

**《操作系统》**

**Linux 内存管理机制及页面淘汰算法探析**

|  |  |
| --- | --- |
|  |  |
| 学 号： |  |
| 姓 名： |  |
| 学 院： |  |
| 日 期： | 2022年 11 月 9 日 |

# 一、实验要求

Linux 内存管理机制及页面淘汰算法探析实验基本要求如下：

（1）下载和研读 Linux 内核源码（可以是任意版本）；

（2）围绕操作系统关于内存管理的内在实现机制，研读 Linux 内核对应源码（包括汇编代码、C 程序、Makefile 及相关配置文件），归纳总结操作系统实现内存管理的基本原理、关键环节（包括硬件平台相关部分和无关部分，同时涵盖系统启动及保护模式进入、地址映射与缺页中断处理、内存保护）及相关数据结构、分配与回收操作过程、页面淘汰算法等；

（3）完成相应 Linux 内核源码的编译和启用；

（4）在虚拟机平台上启用相应 Linux 内核，并通过运行特定命令或程序进行测试验证，检查确认系统相关显示信息和自己关于内核源码分析结果的一致性。

# 二、开发环境、运行环境、测试环境

开发环境：ubuntu 20.04

运行环境：ubuntu 20.04

测试环境：ubuntu 20.04

# 三、Linux内核源码的下载和研读（整体研读）

在Linux操作系统中，内核是其最基础也是最重要的模块，它有着了提供硬件抽象层、磁盘及文件系统控制、多任务等功能，以及一些其他计算机必须用到的组件。严格意义来说，Linux这个词本身也是专指Linux内核，但是目前大部分人都将Linux理解为整个Linux系统。本实验中我完成的内容是专指Linux内核部分源码的研读。

### 进程管理（Process Scheduler)

Linux内核对进程和线程没有特别区分，在实现上没有特别的调度算法或定义数据结构来标识线程，进程和线程都是用相同的进程PCB数据结构。而创建和操作进程需要通过fork()、vfork()、clone()的系统调用，最后进入内核空间。

内核里使用clone()方法来创建线程，其工作方式与fork()类似。但是会区分哪些资源与进程共享，哪些资源是线程独有。

  Linux系统通过fork()系统调用复制父进程来创建子进程，接着调用exec()函数来创建新的地址空间，并把新的程序载入其中。最终，程序通过exit()系统调用退出执行，进入僵死状态，等待父进程调用wait()或waitpid()函数回收其资源。

进程的所有信息被task\_struct数据结构所描述，通常称它为进程PCB。每一个进程对应一个PCB，系统中的所有PCB存放在一个双向循环链表中，称这个链表为任务[队列](https://so.csdn.net/so/search?q=%E9%98%9F%E5%88%97&spm=1001.2101.3001.7020)。内核中current全局变量利用内核栈的特性可以获取当前进程的PCB。首先，从SP寄存器中得到当前内核栈的地址，然后获取到 thread\_info数据结构的指针。进而 thread\_info->task的到PCB结构。

### Memory Manager内存管理

Linux内存管理引入了虚拟地址的概念，但本质上来说，内存管理的工作是对物理内存进行组织。这两者结合起来，就出现了Linux系统关于内存管理的一个重要的功能：将虚拟地址与实际地址对应。

首先，页（page）是内核空间管理的基本单位，内核用struct page结构体表示每个物理页，struct page占用40个字节。而内存管理单元（MMU,把虚拟地址转换为物理地址的硬件设备）通常是以页为单位处理。

而区（zone）是内核将页继续进行了划分，一般Linux系统共有三个区，包括：

ZONE\_DMA(DMA使用的页，物理内存<16MB)  
ZONE\_NORMAL(可以正常寻址的页，物理内存16—896MB)  
ZONE\_HIGHMEM（动态映射的页，物理内存>896MB）

并且页也需要分配和释放，其中用到的函数有alloc\_page、alloc\_pages、\_get\_free\_pages、\_\_get\_free\_pages、\_\_get\_zeroed\_page。

Linux系统内存管理最重要的部分就是slab分配器，它有着缓存频繁分配和释放数据结构的功能。一般情况下，slab由一个或多个物理地址上连续的页组成。每当进程调用fork时，一定会创建一个新的进程描符，这是在dup\_task\_struct()进程中完成，而该函数会被do\_fork()调用.执行完毕后，如果没有子进程在等待的话，它的进程描述符就会被释放。返回给task\_struct\_cachep slab高速缓存。

（3）VFS（Virtual File System），虚拟文件系统。Linux内核将不同功能的外部设备，例如Disk设备（硬盘、磁盘、NAND Flash、Nor Flash等）、输入输出设备、显示设备等等，抽象为可以通过统一的文件操作接口（open、close、read、write等）来访问。

（4）Network，网络子系统。负责管理系统的网络设备，并实现多种多样的网络标准。

（5）IPC（Inter-Process Communication），进程间通信。IPC不管理任何的硬件，它主要负责Linux系统中进程之间的通信。

（3）（4）（5）三部分由于在课程部分还未提及，我只简略了解了这几部分的功能以及主要的函数，故在此不详细列出。

总结：Linux内核中有许多代码，上述列出的是Linux内核中最重要的几个部分，其余部分非核心代码或者编译脚本，配置文件等没有列出。

# 四、Linux内核内存管理相关源码的研读

**1.系统启动和保护模式的进入**

计算机的启动要首先从BIOS说起。在计算机通电后产生一个脉冲给CPU开始工作当cpu收到该复位脉冲后，其硬件逻辑就会置CS寄存器为FFFFH，IP寄存器为0000H，也就是说，会自动地到FFFF:0000H去取第一条指令。

并且BIOS程序在内存最开始的位置（0x00000）用1KB的内存空间（0x00000～0x003FF）构建中断向量表，在紧挨着它的位置用256字节的内存空间构建BIOS数据区（0x00400～0x004FF），并在大约57 KB以后的位置（0x0E05B）加载了8 KB左右的与中断向量表相应的若干中断服务程序。这些是第一步内容，在Linux中后续将在bootsect指挥下开始启动和加载。

2.bootsect.s研读归纳

### （1）首先BIOS在启动自减后，将0号磁头对应的盘面0磁道的1扇区加载到0x07c00。然后bootsect接管cpu，复制bootsect到 0x90000位置，并跳转到这个位置开始执行。

### 其中：jmpi go,INITSEG 是段间跳转，INITSEG 是段地址，go 是偏移地址。 （2）加载setup到0x90200

1. 获得磁盘驱动器参数
2. 加载 system 到 0x10000
3. 确认根文件系统设备号
4. 跳转到 setup 去执行

总结：这六条是bootsec.s这部分代码干的基本工作。这其实是一个引导扇区，主要作用就是帮助后续的加载和启动。这部分代码编译后仅有512字节，可谓是干练简洁。我在此列出的是基本功能解释，而具体的每一个函数，每一个地址的作用我并未一一列出，因为对于整体逻辑来说这部分不是重点。

#### 3. setup.s研读归纳

（1）获取参数保存在0x90000

这些参数包括保存光标位置，获取从1M处开始的扩展内存大小，获取显示模式，检查显示方式并获取参数，赋值硬盘参数表。

（2）关中断

关中断

! 为进入保护模式做准备

  cli         ! no interrupts allowed !

