**操作系统课程设计实验报告**

——实验三：v m m 实验

负责人姓名：林之力

学号：14061133

日期：2016.5.7

**小组成员**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 姓名 | 学号 | 实验分工 |
| 1 | 李子涵 | 14061145 | 实验一 |
| 2 | 陈斌 | 14061144 | 实验二 |
| 3 | 林之力 | 14061133 | 实验三 |
| 4 | 许楷舟 | 14061141 | 实验四 |

目录

[1.实验目的 4](#_Toc446001831)

[2.需求说明 4](#_Toc446001832)

[2.1基本要求 4](#_Toc446001833)

[2.2 提高要求 4](#_Toc446001834)

[2.3 完成情况 4](#_Toc446001835)

[3.设计说明 5](#_Toc446001836)

[3.1 程序流程图 5](#_Toc446001837)

[3.2基本要求实现说明 5](#_Toc446001838)

[3.3 提高要求实现说明 5](#_Toc446001839)

[4.收获和感想 5](#_Toc446001840)

# 1.实验目的

1.了解Linux的内存管理机制

2.掌握页式虚拟存储技术

3.理解虚地址到实地址的定位过程

4.掌握相关页面淘汰算法

# 2.需求说明

## 2.1基本要求

1. 支持页表辅存、实存内容打印
2. 支持命令请求手动输入
3. 正确进行地址转换
4. 正确进行页面装入和页面交换
5. 正确进行存储控制
6. 正确执行读写命令

## 2.2 提高要求

1. 建立一个多级页表
2. 实现多道程序的存储控制
3. 通过进程间通信完成访存控制的模拟
4. 实现其他页面淘汰算法

## 2.3 完成情况

完成了以下功能：

1.支持页表辅存、实存内容打印

2.支持命令请求手动输入

3.正确进行地址转换

4.正确进行页面装入和页面交换

5.正确进行存储控制

6.正确执行读写命令

7.建立一个二级页表

8.实现多道程序的存储控制

9.实现LRU页面淘汰算法

10.FIFO完成访存控制的模拟遇到了困难，总是调不好，最终没能实现，仅在报告中给出了基本思路和相关代码。

# 设计说明

PS.例会记录

1. 第一周课后，召开小组讨论会。在源码已经实现了所有基本要求的基础上，大家一致同意先通读源码，理解源码的设计思路，以便开展后续的实验。
2. 第二周给出了提高要求的实现，小组讨论后交换了对于源码的理解，各自分工负责对提高要求进行实验设计。
3. 第三周汇总各自的成果，整合程序，对实验内容进行综合测试。对于存在的许多BUG和与预期不符的结果，进行修正。

## 3.1 程序流程图

1.初始化环境

2.打印页表

3.产生请求

4.响应请求

5.循环进行2-4

## 3.2基本要求实现说明

1. 页表辅存、实存内容打印

/\* 打印内存 \*/

void do\_print\_actMem()

{

int i;

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

{

printf("%c",actMem[i]);

}

printf("\n");

}

/\* 打印外存 \*/

void do\_print\_auxMem()

{

char c;

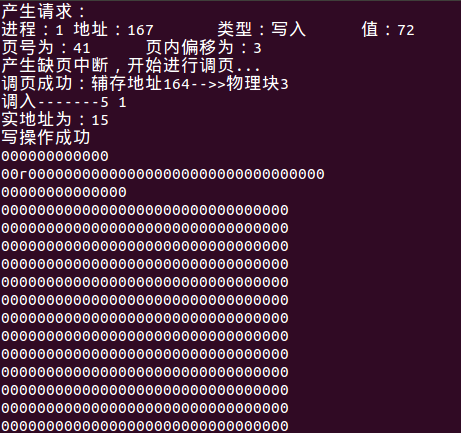
fseek(ptr\_auxMem, 0, SEEK\_SET);

while(fscanf(ptr\_auxMem,"%c",&c)!=EOF)

printf("%c",c);

printf("\n");

}



1. 命令请求手动输入

/\* 手动产生访存请求 \*/

void do\_myrequest()

{

/\* 请求地址 \*/

int virAddr;

printf("请输入请求地址\n");

scanf("%d ",&virAddr);

ptr\_memAccReq->virAddr = virAddr % VIRTUAL\_MEMORY\_SIZE;

/\* 请求所属进程号 \*/

int processNum;

printf("请输入请求所属进程号\n");

scanf("%d ",&processNum);

ptr\_memAccReq->processNum = processNum;

/\* 请求类型 \*/

int reqType;

printf("请输入请求类型\n");

scanf("%d ",&reqType);

switch (reqType % 3)

