**操作系统课程设计实验报告**

——实验三：虚存管理

负责人姓名：孟祥鑫

学号：14061135

日期：2016.5.6

**小组成员**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 姓名 | 学号 | 实验分工 |
| 1 | 孟祥鑫 | 14061135 | 页表、辅存和实存内容的打印，添加命令的手动输入，实现多道程序的存储控制 |
| 2 | 韩世依 | 14061129 | 建立一个多级页表，BUG修改 |
| 3 | 崔博涵 | 14061127 | 实现LRU算法 |
| 4 | 史天泽 | 14061121 | 实现进程间通信（FIFO） |

目录

[1.实验目的](#_Toc446001831)

[2.需求说明](#_Toc446001832)

[2.1基本要求](#_Toc446001833)

[2.2 提高要求](#_Toc446001834)

[2.3 完成情况](#_Toc446001835)

[3.设计说明](#_Toc446001836)

[3.1基本要求实现说明](#_Toc446001838)

[3.2 提高要求实现说明](#_Toc446001839)

4.运行展示

[5.收获和感想](#_Toc446001840)

6.发现的问题和解决方法

# 1.实验目的

# 了解Linux的内存管理机制学习

# 掌握页式虚拟存储技术，理解虚地址到实地址的定位过程

# 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

## 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

## 建立一个一级页表

## 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

## 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

## 实现LFU页面淘汰算法

## 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（ do\_request() 函数是随机生成请求）

## 2.2 提高要求

## 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，通过进程间通信（如 FIFO）完成访存控制的模拟

## 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 完成过程概述

【简述实验完成过程】

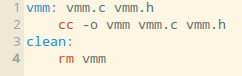
1. 程序能够正常运行；
2. 实现了对页表、辅存和实存内容的打印功能；
3. 添加了手动输入请求的功能，对照源代码中的do\_request函数写了一个do\_request\_manual函数，用于实现手动输入请求，并在main中实现了对测试者自动还是手动运行请求的选择控制；
4. 实现了多道程序的存储控制；
5. 建立了一个二级页表；
6. 实现了进程之间的通信（FIFO）完成访存控制的模拟；
7. 实现了最近最久未使用淘汰算法（LRU）。

# 3.程序功能实现

## 3.1基本要求实现说明

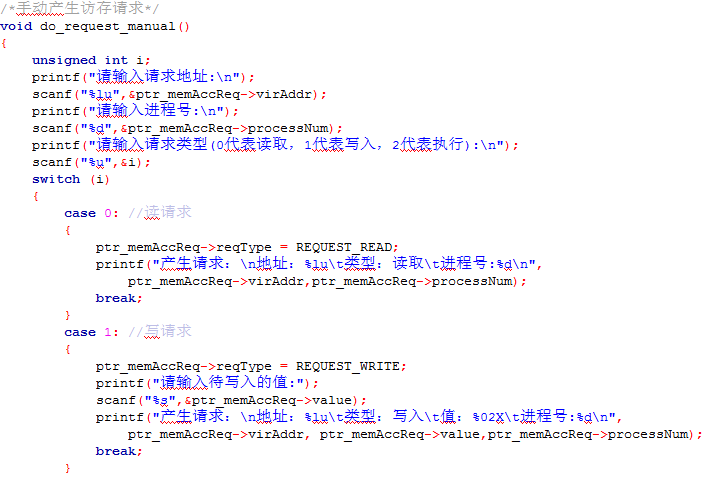
1、程序的正常运行

首先需要自己编写makefile文件，在未实现进程间通信功能时，只需要添加如下内容，之后在终端make，便生成了可执行文件，之后运行./vmm即可：



2、增加手动输入请求命令的功能

仿照源代码中自动生成请求命令的do\_request函数，新建了一个名为do\_request\_manual的函数，函数如图1所示，并在main中建立分支条件，让用户能够自己选择手动或者是自动输入请求，如图2所示。



