Lab5 加载操作系统映像并进入C语言编写的main函数

PB15000102 王嵩超

make工具

The `make` utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them.

在shell内输入make命令，make会读取当前目录的makefile文件并开始处理第一条“规则”。规则的格式如下：

```
OS : prerequisite1 prerequisite2 …
    recipe...
prerequisite1: a b c
    recipe...
prerequisite2: d e f
    recipe...
prerequisite_ignore: ...
```

其中的第一条规则就是OS对应的一条。make会根据依赖关系处理prerequisite1、prerequisite2等条目。若依赖关系树没包含后面的某一条，这条规则默认就不会被执行。

在shell内输入make prerequisite_ignore即可处理未被依赖的条目。

综上，make OS的含义是处理名为OS的条目。

软盘内容的构成与制作过程

本次实验中使用了软盘内前两个扇区（2x512 Byte）。第一个扇区为启动代码（与前面的实验相同，会被BIOS加载入0x7C00处并跳转至此执行），启动代码在后期使用BIOS中断将第二个扇区的内容加载到0x7E00。并跳转至0x7E00处执行。

用两次dd工具分别将start16.bin、myOS.bin的内容写入软盘img映像文件的第一、第二扇区。

start16.bin包含start16.S（后被编译并**链接**为start16.**elf**）的二进制指令；myOS.bin可以理解为myOS.elf的memory dump。myOS.elf又是由start32.S（后被编译成start32.o）和main.c（后被编译成main.o）链接而成的。myOS.bin即是所谓的操作系统映像。

两个ld文件用来指明两个ELF文件的布局。

操作系统映像的加载

```
loadimage:
    /*  Driver Table:
        DL = 00h    1st floppy disk ( "drive A:" )
        DL = 01h    2nd floppy disk ( "drive B:" )
    */
    #reset
    movb $0, %dl
    movb $0, %ah
    int $0x13
    #read
    movw $0x7E00 , %bx
    movw $0       , %ax
    movw %ax      , %es

    movb $0x2, %ah
    movb $1  , %al #number of sectors to read
    movb $0  , %ch #track number track 0 is the first (outer-most) track on floppy or other cylindrical disks.
    movb $2  , %cl #sector number (1-17)
    movb $0  , %dh #head number
    movb $0  , %dl #drive number:0=A
    int $0x13
```

向各寄存器存入各种参数：

Register	Value
ah 程序编号，2是读数据的编号	2
al 要读入的扇区数	1
ch 要读入的磁道号	0
cl 要读入的扇区号（第二）	2
dh 要读入数据所在的磁头	0
dl 数据所在的驱动器（0表示第一软盘）	0

