操作系统实验报告：实验三

姓名 陈学铄 学号 13061116

# 1需求说明

## 1.1基本要求和提高要求

基本要求：

支持页表、辅存和实存内容的打印

支持请求命令的手动输入（do\_request()函数是随机生成请求）

提高要求：

建立一个多级页表

实现多道程序的存储控制

将do\_request()和do\_response()实现在不同进程中，通过进程间通信（如FIFO）完成访存控制的模拟

实现其它页面淘汰算法：如页面老化算法、最近最久未使用淘汰算法（LRU）、最优算法（OPT）等

## 1.2完成情况

上述的基本要求和提高要求都已完成，其中，对于提高要求中的其他淘汰算法，完成的是最近最久未使用淘汰算法（LRU）。

# 2设计说明

## 2.1流程示意图

## 

## 2.2所使用的系统调用的列表

|  |  |
| --- | --- |
| 系统调用名称 | 功能简介 |
| Open() | 打开文件 |
| Close() | 关闭文件 |
| Read() | 读文件 |
| Write() | 写文件 |
| Mkfifo() | 创建FIFO文件 |
| Remove() | 删除文件 |
| Stat() | 获取文件信息 |

## 2.3提高要求实现说明

基础要求实现说明：

1. 打印页表，辅存、实存内容

* 打印页表源代码已经基本实现，增加了“最近访问时间”项来为LRU算法是否是否实现提供直观依据，因实现了二级页表而增加了“页目录”项。

代码实现：

/\* 打印页表 \*/

void do\_print\_info()

{

unsigned int i, j, k;

char str[4];

printf("页目录\t页号\t块号\t装入\t修改\t保护\t程序\t最近访问时间\t辅存\n");

for (i = 0; i < FIRST\_PAGE\_SUM; i++)

{

for(j = 0; j < SECOND\_PAGE\_SUM; j++)

{

printf("%u\t%u\t%u\t%u\t%u\t%s\t%d\t%u\t%u\n", i, j, pageTable[i][j].blockNum, pageTable[i][j].filled,

pageTable[i][j].edited, get\_proType\_str(str, pageTable[i][j].proType), (i < 2 ? 0 : 1) ,

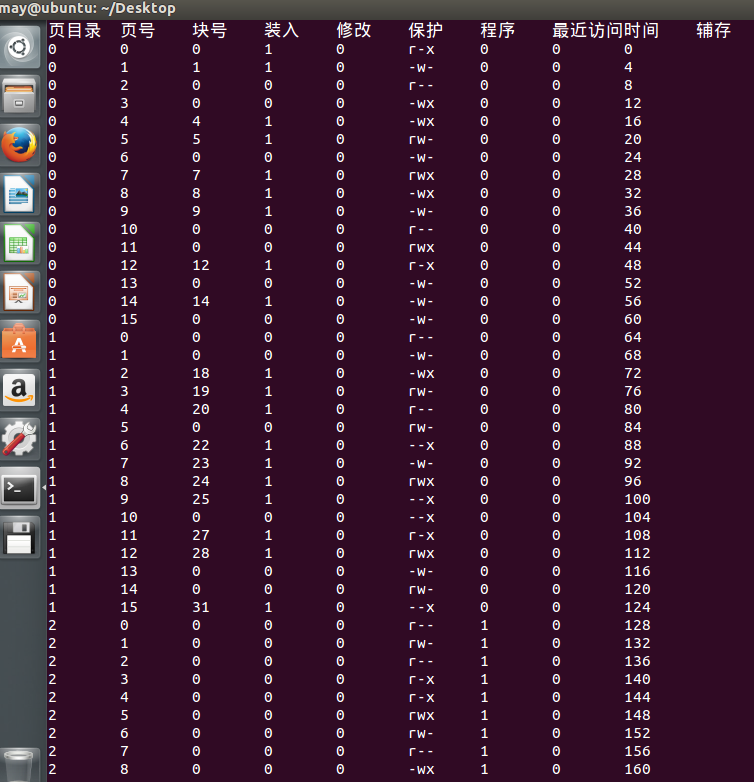
pageTable[i][j].lastVisitTime, pageTable[i][j].auxAddr);

