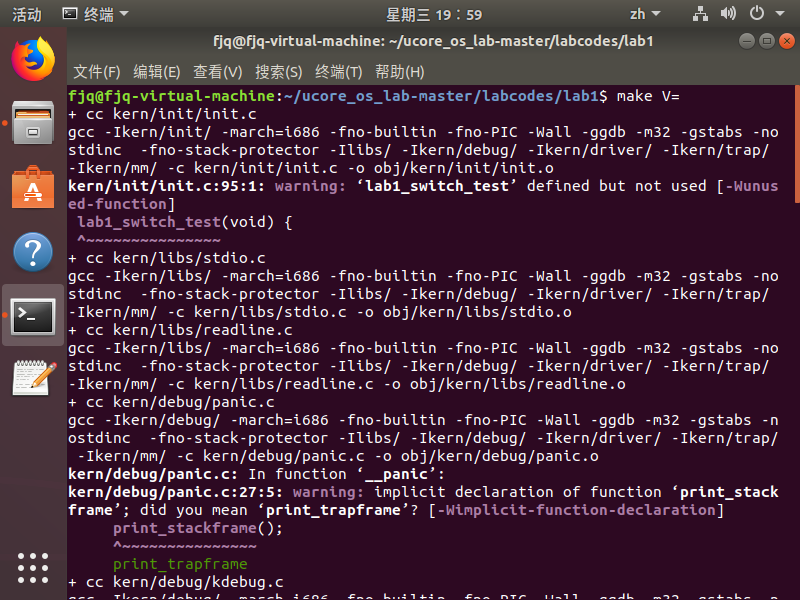
**练习1：理解通过make生成执行文件的过程**

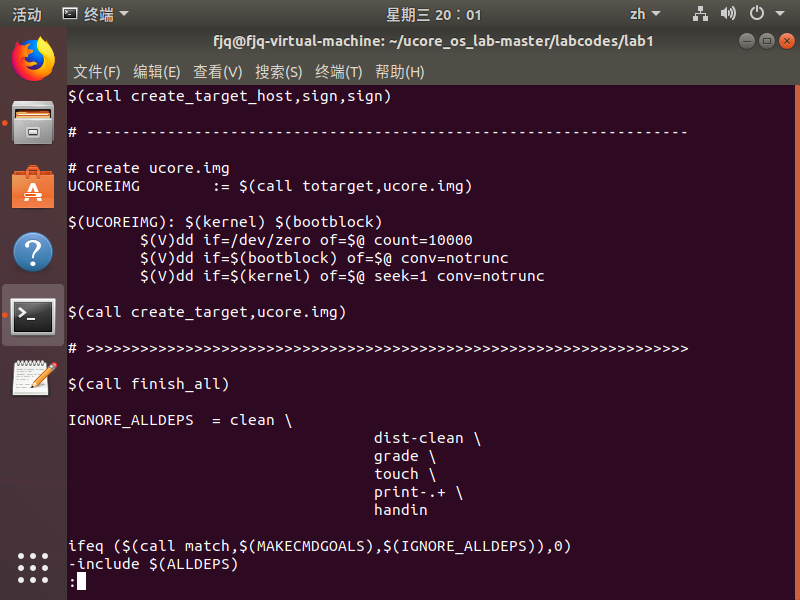
1.操作系统镜像文件ucore.img是如何一步一步生成的？(需要比较详细地解释Makefile中每一条相关命令和命令参数的含义，以及说明命令导致的结果)

了解make执行了哪些命令

输入make V=



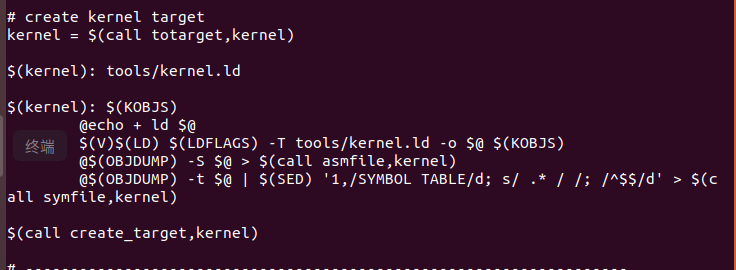
执行less Makefile指令，查看Makefile文件的内容。生成ucore.img的部分如下：