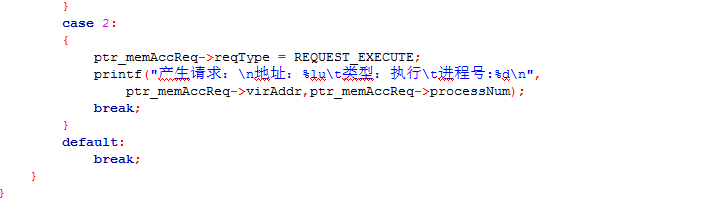


图1

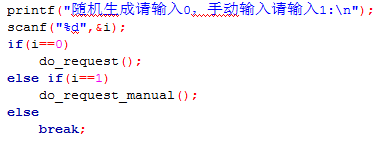
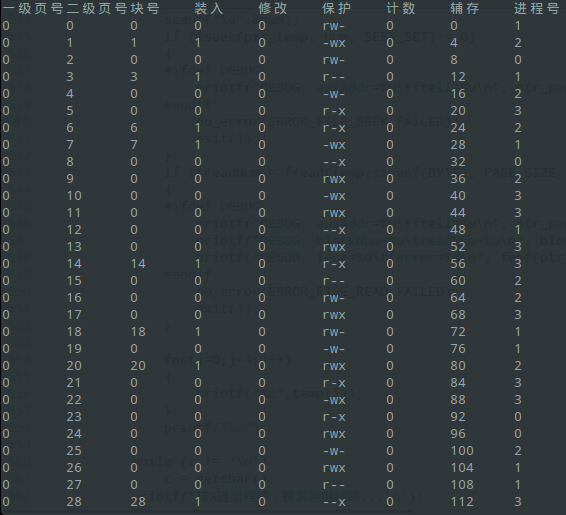


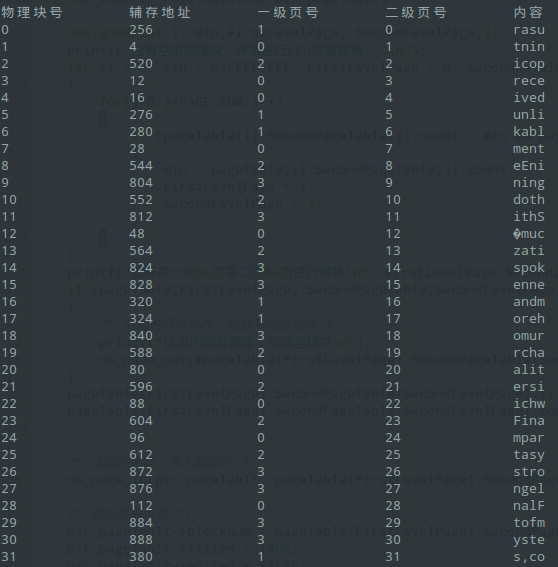
图2

3、打印页表、辅存和实存的内容

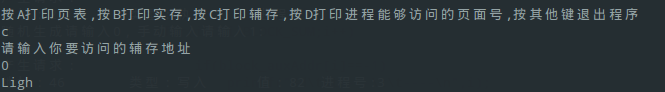
打印页表：在之前源码打印内容的基础上增加了对应的一级页表、二级页表和能够被访问的进程号的打印，使得测试时更方便。如图：



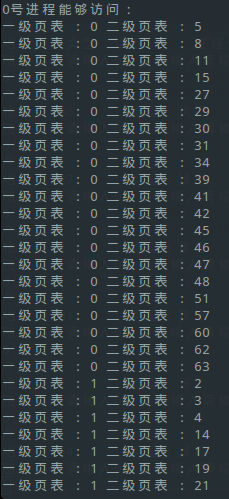
打印实存：



打印辅存：



此外还增设了集中打印进程号能够访问的页面号的内容：



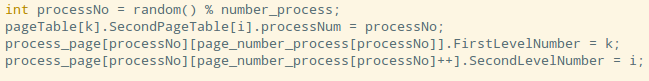
## 3.2 提高要求实现说明

1、实现多道程序存储控制

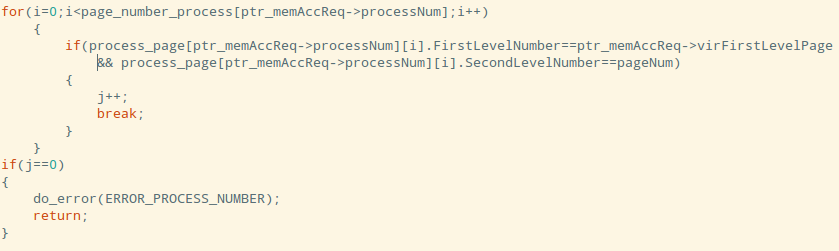
首先在PageTableItem数据结构与MemoryAccessRequest数据结构中添加属性unsigned int processNum，作为进程号的标识。

