



中山大學
SUN YAT-SEN UNIVERSITY

本科生实验报告

实验课程: 操作系统原理实验

任课教师: 刘宁

实验题目: 从实模式到保护模式

专业名称: 信息与计算科学

学生姓名: 陈圳煌

学生学号: 22336044

实验地点: 实验中心 D503

实验时间: 2024.04.01

Section 1 实验概述

学习如何从 16 位的实模式跳转到 32 位的保护模式，然后在平坦模式下运行 32 位程序，同时学习使用 I/O 端口和硬件交互。

Section 2 预备知识与实验环境

该节总结实验需要用到的基本知识，以及主机型号、代码编译工具、重要三方库的版本号信息等。

- 预备知识：x86 汇编语言程序设计、Linux 系统命令行工具
- 实验环境：Linux 环境
 - 虚拟机版本/处理器型号：VirtualBox-7.0.6-155176/Ubuntu-18.04.6
 - 代码编辑环境：gedit
 - 代码编译工具：gcc, nasm-2.15.05
 - 重要三方库信息：无

Section 3 实验任务

- 实验任务 1：完成思考题 9(复现 bootloader)
- 实验任务 2：完成思考题 10
- 实验任务 3：完成思考题 11
- 实验任务 4：完成思考题 12

Section 4 实验步骤与实验结果

----- 实验任务 1 -----

- 任务要求：复现“加载 bootloader”一节，说说你是怎么做的并提供结果截图，也可以参考 Ucore、Xv6 等系统源码，实现自己的 LBA 方式的磁盘访问。
- 思路分析：将 lab2 中输出“Hello World”部分的代码放入到 bootloader 中，然后在 MBR 中加载 bootloader 到内存，并跳转到 bootloader 的初始地址执行。
- 实验步骤：首先安排内存地址，bootloader 安排在 MBR 之后，预留五个扇区的空间。如下：

name	start	length	end
MBR	0x7c00	0x200(512B)	0x7e00
bootloader	0x7e00	0xa00(512B * 5)	0x8800

新建一个 `bootloader.asm` 代码文件，然后在 `mbr.asm` 处放入使用 LBA 模式读取硬盘的代码，然后再 MBR 中加载 `bootloader` 到地址 `0x7e00`。

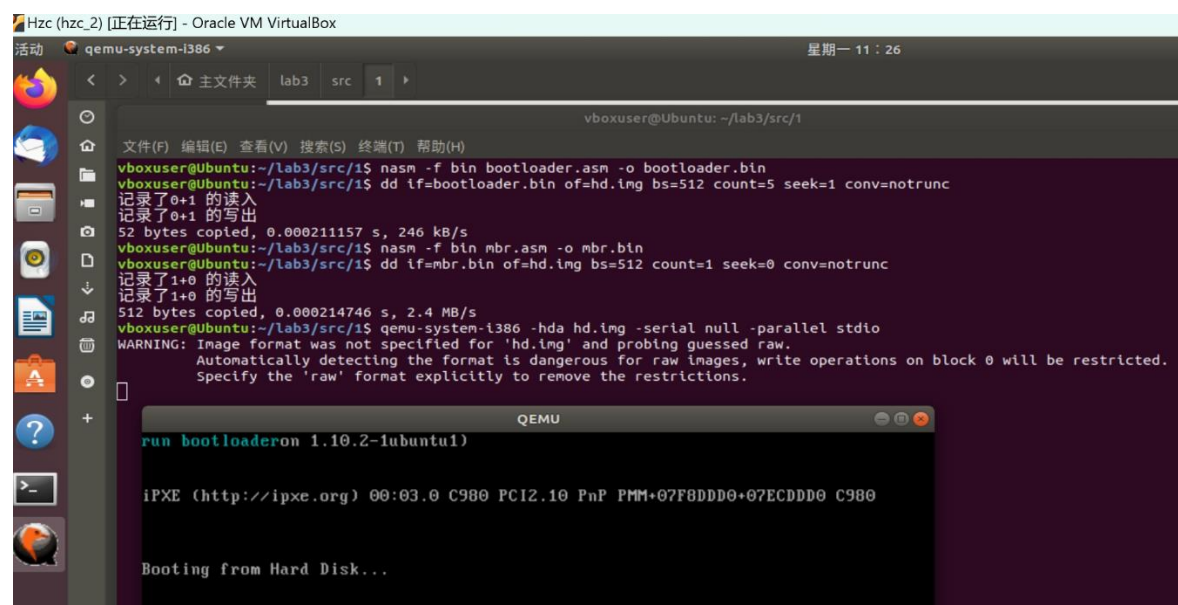
然后编译 `bootloader.asm`，写入硬盘编号为 1 的扇区，共有 5 个扇区。

```
nasm -f bin bootloader.asm -o bootloader.bin
dd if=bootloader.bin of=hd.img bs=512 count=5 seek=1 conv=notrunc
```

`mbr.asm` 也要重新编译和写入硬盘起始编号为 0 的扇区。

```
nasm -f bin mbr.asm -o mbr.bin
dd if=mbr.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
```