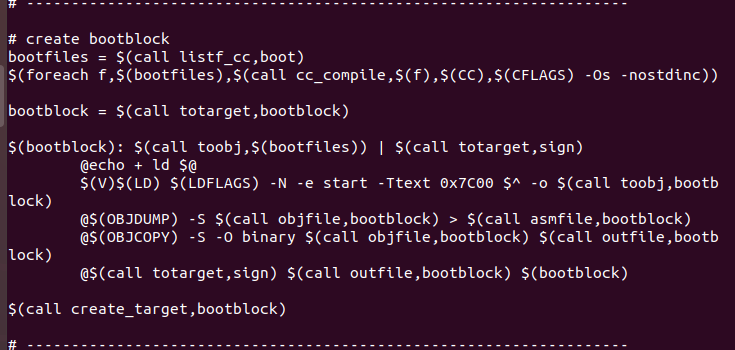
由上述代码知

生成ucore.img需要kernel和bootblock

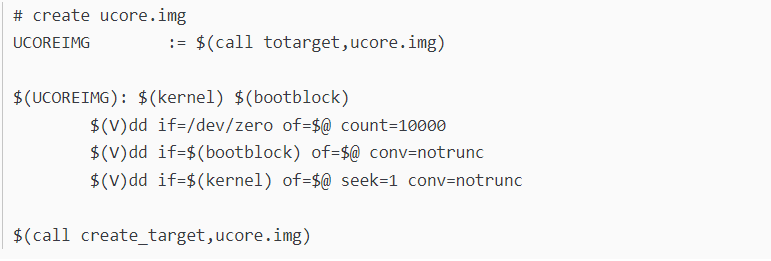
生成kernel：



生成bootblock：



由create ucore.img代码可知：



首先要生成一个有10000个块的文件，每个块默认512字节，用0填充

dd if=/dev/zero of=bin/ucore.img count=10000

然后将bootblock和kernel拷贝过去，bootblock占第一个512字节；

kernel被拷贝到UCOREIMG的512字节之后。

dd的一些参数的含义：

1.-if表示输入文件，如果不指定，那么会默认从stdin中读取输入

