

Report for Lab04-从实模式进入保护模式

Stu : 金泽文

No.PB15111604

实验目的：

学习并掌握 bios 中断机制。

学习并理解 i386 保护模式以及对应的 gdt，寻址方法，掌握由实模式切换到保护模式的方法。

实验内容：

1. 对实验 3 中的汇编程序进行增量编程，仍用作启动扇区代码
2. 链接脚本不变，shell 脚本根据需要修改，仍制作启动软盘，将启动扇区代码安装到软盘的启动扇区(也可以选择其他介质)
3. 启动运行启动扇区代码，最后进入死循环

相关原理学习：

1. bios 中断的学习。

参考 [BIOS 中断-wiki](#)，了解 BIOS 中 int10h，int16h，并参考 [BIOS 下的汇编-fancymore](#)，得到以下信息：

① 对于 INT 10H：

- i. 对于表示“显示模式”的 AH=00 功能号，AL 为设置的模式，由于我们要求 80*25，故 AL 应置为 02。
- ii. 对于表示“写字符串”的 AH=13h 功能号，BL 表示显示字符的属性，

如颜色等。ES : BP 指向字符串的地址。

② 对于 INT 16H :

- i. 对于表示 “等待按键” 的 AH=10h 功能号，AL 得到输入的字符。可以通过比较 AL 是否为 0，来判断是否有任意按键，如果等于 0，则一直循环，这一模式也称为 “轮询”。当然，如果不想等待，还有 AH=11h 这个选项，但是在这里我们只需要使用 AH=10h。

2. i386 保护模式的学习。

·从[保护模式-wiki](#)得到的信息：

保护模式 (Protected Mode)，是目前大部分 x86 操作系统所使用的模式。保护模式下，存储器得到保护，可以避免问题程序破坏其他任务或 OS 核心存储器。I386 保护模式地址总线为 32 位，寻址空间 32GB。

·从《Orange' s：一个操作系统的实现》第 10 章得到的信息：

保护模式下，虽然段值仍然由原来 16 位的 cs、ds 寄存器表示，但此时它们仅仅变成了一个索引，这个索引指向一个数据结构的一个表项，表项中详细定义了段的起始地址、界限、属性等内容。这个数据结构就是 GDT (或 LDT)。GDT 的一个表项叫做描述符(Descriptor)。GDT 的作用是提供段式存储机制，这种机制由段寄存器和描述符共同提供。

要将实模式的逻辑地址转化为之后的线性地址，需要准备 GDT，并用 lgdt 加载 gdt。之后，由于历史原因，8086 的设计者为了不让寻址超过 1MB 出现异常，所以采用了回卷的策略。而 i386 保护模式下，需要避免这一策略带来的问题。所以需要关闭这一策略，这一方法就是打开 A20 地址线。

但是在这些前去准备之后，通过什么手段转入到保护模式呢？

要知道，控制寄存器 CRO 的 PE 位即为保护模式与实模式的标志位。0 表示实模式，1 表示保护模式。只需要将其置为 1，就会进入保护模式。不过，还需要将代码段的 selector 装入 cs 寄存器中。

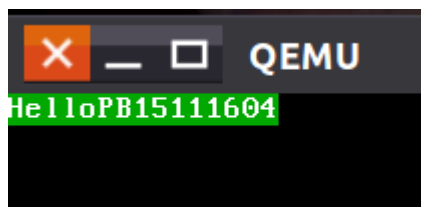
还需要注意什么呢？

关中断。由于保护模式下，中断处理的机制不同于实模式，如果不关中断，就会出现错误，所以保护模式下的关中断比实模式下更具有必要性。而我们之前代码的第一步即为 cli，所以这一步无需设置。

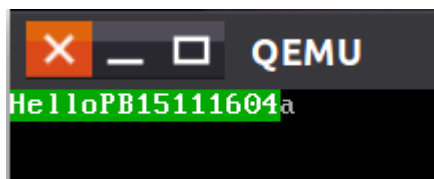
实验过程（以修改代码为主）：

1.等待输入。由“相关原理学习”部分，得到如下修改：

```
40     inc %cx
41     cmp $15, %cx           #比较
42     jz polling
43     jmp print
44
45 polling:                  #轮询
46     movb $0x10, %ah
47     int $0x16
48     cmp $0, %al
49     jz polling
50     movb $'a', %es:(%di)   #测试
51
```

