电子科技大学

计算机科学与工程学院

标 准 实 验 报 告

（实验）课程名称 操作系统

电子科技大学教务处制表

电 子 科 技 大 学

实　验　报　告

学生姓名：王朴真　　学　号：2017060101015　　指导教师：薛瑞尼

实验地点：主楼 A2-412　　实验时间：12 月 20 日

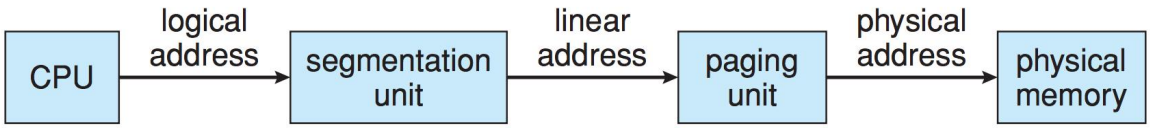
一、实验室名称：　　计算机学院实验中心

二、实验项目名称：Linux 寻址实验

三、实验学时：4 学时

四、实验原理：

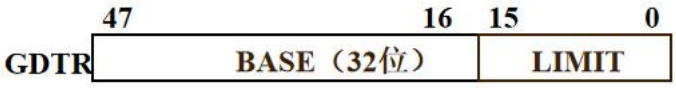
实验的总体原理是：分段机制把逻辑地址转换为线性地址，而分页机制把线性地址转换为物理地址。它们之间的转换关系如下图所示：



另外在本次地址映射过程中，需要掌握下面基本原理：

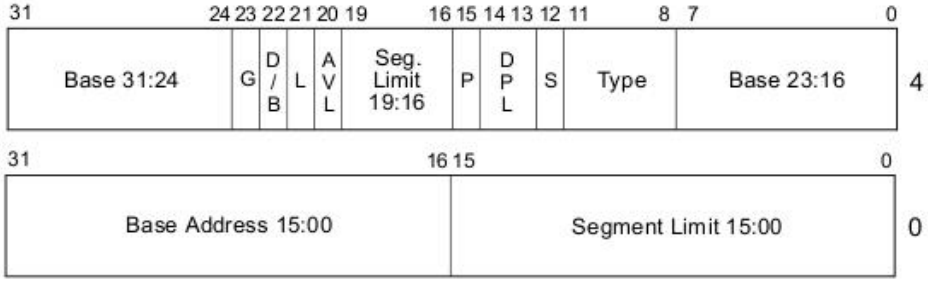
1、线性地址：就是真正的逻辑地址

2、GDTR：全局描述符表寄存器，在物理存储器地址空间中定义全局描述符表 GDT(高 32 位)，如下所示：



3、LDTR：局部描述符表寄存器，16 位的 LDTR 并不直接定义 LDT，它只是一个指向 GDT 中 LDT 描述符的选择符。如果 LDTR 中装入了选择符，相应的描述符将从 GDT 中读出段 LDT 描述符（Base+limit）。

4、段描述符：描述存储器“段”的属性的一个数据结构。其结构如下：



注意其基址被划分成了 3 部分。

5、CR0：第 0 位是保护位，保护模式下 CS、DS、SS、ES、FS、GS 寄存器称为段选择符寄存器，其值不再是基址而是选择符。另外第 31 位是分页 PG 标志，当 CR0 的 31 位是 0 时，表示线性地址就是物理地址。如果是 1，则说明启用分页机制，需进行进一步的地址转换。

6、CR3：含有存放页目录表页面的物理地址（注意，是物理地址！！！），因此 CR3 也被称为 PDBR。因为页目录表页面是页对齐的，所以该寄存器只有高 20 位是有效的。而低 12 位保留供更高级处理器使用，因此在往 CR3 中加载一个新值时低 12 位必须设置为 0。

7、Page Directory Entry/Page Table Entry 结构：20bit base + 12 属性。

五、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

六、实验内容：

通过 bochs 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

八、实验步骤：

1、修改配置文件 mybochsrchd.bxrc 的内容，然后在 linux 中运行 bochs：

```
wangpuzhen@wangpuzhen-virtual-machine:~/os/lab2-linux-bochs$ bochs -f mybochsrc-
hd.bxrc
========================================================================
                       Bochs x86 Emulator 2.6
            Built from SVN snapshot on September 2nd, 2012
========================================================================
```

2、在 vi 中编辑用例程序，并运行：

```
[/usr/root]# ./a.out
the address of  j is 0x3004
```

3、返回 bochs，使用 sreg 查看寄存器内容：

```
<bochs:2> sreg
es:0x0017, dh=0x10c0f300, dl=0x00003fff, valid=1
        Data segment, base=0x10000000, limit=0x03ffffff, Read/Write, Accessed
cs:0x000f, dh=0x10c0fb00, dl=0x00000002, valid=1
        Code segment, base=0x10000000, limit=0x00002fff, Execute/Read, Non-Confo
rming, Accessed, 32-bit
ss:0x0017, dh=0x10c0f300, dl=0x00003fff, valid=1
        Data segment, base=0x10000000, limit=0x03ffffff, Read/Write, Accessed
ds:0x0017, dh=0x10c0f300, dl=0x00003fff, valid=3
        Data segment, base=0x10000000, limit=0x03ffffff, Read/Write, Accessed
fs:0x0017, dh=0x10c0f300, dl=0x00003fff, valid=1
        Data segment, base=0x10000000, limit=0x03ffffff, Read/Write, Accessed
gs:0x0017, dh=0x10c0f300, dl=0x00003fff, valid=1
        Data segment, base=0x10000000, limit=0x03ffffff, Read/Write, Accessed
ldtr:0x0068, dh=0x000082f9, dl=0xc2d00068, valid=1
tr:0x0060, dh=0x00008bf9, dl=0xc2e80068, valid=1
gdtr:base=0x0000000000005cb8, limit=0x7ff
idtr:base=0x00000000000054b8, limit=0x7ff
01164000000i[XGUI ] Mouse capture off
```

