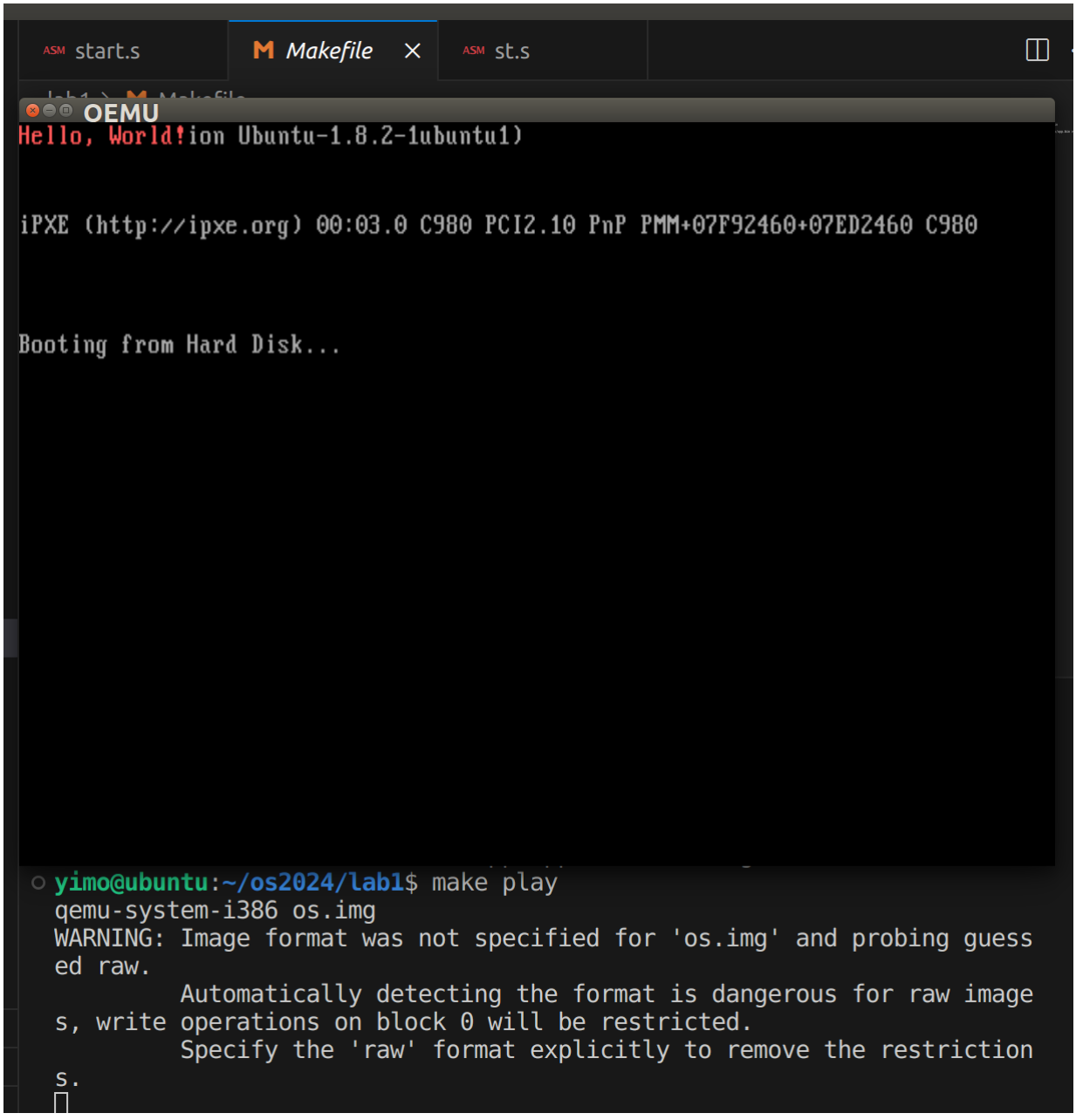
Lab1实验报告

姓名	殷子珺
学号	22122016
邮箱	221220016@smail.nju.edu.cn

实验进度：我完成了所有内容

1.1 实模式下打印 Hello，World!




```
# TODO:通过中断输出Hello World
    pushw $13 # pushing the size to print into stack, the second para
    pushw $message # pushing the address of message into stack, the first para
```

