**电 子 科 技 大 学**

**实 验 报 告**

**实验二**

1. **实验室名称：**

电子科技大学清水河校区主楼A2-412

**二、实验项目名称：**

虚拟内存综合实验

**三、实验内容**

通过手工查看系统内存，并修改特定物理内存的值，实现控制程序运行的目的。

**四、实验目的**

通过实验，掌握段页式内存管理机制，理解地址转换的过程。

**五、实验原理**

**5.1 物理地址，线性地址，逻辑地址，虚拟地址**

**物理地址：**

物理地址最好理解，我们可以简单的把内存比作一个大的数组（为了分析方便），每个数组都有其下标，这个下标标识了内存中的地址，这个实实在在的在内存中的地址，我们称之为物理地址。但是在用于内存芯片级的单元寻址，与处理器和CPU连接的地址总线相对应，相信并不是一个所谓的数组，但是做出这样的比拟，有利于更好的理解。

还依稀记得这张图：

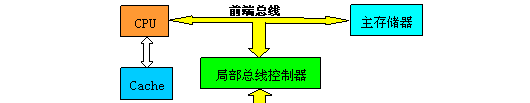


图7 计算机组成局部图

**逻辑地址：**

与物理地址比较相对的是逻辑地址，一种理解是，这个地址就是在程序中我们把它放到的位置，而这个位置通常是由编译器给出的。另外的一种理解是：逻辑地址指的是机器语言指令中，用来指定一个操作数或者是一条指令的地址。Intel段式管理中，一个逻辑地址，是由一个段标识符加上一个指定段内相对地址的偏移量，表示为 [段标识符：段内偏移量]。比如我们在程序中定义一个变量int g=3；相应的汇编代码应该是mov [g],3;那么这个g应该放在哪儿呢？实际上我们可以看到，这个g的地址总在在编译，链接之后就会一个确定的地址;而这个确定的地址我们叫做逻辑地址。

**虚拟地址：**

Virtual Address，简称VA，由于Windows程序时运行在386保护模式下，这样程序访问存储器所使用的逻辑地址称为虚拟地址。实际上因为我们现代程序中地址都是虚拟的，所以这里的虚拟地址和线性地址是等价了的。

**线性地址:**

线性地址（Linear Address）也叫虚拟地址(virtual address)是逻辑地址到物理地址变换之间的中间层。在分段部件中逻辑地址是段中的偏移地址，然后加上基地址就是线性地址。

**5.2 CPU段式内存管理：逻辑地址转换为线性地址**

逻辑地址示意图如图8所示。一个逻辑地址由两部分组成，段标识符: 段内偏移量。段标识符是由一个16位长的字段组成，称为段选择符。其中前13位是一个索引号。后面3位包含一些硬件细节。



图8 逻辑地址示意图

最后两位涉及权限检查。

索引号，或者直接理解成数组下标——那它总要对应一个数组，它应该是指向一个东西的？而这个东西就是“段描述符(segment descriptor)”，段描述符具体地址描述了一个段。这样，很多个段描述符，就组了一个数组，叫“段描述符表”，这样，可以通过段标识符的前13位，直接在段描述符表中找到一个具体的段描述符，这个描述符就描述了一个段，每一个段描述符由8个字节组成，如图9所示。