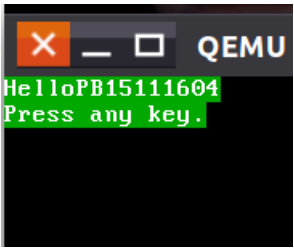


并且输入之后



测试成功。

2.实模式下的换行。只需要在输出下一行字符串前将%di 设置为下一行开头即可。由于代码很短，所以可以手动设置%di 为 160,320 等。如右所示。



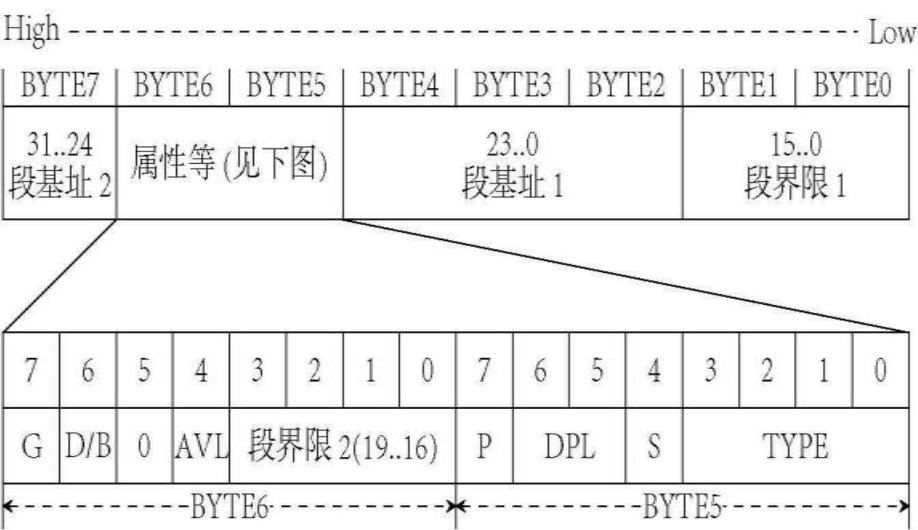
3.进入保护模式。

这一部分是本实验的难点。由“相关原理学习”部分知，要进入保护模式，需要以下五步：

准备 GDT，用 lgdt 加载 gdtr，打开 A20，置 cr0 的 PE 位，跳转，进入保护模式。

i.准备 GDT：准备 GDT，即准备描述符。我们只需要准备数据段和代码段描述符。

描述符格式如图。



- 两个段界限一起表示 segment 最大长度，一共 20 位，如果下面的 G 为 1，则将其长度左移 12 位，否则不左移。
- G 位表示段界限粒度，0 表示字节，1 表示 4KB，我们置为 1。
- D/B 较复杂。Code segment Descriptor 中，1 表示指令默认使用 32 位地

址，0 表示 16 位；Data segment Descriptor 中，1 表示段的上界为 4GB，0 表示 64KB。

- AVL 为保留位。

- 第 S 位 “1” 表示该段在内存中存在。

- DPL 描述特权级，我们置为 0。

- S 位描述数据段/代码段还是系统段/门，我们置为 1。

- TYPE 描述读写执行的类型，我们将 ds 对应的设置为 2，cs 对应的设置为 8，分别表示读/写，执行/写。

在实现中，我选择将 gdt 内容存储在绝对地址 0x0 处。并根据以上格式，得到如下代码：

```
82
83 switch:
84 movl $0x00000000, 0x000 # 第0个descriptor保留
85 movl $0x00000000, 0x004
86 movl $0x0000FFFF, 0x008 # Data segment descriptor
87 movl $0x00CF9200, 0x00C # 读/写
88 movl $0x0000FFFF, 0x010 # Code segment descriptor
89 movl $0x00CF9800, 0x014 # 执行/读
90
```

ii.加载 gdtr

这一步得到如下代码：

```
90
91 lgdt gdt_reg
92
93
94 gdt_reg:
95 .word 0x0800
96 .long 0x00000000
97
```

