**操作系统课程设计实验报告**

——实验三：虚存管理

负责人姓名：兰帅

学号：14061203

日期：2016.5.4

**小组成员**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 姓名 | 学号 | 实验分工 |
| 1 | 黄智勇 | 14061190 | 实验一 |
| 2 | 王贺 | 14061184 | 实验二 |
| 3 | 兰帅 | 14061203 | 实验三 |

目录

[1.实验目的 4](#_Toc446001831)

[2.需求说明 6](#_Toc446001832)

[2.1基本要求 6](#_Toc446001833)

[2.2 提高要求 6](#_Toc446001834)

[2.3 完成情况 6](#_Toc446001835)

[3.设计说明 7](#_Toc446001836)

[3.1 程序流程图 8](#_Toc446001837)

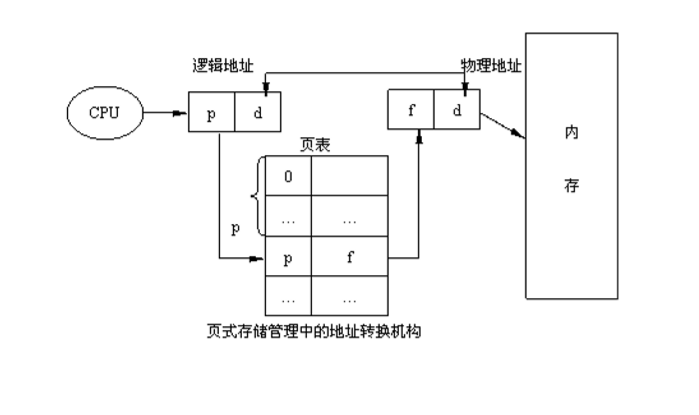
[3.2基本要求实现说明 9](#_Toc446001838)

[3.3 提高要求实现说明 1](#_Toc446001839)0

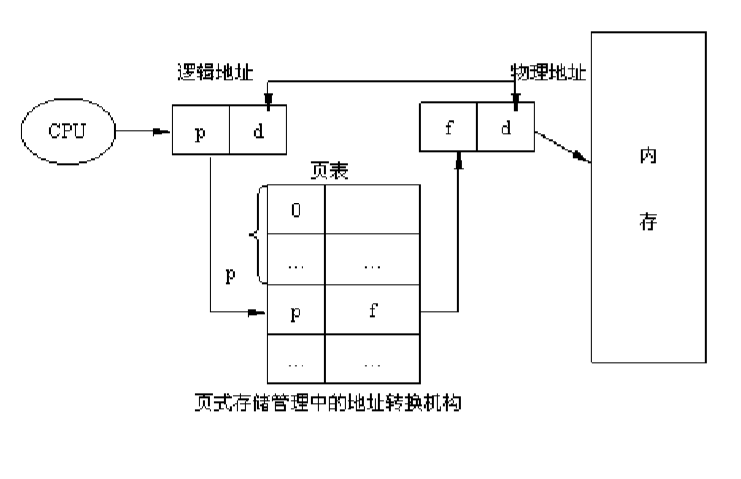
[4.收获和感想 1](#_Toc446001840)8

# 1.实验目的

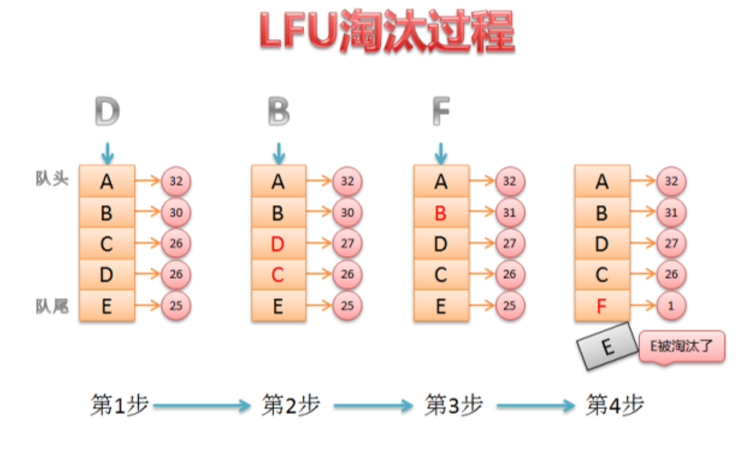
1. 了解Linux的内存管理机制，主要是了解页式存储管理机制，包括：地址转换、缺页中断、页面交换和存取控制。