2.-of表示输出文件，如果不指定，那么会stdout

3.bs表示以字节为单位的块大小

4.count表示被赋值的块数

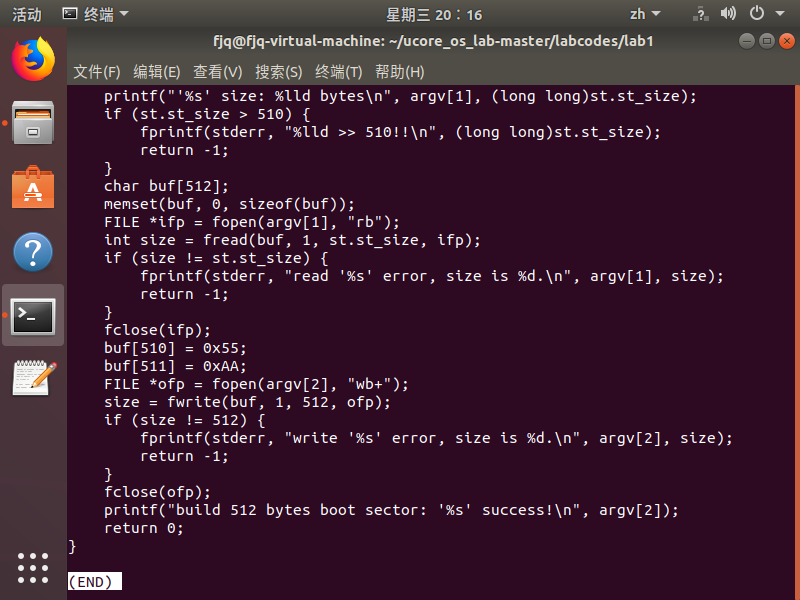
5./dev/zero是一个字符设备，会不断返回0值字节\0

6.conv = notrunc 不截短输出文件

7.seek=blocks 从输出文件开头跳过blocks个块后再开始复制

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？

在lab1 目录下执行命令less tools/sign.c，查看代码段



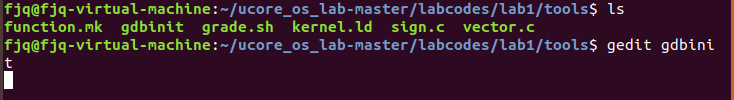
通过sign对bootblock进行了限制

磁盘主引导扇区只有512字节  
磁盘最后两个字节为0x55，0xAA

