操作系统实验报告：实验三

姓名　学号

袁帅　12061113

周华平　13061108

郑承浩　12005036

张烜诚　13061101

1 需求说明

1.1 基本要求和提高要求

基本要求

通过本实验，要求学生能够了解 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。具体要求如下：

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

提高要求

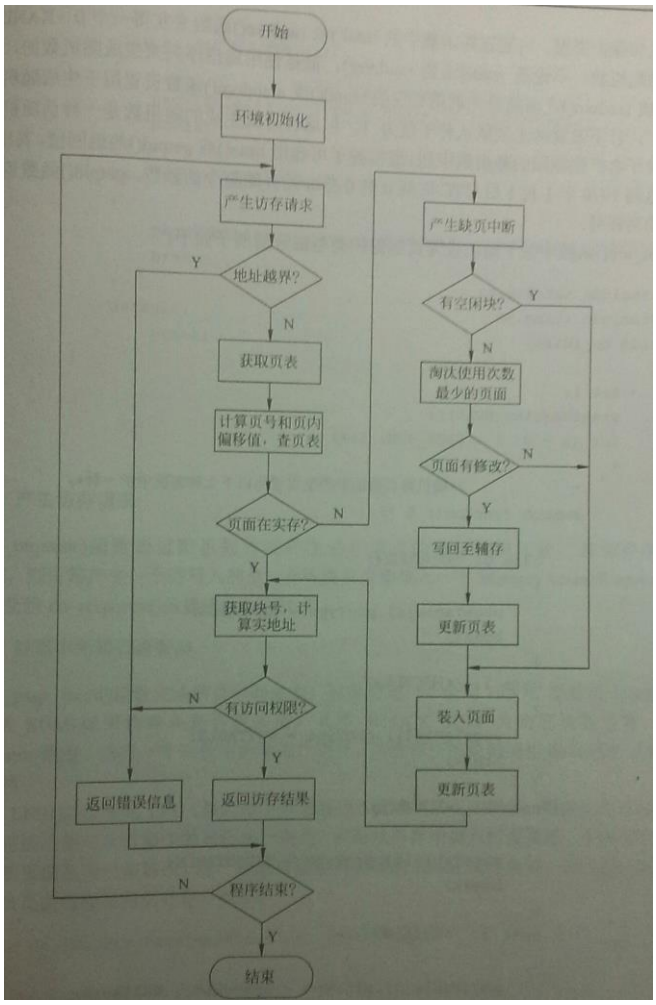
1. 建立一个多级页表
2. 实现多道程序的存储控制
3. 将 do_request()和 do_response()实现在不同进程中，通过进程间通信（如 FIFO）完成访存控制的模拟
4. 实现其它页面淘汰算法：如页面老化算法、最近最久未使用淘汰算法（LRU）、最优算法（OPT）等

1.2 完成情况

我们组本次完成了所有基本要求以及提高要求。

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

名称	作用	返回值
read(int fd,void * buf ,size_t count)	read()会把参数 fd 所指的文件传送 count 个字节到 buf 指针所指的内存中。文件读写位置会随读取到的字节移动。	若参数 count 为 0，则 read 为实际读取到的字节数，如果返回 0，表示已到达文件尾或是无可读取的数据
remove(const char *	remove()会删除参数 pathname 指定的文	成功则返回 0，失败则

pathname)	件。	返回-1，错误原因存于errno
mkfifo(const char * pathname,mode_t mode)	mkfifo ()会依参数 pathname 建立特殊的 FIFO 文件，该文件必须不存在，而参数 mode 为该文件的权限（mode%~umask），因此 umask 值也会影响到 FIFO 文件的权限。	若成功则返回 0，否则返回-1，错误原因存于 errno 中

2.3 提高要求实现说明

1. 建立多级页表

将页表改为 PageTableItem pageTable[OUTER_PAGE_SUM][INNER_PAGE_SUM];即 4*16 的二维数组来存储。其中 OUTER_PAGE_SUM 代表一级页表数目，INNER_PAGE_SUM 代表一级页表项下二级页表的数目。