我将基地址设置为 0x0，同时将 limit 设置为 0x800。

iii.打开 A20

得到如下代码：

```

91
92 A20:
93     in $0x92, %al
94     or $2, %al
95     out %al, $0x92
96

```

iv.CR0

得到如下代码：

```

97 cr0:
98     movl %cr0, %eax
99     or $0x01, %al
100     movl %eax, %cr0
101

```

v.跳转

得到如下代码：

```

101
102     jmp $0x10, $pmode # 跳转到相对于cs描述符
103

```

就此，进入保护模式部分的代码完成。为了检验，得到：

```

109 pmode:
110     movw $0x8, %ax # 让%ds指向Data segment
111     movw %ax, %ds
112     movl $0xb8000, %esi
113     movw $0x2f42, %ds:(%esi) # 检查是否成功
114

```

模拟之后得到如下结果：

A screenshot of a QEMU terminal window. The title bar shows 'QEMU' with standard window controls. The terminal output is 'HelloPB15111604' followed by 'Press any key.' on the next line. The text is green on a black background.

等待输入：

A screenshot of a QEMU terminal window after input. The title bar shows 'QEMU'. The terminal output is 'HelloPB15111604', 'Press any key.a' (where 'a' is the input), and 'Start to switch...' on the next line. The text is green on a black background.

输入之后：，可以注意到第一个字符变为了 'B'，检查通

过。（右边的 'a' 是上面为了检查等待过程。）

4.保护模式下的输出

保护模式下的寻址，不同于实模式，不再是 段基址：段内偏移，而是 selector：段内

偏移。回顾上面检验切换到保护模式时输出字符 'B' 的代码

```
109 pmode:
110     movw $0x8, %ax # 让%ds指向Data segment
111     movw %ax, %ds
112     movl $0xb8000, %esi
113     movw $0x2f42, %ds:(%esi) # 检查是否成功
114
```

由于设置的 ds 的基地址为 0x0，故将 esi 置为 0xb8000，即可直接通过 %ds:(%esi)

得到 vga 显存地址。下面通过这个原理，得到输出字符串的代码：

```
108     .code32 # This part is compiled in 32 bits mode
109 pmode:
110     xorl %eax, %eax
111     movw $0x8, %ax # 让%ds指向Data segment
112     movw %ax, %ds
113     movw %ax, %es
114     movw %di, %ax
115     movl $0xb8000, %edi
116     addl %eax, %edi
117     #movw $0x2f42, %es:(%edi) # 检查是否成功
118     #jmp .
119
120 init_ok:
121     leal ok, %esi # esi存hello偏移地址
122     mov $0, %cx
123
124 print_ok:
125     cld # 设置movsb中di, si自动变化方向
126     movsb # 从ds:si传字符到es:di
127     movb $0x2f, %es:(%edi) # 设置背景、字符颜色
128     inc %edi
129     inc %cx
130     cmp $3, %cx # 比较
131     jz init_done
132     jmp print_ok
133
134 init_done:
135     leal done, %esi
136     mov $0, %cx
137     movl $0xb8000, %edi
138     addl $480, %edi
139
140 print_done:
141     movw $0x2f44, %es:(%edi) # 检查是否成功
142     cld # 设置movsb中di, si自动变化方向
143     movsb # 从ds:si传字符到es:di
144     movb $0x2f, %es:(%edi) # 设置背景、字符颜色
145     inc %edi
146     inc %cx
147     cmp $30, %cx # 比较
148     jz idle
149     jmp print_done
150
151 idle:
152     jmp .
153
```

要求的内容：

1. 如何在实模式下利用 BIOS 实现任意键的获取？

利用 BIOS 16h 中断。对于表示“等待按键”的 AH=10h 功能号，AL 得到输入的字符。可以通过比较 AL 是否为 0，来判断是否有任意按键，如果等于 0，则一直循环，这一模式也称为“轮询”。当然，如果不想等待，还有 AH=11h 这个选项，但是在这里我们使用 AH=10h。