**练习2：使用qemu执行并调试lab1中的软件。**

1.从CPU加电后执行的第一条指令开始，单步跟踪BIOS的执行

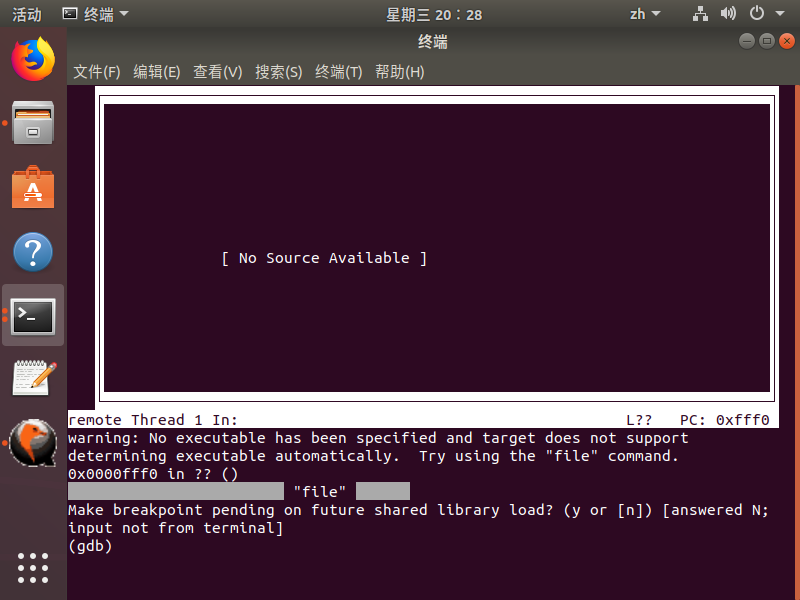
进入lab1/tools ，修改目录下的gdbinit文件



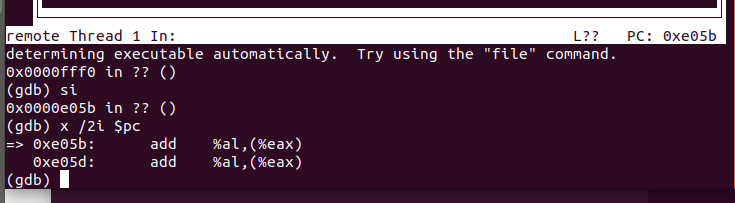
修改结果



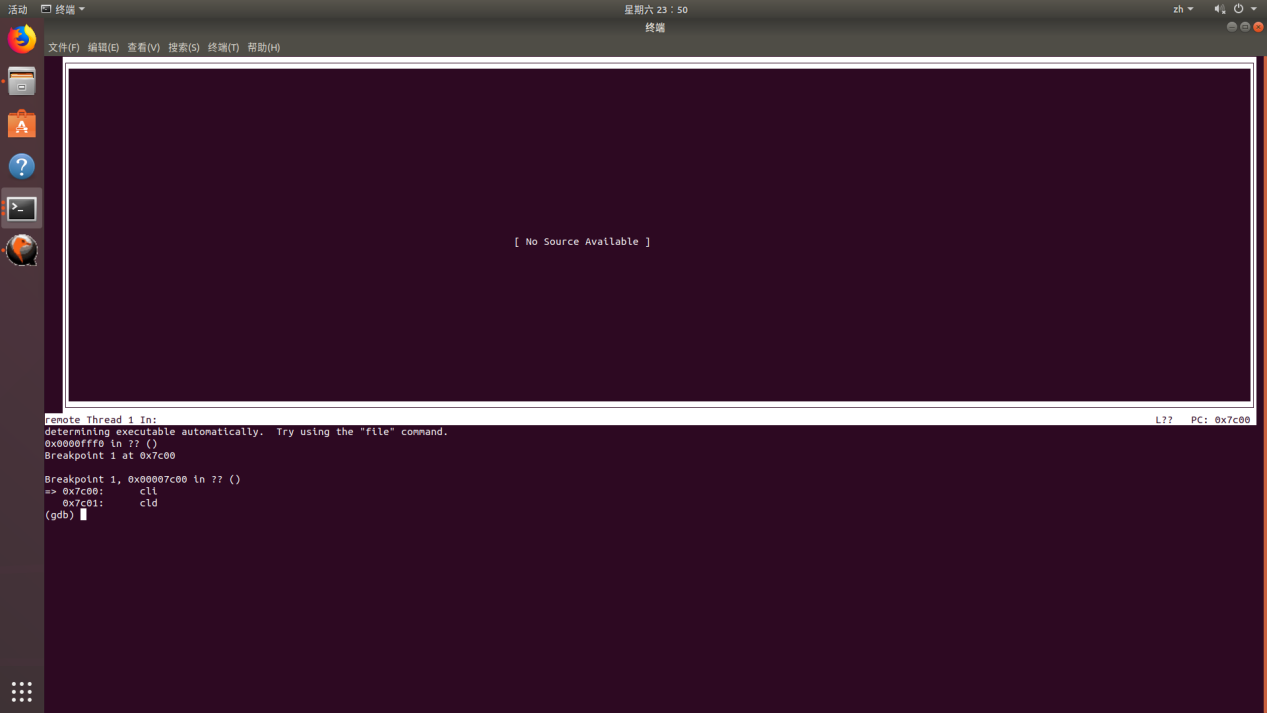
返回到lab1目录下，执行make debug命令



打开gdb调试，并在调试界面输入si指令来进行单步的追踪，x /2i $pc指令来查看汇编代码

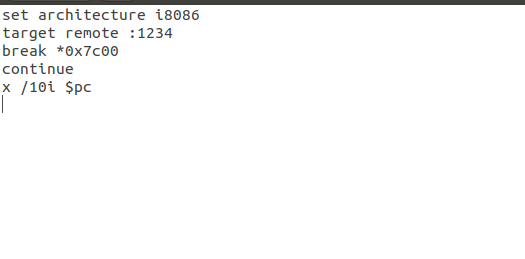


1. 在初始化位置0x7c00设置实地址断点,测试断点正常。

