

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                     操作系统

# 电子科技大学

# 实 验 报 告

学生姓名：蔡与望      学号：2020010801024      指导教师：刘杰彦

实验地点：主楼 A2-412      实验时间：2022/12/03

一、实验室名称：主楼 A2-412

二、实验项目名称：内存地址转换实验

三、实验学时：2 学时

四、实验目的：

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

五、实验环境

Linux 内核 (0.11) +Bochs 虚拟机。

六、实验内容和原理

本实验分为前台和后台两个部分。在前台，Linux 的 shell 中运行了一个 C 程序；它将一直死循环，直到变量 j 的值置零。在后台，我们需要在 Bochs 虚拟机中查看段表、页表 and 关键寄存器，推算出变量 j 的物理地址，并使用 setpmem 指令将 j 置零，从而使前台的 C 程序成功退出。

实验原理分为两个部分：逻辑地址到线性地址，再从线性地址到物理地址。第一部分中，我们可以使用 sreg 命令，得到 GDTR、LDTR 和 DS 的信息，然后逐步得到 GDT 基址、LDT 基址和 DS 基址，并计算出变量 j 的线性地址。第二

部分中，我们可以使用 `creg` 命令，得到一级页表基址，然后根据线性地址指示的一级索引、二级索引和页内偏移量，逐步推算得到变量 `j` 的物理地址。

## 七、实验步骤及结果分析：

(1) 从 GDTR 中获得 GDT 基址：0x00005cb8。

```
<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=0x000082fd, dl=0x92d00068, valid=1
tr:0x0060, dh=0x00008bfd, dl=0x92e80068, valid=1
gdtr:base=0x0000000000005cb8, limit=0x7ff
idtr:base=0x00000000000054b8, limit=0x7ff
```

(2) 从 LDTR 的高 13 位中获得 LDT 在 GDT 中的偏移量：13。

```
<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=0x000082fd, dl=0x92d00068, valid=1
tr:0x0060, dh=0x00008bfd, dl=0x92e80068, valid=1
gdtr:base=0x0000000000005cb8, limit=0x7ff
idtr:base=0x00000000000054b8, limit=0x7ff
```

(3) 与 GDT 基址相加，得到 LDT 基址：0x00fd92d0。

```
<bochs:3> xp /2w 0x00005cb8+13*8
[bochs]:
0x0000000000005d20 <bogus+      0>:      0x92d00068      0x000082fd
```

(4) 从 DS 的高 13 位中获得 DS 在 LDT 中的偏移量：2。

```
<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=0x000082fd, dl=0x92d00068, valid=1
tr:0x0060, dh=0x00008bfd, dl=0x92e80068, valid=1
gdt:base=0x00000000000005cb8, limit=0x7ff
idt:base=0x000000000000054b8, limit=0x7ff
```

(5) 与 LDT 基址相加, 得到 DS 基址: 0x10000000。

```
<bochs:4> xp /2w 0x00fd92d0+2*8
[bochs]:
0x0000000000fd92e0 <bogus+      0>:  0x00003fff  0x10c0f300
```

(6) 从前台 C 程序获得 DS 偏移量: 0x3004。

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

(7) 计算线性地址:  $0x10000000 + 0x00003004 = 0x10003004$ , 可知一级页表索引 0x40, 二级页表索引 0x03, 页内偏移量 0x04。

(8) 使用 creg 命令查看一级页表基址: 0。

```
<bochs:5> creg
CR0=0x8000001b: PG cd nw ac wp ne ET TS em MP PE
CR2=page fault laddr=0x000000001002fa8
CR3=0x0000000000000000
    PCD=page-level cache disable=0
    PWT=page-level write-through=0
CR4=0x00000000: smep osxsave pcid fsgsbase smx vmx osxsmexcpt osfxsr pce pge mce pae pse de tsd
    pvi vme
EFER=0x00000000: ffxsr nxe lma lme sce
```

(9) 与一级页表索引相加, 得到一级页表基址: 0x00fa7000。

```
<bochs:6> xp /2w 0+0x40*4
[bochs]:
0x00000000000000100 <bogus+      0>:  0x00fa7027  0x00000000
```

(10) 与二级页表索引相加, 得到页基址: 0x00fa6000。

```
<bochs:7> xp /2w 0x00fa7000+0x03*4
[bochs]:
0x0000000000fa700c <bogus+      0>:  0x00fa6067  0x00000000
```

(11) 与页内偏移量相加, 得到物理地址: 0x00fa6004。

```
<bochs:11> xp /2w 0x00fa6000+0x04
[bochs]:
0x0000000000fa6004 <bogus+      0>:  0x00801024  0x00003084
```

(12) 将该地址开始的四个字节置零。

```
<bochs:12> setpmem 0x00fa6004 4 0
<bochs:13> xp 0x00fa6004
[bochs]:
0x000000000000fa6004 <bogus+      0>: 0x00000000
```

(13) 回到前台，发现程序已终止。

```
[/usr/root]# gcc -o main main.c
[/usr/root]# ./main
the address of j is 0x3004
program terminated normally!
```

## 八、实验结论：

我们通过 GDTR、LDTR、DS 得到了正确的线性地址，然后将线性地址切分得到了各级页表索引和页内偏移量，最终得到了正确的物理地址：0x00801024，它的最后六位 801024 正是我学号的后六位，并且程序最终成功终止。

## 九、总结及心得体会：

通过本实验，我对于分段、分页的概念有了更深的了解和亲身的实践。对于分段，我学到了 GDT、LDT 的拓展概念；对于分页，我巩固了二级分页的流程和计算。此外，我也了解了 sreg、creg 等命令，vi 的操作也更熟练。

## 十、对本实验过程及方法、手段的改进建议：

实验指导书关于分段部分的实验步骤，顺序有点前后颠倒，我认为顺序最好是 GDTR-LDTR-DS，比较符合计算的渐进顺序，而指导书则相反。

报告评分：

指导教师签字：