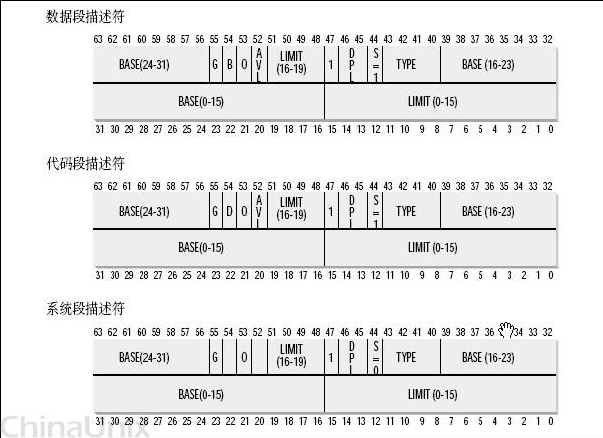


图9 数据段描述符、代码段描述符、系统段描述符示意图

而在汇编里面我们用一个数据结构定义，如图10所示。

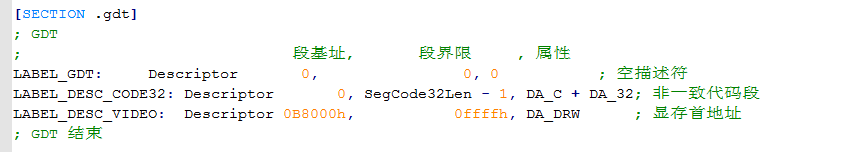


图10 段描述符的数据结构

Base字段，它描述了一个段的开始位置的线性地址。

Intel设计是，一些全局的段描述符，就放在“全局段描述符表(GDT)”中，一些局部的，例如每个进程自己的，就放在所谓的“局部段描述符表(LDT)”中。那究竟什么时候该用GDT，什么时候该用LDT呢？这是由段选择符中的T1字段表示的，=0，表示用GDT，=1表示用LDT。

GDT在内存中的地址和大小存放在CPU的gdtr控制寄存器中，而LDT则在ldtr寄存器中。具体如图11所示。

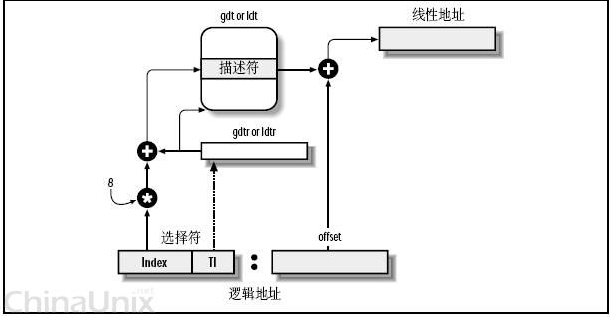


图11 逻辑地址与线性地址的转换示意图

首先，给定一个完整的逻辑地址[段选择符：段内偏移地址]

1. 看段选择符的T1=0还是1，知道当前要转换是GDT中的段，还是LDT中的段，再根据相应寄存器，得到其地址和大小。我们就有了一个数组了。
2. 拿出段选择符中前13位，可以在这个数组中，查找到对应的段描述符，这样，它了Base，即基地址就知道了。
3. 把Base + offset，就是要转换的线性地址了。对于软件来讲，原则上就需要把硬件转换所需的信息准备好，就可以让硬件来完成这个转换了。

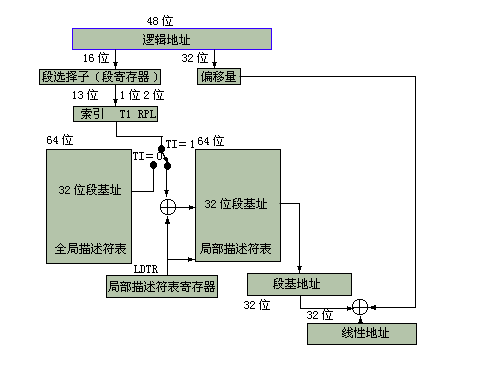


图12 逻辑地址与线性地址的转换完整示意图

但是实际的情况并不是这么简单：linux和windows的做法貌似是不同的。

**5.3 CPU页式内存管理**

　　CPU的页式内存管理单元，负责把一个线性地址，最终翻译为一个物理地址。从管理和效率的角度出发，线性地址被分为以固定长度为单位的组，称为页，例一个32位的机器，线性地址最大可为4G，可以用4KB为一个页来划分，这页，整个线性地址就被划分为一个tatol\_page[2^20]的大数组，共有2的20个次方个页。这个大数组我们称之为页目录。目录中的每一个目录项，就是一个地址——对应的页的地址。

