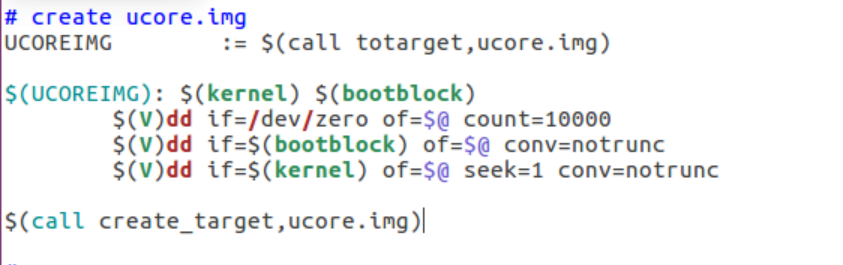
**练习1**

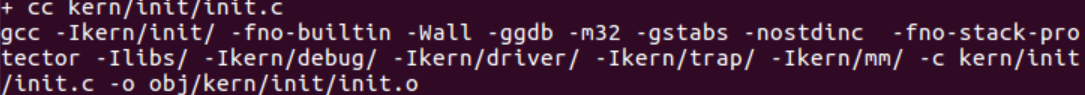
1. 操作系统镜像文件ucore.img是如何一步一步生成的？(需要比较详细地解释Makefile中每一条相关命令和命令参数的含 义，以及说明命令导致的结果)