使用 `qemu` 运行即可，示例效果如下。



```
Hzc (hzc_2) [正在运行] - Oracle VM VirtualBox
活动 qemu-system-i386 星期一 11:26
vboxuser@Ubuntu: ~/lab3/src/1
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
vboxuser@Ubuntu:~/lab3/src/1$ nasm -f bin bootloader.asm -o bootloader.bin
vboxuser@Ubuntu:~/lab3/src/1$ dd if=bootloader.bin of=hd.img bs=512 count=5 seek=1 conv=notrunc
记录了0+1 的读入
记录了0+1 的写出
52 bytes copied, 0.000211157 s, 246 kB/s
vboxuser@Ubuntu:~/lab3/src/1$ nasm -f bin mbr.asm -o mbr.bin
vboxuser@Ubuntu:~/lab3/src/1$ 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, 0.000214746 s, 2.4 MB/s
vboxuser@Ubuntu:~/lab3/src/1$ qemu-system-i386 -hda hd.img -serial null -parallel stdio
WARNING: Image format was not specified for 'hd.img' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.

QEMU
run bootloaderon 1.10.2-1ubuntu1)

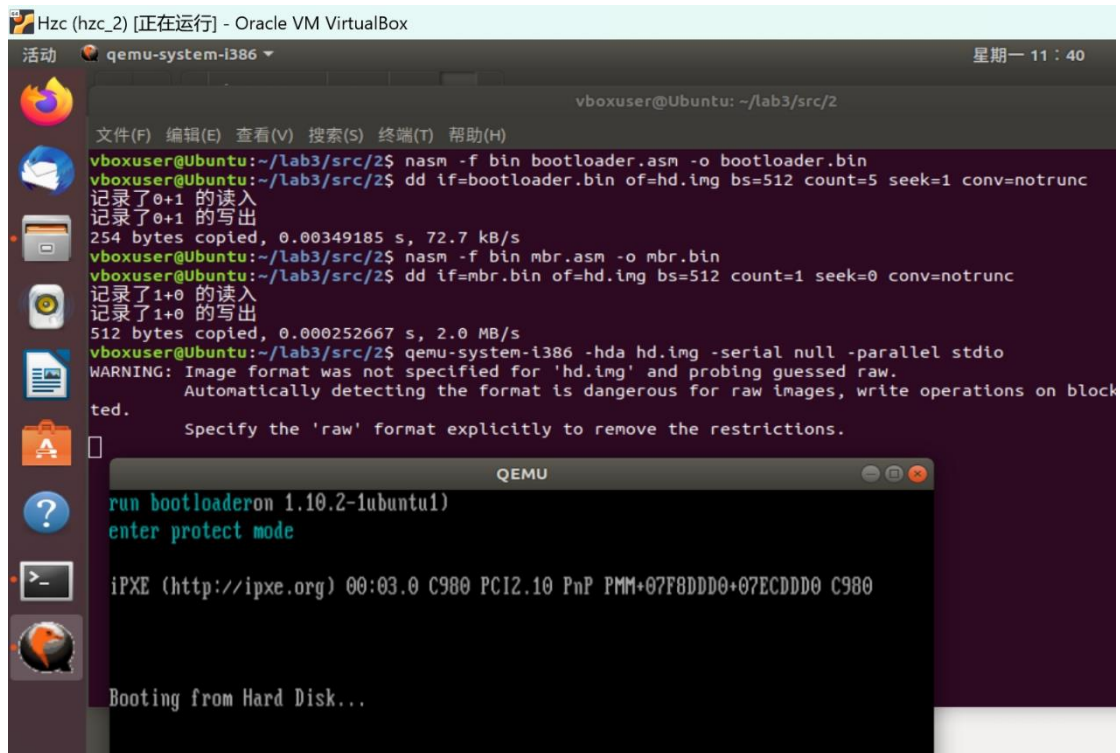
iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DD0+07ECDD0 C980

Booting from Hard Disk...
```

之后：

- (1) 准备 GDT，用 `lgdt` 指令加载 GDTR 信息。
- (2) 打开第 21 根地址线。
- (3) 开启 `cr0` 的保护模式标志位。
- (4) 远跳转，进入保护模式。

修改 `mbr.asm` 的内容，是能够进入保护模式，进行跟上述一样的命令行操作，可以得到以下效果：



上述加载 bootloader 代码如下：

```
org 0x7e00
[bits 16]
mov ax, 0xb800
mov gs, ax
mov ah, 0x03 ;青色
mov ecx, bootloader_tag_end - bootloader_tag
xor ebx, ebx
mov esi, bootloader_tag
output_bootloader_tag:
    mov al, [esi]
    mov word[gs:bx], ax
    inc esi
    add ebx, 2
    loop output_bootloader_tag
jmp $ ; 死循环

bootloader_tag db 'run bootloader'
bootloader_tag_end:
```

mbr.asm 文件代码如下：

```
org 0x7c00
[bits 16]
xor ax, ax ; eax = 0
; 初始化段寄存器，段地址全部设为 0
```

```

mov ds, ax
mov ss, ax
mov es, ax
mov fs, ax
mov gs, ax

; 初始化栈指针
mov sp, 0x7c00
mov ax, 1          ; 逻辑扇区号第 0~15 位
mov cx, 0          ; 逻辑扇区号第 16~31 位
mov bx, 0x7e00     ; bootloader 的加载地址
load_bootloader:
    call asm_read_hard_disk ; 读取硬盘
    inc ax
    cmp ax, 5
    jle load_bootloader
jmp 0x0000:0x7e00   ; 跳转到 bootloader

jmp $ ; 死循环

asm_read_hard_disk:
; 从硬盘读取一个逻辑扇区

; 参数列表
; ax=逻辑扇区号 0~15 位
; cx=逻辑扇区号 16~28 位
; ds:bx=读取出的数据放入地址

; 返回值
; bx=bx+512

    mov dx, 0x1f3
    out dx, al    ; LBA 地址 7~0

    inc dx        ; 0x1f4
    mov al, ah
    out dx, al    ; LBA 地址 15~8

    mov ax, cx

    inc dx        ; 0x1f5
    out dx, al    ; LBA 地址 23~16

    inc dx        ; 0x1f6

