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:保护模式

专业名称: 计算机科学与技术

学生姓名:罗弘杰

学生学号: 22336173

实验地点: 实验中心D503

实验时间: 2024/4/12

实验资料：[lab3 · NelsonCheung/SYSU-2023-Spring-Operating-System - 码云 - 开源中国 (gitee.com)]

实验要求

学习从LBA和C/H/S的磁盘寻址方式，以及使用IO和中断实现的磁盘读取

学习进入保护模式的方式

学习gdb调试源码级程序

实验任务

复现example1, 说说怎么做并截图

在example1的基础上将LBA28的寻址方式改为CHS，同时给出逻辑扇区号向CHS的转换公式

利用gdb进行实验资料例子2的debug分析

实验过程

任务一

在实验资料的基础上编写Mbr和bootloader.asm，mbr负责加载bootloader，bootloader的任务是打印字符。

我的理解是：操作系统为了性能需求，启动时只会自动加载512B的MBR, 其余磁盘操作由MBR内容管理和控制

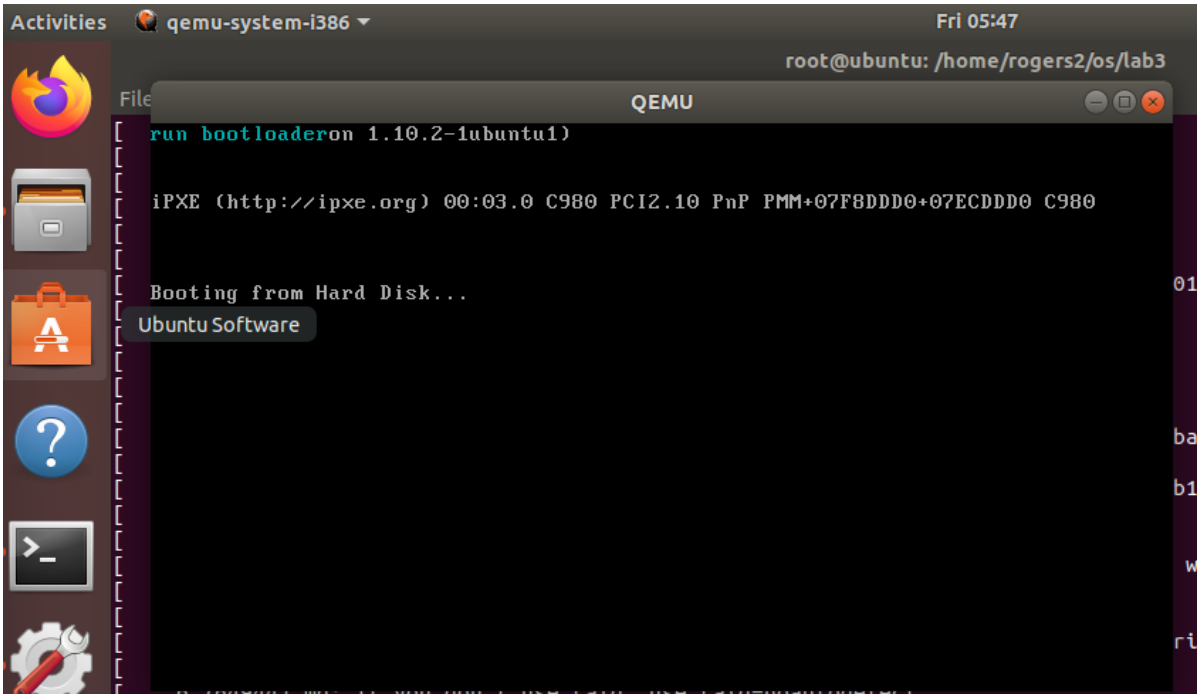
过程：

```
bootloader.asm  bootloader.bin  hd.img  mbr.asm  mbr.bin  mbr_chs.asm
```

