操作系统实验

Lab1：

I386 实模式（1.1）、保护模式（1.2）、保护模式下加载磁盘中 Hello World 程序（1.3）

院系：人工智能学院

姓名：石睿

学号：211300024

班级：操作系统-2023 春季学期

邮箱：211300024@smail.nju.edu.cn

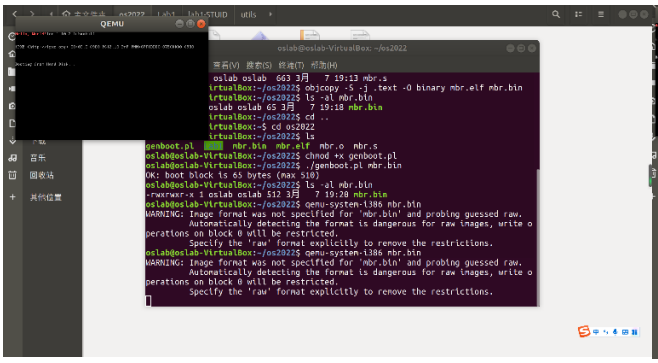
实验时间：2022.3.9

一、 实验进度

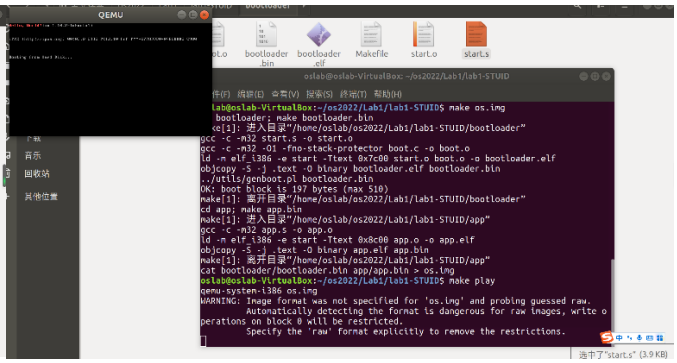
我已经完成了 Lab1 的所有内容。

二、 实验结果

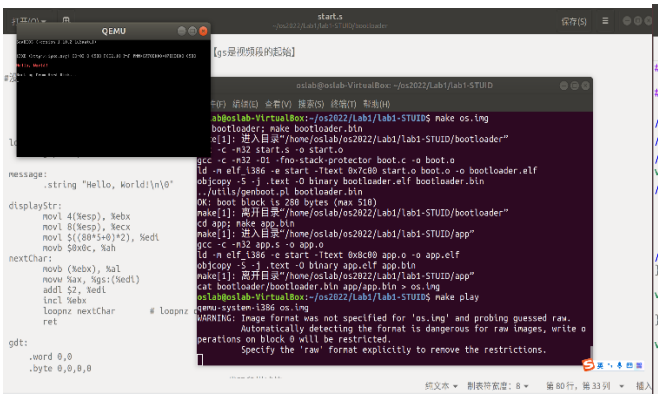
为节省报告的页数，把三个阶段的成果并排摆放。在各个阶段都可以正确在 qemu 中运行出 Hello World 呢！结果如下图所示，具体实现过程在第三部分介绍。



