**操作系统课程设计实验报告**

——实验三：虚存管理实验

负责人姓名：滕叶飞

学号：14061189

日期：2016.5.4

**小组成员**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 姓名 | 学号 | 实验分工 |
| 1 | 陈彦吉 | 14061195 | 实验一 |
| 2 | 叶青河 | 14061202 | 实验二 |
| 3 | 滕叶飞 | 14061189 | 实验三 |
| 4 | 韩青长 | 14061183 | 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，理解虚地址到实地址的定位过程

3.掌握“最不频繁使用淘汰算法”，即LFU页面淘汰算法。

# 2.需求说明

## 2.1基本要求

1.设计并实现一个虚存管理模拟程序，模拟一个单道程序的页式存储管理，用一个一维数组模拟实存，用一个文本文件模拟辅存空间。

2.建立一张一级页表。

3.程序中使用do\_request()随机产生访问请求，访存操作包括读取，写入，执行三种类型。

4.实现do\_response()响应访存请求，完成虚地址到实地址的定位以及读写执行操作，同时判断处理缺页中断。

5.实现LFU页面淘汰算法。

## 2.2 提高要求

1.实现多道程序的存储控制。

2.建立一张多级页表或者快表。

3.将do\_request()和do\_response()是现在不同进程中通过进程间通信(FIFO)完成访存控制的模拟。

4.实现其他页面淘汰算法，如页面老化算法。

## 2.3 完成情况

【简述实验完成过程】。完成了以下功能：

1. 实现了最多四个程序的存储控制。

2. 建立了一张8\*8的二级页表。

3. 实现了不同进程中访存请求控制，并按照FIFO原则处理访存请求

4. 实现了页面老化算法。

5. 支持手动添加访存请求。

6. 支持页表，辅存，内存的打印。

7. 基本要求明确的所有功能。

# 3.设计说明

## 3.1 程序流程图

开始

产生缺页中断

多程序产生请求

主程序环境初始化

有空闲块

从FIFO文件中读取请求

先后顺序写入FIFO文件

N Y

地址越界

淘汰使用次数最少的页面

获取相应页表

页面有修改

N

计算页目录页号页内偏移值

N Y

页面在实存

写回至辅存

Y

更新页表

获得块号，计算实地址

有访问权限

N

装入页面

Y

返回错误信息

更新页表

返回访存结果

程序结束

N

Y

结束

## 3.2基本要求实现说明

1.void do\_init()。

功能：初始化整个辅存空间，为辅存空间的每一页设置随机保护类型，并且将辅存空间的某些页面随机地装入内存。

void do\_init()

{

int i, j;

srand(time(NULL));

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

{

pageTable[i].pageNum = i;

pageTable[i].filled = FALSE;

pageTable[i].edited = FALSE;

pageTable[i].count = 0;

/\* 使用随机数设置该页的保护类型 \*/

switch (rand() % 7)

{

case 0:

{

pageTable[i].proType = READABLE;

break;

}

case 1:

{

pageTable[i].proType = WRITABLE;

break;

}

case 2:

{

pageTable[i].proType = EXECUTABLE;

break;

}

case 3:

{

pageTable[i].proType = READABLE | WRITABLE;

break;

}

case 4:

{

pageTable[i].proType = READABLE | EXECUTABLE;

break;

}

case 5:

{

pageTable[i].proType = WRITABLE | EXECUTABLE;

break;

}

case 6:

{

pageTable[i].proType = READABLE | WRITABLE | EXECUTABLE;

break;

}

default:

break;

}

/\* 设置该页对应的辅存地址 \*/

pageTable[i].auxAddr = i \* PAGE\_SIZE \* 2;

}

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

{

/\* 随机选择一些物理块进行页面装入 \*/

if (rand() % 2 == 0)

{

do\_page\_in(&pageTable[j], j);

pageTable[j].blockNum = j;

pageTable[j].filled = TRUE;

blockStatus[j] = TRUE;

}

else

blockStatus[j] = FALSE;

}

}

1. void do\_print\_info()

功能：打印页表的所有内容。

void do\_print\_info()

{

unsigned int i, j, k;

char str[4];

printf("页号\t块号\t装入\t修改\t保护\t计数\t辅存\n");

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

{

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

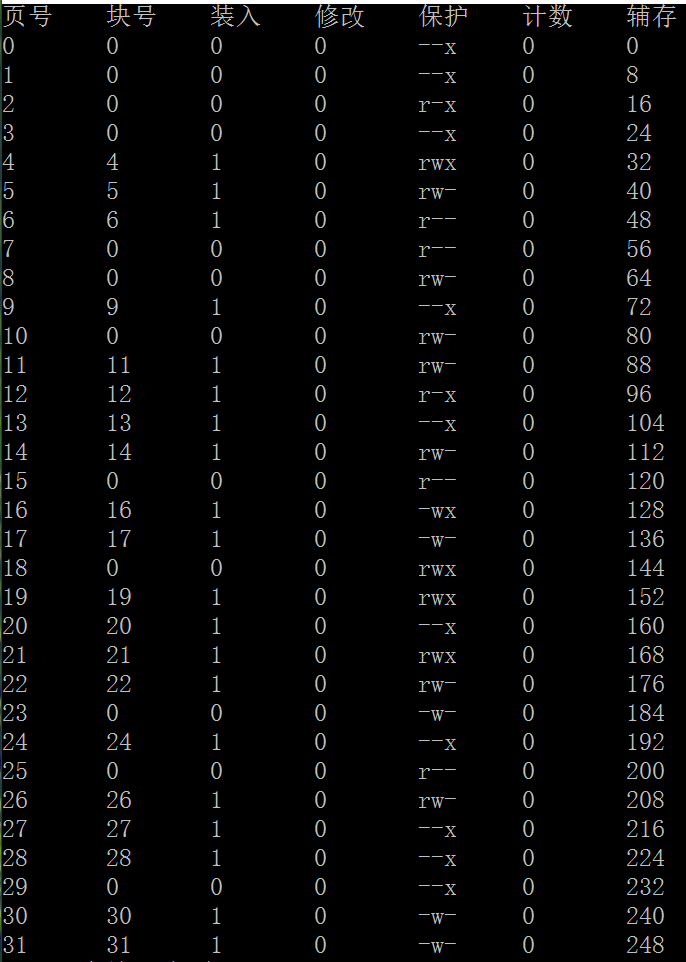
pageTable[i].edited, get\_proType\_str(str, pageTable[i].proType),

