

Lab1 实验报告

李杨 161220071

实验目的:

关键实验: 从实模式切换至保护模式, 在保护模式下读取磁盘 1 号扇区中的 HelloWorld 程序至内存中的相应位置, 跳转执行该 HelloWorld 程序, 并在终端中打印 Hello,World!

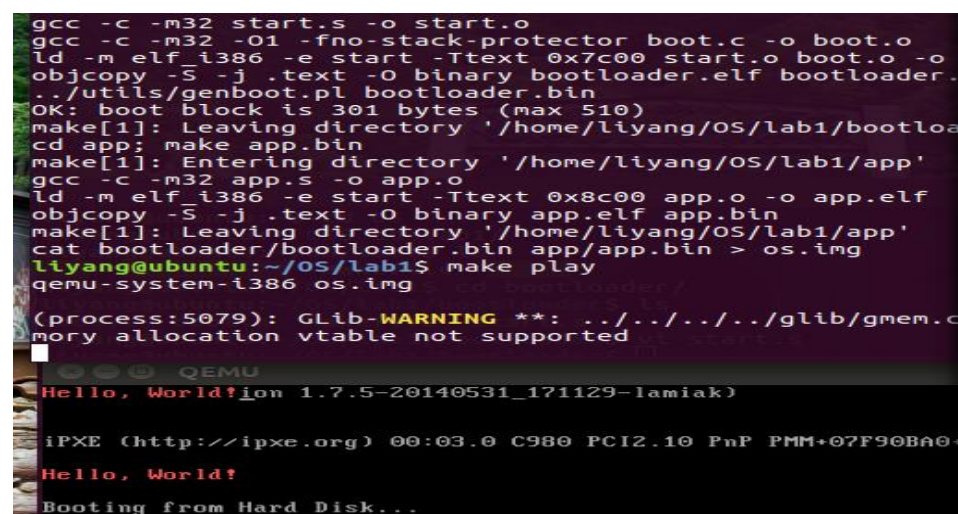
问题解决:

系统启动时, 计算机工作在实模式下, 其中 CS:IP 指向 BIOS 的第一条指令, 即首先取得控制权的是 BIOS。所以在此处可以通过陷入屏幕中断调用 BIOS 打印出 "Hello, World!"。

在进入保护模式后, 关闭中断, 打开 A20 数据总线, 加载 GDTR, 设置 CRO 的 PE 位 (第 0 位) 为 1b, 通过长跳转设置 CS 进入保护模式, 初始化 DS, ES, FS, GS, SS, 这里由于段描述符的长度为 8, 所以 cs,ds,gs 在初始化的时候应该分别初始化 0x8,0x10,0x18, 由于在这次的实验并没有用到 cs, 所以 cs 就没有进行初始化, 对于 esp 的初始化, 堆栈是由高地址向低地址增长的, ESP 指向栈顶, 随便找一个空闲的区域就可以对其进行初始化, 这里选取 1024 (足够大了) 这个地方。

在这之后, 程序会跳转到 bootMain 函数。框架中实现了 readSec(void *dst, int offset) 这一接口, 其通过读写 (in, out 指令) 磁盘的相应端口 (Port) 来实现磁盘特定扇区的读取。通过上述接口读取磁盘 MBR 之后扇区中的程序至内存的特定位置并跳转执行, 通过查询 Makefile, 可以知道 app.s 程序的入口地址为: 0x8c00, `ld -m elf_i386 -e start -Ttext 0x8c00 app.o -o app.elf`, 所以调用 readSec 函数, 可以从磁盘中读取 helloworld 程序, 并通过显存的方式实现 Hello, world 的打印。这里通过设置一个死循环, 来避免程序的崩溃。

实验实现:



```
gcc -c -m32 start.s -o start.o
gcc -c -m32 -O1 -fno-stack-protector boot.c -o boot.o
ld -m elf_i386 -e start -Ttext 0x7c00 start.o boot.o -o
objcopy -S -j .text -O binary bootloader.elf bootloader.
../utils/genboot.pl bootloader.bin
OK: boot block is 301 bytes (max 510)
make[1]: Leaving directory '/home/liyang/OS/lab1/bootloa
cd app; make app.bin
make[1]: Entering directory '/home/liyang/OS/lab1/app'
gcc -c -m32 app.s -o app.o
ld -m elf_i386 -e start -Ttext 0x8c00 app.o -o app.elf
objcopy -S -j .text -O binary app.elf app.bin
make[1]: Leaving directory '/home/liyang/OS/lab1/app'
cat bootloader/bootloader.bin app/app.bin > os.img
liyang@ubuntu:~/OS/lab1$ make play
qemu-system-i386 os.img

(process:5079): GLib-WARNING **: ../../../../glib/gmem.c
mory allocation vtable not supported

QEMU
Hello, World!ion 1.7.5-20140531_171129-lamiak)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F90BA0+
Hello, World!
Booting from Hard Disk...
```