

# 武汉大学国家网络安全学院教学实验报告

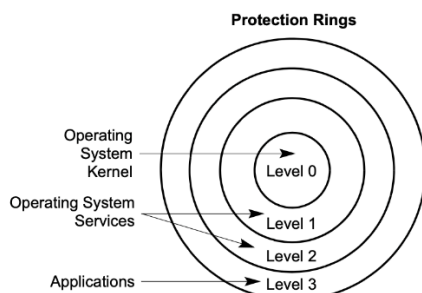
课程名称	操作系统设计与实践	实验日期	2023. 10. 15
实验名称	保护模式工作机理	实验周次	第二周
姓名	学号	专业	班级
聂森	2021302191536	网络空间安全	9 班
程子洋	2021301051114	网络空间安全	9 班
王卓	2021302191791	网络空间安全	9 班
刘琥	2021302121234	网络空间安全	9 班

## 一、实验目的及实验内容

(本次实验所涉及并要求掌握的知识；实验内容；必要的原理分析)

### 实验知识要求：

1. x86 CPU 的基本模式：实模式、保护模式
2. 环保护及其拓展



3. 段的特权类型
4. 不同特权级别段之间的代码转移
5. 不同特权级代码之间切换，上下文如何恢复？
6. 如何实现高特权级 低特权级

### 实验内容：

#### (一)

1. 认真阅读章节资料，掌握什么是保护模式，弄清关键数据结构：GDT、descriptor、selector、GDTR，及其之间关系，阅读 pm.inc 文件中数据结构以及含义，写出对宏 Descriptor 的分析
2. 调试代码，/a/ 掌握从实模式到保护模式的基本方法，画出代码流程图，如果代码/a/中，第 71 行有 dword 前缀和没有前缀，编译出来的代码有区别么，为什么，请调试截图。
3. 调试代码，/b/，掌握 GDT 的构造，体会保护模式下地址空间的变化、从保护模式切换回实模式方法
4. 调试代码，/c/，掌握 LDT 的构造
5. 调试代码，/d/掌握一致代码段、非一致代码段、数据段的权限访问规则，掌握 CPL、DPL、RPL 之间关系，以及段间切换的基本方法
6. 调试代码，/e/掌握利用调用门进行特权级变换的转移的基本方法

#### (二)

1. GDT、Descriptor、Selector、GDTR 结构，及其含义是什么？他们的关

联关系如何？pm.inc 所定义的宏怎么使用？

2. 从实模式到保护模式，关键步骤有哪些？为什么要关中断？为什么要打开 A20 地址线？从保护模式切换回实模式，又需要哪些步骤？
3. 阐述不同权限代码的切换方法，call, jmp, retf 使用场景如何，能够互换吗？
4. 课后动手改：
  - a) 自定义添加 1 个 GDT 代码段、1 个 LDT 代码段，GDT 段内要对一个内存数据结构写入一段字符串，然后 LDT 段内代码段功能为读取并打印该 GDT 的内容；
  - b) 自定义 3 个 GDT 代码段 A、B、C，B 与 A、C 分属于不同特权级，功能自定义，要求实现 A→B 的跳转，以及 B→C 的跳转。

**原理分析：**

**(一)**

### 1. 保护模式与关键数据结构：

保护模式：保护模式提供了一种机制，通过该机制，操作系统可以控制程序对内存和硬件的访问，确保系统的稳定和安全。GDT(Global Descriptor Table)：全局描述符表，是保护模式下用于存放段描述符的表格。Descriptor：描述符，是一个数据结构，用于定义段的属性，如基地址、限长、权限等。Selector：选择子，是一个特殊的值，用于在 GDT 或 LDT（局部描述符表）中索引一个描述符。GDTR：全局描述符表寄存器，存储 GDT 的基地址和限长。pm.inc 文件：通常包括与保护模式相关的数据结构和宏定义。宏“Descriptor”可能用于简化描述符的创建和管理。

### 2. 实模式到保护模式的转换：

关闭中断是为了防止在转换过程中发生中断，可能会导致系统崩溃。打开 A20 地址线是为了能够访问超过 1MB 的内存。对于代码中的 dword 前缀，它指明了操作数的大小。有或没有这个前缀可能会影响编译后的代码和运行结果。

### 3. GDT 的构造和保护模式下的地址空间变化：

在保护模式下，通过 GDT 可以定义多个段，每个段有自己的基地址和限长，保护模式提供了更灵活的内存管理机制。

### 4. LDT 的构造：

LDT(Local Descriptor Table)是与 GDT 类似的数据结构，但是它是局部的，通常用于存放特定任务或进程的段描述符。

### 5. 段的权限访问规则：

CPL(Current Privilege Level)、DPL(Descriptor Privilege Level)和 RPL(Request Privilege Level)是用于控制权限的机制。段间切换通常涉及到权限的检查，以确保只有具有适当权限的代码才能访问特定的资源。

### 6. 利用调用门进行特权级变换：

调用门是一种特殊的段描述符，用于实现不同特权级之间的控制转移。

**(二)**

1. GDT、Descriptor、Selector、GDTR 的结构及含义已在上述内容中解释，它们之间的关系是通过选择子来索引 GDT 或 LDT，从而获得对应的描述符，进而访问或控制相应的内存段。
2. 实模式到保护模式的关键步骤包括关闭中断、打开 A20 地址线、加载 GDTR 和设置控制寄存器。从保护模式切换回实模式，需要重置相关的控制寄存器和标志位，可能还需要重新配置中断和内存管理设置。
3. 不同权限代码的切换通常使用 call、jmp 和 retf 指令。它们之间的使用场景可能会有所不同，例如 call 通常用于子程序调用，jmp 用于无条件跳转，而 retf 用于返回前一指令流。在某些情况下，它们可能是可以互换的，但是需要确保满足特定的条件和约束。
4. 课后动手改：
  1. 自定义添加 GDT 和 LDT 代码段，首先需要定义相应的描述符，然后在 GDT 段内写入数据，在 LDT 段内读取并打印数据。
  2. 自定义三个 GDT 代码段 A、B、C，根据需求设置不同的特权级和功能，通过编写和调试代码实现 A→B 和 B→C 的跳转。

## 二、实验环境及实验步骤

（本次实验所使用的器件、仪器设备等的情况；具体的实验步骤）

### 实验环境：

虚拟机：

VMware Workstation Pro/VMware Workstation 16 player

操作系统：Ubuntu 16.04

模拟系统软件：Bochs 2.7

### 实验步骤：

#### （一）

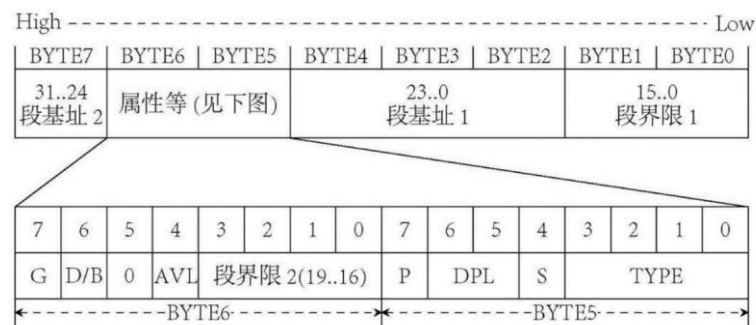
1. 认真阅读章节资料，掌握什么是保护模式，弄清关键数据结构：GDT、descriptor、selector、GDTR，及其之间关系，阅读 pm.inc 文件中数据结构以及含义，写出对宏 Descriptor 的分析
  - a) **保护模式 (Protection Mode)** 是计算机操作系统和中央处理单元 (CPU) 的一种工作模式，用于提供更高级别的内存保护、多任务支持和特权级别管理。
  - b) **GDT (Global Descriptor Table):**

GDT 是一个数据结构，用于存储系统中所有的段描述符。段描述符包含有关内存段的信息，如段的起始地址、大小、访问权限等。

**描述符 (Descriptor):**

描述符是 GDT 中的一个元素，用于描述一个内存段的属性。它包括以下信息：**基址 (Base):** 指定段的起始物理地址。**段限制 (Limit):** 指定段的大小。**访问权限 (Access Rights):** 指定段的访问级别、特权级别以及其他权限信息。**类型信息:** 指定段的类型 (代码段、数据段

等) 以及其他特性。其他控制标志: 用于指定一些特殊的段属性。



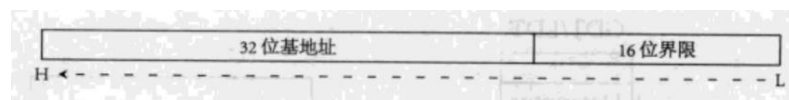
### 选择子 (Selector):

选择子是一个 16 位的值, 用于访问 GDT 中的段描述符。选择子通常存储在 CPU 的段寄存器中 (如 CS、DS、ES 等)。当程序执行时, 选择子用于选择要使用的段描述符, 从而确定如何访问内存。



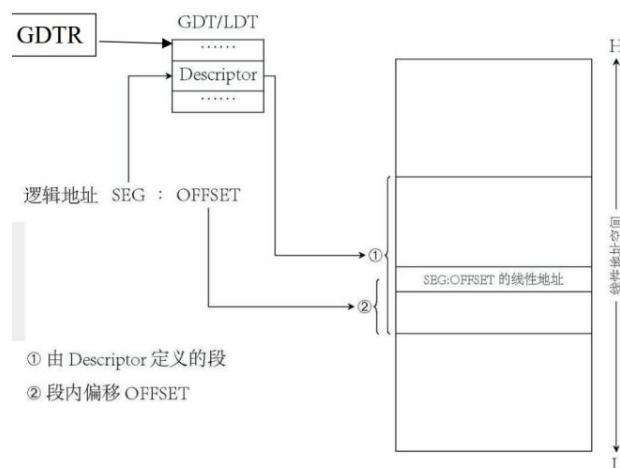
### GDTR (GDT Register):

GDTR 是一个 16 字节的寄存器, 用于存储 GDT 的起始地址和限制。GDTR 包含两个部分: 基址 (Base): 指向 GDT 在内存中的起始地址。限制 (Limit): 指定 GDT 的大小 (字节为单位)。