}

}

}

运行结果（部分）：



* 打印辅存：打开文本文件，在每次循环中读取一个字节的内容，并打印，为了直观，打印的内容选择为这个字节对应的字符的ASCII码。

//打印辅存

void do\_print\_aux()

{

int ch, i = 0;

printf("即将开始打印辅存内容，以ASCII形式打印\n");

FILE\* new = fopen(AUXILIARY\_MEMORY,"r");

while(i < 256 && (ch = fgetc(new)) != EOF)

{

printf("[辅存号: %d 内容: %d]\t", i, ch);

if(i % 3 == 2)

puts("");

i++;

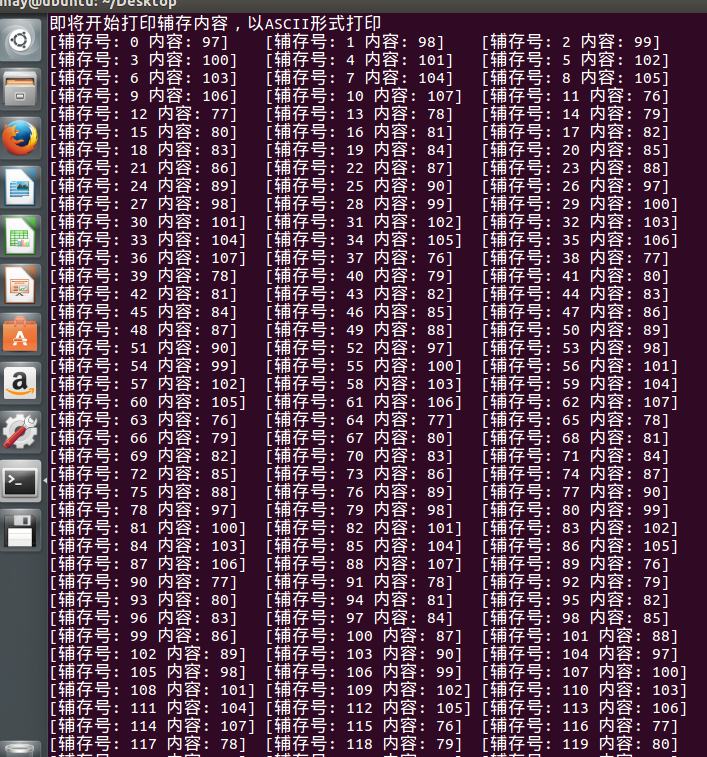
}

puts("");

}

代码实现：

运行结果（部分）：



* 打印实存：遍历一维字节数组，每次打印一个字节的内容，为了直观，打印这个字节对应字符的ASCII码。

代码实现：

//打印实存

void do\_print\_actMem()

{

int i;

printf("即将开始打印实存内容，以ASCII形式打印\n");

for(i = 0; i < ACTUAL\_MEMORY\_SIZE; i++)

{

printf("[实存号: %d 内容: %d]\t", i, actMem[i]);

if(i % 3 == 2)

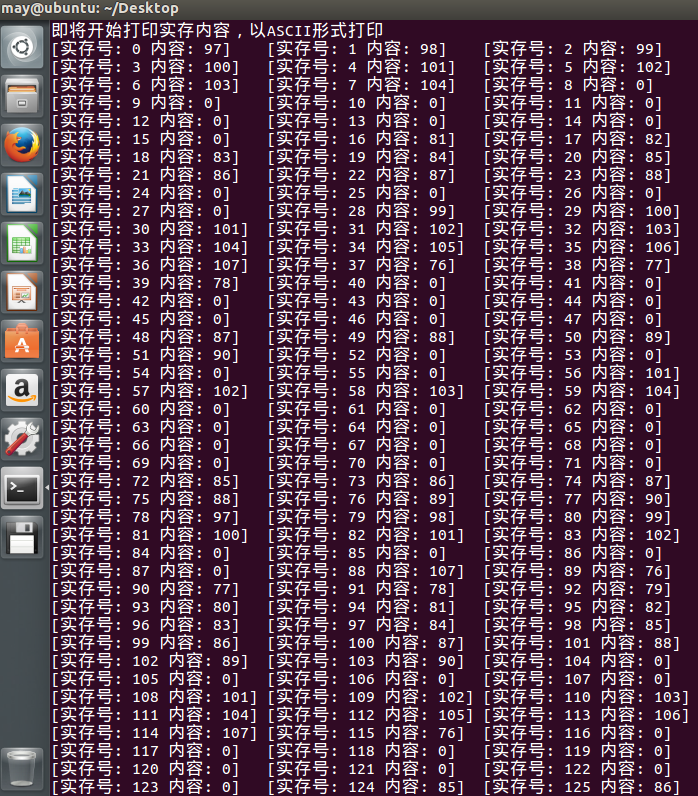
puts("");