```

```

    mov al, ah
    and al, 0x0f
    or al, 0xe0    ; LBA 地址 27~24
    out dx, al

    mov dx, 0x1f2
    mov al, 1
    out dx, al    ; 读取 1 个扇区

    mov dx, 0x1f7    ; 0x1f7
    mov al, 0x20    ; 读命令
    out dx, al

    ; 等待处理其他操作
.waits:
    in al, dx        ; dx = 0x1f7
    and al, 0x88
    cmp al, 0x08
    jnz .waits

    ; 读取 512 字节到地址 ds:bx
    mov cx, 256    ; 每次读取一个字, 2 个字节, 因此读取 256 次即可
    mov dx, 0x1f0
.readw:
    in ax, dx
    mov [bx], ax
    add bx, 2
    loop .readw

    ret

times 510 - ($ - $$) db 0
db 0x55, 0xaa

```

进入保护模式时，mbr.asm 代码修改如下：

```

#include "boot.inc"

[bits 16]
xor ax, ax ; eax = 0
; 初始化段寄存器, 段地址全部设为 0
mov ds, ax
mov ss, ax
mov es, ax
mov fs, ax

```

```

mov gs, ax

; 初始化栈指针
mov sp, 0x7c00

mov ax, LOADER_START_SECTOR
mov cx, LOADER_SECTOR_COUNT
mov bx, LOADER_START_ADDRESS

load_bootloader:
    push ax
    push bx
    call asm_read_hard_disk ; 读取硬盘
    add sp, 4
    inc ax
    add bx, 512
    loop load_bootloader

    jmp 0x0000:0x7e00 ; 跳转到 bootloader

jmp $ ; 死循环

; asm_read_hard_disk(memory,block)
; 加载逻辑扇区号为 block 的扇区到内存地址 memory

asm_read_hard_disk:
    push bp
    mov bp, sp

    push ax
    push bx
    push cx
    push dx

    mov ax, [bp + 2 * 3] ; 逻辑扇区低 16 位

    mov dx, 0x1f3
    out dx, al ; LBA 地址 7~0

    inc dx ; 0x1f4
    mov al, ah
    out dx, al ; LBA 地址 15~8

    xor ax, ax

```

```

inc dx      ; 0x1f5
out dx, al   ; LBA 地址 23~16 = 0

inc dx      ; 0x1f6
mov al, ah
and al, 0x0f
or al, 0xe0  ; LBA 地址 27~24 = 0
out dx, al

mov dx, 0x1f2
mov al, 1
out dx, al   ; 读取 1 个扇区

mov dx, 0x1f7 ; 0x1f7
mov al, 0x20  ; 读命令
out dx, al

; 等待处理其他操作
.waits:
in al, dx      ; dx = 0x1f7
and al, 0x88
cmp al, 0x08
jnz .waits

; 读取 512 字节到地址 ds:bx
mov bx, [bp + 2 * 2]
mov cx, 256    ; 每次读取一个字, 2 个字节, 因此读取 256 次即可
mov dx, 0x1f0
.readw:
in ax, dx
mov [bx], ax
add bx, 2
loop .readw

pop dx
pop cx
pop bx
pop ax
pop bp

ret

times 510 - ($ - $$) db 0

```



```
db 0x55, 0xaa
```

----- 实验任务 2 -----

- 任务要求：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。
- 思路分析：首先要了解 LBA 模式转成 CHS 模式时对应的磁盘扇区、磁头、驱动器是如何对应的。
- 实验步骤：

首先了解两种寻址方式是如何进行转化的，由公式可知：

```
LBA=NH×NS×C+NS×H+S-1;  
C=(LBA div NS)div NH;  
H=(LBA div NS)mod NH;  
S=(LBA mod NS)+1  
例如 LBA = 0 则 CHS = 0/0/1  
从 C/H/S 到 LBA 的计算公式：  
LBA= (C-CS) *PH*PS+ (H-HS) *PS+(S-SS)
```

从而由原本 LBA 模式的 1 可以得到 CHS 模式下为 0/0/2，即将 mbr.asm 文件中的 asm_read_hard_disk 段内的代码修改为 int 13h 中断即可，注意需要保留 0x1f7 地址的内容。

