《操作系统原理》实验报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 李欣宇 | 学号 | U201911658 | 专业班级 | 信安1901 | 时间 | 2022.1.4 |

**一、实验目的**

1）理解页面淘汰算法原理，编写程序演示页面淘汰算法。

2）验证Linux虚拟地址转化为物理地址的机制

3）理解和验证程序运行局部性的原理

**二、实验内容**

1）Win/Linux编写二维数组遍历程序，理解局部性的原理。

2）Windows/Linux模拟实现OPT和LRU等淘汰算法。

3）Linux下利用/proc/pid/pagemap技术计算某个变量或函数虚拟地址对应的物理地址等信息。

前两部分在win10下实现，第三部分在优麒麟下实现。

**三、实验过程**

**3.1 Win下二维数组遍历**

（1）源码

定义一个20480\*10240的二维数组myArray，可以选择大数组遍历或者小数组遍历，大数组即为20480\*10240，小数组为2048\*1024，两种遍历方式，分别为先行后列、先列后行遍历，在遍历前后分别放置两个clock()进行计时，使用GetcurrentProcessId()函数获取当前进程的pid号，根据pid号可以在任务管理器中查看错误页面，观察缺页次数。

以上组合起来共四种情况——大数组先行后列遍历、小数组先行后列遍历、大数组先列后行遍历、小数组先列后行遍历，四组可以分别对照，观察运行时间和页面错误情况。

具体代码如下：

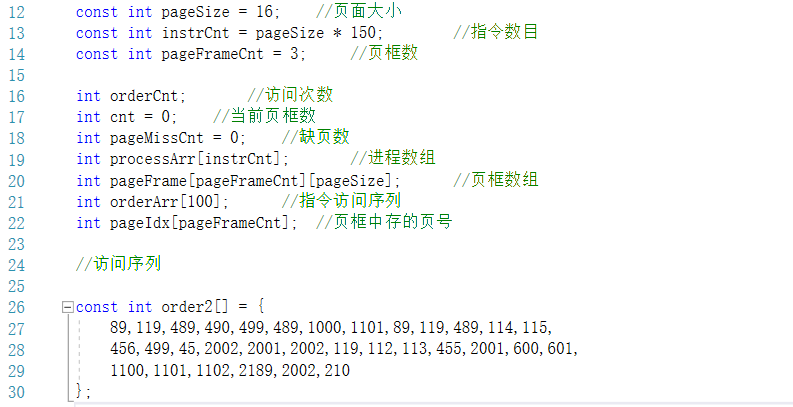


**3.2 Win下模拟OPT和LRU淘汰算法**

首先进行一些全局常量的定义，包括：页面大小pageSize，此处设置为16（字节）；进程中的指令数instrCnt，此处设置为页面大小的150倍；物理内存中页框数pageFrameCnt。

以上三个变量均可根据需要进行调整。

然后定义以下全局变量用于记录模拟过程中的一些信息，包括用于记录当前物理内存中已经装有数据的页框数目cnt；用于记录模拟进程执行过程中缺页次数pageMissCnt；进程数组processArr，其容量为instrCnt，数组中每个元素都表示一个指令，值为一个随机数；带访问的指令序列orderArr，其每一个元素记录了指令序列号，即在processArr数组中的索引；以及用于记录物理内存中每个页框存入页的页号的数组pageIdx，其容量为页框数pageFrameCnt；最后order1和order2分别为测试用的指令访问序列，运行时可以使用以上两个序列，也可以随机生成访问序列。



然后定义了一些模拟页面淘汰过程中用到的一些辅助函数，包括：初始化进程中的指令的函数initProcess；将进程中的一页复制到内存的页框中的函数copyPage；输出访问序列orderArr的函数showOrderArray；初始化指令访问序列的函数initProcessOrder，该函数可以设置访问序列为给定的序列，或者随机生成一组访问序列；输出页框状态的函数showPageFrame，可以输出当前页框中分页的页号、内容。



接下来是核心淘汰算法的编写：

1）OPT：

该算法的思想是淘汰以后不再使用或者最远的将来才会使用的页面。一般说来该算法实践中无法实现，而此实验中由于访问的序列有限，所以是可以模拟实现的。

该部分采用了一个待用信息表，表中记录所有需要访问的页面和其待用信息，该待用信息即该页的所有访问序号。此处采用一个映射实现，键名为页号，键值为一个堆栈，堆栈中将会存有该页面所有的访问序号。