另一类“页”，我们称之为物理页，或者是页框、页桢的。是分页单元把所有的物理内存也划分为固定长度的管理单位，它的长度一般与内存页是一一对应的。这里注意到，这个total\_page数组有2^20个成员，每个成员是一个地址（32位机，一个地址也就是4字节），那么要单单要表示这么一个数组，就要占去4MB的内存空间。为了节省空间，引入了一个二级管理模式的机器来组织分页单元。

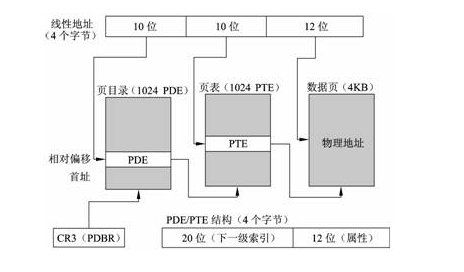


图13 页式内存管理示意图

如图13所示。描述：

1. cr0最高位确定是否采用分页机制，在分页单元中，页目录是唯一的，它的地址放在CPU的cr3寄存器中，是进行地址转换的开始点；
2. 每一个活动的进程，因为都有其独立的对应的虚拟内存（页目录也是唯一的），那么它也对应了一个独立的页目录地址。——运行一个进程，需要将它的页目录地址放到cr3寄存器中，将别个的保存下来；
3. 每一个32位的线性地址被划分为三部份，面目录索引(10位)：页表索引(10位)：偏移(12位)。

转换步骤：

1. 从cr3中取出进程的页目录地址（操作系统负责在调度进程的时候，把这个地址装入对应寄存器）；
2. 根据线性地址前十位，在数组中，找到对应的索引项，因为引入了二级管理模式，页目录中的项，不再是页的地址，而是一个页表的地址。（又引入了一个数组），页的地址被放到页表中去了；
3. 根据线性地址的中间十位，在页表（也是数组）中找到页的起始地址；
4. 将页的起始地址与线性地址中最后12位相加，得到最终我们想要的物理地址。

**5.4 常用命令**

* c，启动linux；
* Ctrl + c，切换控制台；
* sreg，查看段寄存器值；
* creg，查看控制寄存器值；
* xp /2W 0xabcd，显示地址abcd之后的2个字的内容；
* setpmem 0xabcd 4 0，将地址abcd后4字节改为0。

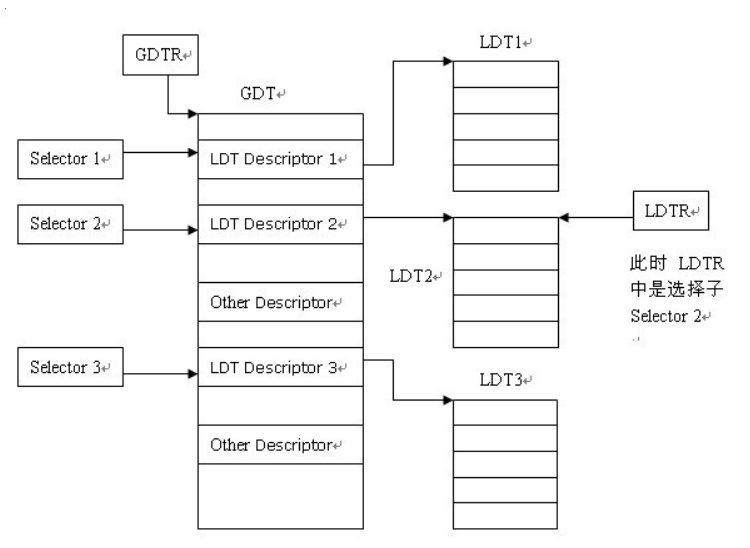


图14 GDTR、GDT、LDTR、LDT示意图

**六、实验器材（设备、元器件）**

处理器：Intel® Core™ i5-8300H CPU @ 2.30GHz 2.30GHz

已安装的内存(RAM)：8GB

系统类型：64位操作系统，基于x64的处理器

Linux内核版本：0.11

Bochs虚拟机版本：Bochs-2.5.1

**七、实验步骤**

1. 运行Bochs-2.5.1.exe，安装Bochs虚拟机，默认安装路径为C:\Program Files (x86)\Bochs-2.5.1；
2. 安装完毕后，复制如图15所示实验所需文件至Bochs根目录下；

