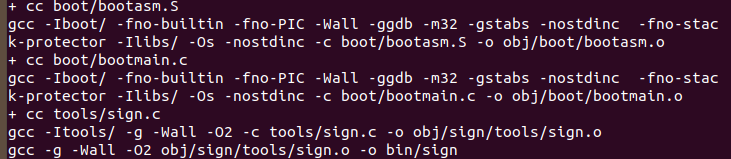
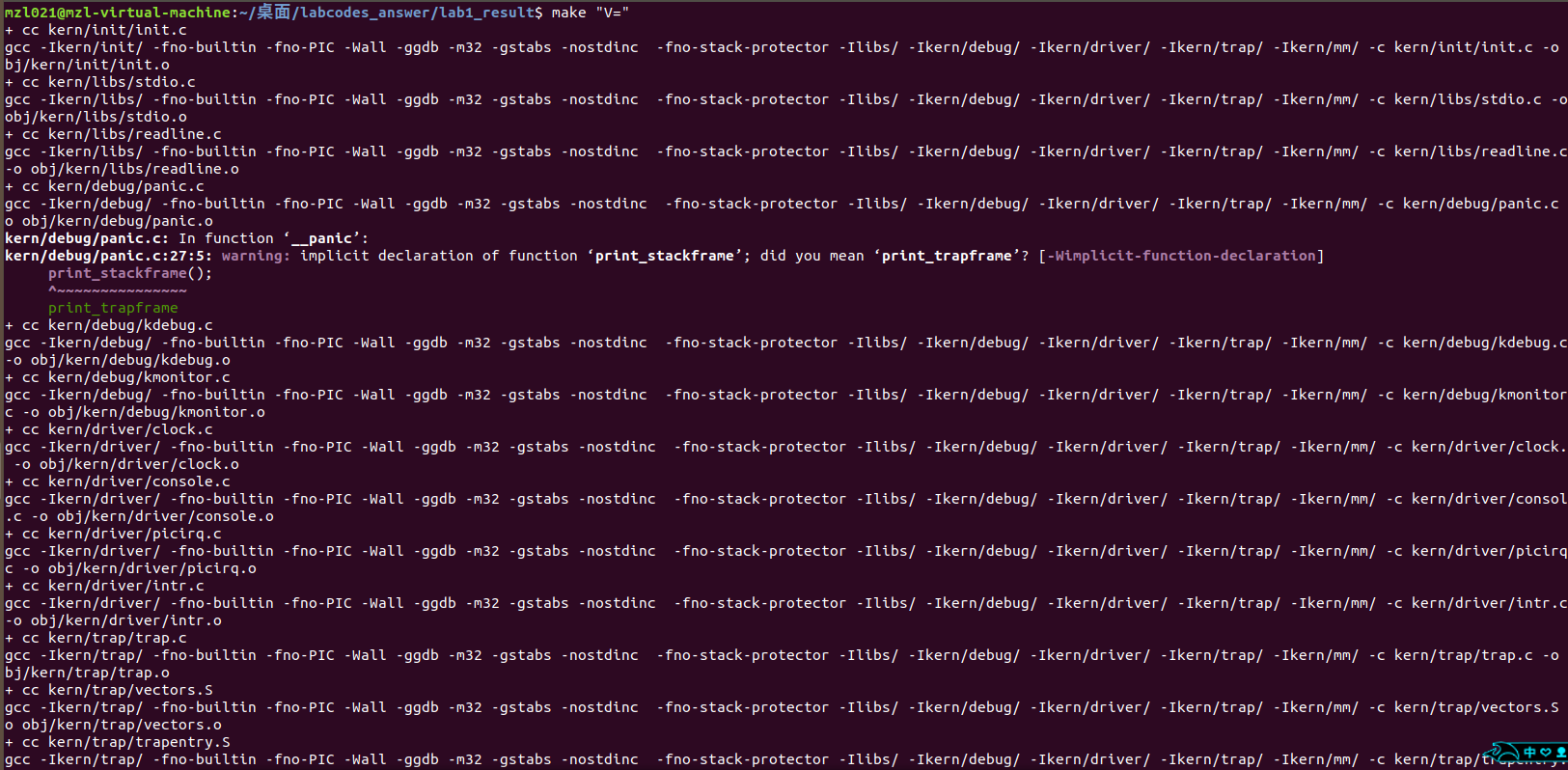
Lab1实验报告

