上海电力学院

操作系统原理课程设计



**课程名称：** **操作系统原理**

**题　　目: 虚拟分页存储管理模拟**

**学生姓名： 黄海琨 学 号： 20131809**

**班 级： 2013054 同组姓名： 顾珏**

**课程设计时间： 2015.5.29---2015.6.12**

**评 语：**

**成绩：**

**目录**

[一、 设计内容及要求 1](#_Toc424153350)

[二、 详细设计 1](#_Toc424153351)

[1） 原理概述 1](#_Toc424153352)

[2） 主要数据结构 3](#_Toc424153353)

[3） 算法（流程图） 3](#_Toc424153354)

[4） 源程序文件名 6](#_Toc424153355)

[三、 实验结果与分析 6](#_Toc424153356)

[四、设计总结 9](#_Toc424153357)

# 设计内容及要求

**设计内容：**

编写程序模拟实现虚拟分页存储管理的基本功能，包括内存的分配、内存的回收、地址变换，在发生缺页时采用页面置换算法。

要求显示每一次内存分配和回收后内存的使用状况，每一个进程占据的内存（页表），计算给定的逻辑地址对应的物理地址。

系统分配的物理块数由程序指定，支持以下两种页面置换算法：

（1）FIFO页面置换算法：

（2）LRU页面置换算法：

要求输入每个进程的页面引用序列，显示每一次页面引用内存状态，最后统计出页面置换算法页面置换的次数。

**小组分工情况：**

黄海琨：内存分配、内存回收、地址变换

顾珏：LRU算法和FIFO算法

# 详细设计

## 原理概述

**LRU（顾珏）：**

先进行内存的分配，建立一个辅助数组记录数据进入的时间。将程序运行序列中的页放入页表中直至页表满结束，并依次置于辅助数组中，同时调用Pro进行内容输出。若在页表满之前出现了已存在于页表之中的数据则不置入页表，但是将该数在辅助数组中调整至已放置数字的末尾，并将其后的数字依次向前提一位。

当页表已满时，若新进数字未存在，则替代当前辅助数组首位的数字，调用Pro进行内容输出，并将其置于辅助数组末位，其余数组内成员依次向前。若新进数字已存在，则将该数在辅助数组中调整至已放置数字的末位，并将其后的数字依次向前提一位。

**public** **class** Lru {

**public** **int**[] disk,t;

**public** **int**[][] page;

**public** **int** disknum,pagenum,len;

Pro pro =**new** Pro();

**public** Lru(**int** pagenum,**int** disknum){

**this**.disknum=disknum;

**this**.pagenum=pagenum;

**this**.len=pagenum\*pro.*memoryEachSize*;

**this**.disk=pro.initMemory(disknum);

**this**.t =**new** **int** [**this**.disknum];

**this**.page=**new** **int**[disknum][2];

**for**(**int** i=0;i<**this**.disknum;i++){

**this**.page[i][1]=**this**.disk[i];

**this**.page[i][0]=-1;

t[i]=-1;

}

}

**FIFO（顾珏）：**

当页表未满时，若数字未存在则放置于页表已存在数的后一位，并调用Pro进行内容输出，若已存在则跳过该数

当页表已满时，若数字未存在则从首位开始依次置换，并调用Pro进行内容输出，若已存在则跳过该数。

**public** **class** Fifo {

**private** **int**[] list;

**private** **int**[] disk;

**private** **int**[][] page;

**private** **int** disknum,pagenum;

Pro pro =**new** Pro();

**public** Fifo(**int** pagenum,**int** disknum){

**this**.disknum=disknum;

**this**.pagenum=pagenum;

**this**.list=pro.createList(pagenum,disknum);

pro.createMemory(pagenum);

**this**.disk=pro.initMemory(disknum);

}

**PRO（黄海琨）：**

先让用户输入程序的页数和物理块数，通过随机数来确定程序的运行序列和确定可运行的物理块号，把程序运行序列和物理块号传出给LRU或者FIFO算法。

接收LRU或者FIFO算法传出的需要页面置换的页号和物理块号，通过块号来修改内存物理块中的内容，计算物理块的物理地址，每次运行都输出内存的使用情况和页面置换的物理块的物理地址。

判断程序LRU或者FIFO算法是否进行完毕，如进行完毕就进行最后的内存回收。

**public** **class** Pro {

**public** **static** **int**[][] *memory*={{0,-1},{1,-1},{2,-1},{3,-1},{4,-1},{5,-1},{6,-1},{7,-1},{8,-1},{9,-1},};

**public** **int**[] memoryFree;

**public** **static** **int** *memoryEachSize*=1024;

## 主要数据结构

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 页号 | 逻辑地址 | 物理块号 | 状态位 | 物理地址 |
| Page | 十进制 | Disk |  | 十六进制 |

