# 操作系统实验报告

指导老师：李治军

学生姓名：李扬名

学生学号：1163200511

学生班级：1603102

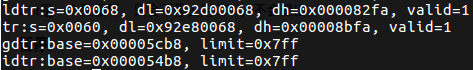
## 实验6：地址映射与共享

1. 对于地址映射实验部分，列出你认为最重要的那几步，并给出获取的实验数据。

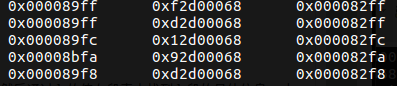
在 oslab下命令 ./dbg-asm进入debug模式，然后输入c启动linux-0.11，再在bochs 中命令 ./test （事先需要 gcc -o test test.c）执行，然后会打印出：



现在需要寻找 ds:0x3004 的物理地址。首先要找到 LDT的位置，命令 sreg有：



可以看到gdtr的物理地址在 0x00005cb8，ldt 存在GDT表的13号位置（可由段选择子ldtr的值0x0068推算出）。然后命令 xp /32w0x00005cb8，打印得到：

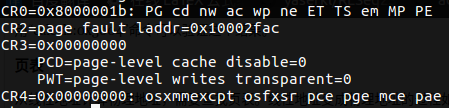


按照8字节一个描述符可得到ldt的描述符0x92d00068 0x000082fa，按照描述符的结构组合可得到LDT的物理基址0x00fa92d0。在之前命令sreg中可得到ds的值为0x0017，按选择子的结构可知TI=1（使用LDT表）、索引为2。命令 xp /8w得到：

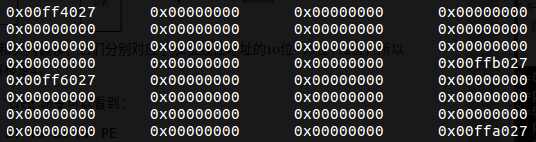


按照段描述符（8个字节）的结构可得0x00003fff 0x10c0f300 -> 0x10000000，然后偏移地址0x3004结合可得0x10003004，可用命令 calc ds:0x3004验证。

从线性地址计算物理地址，需要对线性地址进行解剖。按10+10+12去切0x10003004可得页目录号64、页号3、页内偏移是4。用命令creg可查看系统寄存器：



CR4指定了页目录基址是0，用命令 xp /68w 0 查看物理内存可打印：



页表的结构是1024个32位数（即4k），可得0x00fa5027，去掉属性027则页表所在的物理页框号位0x00fa5，然后命令xp /w 0x00fa5000+3\*4继续查第3号页表项：



去掉属性，与页内偏址0x004结合可得到0x00fa4004。命令xp /w 0x00fa4004打印：



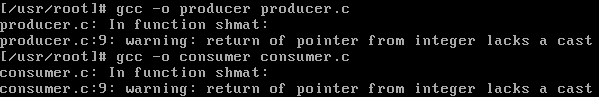
另外，也可以通过命令 page 0x10003004（之前得到的线性地址）验证结果。

1. test.c 退出后，如果马上再运行一次，并再进行地址跟踪，找到异同点及其原因。

虚拟地址、线性地址没有变化，但实际的物理地址改变了。因为前两者都是操作系统抽象出来的，即使再运行一次也在相同的位置，但后者是动态分配的，例如上一次占用的物理地址被别的进程占用了，因此操作系统需要使用一块新的内存。

3，共享内存的演示，只需要表现出两个后台的进程在使用同一块内存即可。

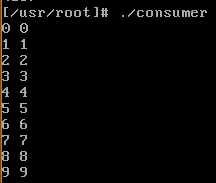
首先命令 gcc -o x x.c 编译 producer.c 和 consumer.c，如下：



然后后台运行producer.c，即：



然后执行 consumer.c，观察输出：



其中左列表示共享数组的下标索引，右列表示数组存储的值。

## 实验7：字符显示的控制

1. 在原始代码中，按下F12，中断响应后，中断程序会调用func？它实现了什么功能？

按下F1-F12中任何功能键都会显示当前进程的信息。它是通过show\_stat函数完成的。

1. 在你的实现中，是否把向文件输出的字符也过滤了？如果是，那么怎么能只过滤终端输出的字符？如果不是，那么怎么能把向文件输出的字符也一并过滤掉？

没有，因为过滤终端的字符是通过修改con\_write实现的，而过滤向文件中输出的字符则需要修改fs/read\_write.c文件。需要在其中函数sys\_write中加入如下代码：

if (f12\_state == 1) {

int i;

for (i = 0; i < count; i++) {

char ch = get\_fs\_byte(buf + i);

if (ch >= 'A' && ch <= 'Z' || ch >= 'a' && ch <= 'z') {

put\_fs\_byte('\*', buf + i);

}

}

}

他的逻辑是：判断f12\_state的状态，为1时表示用户按下F12键。然后循环读入缓冲区的字符并将其中的大小写字母都换成“\*”的形式重新写入缓存。

实验效果如下，首先验证正常的情况：



然后按下F12键并松开观察现象如下：



## 实验8：proc文件系统的实现

1. 如果要求你在psinfo之外再实现另一个结点，具体内容自选，那么你会实现一个给出什么信息的结点？为什么？

实现一个显示内存使用情况的信息，meminfo。因为内存是操作系统中非常重要的资源，实时查看各进程使用内存的情况便于程序员管理。

2，一次read()未必能读出所有的数据，需要继续read()，直到把数据读空为止。而数次read()之间，进程的状态可能会发生变化。你认为后几次read()传给用户的数据，应该是变化后的，还是变化前的？

1，如果是变化后的，那么用户得到的数据衔接部分是否会有混乱？如何防止混乱？

2，如果是变化前的，那么该在什么样的情况下更新psinfo的内容？

3，删除文件以后，/proc/inodeinfo那个inode号的inode，你发现了什么，为什么？

是变化前的，因为再读取f\_pos=0时才更新psinfo的内容，此inode对应的i\_zone[0]仍然存在。换句话说，只是从inode映射中取消了映射inode，但硬盘的数据还在。