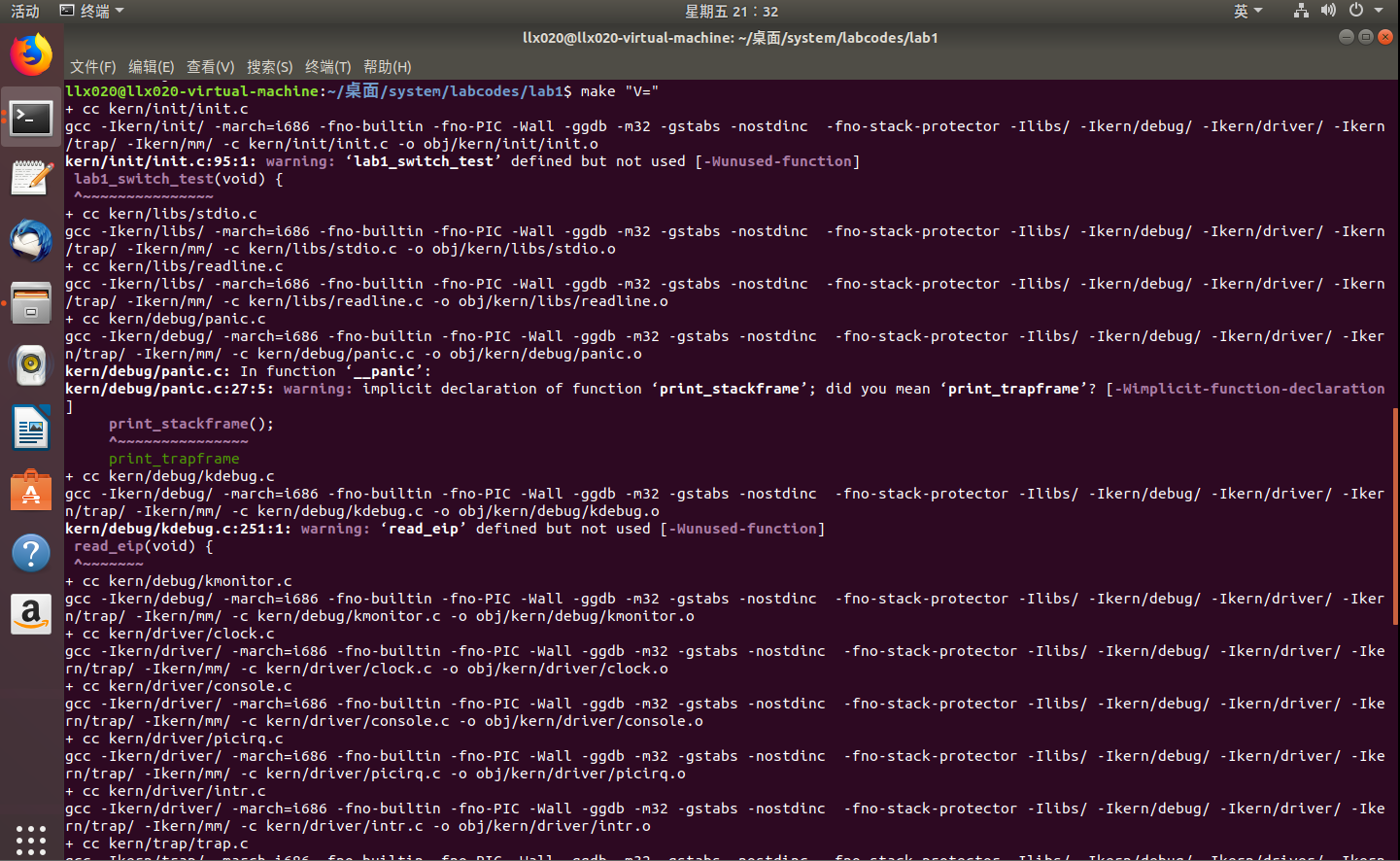
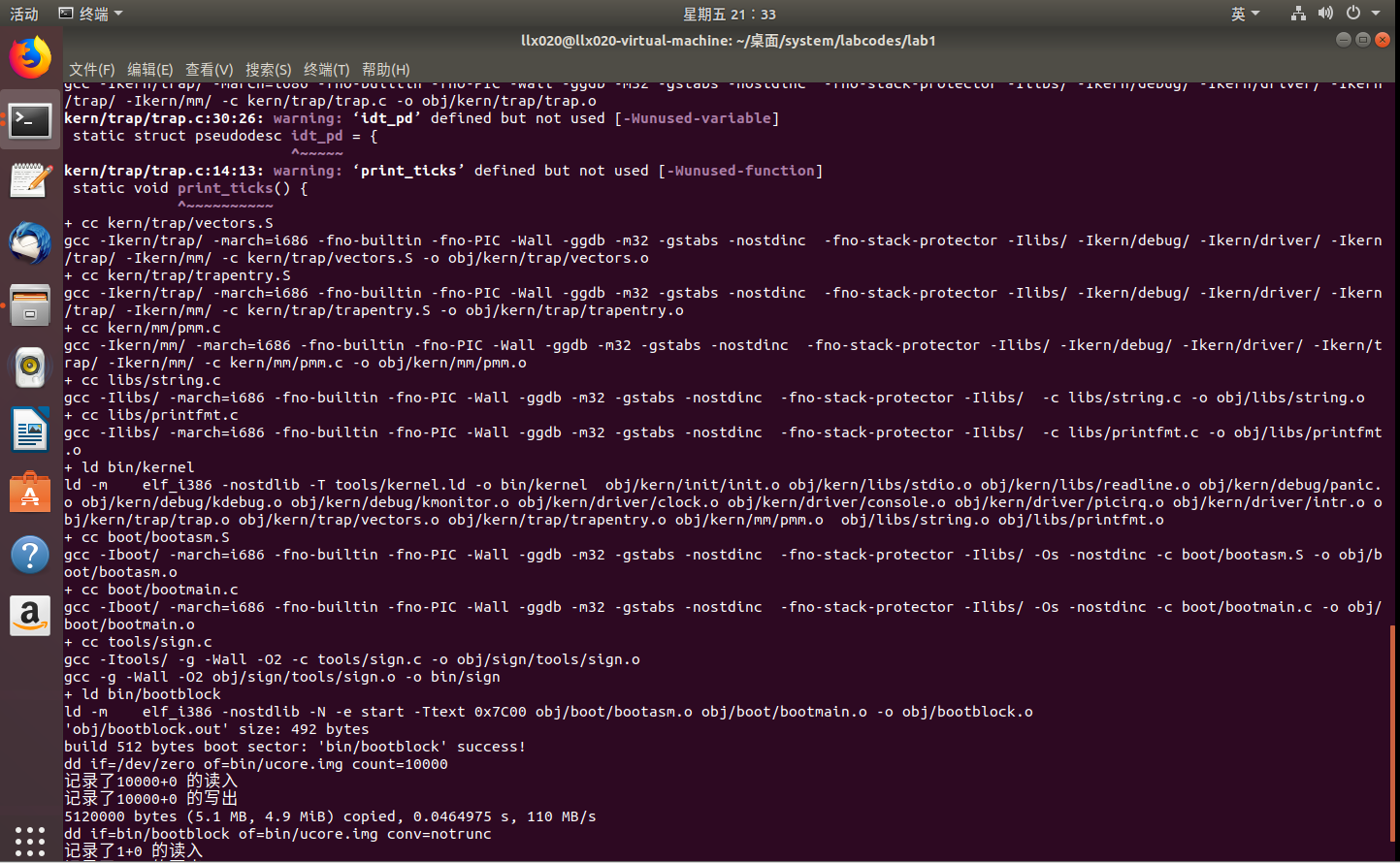
练习1： 理解通过make生成执行文件的过程