（3）移动 SYSTEM 模块到 0X00000

bootsect.s 引导程序将 system 模块读入到 0xl0000 开始的位置。由于当时假设 system 模块最大长度不会超过 0x80000 (512KB)，即其末端不会超过内存地址 0x90000，所以 bootsect.s 会把自己移动到0x90000 开始的地方，并把 setup 加载到它的后面。下面这段程序的用途是再把整个 system 模块移动到 0x00000 位置，即把从 0x10000 到 0x8ffff 的内存数据块(共512KB)整块地向内存低端移动了0x10000(64KB)。

! 从代码实现来看，是一小块(0x10000B=64KB)一小块移动的，共移动8小块。

    mov ax,#0x0000

    cld             ! 'direction'=0, movs moves forward

do\_move:

    mov es,ax       ! es是目的段地址

    add ax,#0x1000

    cmp ax,#0x9000   ! 当 ax==0x9000 时结束移动

    jz  end\_move

    mov ds,ax       ! ds是源段地址，ds比es大0x1000

    sub di,di        ! di = 0

    sub si,si       ！ si = 0

    mov cx,#0x8000  ! 重复 0x8000次

    rep             ! ds:si --> es:di

    movsw           ! 每次移动2B.

    jmp do\_move    ！ 本轮一共移动 0x8000\*2B = 0x10000B=64KB. 准备下一轮移动

end\_move:

（4）设置8259

（5）进入保护模式

首先加载机器状态字（lmsw，Load Machine Status Word），也称控制寄存器 CR0，其比特位 0 置 1 将使 CPU 切换到保护模式，并且运行在特权级0，即当前特权级 CPL = 0。此时各个段寄存器仍然指向与实地址模式中相同的线性地址处（在实地址模式下线性地址与物理地址相同）。在设置该比特位后，随后一条指令必须是一条段间跳转指令，用于刷新 CPU 当前指令队列。因为 CPU 是在执行一条指令之前就已从内存读取该指令并对其进行译码。然而在进入保护模式以后那些属于实模式的预先取得的指令信息就变得不再有效。而一条段间跳转指令就会刷新 CPU 的当前指令队列，即丢弃这些无效信息。

    mov ax,#0x0001  ! protected mode (PE) bit

    lmsw    ax      ! This is it! LMSW:Load Machine State Word 加载机器状态字，将PE位置1，开启保护模式

    jmpi    0,8     ! jmp offset 0 of segment 8 (cs)， 这里是段选择符

总结：setup.s是启动的初始阶段，包括了基础的初始化，加载基本模块和从实模式进入保护模式，是Linux启动中不可获取的一步。

#### 4.head.s研读归纳

head的加载方式是：先将head.s汇编成目标代码，将C语言编写的内核程序编译成目标代码，然后链接成system模块。所以system模块既有内核程序，也有head程序。

setup将system模块复制到0x00000位置，因为head在system模块的前面部分，所以head程序就在0x00000这个位置。head程序占有25KB+184B的空间。head程序后面就是main函数。

head的工作：用程序自身的代码在程序自身所在的内存创建内核分页机制。即在0x00000创建了页目录表、页表、缓冲区、GDT、IDT。并覆盖已执行的代码。这意味着head将自己废弃，即将执行main函数。

**2. 内存管理及内存操作相关代码的研读**

1、 内存管理概述

Linux内存管理是内核最复杂的任务之一，主要是因为它用到许多CPU提供的功能，而且和这些功能密切相关。Linux内存管理有各种机智，如内存的初始化机制，地址映射机制，企业机制交换机制，内存分配和回收机制，缓存和刷新机制以及内存共享机制。而且Linux内存管理设计充分利用了计算机系统的虚拟存储技术，实现了虚拟存储器管理。实验中使用的Linux是Intel 386，具有段机制和页机制。

2、 虚拟内核空间到物理空间的映射

内核空间中存放的是内核代码和数据和进程的用户控件中存放的是用户程序的代码和数据。不管是内核空间还是用户空间，他们都处于虚拟内存中。

Include/asm-i386/page.h中有对内核空间中地址映射的说明：

如果物理内存大于950mb，那么在编译内核设计需要加两个选项。如果物理内存小于950mb，则对于内核空间而言给定一个虚地址x，其物理地址为“x-PAGE\_OFFSET”,给定一个物理地址x其虚地址为“x+PAGE\_OFFSET”。

3、页表初始化

由代码可以看出，页机制的零页，即empty\_zero\_page页，存放的是系统的启动参数和命令行参数。

4、 物理页面的数据结构

对物理页面的描述在/include/linux/mm\_types.h

源码的注释中对这个数据结构给出了一定的说明。当页面的数据来自一个文件，index代表该页面中的数据在文件中的偏移量。系统中的每个物理页都有一个page结构。系统在初始化阶段根据内存的大小建立起一个page结构的数组mem\_map。数组的下标就是内存中物理页面的序号。

在struct page描述一个页框时，我们比较关注的成员变量有unsigned long flags、struct list\_head lru和atomic\_t \_count。

flags：包含有很多信息，包括此页框属于的node结点号，此页框属于的zone号和此页框的属性。

lru：用于将此页描述符放入相应的链表，比如伙伴系统或者每CPU页框高速缓存。

\_count：代表页框的引用计数，-1代表此页框空闲，大于0代表此页框分配给了多少个进程使用(共享)。

即flags存放页的状态，如该页是不是脏页。 \_count域表示该页的使用计数，如果该页未被使用，就可以在新的分配中使用它。

5、 物理页面的数据结构以及获得页

为了对页面管理机制做出初步准备，Linux使用一种叫bootmem分配器的机制。这种机制仅仅用在系统引导时，为整个物理内存建立一个页面位图。建立这个位图的目的就是弄清楚哪一些物理页面是可以动态分配的，用来存放位图的数据结构为boot\_data。