之后，开始增加访问页面时的进程匹配功能，只有页面标识的进程号所对应的进程才能够对页面进行访问与操作，代码如下：

初始化，随机分配进程号：



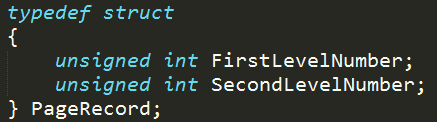
在do\_response中添加进程号匹配的相应内容，如果匹配则可以正常访问，否则会报出do\_error(ERROR\_PROCESS\_NUMBER)的错误。



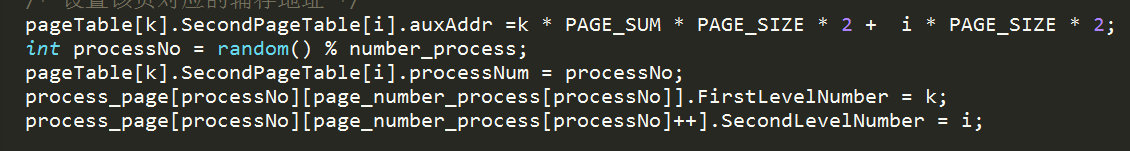
2、建立一个二级页表

建立多级页表的思路大体就是让原来单纯的记住虚拟页号变为了二维的先找到一级页表再从一级页表给出的二级页表的位置寻找二级页表中你想找到的表项，

为此，我们建立了FirstLevelPageTable数据结构来保存二级页表，由于实际的计算机中，是在使用二级页表的时候将其调入内存当中，而我们这次实验的物理内存只有32\*4，要是调入内存感觉效果不会很好，所以就只是单纯的二级索引实现。



