实验一 系统引导

161130118 尹浚宇
908664035@qq.com

实验目的

学习在 linux 系统下编写及调试程序，熟悉相关工具的使用.
学习 AT&T 风格汇编程序的特点.
理解系统引导程序的含义，理解系统引导的启动过程.

实验内容

在保护模式下加载磁盘中的 Hello World 程序并运行

程序设计思路

首先观察实验给出的框架代码，发现代码框架中 lab/bootloader/中包含的 start.s 已经完成了 80386 处理器从实模式到保护模式的切换，并已经完成了各段寄存及栈顶指针 ESP 的初始化. 同时，同一目录下的 boot.c 提供了将存储在 MBR 后的磁盘扇区中的程序加载至内存的特定位置并跳转执行的功能.
根据实验内容推断，我们需要完成的任务就是在 start.s 中编码，使程序跳转到 boot.c 里的 bootMain 函数执行，并且在磁盘的相应文件中写好 Hello World 程序.
首先考虑如何跳转到 bootMain 函数执行，这只需要在 start.s 的相应注释处插入一行汇编代码 calll bootMain 即可.

```
    movl $0x8000, %eax # setting esp
    movl %eax, %esp
# jmp to bootMain in boot.c
    calll bootMain
    pushl $13
    pushl $message
    calll displayStr
```

这是因为 boot.c 编译产生的可链接文件的符号表中，函数 bootMain 是全局强符号，可以在链接阶段被识别.
其次，我们需要在磁盘对应位置准备好 Hello World 程序. 这个位置对应的文件已经在框架代码中给出，为 app/app.s. 因为该程序需要在保护模式下运行，中断已经关闭，所以考虑使用写显存的方法来打印字符串. 观察 start.s 里提供的保护模式下直接打印字符串的代码，我们可以依葫芦画瓢地编写 app.s 来完成所需功能.
这里有几处改动，因为在执行完该程序后，我希望能够跳转回 start.s 之后的代码继续执行，所以需要维护好栈帧以便通过 ret 指令返回执行，下图是我的做法.

```
.global start
start:
        # TODO
        pushl %ebp
        movl %esp, %ebp
        pushl $13
        pushl $message
        calll displayStr
        leave
        ret
```

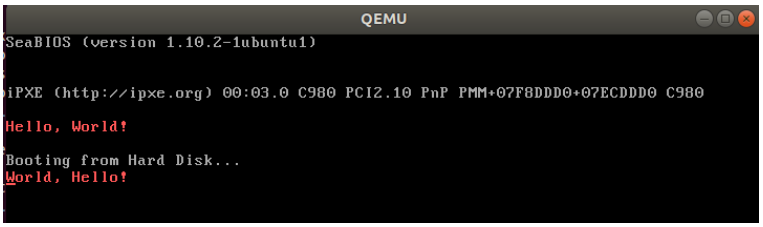
其次需要改动写显存的位置，不然跳转回 start.s 之后打印的字符串会覆盖 app.s 打印的字符

串，这里还略微改动了一下打印内容，来判断是否成功执行 app.s，改动如下图.

```
message:
        .string "World, Hello!\n\0"

movl $((80*8+0)*2), %edi # print at row 8 col 0
```

程序运行结果如下图所示:



实验心得

阅读 Makefile 文件有助于快速掌握程序框架及在终端中控制整个程序的命令.