

# 电子科技大学计算机科学与 工程学院

## 实 验 报 告

(实验) 课程名称 计算机操作系统

学生姓名：巫少华  
学 号：2019081307020  
指导教师：李玉军

## 实验名称：内存地址转换实验

### 实验目的

- 掌握计算机的寻址过程
- 掌握页式地址地址转换过程
- 掌握计算机各种寄存器的用法

### 实验内容

本实验运行一个设置了全局变量的循环程序，通过查看段寄存器，LDT 表，GDT 表等信息，经过一系列段、页地址转换，找到程序中该全局变量的物理地址。

### 实验环境

Linux 内核 (0.11) +Bochs 虚拟机

### 实验原理

逻辑地址：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]。”段标识符：也称为段选择符，属于逻辑地址的构成部分，段标识符是由一个16 位长的字段组成，其中前 13 位是一个索引号。后面 3 位包含一些硬件细节：

索引号：可以看作是段的编号，也可以看做是相关段描述符在段表中的索引位置。系统中的段表有两类：GDT 和 LDT。

GDT：全局段描述符表，整个系统一个，GDT 表中存放了共享段的描述符，以及 LDT 的描述符（每个 LDT 本身被看作一个段）

LDT：局部段描述符表，每个进程一个，进程内部的各个段的描述符，就放在 LDT 中。

TI 字段：Intel 设计思想是：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 TI 字段表示的，TI=0，表示相应的段描述符在 GDT 中，TI=1 表示表示相应的段描述符在 LDT 中。

段描述符(即段表项)：具体描述了一个段。在段表中，存放了很多段描述符。我们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，也就是说，段标识符的前 13 位是相关段描述符在段表中的索引位置。

Base 字段：它描述了一个段的开始位置：段基址。Base(24-31)：基地址的高 8 位，Base(16-23)：基地址的中间 8 位，Base(0-15)：基地址的低 16 位。（这里的段基址，不是相应的段在内存中的起始地址，而是程序编译链接以后，这个段在程序逻辑(虚拟)地址空间里的起始位置。）

### 实验步骤

1. 启动bochs，并编写相应的程序，并将j的值改为学号的后8位为81307020



```
Bochs for Windows - Console
00035128545i[FDD] read() on floppy image returns 0
00035173500i[FDD] read() on floppy image returns 0
00035218455i[FDD] read() on floppy image returns 0
00035263410i[FDD] read() on floppy image returns 0
00035308365i[FDD] read() on floppy image returns 0
00035353320i[FDD] read() on floppy image returns 0
00035398275i[FDD] read() on floppy image returns 0
00035443230i[FDD] read() on floppy image returns 0
00035493676i[BIOS] int13_harddisk: function 15, unmapped device for ELDL=81
00781600000i[    ] Ctrl-C detected in signal handler.
Next at t=781600000
(0) [0x000000000f0c06c] 000f:000000000000006c (unk. ctxt): jz .+2 (0x10000070) ; 7402
<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, 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=0xa2d00068, valid=1
tr:0x0060, dh=0x00008bf9, dl=0xa2e80068, valid=1
gdtr:base=0x00000000000005cb8, limit=0x7ff
idtr:base=0x000000000000054b8, limit=0x7ff
<bochs:3>
```

4. 查看 LDTR 寄存器，当段选择符中的 TI=1 时，表示段描述符存放在 LDT 中，LDTR 存放的就是 LDT 在 GDT 中的索引，其中存放了 LDT 在 GDT 的位置。当前 LDTR 中的值为 0x0068，对应的 TI=0，右移 3 位之后为 0x0D，即在 GDT 中的索引为 (0x0D) 13。gdtr 存放了 GDT 的起始地址，用 `xp /2w 0x00005cb8+13*8`，这条语句的意思是用十六进制的方式查看 `0x5cb8+8×13` 处的内容，其中 `0x5cb8` 表示 GDT 的起始地址，13 表示偏移的项的数量，8 表示每个项的大小，得到的 LDT 段描述符，从而我们可以得到 LDT 的基址为 `0x00f9a2d0`

```
<bochs:3> xp /2w 0x5cb8+13*8
[bochs]:
0x00000000000005d20 <bogus+      0>:      0xa2d00068      0x000082f9
```

5. 用 `xp /2w 0x00f9a2d0+2*8`，在上面的步骤已经求出在 LDT 表中的偏移为 2，这样只要将查看 LDT 表的基址加上 `8×2` 处的内容即可查看 LDT 中第 2 项段的内容。

```
<bochs:4> xp /2w 0xf9a2d0+16
[bochs]:
0x000000000000f9a2e0 <bogus+      0>:      0x00003fff      0x10c0f300
```

6. 使用 `creg` 查看寄存器 CR3 值为 0，即页目录表（第一级页表）的起始地址为 0。

计算出 ds 段的基地址为 `0x10000000`。

计算线性地址 `0x10000000+0x3004= 0x10003004`，将其用 0 补满 32 位 (0001 0000 0000 0000 0011 0000 0000 0100)，然后按照 10-10-12 比特的

方式划分，为 `0x40-0x03-0x04`。即第一级页表内的索引为 `0x40`，第二级页表内的索引为 `0x03`，页内偏移为 `0x04`。

```
<bochs:5> creg
CR0=0x8000001b: PG cd nw ac wp ne ET TS em MP PE
CR2=page fault laddr=0x0000000010002fa8
CR3=0x0000000000000000
    PCD=page-level cache disable=0
    PWT=page-level write-through=0
CR4=0x00000000: smep osxsave pcid fsgsbase smx vmx osxsmmexcpt osfxsr pce pge mce pae pse de tsd pvi vme
EFER=0x00000000: ffxsr nxe lma lme sce
```

7. 使用 `xp /w 64*4` 查看,下一级索引为 `0x00fa9000`。

```
<bochs:6> xp /w 64*4
[bochs]:
0x000000000000000100 <bogus+      0>:  0x00fa9027
```

8. 使用 `xp /w 0x00fa9000+3*4` 查看 `0x00fa7067`, 下一级索引为 `0x00fa7000`, 得到物理地址为 `0x00fa7000+4`。

```
<bochs:7> xp /w 0xfa9000+3*4
[bochs]:
0x000000000000fa900c <bogus+      0>:  0x00fa7067  0x00000000
```

9. 使用 `xp /w 0x00fa7000+4`, 内容为 `0x81307020`。

```
<bochs:9> xp /w 0xfa7000+4
[bochs]:
0x000000000000fa7004 <bogus+      0>:  0x81307020
```

10. 使用 `setpmem 0x00fa7004 4 0`, 设置 `0x00fa7004` 开始的四个字节均为0, 再次查看 `0x00fa7004` 处的内容, 内容变成 `0x0`

```
<bochs:10> setpmem 0xfa7004 4 0
<bochs:11> xp /w 0xfa7000+4
[bochs]:
0x000000000000fa7004 <bogus+      0>:  0x00000000
```