{

case 0: //读请求

{

ptr\_memAccReq->reqType = REQUEST\_READ;

printf("产生请求：\n进程：%u\t地址：%u\t类型：读取\n", ptr\_memAccReq->processNum,ptr\_memAccReq->virAddr);

break;

}

case 1: //写请求

{

ptr\_memAccReq->reqType = REQUEST\_WRITE;

/\* 待写入的值 \*/

int value;

printf("请输入待写入的值\n");

scanf("%d ",&value);

ptr\_memAccReq->value = value % 93 + 33;

printf("产生请求：\n进程：%u\t地址：%u\t类型：写入\t值：%02X\n", ptr\_memAccReq->processNum,ptr\_memAccReq->virAddr, ptr\_memAccReq->value);

break;

}

case 2:

{

ptr\_memAccReq->reqType = REQUEST\_EXECUTE;

printf("产生请求：\n进程：%u\t地址：%u\t类型：执行\n", ptr\_memAccReq->processNum,ptr\_memAccReq->virAddr);

break;

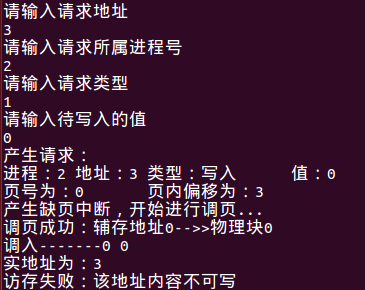
}

default:

break;

}

}



3.其他基本要求源码中已经实现

## 3.3 提高要求实现说明

1. 建立一个多级页表

/\* 二级页表项 \*/

typedef struct

{

unsigned int pageNum;

unsigned int blockNum; //物理块号

BOOL filled; //页面装入特征位

BYTE proType; //页面保护类型

BOOL edited; //页面修改标识

unsigned long auxAddr; //外存地址

unsigned long count; //页面使用计数器

unsigned long processNum;//进程号

} PageTableItem, \*Ptr\_PageTableItem;

/\* 一级页表项 \*/

typedef struct

{

unsigned int aNum;

PageTableItem pageTable[APAGE\_SUM];

} FPageTableItem;

/\* 一级页表数 \*/

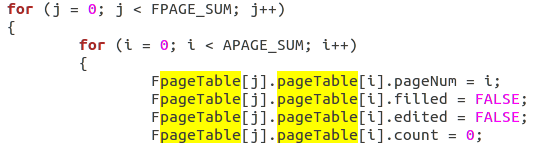
#define FPAGE\_SUM 8

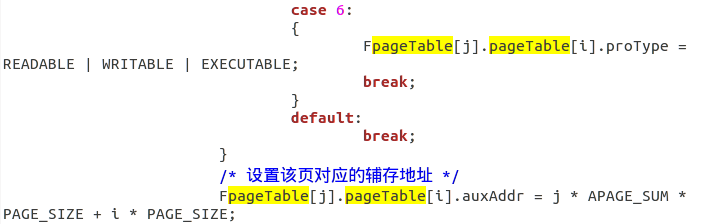
/\* 一级页表指向的二级页表数 \*/

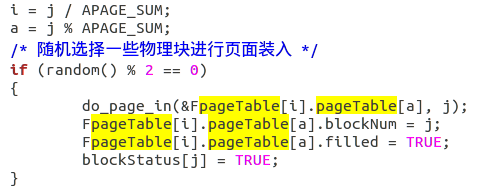
#define APAGE\_SUM (PAGE\_SUM / FPAGE\_SUM)

在整个程序中涉及到页表项的地方都进行了相应的修改。具体如下。

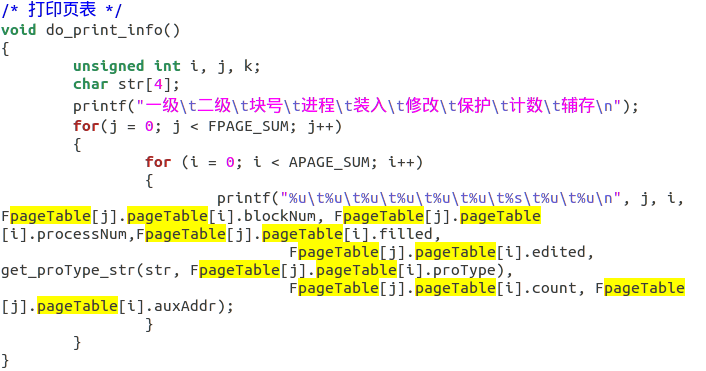








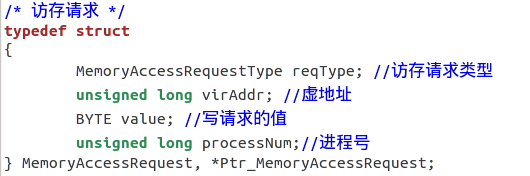




1. 实现多道程序的存储控制

在PageTableItem和MemoryAccessRequest结构中增加属性unsigned long processNum代表所属的进程号。





在页表中随机标注某页属于哪个进程。

/\* 使用随机数设置该页所属进程 \*/

switch (random() % 3)