用新的数据结构来保存对应进程的地址，一级页表号和二级页表位置。

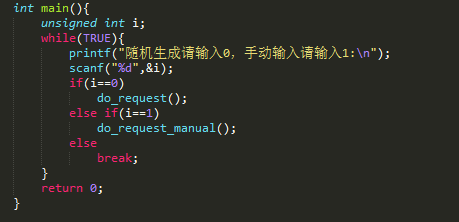


在初始化的时候也针对多级页表进行了改动，这次实验并没有设置多个进程访问同一个页的功能。

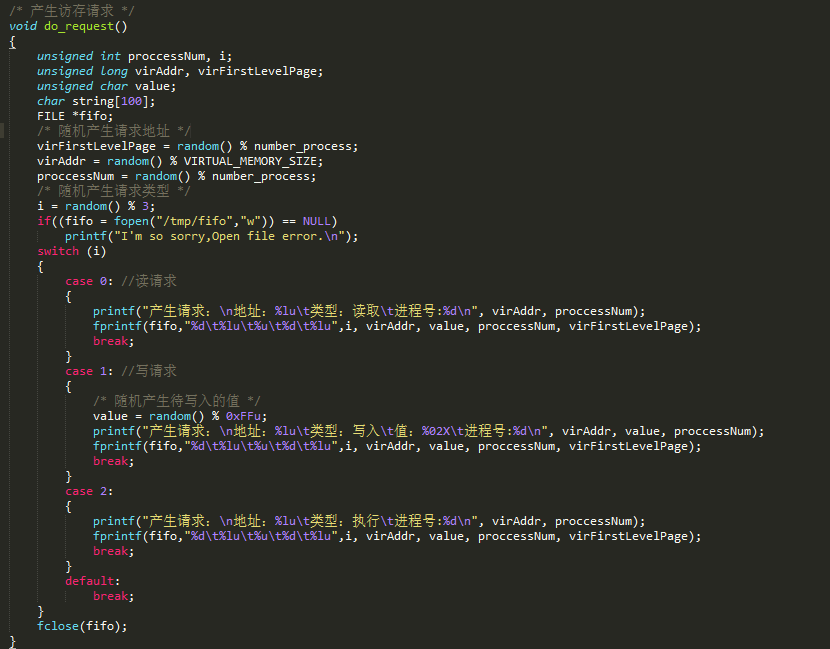
3、实现进程之间的通信

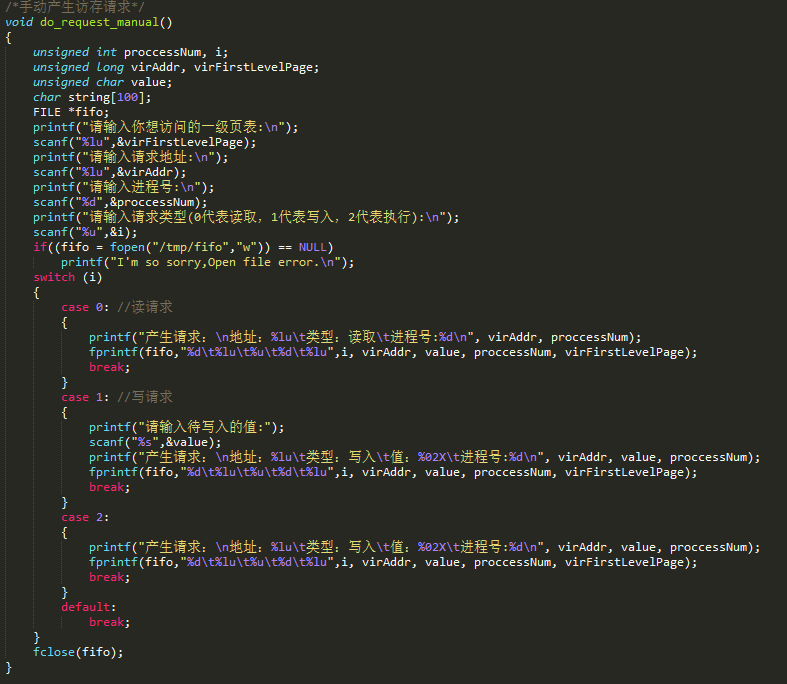
我们组在实现进程间通信时，是将do\_request单独拎出去写成了request.c文件，然后通过写入一个临时文件，然后再由主进程读取该文件内容，将读取的内容赋给需要的值，以此实现通信。代码如下：