从汇编代码进入C代码

其实汇编以后，汇编码和C代码都变成了机器码，并通过ld脚本拼接在一起。我们要做的只是在汇编码末尾加一个跳转指令`jmp main`，跳转到main标签处。

将栈寄存器为0x18h，这在GDT表里对应的是0x0000。

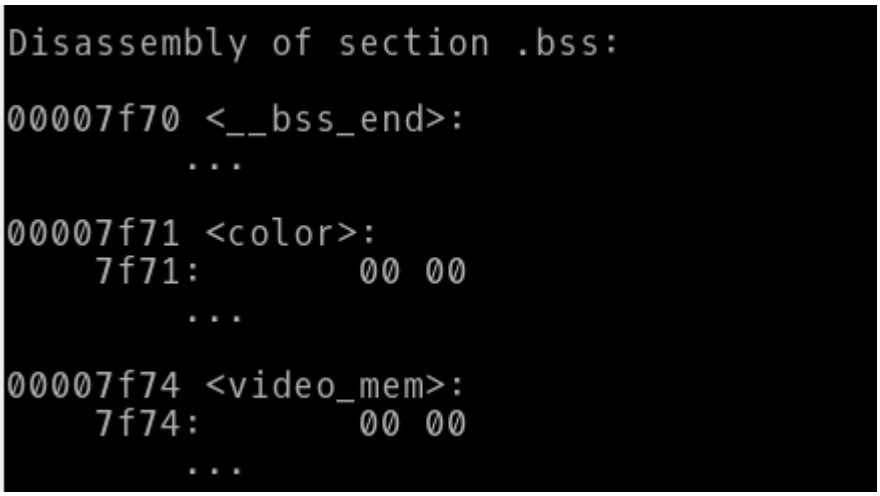
将栈指针设为0x2000h。

清空BSS段

BSS段（**bss** segment）通常是指用来存放程序中未初始化的全局变量的一块内存区域。汇编时，这些未初始化的变量的地址被计算成BSS段内的地址。

清空BSS段本是操作系统在运行时，操作系统的loader将内存中对应BSS段所在的地址清零的行为。这样做的好处之一是：避免BSS段所在处的原数据被错误使用（如作为指针值）。清零后再作为指针访问不会对其他数据产生破坏。

> 其实本实验中所写的main.c很可能不会出现bss段，因为所有可能有的全局变量都应初始化（如vga_memory=0xb800，也可以直接用宏定义vga_memory）。但为了强制生成bss段，我人为让color和video_mem成为未赋值的全局变量，在main函数内再赋值。用objdump得到的myOS.elf反汇编代码如下，可以看到bss段：



使用C语言写VGA显存区

原理与汇编类似。只需建立一个字符指针`char* p = 0xB800`。

按格式顺序解引用，赋值即可。

附加问题I：你能不能从C语言中调用汇编写的VGA输出函数？

可以，将汇编代码的函数所在段加上@function和其他相应标记，在C语言中用`extern`声明该函数，即可调用。

附加问题II：　如果你的main函数最后不是死循环，请说明main函数返回后你的操作系统在执行什么？

我的main函数末尾是死循环。

main函数开始时的指令：

```
    7e36:       55                      push   %ebp
    7e37:       89 e5                   mov    %esp,%ebp
```

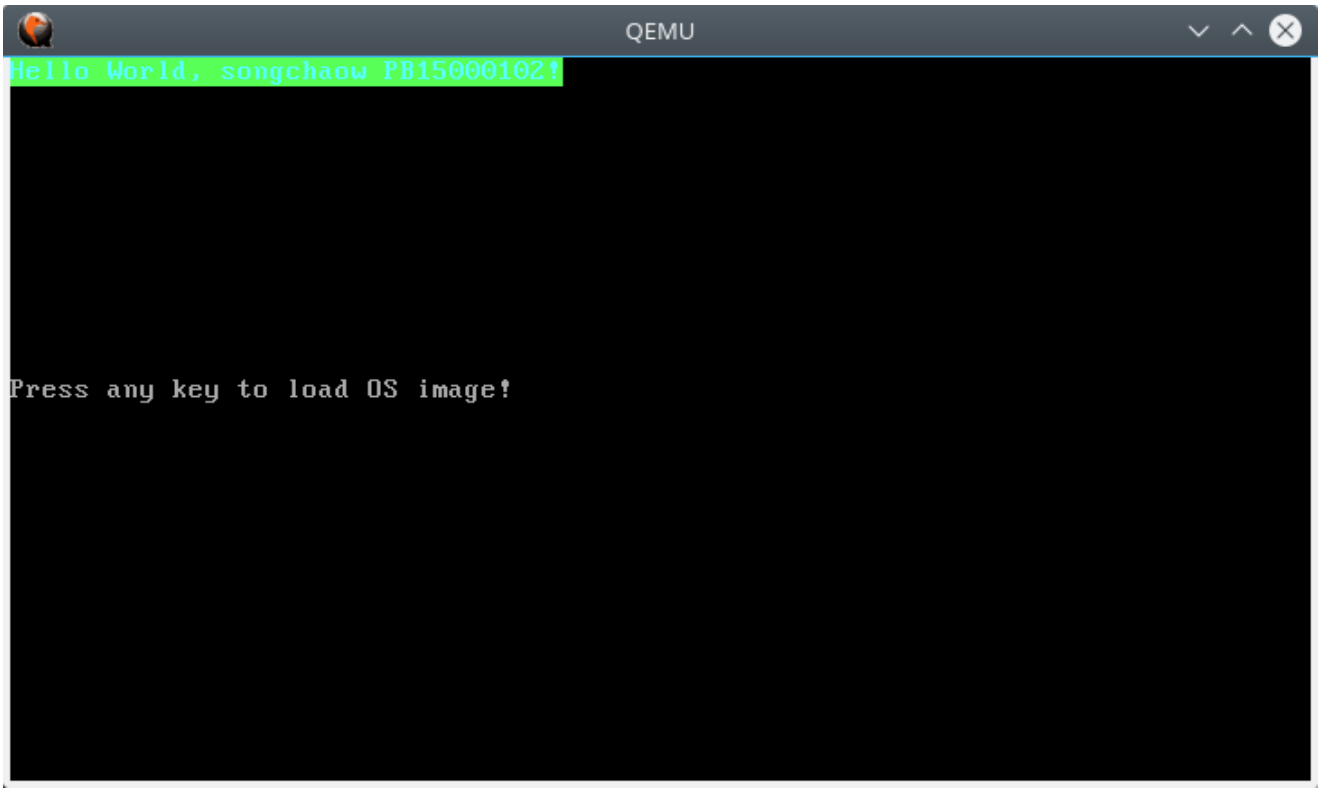
查看main函数结束时的指令：

```
0x7eae  jmp    0x7eae #死循环
0x7eb0  mov    (%esp),%eax
0x7eb3  ret
```