修改后的 mbr.asm 代码如下：

```
;10  
xor ax, ax ; eax = 0  
; 初始化段寄存器，段地址全部设为 0  
mov ds, ax  
mov ss, ax  
mov es, ax  
mov fs, ax  
mov gs, ax  
  
; 初始化栈指针  
mov sp, 0x7c00  
mov ax, 1 ; 逻辑扇区号第 0~15 位  
mov cx, 0 ; 逻辑扇区号第 16~31 位  
mov bx, 0x7e00 ; bootloader 的加载地址  
load_bootloader:  
call asm_read_hard_disk ; 读取硬盘  
inc ax  
cmp ax, 5
```

```

    jle load_bootloader
jmp 0x0000:0x7e00    ; 跳转到 bootloader

jmp $ ; 死循环

asm_read_hard_disk:
; 从硬盘读取一个逻辑扇区

; 参数列表
; ax=逻辑扇区号 0~15 位
; cx=逻辑扇区号 16~28 位
; ds:bx=读取出的数据放入地址

; 返回值
; bx=bx+512

    mov ah, 02h
    mov al, 1
    mov bx, 0
    mov ch, 0
    mov cl, 2
    mov dh, 0
    mov dl, 0x00
    int 13h

    mov dx, 0x1f7    ; 0x1f7
    mov al, 0x20    ; 读命令
    out dx, al

; 等待处理其他操作
.waits:
    in al, dx        ; dx = 0x1f7
    and al, 0x88
    cmp al, 0x08
    jnz .waits

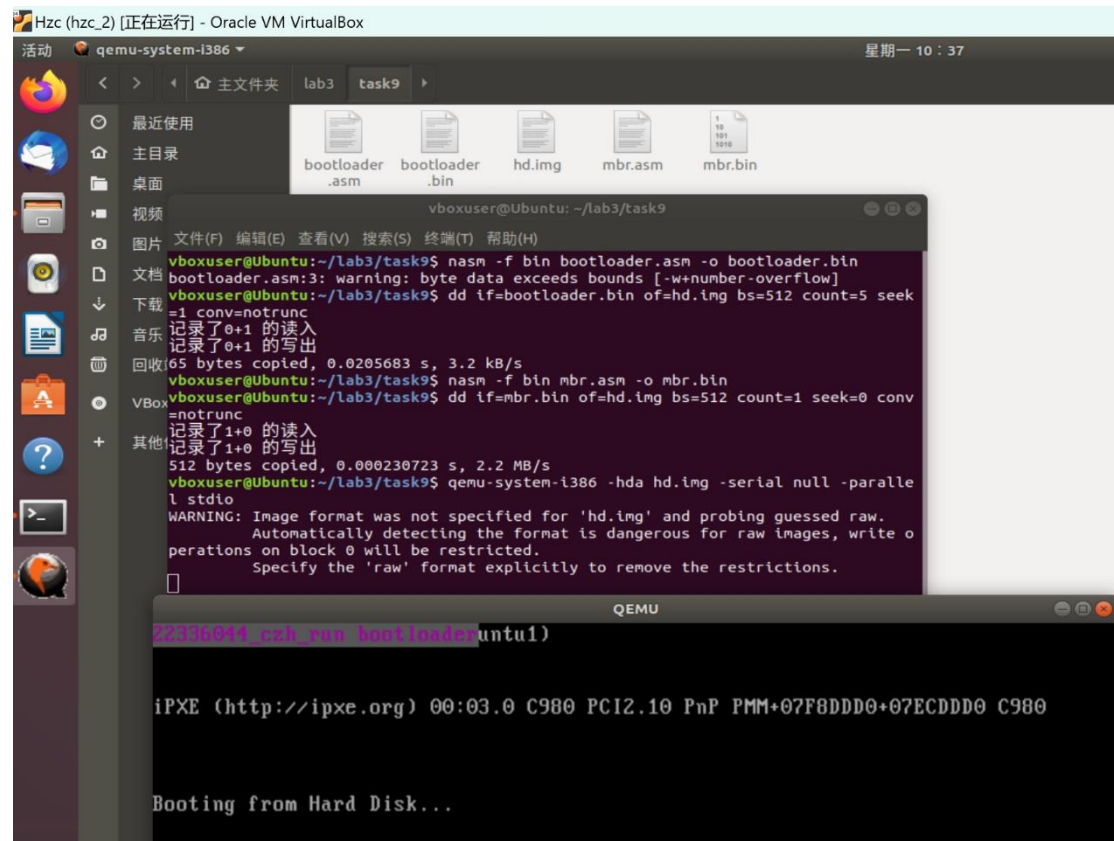
; 读取 512 字节到地址 ds:bx
    mov cx, 256    ; 每次读取一个字, 2 个字节, 因此读取 256 次即可
    mov dx, 0x1f0
.readw:
    in ax, dx
    mov [bx], ax
    add bx, 2
    loop .readw

```

```
ret
```

```
times 510 - ($ - $$) db 0  
db 0x55, 0xaa
```

测试效果如下图：



----- 实验任务 3 -----

- 任务要求：复现“进入保护模式”一节，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
- 实验步骤：

首先要更新 nasm 到 2.15 版本，首先下载压缩包，然后使用指令：

```
tar -xvf nasm-2.15.05.tar.xz
```

解压文件，进入解压的文件夹，使用下列指令：

```
./configure  
make  
sudo make install
```

最后查看 `nasm` 的版本可以看到已经更新到 2.15 了。

下面进行 debug，首先进入 `mbr.asm` 和 `bootloader.asm` 所在的文件夹，然后删掉第二行的伪指令，编译 `mbr.asm`，生成可重定义文件 `mbr.o`，`-g` 是为了加上 debug 信息。

```
nasm -o mbr.o -g -f elf32 mbr.asm
```

然后我们为可重定位文件 `mbr.o` 指定起始地址 `0x7c00`，分别链接生成可执行文件 `mbr.symbol` 和 `mbr.bin`。

```
ld -o mbr.symbol -melf_i386 -N mbr.o -Ttext 0x7c00
```

```
ld -o mbr.bin -melf_i386 -N mbr.o -Ttext 0x7c00 --oformat binary
```

同样，对 `bootloader.asm` 进行同样操作：

```
nasm -o bootloader.o -g -f elf32 bootloader.asm
```

```
ld -o bootloader.symbol -melf_i386 -N bootloader.o -Ttext 0x7e00
```

```
ld -o bootloader.bin -melf_i386 -N bootloader.o -Ttext 0x7e00 --oformat  
binary
```

然后将两个的 `bin` 文件分别写入 `hd.img`

```
dd if=mbr.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
```

```
dd if=bootloader.bin of=hd.img bs=512 count=5 seek=1 conv=notrunc
```