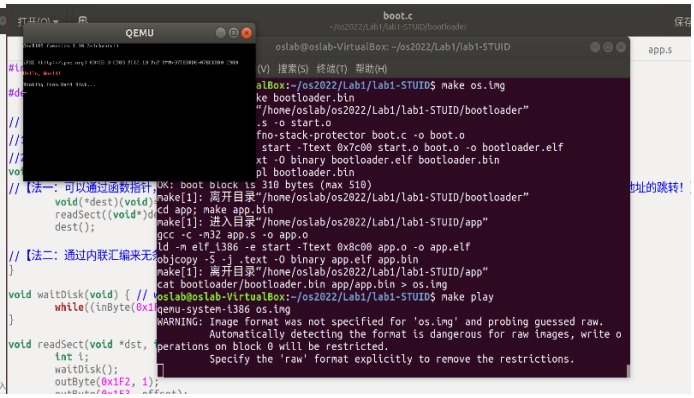
Lab1.1： objcopy 缩小 mbr 程序



Lab1.：make 命令完成编译链接



Lab1.2：保护模式下运行 Hello World



Lab1.3：保护模式下加载 OS（app）

三、 实验修改的代码位置

—简单描述为完成本次实验，修改或添加了那些代码，大致的文件和函数定位即可

① Lab1.1

去掉注释、通过 objcopy 缩小 mbr 程序，通过脚本程序 genboot.pl 生成 MBR 文件即可。

② Lab1.2

2.1 开启 A20 地址线—提升可访问的内存空间

A20 是一个 OR 逻辑电路门，被放置在第 20 位的地址总线上。经查阅资料，共有三种方式开启 A20 地址线：键盘控制器、调用 BIOS 功能、使用系统端口 0x92，本实验采用最后一种方式。0x92 端口的 bit1：为 0 时关闭 A20 总线，为 1 时开启 A20 总线。故做出如下修改。

```
80      inb $0x92,%al        #【TODO】
81      orb $0x02,%al        #【端口倒数第二位设置成1】
82      outb %al,$0x92
83                           #启动A20总线，开启A20的地址线【以上三条均是】
```

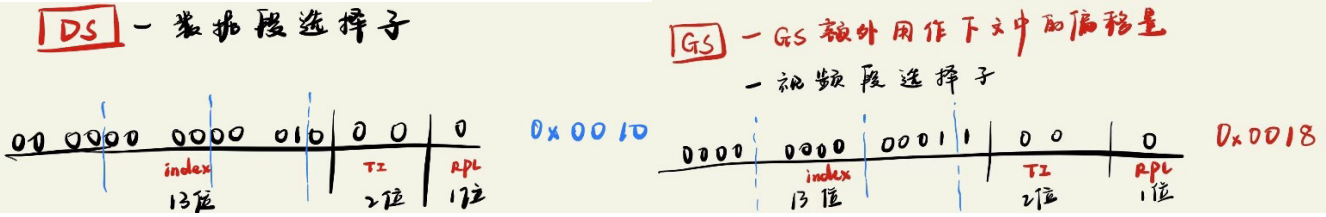
2.2 启动保护模式—从实模式转变为保护模式

通过将 CR0 中的 PE 位设置为 1,来开启保护模式。其中 PE 是 CR0 的最后一位。

```
87      movl %cr0,%eax          #【TODO】
88      orl  $0x00000001,%eax
89      movl %eax,%cr0
90                              #启动保护模式【CR0是32位的】，即设置CR0的PE位（第0位）为1【PE从0设置到1】
```

2.3 初始化 DS ES FS GS SS 寄存器

I386 中段寄存器在保护模式中为使用在 GDT 上的段选择子，其构成为 index+TI+RPL，其中 index 是本.s 文件中新定义的全局描述符中的下标位置。针对 DS 和 GS 的初始化如下图所示（ES FS SS 没有用到，就都给 0x10 啦）。

