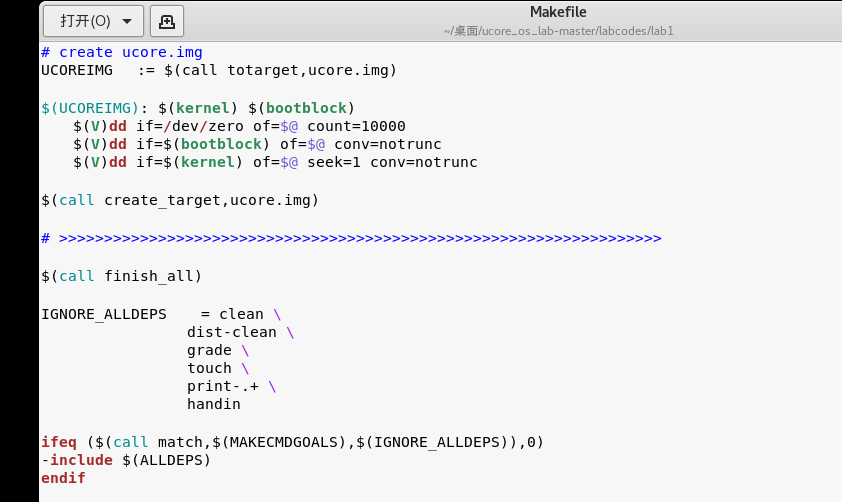
**操作系统lab1实验报告**

练习1：理解通过make生成执行文件的过程。

1. 操作系统镜像文件ucore.img是如何一步一步生成的？

我查看了lab1下的Makefile文件

找到create ucrore.image



里面用到了两个块分别叫做kernel和bootblock，

dd是一条linux命令指令：

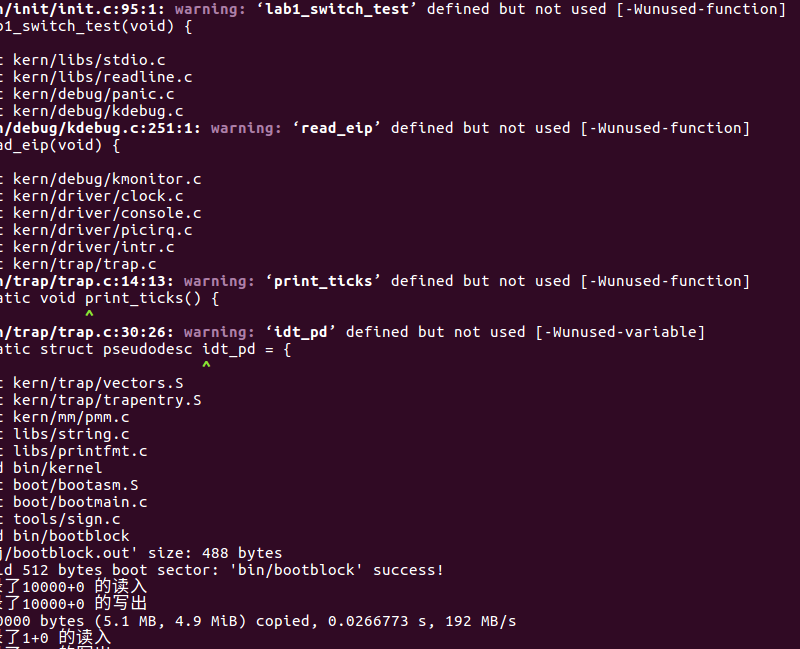
dd：用指定大小的块拷贝一个文件，并在拷贝的同时进行指定的转换。

if=文件名：输入文件名，缺省为标准输入。即指定源文件。< if=input file >

of=文件名：输出文件名，缺省为标准输出。即指定目的文件<of=output file >

count=blocks：仅拷贝blocks个块，块大小等于ibs指定的字节数。conv=conversion：用指定的参数转换文件。 conv=notrunc:不截短输出文件