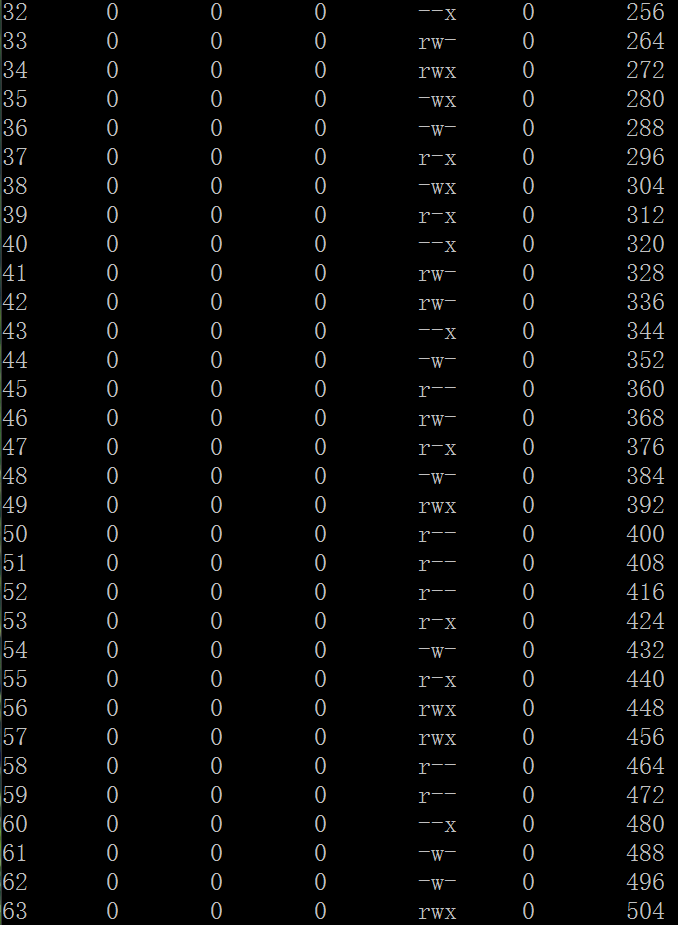
pageTable[i].count, pageTable[i].auxAddr);

}

}

初始化完成后打印页表如下：





1. do\_request()

功能：随机产生访问请求

void do\_request()

{

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

/\* 随机产生请求类型 \*/

switch (rand() % 3)

{

case 0: //读请求

{

ptr\_memAccReq->reqType = REQUEST\_READ;

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

break;

}

case 1: //写请求

{

ptr\_memAccReq->reqType = REQUEST\_WRITE;

/\* 随机产生待写入的值 \*/

ptr\_memAccReq->value = rand() % 0xFFu;

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

break;

}

case 2:

{

ptr\_memAccReq->reqType = REQUEST\_EXECUTE;

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

break;

}

default:

break;

}

}

随机产生请求如下：



1. do\_reponse()

功能：对产生的访问请求进行处理。首先获得辅存中的页号，然后在页表中相应查找。如果没有则发生缺页中断，将该页调入内存中进行读写。

void do\_response()

{

Ptr\_PageTableItem ptr\_pageTabIt;

unsigned int pageNum, offAddr;

unsigned int actAddr;

/\* 检查地址是否越界 \*/

if (ptr\_memAccReq->virAddr < 0 || ptr\_memAccReq->virAddr >= VIRTUAL\_MEMORY\_SIZE)

{

do\_error(ERROR\_OVER\_BOUNDARY);

return;

}

/\* 计算页号和页内偏移值 \*/

pageNum = ptr\_memAccReq->virAddr / PAGE\_SIZE;

offAddr = ptr\_memAccReq->virAddr % PAGE\_SIZE;

printf("页号为：%u\t页内偏移为：%u\n", pageNum, offAddr);

/\* 获取对应页表项 \*/

ptr\_pageTabIt = &pageTable[pageNum];

/\* 根据特征位决定是否产生缺页中断 \*/

if (!ptr\_pageTabIt->filled)

{

do\_page\_fault(ptr\_pageTabIt);

}

actAddr = ptr\_pageTabIt->blockNum \* PAGE\_SIZE + offAddr;

printf("实地址为：%u\n", actAddr);

/\* 检查页面访问权限并处理访存请求 \*/

switch (ptr\_memAccReq->reqType)

{

case REQUEST\_READ: //读请求

{

ptr\_pageTabIt->count++;

if (!(ptr\_pageTabIt->proType & READABLE)) //页面不可读

{

do\_error(ERROR\_READ\_DENY);

return;

}

/\* 读取实存中的内容 \*/

printf("读操作成功：值为%02X\n", actMem[actAddr]);

break;

}

case REQUEST\_WRITE: //写请求

{

ptr\_pageTabIt->count++;

if (!(ptr\_pageTabIt->proType & WRITABLE)) //页面不可写

{

do\_error(ERROR\_WRITE\_DENY);

return;