### 关系:

GDT 中包含多个 descriptor, descriptor 包含段的信息, 包括段基址、界限、属性等。多个 selector 包含对应 descriptor 相对于 GDT 的偏移, 所以 selector 发挥了类似指向 descriptor 的作用。而 GDTR 中包含了 GDT 的基址与界限。三者综合就可以获得某个 descriptor 的地址。而保护模式下寻址就先通过 GDTR 找到 GDT, 然后通过 descriptor 找到对应段的地址, 然后加上段内偏移 offset, 就得到某个线性地址。



c) 代码分析:

macro 代表宏开始。宏名为 Descriptor, 3 代表有三个参数。参数 1-3 分别为段基址、界限、属性。比如 LABEL\_DESC\_VIDEO: Descriptor 0B8000h, 0ffffh, DA\_DRW ; 显存首地址利用宏 Descriptor 定义了基址为 0B8000h 的段 LABEL\_DESC\_VIDEO0B8000h 为显存首地址, 利用该段在屏幕中显示数据。第一行 dw 为两字节, %2 & 0FFFFh 相当于取段界限的低位, 写入这两字节。然后 dw、dd 取段基址 1、2, 构成三四节段基址, 相当于上面结构图的段基址 1, 然后 dw 两字节构成段属性、段界限 2, 然后 dw 两字节构成段基址 3, 其中段基址为该段起始地址, 界限为长度。

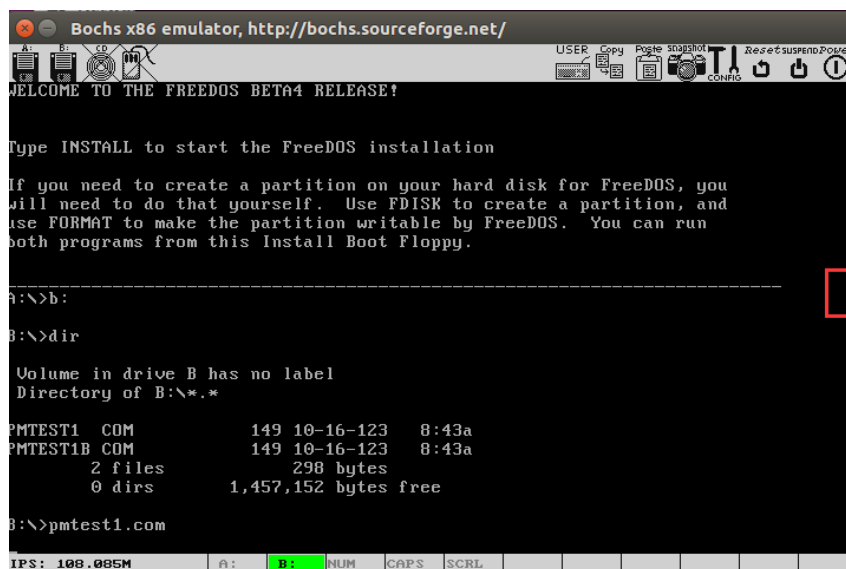
2. 调试代码, /a/ 掌握从实模式到保护模式的基本方法, 画出代码流程图, 如果代码/a/中, 第 71 行有 dword 前缀和没有前缀, 编译出来的代码有区别么, 为什么, 请调试截图。

实模式到保护模式的代码流程:

- 1) 准备 GDT;
- 2) 用 lgdt 加载 gdt;
- 3) 打开 A20;
- 4) 置寄存器 cr0 的 PE 位; (给出 cr0 的结构示意图)
- 5) 跳转, 进入保护模式

当有 dword 前缀时:

运行结果:



```
Bochs x86 emulator, http://bochs.sourceforge.net/
WELCOME TO THE FREEDOS BETA4 RELEASE!

Type INSTALL to start the FreeDOS installation

If you need to create a partition on your hard disk for FreeDOS, you
will need to do that yourself. Use FDISK to create a partition, and
use FORMAT to make the partition writable by FreeDOS. You can run
both programs from this Install Boot Floppy.

A:\>b:
B:\>dir

Volume in drive B has no label
Directory of B:\*. *

PMTEST1  COM           149 10-16-123   8:43a
PMTEST1B COM          149 10-16-123   8:43a
  2 files                298 bytes
   0 dirs             1,457,152 bytes free

B:\>pmtest1.com
```

图 1 有 dword 的运行结果展示

查看编译后的二进制文件内容:

```

Setting up libdata-nexdumper-perl (3.0001-1) ...
wz@ubuntu:~/Desktop/chapter3/a$ hd pntest1.com
00000000 e9 21 00 00 00 00 00 00 00 00 00 00 00 14 00 00 00 |!.....|
00000010 00 98 40 00 ff ff 00 80 0b 92 00 00 17 00 00 00 |..@.....|
00000020 00 00 00 00 8c c8 8e d8 8e c0 8e d0 bc 00 01 66 |.....f|
00000030 31 c0 8c c8 66 c1 e0 04 66 05 80 01 00 00 a3 0e |1...f...f.....|
00000040 01 66 c1 e8 10 a2 10 01 88 26 13 01 66 31 c0 8c |.f.....&..f1..|
00000050 d8 66 c1 e0 04 66 05 04 01 00 00 66 a3 1e 01 0f |.f...f...f....|
00000060 01 16 1c 01 fa e4 92 0c 02 e6 92 0f 20 c0 66 83 |......f..|
00000070 c8 01 0f 22 c0 66 ea 00 00 00 00 08 00 00 00 00 |...".f.....|
00000080 66 b8 10 00 8e e8 bf 7e 07 00 00 b4 0c b0 50 65 |f.....~.....Pe|
00000090 66 89 07 eb fe |f....|
00000095

```

图 2 有 dword 的编译后代码

调试查看寄存器内容:

```

<bochs:2> sreg
es:0x3224, dh=0x00009303, dl=0x2240ffff, valid=1
  Data segment, base=0x00032240, limit=0x0000ffff, Read/Write, Accessed
cs:0x3224, dh=0x00009303, dl=0x2240ffff, valid=1
  Data segment, base=0x00032240, limit=0x0000ffff, Read/Write, Accessed
ss:0x3224, dh=0x00009303, dl=0x2240ffff, valid=1
  Data segment, base=0x00032240, limit=0x0000ffff, Read/Write, Accessed
ds:0x3224, dh=0x00009303, dl=0x2240ffff, valid=7
  Data segment, base=0x00032240, limit=0x0000ffff, Read/Write, Accessed
fs:0x0000, dh=0x00009300, dl=0x0000ffff, valid=1
  Data segment, base=0x00000000, limit=0x0000ffff, Read/Write, Accessed
gs:0x0000, dh=0x00009300, dl=0x0000ffff, valid=1
  Data segment, base=0x00000000, limit=0x0000ffff, Read/Write, Accessed
ldtr:0x0000, dh=0x00008200, dl=0x0000ffff, valid=1
tr:0x0000, dh=0x00008b00, dl=0x0000ffff, valid=1
gdtr:base=0x00032344, limit=0x17
idtr:base=0x00000000, limit=0x3ff
<bochs:3> s
Next at t=304175790
(0) [0x0000000323c0] 0008:00000000 (unk. ctxt): mov ax, 0x0010 ; 66b81000
<bochs:4> sreg
es:0x3224, dh=0x00009303, dl=0x2240ffff, valid=1
  Data segment, base=0x00032240, limit=0x0000ffff, Read/Write, Accessed
cs:0x0008, dh=0x000409903, dl=0x23c00014, valid=1
  Code segment, base=0x000323c0, limit=0x00000014, Execute-Only, Non-Conforming
32-bit
ss:0x3224, dh=0x00009303, dl=0x2240ffff, valid=1
  Data segment, base=0x00032240, limit=0x0000ffff, Read/Write, Accessed
ds:0x3224, dh=0x00009303, dl=0x2240ffff, valid=7
  Data segment, base=0x00032240, limit=0x0000ffff, Read/Write, Accessed
fs:0x0000, dh=0x00009300, dl=0x0000ffff, valid=1
  Data segment, base=0x00000000, limit=0x0000ffff, Read/Write, Accessed
gs:0x0000, dh=0x00009300, dl=0x0000ffff, valid=1
  Data segment, base=0x00000000, limit=0x0000ffff, Read/Write, Accessed
ldtr:0x0000, dh=0x00008200, dl=0x0000ffff, valid=1
tr:0x0000, dh=0x00008b00, dl=0x0000ffff, valid=1
gdtr:base=0x00032344, limit=0x17
idtr:base=0x00000000, limit=0x3ff
<bochs:5> s
Next at t=304175791
(0) [0x0000000323c4] 0008:00000004 (unk. ctxt): mov gs, ax ; 8ee8

```

图 3 有 dword 的调试截图

当没有 dword 前缀时:  
运行结果:

```

Bochs x86 emulator, http://bochs.sourceforge.net/
WELCOME TO THE FREEDOS BETA4 RELEASE!

Type INSTALL to start the FreeDOS installation

If you need to create a partition on your hard disk for FreeDOS, you
will need to do that yourself. Use FDISK to create a partition, and
use FORMAT to make the partition writable by FreeDOS. You can run
both programs from this Install Boot Floppy.

A:\>b:
B:\>dir

Volume in drive B has no label
Directory of B:\*.

PMTEST1 COM          149 10-16-123  12:05p
PMTEST1B COM         149 10-16-123  12:05p
    2 files             298 bytes
    0 dirs             1,457,152 bytes free

B:\>pntest1b.com

IPS: 100.233M  A:  B: NUM CAPS SCRL

