

Lab1 系统引导

201300032 肖思远

实验目的

- 1. 在实模式下实现一个 Hello World 程序
- 2. 在保护模式下实现一个 Hello World 程序
- 3. 在保护模式下加载磁盘中的 Hello World 程序

实验进度

我完成了全部内容

实验结果



修改代码位置

（由于实验 3 的基础为实验 1，2 故这里仅展示实验 3）

bootloader/start.s 中：

行数	功能
11-13	启动 20 总线
19-21	设置 CR0 的 PE 位（第 0 位）为 1
27-35	初始化 DS ES FS GS SS 初始化栈顶指针 ESP
60-69	设置 GDT 表项

Bootloader/boot.c 中:

在 bootMain 函数中添加内容, 可以跳转到地址 0x8c00 中

实验思考题

1. 小结标题中各种名词的含义和他们间的关系:

名词: CPU 内存 BIOS 磁盘 主引导区 操作系统

CPU 执行的命令需要从内存中获得。在开机的时候, 首先跳转到 ROM (内存的一部分) 中 BIOS 固件进行开机自检, 然后从磁盘中读取 MBR (在磁盘的第一扇区) 到内存 (地址为 7c00H 处) 检查魔数等, 执行 MBR 的启动代码, MBR 的启动代码会启动操作系统。上述即这些名词之间的关系。

2. 中断向量表是什么?

中断向量表是一个将一系列中断处理和一系列中断请求联系起来的数据结构 (来自 wiki)。每一个表项 (也叫做中断向量), 是一个处理 (代码) 的地址。系统程序必须维护一份中断向量表, 当外部事件或异常产生时, 用中断号 (乘以特定的倍数) 作为偏移加上中断表首地址, 即得到处理方法的地址, 即可以跳转。中断向量 (Interrupt vector) 是中断服务程序的入口地址, 或中断向量表 (它是一个中断处理程序地址的数组) 的表项。

3. 为什么段的大小最大为 64KB

8086 是 16 位机, 地址由两个 16 位合成: 段地址和偏移地址。偏移地址是 16 位的话, 在一个段的起始到末尾地址相差最多只能相差 2 的 16 次方, 即 64KB。操作系统为了能向下兼容所以段就设置为 64KB。

4. genboot.pl 其实是一个脚本程序, 观察 genboot.pl, 说明它在检查文件是否大于 510 字节之后做了什么, 并解释它为什么这么做。

```
print STDERR "OK: boot block is $n bytes (max 510)\n";
$buf .= "\0" x (510-$n);
$buf .= "\x55\xAA";
open(SIG, ">$ARGV[0]") || die "open >$ARGV[0]: $!";
print SIG $buf;
```

检查文件是否大于 510K, 如果大于会输出错误并推出, 如果不大于会输出是可以的, 将 510 字节剩下部分填充为 "\0" (\$buf .= "\0" x (510-\$n)), 再在末尾加上 "\x55\xAA" (此时 buf 变量共有 512 字节), 打开 SIG 文件, 并将 buf 变量内容存入。这么做的原因: 构造一个合理的 MBR, 比如构造魔数等等

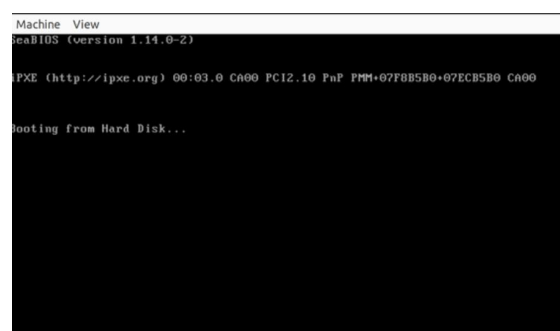
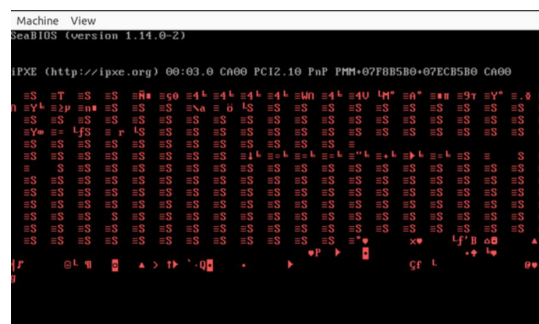
5. 请简述电脑从加电开始，到 OS 开始执行为止，计算机是如何运行的。