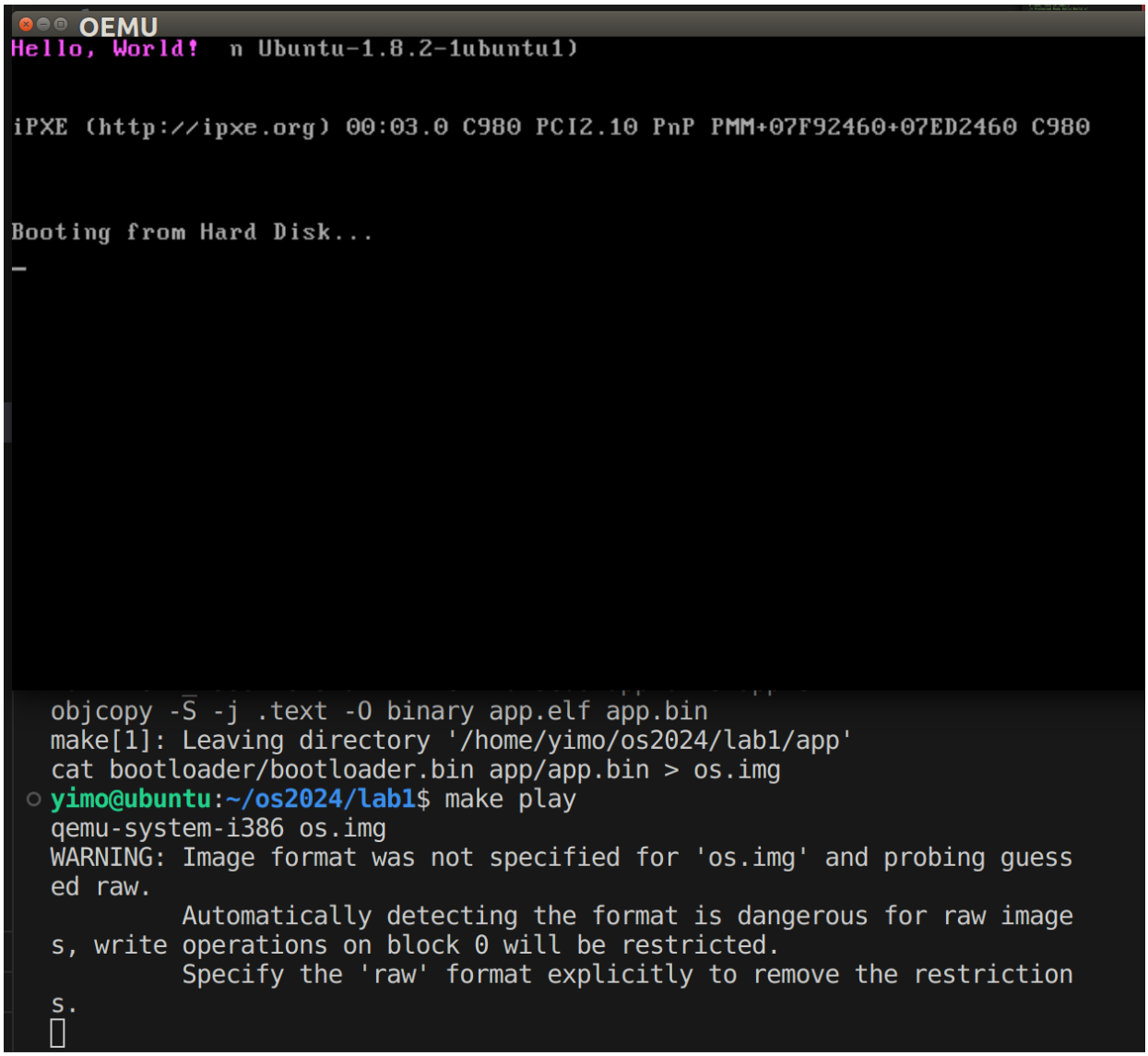
```
    callw displayStr

    displayStr:
    pushw %bp
    movw 4(%esp), %ax #use the first para
    movw %ax, %bp
    movw 6(%esp), %cx #use the second para
    movw $0x1301, %ax
    movw $0x000c, %bx
    movw $0x0000, %dx
    int $0x10
    popw %bp
    ret
```

在 `displayStr` 中执行 `int $0x10` 调用之前，在寄存器中准备调用参数，观察到函数的第一个参数存入 `%bp` 寄存器，则第一个参数应该传入字符串地址； 第二个参数存入 `%cx` 寄存器，则第二个参数应该传入字符串长度。

显示字符串(适用AT)	ES:BP=串地址	
	CX=串长度	
	DH,DL=起始行,列	
	BH=页号	光标返回起始位置
	AL=0,BL=属性	
	串:char,char,...	光标跟随移动
	AL=1,BL=属性	
	串:char,char,...	光标返回起始位置
	AL=2	
	串:char,attr,char,attr,...	光标跟随移动
	AL=3	
	串:char,attr,char,attr,...	

