



本科生实验报告

实验课程：_____操作系统_____

实验名称：_____保护模式_____

专业名称：_____计算机科学与技术_____

学生姓名：_____凌国明_____

学生学号：_____21307077_____

实验地点：_____教室_____

实验成绩：_____

报告时间：_____2023. 03. 22_____

1. 实验要求

- 学习到如何从 16 位的实模式跳转到 32 位的保护模式
- 在平坦模式下运行 32 位程序
- 学习如何使用 I/O 端口和硬件交互。

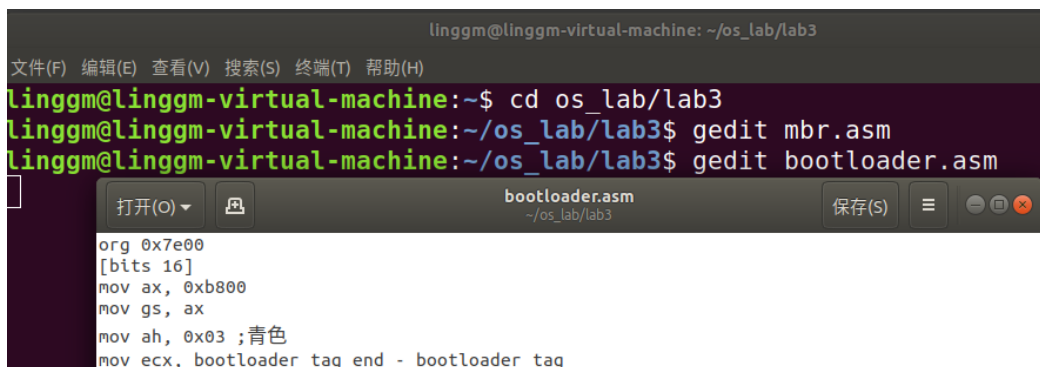
实验任务

- 复现 Example 1，说说你是怎么做的并提供结果截图
- 在 Example 1 中将 LBA28 读取硬盘的方式换成 CHS 读取，同时给出逻辑扇区号向 CHS 的转换公式。（利用 int13h 中断）
- 复现 Example 2，使用 gdb 或其他 debug 工具在进入保护模式的 4 个重要步骤上设置断点，并结合代码、寄存器的内容等来分析这 4 个步骤，最后附上结果截图。gdb 的使用可以参考 appendix 的“debug with gdb and qemu”部份。
- 改造“Lab2-Assignment 4”为 32 位代码，即在保护模式后执行自定义的汇编程序

2. 实验过程

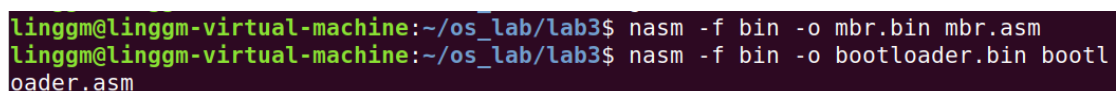
1) 复现 example1

- 第一步：编写 mbr 和 bootloader，mbr 的任务是加载 bootloader，bootloader 的任务是打印字符（代码详见代码展示部分）



```
linggm@linggm-virtual-machine: ~/os_lab/lab3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
linggm@linggm-virtual-machine:~$ cd os_lab/lab3
linggm@linggm-virtual-machine:~/os_lab/lab3$ gedit mbr.asm
linggm@linggm-virtual-machine:~/os_lab/lab3$ gedit bootloader.asm
bootloader.asm
~/os_lab/lab3
org 0x7e00
[bits 16]
mov ax, 0xb800
mov gs, ax
mov ah, 0x03 ;青色
mov ecx, bootloader_tag_end - bootloader_tag
```

- 第二步，编成 bin 格式文件



```
linggm@linggm-virtual-machine:~/os_lab/lab3$ nasm -f bin -o mbr.bin mbr.asm
linggm@linggm-virtual-machine:~/os_lab/lab3$ nasm -f bin -o bootloader.bin bootl
oader.asm
```

- 第三步：创建磁盘

```
linggm@linggm-virtual-machine:~/os_lab/lab3$ qemu-img create hd.img 10m
Formatting 'hd.img', fmt=raw size=10485760
linggm@linggm-virtual-machine:~/os_lab/lab3$ ls
bootloader.asm  bootloader.bin  hd.img  mbr.asm  mbr.bin
```