```

图 4 无 dword 的运行结果展示

查看编译后的二进制文件内容：

```
wz@ubuntu:~/Desktop/chapter3/a$ hd pmtest1b.com
00000000 e9 21 00 00 00 00 00 00 00 00 00 14 00 00 00 |.!.|
00000010 00 98 40 00 ff ff 00 80 0b 92 00 00 17 00 00 00 |..@.|
00000020 00 00 00 00 8c c8 8e d8 8e c0 8e d0 bc 00 01 66 |.....f|
00000030 31 c0 8c c8 66 c1 e0 04 66 05 7c 01 00 00 a3 0e |1...f...f.|
00000040 01 66 c1 e8 10 a2 10 01 88 26 13 01 66 31 c0 8c |.f.....&..f1..|
00000050 d8 66 c1 e0 04 66 05 04 01 00 00 66 a3 1e 01 0f |.f...f.....f...|
00000060 01 16 1c 01 fa e4 92 0c 02 e6 92 0f 20 c0 66 83 |.....f..|
00000070 c8 01 0f 22 c0 87 db ea 00 00 08 00 66 b8 10 00 |..."...f...|
00000080 8e e8 bf 7e 07 00 00 b4 0c b0 50 65 66 89 07 eb |...~.....Pef...|
00000090 fe |.|
00000091
```

图 5 无 dword 的编译后代码

调试查看寄存器内容：

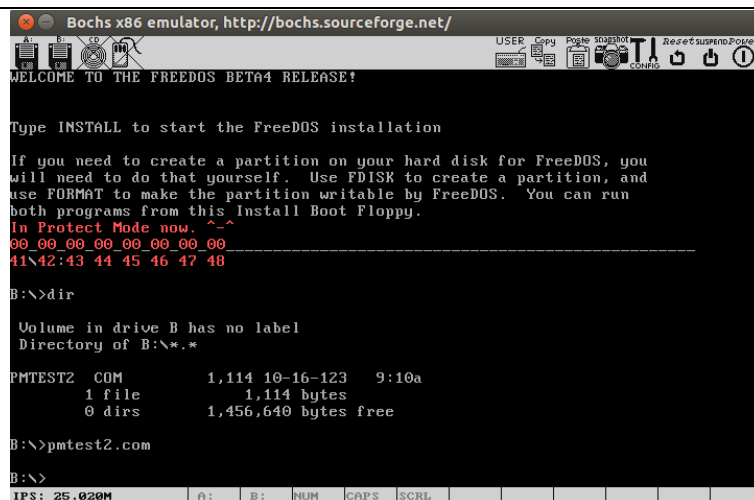
```
(0) [0x000000323b7] 3224:00000177 (unk. ctxt): jmpf 0x0008:0000 ; ea00
<bochs:2> sreg
es:0x3224, dh=0x00009303, dl=0x2240ffff, valid=1
Data segment, base=0x00032240, limit=0x0000ffff, Read/Write, Accessed
cs:0x3224, dh=0x00009303, dl=0x2240ffff, valid=1
Data segment, base=0x00032240, limit=0x0000ffff, Read/Write, Accessed
ss:0x3224, dh=0x00009303, dl=0x2240ffff, valid=1
Data segment, base=0x00032240, limit=0x0000ffff, Read/Write, Accessed
ds:0x3224, dh=0x00009303, dl=0x2240ffff, valid=7
Data segment, base=0x00032240, limit=0x0000ffff, Read/Write, Accessed
fs:0x0000, dh=0x00009300, dl=0x0000ffff, valid=1
Data segment, base=0x00000000, limit=0x0000ffff, Read/Write, Accessed
gs:0x0000, dh=0x00009300, dl=0x0000ffff, valid=1
Data segment, base=0x00000000, limit=0x0000ffff, Read/Write, Accessed
ldtr:0x0000, dh=0x00008200, dl=0x0000ffff, valid=1
tr:0x0000, dh=0x00008b00, dl=0x0000ffff, valid=1
gdtr:base=0x00032344, limit=0x17
ldtr:base=0x00000000, limit=0x3ff
<bochs:3> s
Next at t=259364241
(0) [0x000000323bc] 0008:00000000 (unk. ctxt): mov ax, 0x0010 ; 66b8
<bochs:4> sreg
es:0x3224, dh=0x00009303, dl=0x2240ffff, valid=1
Data segment, base=0x00032240, limit=0x0000ffff, Read/Write, Accessed
cs:0x0008, dh=0x00409903, dl=0x23bc0014, valid=1
Code segment, base=0x000323bc, limit=0x00000014, Execute-Only, Non-Confo
32-bit
ss:0x3224, dh=0x00009303, dl=0x2240ffff, valid=1
Data segment, base=0x00032240, limit=0x0000ffff, Read/Write, Accessed
ds:0x3224, dh=0x00009303, dl=0x2240ffff, valid=7
Data segment, base=0x00032240, limit=0x0000ffff, Read/Write, Accessed
fs:0x0000, dh=0x00009300, dl=0x0000ffff, valid=1
Data segment, base=0x00000000, limit=0x0000ffff, Read/Write, Accessed
gs:0x0000, dh=0x00009300, dl=0x0000ffff, valid=1
Data segment, base=0x00000000, limit=0x0000ffff, Read/Write, Accessed
ldtr:0x0000, dh=0x00008200, dl=0x0000ffff, valid=1
tr:0x0000, dh=0x00008b00, dl=0x0000ffff, valid=1
gdtr:base=0x00032344, limit=0x17
ldtr:base=0x00000000, limit=0x3ff
<bochs:5> s
Next at t=259364242
```

图 6 无 dword 的调试截图

综上所述，对比发现：是否删除 dword，两个可执行文件在 jmp 指令执行前后的寄存值相同，所以运行产生的结果相同，但实际上该问是由于跳转偏移量为 0，尽管发生偏移的截断也仍旧为 0，不会产生影响，但如果目标地址的偏移量较大，就会产生问题。

3. 调试代码，/b/，掌握 GDT 的构造与切换，从保护模式切换回实模式方法





```
Bochs x86 emulator, http://bochs.sourceforge.net/
WELCOME TO THE FREEDOS BETA4 RELEASE!

Type INSTALL to start the FreeDOS installation

If you need to create a partition on your hard disk for FreeDOS, you
will need to do that yourself. Use FDISK to create a partition, and
use FORMAT to make the partition writable by FreeDOS. You can run
both programs from this Install Boot Floppy.
In Protect Mode now. ^-^
00 00 00 00 00 00 00 00
41 42 43 44 45 46 47 48
-----
B:\>dir

Volume in drive B has no label
Directory of B:\*. *

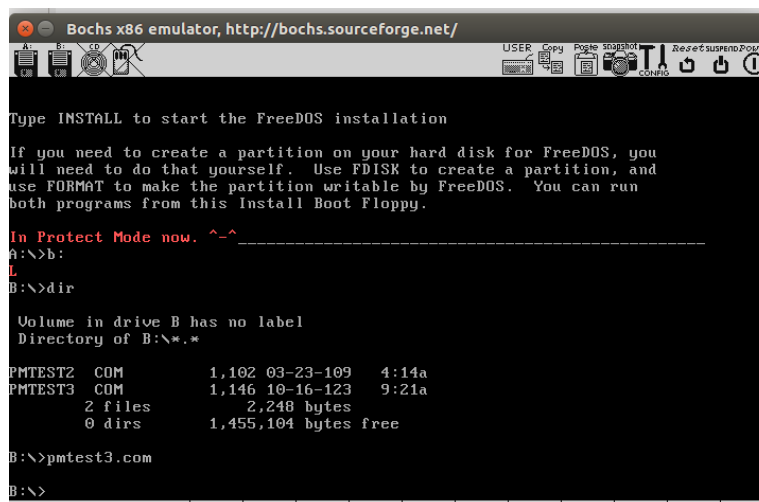
PMTEST2  COM           1,114  10-16-123   9:10a
          1 file             1,114 bytes
          0 dirs           1,456,640 bytes free

B:\>pmtest2.com
B:\>
```

图 7 调试/b/结果

可以看到，程序打印出两行数字，第一行全部是零，说明内存地址 5MB 处都是 0，而下一行变成了 41 42 43...，即十六进制的 A、B、C、...、H，说明写操作成功。同时，程序执行结束后不再像上一个程序一样进入死循环，而是重新出现 DOS 提示符 B:\>，表明我们重新回到了实模式下的 DOS。

#### 4. 调试代码，/c/，掌握 LDT 切换



```
Bochs x86 emulator, http://bochs.sourceforge.net/
WELCOME TO THE FREEDOS BETA4 RELEASE!

Type INSTALL to start the FreeDOS installation

If you need to create a partition on your hard disk for FreeDOS, you
will need to do that yourself. Use FDISK to create a partition, and
use FORMAT to make the partition writable by FreeDOS. You can run
both programs from this Install Boot Floppy.
In Protect Mode now. ^-^
a:\>b:
L
B:\>dir

Volume in drive B has no label
Directory of B:\*. *

PMTEST2  COM           1,102  03-23-109   4:14a
PMTEST3  COM           1,146  10-16-123   9:21a
          2 files             2,248 bytes
          0 dirs           1,455,104 bytes free

B:\>pmtest3.com
B:\>
```

图 8 调试/c/结果

在[SECTION .S32]中打印完 “In Protect Mode now” 字符串后，出现一个红色的字符 L。

#### 5. 调试代码，/d/掌握一致代码段、非一致代码段、数据段的权限访问规则，掌握 CPL、DPL、RPL 之间关系，以及段间切换的基本方法

##### 【一致代码段、非一致代码段、数据段的权限访问规则】

##### 一致代码段（Conforming Code Segment）：

- 允许低特权级别（CPL）的代码跳转（使用 call 或 jmp 指令）到高特权级别的一致代码段。



- 当从低 CPL 的代码段跳转到一致代码段时，CPU 会将 CPL 设置为目标段的 DPL（描述符特权级），这可以使特权级别降低，但不提高。
- 一致代码段通常用于实现库函数或系统调用，以便用户级别代码可以访问内核级别的功能。

#### **非一致代码段（Non-Conforming Code Segment）：**

- 允许低特权级别的代码跳转（使用 call 或 jmp 指令）到高特权级别的非一致代码段。即  $CPL-A \geq DPL-B$ ，此处 RPL 并不检查。
- 跳转到非一致代码段时，CPL 不会改变，仍然保持为低特权级别。
- 非一致代码段通常用于系统内核，以允许高特权级别的代码访问内核功能。