}

puts("");

}

运行结果（部分）：



1. 支持请求命令的手动输入

将手动命令和随机命令全部交给用户选择。用户在指派一条请求命令之前，首先选择这个命令是手动还是随机。输入“P”表示手动（Personal），输入“R”表示随机（Random）。程序识别到输入的是“P”继续要求用户输入访存类型（读／写／执行），访问的虚存地址，如果是写入请求的话，要求用户输入写入的数值，程序会读取这些值并存放到请求命令结构体的相应域中。如果是“R”的话，则由随机函数生成这些数据。

关键部分代码实现：

while(1)

{

printf("请输入访存请求的类型。P代表自定义请求，R代表随机请求\n");

if(getchar() == 'P')

{

req.program = random() % 2;

printf("请输入访存类型。0代表读出， 1代表写入，2代表执行\n");

scanf("%d", &temp);

switch(temp)

{

case 0:

req.reqType = REQUEST\_READ;

break;

case 1:

req.reqType = REQUEST\_WRITE;

break;

case 2:

req.reqType = REQUEST\_EXECUTE;

break;

default:

printf("非法访存类型。正确格式为：0代表读出， 1代表写入，2代表执行\n");

flag = 1;

break;

}

if(! flag)

{

printf("请输入要访问的虚存地址，虚存地址要小于32\*4=128\n");

scanf("%d", &item);

req.virAddr = item;

if(temp == 1)

{

printf("请输入要写入的数值\n");

scanf("%d", &item);

req.value = item % 0xFFu;

printf("产生请求：\n程序:%d\t地址：%u\t类型：写入\t值：%d\n", req.program, req.virAddr, req.value);

}

else if(temp == 0)

printf("产生请求：\n程序:%d\t地址：%u\t类型：读取\n", req.program, req.virAddr);

else

printf("产生请求：\n程序:%d\t地址：%u\t类型：执行\n", req.program, req.virAddr);

}

getchar();

}

else

{

getchar();

req.program = random() % 2;

req.virAddr = random() % ONE\_PROGRAM\_SIZE;

switch (random() % 3)

{

case 0:

{

req.reqType = REQUEST\_READ;

printf("产生请求：\n程序:%d\t地址：%u\t类型：读取\n", req.program, req.virAddr);

break;

}

case 1:

{

req.reqType = REQUEST\_WRITE;

req.value = random() % 0xFFu;

printf("产生请求：\n程序:%d\t地址：%u\t类型：写入\t值：%d\n", req.program, req.virAddr, req.value);

break;

}

case 2:

{

req.reqType = REQUEST\_EXECUTE;

printf("产生请求：\n程序:%d\t地址：%u\t类型：执行\n", req.program, req.virAddr);

break;

}

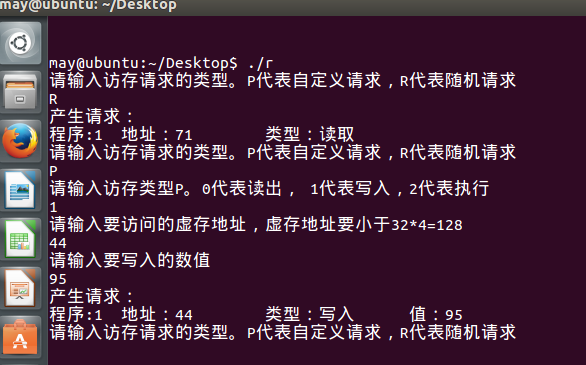
default:

break;