1.2 保护模式下打印Hello，World!



切换到保护模式后，不能再使用BIOS中断在屏幕上输出信息；并且在刚切换到保护模式的时候，为了防止执行实模式下的中断指令导致未定义行为，会关闭中断直到模式切换完成，所以也没有办法使用保护模式下的中断指令实现字符串打印。

计算机启动时会将视频段的内容显示在终端上。利用这个特性将 `Hello, World!` 的字符ASCII码以及字符属性数据存到视频段地址上，即可实现在开机时显示。

立即数包含两个字节，第一个字节给出字符ASCII码，第二个字节给出字符显示属性

Bit(s)	Value
0-7	Ascii code
8-11	Foreground color (字体颜色)
12-14	Background color(背景颜色)
15	Blink(是否闪烁)

我第二个字节设置为 `0x0d` ： 字体颜色为粉色，背景色为黑色（与终端窗口保持一致），不闪烁

```
    movw $0x0d48, 0xb8000
    movw $0x0d65, 0xb8002
    movw $0x0d6c, 0xb8004
    movw $0x0d6c, 0xb8006
    movw $0x0d6f, 0xb8008
    movw $0x0d2c, 0xb800a
    movw $0x0d20, 0xb800c
    movw $0x0d57, 0xb800e
    movw $0x0d6f, 0xb8010
    movw $0x0d72, 0xb8012
    movw $0x0d6c, 0xb8014
    movw $0x0d64, 0xb8016
    movw $0x0d21, 0xb8018
    movw $0x000a, 0xb801a
    movw $0x0000, 0xb801c

    '''

    gdt: # 8 bytes for each table entry, at least 1 entry
    # .word limit[15:0],base[15:0]
    # .byte base[23:16],(0x90|(type)),(0xc0|(limit[19:16])),base[31:24]
    # GDT第一个表项为空
    .word 0,0
    .byte 0,0,0,0

    # TODO: code segment entry
    .word 0xffff,0
    .byte 0,0x9a,0xcf,0

    # TODO: data segment entry
    .word 0xffff,0
    .byte 0,0x92,0xcf,0

    # TODO: graphics segment entry
    .word 0xffff,0x8000
    .byte 0x0b,0x92,0xcf,0

gdtDesc:
    .word (gdtDesc - gdt -1)
    .long gdt
```

还有一部分需要完成的内容是补全 `gdt`

- `limit`：20位补为全1
- `base`：扁平模式下代码段和数据段的基地址为0x0，视频段基地址为给出的 `0xb8000`
- `type`
  - Expand-down: 数据的起始地址在高地址处，向低地址扩展，一般在栈上会用
  - Accessd: 标记数据段是否被访问到，一开始初始化为0，在程序运行过程中由处理器来更改这一位
  - Comforming: 标记一段代码是否可以在系统级和用户级别运行而不造成上下文的切换