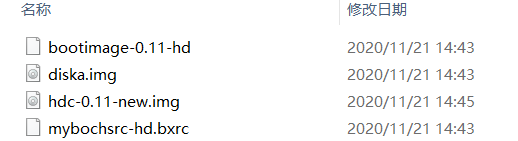


图15 实验所需文件

1. 如图16所示，运行bochsdbg.exe，点击Load加载配置文件，选择mybochssrc-hd.bxrc文件，再点击Start启动Bochs虚拟机；

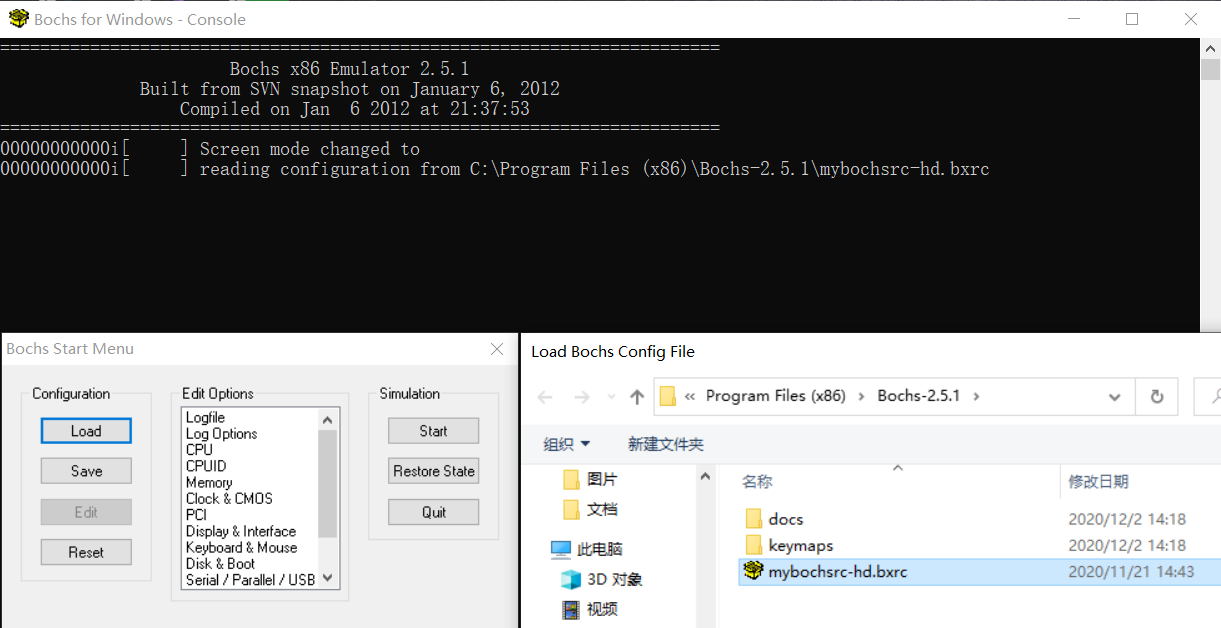


图16 加载实验配置文件并启动Bochs虚拟机

1. Bochs虚拟机启动后，会出现两个窗口：Console窗口和Display窗口，如图17所示。Console窗口为Bochs命令输入窗口，Display窗口为Linux操作系统窗口；

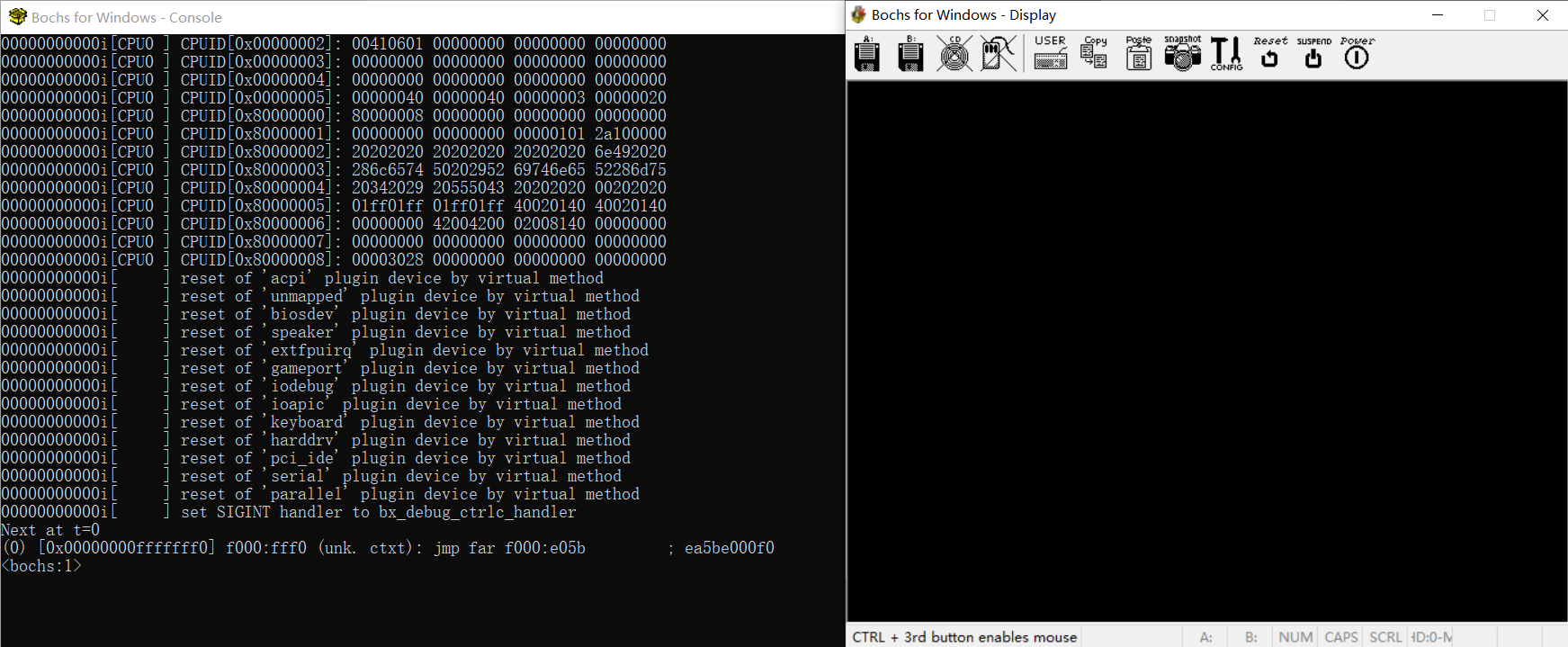


图17 Console窗口和Display窗口

1. 在Console窗口输入命令：c，加载Linux操作系统；

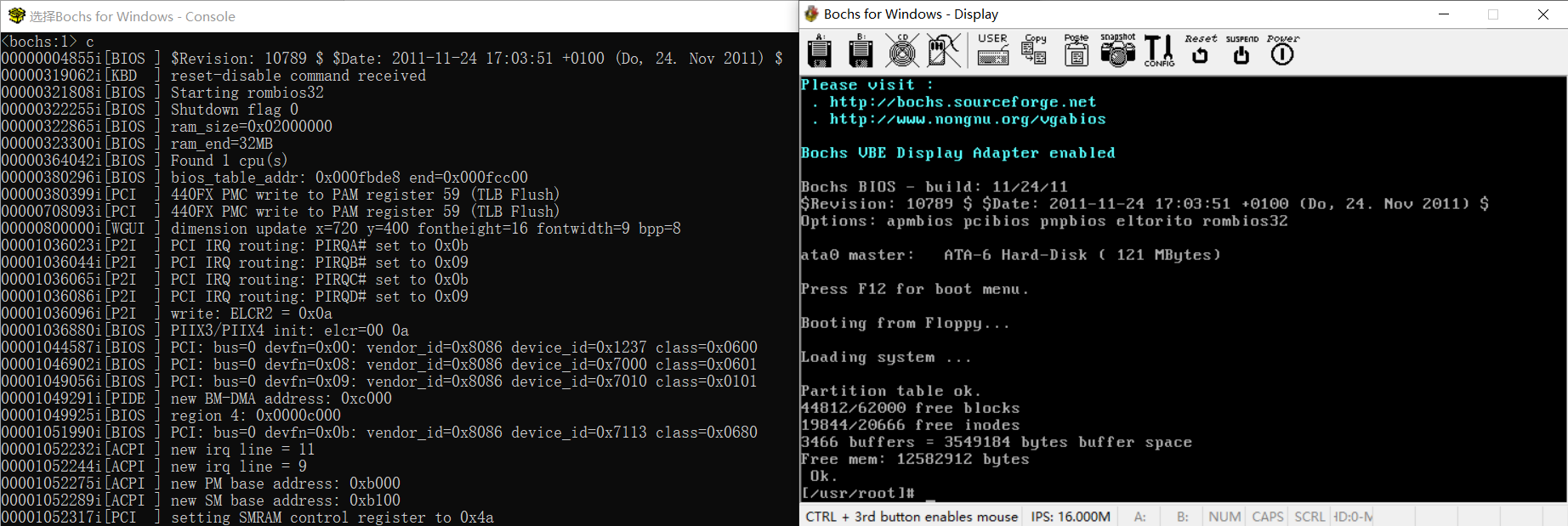


