**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：胡瀚涛

学号：14061178

日期：2016.5.7

**小组成员**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 姓名 | 学号 | 实验分工 |
| 1 | 张绍钧、杨健 |  | 实验一 |
| 2 | 程富瑞 |  | 实验二 |
| 3 | 胡瀚涛 | 14061178 | 实验三 |
| 4 | 张绍钧、杨健 |  | 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空间，用一个文本文件模拟辅存空间；
2. 建立一张一级页表；
3. 程序中使用函数do\_request()随机产生访存请求，访存操作包括读取、写入、执行三种类型；
4. 实现函数do\_response()响应访存请求，完成虚地址到实地址的定位及读/写执行操作，同时判断并处理缺页中断；
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快表；
3. 将do\_request()和do\_response()函数实现在不同进程中，通过进程间通信(如FIFO)完成访存控制的模拟；
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.基本要求全部实现(其实源代码就都实现了)；

2.支持实存辅存页表的打印；

3.模拟了多道程序的存储控制；

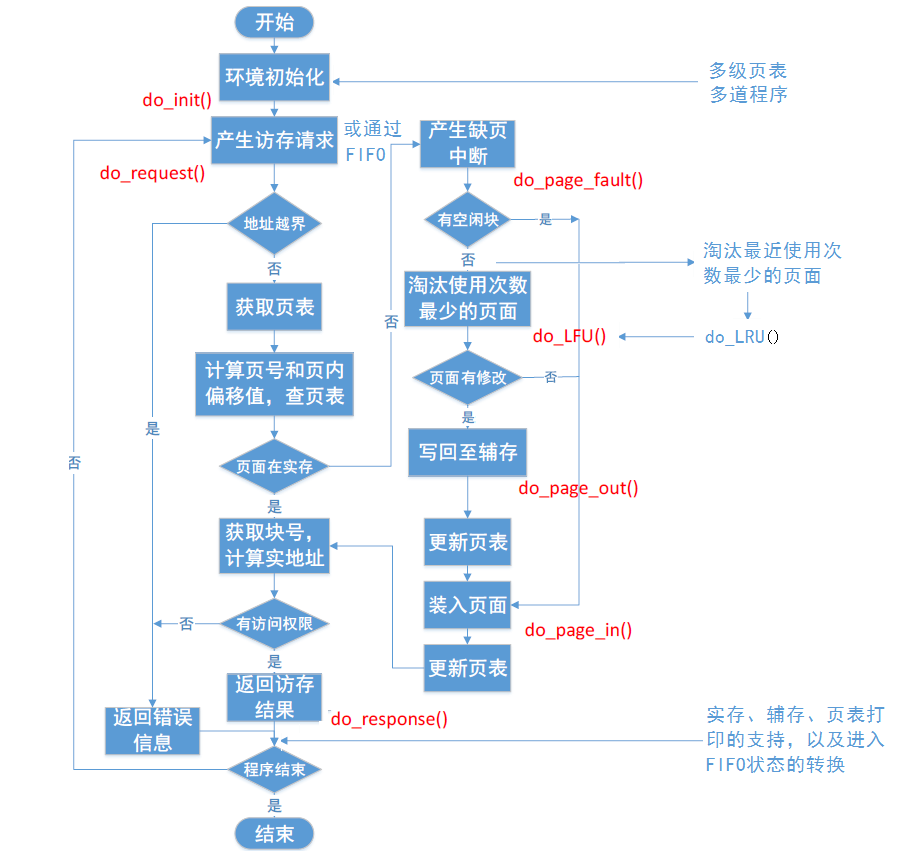
4.在实现多道程序存储控制的基础上，实现了多级页表；

5.基于3、4点实现了对应的LRU页面淘汰算法；

6.模拟了do\_request()和do\_response()函数在不同进程中时的进程间通信，通过FIFO来实现。

# 3.设计说明

## 3.1 程序流程图



## 3.2基本要求实现说明

1. 由于多道程序的覆盖，已无法查看单道程序的模拟，但是一维数组模拟实存以及文本文件模拟辅存空间依然保留了原先设置：

#define AUXILIARY\_MEMORY "vmm\_auxMem"

BYTE actMem[ACTUAL\_MEMORY\_SIZE];

1. 同样由于多级页表的实现，无法再重现一级页表；
2. do\_request函数(由于没有解决转码问题，我将代码中所有出现中文的地方都删除了，下同)：

void do\_request()