}

}

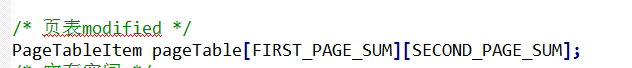
运行结果：



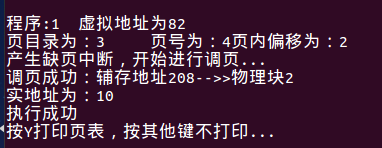
提高要求实现说明：

1. 建立一个多级页表

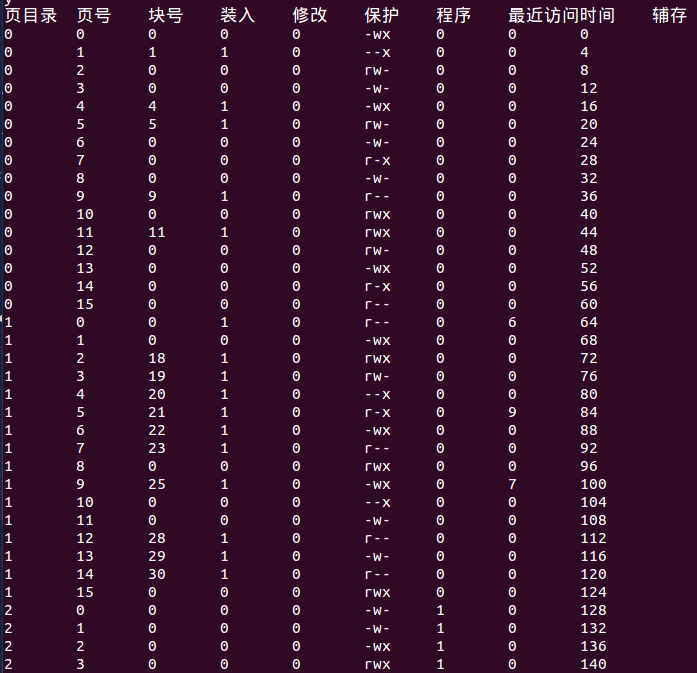
在这里，用一个二维数组表示二级页表，如下：



这样，就有了多级页表



打印出的页表形式如下:



1. 实现多道程序的存储控制

在该实验中，采取的是两道程序的自动选择，其中在request.c中加入新的请求时就采用如下方式来自动生成该请求输入哪一个程序，而对于多级页表，其中的一级页表的目录0和1属于0号程序，2和3属于1号程序，在进行缺页调度时，会选择相同程序号的物理块进行调度。



1. 将do\_request()和do\_response()实现在不同进程中，通过进程间通信（如FIFO）完成访存控制的模拟

**文字描述**

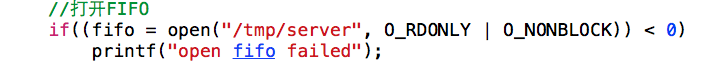
* 建立FIFO：在vmm.c的main函数里使用mkfifo函数建立一个FIFO。
* 打开FIFO：在vmm.c的main函数里打开这个FIFO文件以获取文件描述符，赋给全局变量fifo。
* 写入FIFO：将do\_request()方法从vmm.c中独立出来作为一个单独的request.c文件，在该文件的main函数中，每进行一次读取命令循环后便打开FIFO文件，使用write函数向里面写入请求结构体的内容。
* 读取FIFO：在vmm.c的do\_response函数里，每次该函数被调用，首先在函数内使用全局变量fifo来读取FIFO文件，每次读出一个请求结构体，加以处理