}

/\* 向实存中写入请求的内容 \*/

actMem[actAddr] = ptr\_memAccReq->value;

ptr\_pageTabIt->edited = TRUE;

printf("写操作成功\n");

break;

}

case REQUEST\_EXECUTE: //执行请求

{

ptr\_pageTabIt->count++;

if (!(ptr\_pageTabIt->proType & EXECUTABLE)) //页面不可执行

{

do\_error(ERROR\_EXECUTE\_DENY);

return;

}

printf("执行成功\n");

break;

}

default: //非法请求类型

{

do\_error(ERROR\_INVALID\_REQUEST);

return;

}

}

}

结果如下：

C:\Users\flyx\AppData\Roaming\Tencent\Users\2684128830\QQ\WinTemp\RichOle\%EX4E[6%F@JVJ$@F%HI[0]O.png

1. void do\_page\_fault()

功能：产生缺页中断时，根据内存块是否已满来决定是否要进行页面淘汰算法，否则直接顺序的找出一个空白物理块将页面装入即可。

void do\_page\_fault(Ptr\_PageTableItem ptr\_pageTabIt)

{

unsigned int i;

printf("产生缺页中断，开始进行调页...\n");

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

{

if (!blockStatus[i])

{

/\* 读辅存内容，写入到实存 \*/

do\_page\_in(ptr\_pageTabIt, i);

/\* 更新页表内容 \*/

ptr\_pageTabIt->blockNum = i;

ptr\_pageTabIt->filled = TRUE;

ptr\_pageTabIt->edited = FALSE;

ptr\_pageTabIt->count = 0;

blockStatus[i] = TRUE;

return;

}

}

/\* 没有空闲物理块，进行页面替换 \*/

do\_LFU(ptr\_pageTabIt);

}

实现结果如下：

C:\Users\flyx\AppData\Roaming\Tencent\Users\2684128830\QQ\WinTemp\RichOle\25BP$THP($AC_5QM~@9NN9A.png

1. do\_LFU()

功能：当发生缺页中断的时候，如果内存块已经满了，则根据LFU算法从内存块中淘汰一个页面。

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

{

unsigned int i, min, page;

printf("没有空闲物理块，开始进行LFU页面替换...\n");

for (i = 0, min = 0xFFFFFFFF, page = 0; i < PAGE\_SUM; i++)

{

if (pageTable[i].count < min)

{

min = pageTable[i].count;

page = i;

}

}

printf("选择第%u页进行替换\n", page);

if (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(&pageTable[page]);

}

pageTable[page].filled = FALSE;

pageTable[page].count = 0;

/\* 读辅存内容，写入到实存 \*/

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

/\* 更新页表内容 \*/

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

ptr\_pageTabIt->filled = TRUE;

ptr\_pageTabIt->edited = FALSE;

ptr\_pageTabIt->count = 0;

printf("页面替换成功\n");

}

1. void do\_page\_in()

功能：将辅存内容写入到实存

void do\_page\_in(Ptr\_PageTableItem ptr\_pageTabIt, unsigned int blockNum)

{

unsigned int readNum;

if (fseek(ptr\_auxMem, ptr\_pageTabIt->auxAddr, SEEK\_SET) < 0)

{

#ifdef DEBUG

printf("DEBUG: auxAddr=%u\tftell=%u\n", ptr\_pageTabIt->auxAddr, ftell(ptr\_auxMem));

#endif

do\_error(ERROR\_FILE\_SEEK\_FAILED);

exit(1);

}

if ((readNum = fread(actMem + blockNum \* PAGE\_SIZE,

sizeof(BYTE), PAGE\_SIZE, ptr\_auxMem)) < PAGE\_SIZE)

{

#ifdef DEBUG

printf("DEBUG: auxAddr=%u\tftell=%u\n", ptr\_pageTabIt->auxAddr, ftell(ptr\_auxMem));

printf("DEBUG: blockNum=%u\treadNum=%u\n", blockNum, readNum);

printf("DEGUB: feof=%d\tferror=%d\n", feof(ptr\_auxMem), ferror(ptr\_auxMem));

#endif

do\_error(ERROR\_FILE\_READ\_FAILED);

exit(1);

}

printf("调页成功：辅存地址%u-->>物理块%u\n", ptr\_pageTabIt->auxAddr, blockNum);

}

实现结果如下：

C:\Users\flyx\AppData\Roaming\Tencent\Users\2684128830\QQ\WinTemp\RichOle\N34N26)%AI@}I%DQFZRK4NY.png

1. do\_page\_out()

功能：如果对调入内存的页面进行了写入操作，在页面淘汰时应该将其写回辅存中。

void do\_page\_out(Ptr\_PageTableItem ptr\_pageTabIt)

{

unsigned int writeNum;

if (fseek(ptr\_auxMem, ptr\_pageTabIt->auxAddr, SEEK\_SET) < 0)

{

#ifdef DEBUG

printf("DEBUG: auxAddr=%u\tftell=%u\n", ptr\_pageTabIt, ftell(ptr\_auxMem));

#endif

do\_error(ERROR\_FILE\_SEEK\_FAILED);

exit(1);

}

if ((writeNum = fwrite(actMem + ptr\_pageTabIt->blockNum \* PAGE\_SIZE,

sizeof(BYTE), PAGE\_SIZE, ptr\_auxMem)) < PAGE\_SIZE)

{

#ifdef DEBUG

printf("DEBUG: auxAddr=%u\tftell=%u\n", ptr\_pageTabIt->auxAddr, ftell(ptr\_auxMem));

printf("DEBUG: writeNum=%u\n", writeNum);

printf("DEGUB: feof=%d\tferror=%d\n", feof(ptr\_auxMem), ferror(ptr\_auxMem));