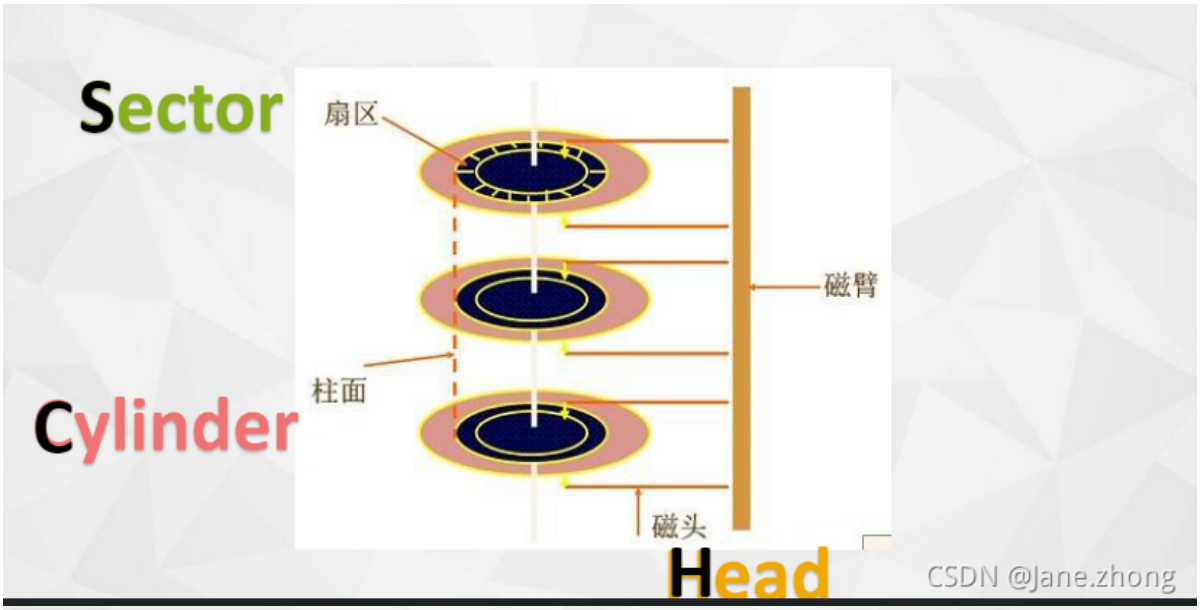
编写bootloader.asm和mbr.asm两个文件，然后编译为可执行文件

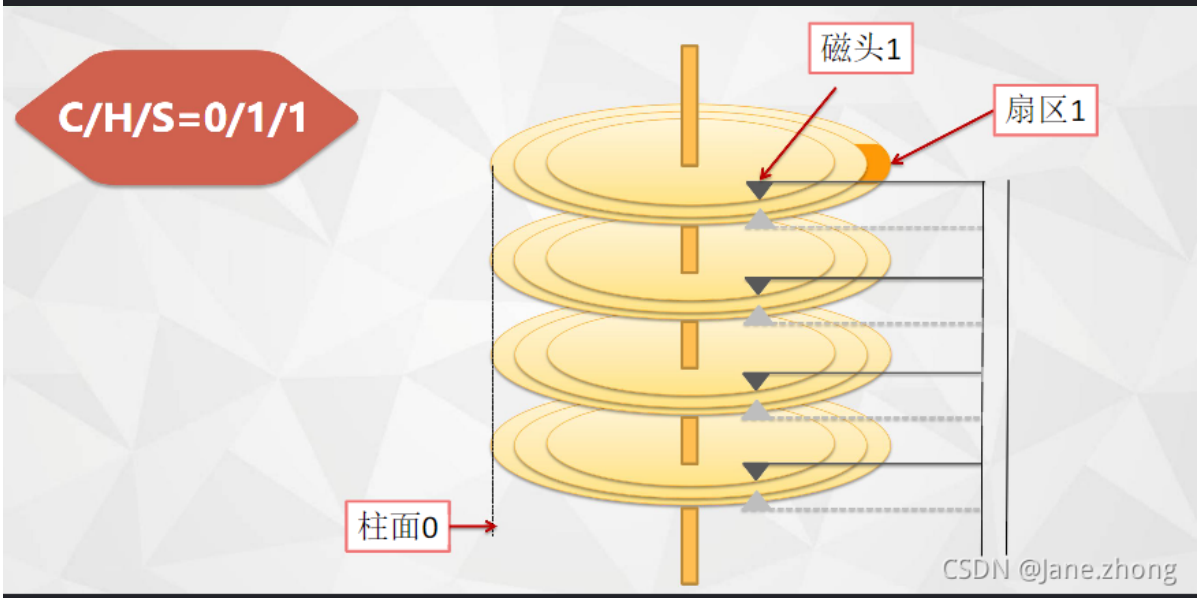
写入到虚拟机的磁盘上，mbr.asm写的位置时0号扇区，数量为1，bootloader为扇区1，数量为5.

然后使用QEMU命令启动虚拟机



任务二





先复习磁盘工作原理：

磁盘空间

$$Space = c * s * h * 512(\text{扇区容量})$$

先确定柱面，在柱面上确定磁头，在磁头确定的磁道上找到扇区

编号差异：LBA规则的扇区编号从0开始，但是C/H/S的编号从1开始，从CHS到LBA转换要-1.