request.c的主函数调用写的do\_request()和do\_request\_manual()实现生成请求，然后将请求值写入文件中：

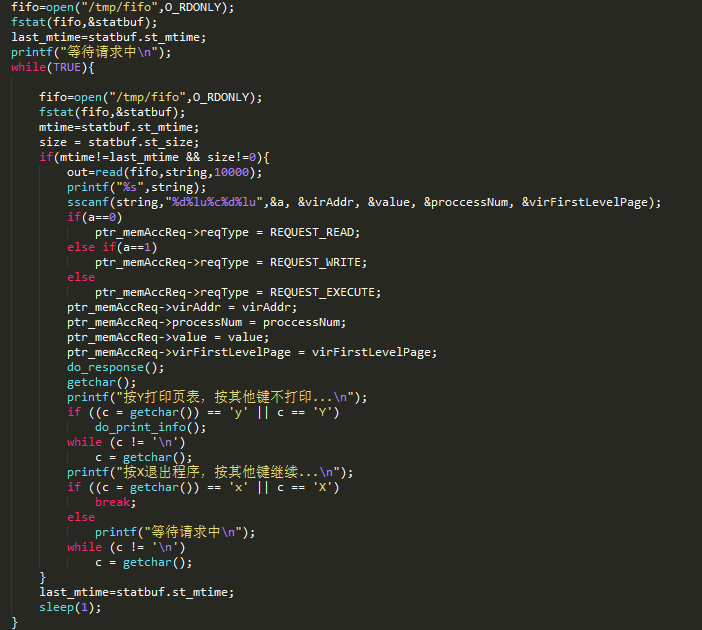


do\_request()和do\_request\_manual()函数也分别进行了改写：



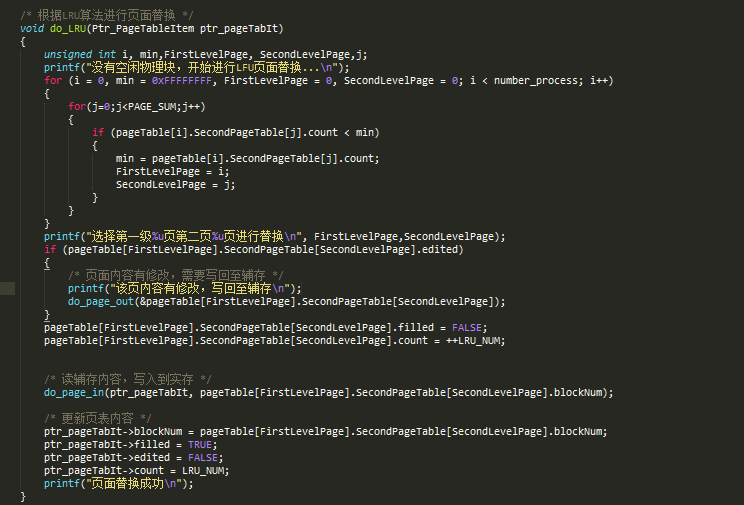


然后在vmm.c的main()函数中每当检测到目标文件发生变化时读取请求信息：



4、实现最近最久未使用淘汰算法（LRU）

为了更好的运用资源，我还是用了页表的count这个变量，想法是先设置一个int的值LRU\_NUM，每次调用页表，count值变为++LRU\_NUM，这样被调用过的值就会变大，没被调用的值就很小，在替换时，只需要找到所有二级页表中count值最小的那个即为最近最久未被使用的。

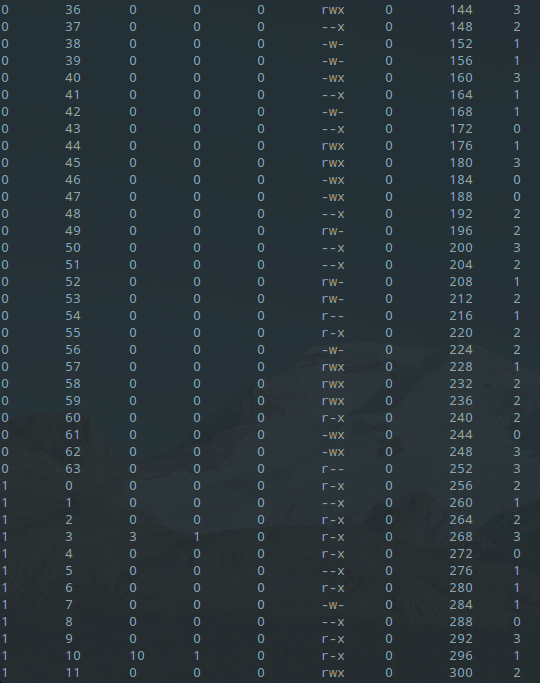
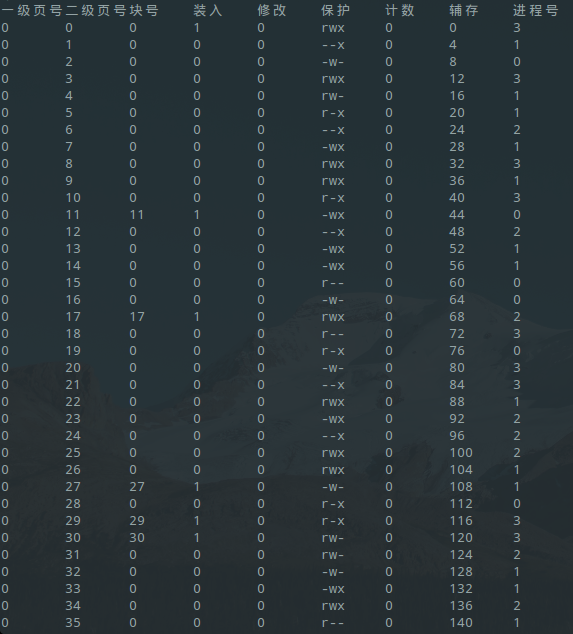