{

case 0:

{

FpageTable[j].pageTable[i].processNum = 0;

break;

}

case 1:

{

FpageTable[j].pageTable[i].processNum = 1;

break;

}

case 2:

{

FpageTable[j].pageTable[i].processNum = 2;

break;

}

default:

break;

}

产生访问请求的时候，生成属于不同进程的请求。

/\* 请求所属进程号 \*/

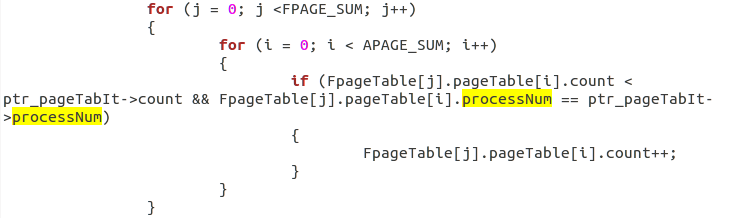
int processNum;

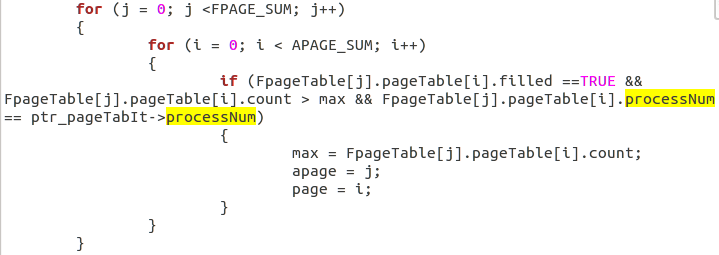
printf("请输入请求所属进程号\n");

scanf(" %d",&processNum);

ptr\_memAccReq->processNum = processNum;

读写内存时进行权限判断





3.实现LRU页面淘汰算法

LRU的实现（计数器法）  
每个页表都设置一个计数器；  
访问命中时，所有页表的计数值与命中页表的计数值进行比较，如果计数值小于命中的页表的计数值，则该页表的计数值加1；如果计数值大于命中的页表的计数值，则数值不变。最后将命中的页表的计数器清为0。  
访问未命中，需要替换时，则选择已装入的页中计数值最大的页表被替换。被替换页表的计数器清0，而其它的计数器则加1。

/\* 根据特征位决定是否产生缺页中断 \*/

if (!ptr\_pageTabIt->filled)

{

do\_page\_fault(ptr\_pageTabIt);

printf("调入-------%d %d\n",a,b);

} else {

for (j = 0; j <FPAGE\_SUM; j++)

{

for (i = 0; i < APAGE\_SUM; i++)

{

if (FpageTable[j].pageTable[i].count < ptr\_pageTabIt->count && FpageTable[j].pageTable[i].processNum == ptr\_pageTabIt->processNum)

{

FpageTable[j].pageTable[i].count++;

}

}

}

ptr\_pageTabIt->count = 0;

}

/\* 根据LRU算法进行页面替换 \*/

void do\_LRU(Ptr\_PageTableItem ptr\_pageTabIt)

{

unsigned int i, j, max=0, apage, page;

printf("没有空闲物理块，开始进行LRU页面替换...\n");

for (j = 0; j <FPAGE\_SUM; j++)

{

for (i = 0; i < APAGE\_SUM; i++)

{

if (FpageTable[j].pageTable[i].filled ==TRUE && FpageTable[j].pageTable[i].count > max && FpageTable[j].pageTable[i].processNum == ptr\_pageTabIt->processNum)

{

max = FpageTable[j].pageTable[i].count;

apage = j;

page = i;

}

}

}

printf("选择第%u个一级页表的第%u页进行替换\n", apage, page);

if (FpageTable[apage].pageTable[page].edited)

{

/\* 页面内容有修改，需要写回至辅存 \*/

printf("该页内容有修改，写回至辅存\n");

do\_page\_out(&FpageTable[apage].pageTable[page]);

}

FpageTable[apage].pageTable[page].filled = FALSE;

printf("拿走-------%d %d\n",apage,page);

FpageTable[apage].pageTable[page].count = 0;

/\* 读辅存内容，写入到实存 \*/

do\_page\_in(ptr\_pageTabIt, FpageTable[apage].pageTable[page].blockNum);

/\* 更新页表内容 \*/

for (j = 0; j <FPAGE\_SUM; j++)

{

for (i = 0, page = 0; i < APAGE\_SUM; i++)

{

FpageTable[j].pageTable[i].count++;

