操作系统 lab1 实验报告

171830638 陈泓宇

171830638@smail.nju.edu.cn

1.相关名词解释

CPU：中央处理器（central processing unit），计算机系统的运算和控制核心，是信息处理、程序运行的最终执行单元
内存：memory，用于暂时存放 CPU 中的运算数据，以及与硬盘等外部存储器交换的数据
BIOS：基本输入输出系统（Basic Input Output System），主要执行系统开机自检和自启动，将 MBR 加载到 0x7c00，位于内存中的上位内存（640KB—1024KB）
磁盘：计算机主要的存储介质
主引导扇区：Master Boot Record（MBR），磁盘的 0 号柱面、0 号磁头、0 号扇区所对应的扇区，大小为 512 字节，末尾两字节为 0x55 和 0xaa。在开机自检后被 BIOS 加载到 0x7c00（使用 0x7c00 是为了与 Intel 的第一代个人电脑芯片 8088 保持兼容）。功能是将操作系统的代码和数据从磁盘加载到内存，以及跳转到操作系统的起始地址
加载程序：在本实验中，加载程序为 MBR 的指令；在现代操作系统中，加载程序除了 MBR 的指令，还包括 GNU GRUB 等第二阶段引导程序
操作系统：Operating System，承接计算机上层的软件和底层的硬件的程序，也是计算机系统的内核与基石

2.实模式与保护模式辨析

2.1 实模式：与保护模式最主要区别在于 20 位的地址总线，寻址能力为 1MB
AX 之类的通用寄存器和段寄存器（CS、DS、SS、ES）仅有 16 位。因为空间太小，只在最初启动时使用
具有中断向量表，位于 0x0000 到 0x03ff，可以在 MBR 中被调用
物理地址=段寄存器<<4+偏移地址（16 位）
2.2 保护模式：32 位地址总线，将寻址能力扩大到 4GB
PC 在启动时进入实模式，并由 Bootloader 完成向保护模式的切换
通用寄存器扩展为 32 位，段寄存器维持 16 位
2.2.1 寻址方式的变化：
分段机制：从段寄存器中取出段选择子+从 GDTR 中取出 GDT 的首地址->段首地址
段首地址+偏移量->线性地址（在本实验中也即物理地址）

3.实验的实现过程

Task1：在实模式下在终端中打印出“Hello, World!”
在将 stack 初始化后，将字符串变量 message 入栈，最后调用 int $0x10 打印
这部分代码在 start.s 中实现

Task2：将实模式切换到保护模式，并在保护模式下在终端中打印出“Hello, World!”
在.code16 中以实模式启动 A20 总线，启动保护模式，设置 CR0 的 PE 位，最后通过 ljmp 指令跳转到保护模式
（注意：此处若不打开 A20 总线将无法进行 32 位寻址）
在保护模式中，完成对段寄存器和 stack 的初始化后，跳转到 bootMain 函数，开始将主引导扇区的内容加载到 0x8c00
如下为对段寄存器（ds、es、ss、fs、gs）和 stack 的初始化：

```
/*here the high 13 bits show the number of segment
in the gdt and set the TI and RPL as 0*/
movw $(2<<3), %ax
movw %ax, %ds
movw %ax, %es
movw %ax, %ss
movw %ax, %fs
movw $(3<<3), %ax
movw %ax, %gs
/*set stack of 16MB and initialize ebp and esp*/
movl $0, %ebp
movl $(128<<20), %esp
```

通过观察 GDT 表，可以看到代码段、数据段、视频段分别为 GDT 的第 1、2、3 项；由有段选择子的结构可知：
cs 寄存器由 ljmp 指令设置
ds 寄存器的数据段选择子为全局段（TI=0），特权级 PRL 为 0，编号 index 为 2；es、fs 寄存器做同样的设置
gs 寄存器的数据段选择子为全局段（TI=0），特权级 PRL 为 0，编号 index 为 3
ebp 置零，由于需要为准备参数和 ebp 预留空间，将 esp 置为（128<<20）
对 cs、ds、gs 描述符的解释：