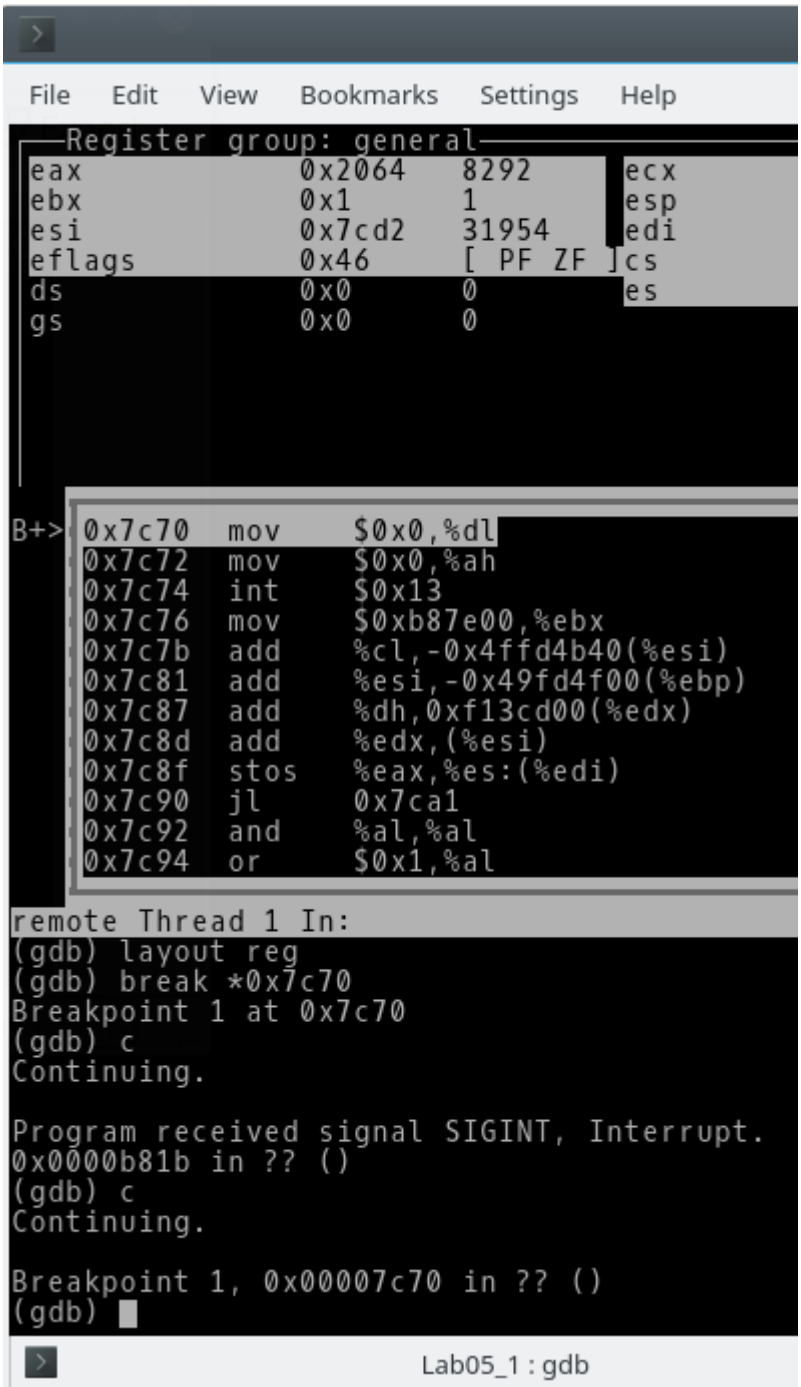
手动跳过死循环后，ret后eip会加载esp所指向的主调函数中的地址。然而本实验中没有主调函数（当时我们是直接跳转至main处），故eip会置为0，接着会导致重启。

