**操作系统课程设计实验报告**

——实验三：虚存管理

负责人姓名：黄秋宇

学号：14061176

日期：2016.5.7

**小组成员**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 姓名 | 学号 | 实验分工 |
| 1 | 李博 | 14061187 | 实验一 |
| 2 | 胡亚龙 | 14061196 | 实验二 |
| 3 | 黄秋宇 | 14061176 | 实验三 |
| 4 | 林子义 | 14061194 | 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 6](#_Toc446001838)

[3.3 提高要求实现说明 7](#_Toc446001839)

[4.收获和感想 9](#_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.支持页表、辅存和实存内容的打印  2.支持请求命令的手动输入  3.地址转换是否正确  4.页面装入/页面交换是否正确  5.存取控制是否正确  6.读命令读取是否正确  7.写命令实存内容是否正确写入

## 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

## 2.3 完成情况

完成了以下功能：

# 支持页表、辅存和实存内容的打印

# 支持请求命令的手动输入

# 地址转换是否正确

# 页面装入/页面交换是否正确

# 存取控制是否正确

# 读命令读取是否正确

# 写命令实存内容是否正确写入

# 建立一个多级页表

# 实现多道程序的存储控制

# 将*do\_request()*和*do\_response()*实现在不同进程中，通过FIFO完成访存控制的模拟

# 实现页面老化算法

# 3.设计说明

## 3.1 程序流程图

## 1234

## 3.2基本要求实现说明

#define EXIT 1

#define READ 2

#define WRITE 3

#define EXECUTE 4

#define SWITCH 5

#define OUTPUT\_MEM 6

#define OUTPUT\_PAGETABLE 7

#define OUTPUT\_AUXMEM 8

# 支持页表、辅存和实存内容的打印

其中输入6为打印实存，输入7为打印页表，输入8为打印虚存

# 支持请求命令的手动输入

其中输入1退出程序；输入2为读请求，后面接地址；输入3为写请求，后面依次为地址、写入的值；输入4为执行请求，后面接地址。

# 地址转换正确，页面装入/页面交换正确，存取控制正确，读命令读取正确写命令实存内容正确写入

## 3.3 提高要求实现说明

## 建立一个多级页表

本实验中完成了二级页表。

源码中采用一级页表，直接可查询页表项得到真实地址。修改为二级页表后，代码如下：

A．二级页表直接定义成二维数组。

B．将原来的一级页表改成了三维数组表示多道程序的多级页表以及一个二维数组存储当前进程的多级页表。

PageTableItem fullpageTable[PROCESS\_NUM][STAGE1\_SIZE][STAGE2\_SIZE];

PageTableItem (\*pageTable)[8][8];

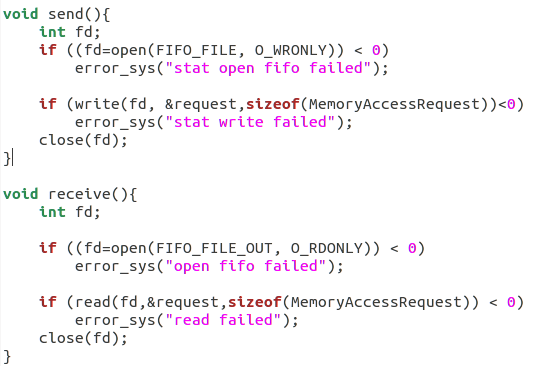
相应的初始化以及地址转换等一系列代码也发生了细微的变化。在此不列举。

## 2.实现多道程序的存储控制

本实验中初始将进程数设置为16（在vmm.h中可调PROCESS\_NUM）。将虚拟空间扩大到64\*4\*16，即每个进程对应赋存axuMem中64\*4的空间。将原来的一级页表转换成三维数组来表示多道程序的多级页表。初始化时将0号进程的二级页表随机装入物理块中，手动输入请求时，输入5为切换进程，后接进程号。其中输入的请求均视为在当前进程下的命令执行。  
 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

do\_request实现在不同进程：

利用命名管道FIFO，在新终端调用./request，根据输入参数生成请求，其中输入其中输入1退出程序；输入2为读请求，后面接地址；输入3为写请求，后面依次为地址、写入的值；输入4为执行请求，后面接地址；输入5为切换进程，后面接进程号；输入6为打印实存，输入7为打印页表，输入8为打印虚存。将请求写入FIFO，在gen\_request中创建FIFO并以只读打开。写入FIFO 以及创建并打开均是在gen\_request.c中完成的。



do\_response()

加入切换进程、打印实存虚存页表内容，以及修改了地址转换等问题。由于代码量较长，在此不加入代码截图。

4.实现页面老化算法

其中设置一个全局变量last\_time；每访问一次则该页面的ptr\_pageTabIt->count = last\_time ++；最后找出count的最小值进行页面替换即可。

# 4．收获和感想

个人感受：本次实验并没有突出多级页表和单级页表的优缺点，但通过查阅资料让我更加了解了为何使用多级页表。这是从管理成本来考虑的。由于是一个表，那么它便有连续内存存储的需求，这样才好根据索引来快速定位。如果是单层页表，那么即使一个页面被分配，也需要建立整个页表，32位的情况下以4K页面为例，需要20位要寻址页面基地址，20位的话单级页表需要一下子建立4M大小的页表。使用多级页表并不是为了减少内存使用，说实话，如果把所有的4G映射都建立页表项的话，采用两级页表还会浪费页目录表占用的4K空间，然而并不能如此考虑问题，内存使用分布是不遵循幂率的，因此你不必考虑黑天鹅事件。大部分情况下，不会建立太多的页表，即使建立1000个页表，它也会多数承载于连续的页目录项中，很多的页表是不需要分配内存的。主旨就是，将管理结构分级往前推，往前推，往前推！

除了页表，对于FIFO以及多道程序的模拟等，由于有前几次实验以及理论课的基础，实现起来比较简单也比较好上手。第一遍看代码时就发现了好几个奇怪的地方（bug），比如地址转换问题（前面的代码地址初始化时乘以2，到后面需要转换回去时就忘了乘以2……..）、LFU装入页面时的替换问题等，所以在读完代码之后基本上把基础要求实现了，相对来说比较顺利。实现提高要求时也没有遇到比较奇葩的bug，只是修改的代码比较繁琐、细节部分比较多，但总体相对比较简单，概念也不难理解，收获也是值得肯定的。