CHS->LBA

$$LBA = (c * HPC + h) * SPT + s - 1$$

HPC是每个柱面的磁头数目，SPT是每个磁道上的扇区总数

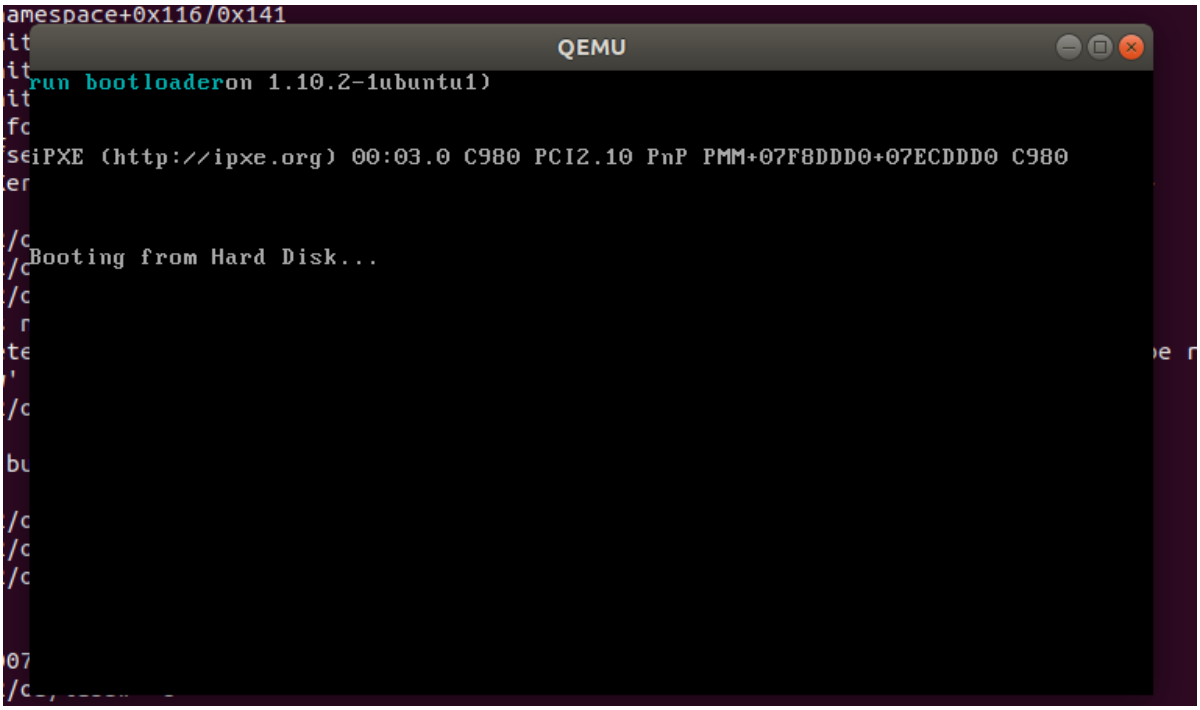
由此可推出反变换公式：

```
C = LBA //(HPC * SPT) #注意是整除
H = [(LBA +1)//每磁道扇区总数SPT]mod HPC
S = (LBA +1) % 每磁道扇区总数SPT
```

在例一中， 读取的LBA从1-5，HPC为18， SPT为63，所以柱面是0，磁道也是0， S从1递增到6；并根据中断读取的参数要求重新编写mbr_chs.asm,编译以后在qemu上运行

```
mov ax, 2                ; 物理扇区第2位
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=起始扇区号

; 返回值
; bx=bx+512
; ax=ax+1
    mov dl, 80h
    mov dh, 0; 磁头号
    mov ch, 0; 柱面号
    mov cl, al; 扇区号


    mov ah, 2; 功能号
    mov al, 1; 扇区数
    int 13h; 调用int 13h中断

    add bx, 512; bx=bx+512, 读取下一个扇区
    ret
```

在这里逻辑扇区和物理扇区编号的差异是要注意的细节（逻辑从0开始，物理从1开始），然后按照任务1的方式，再编译并写入磁盘，启动qemu



实现相同的功能。

任务三

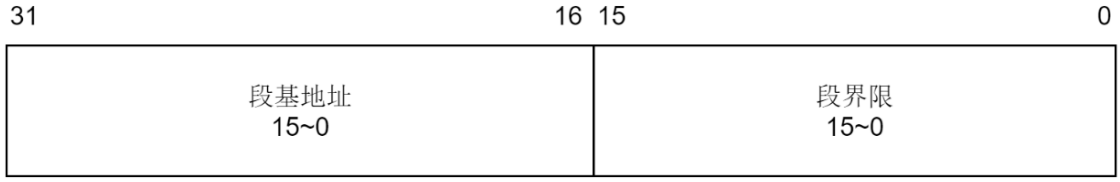
复现实验资料中进入保护模式的程序，然后使用gdb调试。

首先编写boot.inc的头文件

```
; 常量定义区
; _____________Loader_____________
; 加载器扇区数
LOADER_SECTOR_COUNT equ 5
; 加载器起始扇区
LOADER_START_SECTOR equ 1
; 加载器被加载地址
LOADER_START_ADDRESS equ 0x7e00
; _____________GDT_____________
; GDT起始位置
GDT_START_ADDRESS equ 0x8800   ; gdt表的起始位置
```