接着开始 debug 的基本流程

首先用 `qemu` 加载 `hd.img` 运行

```
qemu-system-i386 -s -S -hda hd.img -serial null -parallel stdio
```

之后在另一个终端打开 `gdb` 并连接

```
target remote:1234
```

在第一条指令处设置断点

```
b *0x7c00
```

输入 `c` 运行到断点处暂停

然后可以打开显示源代码的窗口

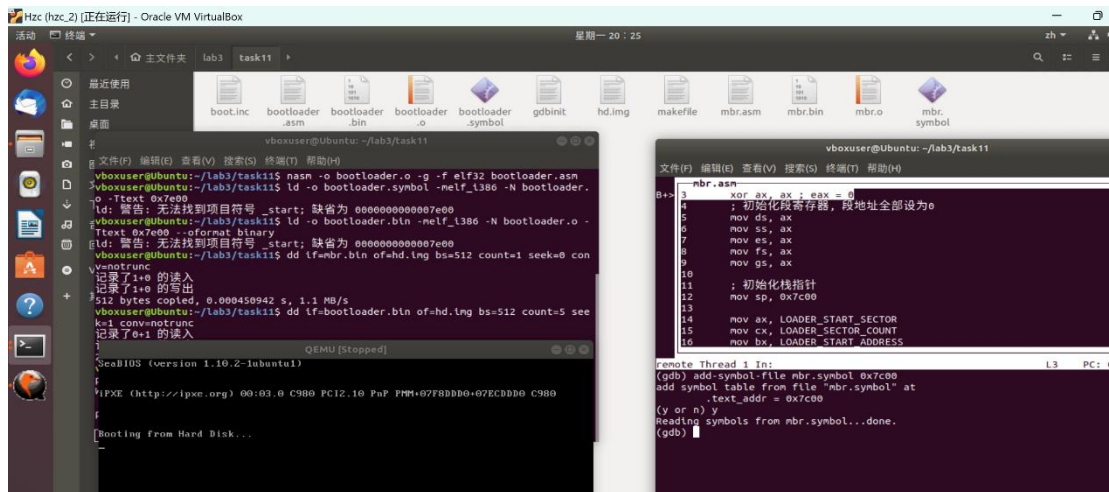
```
layout src
```

并加载对应的符号表

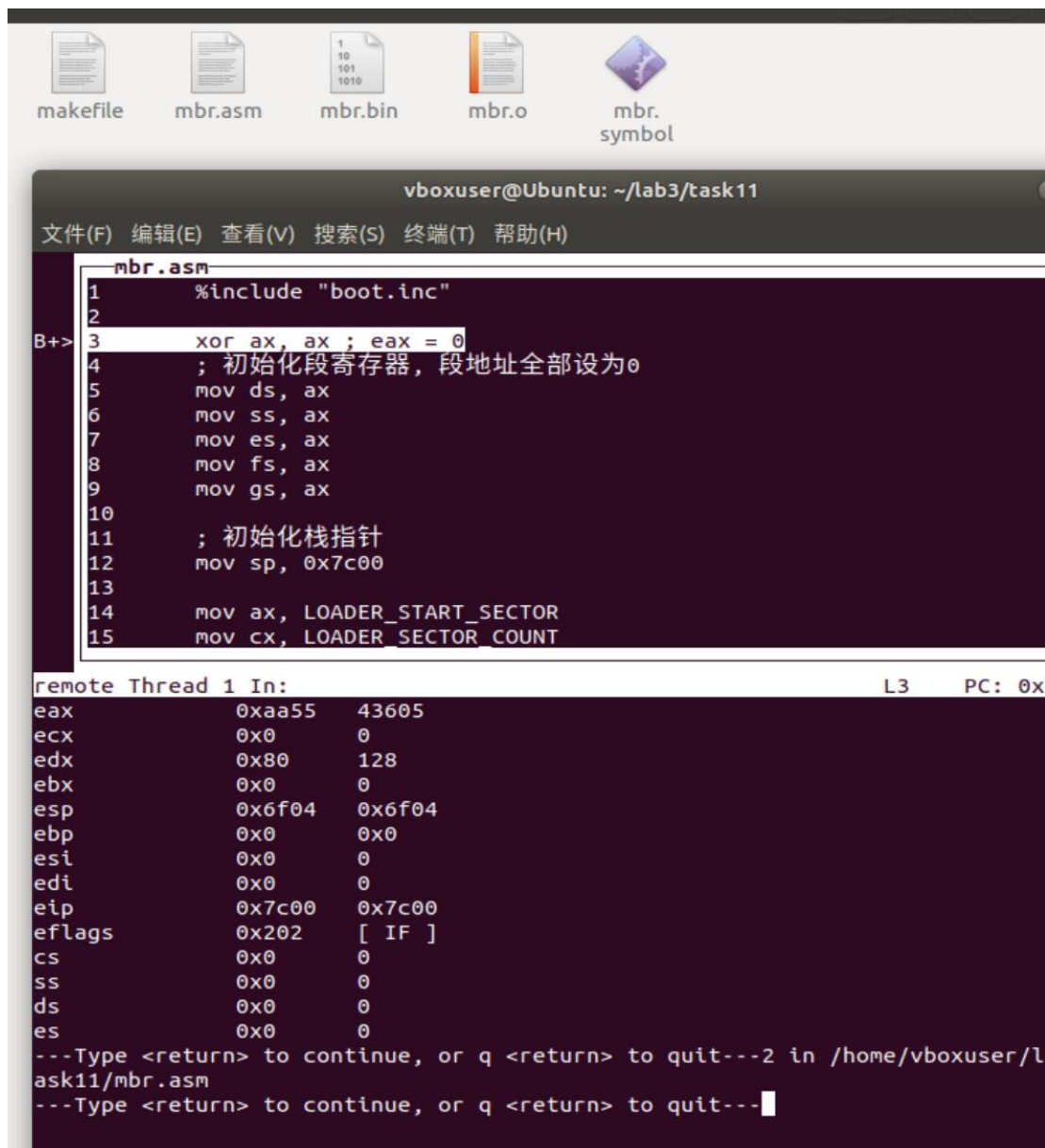
```
add-symbol-file mbr.symbol 0x7c00
```