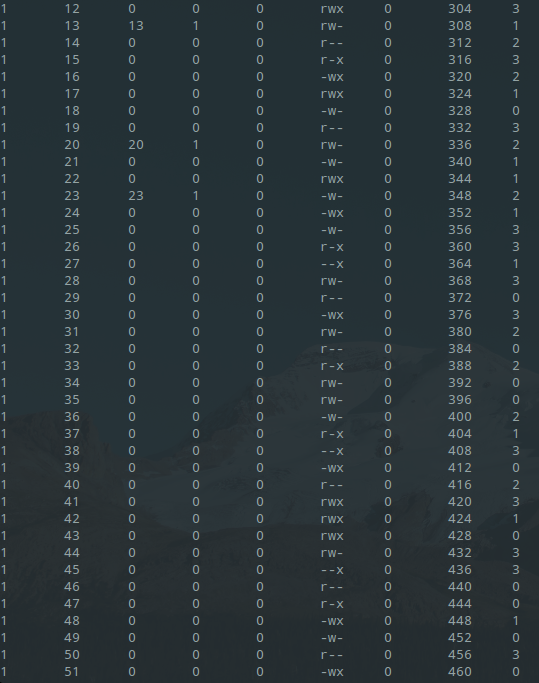
# 4.运行展示

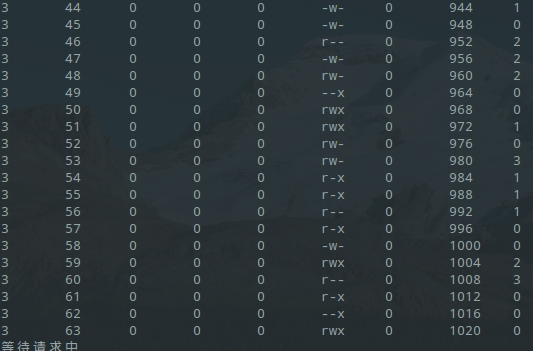
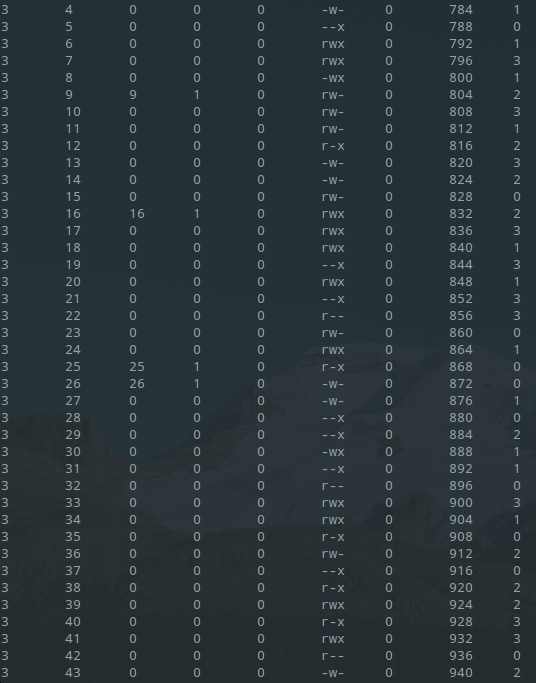
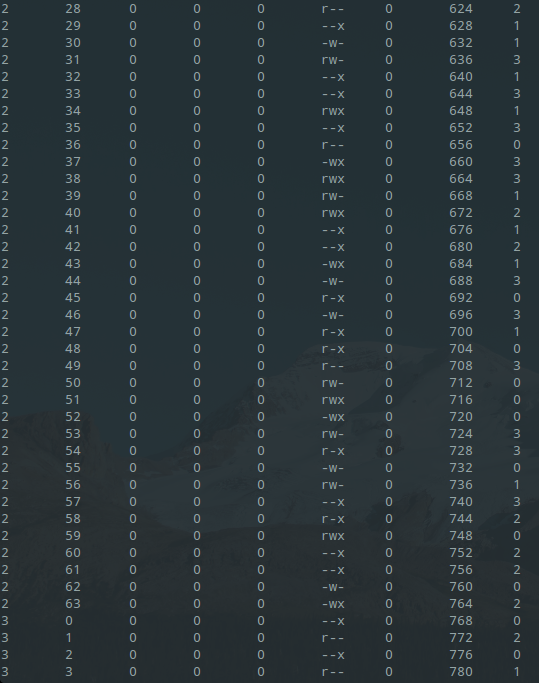
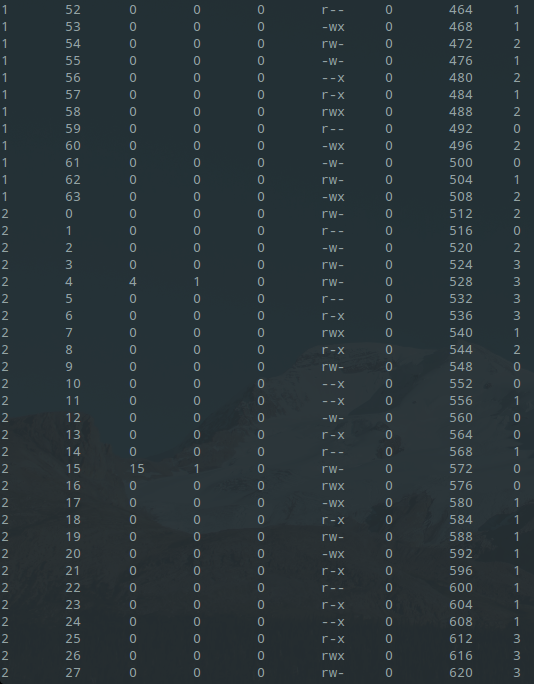
终端输入./vmm开始运行，程序进行初始化，并且打印已经存入内容的物理块号。