在路径/linux-2.6.21/include/asm-i386/page.h中，规定了页面大小：

其中PAGE\_SHIFT指定了偏移量offset字段的位数为12，因此表示页面大小为4096个字节。（2^12=4096）

PAGE\_SIZE为页的大小。

pte\_t表示页表项，pmd\_t表示页上级目录项，pgd\_t表示页全局目录项，pgprot\_t表示与一个单独表项相关的保护标志。

获得页使用的接口是alloc\_pages函数，源码位于linux/gfp.h中。

该函数返回值是指向page结构体的指针，参数gfp\_mask是一个标志，简单来讲就是获得页所使用的行为方式。order参数规定分配多少页面，该函数分配2的order次方个连续的物理页面。返回的指针指向的是第一page页面。

获得页的函数最终调用的都是alloc\_pages函数。alloc\_pages函数是获得页的核心函数。

6、 释放页

当我们不再需要某些页时可以使用下面的函数释放它们：

\_\_free\_pages（struct page \*page, unsigned int order）

\_\_free\_page

free\_pages

free\_page（unsigned long addr, unsigned int order）

这些接口都在linux/gfp.h中

释放页的时候一定要小心谨慎，内核中操作不同于在用户态，若是将地址写错，或是顺序写错，那么都可能会导致系统的崩溃。

7、总结

在这里我主要是分析了Linux内核源码物理页面内存管理的部分源码。对于物理内存来说，系统都是以页框作为最小的分配单位，而分配时必定是要通过管理区分配器进行分配的，在管理区分配器中又必定是通过伙伴系统或每CPU页框分配器进行分配的。

**3. 页面淘汰相关算法**

在linux中所使用的的页面淘汰算法并不是单一的，在不同情况下使用到的算法是不同的。我现在将介绍一些使用较多的页面淘汰相关算法。算法流程如下介绍：

#### 1. 最佳置换算法OPT

1）基本思想

选择永不使用或是在最长时间内不再被访问（即距现在最长时间才会被访问）的页面淘汰出内存。

2）评价

理想化算法，具有最好性能（对于固定分配页面方式，本法可保证获得最低的缺页率），但实际上却难于实现，故主要用于算法评价参照。

#### 2. 先进先出置换算法FIFO

1）基本思想

选择最先进入内存即在内存驻留时间最久的页面换出到外存；

进程已调入内存的页面按进入先后次序链接成一个队列，并设置替换指针以指向最老页面。

2）评价

简单直观，但不符合进程实际运行规律，性能较差，故实际应用极少。

#### 3. 最近最久未使用置换算法LRU

1）基本思想

以“最近的过去”作为“最近的将来”的近似，选择最近一段时间最长时间未被访问的页面淘汰出内存。

2）评价

适用于各种类型的程序，性能较好，但需要较多的硬件支持。

#### 4. 简单Clock置换算法

1）基本思想

当某一页首次装入内存中时，则将该页框的使用位设置为1；当该页随后被访问到时（在访问产生缺页中断之后），它的使用位也会被设置为1。对于页面置换算法，用于置换算法，用于置换的候选页框集合（当前进程：局部范围；整个内存；全局范围）被看做是一个循环缓冲区，并且有一个指针与之相关联。当一页被置换时，该指针被设置成指向缓冲区中的下一页框。当需要置换一页时，操作系统扫描缓冲区，以查找使用位被置为0的一页框。每当遇到一个使用位为1的页框时，操作系统就将该位重新置为0；如果在这个过程开始时，缓冲区中所有页框的使用位均为0时，则选择遇到的第一个页框置换；如果所有页框的使用位均为1时，则指针在缓冲区中完整地循环一周，把所有使用位都置为0，并且停留在最初的位置上，置换该页框中的页。

2）评价

与上面的算法相比，占用系统资源数少。

#### 5. 改进型Clock置换算法

1）基本思想

① 从查寻指针当前位置起扫描内存分页循环队列，选择A=0且M=0的第一个页面淘汰；若未找到，转②

② 开始第二轮扫描，选择A=0且M=1的第一个页面淘汰，同时将经过的所有页面访问位置0；若不能找到，转①

2）评价

与简单Clock算法相比，可减少磁盘的I/O操作次数，但淘汰页的选择可能经历多次扫描，故实现算法自身的开销增大。

#### 6. 页面访问序列随机发生机制

（1）初始化进程逻辑地址空间页面总数 N、各逻辑页面的读写访问方式（是否支持写访问，即 R、RW）、工作集起始页号 s（s∈[0, N)）、工作集中包含的页数 w，工作集移动速率 v（每处理 v 个页面访问，就将工作集起始页号递增即 s+1）以及一个取值区间为[0, 1]的值 t；

（2）生成取值区间为[s, min(s+w, N-1)]的 v 个随机数并添加保存到页面访问序列中，同时为每次页面访问分别生成一个取值区间为[0, 1]的随机数，若该随机数值大于 0.7 且对应所访问页面支持写访问则设定以写方式访问相应页面，否则以读方式访问对应页面；