**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)

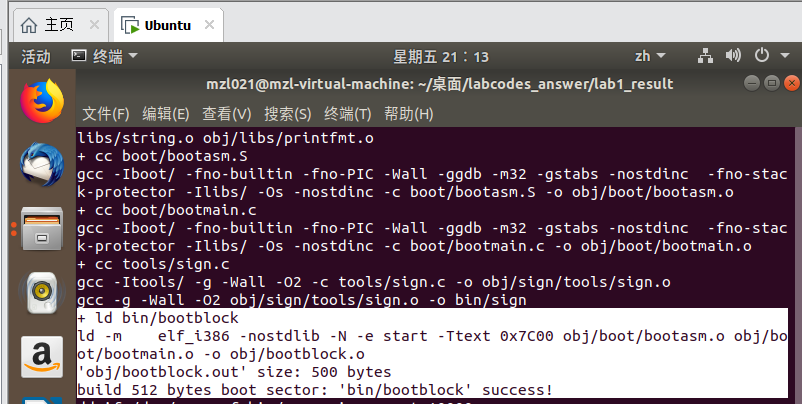
首先进入文件路径lab1，然后执行make：make “V=”。之后可以通过代码看到：

1. GCC编译器将Kernel目录下C的源代码编译成OBJ目录下的.o文件。

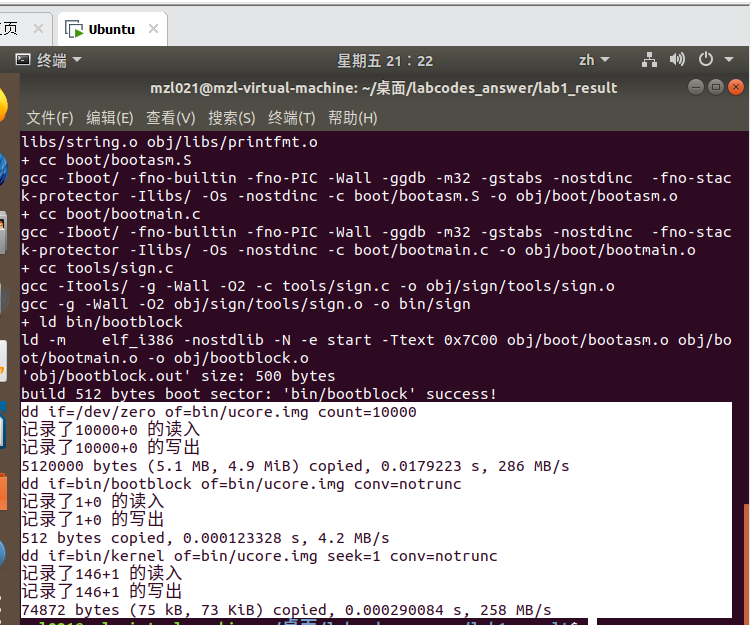


1. ld命令将这些生成的.o文件转变成可执行文件，并且链接成BIN目录下的kernel文件和bootblock文件，最终生成了两个软件Bootloader和kernel。。





1. dd命令把dev/zero, bootloder、kernel放到了ucore.img count的虚拟硬盘中



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?

打开lab1/tools/sign.c：



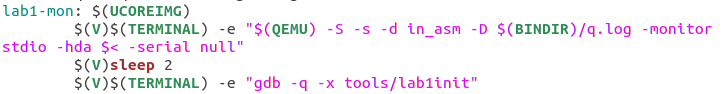
由此可见：磁盘主引导扇区只有512字节，磁盘最后两个字节为0x55AA，由不超过466字节的启动代码和不超过64字节的硬盘分区表加上两个字节的结束符组成。

**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。

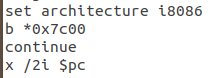
首先改写Makefile 文件，使得在调用qemu时增加-d in\_asm -D q.log参数，以将运行的汇编指令保存在q.log