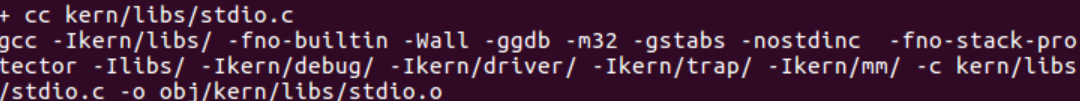
Makefile中的命令：

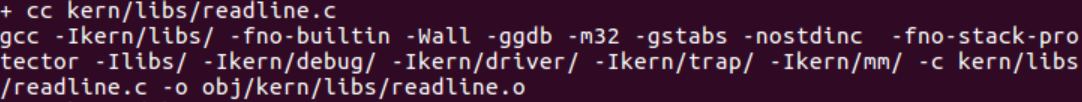


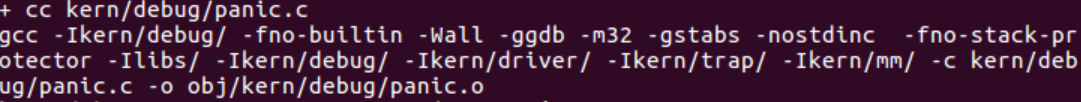
步骤：（输入make V=得到以下命令）

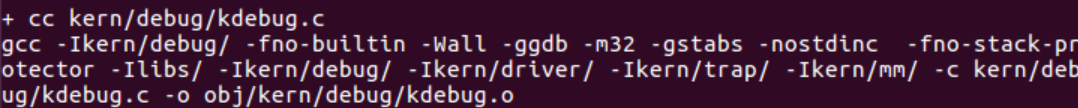
1.编译所有生成bin/kernel所需文件

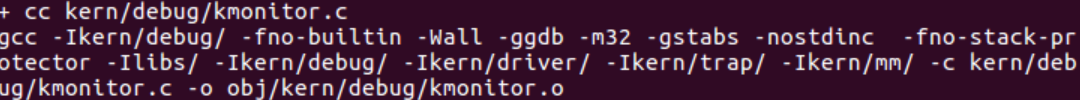


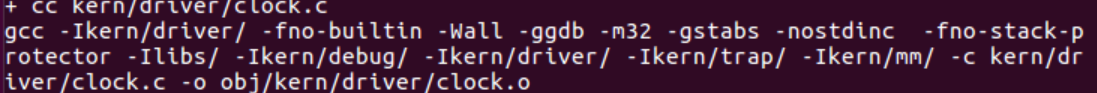


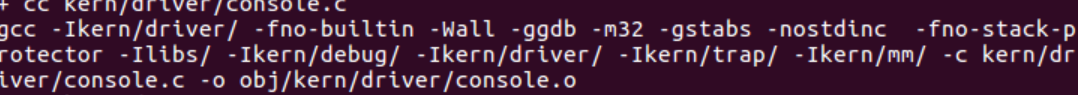


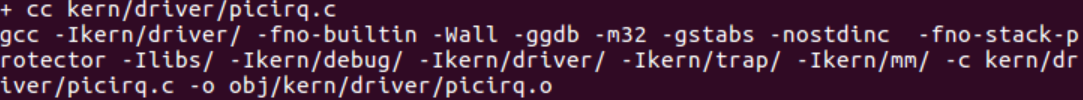


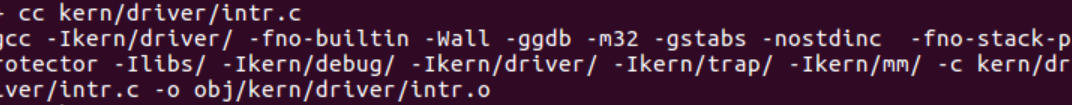




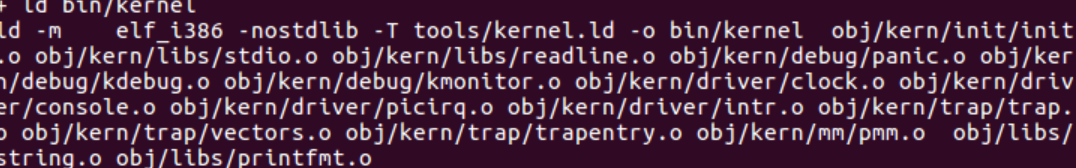




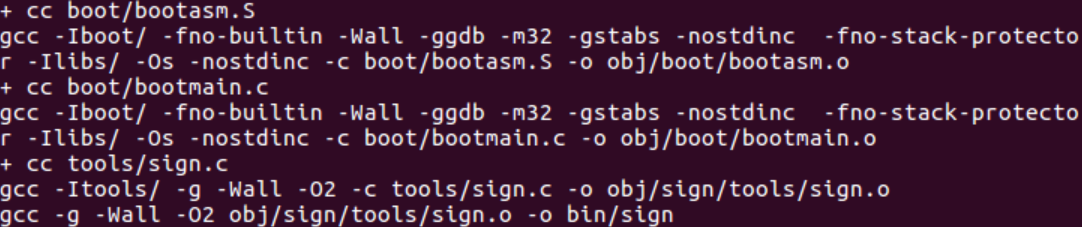




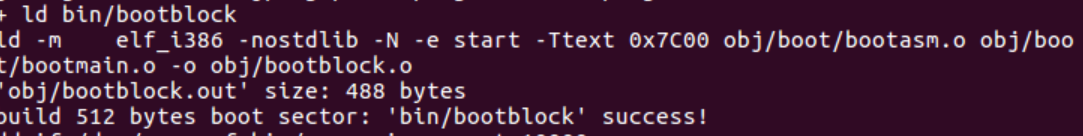
2.链接生成bin/kernel



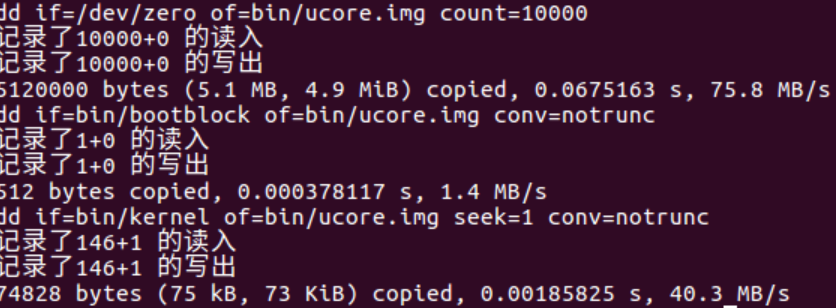
3.编译bootasm.S /bootasm.C /sign.C



4.根据sign规范生成obj/bootblock.o