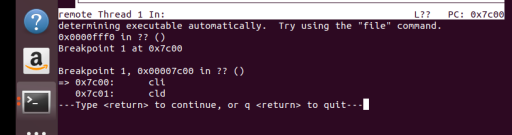


1. 从0x7c00开始跟踪代码运行,将单步跟踪反汇编得到的代码与bootasm.S和 bootblock.asm进行比较。

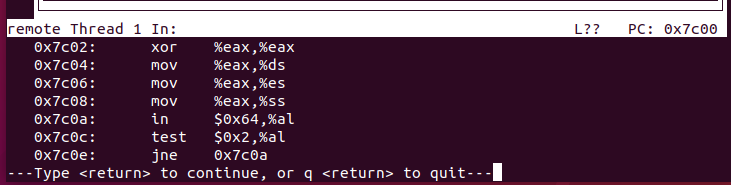
首先显示10条汇编指令x /10i $pc



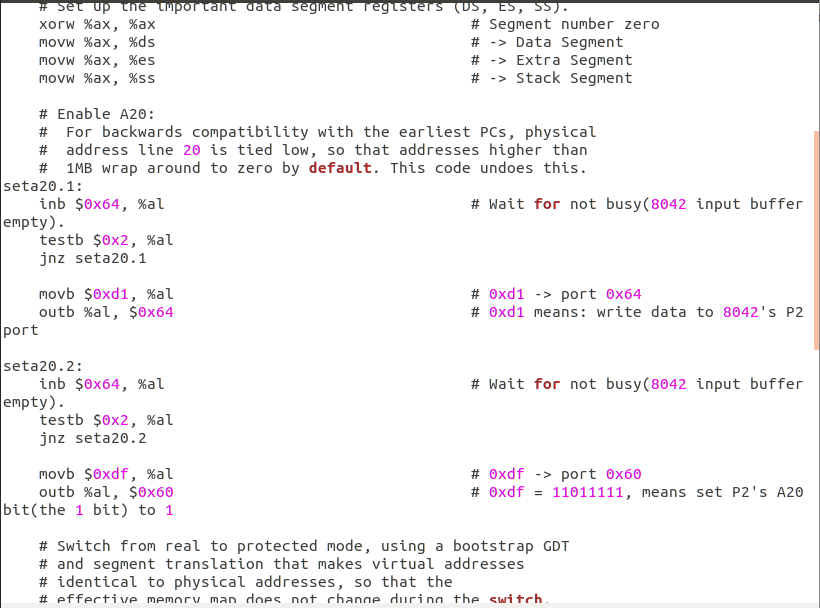
执行make debug指令，如下图：



Continue

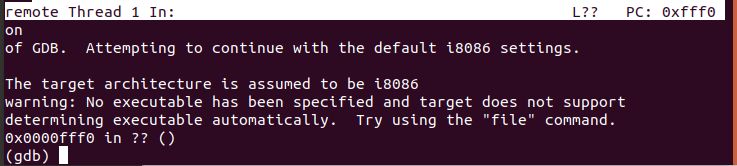


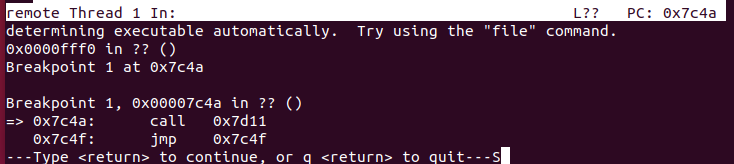
截取的bootasm.s部分代码



经对比可知执行的汇编指令与bootasm.s中的部分一样

1. 自己找一个bootloader或内核中的代码位置，设置断点并进行测试。