2. 什么是保护模式？要进入保护模式需要进行哪些准备？

对第一个问题，【相关原理学习】中的保护模式部分有详细的解答。

对第二个问题，【实验过程】中第 3 点“**进入保护模式**”部分有详细的解答。

3. 【附加】如何在实模式下利用 BIOS 实现 VGA 显示？能不能在保护模式下利用 BIOS 实现 IO？能或者不能都说出你的理由。

实模式下，可以利用 bios 中断机制，通过 int 10h 等，实现 vga 的显示，根据输出的需求，设置功能号 ah 的值，比如 ah=0x13 表示写字符串，ah=0x0e 表示写字符等。

而在保护模式下，**严格**来说是可以通过 bios 中断实现的。正常情况下的保护模式虽然不能利用 bios，因为 bios 中断所使用的是 16 位实模式环境。但是在保护模式下，可以利用“**虚拟 8086 模式**”来模拟一个虚拟的较为逼真的实模式环境。在这个虚拟 8086 模式中，可以通过 bios 中断来实现 IO。虚拟 8086 模式作为任务运行在保护模式下，所以虚拟 8086 模式下的 bios 中断实现，当然可以看做是保护模式下的实现。

4. 保护模式下写 VGA 缓存和实模式下写 VGA 缓存有什么不一样？

寻址方式会略有不同。比如，虽然我用的都是 movsb 这个伪指令，但是在实模式下，这个指令所存取的 %ds:(%si)和%es:(%di)都是实模式的寻址方式。ds，es 分别代表的就是段基址。而在保护模式下，%ds:(%esi)和%es:(%edi)都是保护模式的寻址方式，ds，es 分别代表的是 selector，也就是选择子，是通过 gdt 下的描述符的 index 实现的，而不是实模式下左移 4 位实现的。

5. 你是如何实现回车功能的？

实模式下，只需要在输出下一行字符串前将%di 设置为下一行开头即可。由于代码很短，所以可以手动设置%di 为 160,320 等。

6. 给出代码运行关键时刻的截屏并加以说明

运行脚本：

```
→ Documents ./boot1.sh
gcc -c final.s -o t2.o -m32

ld -Tt11.ld -o t1.elf t2.o

objcopy -O binary t1.elf t1.bin

记录了2880+0 的读入
记录了2880+0 的写出
1474560 bytes (1.5 MB, 1.4 MiB) copied, 0.0686617 s, 21.5 MB/s
dd if=/dev/zero of=a.img bs=512 count=2880

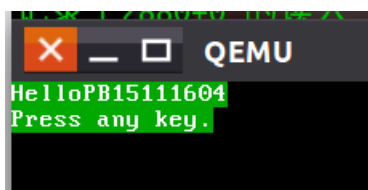
sudo losetup /dev/loop4 a.img

记录了1+0 的读入
记录了1+0 的写出
512 bytes copied, 0.0318831 s, 16.1 kB/s
sudo dd if=t1.bin of=/dev/loop4 bs=512 count=1

qemu -fda a.img
sudo losetup -d /dev/loop4

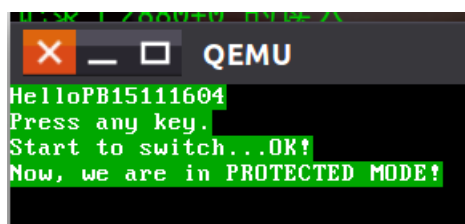
→ Documents WARNING: Image format was not specified for 'a.img' and pr
ssed raw.
Automatically detecting the format is dangerous for raw images
perations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions
→ Documents █
```

等待输入：



A screenshot of a QEMU window. The title bar says 'QEMU'. The main display area shows the text 'HelloPB15111604' on the first line and 'Press any key.' on the second line. The text is in a green monospaced font on a black background.

进入 pmode：



A screenshot of a QEMU window. The title bar says 'QEMU'. The main display area shows the text 'HelloPB15111604' on the first line, 'Press any key.' on the second line, 'Start to switch...OK!' on the third line, and 'Now, we are in PROTECTED MODE!' on the fourth line. The text is in a green monospaced font on a black background.

7. 给出你所有代码的流程图