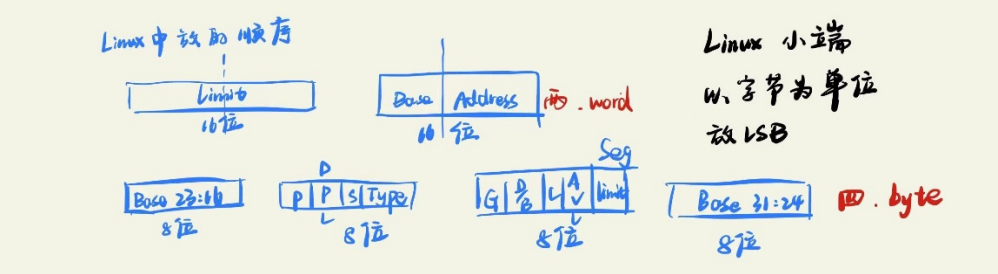


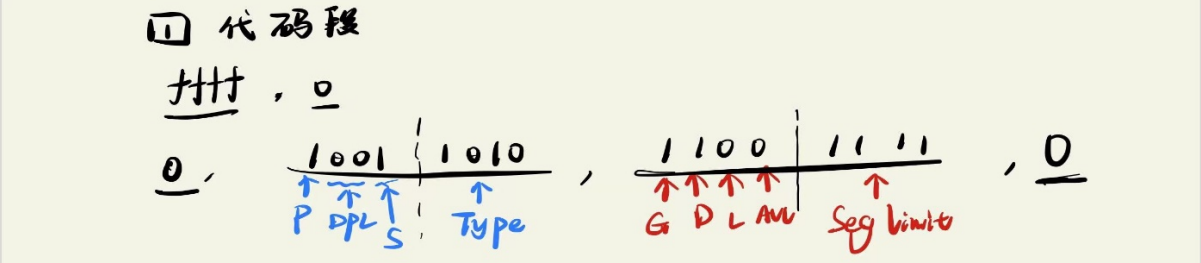

```
98                  #初始化DS ES FS GS SS 初始化栈顶指针ESP
99      movw $0x0010,%ax
100     movw %ax,%ds
101     movw %ax,%es
102     movw %ax,%fs
103     movw %ax,%ssS
104     movw $0x0018,%ax        #【gs是视频段的起始】
105     movw %ax,%gs
```

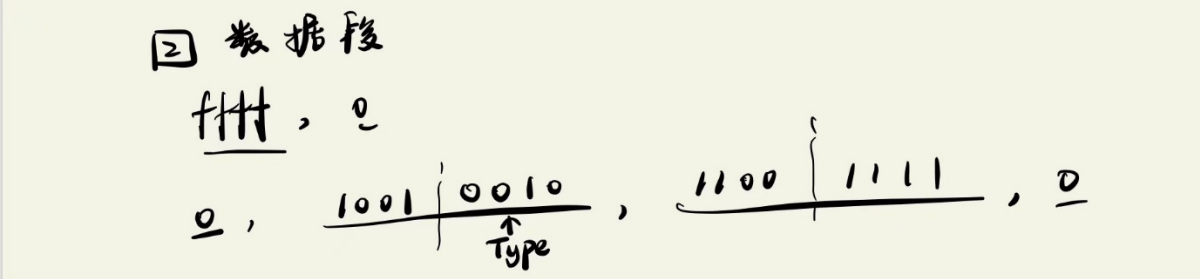
2.4 填写 GDT

查阅资料得到，Linux 内核代码段和数据段的**基地址 Base** 均为 0x00000000,**段限长 Limit** 均为 0xffff,代码段的 **Type** 为 0xa，数据段的 Type 为 0x2。两者**粒度 G** 均为 1，**特权级 DPL** 均为 0。

考虑到 IA-32+Linux 为小端机器，填写 GDT 中的表项如下所示。








```
131 gdt:
132     .word 0,0                   #GDT第一个表项必须为空
133     .byte 0,0,0,0
134     .word 0xffff,0                  #代码段描述符
135     .byte 0,0x9a,0xcf,0
136
137     .word 0xffff,0                  #数据段描述符
138     .byte 0,0x92,0xcf,0
139
140     .word 0xffff,0x8000                 #视频段描述符
141     .byte 0x0b,0x92,0xcf,0
```

其中视频描述段手册中提到了为 0x8000，且是数据段的一部分！

③ Lab1.3

把 MBR 之后扇区中的程序到内存的特定位置并跳转到 0x8c00，有两种实现方式：

1 采用函数指针进行跳转 void(*dest)(void)，2 采用内联汇编中的无条件跳转 jmp。

```
8  void bootMain(void) {
9  //【法一：可以通过函数指针，通过执行一个什麽都不作的函数，其函数首地址在0x8c00(Hello World程序的首地址)，来完成地
10     //void(*dest)(void)=(void(*))0x8c00;
11     //readSect((void*)dest,1);
12     //dest();
```

```
14  // 【法二：通过内联汇编来无条件跳转】
15      void(*dest)(void)=(void(*))0x8c00;
16      readSect((void*)dest,1);
17      asm(
18      "mov $0x8c00,%ax \n\t"
19      "jmp  *%ax"
20      );
21      // 【【jmp后面跟符号、寄存器（编译自动补全*）、*寄存器都可以呢！！！】】
```