（3）生成取值区间为[0, 1]的一个随机数 r，并比较 r 与 t 的大小；

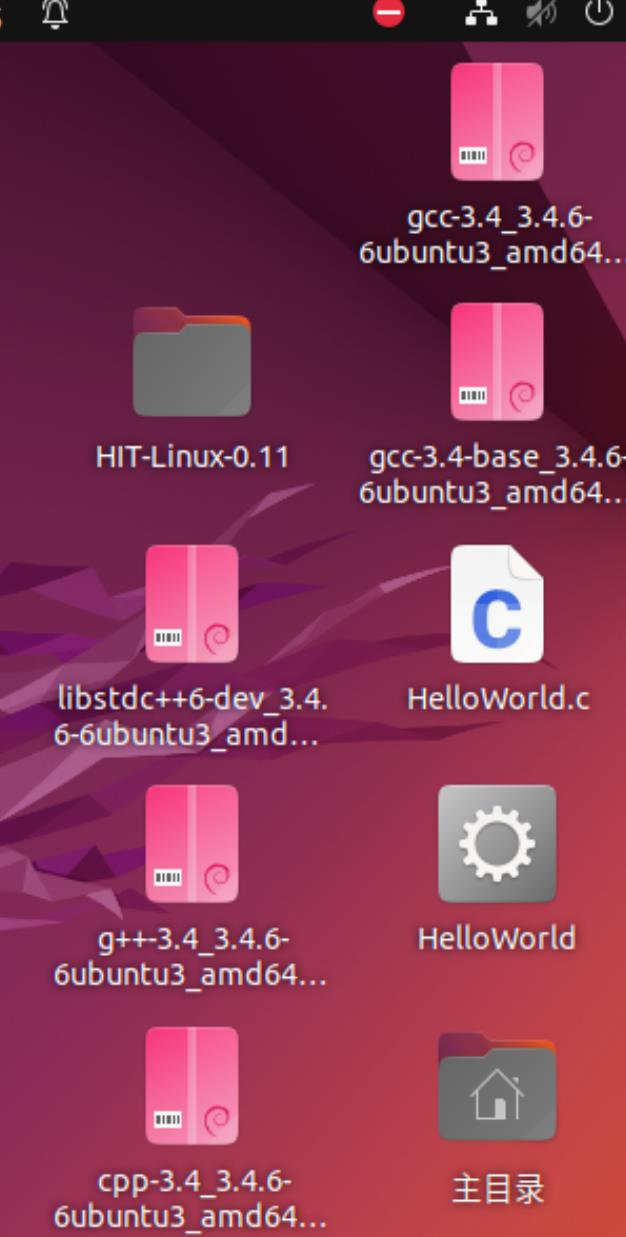
（4）若 r < t，则为 s 生成一个新值（s∈[0, N)），否则 s = (s + 1) mod N；

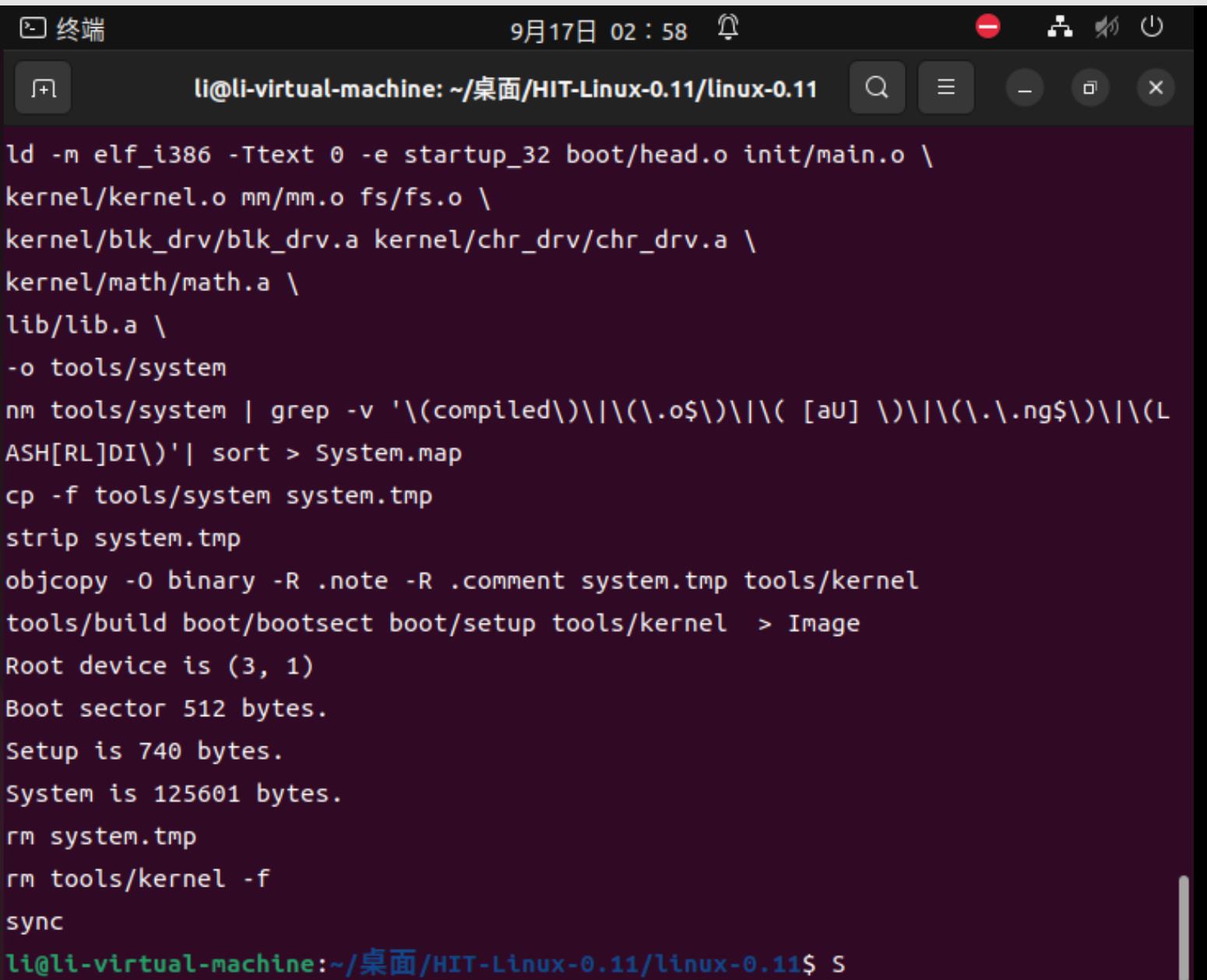
（5）如果想继续加大页面访问序列的长度，返回第 2 步，否则结束。

# 五、实验测试及现象

1. 编译

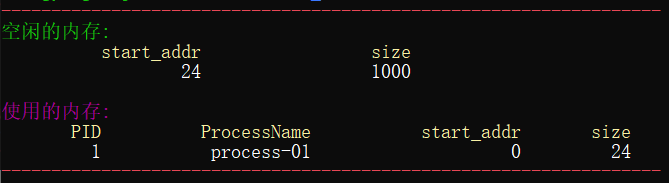
由于linux-0.11版本过于低，导致目前虚拟机上使用的并不是32位环境，首先需要建立32位环境。然后安装汇编工具，安装gcc3.4，这样就可以吧基础的环境配置完成。最后效果如图：