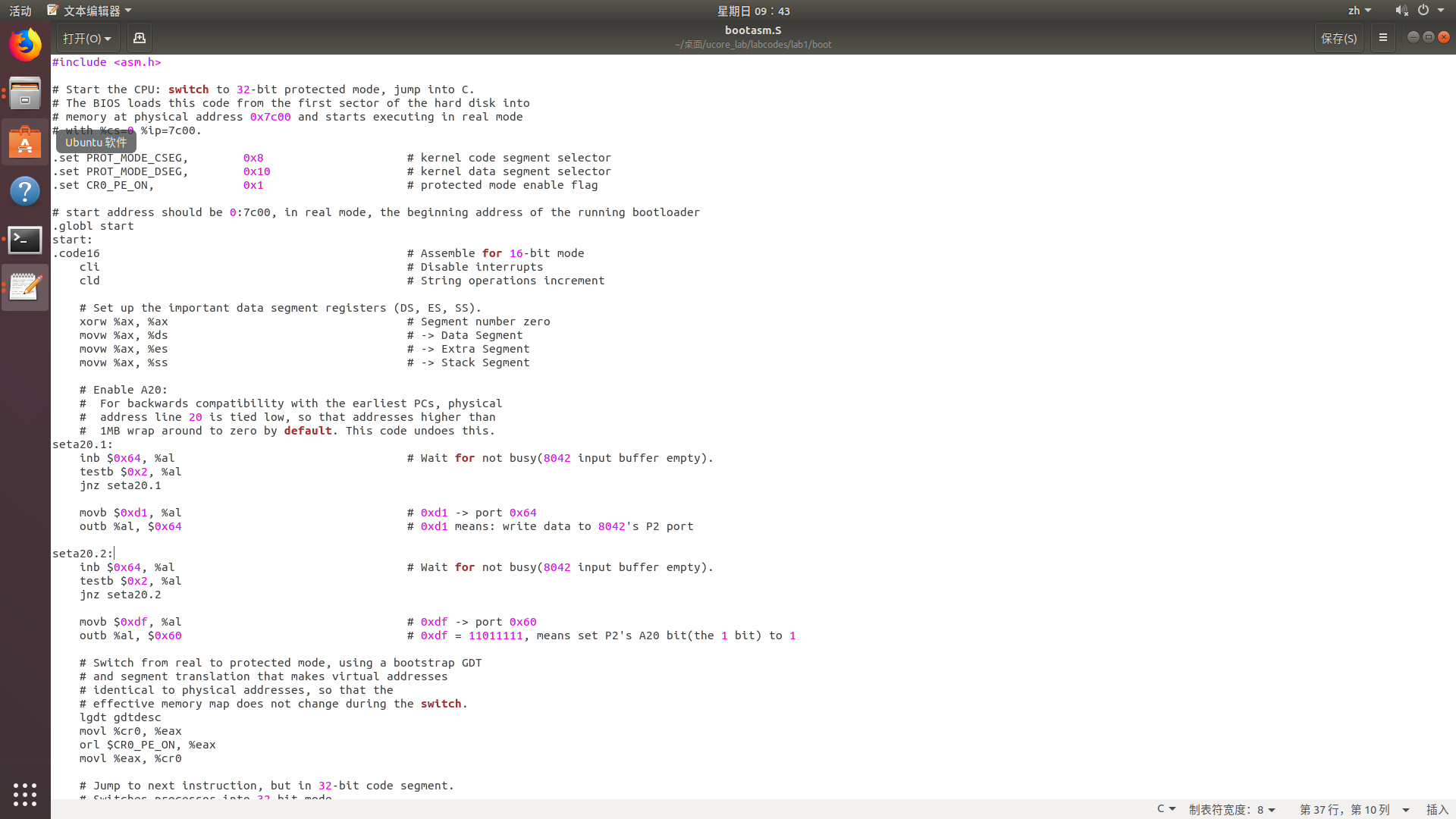




**练习3：分析bootloader进入保护模式的过程。**

BIOS将通过读取硬盘主引导扇区到内存，并转跳到对应内存中的位置执行bootloader。请分析bootloader是如何完成从实模式进入保护模式的。

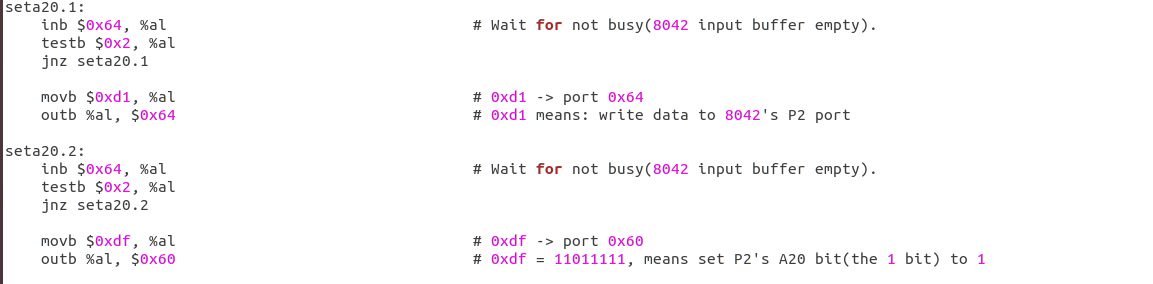
查看bootasm.S源码



1. 为何开启A20，以及如何开启A20

开启A20地址线之后，用来表示内存地址的位数变多了。开启前20位，开启后是32位。如果不开启A20地址线内存寻址最大只能找到1M，对于1M以上的地址访问会变成对address mod 1M地址的访问。通过将键盘控制器上的A20线置于高电位，全部32条地址线可用，可以访问4G的内存空间。