**代码实现（具体代码请根据提示查看源文件）**

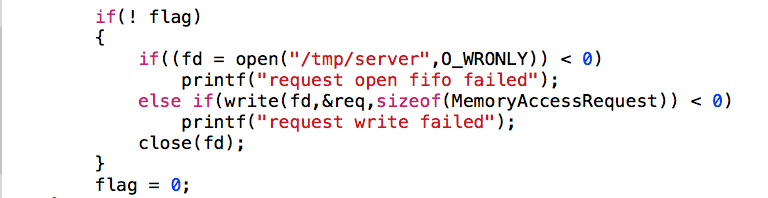
* 建立FIFO（vmm.c的main函数的开始处）：



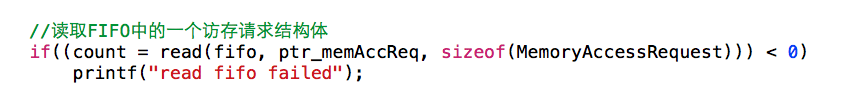
* 打开FIFO（vmm.c的main函数的中部）：



* 写入FIFO（request.c文件的末尾）：



* 读取FIFO（vmm.c文件的do\_response函数的开头）：



1. 实现LRU算法

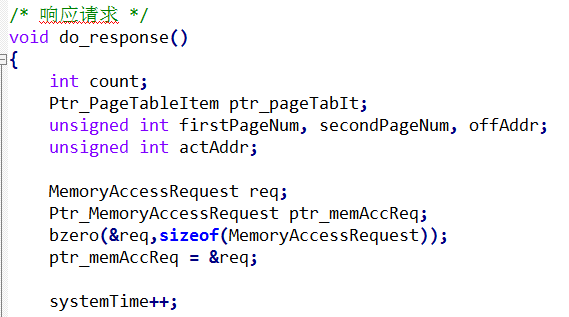
首先，对页表项加入一个

来记录该页表项的“最近使用时间”，且添加一个全局变量



来记录当前“时间”