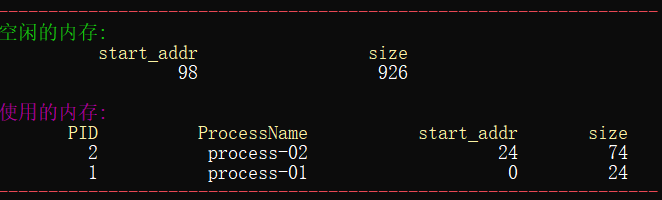


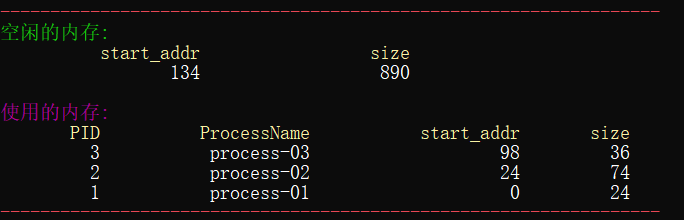
2.在linux-0.11下直接make，结果如图：

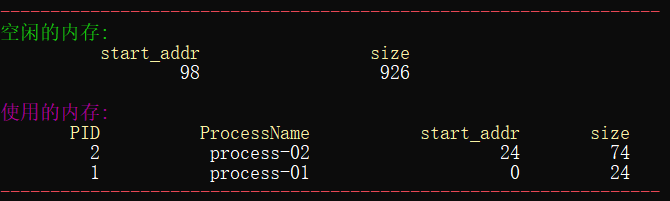
可以看出，我们已经将linux-0.11编译成功。

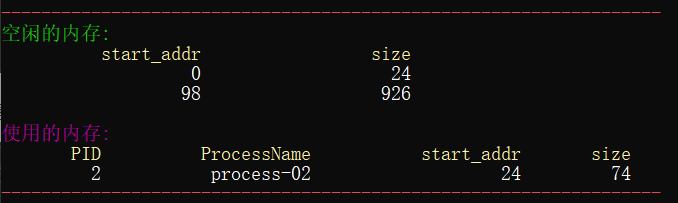
对于内存管理部分编写代码进行测试，结果如下：





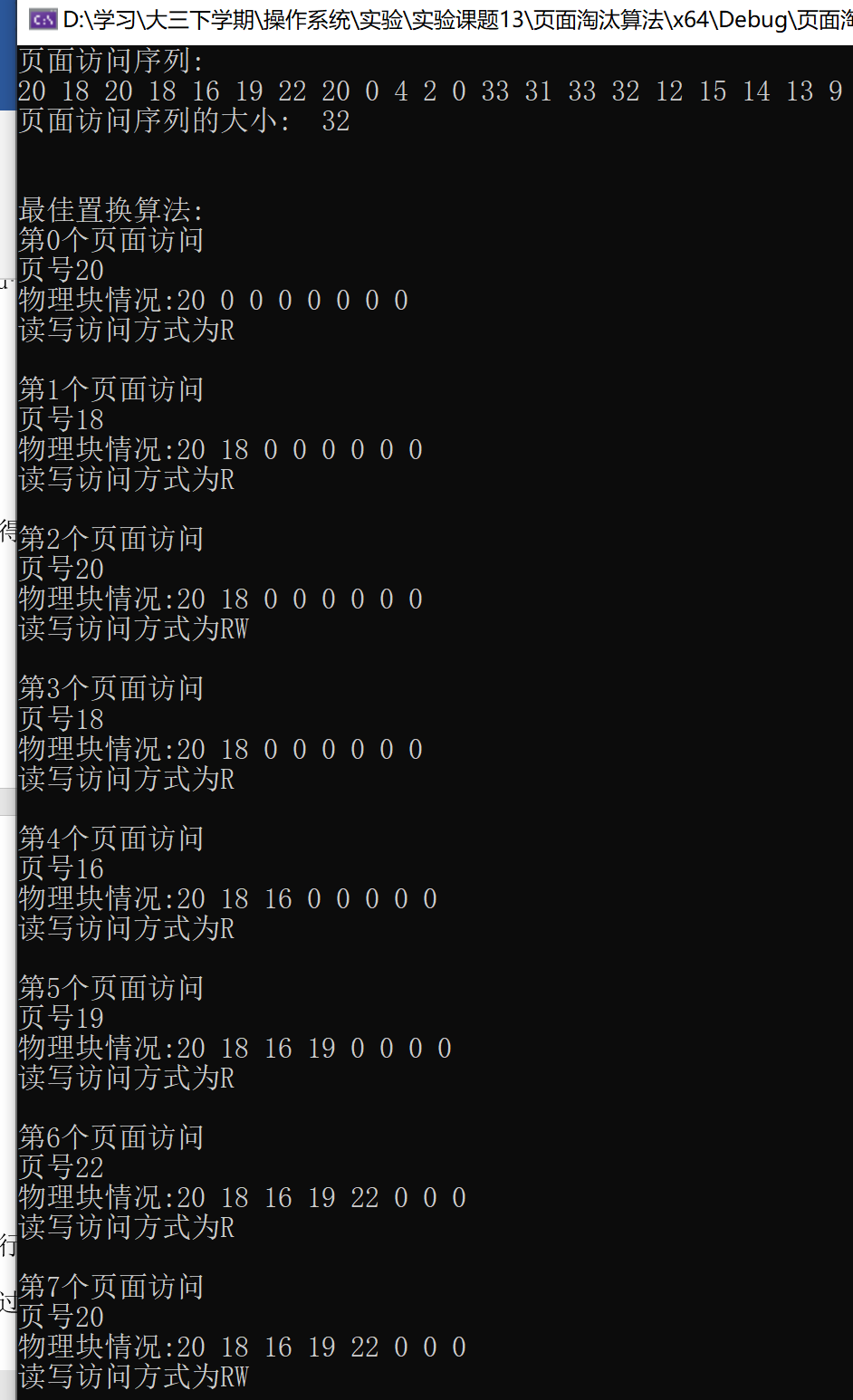


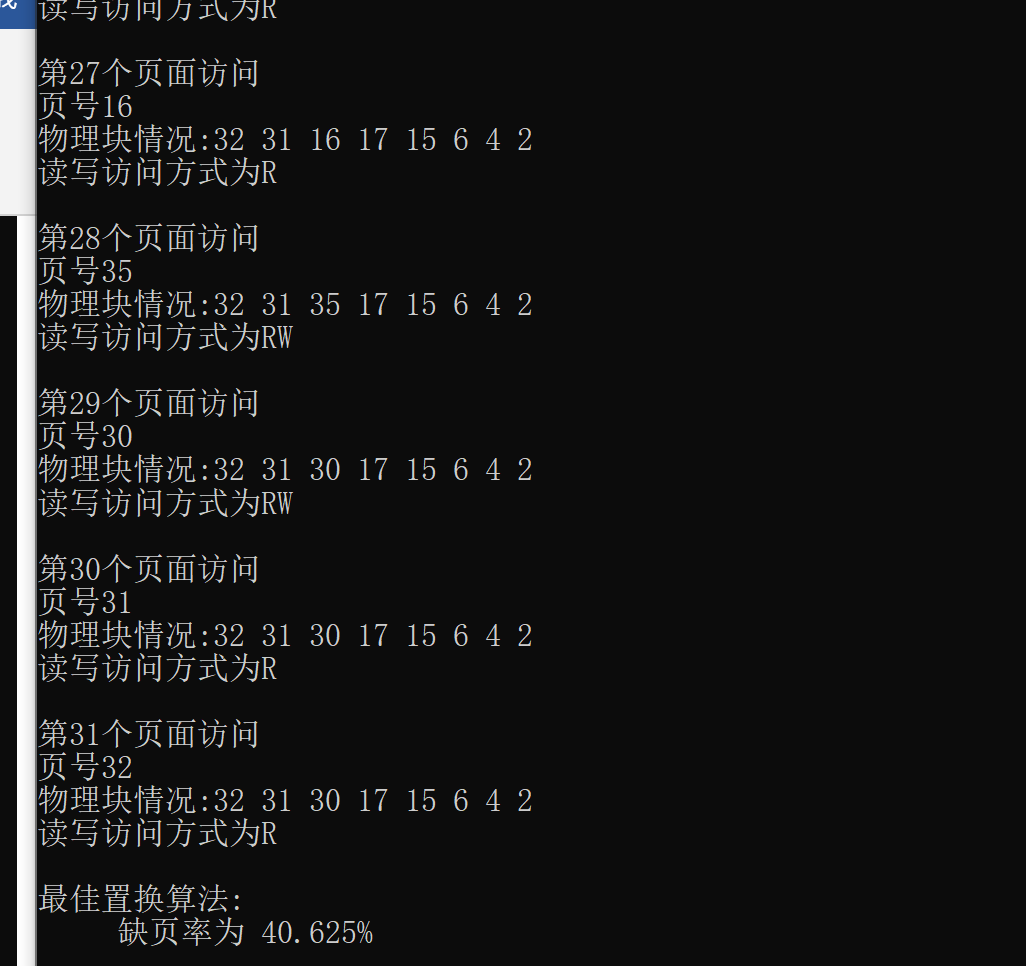


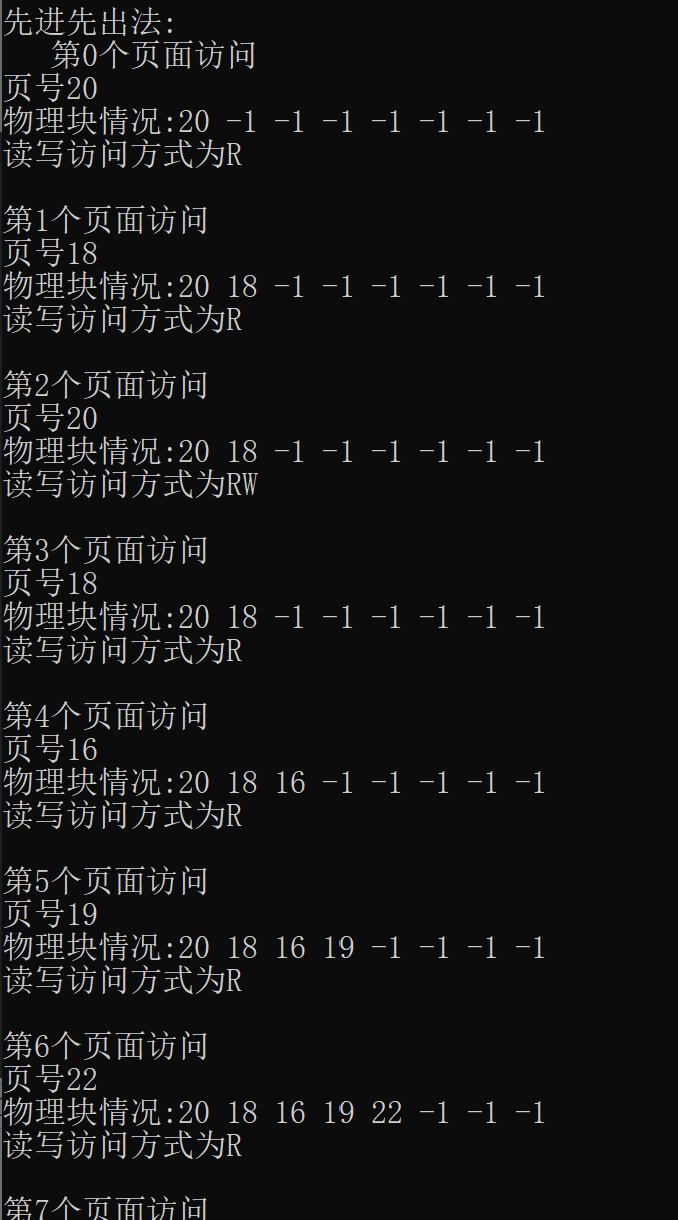


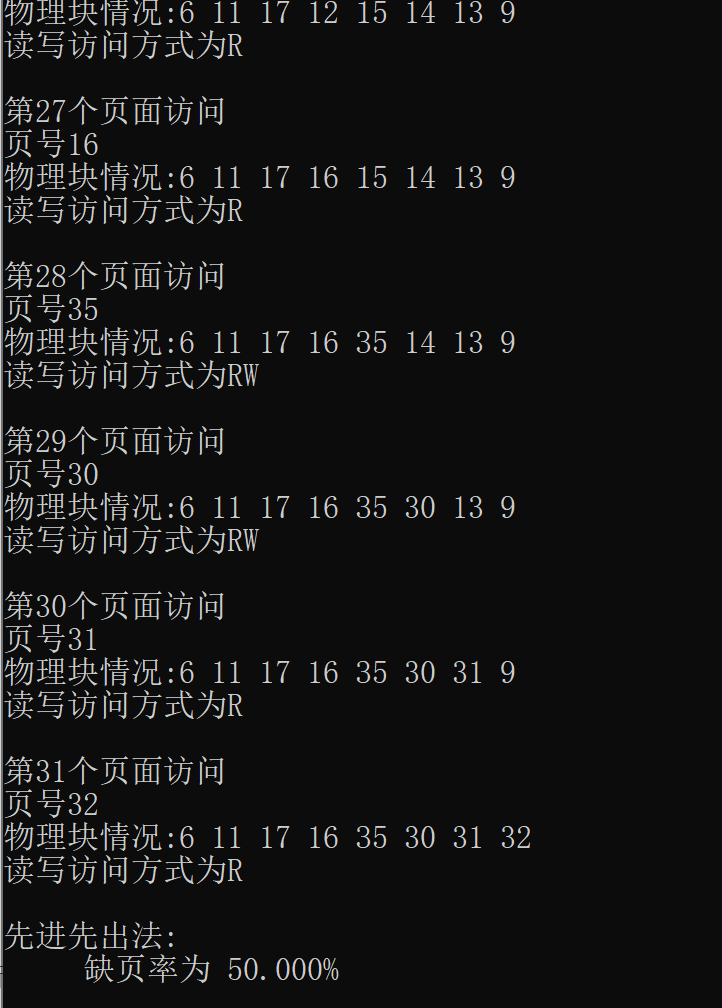