#### **数据段：**

- 数据段用于存储数据，可以包括全局变量、堆内存和栈内存等。
- 数据段的权限通常是读（Read）、写（Write）、执行（Execute）等，权限由段描述符中的属性字段定义。
- 数据段的权限控制读写操作以及是否可以执行其中的指令。

#### **【CPL、DPL、RPL 之间关系】**

##### **CPL：**

- 描述当前执行程序或者任务的特权级。存储在 CS 和 SS 的 bit 0，bit 1。当程序在不同特权级代码间转移，CPL 会发生改变。

##### **DPL：**

- 描述段或者门的特权级，存储在描述符的 DPL 字段中，是这个段的特权级别，用来表明访问这个段时候，所需要的特权。

##### **RPL：**

- 描述选择子的特权级，段选择子（Selector）的 bit0 和 bit1，是可以重载的。例如：被调用的系统过程（CPL=0）从调用应用过程（CPL=3）收到一个选择子，就会把这个选择子的 RPL 设置成调用者的，从而表明当前是代表调用者的特权级在工作 CPL=3，而不是 CPL=0

#### **关系总结：**

- 如果 RPL（请求特权级别）低于等于 CPL（当前特权级别），则访问是允许的。例如，用户代码（CPL 3）可以请求访问用户数据段（DPL 3）或用户代码段（DPL 3）。
- 如果 RPL 高于 CPL，即请求的特权级别更高，访问将被拒绝。例如，用户代码（CPL 3）无法请求访问内核代码段（DPL 0）。
- 如果 CPL 大于 DPL，段描述符中的 DPL 决定了可以访问该段的最高特权级别。

#### **【段间切换的基本方法】**

##### **1. 准备好目标段描述符：**

在要进行切换的特权级别下，需要准备好目标段的描述符，包括选择子（Selector）和段基地址。这通常在全局描述符表（GDT）或局部描述符表（LDT）中。

##### **2. 加载目标选择子：**

使用 mov 指令将目标选择子加载到相应的段寄存器中。不同的段寄存器对应不同的段，例如 CS（代码段）、DS（数据段）、ES（附加数据段）、SS（堆栈段）等。

### 3. 特权级别切换:

特权级别切换的方法取决于从哪个特权级别切换到哪个特权级别。在从用户模式 (CPL 3) 切换到内核模式 (CPL 0) 的情况下, 通常需要执行以下步骤:

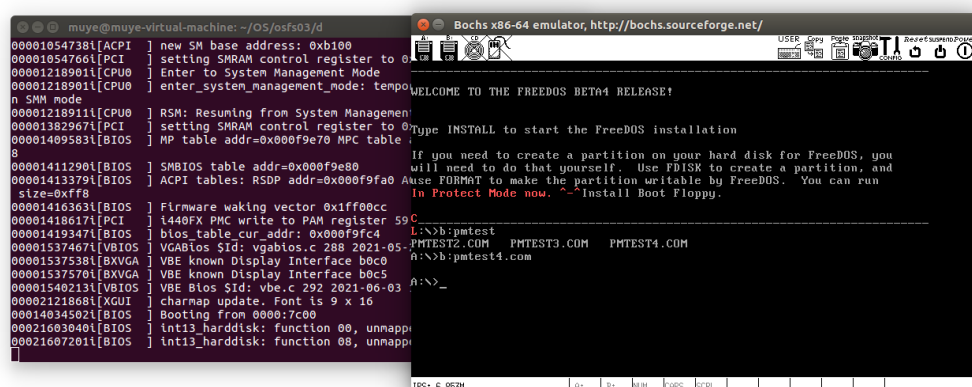
- (1) 关中断 (使用 cli 指令): 这是为了防止在特权级别切换过程中发生中断, 从而确保切换的原子性。
- (2) 打开 A20 地址线: 在某些系统中, 需要通过 I/O 端口操作来打开 A20 地址线, 以允许访问整个物理内存。
- (3) 修改 CR0 寄存器的控制位 (Control Register 0): 将 CR0 寄存器的最低位 (PE 位) 设置为 1, 以进入保护模式。
- (4) 使用 jmp 指令: 通过 jmp 指令, 将控制权转移到新的代码段, 以启动新特权级别的执行。

### 4. 完成特权级别切换:

在新特权级别下, CPU 会从目标段开始执行指令。此时, CPL (当前特权级别) 将与目标段的 DPL (描述符特权级别) 相匹配。

#### 【/d/代码运行结果】

如下图所示, 成功在不同特权级别之间进行切换, 并显示字符 L 和 C。



```
000010547381[ACPI] new SM base address: 0xb100
000010547661[PCI] setting SMRAM control register to 0
000012189011[CPU0] Enter to System Management Mode
000012189011[CPU0] enter_system_management_mode: temp
n SMM mode
000012189111[CPU0] RSM: Resuming from System Management
000013829671[PCI] setting SMRAM control register to 0
000014095831[BIOS] HP table addr=0x000f9e70 HPc table
8
000014112901[BIOS] SMBIOS table addr=0x000f9e80
000014133791[BIOS] ACPI tables: RSDP addr=0x000f9fa0
size=0xffff
000014163631[BIOS] Firmware waking vector 0xffff00cc
000014186171[PCI] i440FX PWC write to PAW register 59C
000014193471[BIOS] bios table cur_addr: 0x000f9fc4
000015374671[VBIOS] VGBios Sid: vgbios.c 288 2021-05-PMTEST2.COM PMTEST3.COM PMTEST4.COM
000015375381[BXVGA] VBE known Display Interface b0c0
000015375701[BXVGA] VBE known Display Interface b0c5
000015402131[VBIOS] VBE Bios Sid: vbe.c 292 2021-06-03
000021218681[XGUL] charmap update, Font is 9 x 16
000148345021[BIOS] Booting from 0000:7c00
000210030401[BIOS] int13_harddisk: function 00, unmappe
000210072011[BIOS] int13_harddisk: function 08, unmappe
```

图 9 /d/代码运行结果

### 6. 调试代码, /e/掌握利用调用门进行特权级变换的转移的基本方法

#### 【调用门进行特权级变换】

调用门本质上是一个添加了属性的特殊入口地址: 代码 A→调用门 G→代码 B;

通过 Call Gate + Call, 实现了从低特权级→高特权级的访问

#### 1. 创建调用门:

在全局描述符表 (GDT) 或局部描述符表 (LDT) 中创建一个调用门描述符。这个描述符定义了目标代码段的选择子 (Selector)、目标代码段的偏移地址、目标代码段的特权级别 (DPL, Descriptor Privilege Level) 以及其他相关属性。

#### 2. 调用调用门:

使用 call 指令来调用创建的调用门。例如, call SelectorCallGate:0, 其中 SelectorCallGate 是调用门的选择子。

call 指令会触发特权级别的转移, 将控制权传递到目标代码段。

#### 3. 执行目标代码段:

一旦调用门被触发，CPU 会开始执行目标代码段。

目标代码段的 DPL（Descriptor Privilege Level）可以与当前特权级别（CPL，Current Privilege Level）不同。

#### 4. 返回：

在目标代码段执行完成后，通常使用 `retf` 指令返回到调用门所在的特权级别。返回时，CPU 将恢复原来的特权级别和状态。

##### 【/e/代码运行结果】

##### 1. pmtest5.asm:

- (1) /e/ 中代码使用 `call` 指令，调用了一个测试的调用门（`LABEL_CALL_GATE_TEST`），这个调用门的 DPL 为 3，表示特权级别为 3。通过 `call` 指令，CPU 触发了特权级别的转移。
- (2) 在调用门目标段（`LABEL_SEG_CODE_DEST`）中，显示了一个字符 C，表明特权级别已经从 3（Ring3）切换到目标段的特权级别。结果如下图所示：

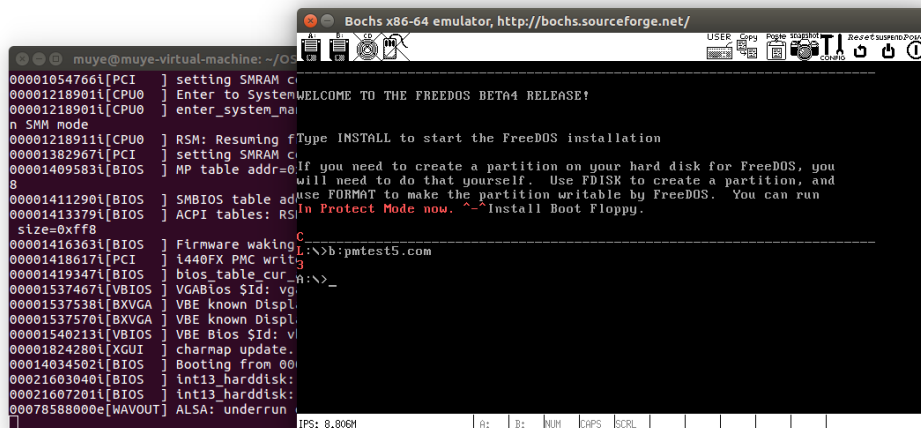


图 10 pmtest5.asm 运行结果

##### 2. pmtest5a.asm:

- (1) 调用门 `SelectorCallGateTest` 允许代码在不改变特权级别的情况下切换到 Ring 0，从而实现了从 Ring 3 代码（用户模式）调用 Ring 0 代码（内核模式）的目的。
- (2) 使用跳转指令调用 `SelectorCallGateTest`。未继续跳转。结果如下图所示：