中。

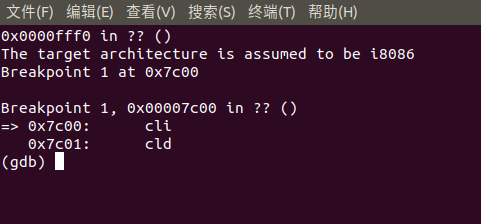


1. 在初始化位置0x7c00设置实地址断点,测试断点正常。

首先在tools/gdbinit中设置断点，断点处为0x7c00



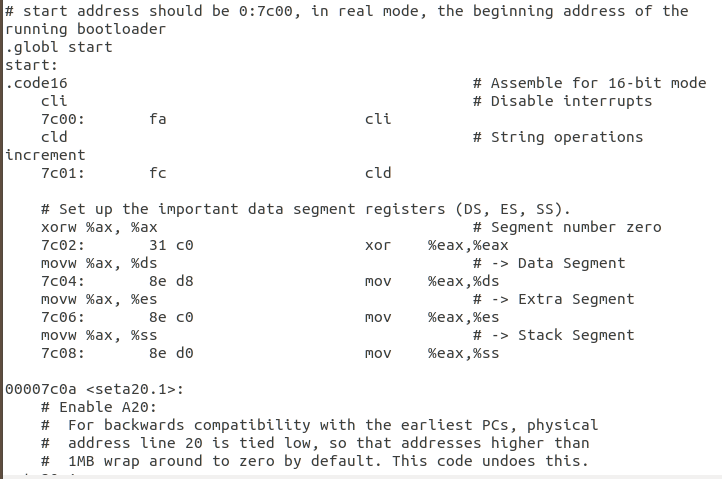
然后通过make lab1-mon指令运行出等待调试的qemu虚拟机，并同时打开一个gdb，可以看到qemu的运行状态并且发现gdb中运行停在了0x7c00处：



以此说明断点正常

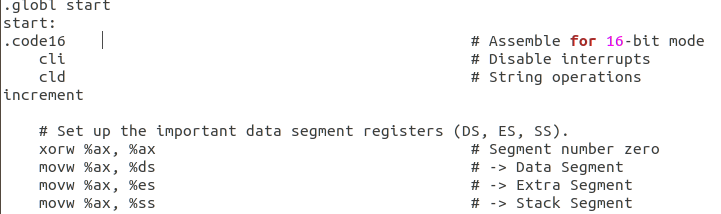
1. 从0x7c00开始跟踪代码运行,将单步跟踪反汇编得到的代码与bootasm.S和 bootblock.asm进行比较。

经比较知，二者代码相同。

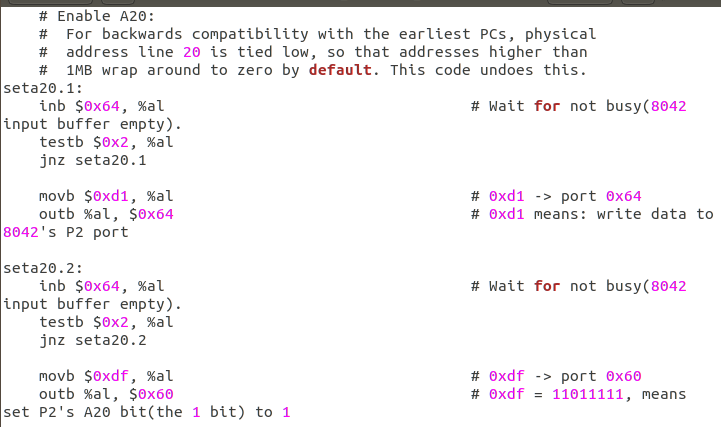


**练习3：分析bootloader****进入保护模式的过程。**

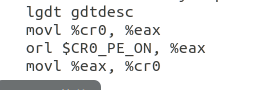
1. 先将各个寄存器置0，即关中断,并清除方向标志



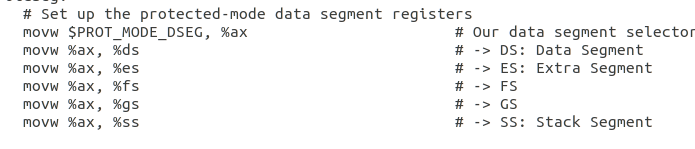
1. 将A20置1，这样 80386 就可以突破 1MB 访存现在,而可访问 4GB 的 32 位地址空间

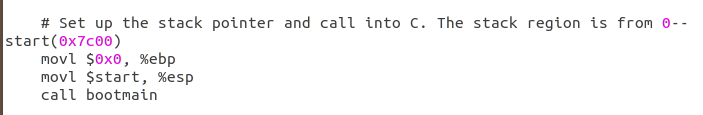


1. 加载GDT表并将CR0的第0位置为1



1. 重置段寄存器DS、ES等，并建立堆栈，至此成功进入保护模式





**练习4：分析bootloader加载ELF格式的OS的过程。**

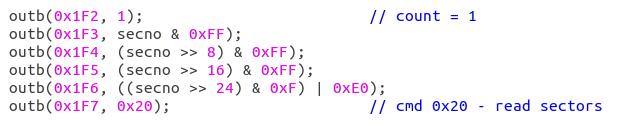
bootloader如何读取硬盘扇区的？

分析bootmain函数：

1. 读 I/O 地址 0x1f7,等待磁盘准备好;