图18 加载Linux操作系统

1. 在Display窗口输入命令：vi test.c，输入实验程序，输入命令：:wq，保存程序并退出；

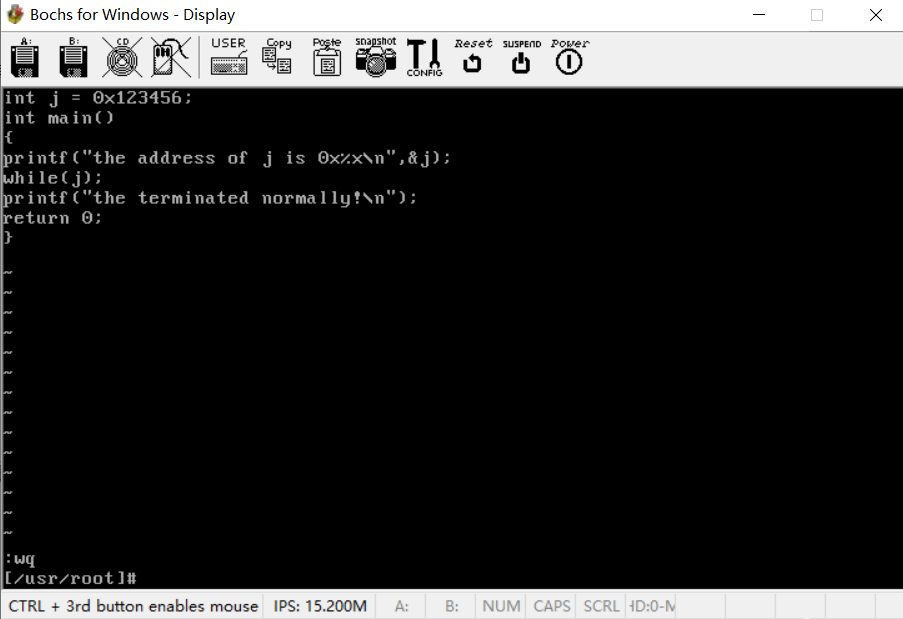


图19 在Display窗口编写test.c程序并保存退出

1. 输入命令：gcc -o test test.c，编译test.c程序，再输入命令：./test，运行程序；

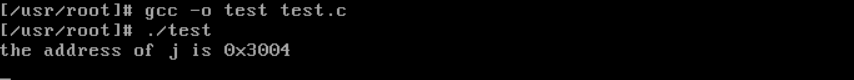


图20 test.c程序运行结果

可以看出，程序运行至while(j)语句时，进入了死循环，所以没有继续运行后面的语句；

1. 在Console窗口按下Ctrl+C，中断当前程序运行，在Console窗口输入命令：sreg，查看段的具体信息；

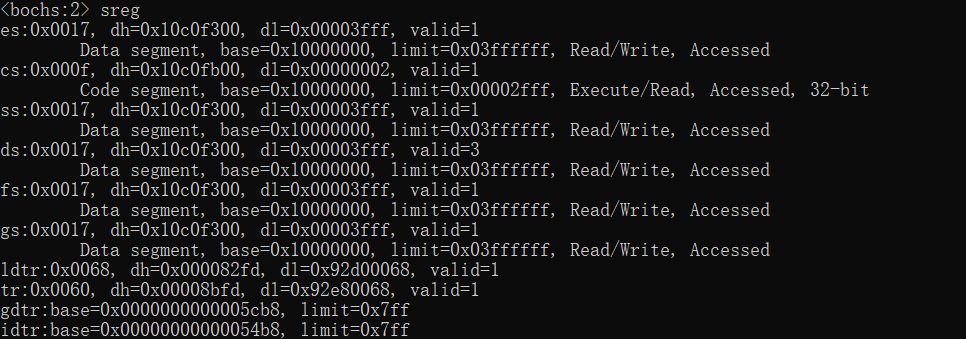


图21段的具体信息