算法首先要记录页面的待用信息：逆序遍历一遍指令访问序列，计算出其对应页号，若该页号未在待用信息表中，则表明此时该页面还未被访问过，则将该页号记录到待用信息表中，同时在堆栈中添加当前访问序号；若页号已经出现在表中，则直接将当前访问序号加入待用信息的堆栈中。遍历一遍访问序列后，整个待用信息表就构建完成，序列中所有会访问的页号都会记录在表中，而其对应的待用信息堆栈会记录该页面所有的访问次序，且由栈顶到栈底访问序号递增。

每次进行指令访问时都会先将其页面在待用信息表中堆栈的栈顶元素（即当前访问序号）出栈。当该指令的页面未命中且内存页框全部占满时，则在待用信息表中查找页框中所有页面的待用信息，若有页面的堆栈为空，则证明该页面在之后不会被需要，则直接选择该页面进行淘汰；否则比较待用信息堆栈的栈顶元素，该栈顶元素即为该页面下次访问的序号，找出序号最大者即为最远的将来才会被访问的页面，然后将其淘汰。





2）LRU

该算法的思路是淘汰内存中最长时间未被使用的页面。算法的实现思路也比较简单，即定义一个数组 timer，用于记录内存中每个已装有页面的页框未被使用的时间。当指令命中时，则对其页面所在页框的计时器置零，同时每次访问对所有有页面的页框时间加 1，这样 timer 中最大数对应的页框，其页面为最长时间未被使用的页面，则被淘汰