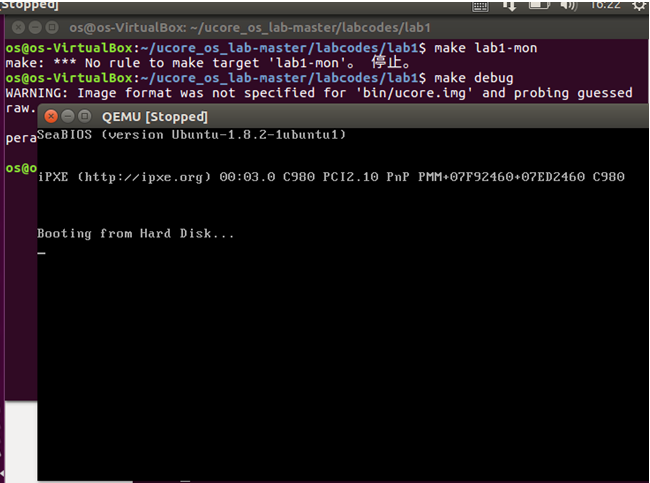
5.生成ucore.img



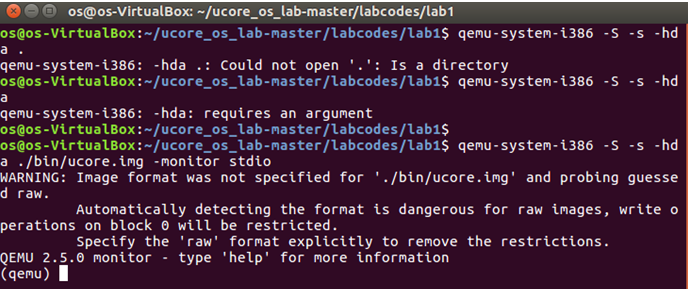
**练习2使用qemu执行并调试lab1中的软件**

2.1 从CPU加电后执行的第一条指令开始，单步跟踪BIOS的执行。

如果直接输入make debug会导致虚拟机直接死机不得不开机重新启动



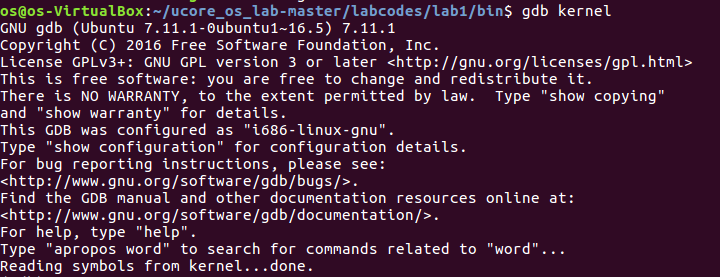
在lab1文件中进入终端，在其终端上输入qemu-system-i386 -S -s -hda ./bin/ucore.img -monitor stdio 来等待操作

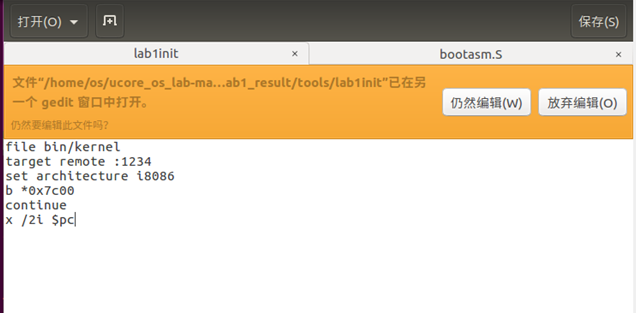


然后，打开另一个终端进入目标文件夹下，输入ls 可查看可操作文件

12

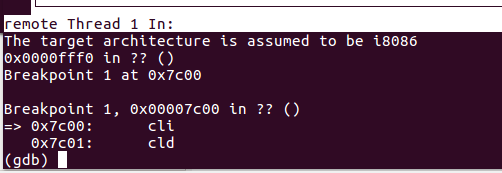
输入gdb kernel进行下一步操作



再打开下面的文件夹目录进行gdb操作

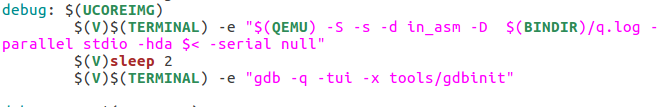
2.2在初始化位置0x7c00设置实地址断点,测试断点正常。

修改gdbinit设置断点得到结果：



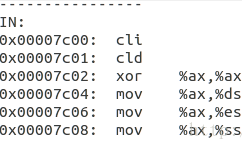
2.3从0x7c00开始跟踪代码运行,将单步跟踪反汇编得到的代码与bootasm.S和 bootblock.asm进行比较。