#endif

do\_error(ERROR\_FILE\_WRITE\_FAILED);

exit(1);

}

printf("写回成功：物理块%u-->>辅存地址%03X\n", ptr\_pageTabIt->auxAddr, ptr\_pageTabIt->blockNum);

}

1. void do\_error()

功能：输出各种错误信息

void do\_error(ERROR\_CODE code)

{

switch (code)

{

case ERROR\_READ\_DENY:

{

printf("访存失败：该地址内容不可读\n");

break;

}

case ERROR\_WRITE\_DENY:

{

printf("访存失败：该地址内容不可写\n");

break;

}

case ERROR\_EXECUTE\_DENY:

{

printf("访存失败：该地址内容不可执行\n");

break;

}

case ERROR\_INVALID\_REQUEST:

{

printf("访存失败：非法访存请求\n");

break;

}

case ERROR\_OVER\_BOUNDARY:

{

printf("访存失败：地址越界\n");

break;

}

case ERROR\_FILE\_OPEN\_FAILED:

{

printf("系统错误：打开文件失败\n");

break;

}

case ERROR\_FILE\_CLOSE\_FAILED:

{

printf("系统错误：关闭文件失败\n");

break;

}

case ERROR\_FILE\_SEEK\_FAILED:

{

printf("系统错误：文件指针定位失败\n");

break;

}

case ERROR\_FILE\_READ\_FAILED:

{

printf("系统错误：读取文件失败\n");

break;

}

case ERROR\_FILE\_WRITE\_FAILED:

{

printf("系统错误：写入文件失败\n");

break;

}

default:

{

printf("未知错误：没有这个错误代码\n");

}

}

}

10.\*get\_proType\_str()

功能：获取保护类型字符串

char \*get\_proType\_str(char \*str, BYTE type)

{

if (type & READABLE)

str[0] = 'r';

else

str[0] = '-';

if (type & WRITABLE)

str[1] = 'w';

else

str[1] = '-';

if (type & EXECUTABLE)

str[2] = 'x';

else

str[2] = '-';

str[3] = '\0';

return str;

}

11．Main()

功能：通过循环来持续产生随机访问请求并及时进行响应

int main(int argc, char\* argv[])

{

char c;

int i;

if (!(ptr\_auxMem = fopen(AUXILIARY\_MEMORY, "r+")))

{

do\_error(ERROR\_FILE\_OPEN\_FAILED);

exit(1);

}

do\_init();

do\_print\_info();

ptr\_memAccReq = (Ptr\_MemoryAccessRequest) malloc(sizeof(MemoryAccessRequest));

/\* 在循环中模拟访存请求与处理过程 \*/

while (TRUE)

{

do\_request();

do\_response();

printf("按Y打印页表，按其他键不打印...hhha\n");

if ((c = getchar()) == 'y' || c == 'Y')

do\_print\_info();

while (c != '\n')

c = getchar();

printf("按X退出程序，按其他键继续...\n");

if ((c = getchar()) == 'x' || c == 'X')

break;

while (c != '\n')

c = getchar();

//sleep(5000);

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

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

}

}

if (fclose(ptr\_auxMem) == EOF)

{

do\_error(ERROR\_FILE\_CLOSE\_FAILED);

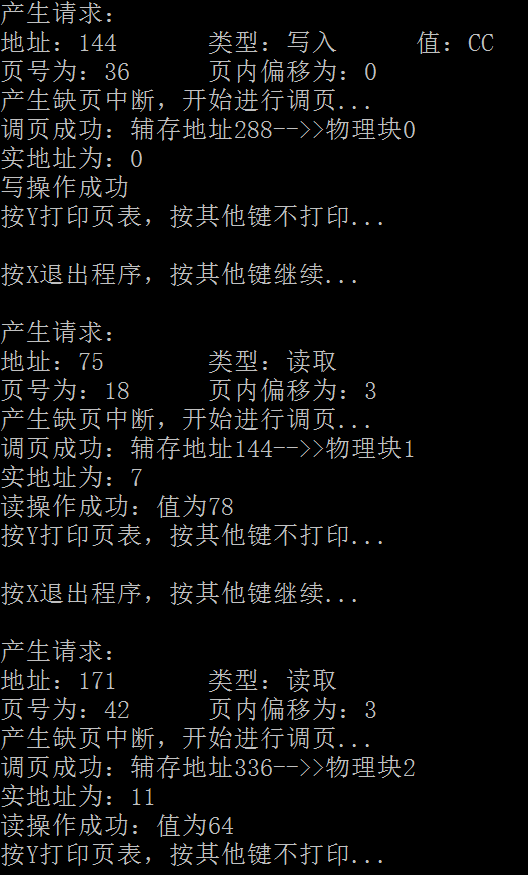
exit(1);

}

return (0);

}

结果如下：



## 3.3 提高要求实现说明

3.3.1 产生请求程序：

功能：手动产生请求，输入辅存地址，请求类型，写入数据，进程号，四个参数，并且每个参数之间用下划线分隔开。并且将每个请求写入MYFIFO文件，供响应请求程序读取。

int main(int argc, char\* argv[])

{

char c;

char Request[20]={'\0'};

char CurrentRequest[20]={'\0'};

FILE \*fp;

while (TRUE)

{

virAddr=0;

reqType=0;

value=0;

processNum=0;

if(!(fp = fopen("MYFIFO","a+"))){

do\_error(ERROR\_FILE\_OPEN\_FAILED);

exit(1);

}

scanf("%s",Request);

if(fwrite(Request,sizeof(BYTE),strlen(Request),fp)<strlen(Request)){

printf("系统错误：写入文件失败\n");

break;