四、 实验心得

在去年的 ICS 课程中，我是非常抵触阅读大篇幅的 Makefile 的。但在 Lab1.3 中，为了搞清楚脚本程序 genboot.pl、加载程序 bootloade、以及建议操作系统 app 之间的关系，我阅读了不同目录下的 Makefile 文件，最终得到以下结论。

bootloader 是加载程序，被放到了 0x7c00，并且通过脚本程序 genboot.pl 产生了 MBR。是 BIOS 把设备第一个扇区放到了 0x7c00(i386 规定)。同时也是一个加载程序，被 BIOS 启动之后，会进一步加载 OS（本实验中就是 app 啦）。app.bin 被放到了 0x8c00 的位置，且文件大小好像是 58 字节，小于 512 字节，所以 bootMain 函数中只需要读取一个磁盘扇区。

所以阅读 Makefile 对搞清楚编译、链接，不同文件之间的关系还是非常重要哒。

此外，Lab1 也锻炼了我信息检索能力，比如查 A20 总线的开启、GDT 的填写，也迫使我静下心来阅读部分 Linux 的内核源码；Lab1 也让我对 i386 架构的开机程序以及几个不同模块之间的关系更加清楚啦。

五、思考题的看法

EX1：CPU、内存、BIOS、磁盘、主引导扇区、加载程序、操作系统之间的关系

CPU 加电之后第一条指令在内存中，**内存**包含 ROM（随机访问存储器，断电内容无）、RAM（只读存储器，断点可保存）。**BIOS（基本输入输出系统）**就在 RAM 之中，且 CPU 加电跳转的第一条指令在内存的 RAM 的 BIOS 固件中，进行开机自检。

BIOS 运行，step1：检查所安装的 RAM 数量、键盘和其他设备可否正常响应。Step2:，扫描 PCI 总线上的所有设备。Step3：尝试 CMOS（内存的一种，不易失）存储器中的设备清单决定启

动设备。Step4：启动设备的第一个扇区，即**磁盘的主引导扇区（MBR）**，BIOS 读入内存加载到 0x7c00（i386 规定）并执行**加载程序**。其主要的功能是：把**操作系统**的代码和数据从磁盘加载到内存中，再跳转到 OS 的起始地址，并启动 OS。

EX2：中断向量表是什么？

是中断服务程序入口地址的偏移量与段基址，一个中断向量占据 4 字节空间。它把所有的中断向量集中起来，按中断类型号从小到大的顺序存放到存储器的某一区域内（i386 放在 0x0000-0x03FF 的区间），故中断向量表即**中断服务程序入口地址表**，总共存储 256 个中断向量。在中断响应过程中，CPU 通过从接口电路获取的中断类型号（中断向量号）计算对应中断向量在表中的位置，并从中断向量表中获取中断向量，将程序流程转向中断服务程序的入口地址。

EX3：为什么段的大小最大为 64KB？

I386 在实模式中，寻址空间 20 位，段寄存器和偏移地址 16 位，其物理地址的计算为**物理地址=段寄存器<<4+偏移地址**。段寄存器左移四位是本段（数据段、代码段、堆栈段……）的基址，偏移地址是相对于基址的偏移量。所以为了保证可以访问到某段的所有内容，在偏移地址只有 16 位的前提下，以及段寄存器后四位均为 0 时，段大小只能最大为 2^{16}=64KB 大小啦！

EX4：对于脚本程序 genboot.pl，先打开 mbr.bin，然后检查文件是否大于 510 字节，说明在此步之后做了什么？为什么这么做？

首先，$n = sysread(SIG, $buf, 1000);读取了文件大小，并且检查这个文件大小是否大于 510 字节【因为要把一个.bin 文件制作成 MBR，一种 512 字节的最后两字节为\x55 和\xAA 的文件，所以只有 510 字节可供文件放置本来的信息】。

然后做了如下两个动作，$buf .= "\0" x (510-$n); 把剩余的位置都放\0，标志着文件有效信息的结束 $buf .= "\x55\xAA";给 MBR 文件配置末尾的魔数，为了在 BIOS 检查文件的时候可以通过。

EX5：简述电脑从加电开始，到 OS 开始执行为止，计算机是如何运行的

同 EX1 的回答啦！

【助教哥哥还请见谅，因为写了五个思考题超过了四页啦ε(┬┬﹏┬┬)3】