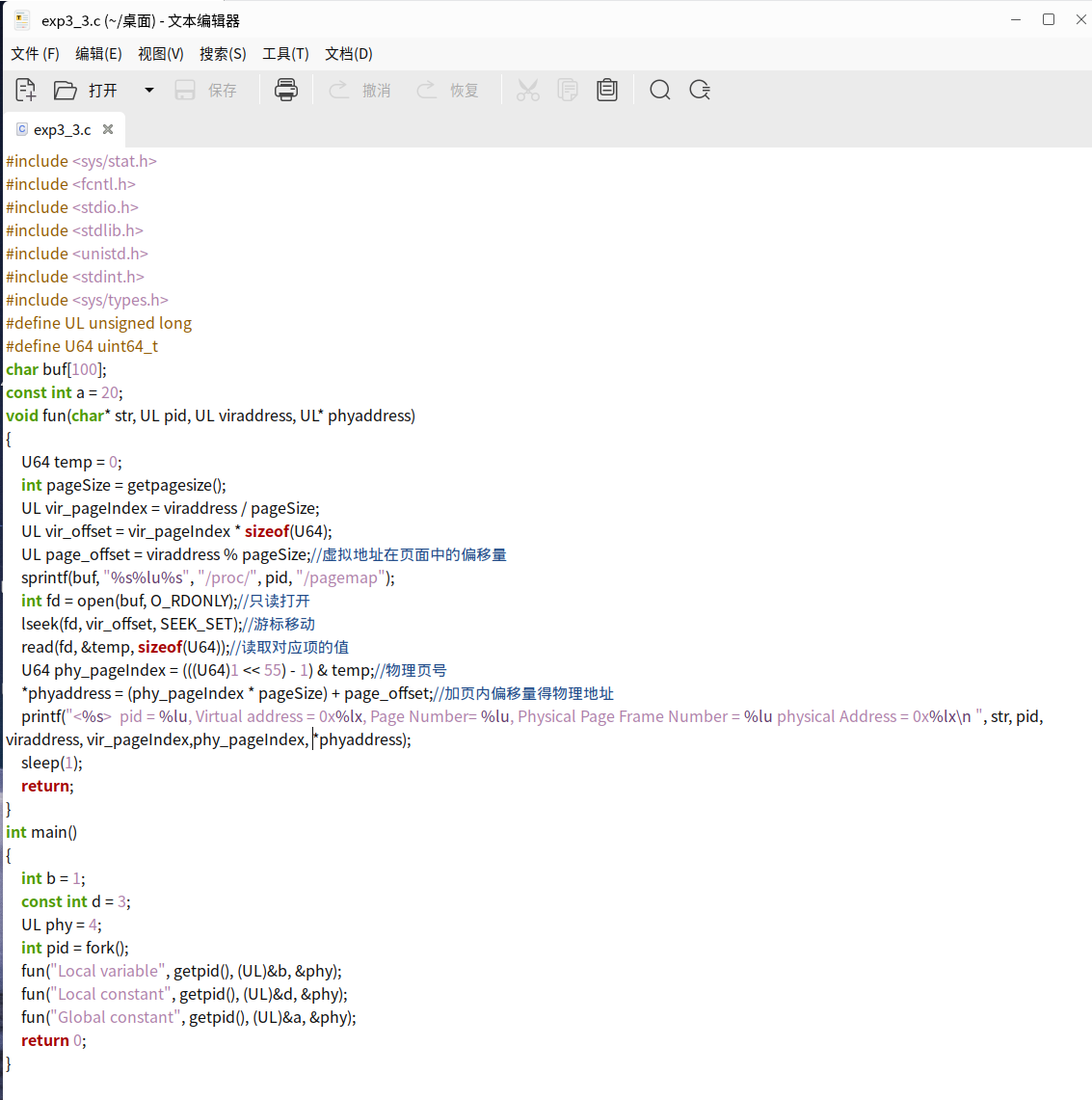




**3.3 Linux下计算虚拟地址对应的物理地址**

Linux的/proc/self/pagemap文件运行用户查看当前进程的虚拟页的物理地址相关信息，其中每个记录均为8字节64位，最高位记录了当前虚拟页是否在内存中，1表示在物理内存中，0表示不在物理内存中；当最高位为1即虚拟页在物理内存时，0~54位则记录了该虚拟页的物理页号。

因此，对于给定的虚拟地址，首先使用getpagesize()函数获取页面大小，利用虚拟地址除以页面大小可以得到虚拟页号vir\_pageIndex，虚拟地址对页面大小取模可以得到页内偏移page\_offset。利用pagemap文件，找到虚拟页号对应的8字节记录，并分析其最高位和第55位，即可找到该虚拟页号对应的物理页号phy\_pageIndex，进而加上页内偏移量计算出虚拟地址对应的物理地址phyaddress。