- 第四步：启动 qemu 测试

```
linggm@linggm-virtual-machine:~/os_lab/lab3$ qemu-system-i386 -hda hd.img -serial
l null -parallel stdio
WARNING: Image is not a raw disk image.
Automatically converted to raw format.
run bootloader on 1.10.2-ubuntu1
```

成功输出，复现完成

2) CHS 读磁盘

- 第一步：改写 mbr 中的 asm_read_hard_disk 函数

```
asm_read_hard_disk:
    mov ch, 0 ; 0==柱面=lba/(63*18)
    mov dh, 0 ; 0==磁头= (lba/63)%18
    mov dl, 80h

    mov cl, al ; lba
    inc cl ; 扇区=lba+1

    mov ah, 02h
    mov al, 1
    int 31h
    add bx, 512 ; 缓冲区首地址+=512
    ret
```

- 第二步：与复现 example1 相同

```
linggm@linggm-virtual-machine:~/os_lab/lab3$ qemu-system-i386 -hda hd.img -serial
l null -parallel stdio
WARNING: Image is not a raw disk image.
Automatically converted to raw format.
run bootloader on 1.10.2-ubuntu1
```

$C = LBA // (\text{每柱面磁道数} * \text{每磁道扇区数})$

$H = (LBA // 63) \% \text{每柱面磁道数}$

$S = (LBA \% \text{每磁道扇区数}) + 1$

C/H/S地址			LBA编号
柱面	磁头	扇区	
0	0	1	0
0	0	2	1
0	0	3~63	2~62

3) 利用 gdb 进行 example2 的 debug 分析

- 第一步：编译 nasm，生成可重定位文件 mbr.o，-g 是添加 debug 信息

```
linggm@linggm-virtual-machine:~/os_lab/lab3/assignment3$ nasm -o mbr.o -g -f elf32 mbr.asm
```

- 第二步：为可重定位文件 mbr.o 指定起始地址 0x7c00，链接生成可执行文件 mbr.symbol 和 mbr.bin

```
linggm@linggm-virtual-machine:~/os_lab/lab3/assignment3$ ld -o mbr.symbol -melf_i386 -N mbr.o -Ttext 0x7c00
ld: 警告：无法找到项目符号 _start; 缺省为 00000000000007c00
linggm@linggm-virtual-machine:~/os_lab/lab3/assignment3$ ld -o mbr.bin -melf_i386 -N mbr.o -Ttext 0x7c00 --oformat binary
ld: 警告：无法找到项目符号 _start; 缺省为 00000000000007c00
```

- 第三步：对 bootloader 做前两步操作

```
linggm@linggm-virtual-machine:~/os_lab/lab3/assignment3$ nasm -o bootloader.o -g -f elf32 bootloader.asm
linggm@linggm-virtual-machine:~/os_lab/lab3/assignment3$ ld -o bootloader.symbol -melf_i386 -N bootloader.o -Ttext 0x7e00
ld: 警告：无法找到项目符号 _start; 缺省为 00000000000007e00
linggm@linggm-virtual-machine:~/os_lab/lab3/assignment3$ ld -o bootloader.bin -melf_i386 -N bootloader.o -Ttext 0x7e00 --oformat binary
ld: 警告：无法找到项目符号 _start; 缺省为 00000000000007e00
```

- 第四步：分别将 mbr.bin 和 bootloader.bin 写入磁盘

```
linggm@linggm-virtual-machine:~/os_lab/lab3/assignment3$ dd if=mbr.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
记录了1+0 的读入
记录了1+0 的写出
512 bytes copied, 0.00011773 s, 4.3 MB/s
linggm@linggm-virtual-machine:~/os_lab/lab3/assignment3$ dd if=bootloader.bin of=hd.img bs=512 count=5 seek=1 conv=notrunc
记录了1+1 的读入
记录了1+1 的写出
534 bytes copied, 0.00013391 s, 4.0 MB/s
```

- 第五步：启动 qemu，连接 gdb，加载符号表

```
linggm@linggm-virtual-machine:~/os_lab/lab3/assignment3$ qemu-system-i386 -s -S -hda hd.img -serial null -parallel stdio
```

```
(gdb) add-symbol-file mbr.symbol 0x7c00
add symbol table from file "mbr.symbol" at
       .text_addr = 0x7c00
(y or n) y
Reading symbols from mbr.symbol...done.
(gdb) add-symbol-file bootloader.symbol 0x7e00
add symbol table from file "bootloader.symbol" at
       .text_addr = 0x7e00
(y or n) y
Reading symbols from bootloader.symbol...done.
```