由于 cr0 的第 0 位是 1(图见下面)，所以处于保护模式，故 ds 是选择符。将 0x0017 转换位二进制可知 Ti = 1，所以选择 LDTR 方式来做转换。

4、根据 GDTR 得到 GDT 的基址 0x5cb8，根据 LDTR（此时为 segment selector）高 13 位确定它的 LDTX 描述符在 GDT 中的位置，注意 GDT/LDT 偏移长度为 8Byte。

```
<bochs:3> xp /2w 0x5cb8 + 0x0068/8 * 8
[bochs]:
0x0000000000005d20 <bogus+       0>:    0xc2d00068      0x000082f9
01164000000i[XGUI ] Mouse capture off
```

5、根据 LDTX 描述符，取高 24-31bit、高 0-7bit，以及低 16-31bit，进行拼接组合，得到 LDT 基址 0x00f9c2d0，再根据段选择符 a 确定的相对位置(0x0017 右移三位，得 2)，确定私有的 LDTY 描述符：

```
<bochs:5> xp /2w 0x00f9c2d0 + 2 * 8
[bochs]:
0x0000000000f9c2e0 <bogus+       0>:    0x00003fff      0x10c0f300
01164000000i[XGUI ] Mouse capture off
```

6、同 5 得到 LDTY 描述符中的基址信息为 0x10000000，再加上 j 的信息 0x3004，得到线性地址：

```
<bochs:6> xp /2w 0x10000000 + 0x3004
[bochs]:
0x0000000010003004 <bogus+       0>:    0xffffffff      0xffffffff
```

7、查看 creg：

```
<bochs:7> creg
CR0=0x8000001b: PG cd nw ac wp ne ET TS em MP PE
CR2=page fault laddr=0x0000000010002f9c
CR3=0x0000000000000000
    PCD=page-level cache disable=0
    PWT=page-level write-through=0
CR4=0x00000000: smep osxsave pcid fsgsbase smx vmx osxmmexcpt osfxsr pce pge mce
 pae pse de tsd pvi vme
CR8: 0x0
EFER=0x00000000: ffxsr nxe lma lme sce
01164000000i[XGUI ] Mouse capture off
```

可以发现 CR0 的保护位和分页机制都开启，且 CR3 地址为 0，故需要将线性地址 0x10003004 转换为二进制，补全 32 位，并按 10-10-12 进行划分，得到的划分结果转换为 16 进制为 0x40-0x3-0x4。

8、由一级页表计算二级页表基址(注意偏移为 4Byte)：

```
<bochs:8> xp /2w 0x40 * 4
[bochs]:
0x0000000000000100 <bogus+       0>:    0x00faa027      0x00000000
01164000000i[XGUI ] Mouse capture off
```

9、由二级页表基址(高 20bit)加偏移得到目标页基址：

```
<bochs:9> xp /2w 0x00faa000 + 0x3*4
[bochs]:
0x0000000000faa00c <bogus+       0>:    0x00fa7067      0x00000000
01164000000i[XGUI ] Mouse capture off
```

10、再由目标页基址加上偏移 0x4 得到最后的逻辑地址：

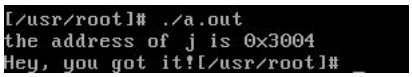
```
<bochs:10> xp /2w 0x00fa7000 + 4
[bochs]:
0x0000000000fa7004 <bogus+       0>:    0x00123456      0x00003084
01164000000i[XGUI ] Mouse capture off
```

成功找到 0x123456.

11、使用 setpmem 改变其值：

```
<bochs:13> setpmem 0xfa7004 4 0
00093000798i[XGUI ] Mouse capture off
```

九、实验数据及结果分析：



可以发现，最终修改成功，跳出循环，程序结束。

十、实验结论：

通过段页式地址转换机制，最终成功定位到目标地址。

十一、总结及心得体会：

通过本次实验，我掌握了实际操作系统中段页式内存管理机制，及时巩固了课堂中学习到的理论知识，实际操作后发现真实系统与课堂中所讲的段页式还是有很多不同，这让我对地址转换的过程有了更深更广的理解。

另外通过 bochs 手工查看系统内存，并修改特定物理内存的值这个过程中我学习了很多操作系统的东西，如 CR 寄存器，它的特定位决定 ds 等寄存器的作用；再如 LDT/GDT 由 Ti 位控制使用哪种方式等等。

在实验过程中也出了很多错误，最重要的是开始没有搞清楚偏移的字节大小，导致地址定位失败，这些错误都令我印象深刻，为以后的学习提醒。总之，这次实验让我受益良多。

十二、对本实验过程及方法、手段的改进建议：

对于 linux 中 bochs 环境的配置可以稍微做一些指导，因为一开始都不知道出现什么样的信息算环境配置好了，自己盲目摸索花费时间太长，有点打击信心。建议给出一些配置环境指导以及配置好之后的界面。

报告评分：

指导教师签字：