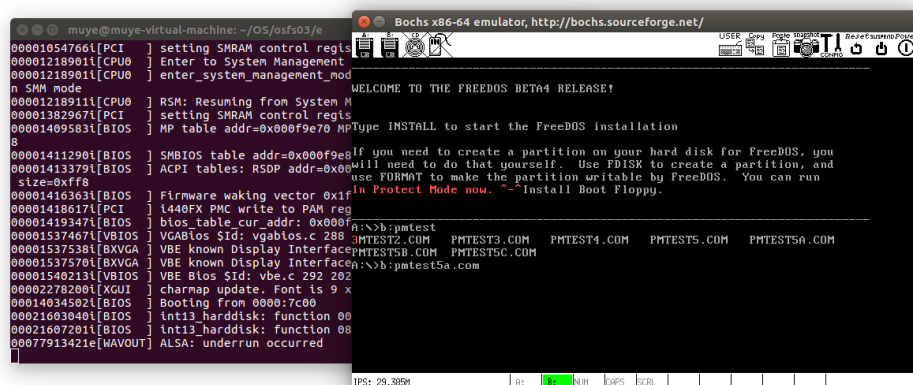


图 11 pmtest5a.asm 运行结果

### 3. pmtest5b.asm:

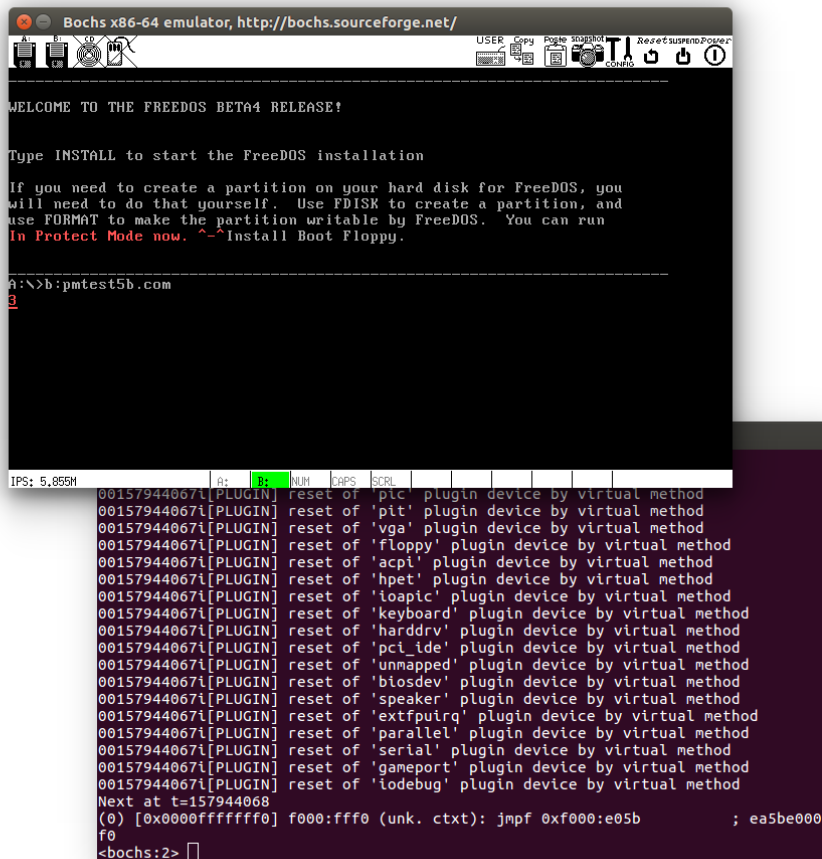
- (1) 较上述代码调整了如下部分，修改了数据段选择子的特权级(SA\_RPL3)。

```
SelectorData equ LABEL_DESC_DATA - LABEL_GDT + SA_RPL3
```

- (2) 结果发生中止，我们违反了特权级的规则，用 RPL=3 的选择子去访问 DPL=1 的段，于是引起异常。结果如下图所示：

```
00146525604e[WAVOUT] ALSA: underrun occurred
00174628399e[CPU0] ] fetch_raw_descriptor: GDT: index (ff57) 1fea > limit (5f)
00174628399e[CPU0] ] interrupt(): gate descriptor is not valid sys seg (vector=0
x0a)
00174628399e[CPU0] ] interrupt(): gate descriptor is not valid sys seg (vector=0
x08)
00174628399i[CPU0] ] CPU is in protected mode (active)
```

图 12 结果中止



```
Bochs x86-64 emulator, http://bochs.sourceforge.net/

WELCOME TO THE FREEDOS BETA4 RELEASE!

Type INSTALL to start the FreeDOS installation

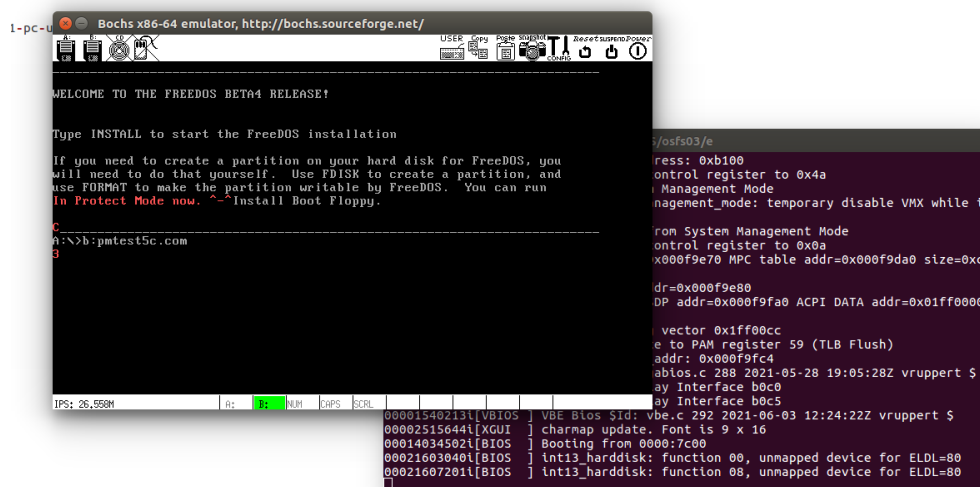
If you need to create a partition on your hard disk for FreeDOS, you
will need to do that yourself. Use FDISK to create a partition, and
use FORMAT to make the partition writable by FreeDOS. You can run
In Protect Mode now. ^~ Install Boot Floppy.

A:\>b:pmtest5b.com
3
IPS: 5,855H
A: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
00157944067i[PLUGIN] reset of 'pic' plugin device by virtual method
00157944067i[PLUGIN] reset of 'pit' plugin device by virtual method
00157944067i[PLUGIN] reset of 'vga' plugin device by virtual method
00157944067i[PLUGIN] reset of 'floppy' plugin device by virtual method
00157944067i[PLUGIN] reset of 'acpi' plugin device by virtual method
00157944067i[PLUGIN] reset of 'hpet' plugin device by virtual method
00157944067i[PLUGIN] reset of 'ioapic' plugin device by virtual method
00157944067i[PLUGIN] reset of 'keyboard' plugin device by virtual method
00157944067i[PLUGIN] reset of 'harddrv' plugin device by virtual method
00157944067i[PLUGIN] reset of 'pci_ide' plugin device by virtual method
00157944067i[PLUGIN] reset of 'unmapped' plugin device by virtual method
00157944067i[PLUGIN] reset of 'biosdev' plugin device by virtual method
00157944067i[PLUGIN] reset of 'speaker' plugin device by virtual method
00157944067i[PLUGIN] reset of 'extfpuirq' plugin device by virtual method
00157944067i[PLUGIN] reset of 'parallel' plugin device by virtual method
00157944067i[PLUGIN] reset of 'serial' plugin device by virtual method
00157944067i[PLUGIN] reset of 'gameport' plugin device by virtual method
00157944067i[PLUGIN] reset of 'iodebug' plugin device by virtual method
Next at t=157944068
(0) [0x0000fffff0] f000:fff0 (unk. ctxt): jmpf 0xf000:e05b ; ea5be000
f0
<bochs:2>
```

图 13 pmtest5b.asm 运行结果

#### 4. pmtest5c.asm:

- (1) 调用门不进行特权级别变换，仅简单地显示字符 C。
- (3) 控制回到 SelectorCodeRing3，并继续执行，然后进入无限循环，无法再继续执行。故而未显示出字符 L。结果如下图所示：



```
Bochs x86-64 emulator, http://bochs.sourceforge.net/

WELCOME TO THE FREEDOS BETA4 RELEASE!

Type INSTALL to start the FreeDOS installation

If you need to create a partition on your hard disk for FreeDOS, you
will need to do that yourself. Use FDISK to create a partition, and
use FORMAT to make the partition writable by FreeDOS. You can run
In Protect Mode now. ^~ Install Boot Floppy.

A:\>b:pmtest5c.com
C
IPS: 26,558H
A: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
00001540213i[VB105] VBE Bios Sid: Vbe-c 292 2021-06-03 12:24:22Z vruppert $
00002515644i[XGUI] charmap update, Font is 9 x 16
00014034502i[B105] Booting from 0000:7c00
00021603040i[B105] int13_harddisk: function 00, unmapped device for ELDL=80
00021607201i[B105] int13_harddisk: function 08, unmapped device for ELDL=80
i/osfs03/e
ress: 0xb100
ontrol register to 0x4a
Management Mode
nagement_mode: temporary disable VMX while i
rom System Management Mode
ontrol register to 0x0a
x000f9e70 MPC table addr=0x000f9da0 size=0xc
dr=0x000f9e80
DP addr=0x000f9fa0 ACPI DATA addr=0x01ff0000
vector 0x1ff00cc
e to PAM register 59 (TLB Flush)
addr: 0x000f9fc4
abios.c 288 2021-05-28 19:05:28Z vruppert $
ay Interface b0c0
ay Interface b0c5
VBE Bios Sid: Vbe-c 292 2021-06-03 12:24:22Z vruppert $
charmap update, Font is 9 x 16
Booting from 0000:7c00
int13_harddisk: function 00, unmapped device for ELDL=80
int13_harddisk: function 08, unmapped device for ELDL=80
```

图 14 pmtest5c.asm 运行结果

1. GDT、Descriptor、Selector、GDTR 结构，及其含义是什么？他们的关联关系如何？pm.inc 所定义的宏怎么使用？

详见（一）问题 1

2. 从实模式到保护模式，关键步骤有哪些？为什么要关中断？为什么要打开 A20 地址线？从保护模式切换回实模式，又需要哪些步骤？

**(1) 实模式到保护模式的关键步骤：**

- ① 初始化 GDT 描述符；
- ② 加载 gdt；
- ③ 打开 A20 地址线；
- ④ 设置寄存器 CR0 的 PE 位为 1，使之运行于保护模式；
- ⑤ 执行跳转指令，让系统进入保护模式