1. 理解虚地址到实地址的定位过程。



1. 掌握LFU页面淘汰算法：
2. 掌握缺页中断和页面交换的原理
3. Linux系统使用老化算法
4. 实现最不频繁使用淘汰算法，因为经常使用的页应该有一个较大的引用次数，所以要求在页置 换时置换引用计数最小的页



# 2.需求说明

## 2.1基本要求

总体要求：

了解Linux系统下页式存储管理机制，并实现一个简单的虚存管理模拟程序。

具体要求：

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. 建立一个一级页表。
3. 程序中使用一个函数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. 实现一个函数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，同时判断并处理缺页中断
5. 实现LFU页面淘汰算法。

## 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. 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。
4. 实现其他页面淘汰算法：如页面老化算法、最久未使用淘汰算法(LRU)、最有算法(OPT)等。

## 2.3 完成情况

基本要求与提高要求全部完成，其中提高要求中实现其他页面淘汰算法实现的是FIFO算法。后来增加了页面老化算法。

# 3.设计说明

## 3.1 程序流程图

开始

产生缺页中断

do\_page\_fault()

有空闲块

环境初始化

是

do\_init() 否

产生访存请求

淘汰使用次数最少的页面

do\_request()

获取页表

地址越界

do\_LFU()

页面有修改

否 否

否 是

页面在实存

计算页号和页内偏移量，查页表

写回辅存

do\_page\_out

更新页表

装入页面

do\_page\_in

更新页表

是

有权访问

获取块号，计算实地址

是

程序结束

返回访存结果

返回错误信息

do\_response()

结束

## 3.2基本要求实现说明

源代码已经实现了所有的基本要求，修改了do\_request使其成为手动输入代码如下：

//输入请求所属进程

printf("请输入请求所属的进程(1~3)：\n");

scanf("%u", &num);

ptr\_memAccReq->proccessNum = num;

//输入请求地址

printf("请输入请求地址：\n");

scanf("%lu", &addr);

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

//输入请求类型

printf("请输入请求类型(r/w/e)：\n");

scanf(" %c", &cmdType);

增加了打印辅存内容和打印实存内容：

/\* 打印辅存内容 \*/

void do\_print\_vir()

{

char c;

FILE \*fp = NULL;

fp=fopen(AUXILIARY\_MEMORY, "r");

if(fp == NULL)

{

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

exit(1);

}

while(fscanf(fp,"%c",&c) != EOF)

{

printf("%c",c);

}

fclose(fp);

fp=NULL;

printf("\n");

}

/\* 打印实存内容 \*/

void do\_print\_act()

{

unsigned int i;

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

{

printf("%02X\t", actMem[i]);

}

printf("\n");

}

## 3.3 提高要求实现说明

实现二级页表时，将pageTable改为二维，将原来的pageTable进行修改。多道程序的控制在初始化中修改，在输入时输入是属于哪个程序控制的。大致代码如下：

void do\_init()

{

int i, j, k;

int time\_n = 0;

srand(time(NULL));

for (i = 0; i < PAGE\_SUM1; i++)

{

for (j = 0; j < PAGE\_SUM2; j++)

{

pageTable[i][j].pageNum = j + i\*PAGE\_SUM1;

pageTable[i][j].filled = FALSE;

pageTable[i][j].edited = FALSE;

pageTable[i][j].count = 0;

/\* 使用随机数设置该页的保护类型 \*/

switch (rand() % 7)

{

case 0:

{

pageTable[i][j].proType = READABLE;

pageTable[i][j].proccessNum = 1;

break;

}

case 1:

{

pageTable[i][j].proType = WRITABLE;

pageTable[i][j].proccessNum = 1;

break;

}

case 2:

{

pageTable[i][j].proType = EXECUTABLE;

pageTable[i][j].proccessNum = 1;

break;

}

case 3:

{

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

pageTable[i][j].proccessNum = 2;

break;

}

case 4:

{

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

pageTable[i][j].proccessNum = 2;

break;

}

case 5:

{

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

pageTable[i][j].proccessNum = 2;

break;

}

case 6:

{

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

pageTable[i][j].proccessNum = 3;

break;

}

default:

break;

}

/\* 设置该页对应的辅存地址 \*/

//pageTable[i][j].auxAddr = j \* PAGE\_SIZE \* 2 + i \* PAGE\_SUM1 \* PAGE\_SUM2;

pageTable[i][j].auxAddr = j \* PAGE\_SIZE + i \* 32;

}

}

int rand1, rand2;

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

{

/\* 随机选择一些物理块进行页面装入 \*/

rand1 = rand() % 4;

rand2 = rand() % 8;

if (rand() % 2 == 0)

{

do\_page\_in(&pageTable[rand1][rand2], k);

pageTable[rand1][rand2].blockNum = k;

pageTable[rand1][rand2].filled = TRUE;

blockStatus[k] = TRUE;

Time[time\_n++]=pageTable[rand1][rand2].pageNum;

}

else

blockStatus[k] = FALSE;

}

}

实现do\_request和do\_response的异终端是通过管道通信实现的，在这两个函数中加入管道，加入写入管道和读取管道的操作，通过fifo实现通信，大致代码如下：

do\_request在另一个终端，所以重新写了一个c文件：

int main()

{

unsigned long addr;

unsigned int num;

char c;

char cmdType;

char writeValue;

FILE \*fifo;

Ptr\_MemoryAccessRequest ptr\_memAccReq;

while (TRUE)

{

//输入请求所属进程

printf("请输入请求所属的进程(1~3)：\n");

scanf("%u", &num);

ptr\_memAccReq->proccessNum = num;

//输入请求地址

printf("请输入请求地址：\n");

scanf("%lu", &addr);

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

//输入请求类型

printf("请输入请求类型(r/w/e)：\n");

scanf(" %c", &cmdType);

getchar();

switch (cmdType)

{

case 'r': //读请求

{

ptr\_memAccReq->reqType = REQUEST\_READ;

printf("请求内容：\t地址：%lu\t类型：读取\n", ptr\_memAccReq->virAddr);

break;

}

case 'w': //写请求

{

ptr\_memAccReq->reqType = REQUEST\_WRITE;

//手动输入待写入的值

printf("请输入要写入的内容：\n");

scanf(" %c", &writeValue); //只能取到第一个字符

getchar();

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

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

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

break;

}

case 'e':

{

ptr\_memAccReq->reqType = REQUEST\_EXECUTE;

printf("请求内容：\t地址：%lu\t类型：执行\n", ptr\_memAccReq->virAddr);

break;

}

default:

break;

}

fifo = fopen(FIFO,"r+");

if(fifo < 0)

{

printf("open fifo failed");

}

if(fwrite(ptr\_memAccReq,sizeof(MemoryAccessRequest),1,fifo) < 0)

{

printf("write failed");

}

fclose(fifo);

printf("按X退出程序，按其他键继续...\n");

c = getchar();

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

break;

while (c != '\n')

{

c = getchar();

}

}

return 0;

}

void do\_response()

{

Ptr\_PageTableItem ptr\_pageTabIt;

unsigned int pageNum, offAddr;

unsigned int pageNum1, pageNum2;

unsigned int actAddr;

/\* 检查地址是否越界 \*/

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

if (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;

pageNum1 = ptr\_memAccReq->virAddr / PAGE\_SIZE / 8;

pageNum2 = ptr\_memAccReq->virAddr / PAGE\_SIZE % 8;