}

}

ptr\_pageTabIt->blockNum = FpageTable[apage].pageTable[page].blockNum;

ptr\_pageTabIt->filled = TRUE;

ptr\_pageTabIt->edited = FALSE;

ptr\_pageTabIt->count = 0;

printf("页面替换成功\n");

}

4.FIFO实现思路及部分代码

原程序中通过do-request随机产生一个Ptr\_MemoryAccessRequest结构ptr\_memAccReq，并设置其相应属性，再通过do-response响应请求。

（1）先使用以下代码在vvm.c的do\_init中建立一个FIFO文件：

if(stat("/tmp/server",&buf)==0){ // 如果FIFO文件存在,删掉

if(remove("/tmp/server")<0)

error\_sys("remove failed");

}

// 在非阻塞模式下打开FIFO

if((fifo=open("/tmp/server",O\_RDONLY|O\_NONBLOCK))<0)

error\_sys("open fifo failed");

（2）将do-request函数独立出来作为一个do\_request.c文件，运行./ do-request即可。

在do\_request.c中先打开FIFO，再将结构体写入FIFO文件中，等待读取

if((fifo\_stat=open("/tmp/server\_stat",O\_WRONLY|O\_NONBLOCK))<0)

error\_sys("stat open fifo\_stat failed");

if(ptr\_memAccReq==NULL)

single\_fifo = 1;

/\*将ptr\_memAccReq写入FIFO中\*////

if(write(fifo, ptr\_memAccReq,sizeof(structPtr\_MemoryAccessRequest))<0)

error\_sys("stat write failed");

（3）将do-response函数独立出来作为一个do\_response.c文件，运行./ do- response即可。

在do\_response.c使用FIFO读取Ptr\_MemoryAccessRequest结构

设置single\_fifo是为了防止FIFO为空使得读取失败的情况。

if((fd =open("/tmp/server ",O\_RDONLY))<0)

error\_sys("stat open fifo failed");

while((count=read(fd,& ptr\_memAccReq,sizeof(struct Ptr\_MemoryAccessRequest)))<=0) {

if(single\_fifo = 1) {

single\_fifo = 0;

close(fd);

close(fd\_stat);

return 0;

}

}

do{

}while((count=read(fd,& ptr\_memAccReq,sizeof(struct Ptr\_MemoryAccessRequest)))>0);

-FIFO参数问题

刚开始使用FIFO只看了函数的大概参数描述就开始写程序，调试的时候一直段错误，segmentation fault，后来查阅文档发现

read(fd,& ptr\_memAccReq,sizeof(struct Ptr\_MemoryAccessRequest)的第二个参数是一个指针而不是一个结构，所以要将结构取值再传入，加&符错误消除。

-ptr\_memAccReq为空问题

当ptr\_memAccReq为空时，FIFO会读取错误，显示segmentation fault

在.h文件中添加single\_fifo标志位，ptr\_memAccReq为空时置1，当single\_fifo=1时，FIFO将直接关闭而不读取，避免读取错误。

# 4.收获和感想

之前都是跟着别人做的，这次第一次负责，有些紧张啊。对于这次实验的二级页表，不是很明白。查阅了网上了相关资料，并翻阅了去年的计算机组成课件的相关章节，懂得了页表反映了虚拟地址到物理地址的映射。

在建立二级页表的时候遇到了困难，向同学请教后清晰了基本思路是将原先的一级页表项改成二级页表项，然后再建一个一级页表的结构，存储一级页表号并指向对应的二级页表，通过一个结构数组来实现。如果原先的一级页表看作一个一维数组，那么二级页表可以视为二维数组，这么理解起来就好实现多了。

建立完二级页表后，需要对程序中所有涉及页表的部分进行修改，第一遍好像漏改了几处，出现了各种数组越界。后来从头到尾ctrl+F查找了相关项，一项一项修正好了。

在测试页面替换算法的时候，也遇到了一些奇怪的情况。在执行多次以后，装入位基本全变成1了，已经超过实存的32容量。经过DEBUG发现是在页面调出的时候只判断了计数器的大小，而忽略了页面应该在装入状态才能被调出，导致未装入的页面被调出（实际根本没有装入），最后所有页面都被装入了。

解决方法是在判断条件中加入FpageTable[j].pageTable[i].filled ==TRUE，经测试结果符合预期要求。

FIFO这块我不是很懂，是由其他组员负责的，期间一直有bug，在查阅相关文档和交流讨论后消除了一些error，遗憾的是最终没能实现要求，不过这个过程还是让我们的能力得到了提高。

经过这次实验，我懂得了团队协作的重要性，很多时候你修改了一块代码，会影响到其他人代码的执行，所以要及时分享最新更改的代码，以防最终整合的时候BUG过多，不好定位。