	段基址	段限长	G	D/B	AVL	P	DPL	S	TYPE	A
代码段	0x0	0xFFFFFFFF	0x1	0x1	0x0	0x1	0x00	0x1	0x101	0x0
数据段	0x0	0xFFFFFFFF	0x1	0x1	0x0	0x1	0x00	0x1	0x001	0x0
视频段	0xb8000	0xFFFFFFFF	0x1	0x1	0x0	0x1	0x00	0x1	0x001	0x0

（G=1 表示最大的段为 4GB，D/B=1 表示地址和数据为 32 位，P=1 表示该段已在主存中，DPL=0 表示只有内核态才可以访问，S=1 表示是普通的代码段或数据段描述符，TYPE=0x101 表示只读代码段，TYPE=0x001 表示只读数据段，A=0 表示未被访问过）
此时已经切换到保护模式，不能再依靠中断处理来进行打印，于是考虑应用 VGA 模式，即在 gs 段设置需要打印的字符参数：

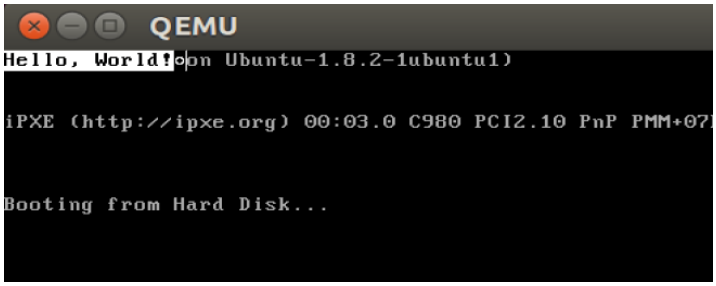
```
/* task2: write the charactor 'H' under Protect Mode*/
/*movl $((80*5+0)*2), %edi #在第5行第0列打印
movb $0x0c, %ah #黑底红字
movb $0x48, %al #48为H的ASCII码
movw %ax, %gs:(%edi) #写显存*/
```

（以上以一个‘H’字符为例）
此后正常跳转到 bootMain 函数

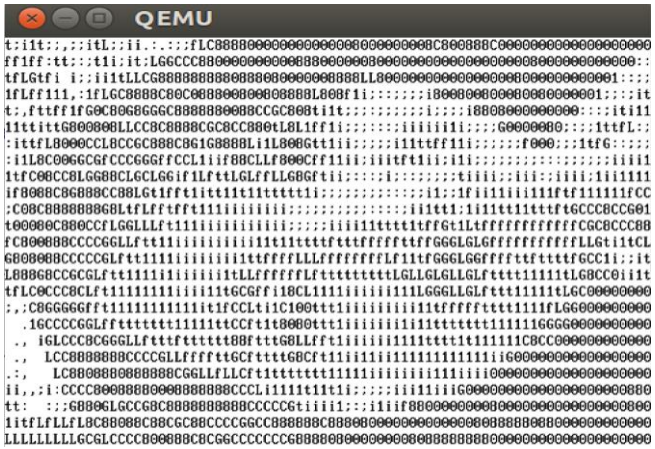
Task3：在保护模式下加载磁盘中的 Hello World 程序
切换保护模式和段设置与 Task2 相同，这一实验中主要在 bootMain 函数中，将磁盘内容加载到 0x8c00 并从 0x8c00 开始执行

```
void bootMain(void) {
    int i = 0;
    void (*elf)(void);
    elf = (void(*)(void))0x8c00;
    for (i = 0; i < 200; i++) {
        readSect((void*)(elf + i*512), 1+i);
    }
    // readSect((void*)elf, 1); // loading sector 1 to 0x8c00
    // TODO: jumping to the loaded program (DONE)
    elf();//here elf() point at the address 0x8c00
}
```

（加载过程调用的 readSect 函数调用的 outByte 函数，内联汇编了 out 指令）
这一函数模拟了将地址 0x8c00 开始的 200 个 512 字节从磁盘读取到内存，模拟的磁盘中的内容为 app.s 中的指令，即一个 Hello World 程序
这一程序从地址 0x8c00 开始，这是由编译文件 Make file 决定的，不过虽然地址的值相同，但不能不经过读取直接从 0x8c00 执行
elf 为函数指针，所以在经过加载后，直接执行这一地址开始的第一条指令
执行结果为：



在将 app.s 中的 message 变量替换为 ascpic 并修改字符串长度参数后，得到的字符画为：



以上即完成了本次实验的全部内容