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

printf("一级索引：%u\t二级索引：%u\t页号为：%u\t页内偏移为：%u\n", pageNum/8, pageNum%8, pageNum, offAddr);

/\* 获取对应页表项 \*/

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

/\*书上的R位与右移

ptr\_pageTabIt->R = 1;//此页被访问，R位值置1

ptr\_pageTabIt->clock\_time = ptr\_pageTabIt->clock\_time >> 1;//计数器右移1位

ptr\_pageTabIt->clock\_time = ptr\_pageTabIt->clock\_time | 0x80000000;//“插入”R值

ptr\_pageTabIt->R = 0;//计时器已增加，R位置0

\*/

/\* 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 \*/

if (ptr\_pageTabIt->proccessNum != ptr\_memAccReq->proccessNum)

{

printf("所属进程不同，无法进行访存操作\n");

return;

}

else

{

switch (ptr\_memAccReq->reqType)

{

case REQUEST\_READ: //读请求

{

//clock():函数返回从“开启这个程序进程”到“程序中调用clock()函数”时之间的CPU时钟计时单元数

ptr\_pageTabIt->clock\_time = clock();//每过1毫秒，调用clock()函数返回的值就加1

ptr\_pageTabIt->count++;

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

{

do\_error(ERROR\_READ\_DENY);

return;

}

/\*if (ptr\_pageTabIt->proccessNum != ptr\_memAccReq->proccessNum)

{

printf("所属进程不同，无法读取\n");

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->clock\_time = clock();

ptr\_pageTabIt->count++;

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

{

do\_error(ERROR\_WRITE\_DENY);

return;

}

/\*if (ptr\_pageTabIt->proccessNum != ptr\_memAccReq->proccessNum)

{

printf("所属进程不同，无法写入\n");

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->clock\_time = clock();

ptr\_pageTabIt->count++;

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

{

do\_error(ERROR\_EXECUTE\_DENY);

return;

}

/\*if (ptr\_pageTabIt->proccessNum != ptr\_memAccReq->proccessNum)

{

printf("所属进程不同，无法执行\n");

return;

}\*/

printf("执行成功\n");

break;

}

default: //非法请求类型

{

do\_error(ERROR\_INVALID\_REQUEST);

return;

}

}

}

}

# 4.收获和感想

不知不觉已经进行到了第三次实验了，前两次实验时的不知所措茫茫然毫无头绪好像还只是一两周之前的事情。这次实验的难度相对第一次要简单一些，而且在给出的源码中，基本要求已经完全实现了，有部分需要修改的地方也在PPT中给出了说明。我们便也不用为基本要求困扰了。提高要求的实现在有了前两次实验的基础后也有了一些想法，写起来也得心应手了一些。此外，我们三个人的协作能力也有了很大的提高，分工更加明确，各自任务完成度都很高。当然，还有许多不足之处需要我们不断改进提高

这次实验是虚存管理，虚存管理可以分为页式管理、段式管理和段页式管理三类。通过实验，我们了解到Linux的内存管理采取的是分页存取机制，为保证物理内存能得到充分利用，内核会在适当的时候将物理内存中不经常使用的数据块自动交换到虚拟内存中，而将经常使用的信息保留到物理内存中。还了解到linux中虚地址是如何定位到实地址的。通过对源代码的解读，大致了解了LFU页面淘汰算法的原理和过程。自己也写了一个FIFO页面淘汰算法，随着对分页机制理解的加深，随后我们又在代码中添加了页面老化算法。对各种页面淘汰算法也算是有了一定的了解。同时通过对do\_request和do\_response的异终端的修改，使得我们对进程间的通信方法使用得更加熟练。

通过这次实验，我们对Linux系统的了解更加深了一些，对Linux系统的使用也更加得心应手。同时，我们小组的合作能力也有了很大的提升，不再是手忙脚乱的感觉，整体都很有秩序。我们会尽自己最大的努力去完成最后一次实验，做到尽善尽美。