

# Lab 1 实验报告

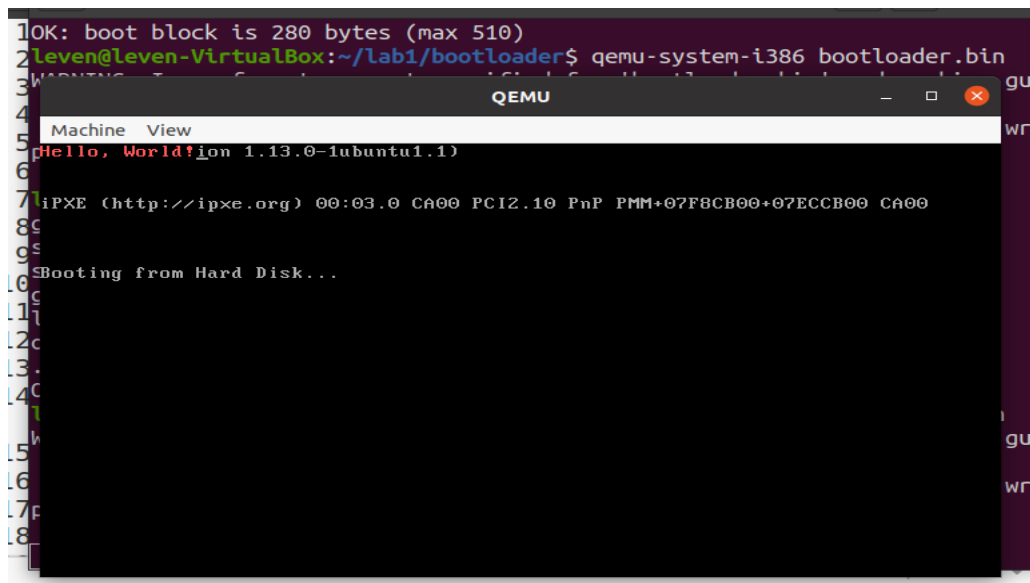
南京大学 人工智能学院

191300036 刘晔闻 [450008668@qq.com](mailto:450008668@qq.com)

实验进度：我完成了所有内容。

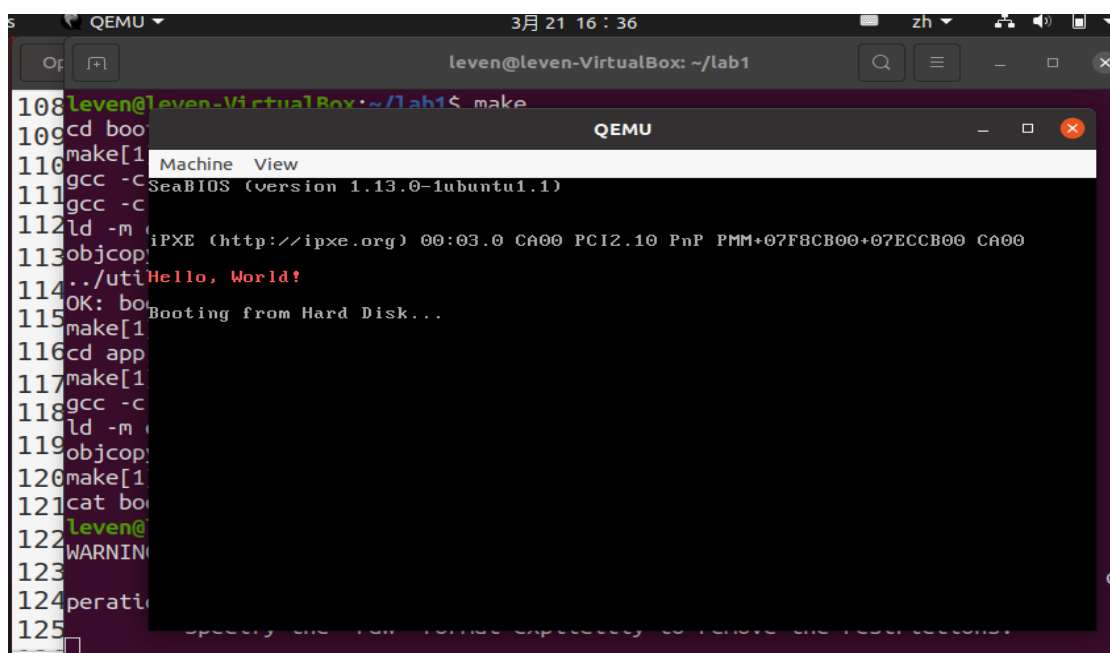
实验结果：

1. 实模式：



```
1 OK: boot block is 280 bytes (max 510)
2 Leven@Leven-VirtualBox:~/lab1/bootloader$ qemu-system-i386 bootloader.bin
3 WARNING: TSC not synced to host time. This may affect timing of operations.
4
5 Machine View
6 Hello, World!
7 iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8CB00+07ECCB00 CA00
8
9 Booting from Hard Disk...
```