**(2) 为什么要进行关中断：**

关中断是为了确保在切换到保护模式时，不会发生中断处理程序的执行。这是因为在实模式下，中断处理程序的地址是通过中断向量表中的中断向量来确定的，而在保护模式下，中断处理程序的地址是通过中断描述符表（IDT）中的中断门来确定的。如果在切换过程中发生中断，处理器可能会尝试执行错误的中断处理程序，导致系统异常。

**(3) 为什么要打开 A20 地址线：**

A20 地址线是用于扩展地址总线的一条线路。在实模式下，8086 处理器只能访问 1MB 的物理内存空间，而在保护模式下，处理器可以访问更大的内存空间。打开 A20 地址线可以使处理器能够访问超过 1MB 的内存，以充分利用保护模式下的内存管理功能。

**(4) 保护模式切换回实模式的步骤：**

- ① 关中断：与从实模式到保护模式的切换一样，切换回实模式前需要关中断。
- ② 清除控制寄存器：将控制寄存器 CR0 的第 0 位清零，以禁用保护模式。
- ③ 刷新段选择子：在切换回实模式后，需要刷新所有段选择子，以便它们指向实模式下的段描述符。
- ④ 重新加载段寄存器：由于实模式下的段寄存器与保护模式下的段寄存器不同，需要重新加载段寄存器，以确保正确访问内存。
- ⑤ 关闭 A20 地址线：如果在保护模式下打开了 A20 地址线，切换回实模式时可能需要关闭它，以限制对 1MB 内存空间的访问。

3. 解释不同权限代码的切换原理，call, jmp, retf 使用场景如何，能够互换吗？

**切换方法：**

**call 指令：**call 指令用于调用一个过程或函数，并将控制权转移到目标代码段中的指定地址。当使用 call 指令时，处理器会将当前代码段的返回地址（即下一条指令的地址）压入堆栈，并跳转到目标代码段中的指定地址。

**call 指令**可以在不同的权限级别之间进行切换，例如从用户态（Ring 3）切换到内核态（Ring 0）。

**jmp 指令：** jmp 指令用于无条件跳转到目标代码段中的指定地址。与 call 指令不同， jmp 指令不会将返回地址压入堆栈，因此无法直接实现权限级别的切换。但是，通过在目标代码段中设置适当的段选择子，可以实现从一个权限级别的代码段跳转到另一个权限级别的代码段。

**retf 指令：** retf 指令用于从过程或函数返回，并将控制权转移到调用者的代码段中的指定地址。 retf 指令会从堆栈中弹出返回地址，并跳转到该地址。与 call 指令类似， retf 指令可以在不同的权限级别之间进行切换，例如从内核态（Ring 0）返回到用户态（Ring 3）。

#### 使用场景：

**call 指令：** call 指令通常用于调用子程序或函数，将控制权转移到另一个代码段，并在返回时继续执行调用指令的下一条指令。它常用于实现函数调用、子程序调用和中断处理等场景。

**jmp 指令：** jmp 指令通常用于实现条件跳转或无条件跳转，将控制权直接转移到目标代码段中的指定地址。它常用于实现循环、条件语句和跳转表等场景。

**retf 指令：** retf 指令通常用于从过程或函数返回，并将控制权转移到调用者的代码段中的指定地址。它常用于实现函数返回、中断返回和任务切换等场景。

这些指令在某些情况下可以互换使用。例如，可以使用 Jmp 指令实现从一个权限级别的代码段跳转到另一个权限级别的代码段，然后使用 Retf 指令返回到原来的权限级别。

4. 自定义添加 1 个 GDT 代码段、1 个 LDT 代码段，GDT 段内要对一个内存数据结构写入一段字符串，然后 LDT 段内代码段功能为读取并打印该 GDT 的内容；

该问题参考 pmtest2.asm 和 pmtest3.asm 文件进行修改，拟打印的字符串为自己的名字，具体步骤如下：

- ① 修改[SECTION .data1]数据段中内容

将 StrTest 中的内容修改为 StrTest: db "Wang Zhuo", 0

- ② 编写 GDT 代码段

在 pmtest3.asm 的[SECTION .S32]代码段基础上， call DispReturn 和； Load LDT 之间加上 call TestRead 和 call TestWrite



```
call    DispReturn
```

```
call    TestRead  
call    TestWrite
```

```
; Load LDT
```

```
mov     eax, SelectTestLDT
```

TestRead 代码用来实现从内存中读取数据并显示在屏幕上，如下所示：

```
; -----  
TestRead:  
    xor     esi, esi  
    mov     ecx, 8  
.loop:  
    mov     al, [es:esi]  
    call    DispAL  
    inc     esi  
    loop    .loop  
  
    call    DispReturn  
  
    ret  
; TestRead 结束-----
```

TestWrite 代码用来实现将字符串数据写入内存，如下所示：

```
; -----  
TestWrite:  
    push    esi  
    push    edi  
    xor     esi, esi  
    xor     edi, edi  
    mov     esi, OffsetStrTest    ; 源数据偏移  
    cld  
.1:  
    lodsb  
    test    al, al  
    jz      .2  
    mov     [es:edi], al  
    inc     edi  
    jmp     .1  
.2:  
    pop     edi  
    pop     esi  
    ret  
; TestWrite 结束-----
```

③ 编写 LDT 代码段，用来读取 GDT 内容并打印

```

[SECTION .la]
ALIGN 32
[BITS 32]
LABEL_CODE_A:
    mov     ax, SelectorData
    mov     ds, ax           ; 数据段选择子
    mov     ax, SelectorVideo
    mov     gs, ax          ; 视频段选择子

    mov     ax, SelectorStack
    mov     ss, ax          ; 堆栈段选择子

    mov     esp, TopOfStack

    ; 下面显示一个字符串
    mov     ah, 0Ch          ; 0000: 黑底    1100: 红字
    xor     esi, esi
    xor     edi, edi
    mov     esi, OffsetStrTest ; 源数据偏移
    mov     edi, (80 * 12 + 0) * 2 ; 目的数据偏移。屏幕第 12 行, 第 0 列。
    cld

.1:
    lodsb
    test    al, al
    jz      .2
    mov     [gs:edi], ax
    add     edi, 2
    jmp     .1
.2:
    ; 显示完毕

    call DispReturn
    call TestRead

    ; 准备经由16位代码段跳回实模式
    jmp     SelectorCode16:0
CodeALen equ $ - LABEL_CODE_A
; END of [SECTION .la]

```

#### ④ 编译执行

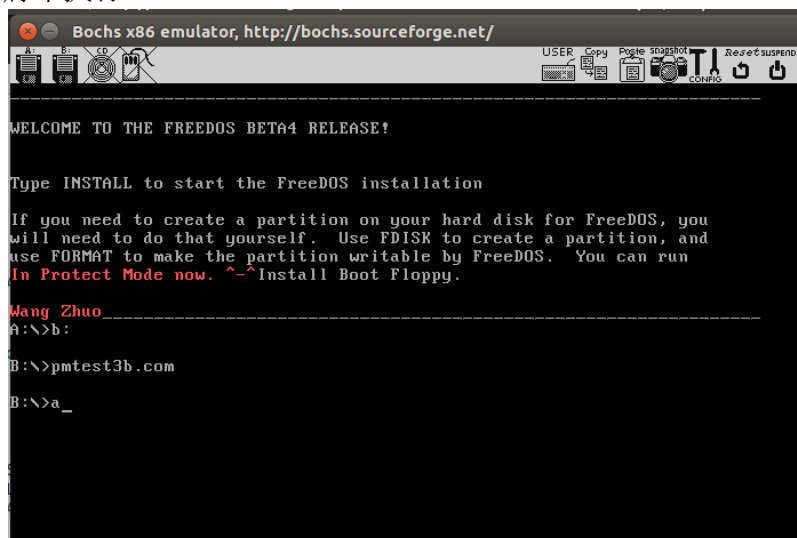


图 15 课后动手改 1

整个代码运行流程为:

1. 从实模式 jmp 到保护模式进入[SECTION .s32], 显示字符串 In Protect Mode now. ^-^;
2. [SECTION .s32]中通过 lldt 加载 LDT 的 ldt, 然后跳转到 [SECTION .la];
3. [SECTION .la]段显示字符串 StrTest 即 Wang Zhuo, 然后 jmp 回实

模式。

5. 自定义 2 个 GDT 代码段 A、B，分属于不同特权级，功能自定义，要求实现 A → B 的跳转，以及 B → A 的跳转。

① 步骤 1: 修改 data 数据段

修改 data 数据段的 DPL 为 3，以便后续添加的代码段可以访问，并添加后续代码段输出的字符串。

```
[SECTION .data1]      ; 数据段
ALIGN 32
[BITS 32]
LABEL_DATA:
SPValueInRealMode    dw 0
; 字符串
PMMessage:           db "In Protect Mode now. ^-^", 0 ; 进入保护模式后显示此字符串
Ring3Message:        db "to ring3!",0
Ring2Message:        db "to ring2!",0
Ring1Message:        db "to ring1!",0
OffsetRing3:         equ Ring3Message - $$
OffsetRing2:         equ Ring2Message - $$
OffsetRing1:         equ Ring1Message - $$
OffsetPMMessage      equ PMMessage - $$
StrTest:             db "ABCDEFGH IJKLMNOPQRSTUVWXYZ", 0
OffsetStrTest        equ StrTest - $$
DataLen              equ $ - LABEL_DATA
; END of [SECTION .data1]
```