## 算法（流程图）





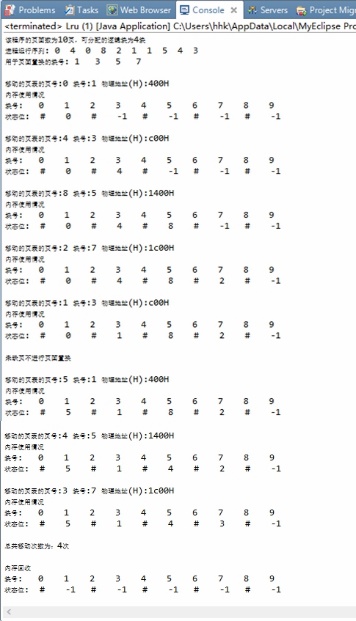


## 源程序文件名

20131805-1809虚拟分页存储管理模拟

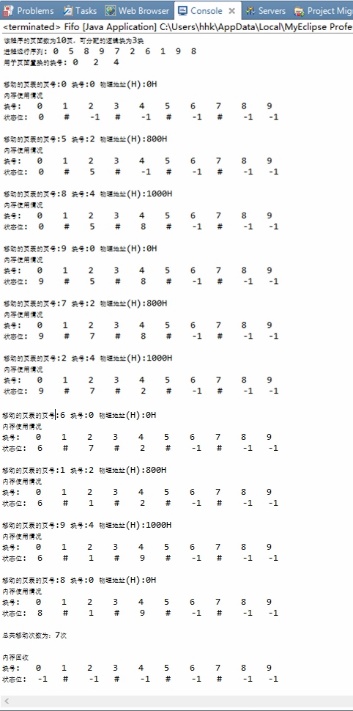
# 实验结果与分析

**LRU：**

****

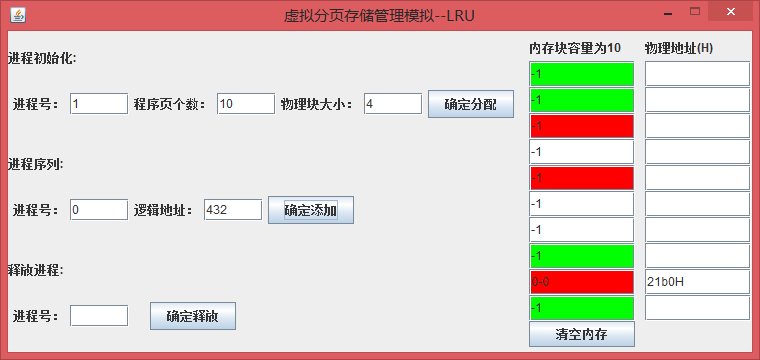
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 4 | 0 | 8 | 2 | 1 | 1 | 5 | 4 | 3 |
| 0 | 0 |  | 0 | 0 | 0 |  | 5 | 5 | 5 |
|  | 4 |  | 4 | 4 | 1 |  | 1 | 1 | 1 |
|  |  |  | 8 | 8 | 8 |  | 8 | 4 | 4 |
|  |  |  |  | 2 | 2 |  | 2 | 2 | 3 |

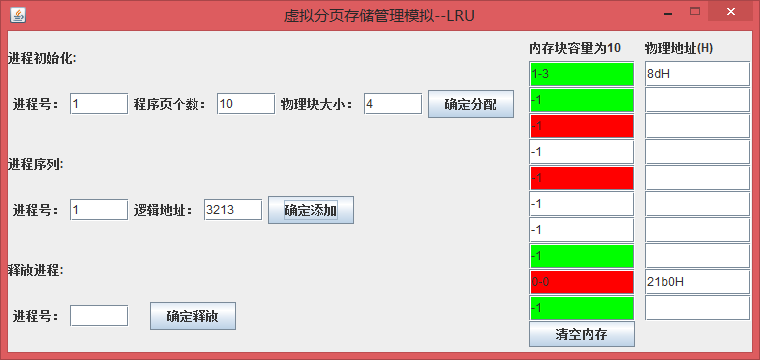
**FIFO：**

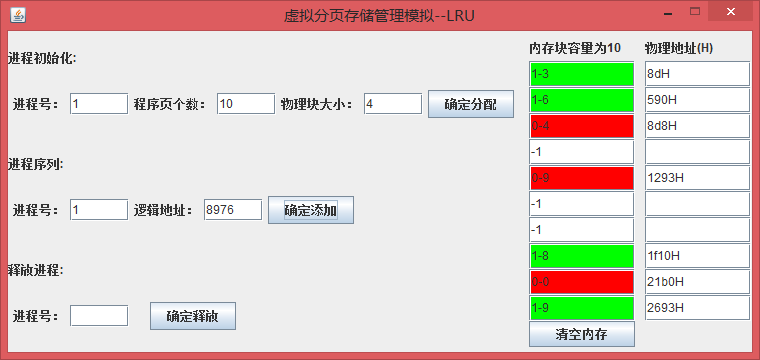
****

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 5 | 8 | 9 | 7 | 2 | 6 | 1 | 9 | 8 |
| 0 | 0 | 0 | 9 | 9 | 9 | 6 | 6 | 6 | 8 |
|  | 5 | 5 | 5 | 7 | 7 | 7 | 1 | 1 | 1 |
|  |  | 8 | 8 | 8 | 2 | 2 | 2 | 9 | 9 |

**LRU界面演示：**







# 四、设计总结

在本次课程设计中，我主要负责内存的分配回收和地址变换。在课程设计之前，我先要弄清楚虚拟分页存储管理模拟的整个过程。经过我和我同组成员的分析，虚拟分页存储管理模拟先是从模拟程序运行序列、内存分配开始的，接着再执行LRU或FIFO的算法，每一次页面置换都会产生地址变换，最后进行内存回收。

在一开始，Memory的一系列函数给LRU或FIFO算法提供程序运行序列和可用物理块号。途中LRU或FIFO的算法给Memory一系列函数要页面置换的页号和块号，接着地址变换。最后，程序结束内存回收。

在算法的编写过程中，有着第一次转入内存的页号操作是哪个算法实现的问题，最后得出结论是LRU或FIFO算法来实现的。