}

fputc('\n',fp);

fclose(fp);

}

if (fclose(ptr\_auxMem) == EOF)

{

do\_error(ERROR\_FILE\_CLOSE\_FAILED);

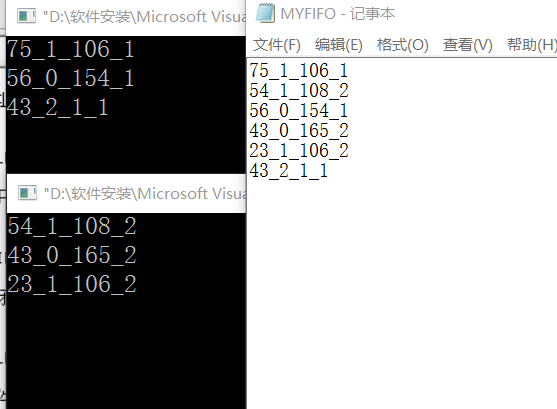
exit(1);

}

return (0);

}

结果如下：



3.3.2 响应请求程序

1. void do\_init()

功能，初始化四个进程的初始页表并随机找一些页面装入物理块

void do\_init()

{

unsigned int i, j, k,n;

srand(time(NULL));

for(n=0;n<4;n++){

for (k = 0; k<SecondPAGE\_SUM;k++){

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

{

pageTable[n][k][i].pageNum = i;

pageTable[n][k][i].filled = FALSE;

pageTable[n][k][i].edited = FALSE;

pageTable[n][k][i].count = 0;

for(j=0;j<8;j++)

{

pageTable[n][k][i].LRUcount[j]=0;

}

pageTable[n][k][i].processNum=n;

pageTable[n][k][i].visited=0;

/\* 使用随机数设置该页的保护类型 \*/

switch (rand() % 7)

{

case 0:

{

pageTable[n][k][i].proType = READABLE;

break;

}

case 1:

{

pageTable[n][k][i].proType = WRITABLE;

break;

}

case 2:

{

pageTable[n][k][i].proType = EXECUTABLE;

break;

}

case 3:

{

pageTable[n][k][i].proType = READABLE | WRITABLE;

break;

}

case 4:

{

pageTable[n][k][i].proType = READABLE | EXECUTABLE;

break;

}

case 5:

{

pageTable[n][k][i].proType = WRITABLE | EXECUTABLE;

break;

}

case 6:

{

pageTable[n][k][i].proType = READABLE | WRITABLE | EXECUTABLE;

break;

}

default:

break;

}

/\* 设置该页对应的辅存地址 \*/

pageTable[n][k][i].auxAddr =(k\*8+ i) \* PAGE\_SIZE \* 2;

}

}

}

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

{

/\* 随机选择一些物理块进行页面装入 \*/

if (rand() % 2 == 0)

{

do\_page\_in(&pageTable[j%4][j/8][j%8], j);

pageTable[j%4][j/8][j%8].blockNum = j;

pageTable[j%4][j/8][j%8].filled = TRUE;

blockStatus[j] = TRUE;

}

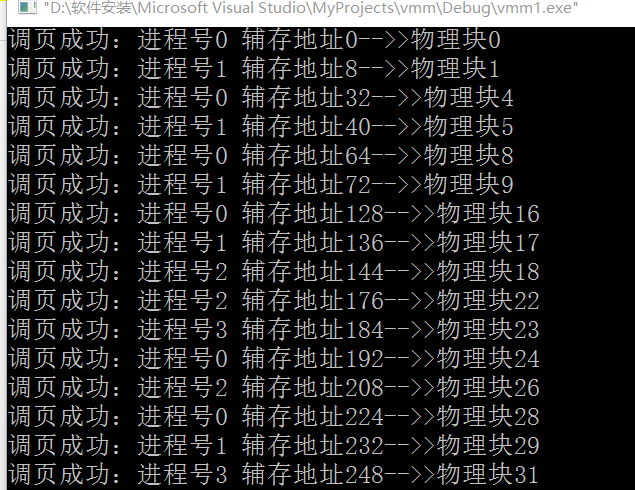
else

blockStatus[j] = FALSE;

}

}

运行结果如下：



2. void do\_print\_info(int i)

功能：打印第i号进程的页表

void do\_print\_info(int processNum)

{

unsigned int i, k;

char str[4];

printf("页号\t块号\t装入\t修改\t保护\t计数\t辅存\n");

for(k = 0; k< SecondPAGE\_SUM;k++){

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

{

printf("%u\t%u\t%u\t%u\t%s\t%u\t%u\n", k\*8+i, pageTable[processNum][k][i].blockNum, pageTable[processNum][k][i].filled,

pageTable[processNum][k][i].edited, get\_proType\_str(str, pageTable[processNum][k][i].proType),