此时在 `src` 窗口显示出了源代码，其中 `B+` 表示断点，白色条框表示下一条执行的指令。

如下图：



输入 info registers 可以查看寄存器

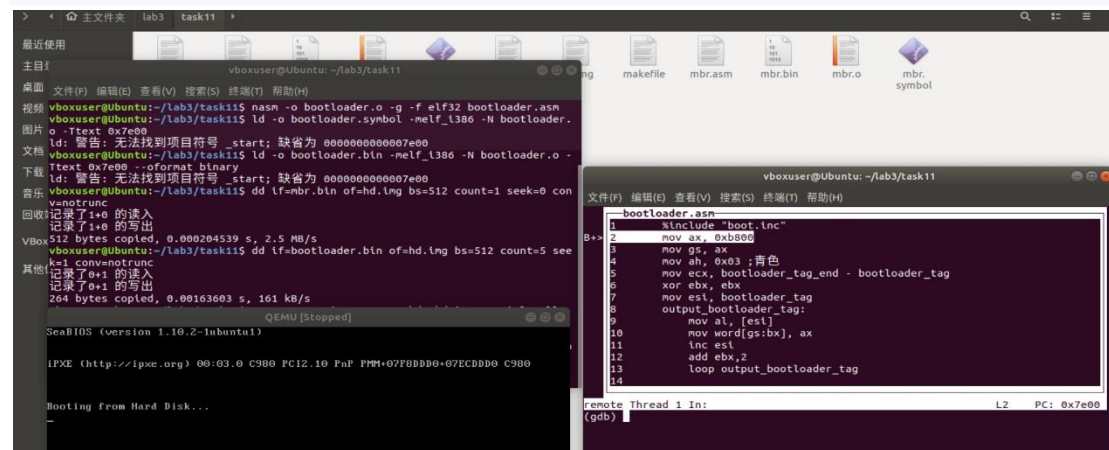


接着在 bootloader 的起始位置设置断点并执行且显示源代码

```
b *0x7e00
```

```
c
```

```
add-symbol-file bootloader.symbol 0x7e00
```