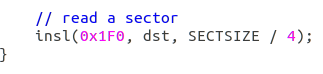
1. 写 I/O 地址 0x1f2~0x1f5,0x1f7,发出读取第 offseet 个扇区处的磁盘数据的命令;



1. 读 I/O 地址 0x1f7,等待磁盘准备好;



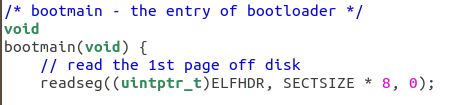
1. 连续读 I/O 地址 0x1f0,把磁盘扇区数据读到指定内存。



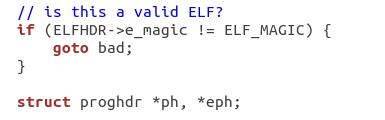
bootloader是如何加载ELF格式的OS？

在读取完磁盘之后，开始加载ELF格式的文件

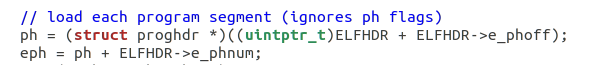
1. 读取ELF的头部



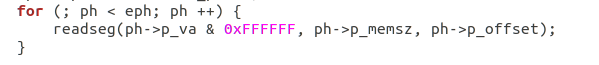
1. 判断ELF文件是否是合法



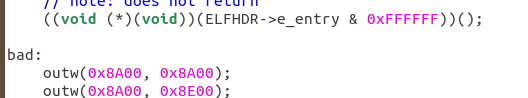
1. 将描述表的头地址存在ph



1. 按照描述表将ELF文件中数据载入内存

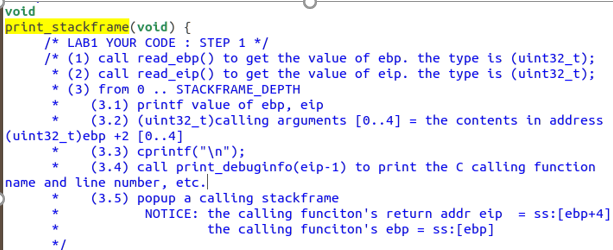


1. 根据ELF头部储存的入口信息，找到内核的入口

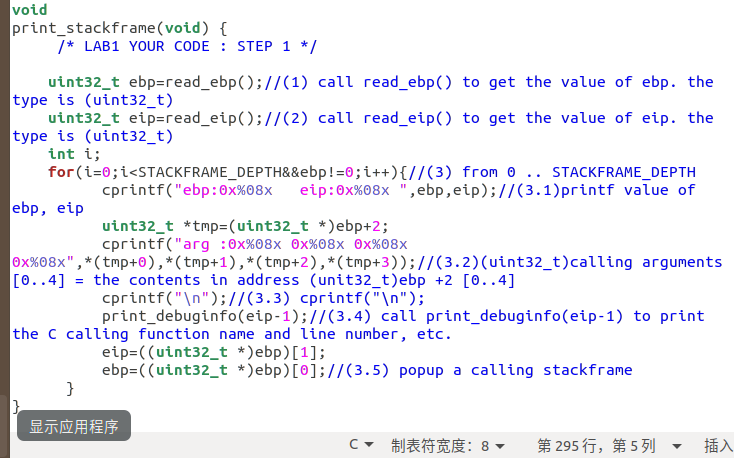


**练习5：实现函数调用堆栈跟踪函数**

1. 打开lab1/kern/kdebug.c,找到print\_stackframe函数

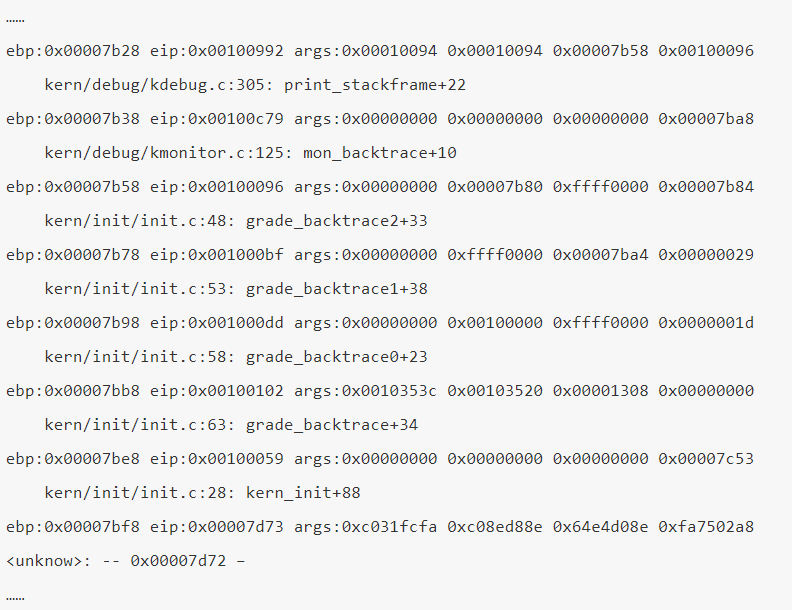


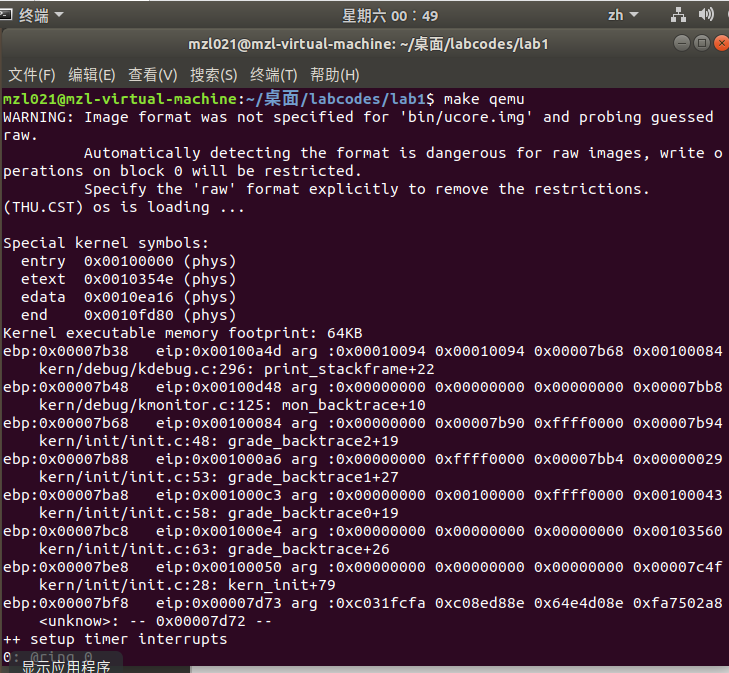