② 步骤 2: 定义新增段的描述符和选择符，其中 ring3 与 ring1 的特权级一致

```
LABEL_DESC_CODE_RING3: Descriptor 0, SegCodeRing3Len-1, DA_C+DA_32+DA_DPL3
LABEL_DESC_CODE_RING2: Descriptor 0, SegCodeRing2Len-1, DA_C+DA_32+DA_DPL2
LABEL_DESC_CODE_RING1: Descriptor 0, SegCodeRing1Len-1, DA_C+DA_32+DA_DPL3
LABEL_DESC_DATA:        Descriptor 0, DataLen-1, DA_DRW+DA_DPL3 ;Data
LABEL_DESC_STACK:       Descriptor 0, TopOfStack, DA_DRWA+DA_32 ;Stack,32
LABEL_DESC_STACK3:      Descriptor 0, TopOfStack3, DA_DRWA+DA_32+DA_DPL3
LABEL_DESC_STACK2:      Descriptor 0, TopOfStack2, DA_DRWA+DA_32+DA_DPL2
LABEL_DESC_STACK1:      Descriptor 0, TopOfStack1, DA_DRWA+DA_32+DA_DPL3

SelectorCodeRing3      equ LABEL_DESC_CODE_RING3 - LABEL_GDT + SA_RPL3
SelectorCodeRing2      equ LABEL_DESC_CODE_RING2 - LABEL_GDT + SA_RPL2
SelectorCodeRing1      equ LABEL_DESC_CODE_RING1 - LABEL_GDT + SA_RPL3
SelectorData           equ LABEL_DESC_DATA - LABEL_GDT + SA_RPL3
SelectorStack          equ LABEL_DESC_STACK - LABEL_GDT
SelectorStack3         equ LABEL_DESC_STACK3 - LABEL_GDT + SA_RPL3
SelectorStack2         equ LABEL_DESC_STACK2 - LABEL_GDT + SA_RPL2
SelectorStack1         equ LABEL_DESC_STACK1 - LABEL_GDT + SA_RPL3
```

③ 步骤 3: 定义堆栈段 ring3, ring2, ring1

不同特权级别的任务通常需要自己的堆栈段，以确保堆栈数据的隔离。

```

; 堆栈段ring3
[SECTION .s3]
ALIGN 32
[BITS 32]
LABEL_STACK3:
    times 512 db 0
TopOfStack3 equ $ - LABEL_STACK3 - 1
; END of [SECTION .s3]

; 堆栈段ring2
[SECTION .s2]
ALIGN 32
[BITS 32]
LABEL_STACK2:
    times 512 db 0
TopOfStack2 equ $ - LABEL_STACK2 - 1
; END of [SECTION .s2]

; 堆栈段ring1
[SECTION .s1]
ALIGN 32
[BITS 32]
LABEL_STACK1:
    times 512 db 0
TopOfStack1 equ $ - LABEL_STACK1 - 1
; END of [SECTION .s1]

```

#### ④ 步骤 4: 初始化堆栈段描述符

```

xor eax, eax
mov ax, ds
shl eax, 4
add eax, LABEL_STACK1
mov word [LABEL_DESC_STACK1 + 2], ax
shr eax, 16
mov byte [LABEL_DESC_STACK1 + 4], al
mov byte [LABEL_DESC_STACK1 + 7], ah

xor eax, eax
mov ax, ds
shl eax, 4
add eax, LABEL_STACK2
mov word [LABEL_DESC_STACK2 + 2], ax
shr eax, 16
mov byte [LABEL_DESC_STACK2 + 4], al
mov byte [LABEL_DESC_STACK2 + 7], ah

; 初始化堆栈段描述符(ring3)
xor eax, eax
mov ax, ds
shl eax, 4
add eax, LABEL_STACK3
mov word [LABEL_DESC_STACK3 + 2], ax
shr eax, 16
mov byte [LABEL_DESC_STACK3 + 4], al
mov byte [LABEL_DESC_STACK3 + 7], ah

```

#### ⑤ 步骤 5: 修改 TSS

在 TSS 中加入三级堆栈的信息，为了提高任务切换的效率，可以在 TSS 中包括多个堆栈段的信息。这允许操作系统在切换任务时更快地加载相应的堆栈，而无需额外的堆栈切换操作。

### ⑥ 步骤 6: 修改调用门定义

添加调用门定义, 以实现从低特权级到高特权级的转换。

⑦ 步骤 7: 编写代码段 ring3, ring2, ring1, 并完成 ring3->ring2->ring1 的跳转

- 输出 to ring3!与字符 3
- 使用 call 指令调用一个称为 SelectorCallGateTest1 的调用门，跳转到 ring2

ring2:

- 输出 to ring2!与字符 2
- 使用 retf 方法跳转到 ring1

```

; CodeRing2
[SECTION .ring2]
ALIGN 32
[BITS 32]
LABEL_CODE_RING2:
    mov ax, SelectorVideo
    mov gs, ax

    mov ax, SelectorData
    mov ds, ax
    mov ah, 0Ch
    xor esi, esi
    xor edi, edi
    mov esi, OffsetRing2 ; 源数据偏移
    mov edi, (80 * 15 + 5) * 2
    cld

.1:
    lodsb
    test al, al
    jz .2
    mov [gs:edi], ax
    add edi, 2
    jmp .1
.2: ; 显示完毕
    mov edi, (80 * 15 + 0) * 2
    mov ah, 0Ch ; 0000: 黑底 1100: 红字
    mov al, '2'
    mov [gs:edi], ax
    push SelectorStack1
    push TopOfStack1
    push SelectorCodeRing1
    push 0
    retf
; call SelectorCallGateTest:0
SegCodeRing2Len equ $ - LABEL_CODE_RING2
; END of [SECTION .ring2]

```

ring1:

- 输出 to ring1!与字符 1
- 使用 call 指令调用一个称为 SelectorCallGateTest 的调用门,准备经由 16 位代码段跳回实模式,完成后续代码逻辑

```

; CodeRing1
[SECTION .ring1]
ALIGN 32
[BITS 32]
LABEL_CODE_RING1:
    mov ax, SelectorVideo
    mov gs, ax

    mov ax, SelectorData
    mov ds, ax
    mov ah, 0Ch
    xor esi, esi
    xor edi, edi
    mov esi, OffsetRing1 ; 源数据偏移
    mov edi, (80 * 16 + 5) * 2
    cld

.1:
    lodsb
    test al, al
    jz .2
    mov [gs:edi], ax
    add edi, 2
    jmp .1
.2: ; 显示完毕
    mov edi, (80 * 16 + 0) * 2
    mov ah, 0Ch ; 0000: 黑底 1100: 红字
    mov al, '1'
    mov [gs:edi], ax
    call SelectorCallGateTest:0
SegCodeRing1Len equ $ - LABEL_CODE_RING1
; END of [SECTION .ring1]

```

运行结果如图,实验成功:

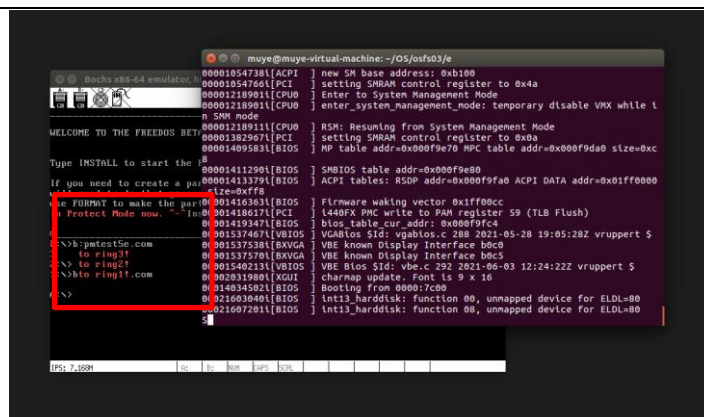


图 16 课后动手改 2

### 三、实验过程分析

(实验分工, 详细记录实验过程中发生的故障和问题, 进行故障分析, 说明故障排除的过程及方法。根据具体实验, 记录、整理相应的数据表格等)

#### 实验问题与故障分析:

##### 1. 没有完全理解不同特权级之间跳转的过程, 导致代码中止运行

具体来说, 我没有正确理解如何利用调用门进行低特权级到高特权级的转换, 因此无法实现三个代码段之间的跳转。我尝试使用直接的 `call` 指令进行跳转, 但结果却是跳转失败。这个错误和程子洋同学出现的情况一致, 即我只能出现“A This is A”。

为了解决这个问题, 我进一步学习如何使用调用门和 `call` 指令进行特权级之间的跳转。经过仔细研究并和程子洋同学交流, 修改代码, 我成功地完成了实验, 并实现了代码段之间的跳转。

##### 2. 将 `pmtest1.com` 复制到虚拟软驱 `pm.img` 报错: `mount point /mnt/floppy does not exist`

```
mount: mount point /mnt/floppy does not exist
```

图 17 报错:不存在文件

原因在于在 `/mnt` 目录下不存在 `floppy` 文件, 只需要指令 `mkdir` 创建即可, 但要进入管理员模式才能在 `/mnt` 目录下建立文件夹; 输入指令 `sudo` 进入超级管理员模式输入密码, 即可完成创建

##### 3. 未理解不同特权级之间跳转过程, 造成代码中止

由于未理解好如何利用调用门进行低特权级到高特权级的转换, 造成程序未实现三个代码段之间的跳转, 即使用直接 `call` 的方式, 造成跳转失败。在理解好使用调用门+`call` 指令进行跳转后, 修改代码, 完成了实验。





图 18 课后动手改 2 错误结果

#### 4. 数据段修改错误(课后动手改 1)

由于疏忽,新增的输出字符串未正确写入数据段,造成输出失败,始终无法输出既定的字符串 HELLO。发现新增的输出字符串越出数据段,及时修改后,成功输出。

```
[SECTION .data1]      ; 数据段
ALIGN 32
[BITS 32]
LABEL_DATA:
SPValueInRealMode    dw 0
; 字符串
PMMessage:           db "In Protect Mode now. ^-^", 0 ; 进入保护模式后显示此字符串
OffsetPMMessage       equ PMMessage - $$
StrTest:              db "ABCDEFGHJKLMNOPQRSTUVWXYZ", 0
OffsetStrTest         equ StrTest - $$
DataLen              equ $ - LABEL_DATA
StrWrite:             db "HELLO",0
OffsetStrWrite        equ StrWrite - $$
; END of [SECTION .data1]
```