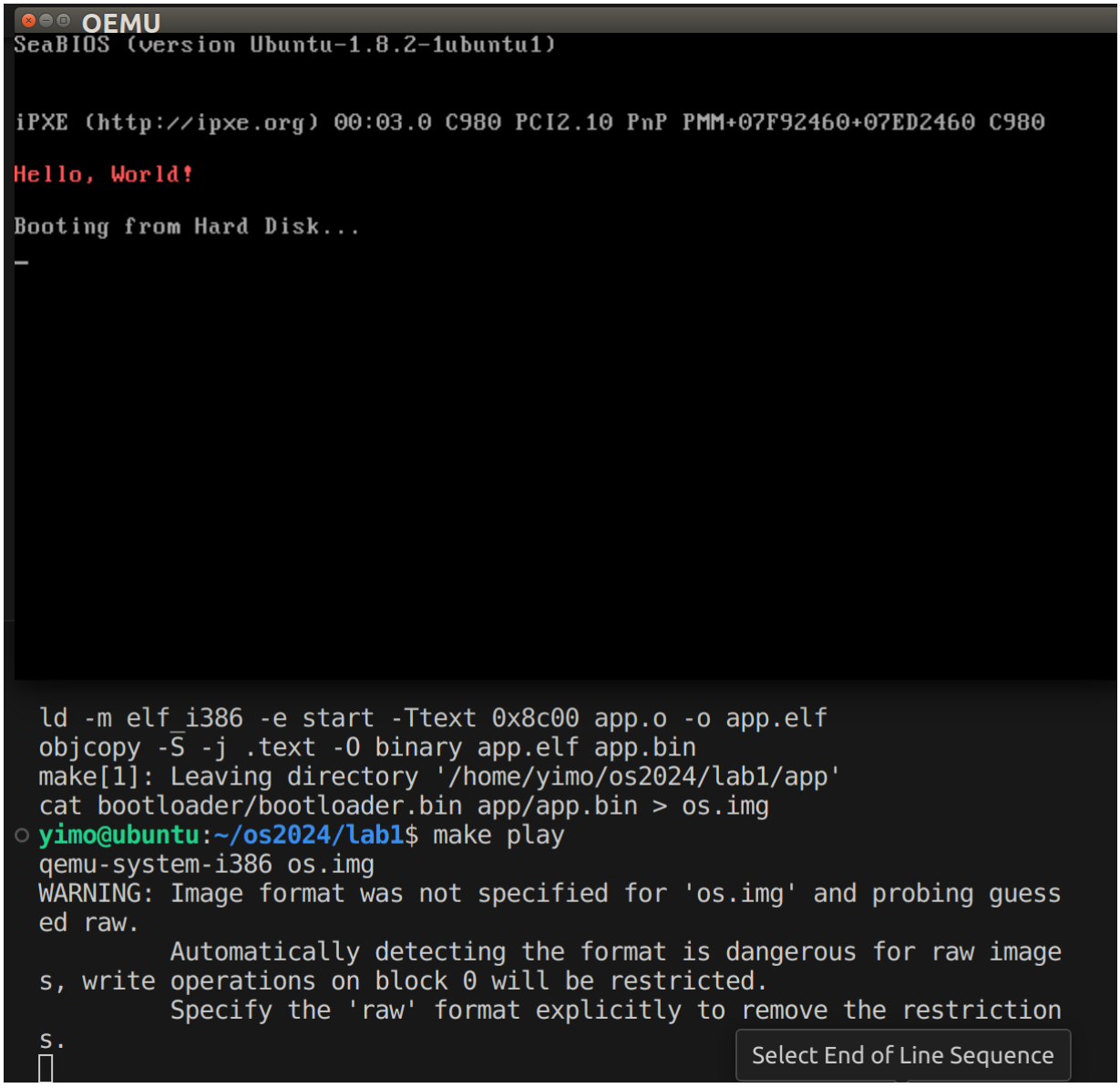
- TYPE：当 S 为 1，TYPE 表示的代码段，数据段的各种属性 如下表所示

bit 3	Data/Code	0 (data)
bit 2	Expand-down	0 (normal) 1 (expand-down)
bit 1	Writable	0 (read-only) 1 (read-write)
bit 0	Accessed	0 (hasn't) 1 (accessed)

bit 3	Data/Code	1 (code)
bit 2	Conforming	0 (non-conforming) 1 (conforming)
bit 1	Readable	0 (no) 1 (readable)
bit 0	Accessed	0 (hasn't) 1 (accessed)

  - `0xA`: Code, Execute/Read，`0x2`: Data, Read/Write

1.3 磁盘中加载程序运行



在app的Makefile文件中，`ld -m elf_i386 -e start -Ttext 0x8c00 app.o -o app.elf` 说明对程序进行链接时，代码文件是从内存地址为 `0x8c00` 处开始的。为执行这一段代码，首先需要利用 `readSect` api读取磁盘第1块扇区内容并加载到dst地址，再利用函数指针指向该地址并实现函数调用。

```
void bootMain(void)
{
    /**
     * This code represents the bootloader's entry point.
     * It reads a sector from a specific memory address (0x8c00) into the
destination (dst) buffer.
     * Then, it sets the entry point of the bootloader to the address 0x8c00 and
jumps to it.
     */
    void *dst = (void *)0x8c00;
    readSect(dst, 1);
    void (*entry)(void) = (void *)0x8c00;
    entry();
}
```

运行

在lab1根目录下执行

```
make
make play
```