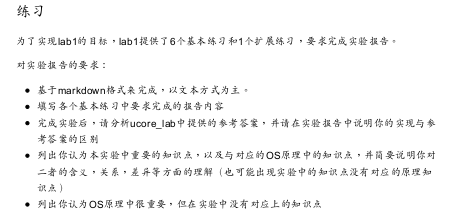
实验一

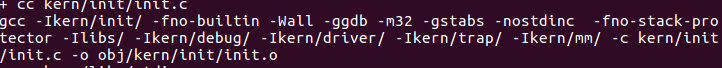
练习一：理解通过make生成执行文件的过程



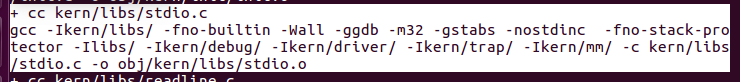
gcc把一些C的源代码编译成.o文件（目标文件）

第一 : 生成bin /kernel

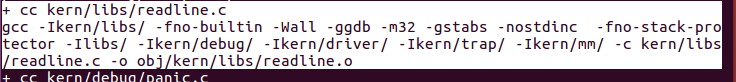
//生成init.o



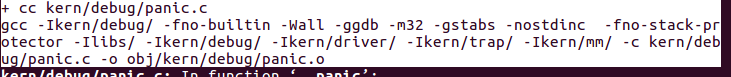
//生成stdio.o



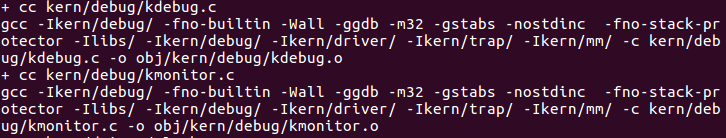
//生成readline.o



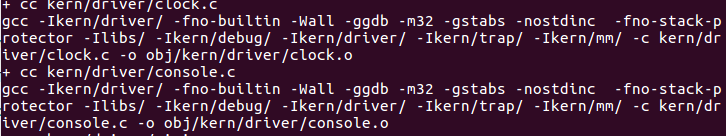
//生成panic.o



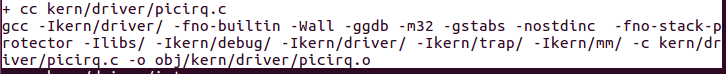
//生成kdebug.o和kmonitor.o

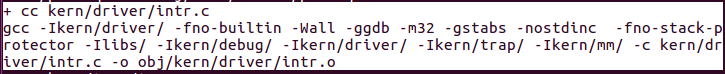


//生成clock.o和console.o

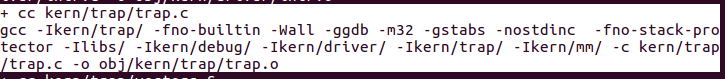


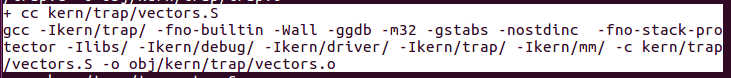
//生成picirq.o和intr.o



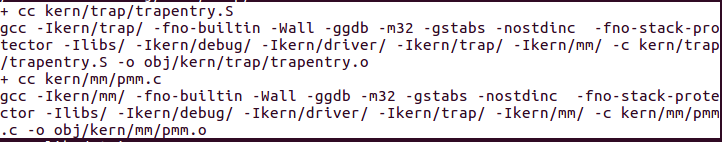


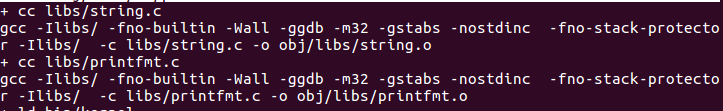
生成trap.o和vector.o



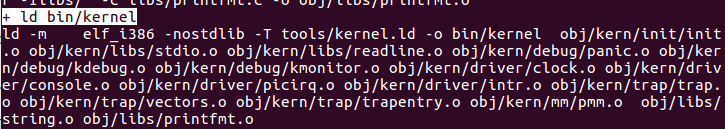


//生成trapentry.o和pmm.o

//生成string.o和printfmt.o



//连接.o文件生成bin/kernel



第二:生成bin/bootblock

//生成bootasm.o和bootmain.o

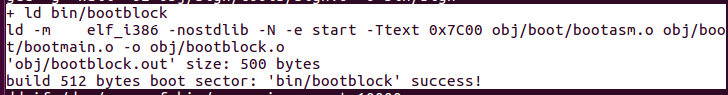




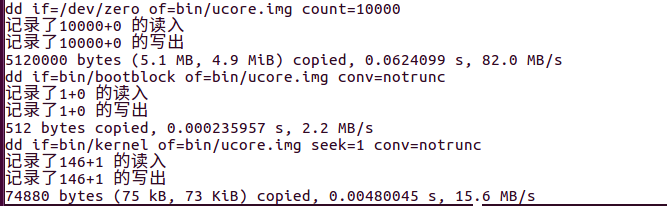
第三：生成bin/sihn



//连接，生成bootblock

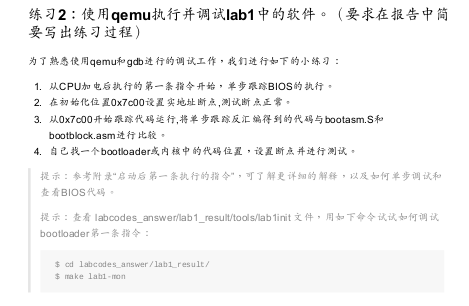


第四：生成ucore.img



这条命令即可得知bootloader和ucore是如何生成的

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

2. 在初始化位置0x7c00设置实地址断点,测试断点正常。

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

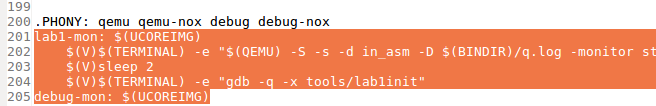
bootblock.asm进行比较。

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

makefile labi-mon

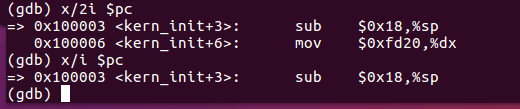
让qume把它执行的指令记录下来放到q.log

和GDB结合调试正在执行的Bootloader

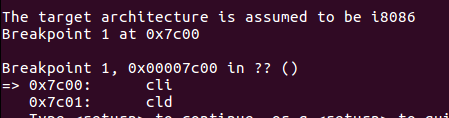


显示当前的十条指令

1.

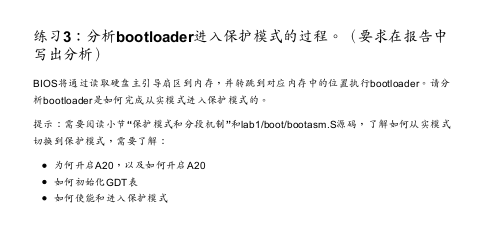


1. 设置断点



3.单步执行，将gdb的结果与bootblock.asm比较，直到0x7ccf,除一些表达上的差异，仍没有发现明显区别。

练习3：分析bootloader进入保护模式的过程。



为何开启A20，以及如何开启A20

如何初始化GDT表

如何使能和进入保护模式

答：

bootloader首先屏蔽所有中断，之后将段寄存器清零，打开A20地址线，加载

GDT的基地址，切换到保护模式，跳转到32位代码，在32位代码中，bootloader重新设置保护模式下的段寄存器，然后设置栈顶指针，之后跳转到C代码。