****

**四、实验结果**

**4.1 Win下二维数组遍历**

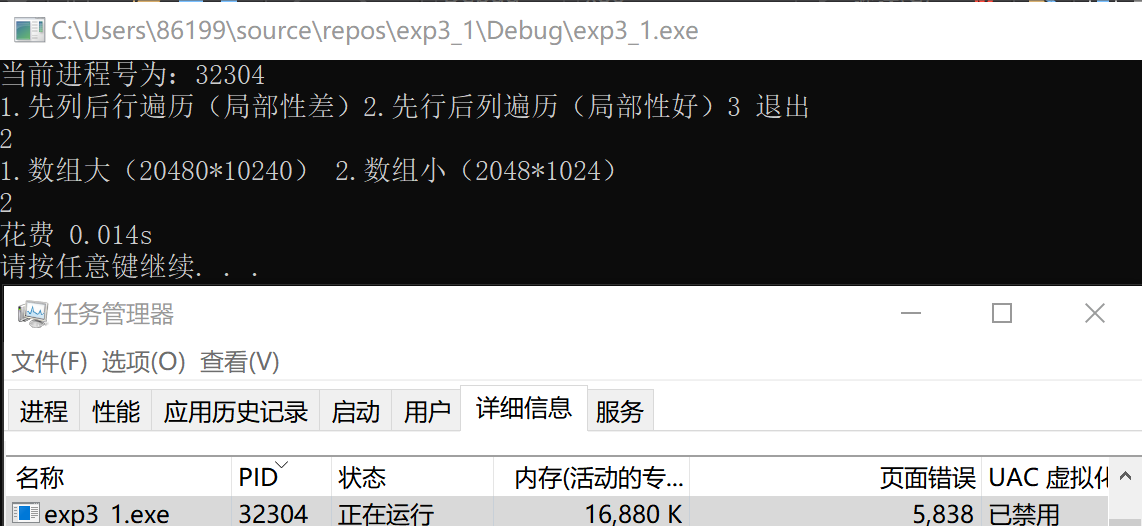
根据实验结果可知

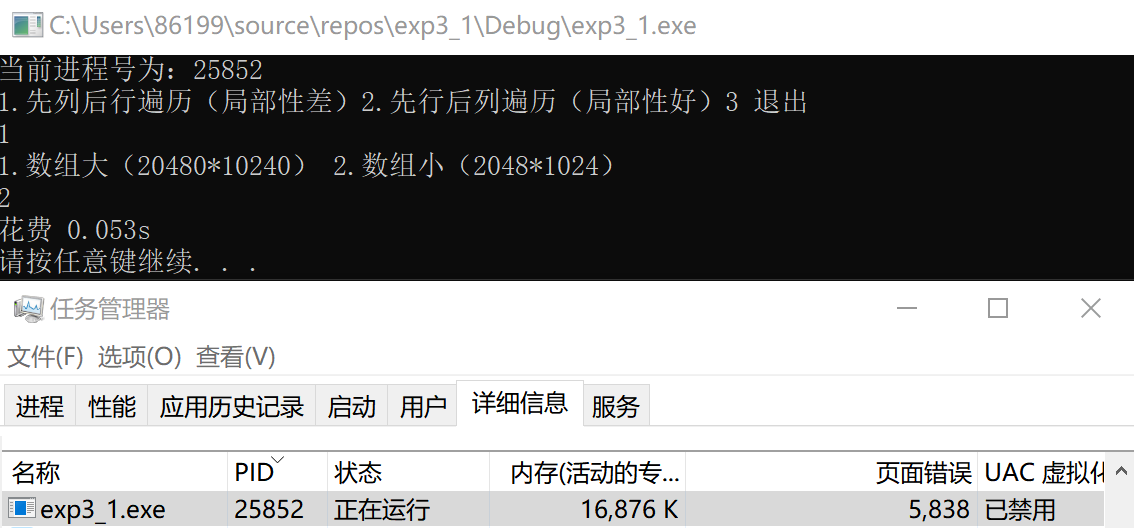
①访问数组时会将部分页面调到内存中，占用内存增加、页面错误增加。

②局部性差的情况需要消耗更多的时间。  
 ③局部性好坏不影响缺页次数。

****

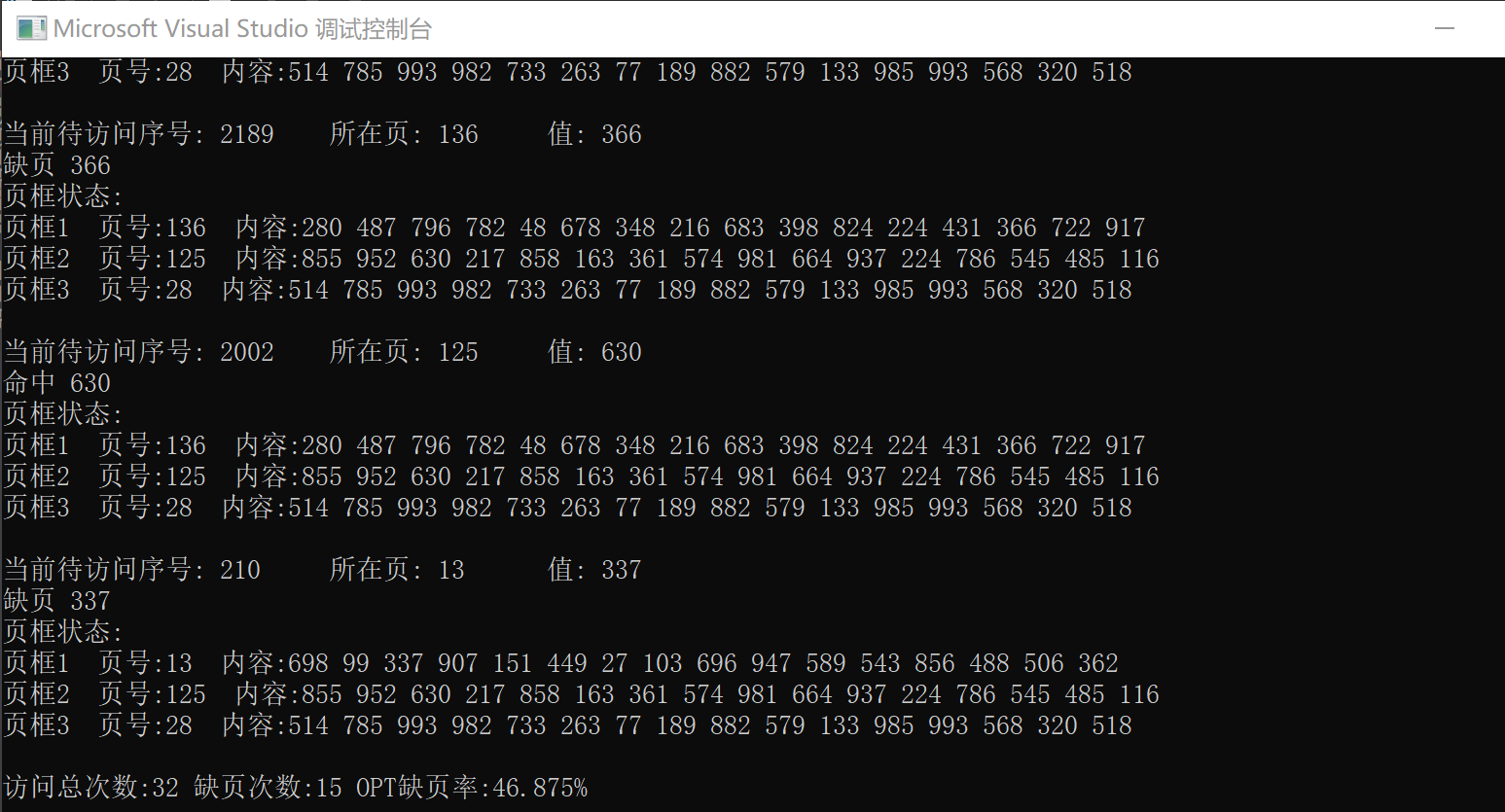
****

****

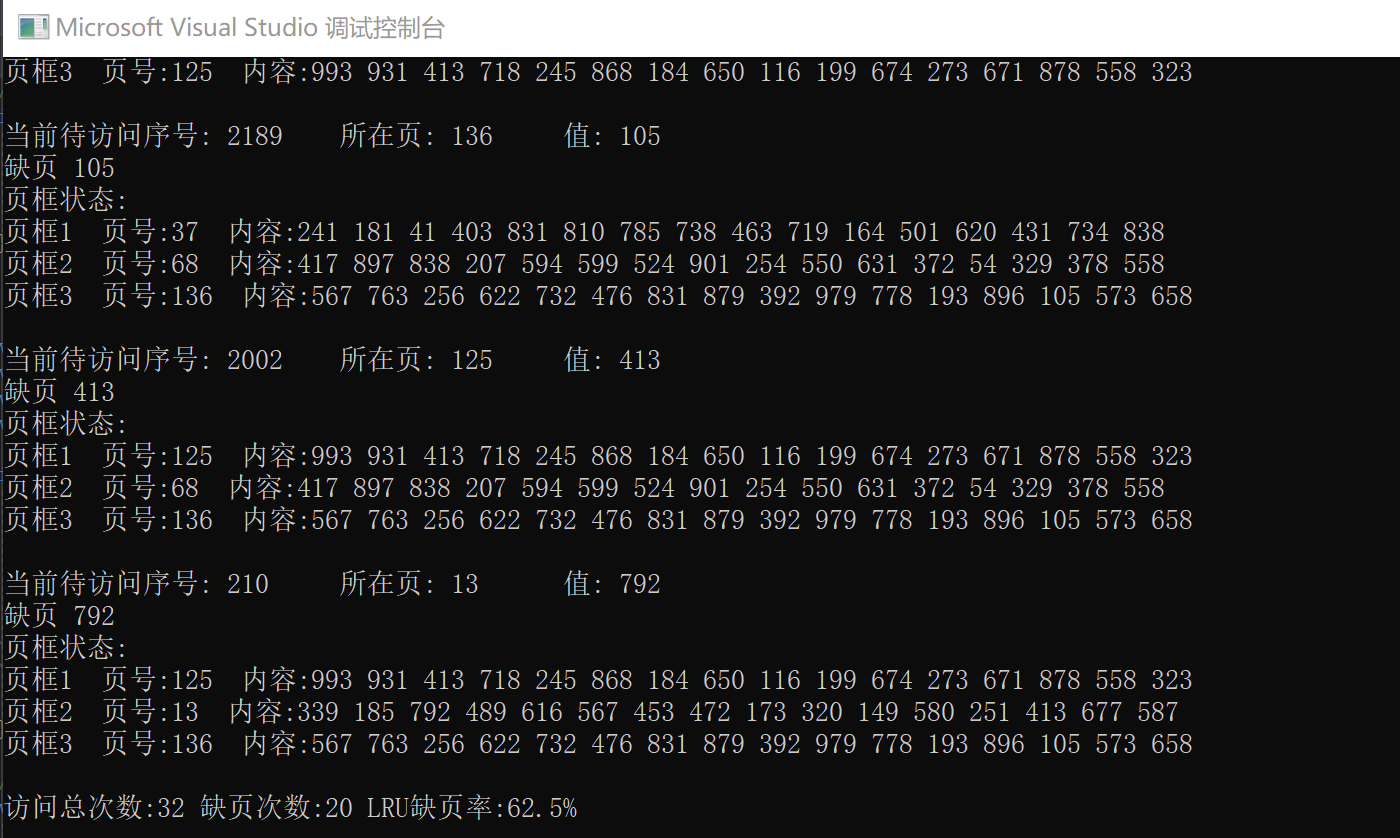
****

**4.2 Win下模拟OPT和LRU淘汰算法**

1）OPT

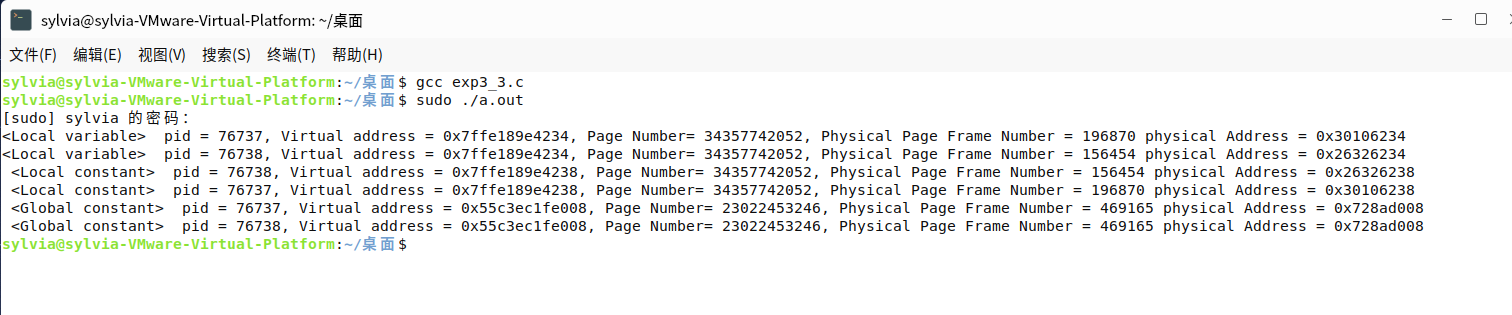
****

2）LRU

****

**4.3 Linux下计算虚拟地址对应的物理地址**

计算了局部变量、局部常量、全局常量的虚拟地址的物理地址