电脑加电后，BIOS 进行跳转到 ROM 进行自检，自检完成加载磁盘第一个盘区的 MBR，执行 MBR 上指令来完成初始化段寄存器，开启保护模式，载入操作系统并跳转等工作。

实验思考

（感谢我的勇于乱改代码的大无畏精神以及改了就忘的坏习惯，为我的实验平添了许多难度花了我三天）在实验中，我遇到了许多的 bug，其中有两条我觉得比较值得思考。

1. 在 app.s 加了 ret, 在 Mainboot 中到 app.s 是通过 call, 在 bootloader 中的 start.s 中写为 call Mainboot, 然后 jmp loop（我的目前知识理论上来说我觉得可行）导致后果如下：



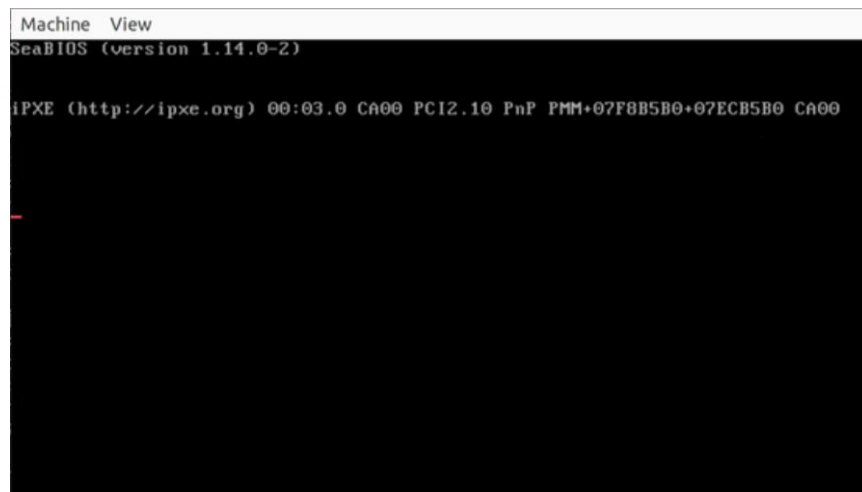
（乱码和载入交替闪烁）

比较费解的是为什么会闪，理论上我没有反复改 VGA 处的内容..? 同样为什么乱码我也分析不出来，以及执行过程。我通过 qemu 自带的 monitor 输出 EIP 发现执行到 0xd057(此处不是有效程序的地址)。比较可惜的是我想找到 IP 的变化过程但是我找不到这样的记录。其实我也非常好奇 QEMU 的机制，屏幕的显示刷新是如何，希望我学完线程和终端后能有所感悟。

2.callw 和 calll

之前学的汇编中，似乎只有 call。好奇了一下 callw 和 calll 的区别之后，在 app.s

中将 `calll` 改为 `callw`,于是得到结果如下:

A screenshot of a SeaBIOS boot screen. The title bar at the top says "Machine View". The main text reads "SeaBIOS (version 1.14.0-2)". Below that, it says "iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8B5B0+07ECB5B0 CA00". A red cursor is visible on the left side of the screen, indicating that the user is in the boot menu.

(有红色的输入键但是打不出 Hello Word)

最后去查了一下 `callw` 等的区别,发现后缀规定的是返回地址的位数,所以在开启保护模式后用 `callw` 会导致函数调用结束跳转错误。所以 lab1.1 的实模式需要用 `callw`,在开启保护模式后均用 `calll`