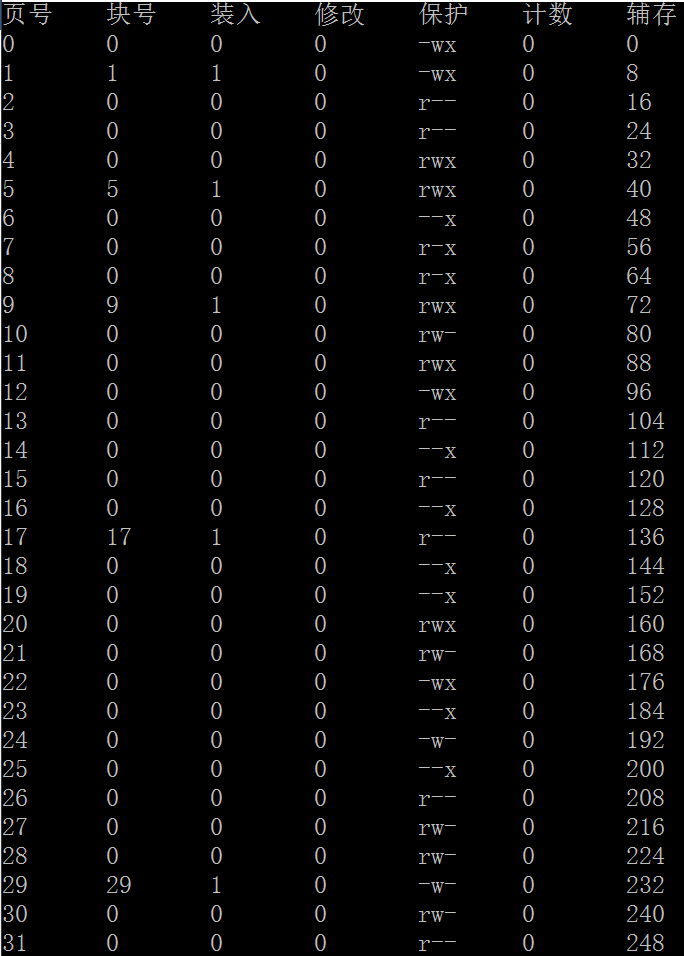
pageTable[processNum][k][i].count, pageTable[processNum][k][i].auxAddr);

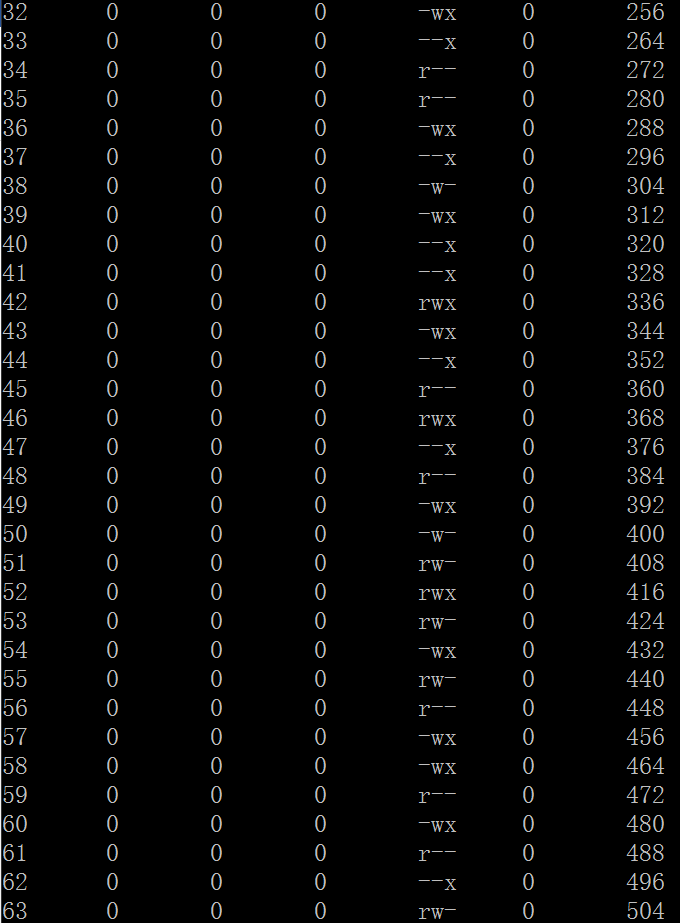
}

}

}

打印一号进程页表结果如下：





3．Void do\_response(int processNum)

功能：处理processNum号进程发出的请求。

void do\_response(int processNum)//响应请求的时候标记他最近被访问过

{

Ptr\_PageTableItem ptr\_pageTabIt;

unsigned int pageNum1, pageNum2,offAddr;

unsigned int actAddr;

/\* 检查地址是否越界 \*/

if (ptr\_memAccReq->virAddr < 0 || ptr\_memAccReq->virAddr >= VIRTUAL\_MEMORY\_SIZE)

{

do\_error(ERROR\_OVER\_BOUNDARY);

return;

}

/\* 计算页号和页内偏移值 \*/

pageNum1 = (ptr\_memAccReq->virAddr / PAGE\_SIZE)/SecondPAGE\_SUM;//页表目录号

pageNum2 = (ptr\_memAccReq->virAddr / PAGE\_SIZE)%SecondPAGE\_SUM;//页表项

offAddr = ptr\_memAccReq->virAddr % PAGE\_SIZE;

printf("进程号：%d\t页目录号为：%u\t页号为：%u\t页内偏移为：%u\n", processNum+1,pageNum1, pageNum2,offAddr);

/\* 获取对应页表项 \*/

ptr\_pageTabIt = &pageTable[processNum][pageNum1][pageNum2];

/\* 根据特征位决定是否产生缺页中断 \*/

if (!ptr\_pageTabIt->filled)

{

do\_page\_fault(ptr\_pageTabIt,processNum);

}

actAddr = ptr\_pageTabIt->blockNum \* PAGE\_SIZE + offAddr;

printf("实地址为：%u\n", actAddr);

/\* 检查页面访问权限并处理访存请求 \*/

switch (ptr\_memAccReq->reqType)

{

case REQUEST\_READ: //读请求

{

ptr\_pageTabIt->count++;

ptr\_pageTabIt->visited=1;

if (!(ptr\_pageTabIt->proType & READABLE)) //页面不可读

{

do\_error(ERROR\_READ\_DENY);

return;

}

/\* 读取实存中的内容 \*/

printf("读操作成功：值为%02x\n", actMem[actAddr]);

break;