**五、实验错误排查和解决方法**

**5.1 Win下二维数组遍历**

1.在win10中跑会发现不同大小数组同样的遍历方式也会导致页面错误不同，并且同样的数组同样的遍历方式跑几次页面错误也是不同的，但是差别都比较小，在win7虚拟机中跑则一样的，目前该问题未得到解答，猜测可能是该程序编译时初始的一些要调用的数据在内存中的缺页情况不同（比如刚运行过一次的一些数据仍存在内存中，或过一段时间已经全部被淘汰）

**5.2 Win下模拟OPT和LRU淘汰算法**

LRU实现比较简单，在数电中已经学习过。

OPT原理也很简单，但是要保存下整个指令序列进行查找最远使用或再也不使用的指令比较难实现，这个是在同学的帮助下完成的，采用了编译原理中目标代码生成中对变量记录“待用信息和活跃信息”的方法，使用一个待用信息表，表中记录所有需要访问的页面和代用信息，该待用信息即该页面的所有访问序号，此处采用一个映射实现，键号为页号，键值为一个堆栈，栈中将会存有该页面所有的访问序号。

**5.3 Linux下计算虚拟地址对应的物理地址**

这里主要问题是运行./a.out物理页号总为0，经过阅读读文档“Documentation/v

m/pagemap.txt”发现，自 Linux4.0 之后，只有系统管理员才能获取到 PFNs 即物理页号，而普通用户只能得到物理页号为 0。因此需要在执行程序时加上“sudo”才能够正确运行程序得到变量正确的物理页号。

**六、实验参考资料和网址**

（1）教学课件

（2）https://www.cnblogs.com/pengdonglin137/p/6802108.html

（3）https://www.cnblogs.com/wasi-991017/p/13072328.html

（4）https://blog.csdn.net/qq\_42615643/article/details/97512361

（5）https://www.jianshu.com/p/544ee20e307c

（6）https://blog.csdn.net/wf824284257/article/details/53517264?utm\_term=opt%E7%A

E%97%E6%B3%95%E5%AE%9E%E7%8E%B0&utm\_medium=distribute.pc\_aggpage\_search\_result.none-task-blog-2~all~sobaiduweb~default-0-53517264&spm=3001.4430