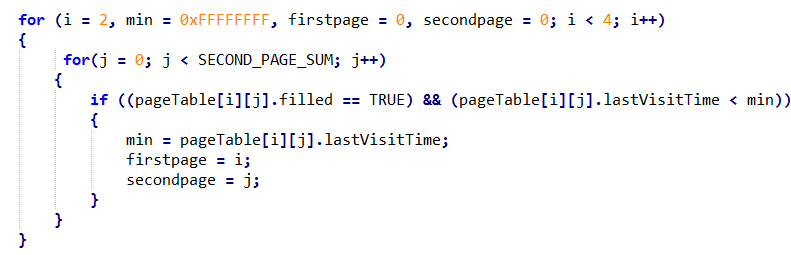
每次响应请求时，systemTime都会加一



且被访问到的页表项的lastVisitTime会置为systemTime

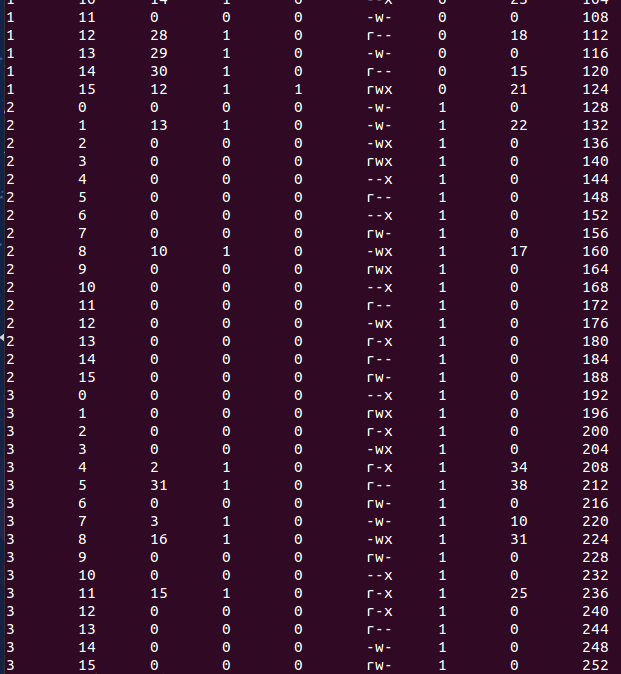


当进行缺页调度时，会选择最小的lastVisitTime的页表项



实验结果截图如下

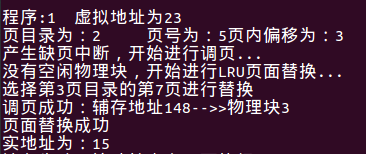
在进行LRU替换前，页表情况如下



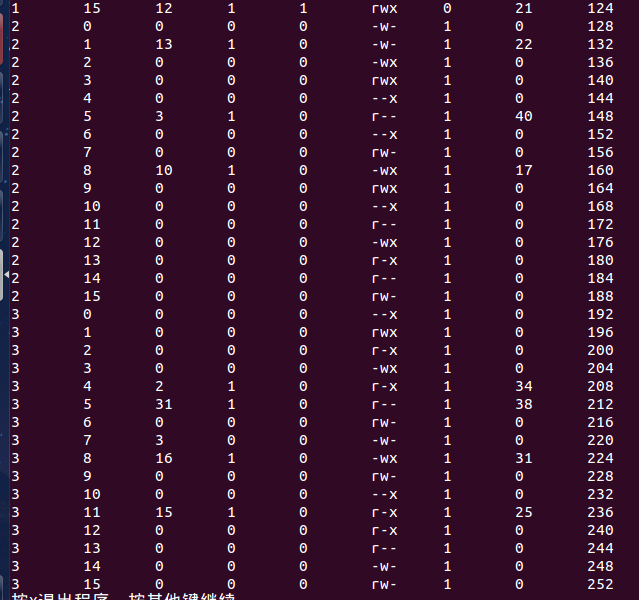
此时，看到程序1的占用的物理块中最近访问时间里最小的是



此时，进行LRU替换时，会把该页替换



替换后的页表情况如下：



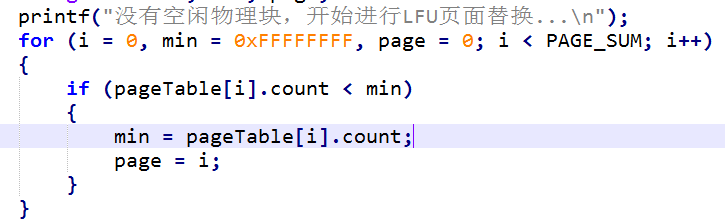
# 3收获与感想

## 3.1给予你帮助的人

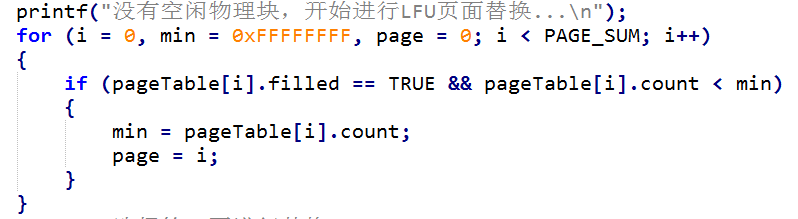
在本次实验中，大多都是自己组员之间相互讨论完成。而在实现要求时，也发现了源码中的一个错误(该错误将在下文中提到)，此时，与其他组成员进行了讨论。

## 3.2从实验中学到的东西

在本次实验前，自身本以为对存储管理有一定的理解，认为对书本内容中提到的页表结构、实存、辅存之间的关系已经了解。但是，在完成实验过程中，却发现了自身的诸多不足之处，纸上得来终觉浅，而在实验后，才对这些知识有了进一步的理解。并且，在实验中，也发现了源代码中的一个错误之处：



在源代码中的LFU算法中，直接选取了最小的pageTable[i].count所对应的页表项，但是，此页表项却不一定对应了一个物理块，由于页表项有64个，而物理块只有32块，那么，页表中一定是至少有32个页表项的“count”等于0，因而，该算法并未正确替换。而纠正次算法要在“if (pageTable[i].count < min)”中增加一个判断条件” pageTable[i].filled == TRUE”，修改后代码如下:



在此次实验中，从运行源代码时发现错误到最终找到此错误处也让我对整个程序有了更深的理解。