运行截图

等待按键：


QEMU
Hello World, songchaow PB15000102!
Press any key to load OS image!

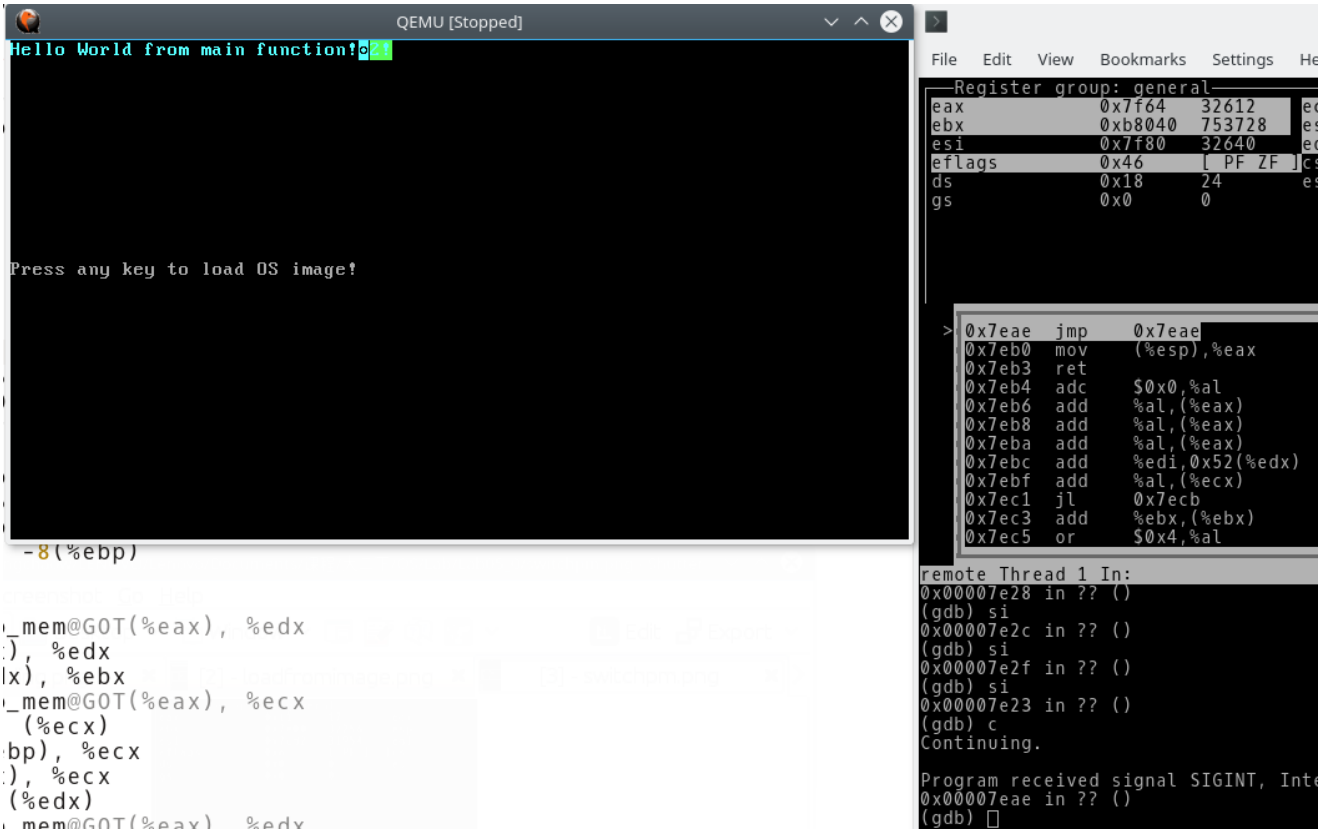
加载映像：


File Edit View Bookmarks Settings Help
Register group: general
eax 0x2064 8292 ecx
ebx 0x1 1 esp
esi 0x7cd2 31954 edi
eflags 0x46 [PF ZF] cs
ds 0x0 0 es
gs 0x0 0
B+> 0x7c70 mov $0x0,%dl
0x7c72 mov $0x0,%ah
0x7c74 int $0x13
0x7c76 mov $0xb87e00,%ebx
0x7c7b add %cl,-0x4ffd4b40(%esi)
0x7c81 add %esi,-0x49fd4f00(%ebp)
0x7c87 add %dh,0xf13cd00(%edx)
0x7c8d add %edx,(%esi)
0x7c8f stos %eax,%es:(%edi)
0x7c90 jl 0x7ca1
0x7c92 and %al,%al
0x7c94 or $0x1,%al
remote Thread 1 In:
(gdb) layout reg
(gdb) break *0x7c70
Breakpoint 1 at 0x7c70
(gdb) c
Continuing.
Program received signal SIGINT, Interrupt.
0x0000b81b in ?? ()
(gdb) c
Continuing.
Breakpoint 1, 0x00007c70 in ?? ()
(gdb)
Lab05_1 : gdb

切换至保护模式：


Register group: general
eax 0x11 17 ecx
ebx 0x7e00 32256 esp
esi 0x7cd2 31954 edi
eflags 0x6 [PF]]cs
ds 0x0 0 es
gs 0x0 0
0x7c91 mov %cr0,%eax
0x7c94 or $0x1,%al
0x7c96 mov %eax,%cr0
> 0x7c99 ljmp $0x1bb,$0x87e00