对页面淘汰算法编写代码进行测试：

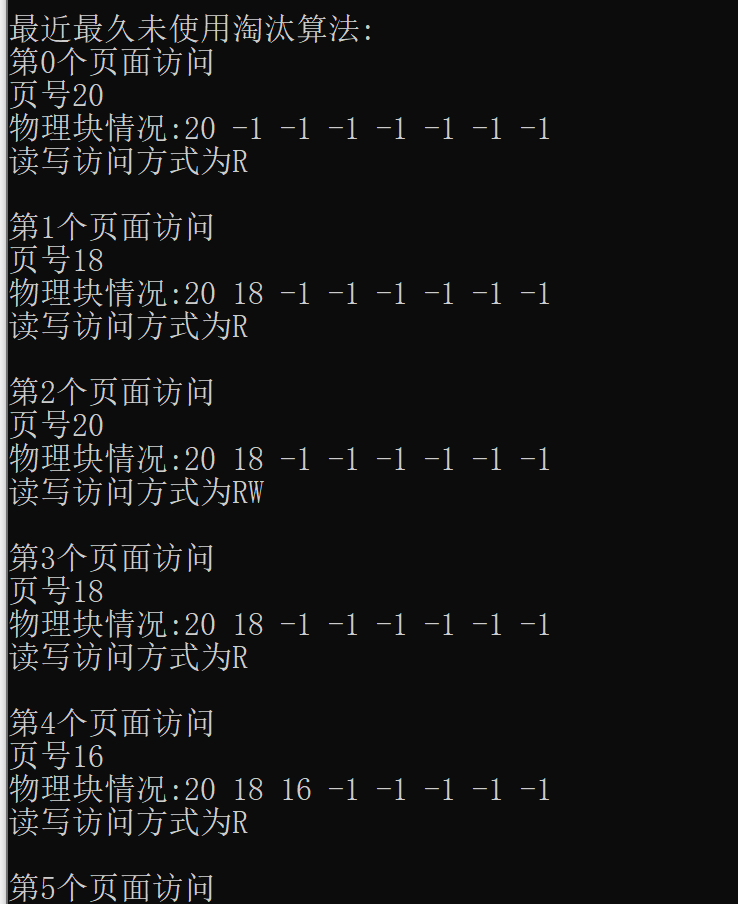
五个算法的部分测试结果如下图所示：

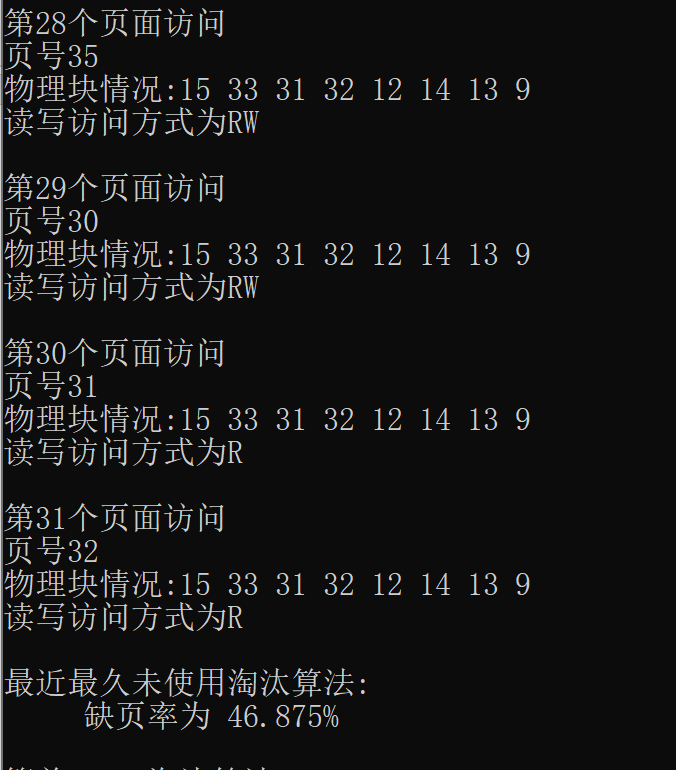


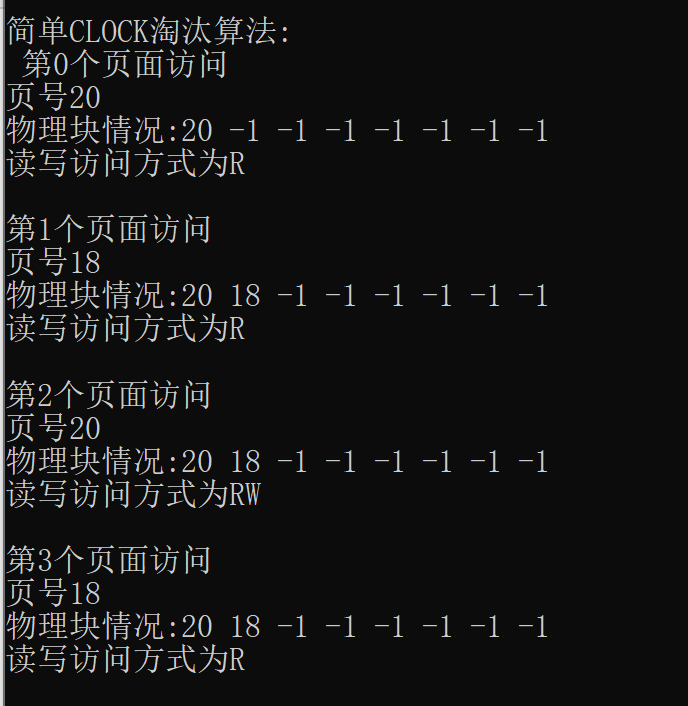


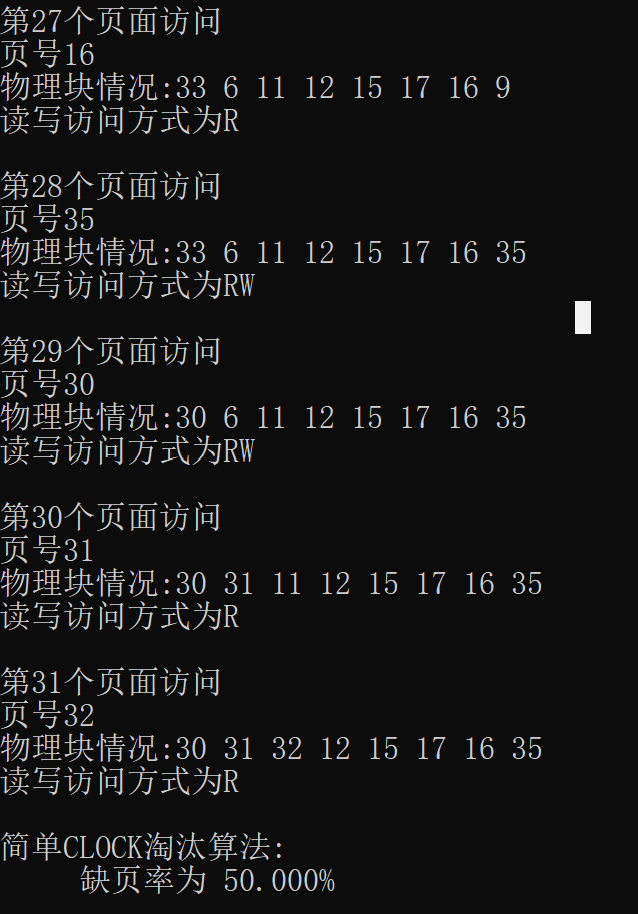


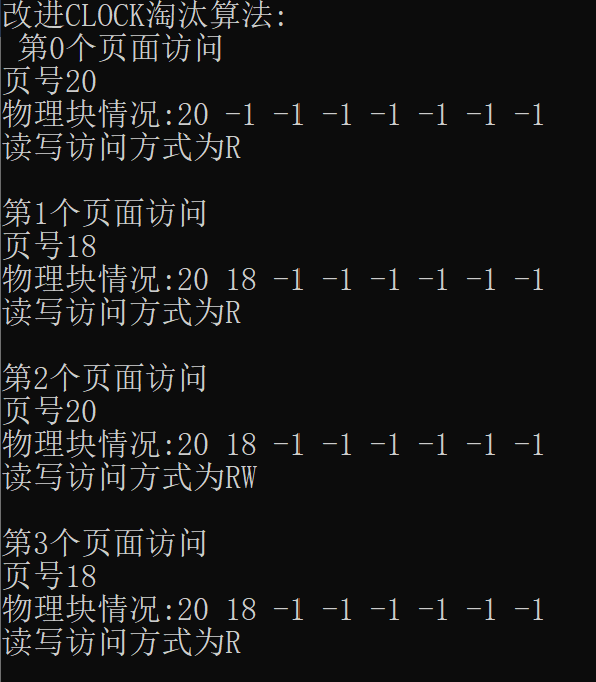


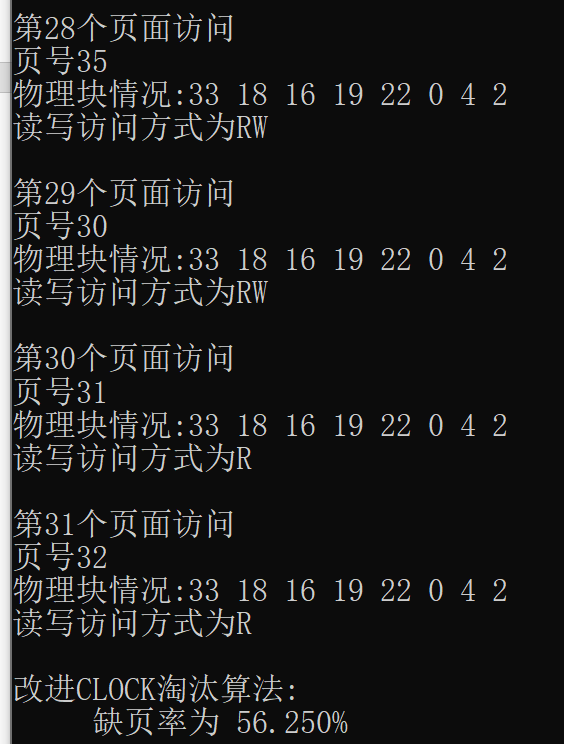












通过不断改变页面数和物理块数进行测试得到的结果如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **物理块大小** | OPT | FIFO | LRU | CLOCK | LCLOCK |