1. 根据代码注释处的提示补全代码：



1. 在lab1中执行 “make qemu”

由图可见，在qemu模拟器中得到了如实验手册上一致的输出：





bootloader设置的堆栈从0x7c00开始，使用”call bootmain”转入bootmain函数。

call指令压栈，所以bootmain中ebp为0x7bf8。

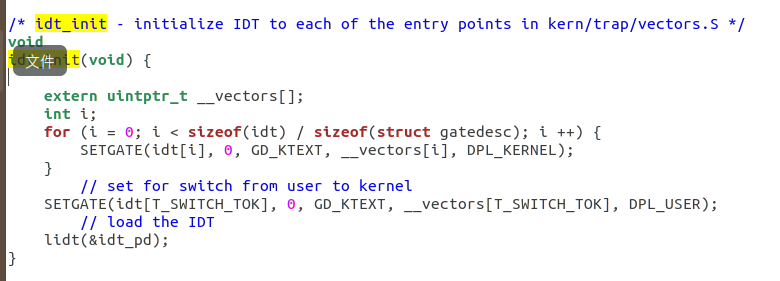
**练习6：完善中断初始化和处理**

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？

答：由向量表结构描述知，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，入口地址=段选择子+段内偏移量。

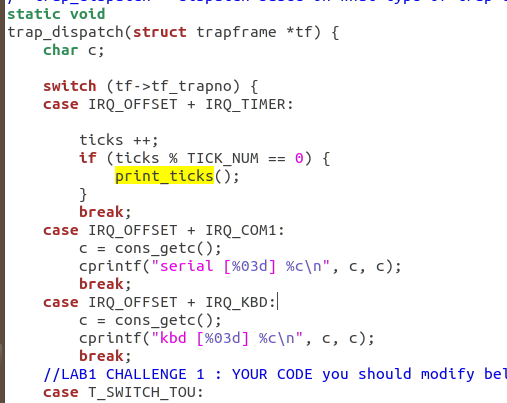
1. 请编程完善kern/trap/trap.c中对中断向量表进行初始化的函数idt\_init。在idt\_init函数中，依次对所有中断入口进行初始化。使用mmu.h中的SETGATE宏，填充idt数组内容。每个中断的入口由tools/vectors.c生成，使用trap.c中声明的vectors数组即可。

编写代码如下：



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”。

找到函数trap\_dispatch,补全代码如下：



最终运行结果如下：运行系统后，可以看到大约每1秒会输出一次”100 ticks”，并且按下键后会有回显信息：