1. 读ds段信息，根据ds段为0x0017=0000 0000 0001 0111。其中，高13位代表索引号（用红色标注出），可以读出索引号为2。索引号后一位（即低位起倒数第三位，用高亮标识的位置）为T1位，可以读出T1=1，所以段描述符存放在局部段描述符表(ADT)中，且应在局部段描述符表的第3项（起始号为0）；
2. 读LDTR寄存器信息，其存放了LDTX描述符在GDT中的位置。LDTR为0x0068=0000 0000 0110 1000。其中，高13位代表索引号（用红色标注出），可以读出索引号为13，则表示LDT起始地址存放在GDT表的第14项。（起始号为0）；
3. 读GDTR寄存器信息，GDTR为0x5cb8，即GDT在内存中的起始地址为0x5cb8。注意到每个段描述符由8个字节组成，注意到每个段描述符由8个字节组成，可以计算LDT的首地址为：0x5cb8（起始地址）+8\*13（偏移）=0x5d20。
4. 输入命令：xp /2w 0x5d20，查看GDT中对应的表项，如图22所示。



图22 GDT中对应的表项

接下来进行计算与地址拼接转化：注意到0x92d00068为低位（0-15位），0x000082fd为高位（16-31位），即LDT的段描述符用二进制位表示应为：

0000 0000 0000 0000 1000 0010 1111 1101 1001 0010 1101 0000 0000 0000 0110 1000，根据段描述符结构，第16-31位对应基址的第0-15位（用红色标注出），高32位的第0-7位对应基址的第16-23位（用蓝色标注出），高32位的第24-31位对应基址的第24-31位（用绿色标注出）。故按此原理拼接地址，可以得到LDT的基址为：0000 0000 1111 1101 1001 0010 1101 0000（0x00fd92d0）。可以作如下验证：由之前的结果可得：因为段描述符在LDT表中的偏移为2，输入命令：xp /2w 0x00fd92d0+2\*8，得到的结果如图23所示。