2. 保护模式：



```
108 Leven@Leven-VirtualBox:~/lab1$ make
109 cd boot
110 make[1]: gcc -c
111 gcc -c
112 ld -m
113 objcopy
114 ../util/Hello, World!
115 OK: boot
116 make[1]: gcc -c
117 cd app
118 make[1]: gcc -c
119 ld -m
120 objcopy
121 cat boot
122 Leven@
123 WARNING:
124 operation
125
```

## 实验过程：

### 1. 实模式下实现 Hello, world! 程序：(bootloader/start.s)

（在 start.s 文件中已经注释掉了）

类似于 indexGuide.pdf 文件中的“简单上手”。

其主要思路为通过陷入屏幕中断调用 BIOS 打印字符串 Hello, world!（在 message: 中）。首先在 start: 中，先建立一个 13 字节的栈，然后将 message: 的地址填入栈中，然后调用 displayStr: 来打印字符串。之后通过 Makefile 中的 gcc 编译得到.o 文件，通过 ld 链接得到.elf 文件，再用 objcopy 命令减少程序大小，然后利用 utils/genboot.pl 生成一个 MBR，最后用 qemu 运行.bin 文件就得到上文中实模式下的结果。

### 2. 实模式切换保护模式：(bootloader/start.s 中)

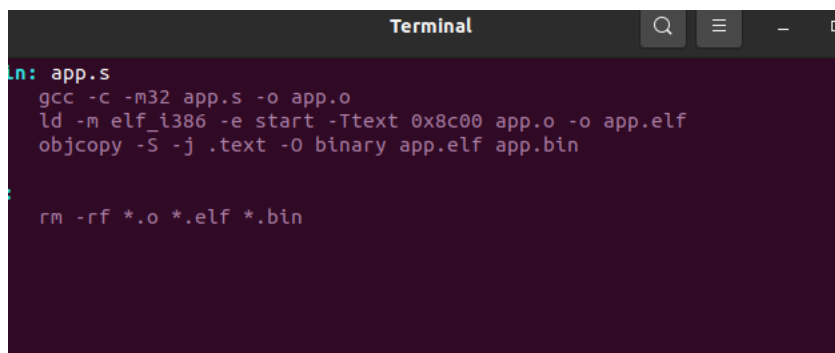
```
78 .global start
79 start:
80     movw %cs, %ax
81     movw %ax, %ds
82     movw %ax, %es
83     movw %ax, %ss
84     #TODO: Protected Mode Here
85     cli                                #关闭中断
86     inb $0x92, %al                     #启动A20总线
87     orb $0x02, %al
88     outb %al, $0x92
89     data32 addr32 lgdt gdtDesc          #加载GDTR
90     movl %cr0, %eax                    #启动保护模式
91     orb $0x01, %al                     #设置CR0的PE位（第0位）为1
92     movl %eax, %cr0
93     data32 ljmp $0x08, $start32        #长跳转切换至保护模式
94
95 _code32
```

其中部分代码框架中已给出。我填写了启动 A20 总线，启动保护模式和设置 CR0 的 PE 位（即保护位）为 1。启动 A20 是因

为在保护模式下，默认 CPU 访问第 20 位地址是无效的，只有启动 A20，CPU 才能连续访问地址。之后是调用 boot.c 中的 bootMain（）函数来打印字符。然后初始化 DS ES FS GS SS 初始化栈顶指针 ESP，设置 GDT 表项，其中代码段和数据段的基地址为 0x0，视频段的基地址为 0xb8000。

### 3. 加载磁盘中的程序并运行（bootloader/boot.c）

由 app/Makefile 得知，Hello, world! 入口地址为 0x8c00;



```
ln: app.s
gcc -c -m32 app.s -o app.o
ld -m elf_i386 -e start -Ttext 0x8c00 app.o -o app.elf
objcopy -S -j .text -O binary app.elf app.bin

rm -rf *.o *.elf *.bin
```

然后在 boot. 中的 bootMain（）函数中将 elf 函数指针指向这个入口地址即可。

### 实验心得：

1. utils/genboot.pl 文件中的代码没有完全搞懂……大概是与生成一个 MBR 有关的代码。

2.

```
}
displayStr:
    movl 4(%esp), %ebx
    movl 8(%esp), %ecx
    movl $((80*5+0)*2), %edi
    movb $0x0c, %ah
nextChar:
```

在 app/app.s 中可以调节最终打印 Hello, world! 的位置和颜色，如将 movb 后的 0x0c 改为 0x0d，打印出的字符颜色即为粉

红色。