我感觉这里应该是说先创建一个大小为10000字节的内存块，然后再将bootblock和kernel拷贝过去



其中相关参数的含义为：

ggdb 生成可供gdb使用的调试信息

m32生成适用于32位环境的代码

gstabs 生成stabs格式的调试信息

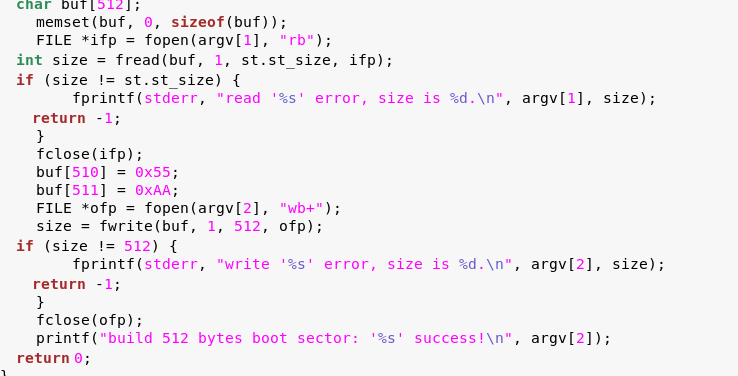
nostdinc 不使用标准库

fno-stack-protector 不生成用于检测缓冲区溢出的代码

0s 位减小代码长度进行优化

2.一个被系统认为是符合规范的硬盘主引导扇区的特征是什么？

查看lab1/tools/sign.c源代码



由代码可以看出、符合规范的硬盘主引导扇区的大小为512个字节，且最后两个字节为0x55、0xAA。

**练习2** 使用qemu执行并调试lab1中的软件

这个实验只是练习一下gdb调试

1.首先修改 lab1/tools/gdbinit,内容为：

set architecture i8086

target remote :1234

意思是与qemu建立连接

2.在 lab1目录下，执行make debug

3.然后在gdb进行单步调试

4.gdb界面下，可通过如下命令来看BIOS的代码

x /2i $pc

**练习2.2**在初始化位置0x7c00设置实地址断点,测试断点正常。

在tools/gdbinit结尾加上

b \*0x7c00 //在0x7c00处设置断点。此地址是bootloader入口点地址，可看boot/bootasm.S的start地址处

c //continue简称，表示继续执行

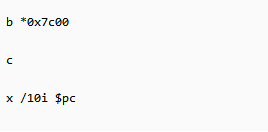
x /5i $pc //显示当前eip处的汇编指令

set architecture i386 //设置当前调试的CPU是80386

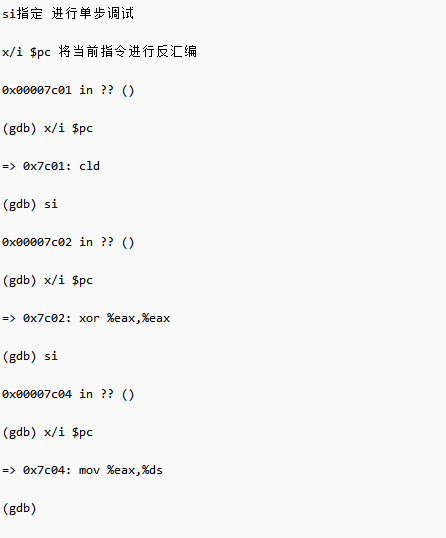
在lab1目录下，运行make debug，可发现gdb打印出 the target architecture is assumed to be i386。