图 19 新增字符串越出数据段

```
[SECTION .data1]      ; 数据段
ALIGN 32
[BITS 32]
LABEL_DATA:
SPValueInRealMode    dw 0
; 字符串
PMMessage:           db "In Protect Mode now. ^-^", 0 ; 进入保护模式后显示此字符串
OffsetPMMessage       equ PMMessage - $$
StrTest:              db "ABCDEFGHJKLMNOPQRSTUVWXYZ", 0
OffsetStrTest         equ StrTest - $$
StrWrite:             db "HELLO",0
OffsetStrWrite        equ StrWrite - $$
DataLen              equ $ - LABEL_DATA
; END of [SECTION .data1]
```

图 20 新增字符串越出数据段(修改)

5. 编译发现报错 no bootable device, 查找资料找到的类似错误的解决方案无效。

```

000010546731[ACPI] ] new SM base address: 0xb100
000010547011[PCI] ] setting SHRAM control register to 0x4a
000012180361[CPU0] ] Enter to System Management Mode
000012188471[CPU0] ] RSM: Resuming from System Management Mode
000013029031[PCI] ] setting SHRAM control register to 0x0a
000014095191[BIOS] ] MP table addr=0x000f9e70 MPC table addr=0x000f9da0 size=0xc8
000014112261[BIOS] ] SMBIOS table addr=0x000f9e00
000014133151[BIOS] ] ACPI tables: RSDP addr=0x000f9fa0 ACPI DATA addr=0x01ff0000 size=0xff8
000014162991[BIOS] ] Firmware waking vector 0x1ff0cc
000014185531[PCI] ] i440FX PMC write to PAM register 59 (TLB Flush)
000014192831[BIOS] ] bios table_cur_addr: 0x000f9fc4
000015469011[BIOS] ] VgABios Sid: vgabios.c.v 1.75 2011/10/15 14:07:21 vrupper Exp $
000015469721[BXVGA] ] VBE known Display Interface b0c0
000015470041[BXVGA] ] VBE known Display Interface b0c5
000015499291[BIOS] ] VBE Bios Sid: vbe.c.v 1.64 2011/07/19 18:25:05 vrupper Exp $
000139930401[FLOPPY] attempt to read/write sector 1 with media not present
000219763711[FLOPPY] controller reset in software
00022030188p[BIOS] ] >>PANIC<< No bootable device.
=====
bochs is exiting with the following message:
[BIOS] ] No bootable device.
=====
000220301881[CPU0] ] CPU is in real mode (active)
000220301881[CPU0] ] CS.mode = 16 bit
000220301881[CPU0] ] SS.mode = 16 bit
000220301881[CPU0] ] EFER = 0x00000000
000220301881[CPU0] ] EAX=0000040a EBX=0000cd24 ECX=00090004 EDX=00000402
000220301881[CPU0] ] ESP=0000ffa8 EBP=0000ffac ESI=000e0000 EDI=0000ffac
000220301881[CPU0] ] IORPL=0 id vip vrf ac vm rft nt of df lf tf sf ZF af PF cf
000220301881[CPU0] ] SEG sltr(index|tl|rl) base limit G D
000220301881[CPU0] ] CS:000( 0004) 0| 0| 000f0000 0000ffff 0 0
000220301881[CPU0] ] DS:000( 0005) 0| 0| 000f0000 0000ffff 0 0
000220301881[CPU0] ] SS:000( 0005) 0| 0| 00000000 0000ffff 0 0
000220301881[CPU0] ] ES:07c0( 0005) 0| 0| 00007c00 0000ffff 0 0
000220301881[CPU0] ] FS:0000( 0005) 0| 0| 00000000 0000ffff 0 0
000220301881[CPU0] ] GS:0000( 0005) 0| 0| 00000000 0000ffff 0 0
000220301881[CPU0] ] EIP=0000054a (0000054a)
000220301881[CPU0] ] CR0=0x00000010 CR2=0x00000000
000220301881[CPU0] ] CR3=0x00000000 CR4=0x00000000
(0).[22030188] [0x0000000f054a] f000:054a (unk. ctxt): out dx, al ; ee
000220301881[CMOS] ] Last time is 1697441520 (Mon Oct 16 15:32:00 2023)

```

经过检查发现为 bochsrc 文件中 b 盘写不规范。

```

bochsrc (-/Desktop/chapter3/a) - gedit
Open Save
#####
# Configuration file for Bochs
#####
# how much memory the emulated machine will have
megs: 32
# filename of ROM images
romimage: file=/home/shizi/Desktop/bochs-2.7/bios/BIOS-bochs-latest
vgaromimage: file=/usr/share/vgabios/vgabios.bin
# what disk images will be used
floppya: 1_44=freedos.img, status=inserted
floppyb: 1_44=pm.img, status=inserted
# choose the boot disk.
boot: a
# where do we send log messages?
# log: bochsout.txt
# disable the mouse
mouse: enabled=0
# enable key mapping, using US layout as default.
# keyboard: enabled=1, map=/usr/share/bochs/keymaps/x11-pc-us.map

```

#### 四、实验结果总结

（对实验结果进行分析，完成思考题目，并提出实验的改进意见）

刘琥：

课后动手改部分指引不清晰，如果能过教学示例，学生自己操作时可以更快上手

程子洋：

1. 错误排除指南：提供学生在遇到常见问题时进行自我排除的指南。这可以包括常见错误消息的解释以及如何解决这些问题的步骤。
2. 实验内容：可以增设更多关于调用门的有关实验，帮助学生更清晰认识调用门；提供更多示例，体现多个特权级之间转换的过程，促进学生理解与掌握。

王卓：

1. 提供更详细的实验指导：在每个实验步骤中，提供更详细的指导和说明，比如预期结果等，这样能帮助我们更好地完成实验；
2. 介绍实验目的和背景：实验开始之前提供实验的目的和背景，解释一下为什么需要进行该实验以及其与操作系统的关系，这能帮助我们更好理解实验的意义和重要性。

聂森：

1. 可以先让同学阅读教材资料后再介绍实验内容和实验相关的知识，可以帮助同学更快更好的了解实验的目标和实验的原理及任务
2. 给出具体的实验指导或给出更加丰富的实验参考资料，可以帮助同学更快上手实验并在有问题时找到解决方式

#### 五、各人实验贡献与体会（每人各自撰写）

同学：刘琥

此次实验为本人独立完成大部分实验内容，并负责实验题 1，2 和自己部分的实验报告。

这次实验是我对保护模式有了更加清晰的了解，在调试 `pmtest2.asm` 和 `pmtest3.asm` 两个 `nasm` 汇编文件的过程中我复习了大一下所学习的汇编语言的相关知识；在整理的过程中，总结了从实模式到保护模式的关键步骤，并对其中的一些操作有了更深刻的理解，如为什么要关中断，为什么要打开 A20 地址线，又如从保护模式切换回实模式需要哪些步骤等等；在与小组成员交流讨论的过程中，我对特权级转换、堆栈切换也有了更深刻的理解，也增强了我与小组成员沟通合作的能力。

同学：程子洋

此次实验为本人独立完成大部分实验内容，并主要负责实验题 5，6，课后动手改 2，并参与实验报告的撰写。

实验体会：

1. 从实模式到保护模式的转变：我学会了如何从实模式切换到保护模式。这个过程是理解操作系统如何工作的基础，因为它涉及到对内存和权限的灵活管理。
2. 特权级别的作用：通过实验，我更清楚地了解了特权级别的概念。它们是计算机系统中的关键要素，决定了哪些代码可以访问哪些资源，因此对于操作系统和软件开发非常重要。
3. 调用门的奥秘：我学会了如何使用调用门来实现不同特权级别之间的通信。这是一项关键技能，实现了低特权级到高特权级的高效转换。
4. 错误排除能力：当遇到问题时，我必须花费很多的时间去 debug，这方面能力的培养非常重要，对未来的工程开发有非常大的帮助。

复习与巩固：这次实验让我学习了很多关于操作系统的理论知识，也回顾了前面 `bochs` 相关的内容，锻炼了我通过查找资料和教材解决问题的能力，也提高了我和同学间互相交流协作的能力。

同学：王卓

此次实验为本人独立完成大部分实验内容，并主要针对实验（二）的 2,3 以及课后动手改 1 进行对应部分内容的实验报告的撰写。这次实验给我的体会如下：

1. 深入理解保护模式：通过实验，我深入理解了保护模式的概念和工作原理。我学会了如何构造和加载 GDT 表、LDT 表，以及如何进行特权级的切换和段间的切换。这使我对操作系统的内存管理和权限控制有了更深入的理解。

<p>2. 团队合作与交流：在实验过程中，我与同学们进行了讨论和交流，分享了彼此的经验和问题。这促进了团队合作和互助，使我们能够共同解决实验中遇到的困难和挑战。这种合作与交流的经验对我今后的团队合作能力和沟通能力有着积极的影响。</p> <p>实验的挑战与收获：实验内容相对复杂，需要深入理解和掌握多个概念和技术。在实验过程中，我遇到了一些困难和挑战，但通过不断的学习和尝试，我成功地完成了实验，并从中获得了很多收获和成就感。</p> <p>同学：聂森</p> <p>此次实验为本人独立完成大部分实验内容，并主要负责实验题 3，4 和全部实验报告的撰写、整合。</p> <p>实验体会：</p> <p>本次实验的内容量大，需要掌握的知识比较多。在进行实验时，我详细阅读了教材的相关章节并在互联网上搜集信息以了解相关知识：保护模式、实模式、GDT 与 LDT、特权切换。在下次实验时，可以提高效率，在上实验课之前就阅读相关资料以提高课堂效率。在实验时，由于之前的汇编知识有所遗忘，所以不得不再次复习之前的汇编语言，本次实验帮助我捡起来了之前遗忘的知识并对现在所学的知识进行的实操练习和巩固，而实验中出现的問題和报错也进一步督促我更加深入的了解相关知识点。同时，在与小组成员一起讨论問題如何解決的过程中也增加了团队合作能力和問題分析解決能力。</p>		
六、教师评语		
教师评分（请填写好姓名、学号）		
姓名	学号	分数
聂森	2021302191536	
程子洋	2021301051114	
王卓	2021302191791	
刘琥	2021302121234	
教师签名：		
年 月 日		