{

int i,j,k;

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

ptr\_memAccReq->FromProgress = rand() % VIRTUAL\_PROGRESSES;

switch (rand() % 3)

{

case 0:

{

ptr\_memAccReq->reqType = REQUEST\_READ;

break;

}

case 1:

{

ptr\_memAccReq->reqType = REQUEST\_WRITE;

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

break;

}

case 2:

{

ptr\_memAccReq->reqType = REQUEST\_EXECUTE;

break;

}

default:

break;

}

随机生成请求包括三个部分：

(1).虚地址，范围为0-255；

(2).所属进程，这是模拟多道进程的一部分，在本程序中范围为0-1；

(3).访问类型，分为读、写、执行，在生成写指令时还要随机生成写入的数据；

4.do\_response()函数：

void do\_response()

{

Ptr\_PageTableItem ptr\_pageTabIt;

unsigned int pageNum1, pageNum2, offAddr;

unsigned int actAddr,i;

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

{

do\_error(ERROR\_OVER\_BOUNDARY);

return;

}

pageNum1 = ptr\_memAccReq->virAddr / LV1\_PAGE\_SIZE;

offAddr = ptr\_memAccReq->virAddr % LV1\_PAGE\_SIZE;

pageNum2 = offAddr / LV2\_PAGE\_SIZE;

offAddr = offAddr % LV2\_PAGE\_SIZE;

ptr\_pageTabIt = &pageTable[ptr\_memAccReq->FromProgress][pageNum1][pageNum2];

if (!ptr\_pageTabIt->filled)

{

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

{

actmemcount[ptr\_memAccReq->FromProgress][pageNum1 \* LV1\_PAGE\_SUM + pageNum2][i]=0;

}

do\_page\_fault(ptr\_pageTabIt);

}

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

switch (ptr\_memAccReq->reqType)

{

case REQUEST\_READ:

{

ptr\_pageTabIt->count++;

actmemcount[ptr\_memAccReq->FromProgress][pageNum1\* LV1\_PAGE\_SUM + pageNum2][0]=1;

if (!(ptr\_pageTabIt->proType & READABLE))

{

do\_error(ERROR\_READ\_DENY);

return;

}

break;

}

case REQUEST\_WRITE:

{

ptr\_pageTabIt->count++;

actmemcount[ptr\_memAccReq->FromProgress][pageNum1 \* LV1\_PAGE\_SUM + pageNum2][0]=1;

if (!(ptr\_pageTabIt->proType & WRITABLE))

{

do\_error(ERROR\_WRITE\_DENY);

return;

}

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

ptr\_pageTabIt->edited = TRUE;

break;

}

case REQUEST\_EXECUTE:

{

ptr\_pageTabIt->count++;

actmemcount[ptr\_memAccReq->FromProgress][pageNum1 \* LV1\_PAGE\_SUM + pageNum2][0]=1;

if (!(ptr\_pageTabIt->proType & EXECUTABLE)) 

{

do\_error(ERROR\_EXECUTE\_DENY);

return;

}

break;

}

default:

{

do\_error(ERROR\_INVALID\_REQUEST);

return;

}

}

}

do\_response()函数包括几个部分：第一部分，地址越界判断；第二部分，寻找页表；第三部分，进行缺页中断判断；第四部分，调页；第五部分，处理请求，可以说，do\_response()函数是整个作业最主要的函数之一。

1. LFU算法我会和下面的LRU算法一起介绍。

## 3.3 提高要求实现说明

1. 二级页表的实现&&多道程序的实现

新的结构：

PageTableItem pageTable[VIRTUAL\_PROGRESSES][LV1\_PAGE\_SUM][LV2\_PAGE\_SUM];

三维数组的第一维是页表所属进程；第二维是第一级页表；第三维是第二级页表，在这里三个宏的具体值为2，8，8，也就是说，我们模拟了两个进程，每一个进程都有一张8\*8的虚拟页表。这样做的好处在于我们无需对请求进行权限判断（比如该请求访问的虚地址位于进程1，那么我们直接对进程1的虚拟页表进行查询即可）且这样做更符合真实情况。

多级页表的涉及的改动太多，这里只以寻址方式改变为例：

pageNum1 = ptr\_memAccReq->virAddr / LV1\_PAGE\_SIZE;

offAddr = ptr\_memAccReq->virAddr % LV1\_PAGE\_SIZE;

pageNum2 = offAddr / LV2\_PAGE\_SIZE;

offAddr = offAddr % LV2\_PAGE\_SIZE;