之后打印一次页表，便于测试者查看相关信息。

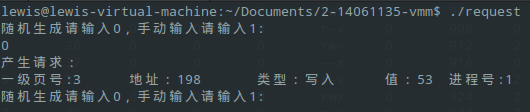




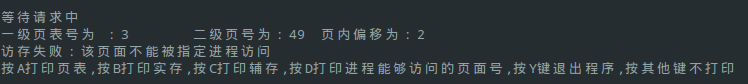


打印以上内容后，vmm进程进入等待状态，这时候再打开新的终端，运行request。

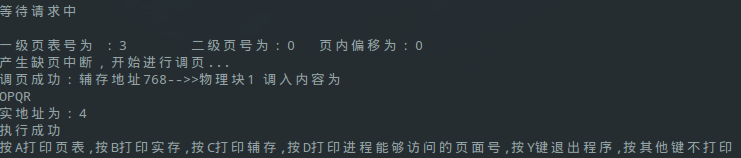
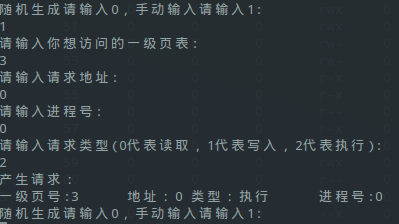
0表示自动生成请求，1表示手动生成请求。



请求输入完毕回车后，发现vmm进程输出此次请求的执行信息，是否成功。



接下来是手动输入的一个实例，我先查找了页表，找到一个未被装入的页面，记录它的一级页号，二级页号，能够被那个进程访问，能够实现读写执行中的哪个功能。之后，在request中手动输入确保能够正确调入的信息，之后发现request中出现了正确的调入操作，并且显示执行操作也是成功的。



整体而言，此次实验我们测试了所有的可能情况，包括，已经存入到物理块中的页面的读写执行操作，在实存未满情况下的调入操作，在实存已满情况下的调入操作，在实存内容被改变且被调出时辅存是否响应更新了内容，调入调出中断的操作是否严格按照算法设计去执行，进程之间的通信是否可行，是否存在读写错误的情况，确定页面是否只能够被相应进程访问，二级页表的页面大小等属性是否依照一级页表的规则来写等等。经过我们的测试，发现以上功能都已得到正确实现。

# 5.收获和感想

虽然之前两次实验不是自己负责的，但是也在小组实验群策群力、共同奋斗的过程中收获了很多，同时也为此次实验打下了良好的基础。此次实验难度不算大，助教给的源码基本能够全文看懂。在对此次实验的工作内容进行梳理之后，便召开小组会议，解决了分工问题。在实验的过程中，很享受与小组成员讨论的过程，在最需要帮助的时候，其他成员能够踊跃站出来替自己扛一扛，这既是一种感动，也是一种激励。此次试验的硬性要求相比上两次实验更少，像是实现多道程序控制功能，更像是一个讨论创新性的问题，可以有多种实现模式。还有在淘汰算法的选择上，也给足了大家自由，可以挑选其一进行方法替换。如果说之前的两次实验更看重于自学内容的话，我认为这次实验更看重于团队协作。四个提高要求在4个小组成员的手中慢慢变成了能够实现的代码，这是一件非常美妙的事情，同时我也很享受作为中间人，进行4个程序的合一拼接的过程。最终展现在同学们眼前的，是小组成员共同努力的结果，是互相帮助、互相砥砺后绽放的美丽花朵。

# 6.实验遇到的问题和解决办法

由于此次实验难度适中，分工也比较均衡，因此每个人都有时间对自己的工作范围内的事项进行比较透彻的学习与体会。如果说有比较难的一点，那就是在实现进程间通信（FIFO）的时候，我们写了一个FIFO文件，把它当作vmm进程和request之间的通信站。但是在vmm读取FIFO的时候，有时候会出现读错的情况，但是打开FIFO文件看看却是正常的读入内容。经过一遍又一遍的修改测试，最终发现，读入错误是由于没有将前一次的读入内容清空导致的。因此，我们在实现了每写入一次便清空一次的功能，之后测试时，内容的传送便没问题了。