然后重新编写bootloader.asm，在输出字符后进入保护模式

CPU需要知道当前运行中程序的段地址空间信息，然后才能执行地址保护，阻止程序越界访问。段地址空间信息是通过段描述符(segment descriptor)来给出的。段描述符中包含了段基地址(段的起始地址)、段界限(段的长度)等，共计64字节，如下所示。上面一行是高32字节、下面一行是低32字节。




```
%include "boot.inc"
[bits 16]
mov ax, 0xb800 ;显存段地址
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

;空描述符
mov dword [GDT_START_ADDRESS+0x00],0x00
mov dword [GDT_START_ADDRESS+0x04],0x00

;创建描述符，这是一个数据段，对应0~4GB的线性地址空间
mov dword [GDT_START_ADDRESS+0x08],0x0000ffff    ; 基地址为0，段界限为0xFFFF
mov dword [GDT_START_ADDRESS+0x0c],0x00cf9200    ; 粒度为4KB，存储器段描述符

;建立保护模式下的堆栈段描述符
mov dword [GDT_START_ADDRESS+0x10],0x00000000    ; 基地址为0x00000000，界限0x0
mov dword [GDT_START_ADDRESS+0x14],0x00409600    ; 粒度为1个字节

;建立保护模式下的显存描述符
mov dword [GDT_START_ADDRESS+0x18],0x80007fff    ; 基地址为0x000B8000，界限0x07FF
mov dword [GDT_START_ADDRESS+0x1c],0x0040920b    ; 粒度为字节

;创建保护模式下平坦模式代码段描述符
mov dword [GDT_START_ADDRESS+0x20],0x0000ffff    ; 基地址为0，段界限为0xFFFF
mov dword [GDT_START_ADDRESS+0x24],0x00cf9800    ; 粒度为4kb，代码段描述符


;初始化描述符表寄存器GDTR
```

```
mov word [pgdt], 39      ;描述符表的界限
lgdt [pgdt]

; _____________Selector_____________
;平坦模式数据段选择子
DATA_SELECTOR equ 0x8
;平坦模式栈段选择子
STACK_SELECTOR equ 0x10
;平坦模式视频段选择子
VIDEO_SELECTOR equ 0x18
VIDEO_NUM equ 0x18
;平坦模式代码段选择子
CODE_SELECTOR equ 0x20

in al,0x92                          ;南桥芯片内的端口
or al,0000_0010B
out 0x92,al                         ;打开A20

cli                                 ;禁用中断
mov eax,cr0
or eax,1
mov cr0,eax                         ;设置PE位

jmp dword CODE_SELECTOR:protect_mode_begin

;16位的描述符选择子：32位偏移
;清流水线并串行化处理器
[bits 32]
protect_mode_begin:

mov eax, DATA_SELECTOR                      ;加载数据段(0..4GB)选择子
mov ds, eax
mov es, eax
mov eax, STACK_SELECTOR
mov ss, eax
mov eax, VIDEO_SELECTOR
mov gs, eax

mov ecx, protect_mode_tag_end - protect_mode_tag
mov ebx, 80 * 2
mov esi, protect_mode_tag
mov ah, 0x3
output_protect_mode_tag:
    mov al, [esi]
    mov word[gs:ebx], ax
    add ebx, 2
    inc esi
    loop output_protect_mode_tag

jmp $ ; 死循环

pgdt dw 0    ; pgdt 将占据48位，前16位为界限，在这里是39，后32位是起始地址
    dd GDT_START_ADDRESS

bootloader_tag db 'bootloader'
bootloader_tag_end:
```

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

代码分析：

进入保护模式需要4个过程：

1. **准备GDT，用lgdt指令加载GDTR信息。**
2. **打开第21根地址线。**//扩大内存访问空间是保护模式的出现背景
3. **开启cr0的保护模式标志位。**
4. **远跳转，进入保护模式。**

怎么做：

1. GDTR是一个x86架构专用寄存器，是48位存储全局描述符表的寄存器，在这里我们先把GDT信息存储在内存中，然后使用lgdt指令加载到该寄存器，修改全局描述符表信息。

```
mov word [pgdt], 39      ;描述符表的界限
lgdt [pgdt]
```

2. 南桥芯片0x92端口的第二位控制第二十条地址线的开关，将其置为1就能打开第二十条地址线；

   使用与方法可以将其置为一

```
in al,0x92                          ;南桥芯片内的端口
or al,0000_0010B                     ; 将第二位置为1
out 0x92,al                         ;打开A20
```

3. cr0是专用寄存器，将其最低位（protection enable）置为1，就可以启用保护模式。

   禁用中断保证当前代码执行

```
cli                                 ;禁用中断
mov eax,cr0
or eax,1
mov cr0,eax                         ;设置PE位
```