由于页表多了一级，寻址因此也需逐级进行。

1. FIFO的实现：

我们模拟FIFO的方法是建立另一个.c文件不停生成请求模拟另一个线程的do\_request()，在由主程序负责接收，由于设计上的限定，运行人需要按照提示按特定键进入接受请求状态，在这个状态下请求不会自动生成而是会等待另一个程序生成，以完成这部分的实验要求。

giveRequest程序的源代码（负责生成请求管道的一端）：

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <unistd.h>

#include "vmm.h"

#define FIFO "myFIFO"

int main()

{

unsigned long addr;

unsigned char value;

char s[100];

srandom(time(NULL));

FILE \*fp;

int number;

int pid;

while(1){

pid=random()%2;

if((fp=fopen(FIFO,"w"))==NULL){

printf("write FIFO wrong");

return 0;

}

/\* 随机产生请求地址 \*/

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

scanf("%s",s);

addr=random() % VIRTUAL\_MEMORY\_SIZE;

/\* 随机产生请求类型 \*/

switch (random() % 3)

{

case 0: //读请求

{

//ptr\_memAccReq->reqType = REQUEST\_READ;

number=0;

printf("产生请求：\n地址：%lu\t类型：读取 pid:%d\n", addr,pid);

fprintf(fp,"%d %lu %d",number,addr,pid);

break;

}

case 1: //写请求

{

number=1;

//ptr\_memAccReq->reqType = REQUEST\_WRITE;

/\* 随机产生待写入的值 \*/

value = random() % 0xFFu;

fprintf(fp,"%d %lu %02X %d",number,addr,value,pid);

printf("产生请求：\n地址：%lu\t类型：写入\t值：%02X pid:%d\n",addr, value,pid);

break;

}

case 2:

{

number=2;

//ptr\_memAccReq->reqType = REQUEST\_EXECUTE;

fprintf(fp,"%d %lu %d",number,addr,pid);

printf("产生请求：\n地址：%lu\t类型：执行 pid:%d\n", addr,pid);

break;

}

default:

break;

}

fclose(fp);

sleep(5);

}

return 0;

}

主函数中用于接收管道请求的函数deal\_request();

void deal\_request(char buf[])

{

unsigned long addr;

unsigned char value;

int number;

int pid;

printf("%s\n",buf);

switch (buf[0]-'0')

{

case 0:

{

ptr\_memAccReq->reqType = REQUEST\_READ;

sscanf(buf,"0 %lu %d",&addr,&pid);

printf("%lu\n",addr);

break;

}

case 1:

{

ptr\_memAccReq->reqType = REQUEST\_WRITE;

sscanf(buf,"1 %lu %02X %d",&addr,&value,&pid);

printf("%lu\n",addr);

ptr\_memAccReq->value = value;

break;

}

case 2:

{

ptr\_memAccReq->reqType = REQUEST\_EXECUTE;

sscanf(buf,"2 %lu %d",&addr,&pid);

printf("%lu\n",addr);

break;

}

default:

break;

}

ptr\_memAccReq->virAddr = addr;

ptr\_memAccReq->FromProgress=pid;

printf("ptr\_memAccReq->FromProgress = %u\n", ptr\_memAccReq->FromProgress);

}

main函数部分：

loop3: while (TRUE)

{

umask(0);

FILE \*fp;

char \*buf[100];

mkfifo(FIFO,S\_IFIFO|0666);

fp=fopen(FIFO,"r");

fgets(buf,sizeof(buf),fp);

deal\_request(buf);

do\_response();

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

do\_print\_info();

while (c != '\n')

c = getchar();

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

break;

if(c == 't' || c == 'T'){

flag = 1;

goto loop4;

}

while (c != '\n')

c = getchar();

//sleep(5000);

}

1. LRU算法

LFU算法的思想在于，对该页面是否利用的Count次数相加是线性的，意思是，不管调用者是100s用还是1s前用，count自增的值都为1，而LRU的算法则是，利用后过去的时间越长，它在做页面淘汰判断时占的权重就越低——

int actmemcount[VIRTUAL\_PROGRESSES][PAGE\_SUM][8];