最后在进入保护模式第一条指令处设置断点

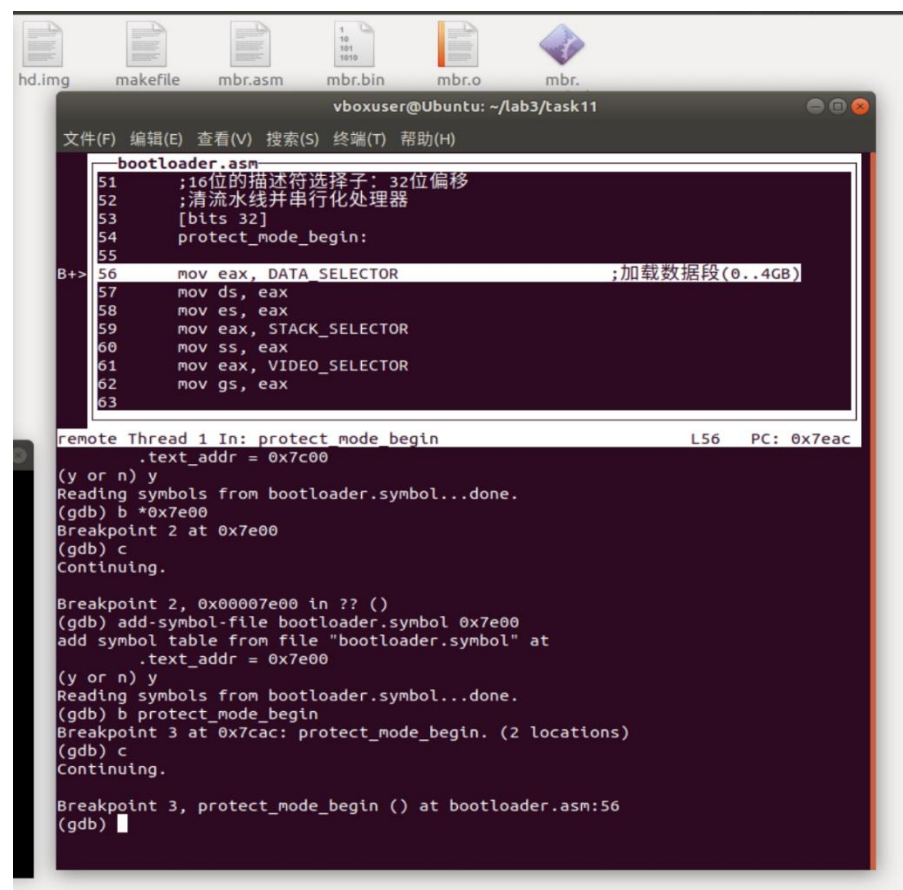
```
b protect_mode_begin
```

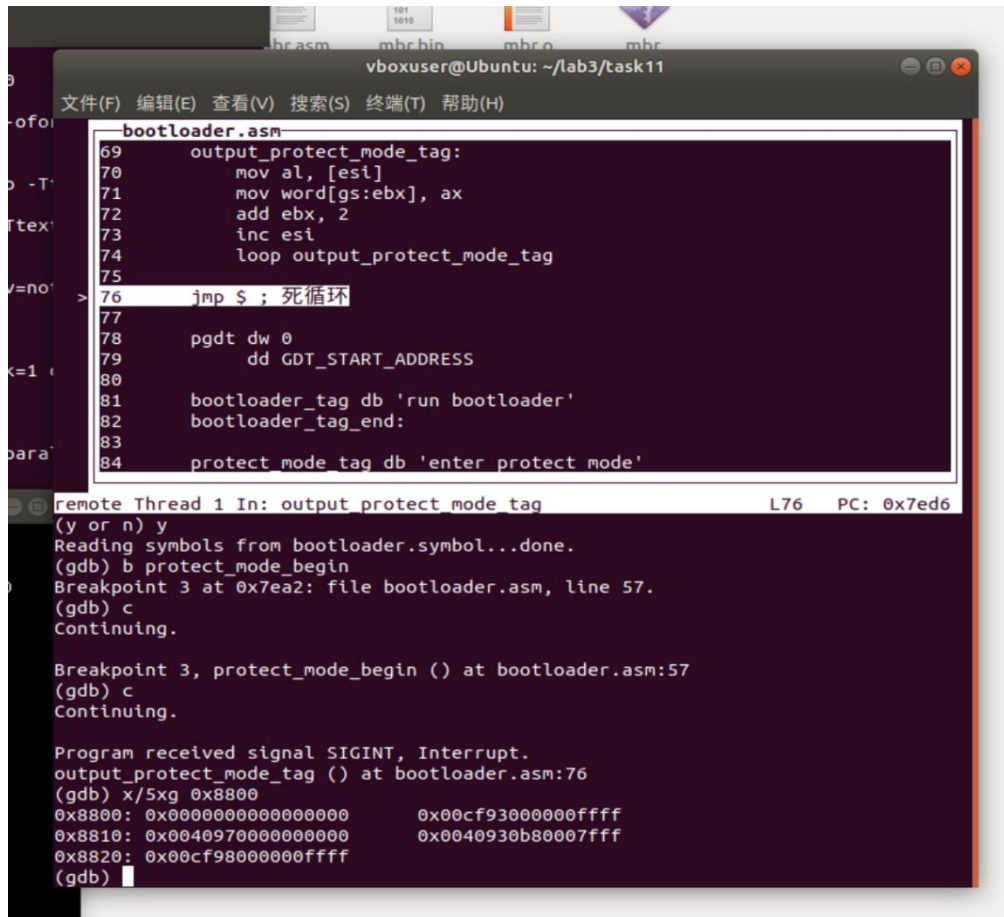
```
c
```

最后执行到死循环处使用 ctrl+c 中断后使用

```
x/5xg 0x8800
```

查看 GDT 的 5 个描述符的内容，如下：





```
bootloader.asm
69     output_protect_mode_tag:
70         mov al, [esi]
71         mov word[gs:ebx], ax
72         add ebx, 2
73         inc esi
74         loop output_protect_mode_tag
75
76     jmp $ ; 死循环
77
78     pgdt dw 0
79         dd GDT_START_ADDRESS
80
81     bootloader_tag db 'run bootloader'
82     bootloader_tag_end:
83
84     protect_mode_tag db 'enter protect mode'
```

```
remote Thread 1 In: output_protect_mode_tag L76 PC: 0x7ed6
(y or n) y
Reading symbols from bootloader.symbol...done.
(gdb) b protect_mode_begin
Breakpoint 3 at 0x7ea2: file bootloader.asm, line 57.
(gdb) c
Continuing.

Breakpoint 3, protect_mode_begin () at bootloader.asm:57
(gdb) c
Continuing.

Program received signal SIGINT, Interrupt.
output_protect_mode_tag () at bootloader.asm:76
(gdb) x/5xg 0x8800
0x8800: 0x0000000000000000      0x00cf93000000ffff
0x8810: 0x0040970000000000      0x0040930b80007fff
0x8820: 0x00cf98000000ffff
(gdb)
```

可以看到与我们设置的吻合。

在实验过程中遇到的问题：在进行 debug 的时候由于将[bits 16]一起删除了，导致了在设置 0x7e00 为断点的时候会在死循环处不断 continuing，将代码中循环删去后在最后的死循环处无法停止，且 GDT 内容为 0，最后加上[bits 16]后可以正常 debug 并且最后 GDT 描述内容相符。

----- 实验任务 4 -----

- 任务要求：进入保护模式后，编写并执行一个自己的 32 位汇编程序。
- 思路分析：使用两种不同的自定义颜色和一个自定义的起始位置(x,y)，使得 bootloader 加载后，在显示屏坐标(x,y)处开始输出学号+名字的缩写。
- 实验步骤：

在复现 bootloader 的程序的基础上，先设置偏移量(行坐标*行数+列坐标)bootloader.asm 代码文件的循环之前加上输出信息，将输出字符传输给 al 寄存器，将颜色信息传输给 ah 寄存器，使用

```
mov [gs:di], ax
```

