的运行情况。

界面模块共分为2个部分，在MainUI中展示总体情况，并依托MainUI打开其他UI界面查看各UI所展示的信息。

**七、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**

│ .classpath

│ .project

│

│

├─input

│ cpu.dat

│ harddisk.dat

│ memory.dat

│

└─src

│ Main.java

│

├─hardware

│ AddressLine.java

│ CPU.java

│ DataLine.java

│ HardDisk.java

│ Memory.java

│ MMU.java

│ Timer.java

│

├─os

│ Control.java

│ DeadLock.java

│ JCB.java

│ JobModule.java

│ kernel.java

│ Page.java

│ PageModule.java

│ PCB.java

│ ProcessModule.java

│ Scheduling.java

│

└─ui

CPUInfoUI.java

CreateJobUI.java

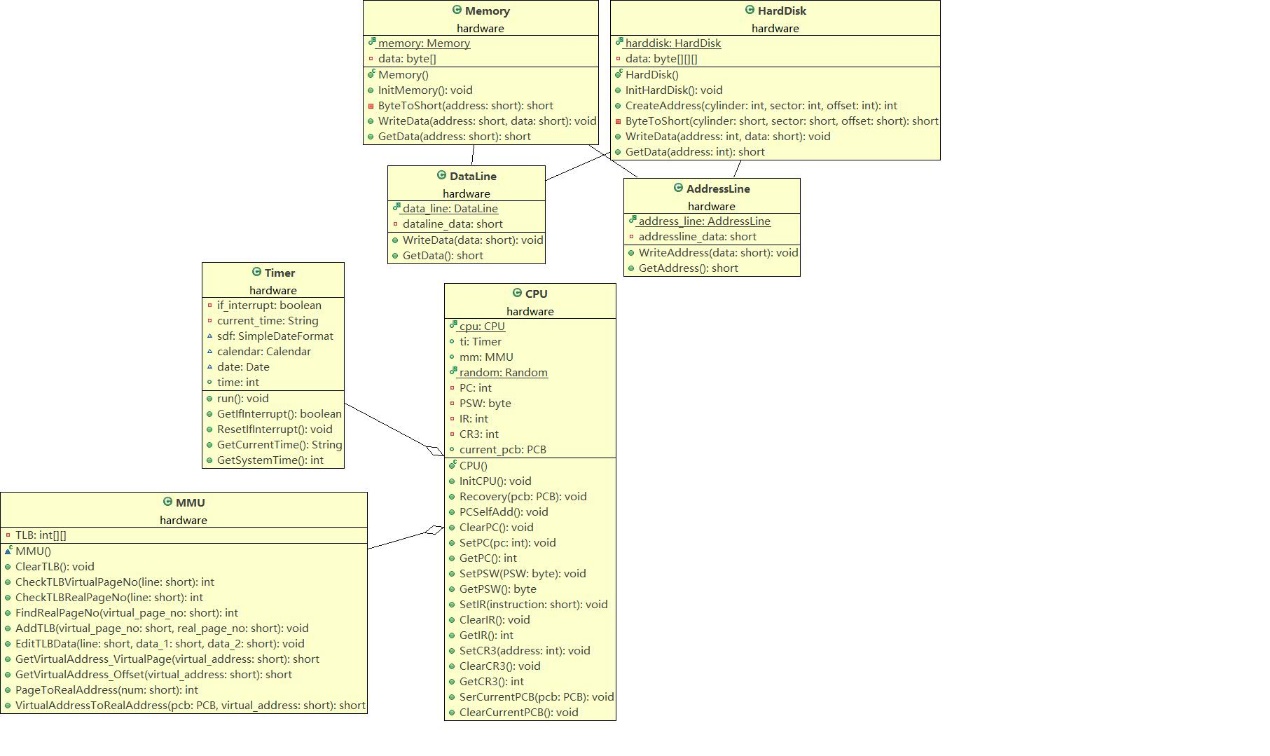
HardDiskUI.java

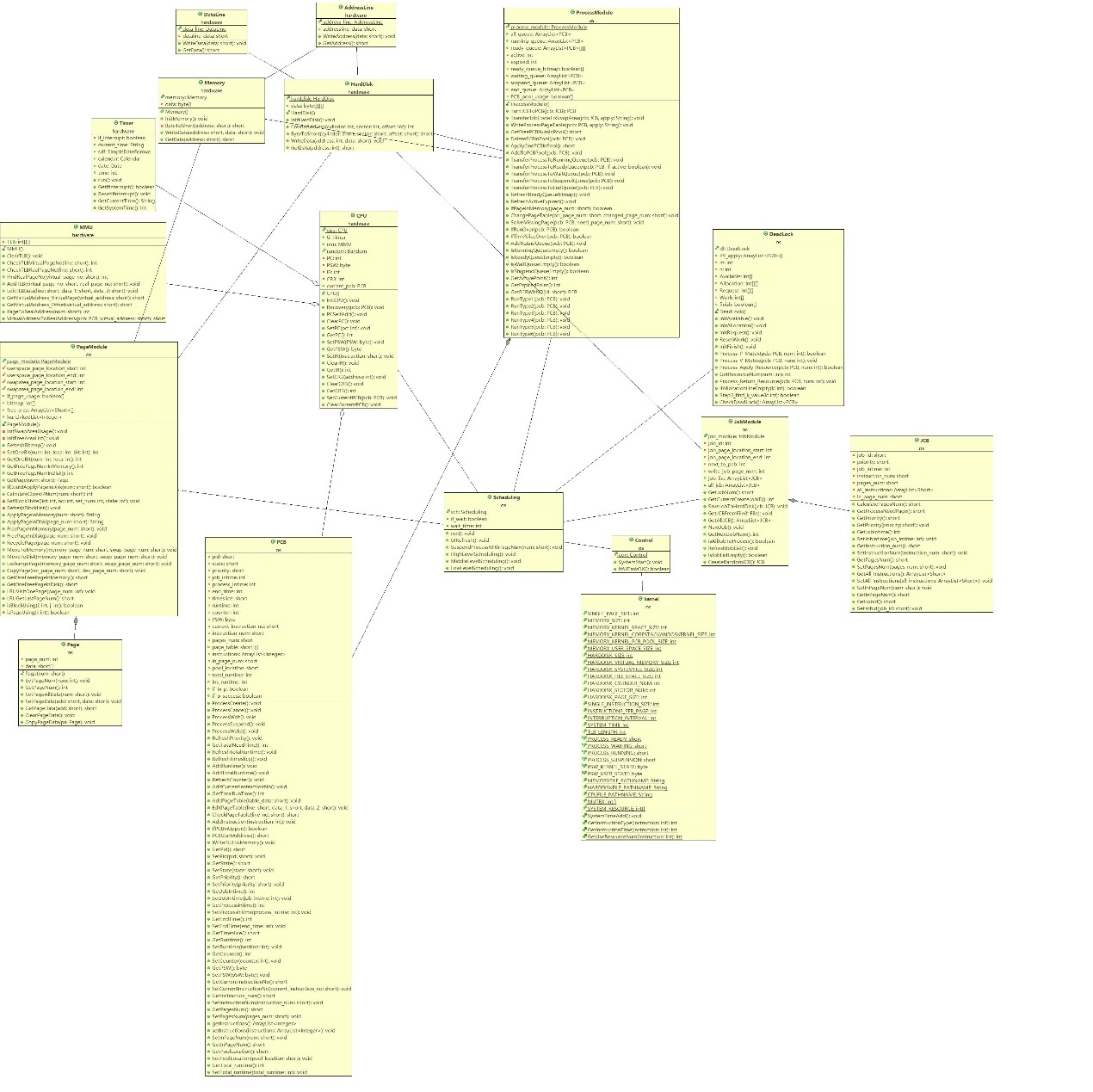
MainUI.java

MemoryUI.java

PageModuleUI.java

**附件2：类图说明**





注：类图也存储在在同级目录下的 **附件3-硬件类图.jpg** 和 **附件3-系统类图.jpg**