**练习2.3** 从0x7c00开始跟踪代码运行，将单步跟踪反汇编得到的代码与bootasm.S和bootblock.asm进行比较

1. 在tools/gdbinit结尾加上



1. 在0x7c00会进行break，然后使用si和x/i $pc 进行单步调试



1. 经过对比发现，上述反汇编指令与两个汇编文件相同

**练习3** 分析bootloader进入保护模式的过程

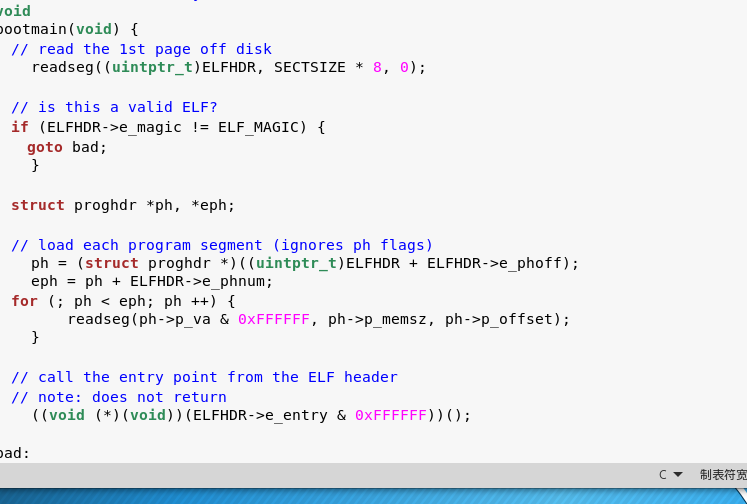
boot.s是一个汇编文件，用于打开A20 gate，A20是一个地址线，打开A20后内存地址位数由20位变到32位

实模式： cs：ip寻址模式 也就是cs乘以16（左移4位）+ip 最大寻址空间1M

保护模式： 保护模式与实模式相比，主要是两个差别：一是提供了段间的保护机制，防止程序间胡乱访问地址带来的问题，二是访问的内存空间变大。

**练习4**：分析bootloader加载ELF格式的OS的过程。

首先看bootmain.c的源码



ELF就是一个可执行文件的意思，分析得到功能分别是

读取第一个扇区

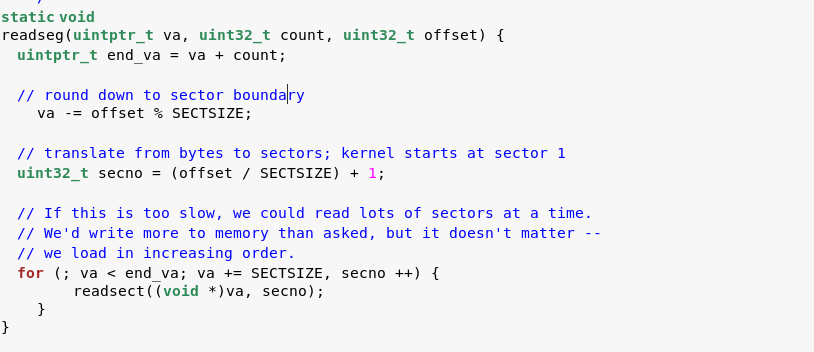
判断是否是一个有效的elf

加载接下来的扇区

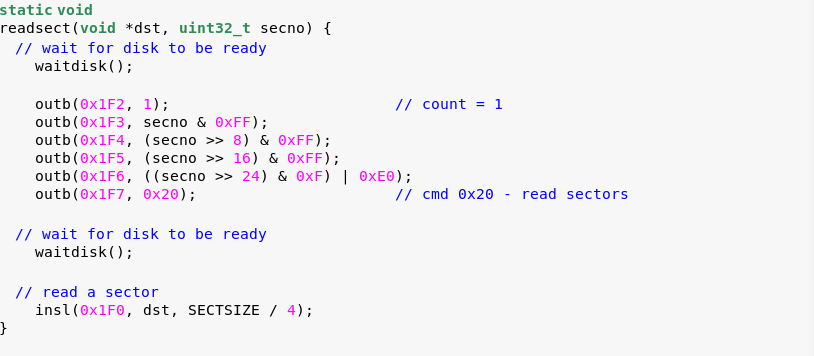
利用强转为函数指针调用函数

**练习4.1** bootloader如何读取硬盘扇区的？

查看 readsect函数 源码



readsect函数是被readseg里面调用的一个函数



各地址代表的寄存器意义如下：

0x1f0 读数据，当0x1f7不为忙状态时，可以读。

0x1F3 R/W，数据寄存器

0x1F2 R/W，扇区数寄存器，记录操作的扇区数

0x1F3 R/W，扇区号寄存器，记录操作的起始扇区号

0x1F4 R/W，柱面号寄存器，记录柱面号的低 8 位

0x1F5 R/W，柱面号寄存器，记录柱面号的高 8 位

0x1F6 R/W，驱动器/磁头寄存器，记录操作的磁头号，驱动器号，和寻道方式，前 4 位代表逻辑扇区号的高 4 位，DRV = 0/1 代表主/从驱动器，LBA = 0/1 代表 CHS/LBA 方式