在访问虚存时通过以下操作来取得页号及页内偏移值：

```
/* 计算页号和页内偏移值 */
pageIndex = ptr_memAccReq->virAddr / OUTER_PAGE_SIZE;
pageNum = ptr_memAccReq->virAddr / PAGE_SIZE - pageIndex * INNER_PAGE_SUM;
offAddr = ptr_memAccReq->virAddr % PAGE_SIZE;
printf("页目录号为：%u\t页号为：%u\t页内偏移为：%u\n", pageIndex, pageNum, offAddr);
```

在需要遍历页表项时采用如下的二重循环来遍历：

```
for (i = 0; i < OUTER_PAGE_SUM; ++i){
    for (j = 0; j < INNER_PAGE_SUM; ++j){
        printf("%u\t%u\t%u\t%u\t%u\t%s\t%lu\t%lu\t%02X\n", i, j, pageTable[i][j].blockN
            pageTable[i][j].edited, get_proType_str(str, pageTable[i][j].proType),
            pageTable[i][j].count, pageTable[i][j].auxAddr, pageTable[i][j].shiftReg);
    }
}
```

2. 实现多道程序的存储控制

需要在页表项以及访存请求中添加所属进程号 proccessNum

在处理对应请求时应该判断请求与请求页的进程号是否相同，如果不是则报错：

```
/* 获取对应页表项 */
ptr_pageTabIt = &pageTable[pageIndex][pageNum];
//检查是否为对应进程
if (ptr_pageTabIt->proccessNum != ptr_memAccReq->proccessNum){
    do_error(ERROE_PROCESS_DENY);
    return;
}
```

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

源码中的 do_request()单独抽出来放在一个 req.c 文件中，并进行修改使其支持手动输入请求。

在 vmm.c 中创建并初始化 fifo 文件:

```
//初始化FIFO
void init_fifo(){
    struct stat statbuf;
    if (stat("/tmp/server", &statbuf) == 0){
        /* 如果FIFO文件存在,删掉 */
        if (remove("/tmp/server")<0){
            printf("remove failed");
            exit(-1);
        }
    }

    if (mkfifo("/tmp/server", 0666)<0){
        printf("mkfifo failed");
        exit(-1);
    }
    /* 在阻塞模式下打开FIFO */
    if ((fifo = open("/tmp/server", O_RDONLY))<0){
        printf("open fifo failed");
        exit(-1);
    }
}
```

然后再 main 函数中循环读取 fifo 文件中的指令，并对其进行处理：

```
//从FIFO中读取命令
//memset(ptr_memAccReq, 0, DATALEN);
if (read(fifo, ptr_memAccReq, DATALEN)==DATALEN){
    printf("收到请求\n");
    do_response();
    do_update();
```

另一方面，在 req.c 文件中，需要产生请求并写入 fifo 文件中：

```
if ((fd = open("/tmp/server", O_WRONLY))<0){
    printf("req open fifo failed");
    exit(-1);
}
if (write(fd, ptr_memAccReq, DATALEN)<0){
    printf("req write failed");
    exit(-1);
}
```

4. 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

本次实验我们组实现的是页面老化算法。

首先，需要在页表项中加入 shiftReg 移位寄存器以及 r 页面访问位:

```
/* 页表项 */
typedef struct
{
    unsigned int pageIndex;//页目录号
    unsigned int pageNum;//页号
    unsigned int blockNum; //物理块号
    unsigned int proccessNum;//该页所属的进程号
    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 char shiftReg;//移位寄存器，用于页面老化算法
    BOOL r;//页面访问位，表示最近一段时间该页面有无被访问
} PageTableItem, *Ptr_PageTableItem;
```

在 vmm.h 中定义页面老化算法的周期，这里为了效果明显所以设置成 1：

```
/*页面老化算法的周期*/
#define CYCLE 1
```

每当页面被访问时，访问标识置位:

```
    /* 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->r = TRUE;
```