修改makefile:

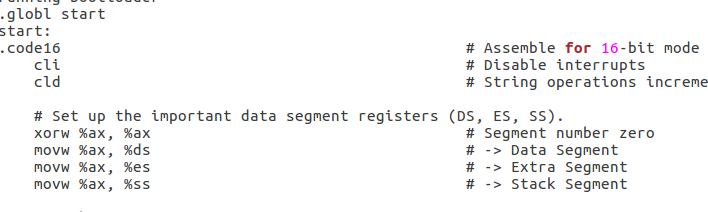


执行make debug

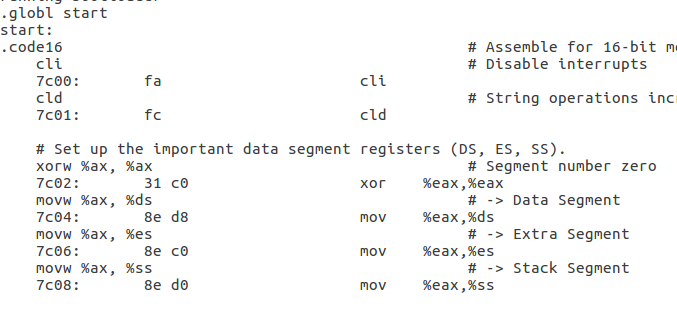
得到的q.log



查看bootasm.S



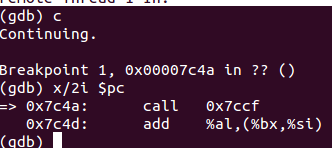
查看bootblock.asm



bootasm.S与bootblock.asm 中代码一样，q.log断点之后代码和它们一样

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

修改gdbinit 在0x7c4a处设置断点



断点设置正常。

练习3：分析bootloader进入保护模式的过程。

**1为何开启A20，以及如何开启A20**

初始时A20为0，访问超过1MB的地址时，就会从0循环计数，将A20地址线置为1之后，才可以访问4G内存。A20地址位由8042控制，8042有2个有两个I/O端口：0x60和0x64。

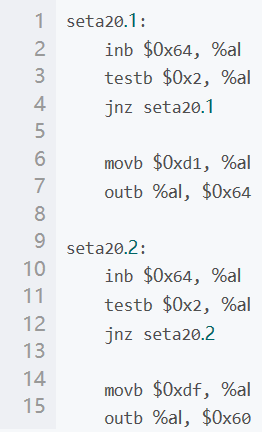
打开流程：

1.等待8042 Input buffer为空；

2.发送Write 8042 Output Port （P2）命令到8042 Input buffer；

3.等待8042 Input buffer为空；

4.将8042 Output Port（P2）得到字节的第2位置1，然后写入8042 Input buffer；



**2如何初始化GDT表**

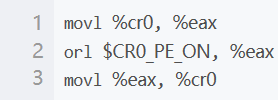
**1载入GDT表**1

****

2 进入保护模式：

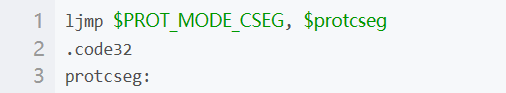
通过将cr0寄存器PE位置1便开启了保护模式

cro的第0位为1表示处于保护模式



### 3 通过长跳转更新cs的基地址:

​ 上面已经打开了保护模式，所以这里需要用到逻辑地址。$PROT\_MODE\_CSEG的值为0x80



4 设置段寄存器，并建立堆栈



5 转到保护模式完成，进入boot主方法



**3如何使能和进入保护模式**

将cr0寄存器置1