0x7ca0 add %dh,0x3cac10cd(%esi,%ecx,1)
0x7ca7 add %dh,-0xc(%ebp)
0x7caa ret
0x7cab mov $0x7c0200,%edi
0x7cb0 add %dl,0x72(%eax)
0x7cb3 gs jae 0x7d29
0x7cb6 and %ah,0x6e(%ecx)
0x7cb9 jns 0x7cdb
remote Thread 1 In:
(gdb) c
Continuing.
Breakpoint 3, 0x00007c8c in ?? ()
(gdb) si
0x00007c91 in ?? ()
(gdb) si
0x00007c94 in ?? ()
(gdb) si
0x00007c96 in ?? ()
(gdb) si
0x00007c99 in ?? ()
(gdb)

写显存完毕后的死循环：


QEMU [Stopped]
Hello World from main function!
Press any key to load OS image!
File Edit View Bookmarks Settings
Register group: general
eax 0x7f64 32612
ebx 0xb8040 753728
esi 0x7f80 32640
eflags 0x46 [PF ZF]
ds 0x18 24
gs 0x0 0
0x7eae jmp 0x7eae
0x7eb0 mov (%esp),%eax
0x7eb3 ret
0x7eb4 adc $0x0,%al
0x7eb6 add %al,(%eax)
0x7eb8 add %al,(%eax)
0x7eba add %al,(%eax)
0x7ebc add %edi,0x52(%edx)
0x7ebf add %al,(%ecx)
0x7ec1 jl 0x7ecb
0x7ec3 add %ebx,(%ebx)
0x7ec5 or $0x4,%al
remote Thread 1 In:
0x00007e28 in ?? ()
(gdb) si
0x00007e2c in ?? ()
(gdb) si
0x00007e2f in ?? ()
(gdb) si
0x00007e23 in ?? ()
(gdb) c
Continuing.
Program received signal SIGINT, Inte
0x00007eae in ?? ()
(gdb)