指令将信息放到指定内存地址，并将存放起始偏移量的寄存器自加(因为是 16 位所以要加 2)，依次输出字符信息。

代码如下：

```
mov ax, 0xb800
mov gs, ax
mov ah, 0901
mov ecx, bootloader_tag_end - bootloader_tag
xor ebx, ebx
mov esi, bootloader_tag

xor di, di      ; DI 寄存器用于表示在显存中的偏移量
mov di, 658     ; 设置偏移量到 (4,9) 的位置

; 输出第一个 '2', 属性为亮白色 (0x0F) 背景为黑色
mov al, '2'
mov ah, 0x0F    ; 白色
mov [gs:di], ax
add di, 2

mov al, '2'
mov ah, 0901
mov [gs:di], ax
add di, 2

mov al, '3'
mov ah, 0x0F
mov [gs:di], ax
add di, 2

mov al, '3'
mov ah, 0901
mov [gs:di], ax
add di, 2

mov al, '6'
mov ah, 0x0F
mov [gs:di], ax
add di, 2

mov al, '0'
mov ah, 0901
mov [gs:di], ax
add di, 2
```



```

mov al, '4'
mov ah, 0x0F
mov [gs:di], ax
add di, 2

mov al, '4'
mov ah, 0901
mov [gs:di], ax
add di, 2

mov al, 'c'
mov ah, 0x0F
mov [gs:di], ax
add di, 2

mov al, 'z'
mov ah, 0901
mov [gs:di], ax
add di, 2

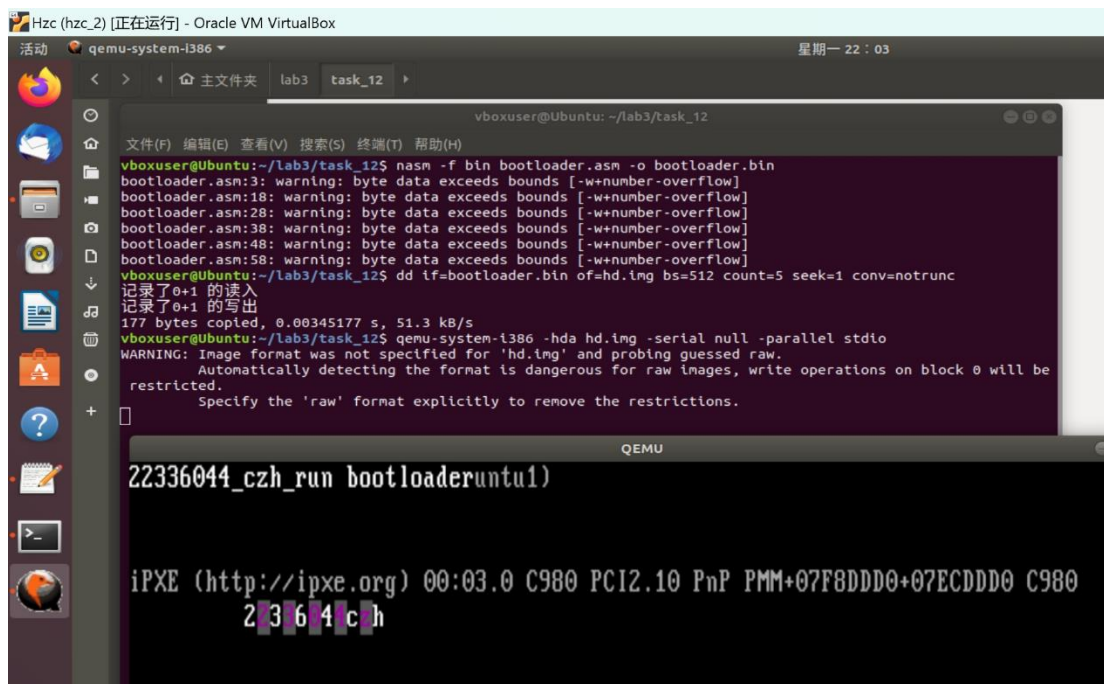
mov al, 'h'
mov ah, 0x0F
mov [gs:di], ax

output_bootloader_tag:
    mov al, [esi]
    mov word[gs:bx], ax
    inc esi
    add ebx, 2
    loop output_bootloader_tag
jmp $ ; 死循环

bootloader_tag db '22336044_czh_run bootloader'
bootloader_tag_end:

```

实验效果展示;



Section 5 实验总结与心得体会

在这次实验中，刚开始对加载 bootloader，进入保护模式都算顺利，在 debug 阶段，更新了 nasm 后也可以顺利进行，直到跳转地址到 0x7e00 时发现问题，经过查资料和与其他同学讨论，最后解决了这个问题。

Section 6 附录：参考资料清单

- [1] [SYSU-2023-Spring-Operating-System: 中山大学 2023 学年春季操作系统课程 - Gitee.com](#)
- [2] [磁盘寻址方式--CHS 和 LBA 寻址方式-CSDN 博客](#)
- [3] [使用 GDB 查看和修改寄存器的值_gdb 查看寄存器的值-CSDN 博客](#)
- [4] [如何利用 gdb 调试程序之细节（info reg 命令以及寄存器地址）-CSDN 博客](#)
- [5] [INT13 中断详解_int13 中断功能-CSDN 博客](#)
- [6] [磁盘数据寻址方式（CHS 与 LBA 相互转换）_将 chs 地址转换为 lba-CSDN 博客](#)
- [7] [LBA 和 CHS 转换\(转\) - 姜大伟 - 博客园 \(cnblogs.com\)](#)
- [8] [BIOS int 13H 中断介绍_bois 13 中断驱动器编号-CSDN 博客](#)
- [9] [磁盘数据寻址方式（CHS 与 LBA 相互转换）_将 chs 地址转换为 lba-CSDN 博客](#)
- [10] [读取磁盘：CHS 方式-CSDN 博客](#)