Oslab1 系统引导实验报告

一. 实验目的

1. 学习在 linux 环境下编写、调试程序，初步掌握 Shell、Vim、GCC、Binutils、Make、QEMU、GDB 的使用
2. 学习 AT&T 汇编程序的特点
3. 理解系统引导程序的含义，理解系统引导启动的过程

二. 实验设计

1. 程序的流程是：

（1）关中断、开 A20 地址线、加载 GDTR 寄存器

（2）设置 CR0 寄存器 PE 位为 1（表示进入保护模式）

（3）长跳转进入保护模式代码

（4）初始化 DS、SS、ES、FS、GS 这些段寄存器和 ESP

（5）跳转到 bootMain 中的代码

（6）加载 sector 1 to 0x8c00 程序然后跳转执行

（7）执行 hello，world

2. 已经完成的代码：

（1）80386 处理器从实模式切换到 32 位的保护模式；

（2）各个段寄存器以及栈顶指针 ESP 的初始化；

（3）加载存储在 MBR 之后的磁盘扇区中的程序到内存特定位置并跳转执行；

（4）生成 MBR 的 genboot.pl；

（5）整个框架的 makefile

3. 需要完成的任务：

（1）app.s 中具体的用户程序 hello，world 的实现；

（2）bootloader 中 start.s 里面被注释的跳转指令

4. 代码思路

（1）app.s 实现

在课程网站上提供了两种实现 app.s 的参考方案，一个是通过陷入屏幕中断调用 BIOS 打印字符串“Hello，World!”；另一个是通过写显存的方式打印字符。个人在本次实验选择了后者方案，具体实现截图如下

```
# TODO
//write memory from row 10 column 0 to row 10 column 12,using various colors by modifying the value of %ah
//show the right charactor by searching the ascii table and setting the corresponding value in %al
movl $((80*10+0)*2), %edi
movb $0x0a, %ah
movb $72, %al
movw %ax, %gs:(%edi)

movl $((80*10+1)*2), %edi
movb $0x0b, %ah
movb $101, %al
movw %ax, %gs:(%edi)

movl $((80*10+2)*2), %edi
movb $0x0c, %ah
movb $108, %al
movw %ax, %gs:(%edi)

movl $((80*10+3)*2), %edi
movb $0x0d, %ah
movb $108, %al
movw %ax, %gs:(%edi)

movl $((80*10+4)*2), %edi
movb $0x0e, %ah
movb $111, %al
movw %ax, %gs:(%edi)

movl $((80*10+5)*2), %edi
movb $0x0f, %ah
movb $44, %al
movw %ax, %gs:(%edi)

movl $((80*10+6)*2), %edi
movb $0x09, %ah
movb $32, %al
movw %ax, %gs:(%edi)
```

```
    movl $((80*10+7)*2), %edi
    movb $0x08, %ah
    movb $87, %al
    movw %ax, %gs:(%edi)

    movl $((80*10+8)*2), %edi
    movb $0x07, %ah
    movb $111, %al
    movw %ax, %gs:(%edi)

    movl $((80*10+9)*2), %edi
    movb $0x06, %ah
    movb $114, %al
    movw %ax, %gs:(%edi)

    movl $((80*10+10)*2), %edi
    movb $0x05, %ah
    movb $108, %al
    movw %ax, %gs:(%edi)

    movl $((80*10+11)*2), %edi
    movb $0x04, %ah
    movb $100, %al
    movw %ax, %gs:(%edi)

    movl $((80*10+12)*2), %edi
    movb $0x03, %ah
    movb $33, %al
    movw %ax, %gs:(%edi)

    ret
```

对于每个字符，设置字符的位置、颜色、ascii 码

再用 movw %ax，%gs:(%edi)写显存

每个字符即四行代码，一共 13 个字符“Hello， World!”

Ascii 分别为 72（H)、101(e)、108（l)、108(l)、111(o)、44(,)、32(space)、87(W)、111(o)、114(r)、108(l)、100(d)、33(!)

最后 ret 即可

（2）jmp 跳转指令补充

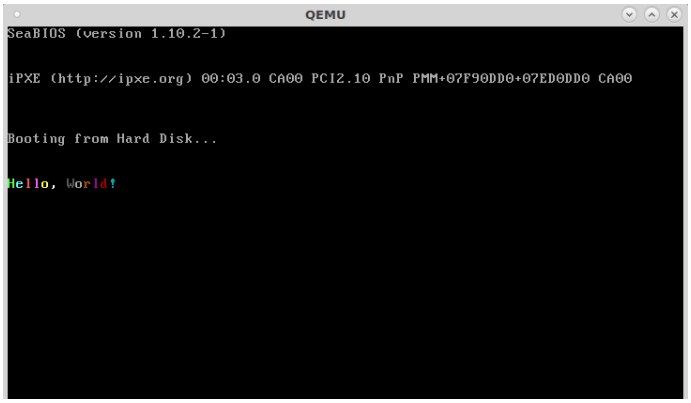
只要在 start.s 中注释的位置 jmp 到对应得函数即可，如下代码

```
        jmp bootMain
  # jmp to bootMain in boot.c
```

三. 实验结果

首先 chmod 777 打开 genboot.pl 的权限，然后就可以 make 编译

用 make play 运行得到以下运行结果截图



四. 实验心得

（1） 本次实验相对简单，但是也需要仔细的去阅读 manual 和网站上的 guide，了解实验基本原理，和实现要求

（2） 实验过程中遇到的第一个问题是无法 make 编译，找到原因是 permission denied，当我不知道如何处理的时候，通过留心群信息得知用 chmod 更改文件的访问权限即可。

（3） 遇到的第二个问题是没有添加 jmp 指令，通过仔细阅读框架代码，找到了 jmp 一条注释的信息，添加注释后成功得到结果！

五. 实验建议

暂无。