4. `jmp dword CODE_SELECTOR:protect_mode_begin` 用于执行跳转到指定代码段的指定地址，进入保护模式下的代码执行。

```
jmp dword CODE_SELECTOR:protect_mode_begin

;16位的描述符选择子：32位偏移
;清流水线并串行化处理器
[bits 32]
protect_mode_begin:

mov eax, DATA_SELECTOR                          ;加载数据段(0..4GB)选择子
mov ds, eax
mov es, eax
mov eax, STACK_SELECTOR
```

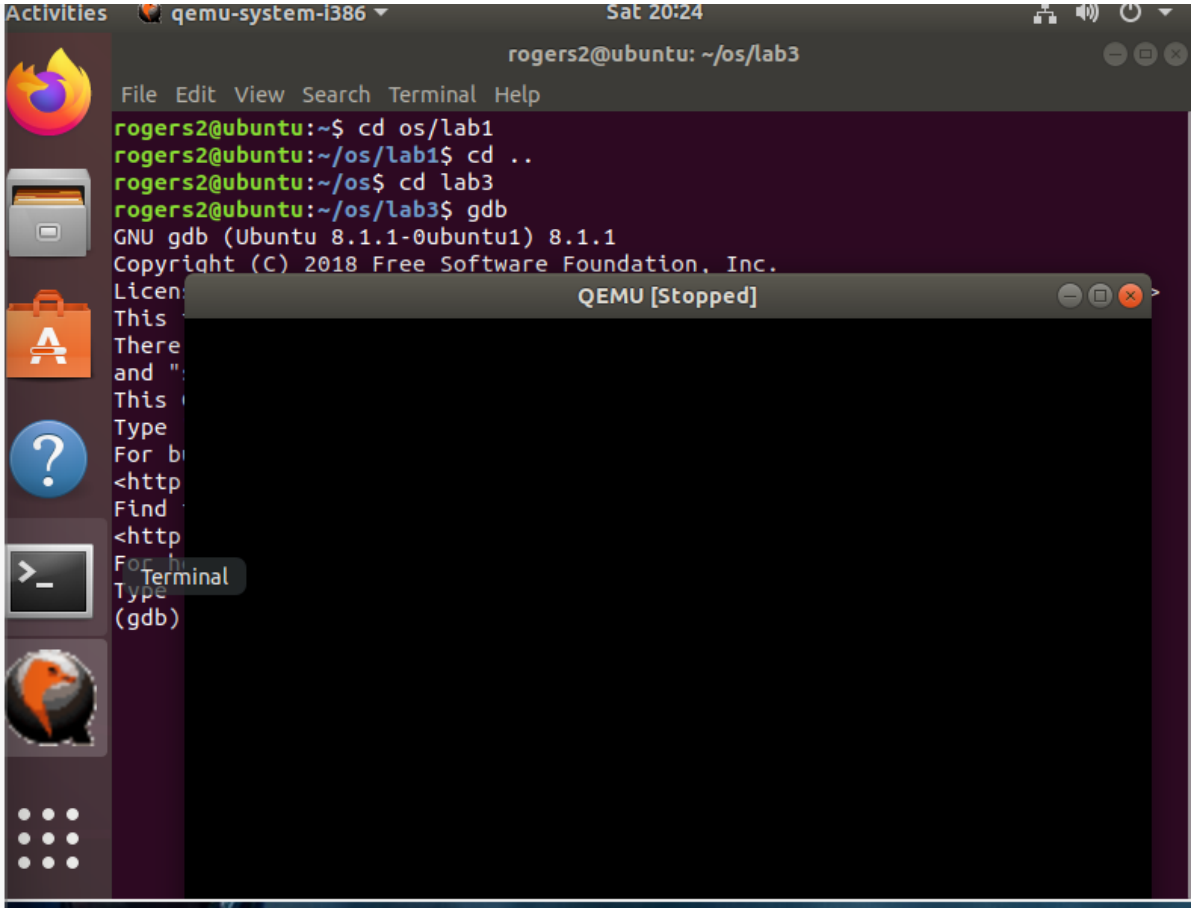
```
mov ss, eax
mov eax, VIDEO_SELECTOR
mov gs, eax
```