在这里我们用的时钟长度为8，这个时间片里如果该页面被利用，则它最左边的值加1，随后每一个时间片移动一格。

LFU算法：

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

{

unsigned int i, j, k,min, firstpage, page;

int whichprogress=0;

for(k=0;k<VIRTUAL\_PROGRESSES;k++)

for (i = 0, min = 0xFFFFFFFF, firstpage = 0, page = 0; i < LV1\_PAGE\_SUM; i++)

{

for (j = 0; j < LV2\_PAGE\_SUM; j++)

{

if (pageTable[k][i][j].count < min)

{

whichprogress=k;

min = pageTable[k][i][j].count;

firstpage = i;

page = j;

}

}

}

if (pageTable[whichprogress][firstpage][page].edited)

{

do\_page\_out(&pageTable[whichprogress][firstpage][page]);

}

pageTable[whichprogress][firstpage][page].filled = FALSE;

pageTable[whichprogress][firstpage][page].count = 0;

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

{

actmemcount[whichprogress][firstpage \* LV1\_PAGE\_SUM + page][i]=0;

}

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

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

ptr\_pageTabIt->filled = TRUE;

ptr\_pageTabIt->edited = FALSE;

ptr\_pageTabIt->count = 0;

}

LRU算法:

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

{

unsigned int i,j,k,min,page,firstpage;

int flag=0;

int count[VIRTUAL\_PROGRESSES][PAGE\_SUM]={0};

int Whichprogress=0;

for(k=0;k<VIRTUAL\_PROGRESSES;k++)

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

{

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

count[k][i]=count[k][i]\*2+actmemcount[k][i][j];

//printf("%d\n",count[i]);

}

for(k=0;k<VIRTUAL\_PROGRESSES;k++)

for (i = 0, firstpage = 0, page = 0; i < LV1\_PAGE\_SUM; i++)

{

for (j = 0; j < LV2\_PAGE\_SUM; j++)

{

if (pageTable[k][i][j].filled)

{

Whichprogress=k;

firstpage = i;

page = j;

flag=1;

}

break;

}

if(flag==1)

{

break;

}

}

for(k=0;k<VIRTUAL\_PROGRESSES;k++)

for (i = 0, min = 0xFFFFFFFF; i < LV1\_PAGE\_SUM; i++)

{

for (j = 0; j < LV2\_PAGE\_SUM; j++)

{

if (count[k][i\*LV1\_PAGE\_SUM+j] < min

&&pageTable[k][i][j].filled)

{

Whichprogress=k;

min = count[k][i\*LV1\_PAGE\_SUM+j];

firstpage = i;

page = j;

}

}

}

if (pageTable[Whichprogress][firstpage][page].edited)

{

do\_page\_out(&pageTable[Whichprogress][firstpage][page]);

}

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

{

actmemcount[Whichprogress][firstpage \* LV1\_PAGE\_SUM + page][i]=0;

}

whichprogress[pageTable[Whichprogress][firstpage][page].blockNum]=ptr\_memAccReq->FromProgress;

pageTable[Whichprogress][firstpage][page].filled = FALSE;

pageTable[Whichprogress][firstpage][page].count = 0;

pageTable[Whichprogress][firstpage][page].blockNum = 0;

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

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

ptr\_pageTabIt->filled = TRUE;

ptr\_pageTabIt->edited = FALSE;

ptr\_pageTabIt->count = 0;

//ptr\_pageTabIt->progressNum=ptr\_memAccReq->FromProgress;

}

# 收获和感想

这是我负责的实验，事实上，之前的实验我都基本没怎么参与过.....所以刚开始看代码、构思的时候我都遇到了很大的困难，我要非常感谢张绍钧同学给予我的帮助——首先，是他告诉我，这个实验的大半部分都是可以在Window下进行编写和测试的，能够暂时使用较为熟悉的Windows系统对我这种初学者而言无疑是一个福音(当然，使用Windows系统进行程序编写也带来了其他困难，比如.....转码问题，我的.h文件在linux上就是一堆乱码，根本无法添加或删减或修改，不过我们最后都找到了解决办法)。

这次实验的难度我认为是比较低的，抛开所有基础要求不谈，将一级页表转化为二级页表就是一个将一维数组转化为二维数组的过程；将多道程序这个元素添加进去就是一个将二维数组转化为三维数组的过程；LRU(由张绍钧同学完成)和FIFO(由杨健同学完成)不是我自己做的所以不好说感受，但依然非常感谢他们两人在实验过程中给予我的支持，没有他们的帮助，尽管这次实验难度不高，我依然是无法完成本次实验的。