}

case REQUEST\_WRITE: //写请求

{

ptr\_pageTabIt->count++;

ptr\_pageTabIt->visited=1;

if (!(ptr\_pageTabIt->proType & WRITABLE)) //页面不可写

{

do\_error(ERROR\_WRITE\_DENY);

return;

}

/\* 向实存中写入请求的内容 \*/

actMem[actAddr] = ptr\_memAccReq->value;

ptr\_pageTabIt->edited = TRUE;

printf("写操作成功,写入值为%02x\n",ptr\_memAccReq->value);

break;

}

case REQUEST\_EXECUTE: //执行请求

{

ptr\_pageTabIt->count++;

ptr\_pageTabIt->visited=1;

if (!(ptr\_pageTabIt->proType & EXECUTABLE)) //页面不可执行

{

do\_error(ERROR\_EXECUTE\_DENY);

return;

}

printf("执行成功\n");

break;

}

default: //非法请求类型

{

do\_error(ERROR\_INVALID\_REQUEST);

return;

}

}

}

4.void do\_LRU()

功能：根据页面老化算法进行页面淘汰调度

void do\_LRU(Ptr\_PageTableItem ptr\_pageTabIt,int processNum)

{

unsigned int i, j,k,min, page1,page2;

unsigned int Temp=0;

printf("没有空闲物理块，开始进行LFU页面替换...\n");

for (k = 0; k < SecondPAGE\_SUM; k++){

for (i = 0, min = 0xFFFFFFFF, page1 = 0,page2=0; i < PAGE\_SUM; i++)

{

for(j=0;j<8;j++){

Temp=10\*Temp+pageTable[processNum][k][i].LRUcount[j];

}

if (Temp < min)

{

min = Temp;

page1 = k;

page2 = i;

}

}

}

printf("选择第%u页进行替换\n", page1\*SecondPAGE\_SUM+i);

if (pageTable[processNum][page1][page2].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(&pageTable[processNum][page1][page2]);

}

pageTable[processNum][page1][page2].filled = FALSE;

for(j=0;j<8;j++)

{

pageTable[processNum][page1][page2].LRUcount[j]=0;

}

/\* 读辅存内容，写入到实存 \*/

do\_page\_in(ptr\_pageTabIt, pageTable[processNum][page1][page2].blockNum);

/\* 更新页表内容 \*/

ptr\_pageTabIt->blockNum = pageTable[processNum][page1][page2].blockNum;

ptr\_pageTabIt->filled = TRUE;

ptr\_pageTabIt->edited = FALSE;

for(j=0;j<8;j++)

{

ptr\_pageTabIt->LRUcount[j]=0;

}

printf("页面替换成功\n");

}

5.Main()

功能：从MIFIFO文件读取一行字符串（每次读取的字符串为上一次读取字符串的下一行）。并对读取的字符串进行分析，并分理出，请求地址，请求类型，写入数据，进程号这四个参数，再将这四个参数传给do\_response()进行处理。每当处理一百个请求的时候，修改所有页表中的老化算法八位字符标识。

int main(int argc, char\* argv[])

{

char c;

char Request[20]={'\0'};

char buffer[5]={'\0'};

char CurrentRequest[20]={'\0'};

int i,j,k,n,underlinecount;

int Requestcount=0;

int virAddr,reqType;

char value;

int processNum=1;

int RequestCount=0;//用来每处理一定的请求就根据页表的脏位来修改LRUcount;

ptr\_memAccReq = (Ptr\_MemoryAccessRequest) malloc(sizeof(MemoryAccessRequest));

if (!(ptr\_auxMem = fopen(AUXILIARY\_MEMORY1, "r+")))

{

do\_error(ERROR\_FILE\_OPEN\_FAILED);

exit(1);

}

do\_init();

do\_print\_info(1);

/\* 在循环中模拟访存请求与处理过程 \*/

while (TRUE)

{

virAddr=0;

reqType=0;

value='0';

processNum=0;

Requestcount++;

if(!(fp = fopen("MYFIFO","r"))){

do\_error(ERROR\_FILE\_OPEN\_FAILED);

exit(1);

}

for(n=1;n<=Requestcount;n++){

if(fgets(CurrentRequest,256,fp)==NULL){

printf("系统错误：读入文件失败\n");

break;

}

}

fclose(fp);

printf("请求读取完成:%s\n",CurrentRequest);

underlinecount=0;

for(j=0;j<strlen(CurrentRequest)-1;j++){

if(CurrentRequest[j]=='\_'){

underlinecount++;

}

else{

if(underlinecount==0){

virAddr=10\*virAddr+CurrentRequest[j]-'0';

}

else if(underlinecount==1){

reqType=10\*reqType+CurrentRequest[j]-'0';

}

else if(underlinecount==2){

value=10\*(value-'0')+CurrentRequest[j];

}

else{

processNum=10\*processNum+CurrentRequest[j]-'0';

}

}

}

ptr\_memAccReq->virAddr=virAddr;

ptr\_memAccReq->reqType=reqType;

ptr\_memAccReq->value=value;

ptr\_memAccReq->processNum=processNum;

if(processNum==1){

if (!(ptr\_auxMem = fopen(AUXILIARY\_MEMORY1, "r+")))

{

do\_error(ERROR\_FILE\_OPEN\_FAILED);

exit(1);

}

}

else if(processNum==2){

if (!(ptr\_auxMem = fopen(AUXILIARY\_MEMORY2, "r+")))

{

do\_error(ERROR\_FILE\_OPEN\_FAILED);

exit(1);

}

}

else if(processNum==3){

if (!(ptr\_auxMem = fopen(AUXILIARY\_MEMORY3, "r+")))

{

do\_error(ERROR\_FILE\_OPEN\_FAILED);

exit(1);