编写MBR.asm

与之前的相似，更改了硬盘读取函数的传参方式

gdb调试：

调用qemu后，没有显示内容



断点1：准备GDT，用lgdt指令加载GDTR信息。

根据实验资料；

为了让CPU知道GDT的位置，我们需要设置GDTR寄存器。回忆一下GDTR寄存器，其高32位表示GDT的起始地址，低16位表示GDT的界限。所谓界限，就是GDT的长度减去1。此时，我们已经放入5个段描述符，因此，GDT的界限如下所示。

$$界限 = 8 * 5 - 1 = 39$$

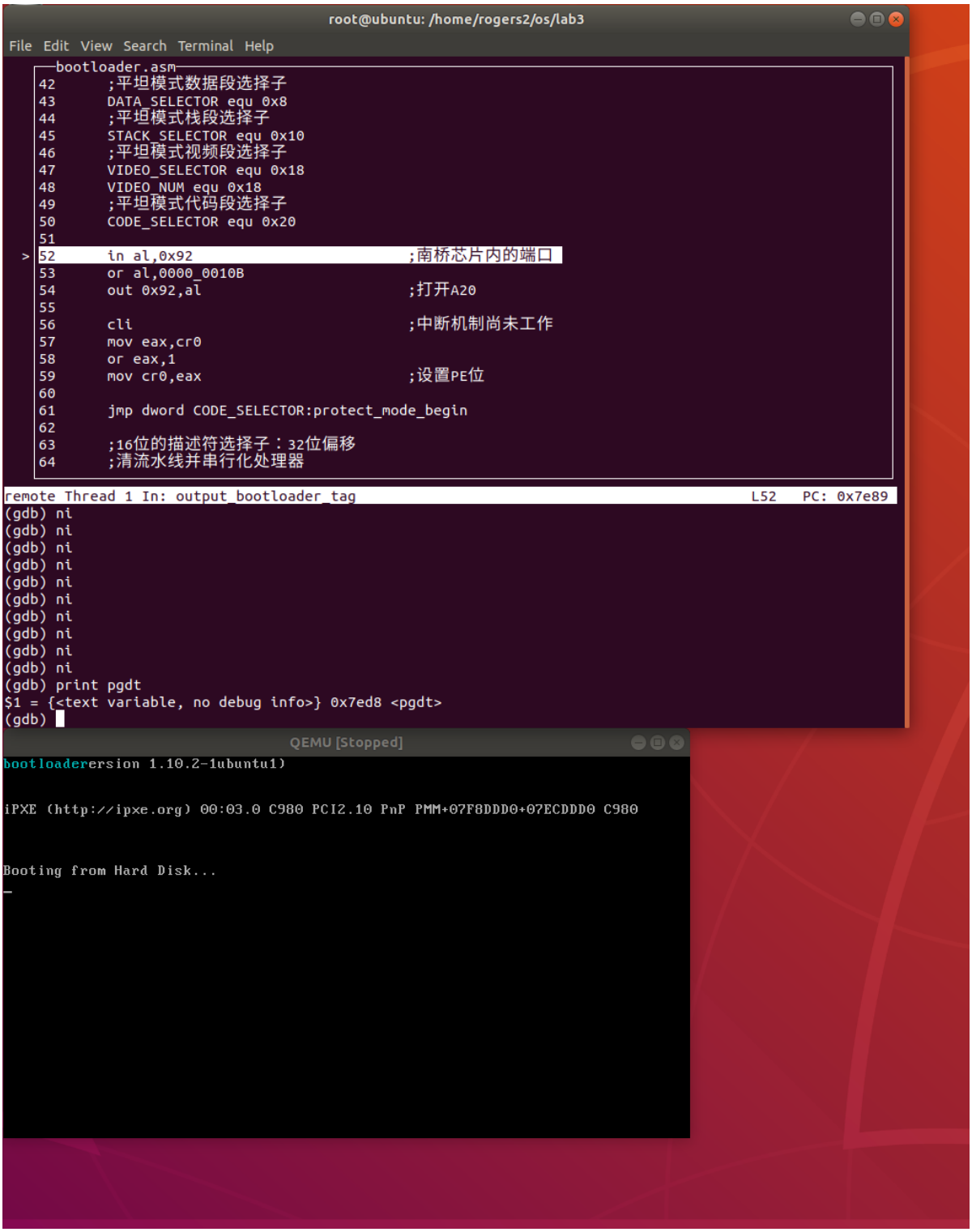
我们在内存中使用一个48位的变量来暂时存放GDTR的内容。

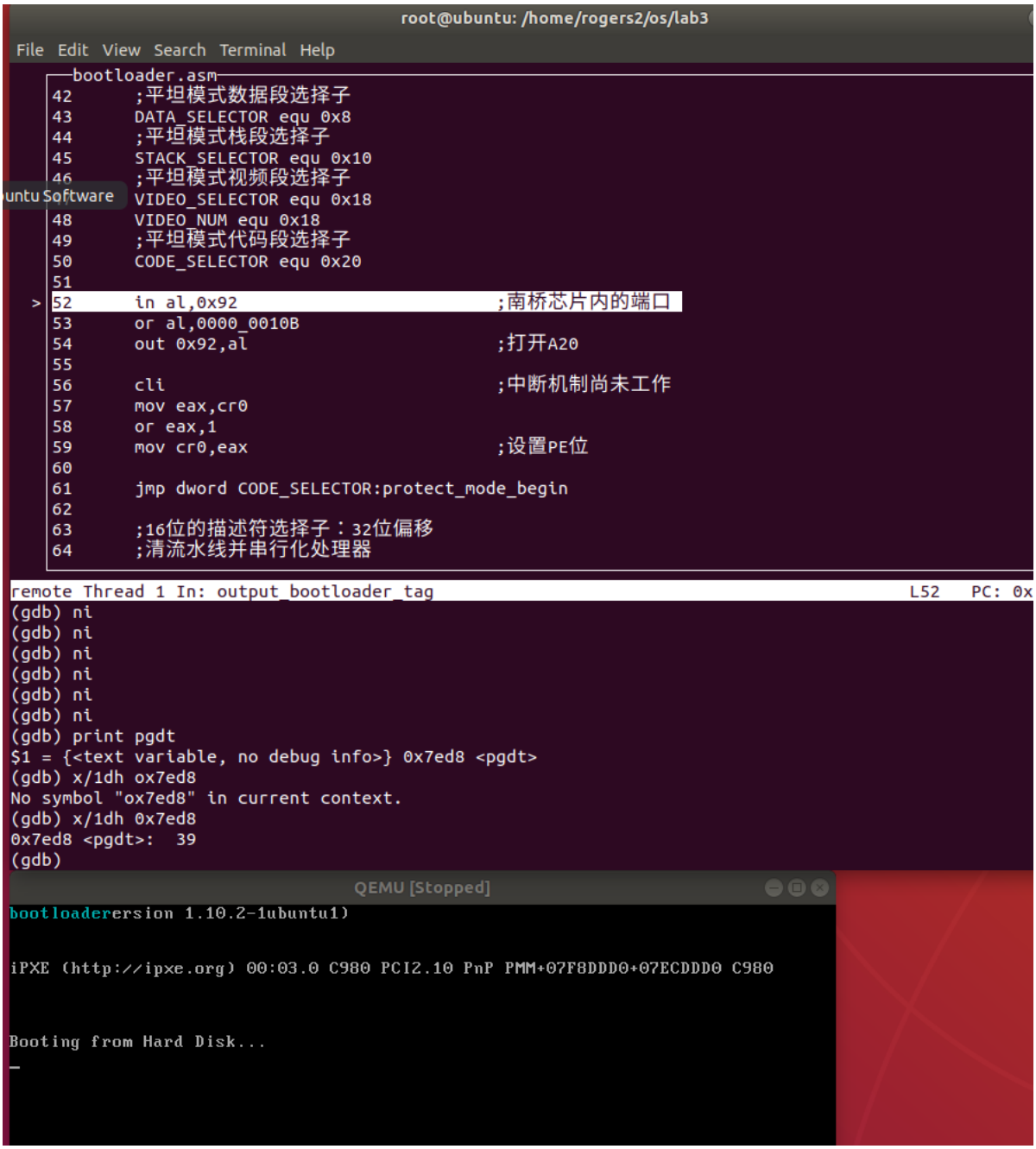
```
pgdt dw 0
    dd GDT_START_ADDRESS
```

然后把GDT的信息写入变量 `pgdt` ，把 `pgdt` 的内容加载进GDTR。

```
;初始化描述符表寄存器GDTR
mov word [pgdt], 39      ;描述符表的界限
lgdt [pgdt]
```

通过调试，先获取PGDT在内存的位置，然后查看里面的内容，确认是39


root@ubuntu: /home/rogers2/os/lab3
File Edit View Search Terminal Help
bootloader.asm
42 ;平坦模式数据段选择子
43 DATA_SELECTOR equ 0x8
44 ;平坦模式栈段选择子
45 STACK_SELECTOR equ 0x10
46 ;平坦模式视频段选择子
47 VIDEO_SELECTOR equ 0x18
48 VIDEO_NUM equ 0x18
49 ;平坦模式代码段选择子
50 CODE_SELECTOR equ 0x20
51
> 52 in al,0x92 ;南桥芯片内的端口
53 or al,0000_0010B
54 out 0x92,al ;打开A20
55
56 cli ;中断机制尚未工作
57 mov eax,cr0
58 or eax,1
59 mov cr0,eax ;设置PE位
60
61 jmp dword CODE_SELECTOR:protect_mode_begin
62
63 ;16位的描述符选择子：32位偏移
64 ;清流水线并串行化处理器
remote Thread 1 In: output_bootloader_tag L52 PC: 0x7e89
(gdb) ni
(gdb) ni
(gdb) ni
(gdb) ni
(gdb) ni
(gdb) ni
(gdb) ni
(gdb) ni
(gdb) ni
(gdb) print pgdt
$1 = {<text variable, no debug info>} 0x7ed8 <pgdt>
(gdb)
QEMU [Stopped]
bootloaderersion 1.10.2-1ubuntu1)
iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDD0+07ECDDD0 C980
Booting from Hard Disk...