- 第五步：查看加载 GDTR 时

Layout src

```
bootloader.asm
31  mov dword [GDT_START_ADDRESS+0x1c],0x0040920b ; 粒度为字节
32
33  ;创建保护模式下平坦模式代码段描述符
34  mov dword [GDT_START_ADDRESS+0x20],0x0000ffff ; 基地址为0,
35  mov dword [GDT_START_ADDRESS+0x24],0x00cf9800 ; 粒度为4kb,
36
37  ;初始化描述符表寄存器GDTR
38  mov word [pgdt], 39 ;描述符表的界限
39  lgdt [pgdt]
40
41  in al,0x92 ;南桥芯片内的端口
42  or al,0000_0010B
43  out 0x92,al ;打开A20
44
```

Note Thread 1 In: output_bootloader_tag L41 PC: 0x7e89

设置断点

```
(gdb) print pgdt
$1 = {<text variable, no debug info>} 0x7ed8 <pgdt>
(gdb) x/16wx 0x7ed8
```

确实为 39

```
0x7ed8: 25051 25205
(gdb) x/1dh 0x7ed8
0x7ed8 <pgdt>: 39
(gdb)
```

- 第六步：查看开第二十一根地址线

```
(gdb) x/1xh 0x92
0x92: 0xf000
(gdb) info register al
al 0x2 2
(gdb)
```

- 第七步：查看打开 cr0 的保护模式标志位

修改前 cr0 是 16，修改后是 17

```
(gdb) ni
(gdb) info register eax
eax 0x10 16
(gdb) ni
(gdb) info register eax
eax 0x11 17
```

- 第八步：远跳转，进入保护模式

```
(gdb) info registers
eax 0x11 17
ecx 0x0 0
edx 0x80 128
ebx 0x1c 28
esp 0x7c00 0x7c00
ebp 0x0 0x0
esi 0x7eec 32492
edi 0x0 0
eip 0x7e9a 0x7e9a <output_bootloader_tag+132>
```

4) 将字符弹射程序改成 32 位，并运行

步骤与复现 example2 步骤相同，mbr 代码也相同，bootloader 前面部分代码也相同，只是在 bootloader 后面的 enter_protect_mode 后加入代码

```
protect_mode_tag db 'enter protect mode'
protect_mode_tag_end:

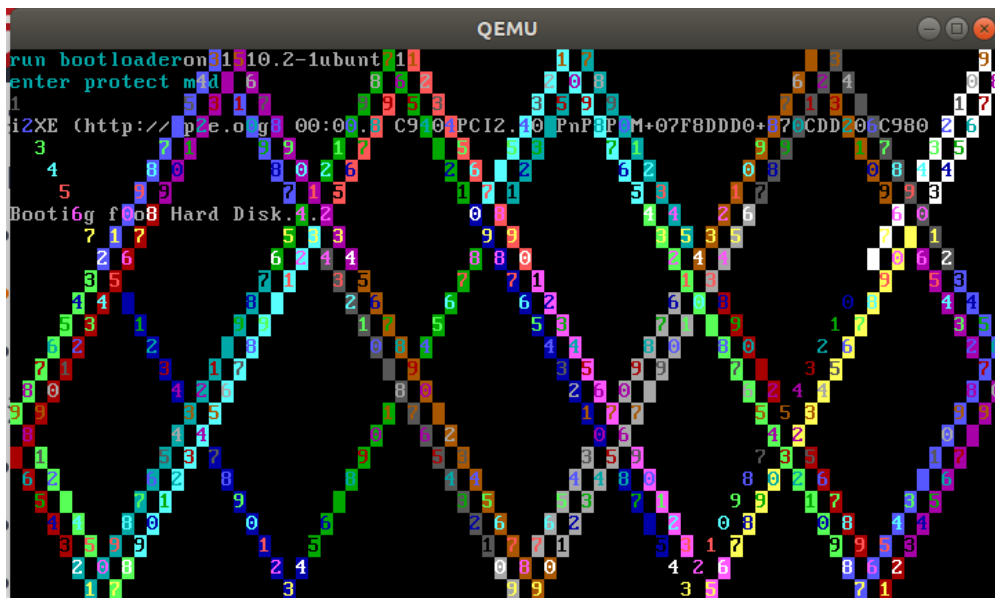
mov ecx, 0
mov dh, 2
mov dl, 0
mov ah, 0x07
mov al, 48

loop:
    call get_random
    call print_al
    call sleep
    jmp loop

; 函数功能：取随机数，打印数字的随机数，颜色的随机数
get_random:
```