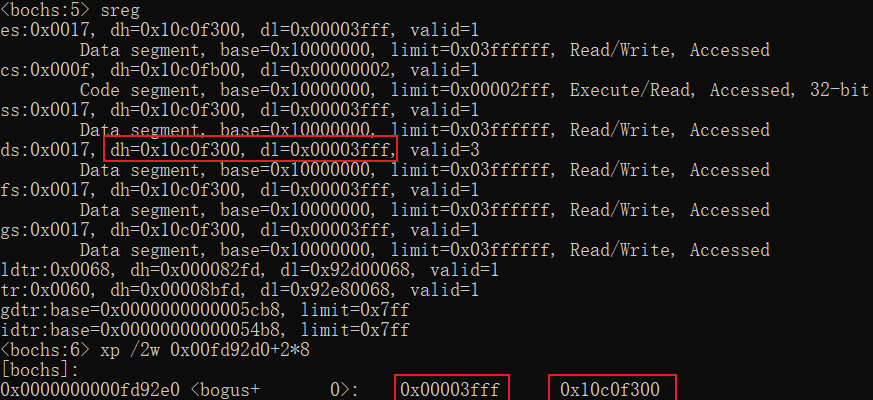


图23 xp /2w 0x00fd92d0+2\*8的结果图

查看ds段的段描述符信息，与sreg显示的ds段的dl、dh寄存器的值相同。

1. 由图21可知：ds段的基址为0x10000000，程序运行显示j的段内偏移地址为0x3004，所以线性地址为：0x10000000+0x3004=0x00003004=0001 0000 0000 0000 0011 0000 0000 0100。线性地址被划分为三部分，前10位为页目录索引(用红色标注出)，接下来的10位为页表索引(用蓝色标注出)，最低12位为偏移(用绿色标注出)。可得页目录索引为64，页表索引为3，偏移量为4；
2. 输入命令：creg，如图24所示。

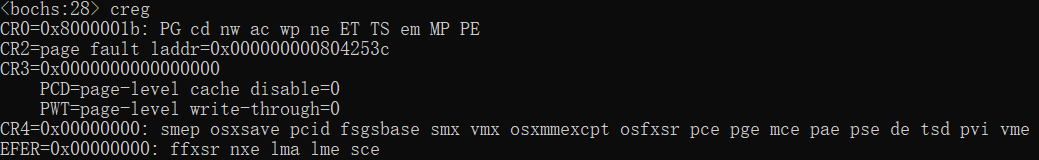


图24 CR寄存器的值

寄存器CR3的值为0，即页目录表的起始地址为0。因此，对应页目录（PDE）地址为0+64\*4=0=0x100。输入命令：xp /w 0x100，查看页目录（PDE）的值，结果如图25所示；



图25 PDE的值

PTE的值为0x00fa6027=0000 0000 1111 1010 0110 0000 0010 0111。只取其前20位（用红色标注出）作为下一级的索引，即下一级的索引为0x00fa6000。同理，对应页表（PTE）地址为0x00fa6000+3\*4=0x00fa600c。输入命令：xp /w 0x00fa600c，查看页表（PTE）的值，结果如图26所示。



图26 PTE的值

页表（PTE）的值为：0x00fa3067=0000 0000 1111 1010 0011 0000 0110 0111。同理只取其前20位（用红色标注出）作为下一级的索引，即下一级的索引为0x00fa3000。因此，得到物理地址为0x00fa3000+4=0x00fa3004；

1. 输入命令：xp /w 0x00fa3004，结果如图27所示。



图27 地址0x00fa3004的值

将结果与j所在物理地址的值对比，显示的结果正确，即已经找到了j所在的正确的物理地址0x00003004；

1. 输入命令：setpmem 0x00fa3004 4 0，将物理地址0x00fa3004的开始4个字节的值设置为0，随后输入命令：c，继续运行Linux系统。Display窗口如图28所示。



图28 Console窗口输入的命令和Display窗口显示的结果

注意到程序成功执行了第二条输出语句并返回退出，该结果证明了以上实验步骤是正确的。至此，实验顺利完成。

**八、总结及心得体会：**

通过本次实验，我详细学习了计算机的段页式内存管理机制，掌握了地址转换的过程，并且在实际操作中，能成功寻找到变量存储的具体位置，并对变量的数值成功进行修改。 在实验过程中，我错误地在Display窗口按下Ctrl+C，使得Linux操作系统中断运行，导致后面实验的错误，在纠正错误后，程序能正常运行退出，实验取得成功。

**九、对本实验过程及方法、手段的改进建议：**

完善地址映射实验指导书，使同学能够更好地书写实验内容、实验目的、实验原理等部分，也使得同学能够更好地完成实验。