在 do_response();之后添加一个 do_update();用于定期更新页表中的移位寄存器：

```
void do_update(){
    int i, j;
    if ((++count) >= CYCLE){//到达更新时间片的时间
        for (i = 0; i < OUTER_PAGE_SUM; ++i){
            for (j = 0; j < INNER_PAGE_SUM; ++j){
                //将shiftReg右移1位
                pageTable[i][j].shiftReg = pageTable[i][j].shiftReg >> 1;
                //判断读取位
                if (pageTable[i][j].r){
                    pageTable[i][j].shiftReg = pageTable[i][j].shiftReg | 0x80;
                    pageTable[i][j].r = 0;
                }
            }
        }
        count = 0;
    }
}
```

将原本的 do_LFU 替换为页面老化算法的函数 do_LRU:

```
/* 根据页面老化算法进行页面替换 */
void do_LRU(Ptr_PageTableItem ptr_pageTabIt){
    unsigned int i, j, index, page;
    unsigned char min;
    printf("没有空闲物理块，开始进行页面老化页面替换...\n");
    for (i = 0, min = 0xFF, index = 0, page = 0; i < OUTER_PAGE_SUM; i++){
        for (j = 0; j < INNER_PAGE_SUM; ++j){
            if (pageTable[i][j].filled==TRUE&&pageTable[i][j].shiftReg < min){
                min = pageTable[i][j].shiftReg;
                index = i;
                page = j;
            }
        }
    }
    printf("选择第%u_%u页进行替换\n", index, page);
    if (pageTable[index][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[index][page]);
    }
    pageTable[index][page].filled = FALSE;
    pageTable[index][page].count = 0;
    pageTable[index][page].shiftReg = 0;
    pageTable[index][page].r = FALSE;
    /* 读辅存内容，写入到实存 */
    do_page_in(ptr_pageTabIt, pageTable[index][page].blockNum);

    /* 更新页表内容 */
    ptr_pageTabIt->blockNum = pageTable[index][page].blockNum;
    ptr_pageTabIt->filled = TRUE;
    ptr_pageTabIt->edited = FALSE;
    ptr_pageTabIt->count = 0;
    ptr_pageTabIt->shiftReg = 0;
    ptr_pageTabIt->r = FALSE;
    printf("页面替换成功\n");
}
```

3 收获与感想

3.1 给予你帮助的人

周华平，郑承浩，张烜诚

3.2 从实验中学到的东西

本次实验的主题是虚存管理，虽然并没有真正在底层，通过系统调用等手段实现访存控制虚存管理等目标，而是用文件模拟外存，数组模拟内存来模拟实现虚存管理系统，但是这并不妨碍我们对知识本身的理解。我们从中体会到了，之所以采用虚存，是因为实际内存极其有限，而对内存的需求却很大，加上多道程序并发执行导致实存的空间被离散化，每一个

程序申请到的内存空间难以避免的出现离散，这就需要通过页表来建立起实存与虚存之间的桥梁，使得每一道程序都能拥有连续的逻辑内存地址空间，少量的实际地址空间加上外存空间的辅助能够满足大量逻辑地址空间的需求。而实存与虚存在空间上的巨大差异就需要页面淘汰机制来支撑。我们了解了 LFU 算法，页面老化算法，OPT 最优算法等等这些页面淘汰算法在理论和实践中的应用，并实现了页面老化算法。为了减少实存开销，减少实存中保存的页表大小，我们尝试着实现了多级页表，并通过加权限位标志位和访存检查的方式支持多道程序的访存请求。同时，我们还将请求生成和处理的函数分离到两个进程并发执行，并通过进程间通信来交流，这是对我们之前所学知识的巩固。

这次试验我们是分团队合作完成的。团队编程和个人编程的区别就是需要通过大量的沟通保证工作之间良好的协调，每个人写的程序要提前留好“接口”，这样最后整合的程序才能成为一个整体。这次试验为我们今后的团队项目开发给予了很多的经验，我们每一个人都受益良多。

3.3 任务分配

12061113 袁帅 负责基础要求、页面老化算法的实现以及全程的组织协调。

13061108 周华平 实现了多道程序控制，并整合小组成员的程序使其测试通过。

12005036 郑承浩 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

13061101 张烜诚 实现多级页表