0x1F7 R，状态寄存器，第 6、7 位分别代表驱动器准备好，驱动器忙

0x1F8 W，命令寄存器，0x20 命令代表读取扇区

代码分析

设置读取扇区的数目为1

在这4个字节线联合构成的32位参数中

29-31位强制设为1

28位(=0)表示访问"Disk 0"

0-27位是28位的偏移量

0x20 命令 读取扇区

从0x1F0读取SECTSIZE字节数到dst的位置,每次读四个字节，读取 SECTSIZE/ 4次。

**练习4.2** bootloader是如何加载ELF格式的OS的?

也是bootmain函数内容

**练习5**：实现函数调用堆栈跟踪函数

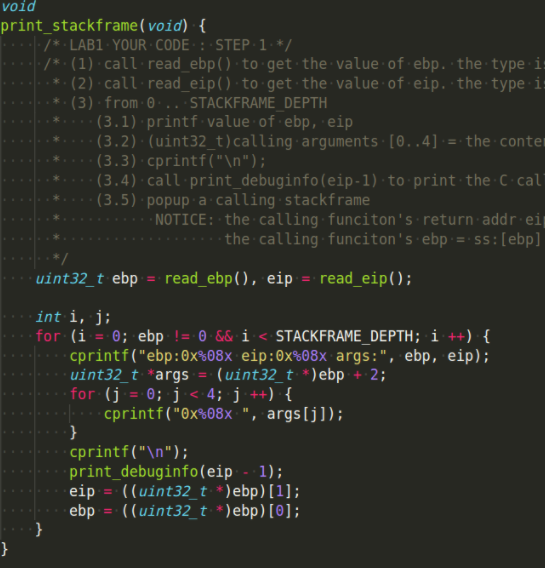
函数堆栈

栈是一个很重要的编程概念,与编译器和编程语言有紧密的联系。理解调用栈最重要的两点是:栈的结构,EBP寄存器的作用。一个函数调用动作可分解为:零到多个PUSH指令(用于参数入栈),一个CALL指令。CALL指令内部其实还暗含了一个将返回地址(即CALL指令下一条指令的地址)压栈的动作(由硬件完成)。几乎所有本地编译器都会在每个函数体之前插入类似如下的汇编指令:

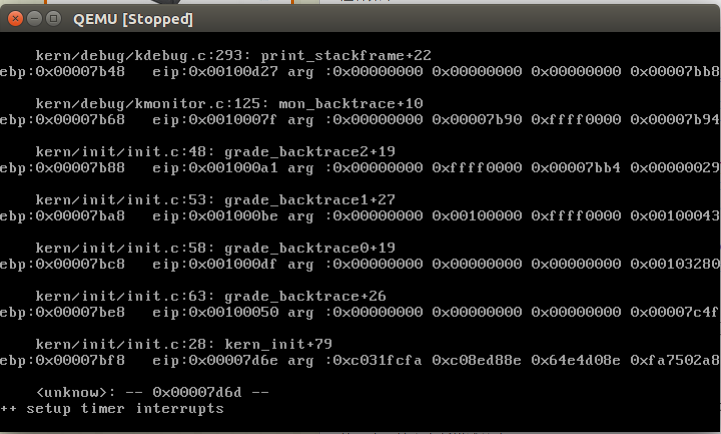
pushl %ebp

movl %esp , %ebp

这两条汇编指令的含义是:首先将ebp寄存器入栈,然后将栈顶指针esp赋值给ebp。“mov ebp esp”这条指令表面上看是用esp覆盖ebp原来的值,其实不然。因为给ebp赋值之前,原ebp值已经被压栈(位于栈顶),而新的ebp又恰恰指向栈顶。此时ebp寄存器就已经处于一个非常重要的地位,该寄存器中存储着栈中的一个地址(原ebp入栈后的栈顶),从该地址为基准,向上(栈底方向)能获取返回地址、参数值,向下(栈顶方向)能获取函数局部变量值,而该地址处又存储着上一层函数调用时的ebp值。



运行效果



**练习6** 完善中断初始化和处理

**练习6.1** 中断描述符表（也可简称为保护模式下的中断向量表）中一个表项占多少字节？其中哪几位代表中断处理代码的入口？

一个表项占八个字节。入口地址为：gd\_off\_31\_16 << 16 + gd\_off\_15\_0 ;