}

}

else if(processNum=4){

if (!(ptr\_auxMem = fopen(AUXILIARY\_MEMORY4, "r+")))

{

do\_error(ERROR\_FILE\_OPEN\_FAILED);

exit(1);

}

}

do\_response(processNum);

RequestCount++;

printf("按Y打印页表，按其他键不打印...\n");

if ((c = getchar()) == 'y' || c == 'Y')

do\_print\_info(processNum);

while (c != '\n')

c = getchar();

printf("按E打印实存，按其他键不打印...\n");

if ((c = getchar()) == 'E' || c == 'e')

{

for(i=0;i<ACTUAL\_MEMORY\_SIZE;i=i+4){

printf("物理块号：%d:%c%c%c%c\n",i/4,actMem[i],actMem[i+1],actMem[i+2],actMem[i+3]);

}

}

while (c != '\n')

c = getchar();

printf("按R打印辅存，按其他键不打印...\n");

if ((c = getchar()) == 'R' || c == 'r')

{

for(i=0;i<VIRTUAL\_MEMORY\_SIZE;i=i+4){

fread(buffer,sizeof(BYTE),4,ptr\_auxMem);

buffer[4]='\0';

printf("辅存块号：%d:%s\n",i/4,buffer);

}

}

while (c != '\n')

c = getchar();

printf("按X退出程序，按其他键继续...\n");

if ((c = getchar()) == 'x' || c == 'X')

break;

while (c != '\n')

c = getchar();

if(RequestCount==100){

RequestCount=0;

for(n=0;n<4;n++){

for(k=0;k<SecondPAGE\_SUM;k++){

for(i=0;i<PAGE\_SUM;i++){

for(j=7;j>0;j++){

pageTable[n][k][i].LRUcount[j]=pageTable[n][k][i].LRUcount[j-1];

}

pageTable[n][k][i].LRUcount[0]=pageTable[n][k][i].visited;

}

}

}

}

}

if (fclose(ptr\_auxMem) == EOF)

{

do\_error(ERROR\_FILE\_CLOSE\_FAILED);

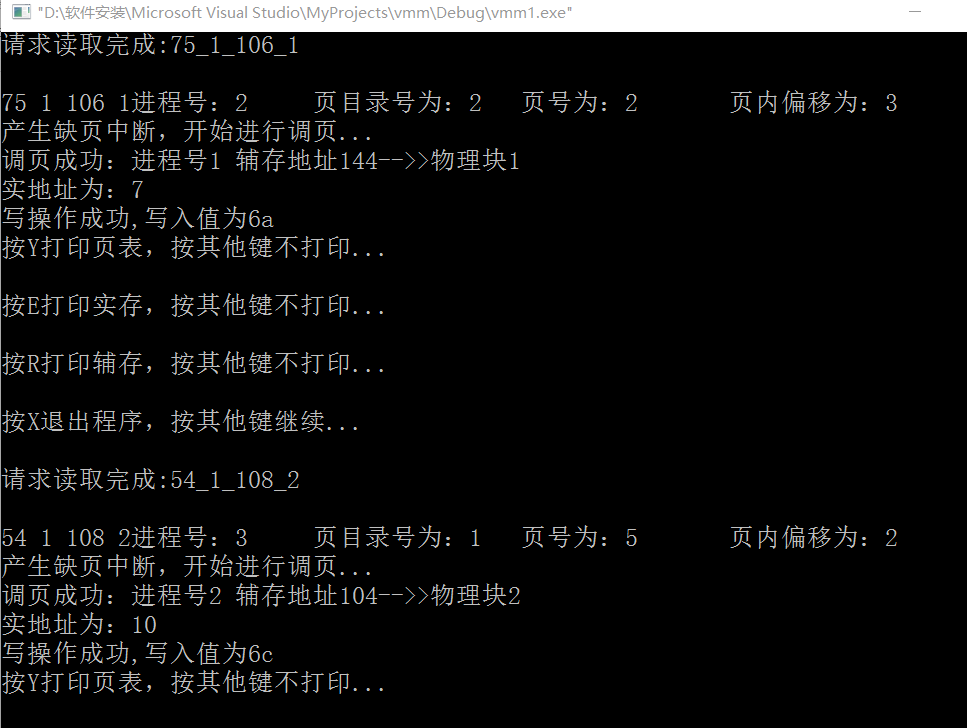
exit(1);

}

return (0);

}

结果示意图如下：



# 4.收获和感想

经过这次实验，收获的确是很多。首先第一次独立完成这样一次的OS实验，对自己的独立思考和开发都有了极大的帮助。虽然提高要求的代码量并不是很高，但是在一个四五百行代码量的修改时往往会牵一发而动全身，所以往往会出现这个地方改了那个小地方忘记了没改。这让我认识到不管是修改代码还是书写代码都要从整个代码的设计和逻辑层次去考虑，这样才不容易会出错。其实我们在计算机组成原理这么课上也有学过虚拟存储和页面调度，但是由于那时候没有动手做实验，所以对这方面理解的还不够深刻，经过这次实验，我对页面调度算法，以及程序是如何从虚实存之间进行调度的都有了很深的理解，不仅如此，在页表内部结构和多级页表也理解的更加深刻了。真的是自己做过实验才会理解的更加深刻，纸上得来终觉浅，绝知此事要躬行。由于我这次是将请求写入文件的再从文件中读出的，在多次的练习和摸索中我也对C语言的文件操作运用的更加熟练了。通过仔细阅读基本要求的源代码，我感觉书上的代码风格非常好，多出利用宏定义变量，这样在修改某个变量大小时就不用到处修改，代码也变得更容易阅读。模块化做得很好，main函数的代码并不多，简洁。我也会学习这种代码风格并且运用到自己日后的代码书写中。