注意：GDT 中未引入中断向量表，故不可以使用 intxx 系列软中断

加入代码后的步骤与复现 example1 的步骤一致，不过多叙述



3 关键代码

1) 复现 example1 的代码没有改动

2) 利用 int31h 通过 CHS 方式读取磁盘的代码如下

```
asm_read_hard_disk:
    mov ch, 0 ; 0==柱面=lba/(63*18)
    mov dh, 0 ; 0==磁头= (lba/63)%18
    mov dl, 80h

    mov cl, al ; lba
    inc cl ; 扇区=lba+1

    mov ah, 02h
    mov al, 1
    int 31h
    add bx, 512 ; 缓冲区首地址+=512
    ret
```

$C = LBA // (\text{每柱面磁道数} * \text{每磁道扇区数})$

$H = (LBA // 63) \% \text{每柱面磁道数}$

$S = (LBA \% \text{每磁道扇区数}) + 1$

3) 复现 example2 代码没有改动

4) 字符弹射代码如下

```
mov ecx, 0
mov dh, 2
mov dl, 0
mov ah, 0x07
mov al, 48

loop:
    call get_random
    call print_al
    call sleep
    jmp loop
```

主函数：开始行号 dh 为 2，列号 dl 为 0，颜色 ah，字符 al

```

; 函数功能：取随机数，打印数字的随机数，颜色的随机数
get_random:
    inc ah
    cmp al, 57
    jge set_0

    inc al
    ret

set_0:
    mov al, 48
    ret ;

; 函数功能：睡眠一定时间
sleep:
    pushad
    mov ebx, 12000000 ; 设置需要延迟的时钟周期数
delay:
    dec ebx ; 浪费1个时钟周期
    jnz delay ; 如果还有剩余时钟周期，则继续浪费
    popad
    ret ; 返回调用者

print_al:
    pushad
    mov ebx, 0
    add bl, dh
    imul ebx, ebx, 80
    and edx, 0x000000FF
    add ebx, edx
    imul ebx, ebx, 2
    mov [gs:ebx], ax
    popad
    call cursor_inc
    ret

cursor_inc:
    branch1: cmp ecx, 0 ; 右下
              jne branch2

    b11: cmp dh, 24 ; 行号24
          jne b12
          sub dh, 1
          add dl, 1
          mov ecx, 1 ; 右上
          jmp end_1

    b12: cmp dl, 79 ; 列号79
          jne b13
          sub dl, 1
          add dh, 1
          mov ecx, 3 ; 左下
          jmp end_1

    b13: ; 正常
          add dl, 1
          add dh, 1

    end_1:
    jmp end_if

    branch2: cmp ecx, 1 ; 右上
              jne branch3

    b21: cmp dh, 0 ; 行号0
          jne b22
          add dh, 1

```

Get_random 函数设置字符及其颜色

Sleep 函数睡眠一定时间

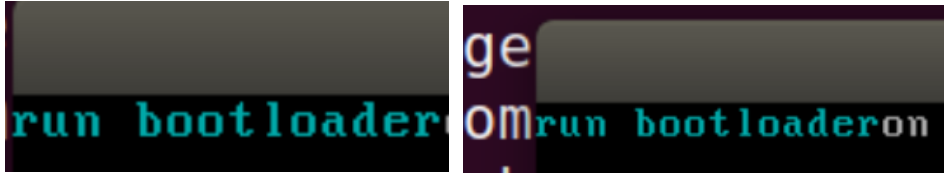
Print_al 函数将字符打印在屏幕上

Crusor_inc 函数设置行列号，进行弹射

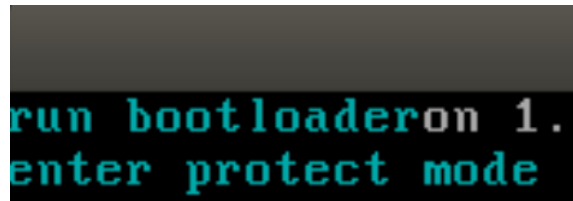
注意：GDT 中未引入中断向量表，故不可以使用 intxx 系列软中断

4 实验结果

1) Assignment1



2) Assignment2 (debug 信息见过程部分)



3) Assignment3