断点2：打开第21根地址线。

怎么打开：参照以下资料将0x92第二位置为1

(2) 操作 System Control Port A，这种最常见。

MCA, EISA and other systems can also control A20 via port 0x92.

Bits 0,1,3,6,7 seem to have the same meaning everywhere this port is implemented.

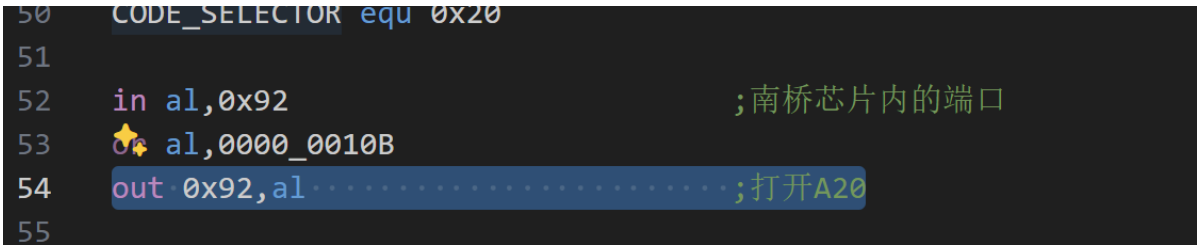
Bit 0 (w): writing 1 to this bit causes a fast reset (used to switch back to real mode; for MCA this took 13.4 ms).

Bit 1(rw): 0: disable A20, 1: enable A20.

Bit 3 (rw?): 0/1: power-on password bytes (stored in CMOS bytes 0x38-0x3f or 0x36-0x3f) accessible/inaccessible. This bit can be written to only when it is 0.

Bits 6-7 (rw): 00: hard disk activity LED off, 01,10,11: hard disk activity LED on.

Bits 2,4,5 are unused or have varying meanings. (On MCA bit 4 (r): 1: watchdog timeout occurred.)



```
remote Thread 1 In: output_bootloader_tag                                  L56   PC: 0x7e8f
Breakpoint 2, output_bootloader_tag () at bootloader.asm:54
(gdb) ni
(gdb) x/x1h 0x92
Invalid number "1h".
(gdb) x/x1dh 0x92
Invalid number "1dh".
(gdb) x/idh 0x92
0x92:   -4096
(gdb) info register al
al             0x2      2
(gdb) x/1xh 0x92
0x92:   0xf000
(gdb)
```

断点3：开启cr0的保护模式标志位。

```
55
56    cli                                    ;中断机制尚未工作
57    mov eax,cr0
58    or eax,1
59    mov cr0,eax                            ;设置PE位
```

修改前是16，修改后是17，16和17相比只有最后一位不一样，17说明进入保护模式了

```
remote Thread 1 In: output_bootloader_tag                                  L61   PC: 0x7e9a
(gdb) info register eax
eax            0x302    770
(gdb) x/1xh 0x92
0x92:   0xf000
(gdb) ni
(gdb) ni
(gdb) info register eax
eax            0x10     16
(gdb) ni
(gdb) ni
(gdb) info register eax
eax            0x11     17
(gdb)
```

断点4：远跳转，进入保护模式。

最后一步，远跳转进入保护模式。

```
jmp dword CODE_SELECTOR:protect_mode_begin
```

此时，jmp指令将 `CODE_SELECTOR` 送入cs，将 `protect_mode_begin + LOADER_START_ADDRESS` 送入eip，进入保护模式。然后我们将选择子放入对应的段寄存器。

```
61    jmp dword CODE_SELECTOR:protect_mode_begin
62
63    ;16位的描述符选择子：32位偏移
64    ;清流水线并串行化处理器
65    [bits 32]
66    protect_mode_begin:
67
68    mov eax, DATA_SELECTOR                    ;加载数据段(0..4GB)选择子
69    mov ds, eax
70    mov es, eax
71    mov eax, STACK_SELECTOR
72    mov ss, eax
73    mov eax, VIDEO_SELECTOR
74    mov gs, eax
```

可以查看跳转到保护模式时各个寄存器的状态

```
remote Thread 1 In: protect_mode_begin                                L68   PC: 0x7ea2
eax            0x11       17
ecx            0x0        0
edx            0x80       128
ebx            0x14       20
esp            0x7c00     0x7c00
ebp            0x0        0x0
esi            0x7ee8     32488
edi            0x0        0
eip            0x7ea2     0x7ea2 <protect_mode_begin>
eflags         0x6        [ PF ]
cs             0x20       32
ss             0x0        0
---Type <return> to continue, or q <return> to quit---
```