| 3 | 61.80% | 81.27% | 81.27% | 81.07% | 80.93% |
| 4 | 52.60% | 73.60% | 74.13% | 74.40% | 75.93% |
| 5 | 45.73% | 66.60% | 66.53% | 69.13% | 67.53% |
| 6 | 42.47% | 63.40% | 65.00% | 67.53% | 65.60% |
| 7 | 37.13% | 55.93% | 56.60% | 65.47% | 58.13% |
| 8 | 33.07% | 51.47% | 52.27% | 62.27% | 55.20% |

当物理块数固定为8时，改变页面总数的大小结果如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **页面总数大小** | OPT | FIFO | LRU | CLOCK | LCLOCK |
| 32 | 32.25% | 48.25% | 52.25% | 62.07% | 55.93% |
| 64 | 49.13% | 61.93% | 59.60% | 72.47% | 68.13% |
| 128 | 62.47% | 75.40% | 66.00% | 78.53% | 72.60% |
| 256 | 79.53% | 81.22% | 84.33% | 89.63% | 85.53% |
| 512 | 89.92% | 92.13% | 91.29% | 92.33% | 92.93% |
| 1024 | 91.73% | 98.90% | 95.99% | 94.90% | 94.93% |

# 六、实验总结及体会

通过这次实验，我更加详细的研读了linux关于自启动和内存管理部分的源码，并手动实现了部分linux源码并进行了测试。在这次实验中，我学习到了很多新的知识和技能，收获颇多。