# 操作系统 Lab1

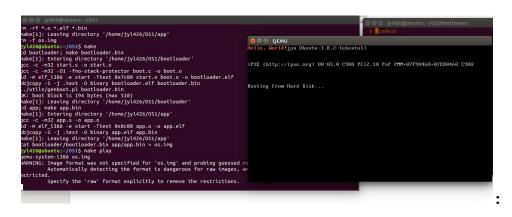
#### 一. 实验目的:

通过实现一个简单的引导程序,了解系统启动的基本过程。最终实现在保护模式下。

#### 二. 实验思路

1. 首先在实模式下实现 Hello World 程序, 能够打印字符串 Hello, World!

分别在 start. s 和 app. s 加入相应的打印字符串代码 实现效果如下



### 2. 实模式切换为保护模式:

框架代码中给出了大部分内容实现切换到保护模式,我们需要实现初始化寄存器 DS, ES, FS, GS, SS。

因为 8 个字节存储一个段, $0^{\circ}$ 0x10 是代码段。所以 DS, FS, ES, SS 初始化的值为 0x10;而 0x10 $^{\circ}$ 0x18 的 8 个字节是视频段,因此 GS 的值初始化为 0x18.

## 3. 加载磁盘中的程序并运行:

上一步中初始化完之后, 跳转到 bootMain 中继续运行。在

bootMain中我们需实现特定扇区读取,以加载运行程序。通过函数接口 readSec()实现所需,因为 Makefile 中所写入口地址为0x7c00;所以 elf=(void \*)0x7c00。参数 In 入口地址也即为0x7c00。

完成上述三步后在 app. s 中通过提供的通过写显存打印字符的代码,一个字符一个字符实现对 Hello World!字符串的打印。(我未采用汇编的循环代码,通过简化提供代码设置一变量每次递增,重复13 遍之后 打印的字符串"Hello, World!")

完成后的效果截图:

cd bootloader, make bootloader.bin
make[1]: Enaving directory '/home/jyl426/0S/bootloader'
gcc -c -m32 start.s -o start.o
gcc -c -m32 -01 -fno-stack-protector boot.c -o boot.o
ld -m elf\_1386 -e start -ftext 0x7c00 start.o boot.o -o bootloader.elf
ld: warning: cannot find entry symbol start; defaulting to 000000000000000000
../utils/genboot.pl bootloader.bin
OK: boot block is 202 bytes (max 510)
make[1]: Leaving directory '/home/jyl426/OS/bootloader'
cd app; make app.bin
make[1]: Leaving directory '/home/jyl426/OS/app'
gcc -c -m32 app.s -o app.o
ld -m elf\_1386 -e start -ftext 0x8c00 app.o -o app.elf
objcopy -5 -j .text -0 binary app.elf app.bin
make[1]: Leaving directory '/home/jyl426/OS/app'
gcat bootloader/bootloader-bin app/app.bin > os.img
jyl426@ubuntu:-OSS make play
qemu-system-1386 os.img
MARNING: Inage format was not specified for 'os.img' and probing guessed raw.
Automatically detecting the format is dangerous for raw inages, write o
perations on block 0 will be restricted.

Specify the 'raw' format explicitly to remove the restrictions.