因此在进入保护模式(需要访问超过1MB的内存空间)前，我们需要开启A20地址线(20bit以上的地址线可为0或者1)。具体代码如下：

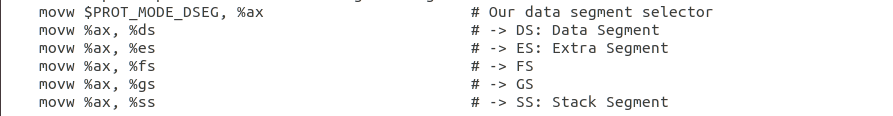


1. 如何初始化GDT表

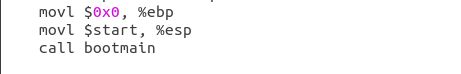
（1）加载GDT表



1. 将CR0的第0位置1



1. 将段寄存器重装



转到保护模式完成，进入boot主函数。call bootmain即为调用bootmain函数

1. 如何使能和进入保护模式

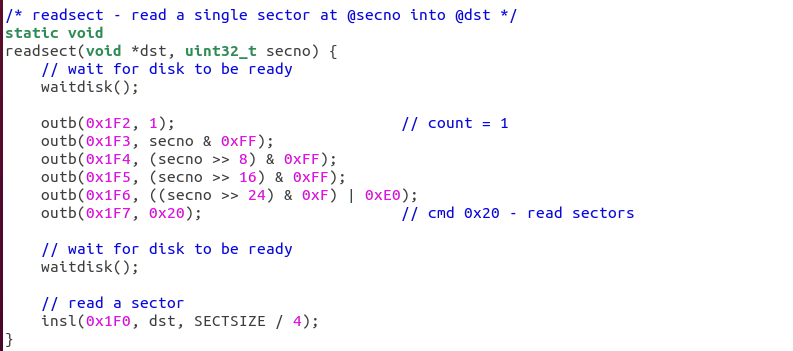
将cr0的值置1就可以使能和进入保护模式

**练习4：分析bootloader加载ELF格式的OS的过程。**

通过阅读bootmain.c，了解bootloader如何加载ELF文件。通过分析源代码和通过qemu来运行并调试bootloader&OS，

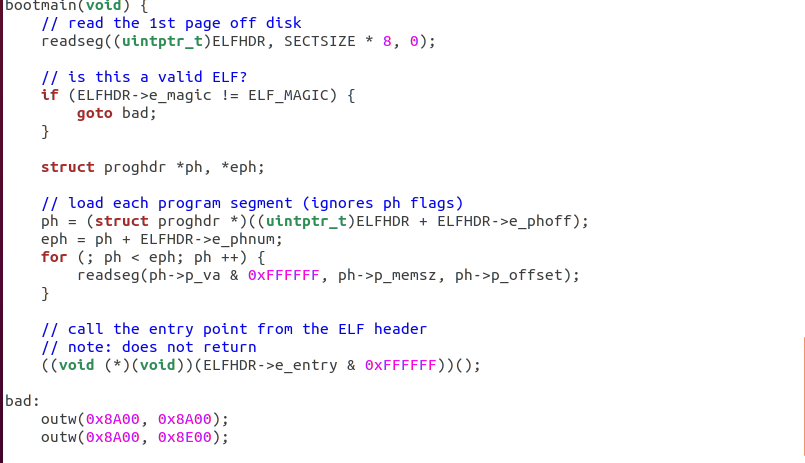
1. bootloader如何读取硬盘扇区的？

读硬盘扇区的代码如下:



从outb()可以看出这里是用LBA模式的PIO（Program IO）方式来访问硬盘的。从磁盘IO地址和对应功能表可以看出，该函数一次只读取一个扇区。

1. bootloader是如何加载ELF格式的OS？



从硬盘读了8个扇区数据到内存0x10000处，并把这里强制转换成elfhdr使用；

校验e\_magic字段；

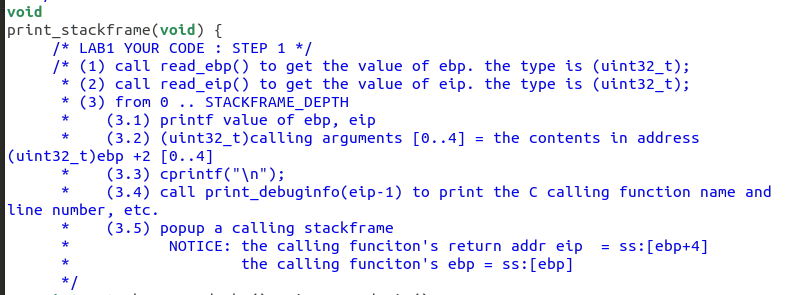
根据偏移量分别把程序段的数据读取到内存中。

**练习5：实现函数调用堆栈跟踪函数**

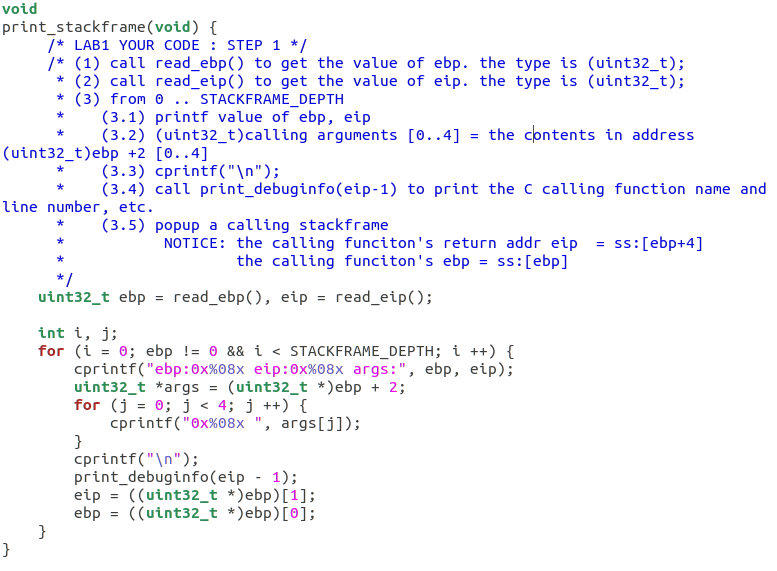
ss:ebp指向的堆栈位置储存着caller的ebp，以此为线索可以得到所有使用堆栈的函数ebp。

ss:ebp+4指向caller调用时的eip，ss:ebp+8等是（可能的）参数。

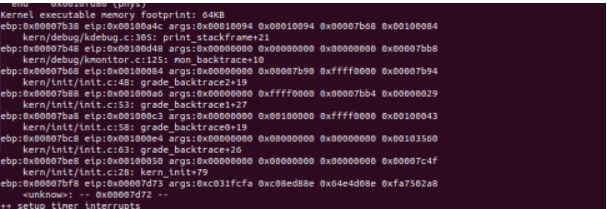
在kdebug.c中查看print\_stackframe()函数



实现函数如下：



在qemu中得到:



注意：

read\_ebp读取的当前ebp寄存器的值

而read\_eip读取的不是当前eip寄存器的值，而是调用者的返回地址即ss:[ebp+4]处的值

**练习6 完善中断初始化和处理**

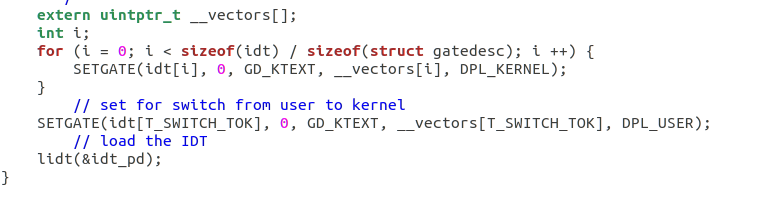
1. 中断描述符表（也可简称为保护模式下的中断向量表）中一个表项占多少字节？其中哪几位代表中断处理代码的入口？

中断向量表一个表项占用8字节，其中2-3字节是段选择子，0-1字节和6-7字节拼成位移，通过段选择子去GDT中找到对应的基地址

两者联合便是中断处理程序的入口地址。

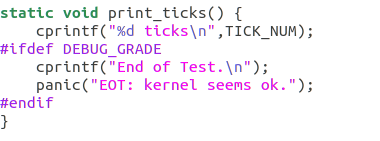
中断描述符表中一个表项占8个字节，其中每个位的作用如图：

1. 请编程完善kern/trap/trap.c中对中断向量表进行初始化的函数idt\_init。在idt\_init函数中，依次对所有中断入口进行初始化。使用mmu.h中的SETGATE宏，填充idt数组内容。每个中断的入口由tools/vectors.c生成，使用trap.c中声明的vectors数组即可。

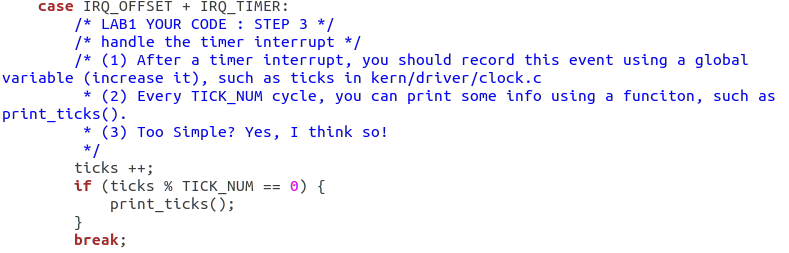


1. 请编程完善trap.c中的中断处理函数trap，在对时钟中断进行处理的部分填写trap函数中处理时钟中断的部分，使操作系统每遇到100次时钟中断后，调用print\_ticks子程序，向屏幕上打印一行文字”100 ticks”。

print\_ticks子程序



实现



运行结果