问题1：操作系统镜像文件ucore.img是如何一步一步生成的？

实验过程：下载gcc、make、qemu

在lab1下输入命令行make “V=”







1、首先把C的源代码进行编译成为.o文件。

2、ld命令将这些目标文件转变成可执行文件。

3、dd命令把bootloder放到ucore.img count的虚拟硬盘之中

4、还生成了两个软件，一个是Bootloader，另一个是kernel。

练习二：使用qemu执行并调试lab1中的软件

内容：

1、从 CPU加电后执行的第一条指令开始，单步跟踪 BIOS的执行

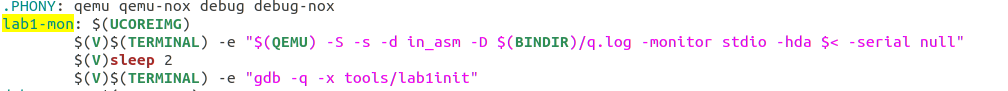
2、在初始化位置 0x7c00 设置实地址断点,测试断点正常

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

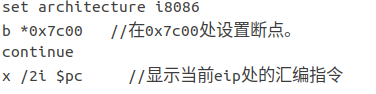
4、自己找一个 bootloader或内核中的代码位置，设置断点并进行测试

实验过程：

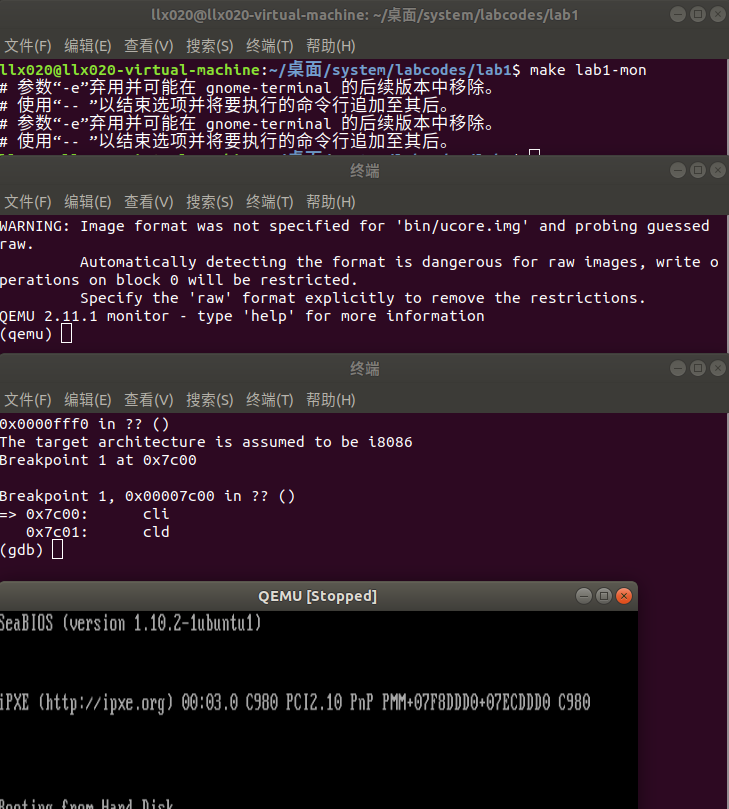
改写Makefile文件



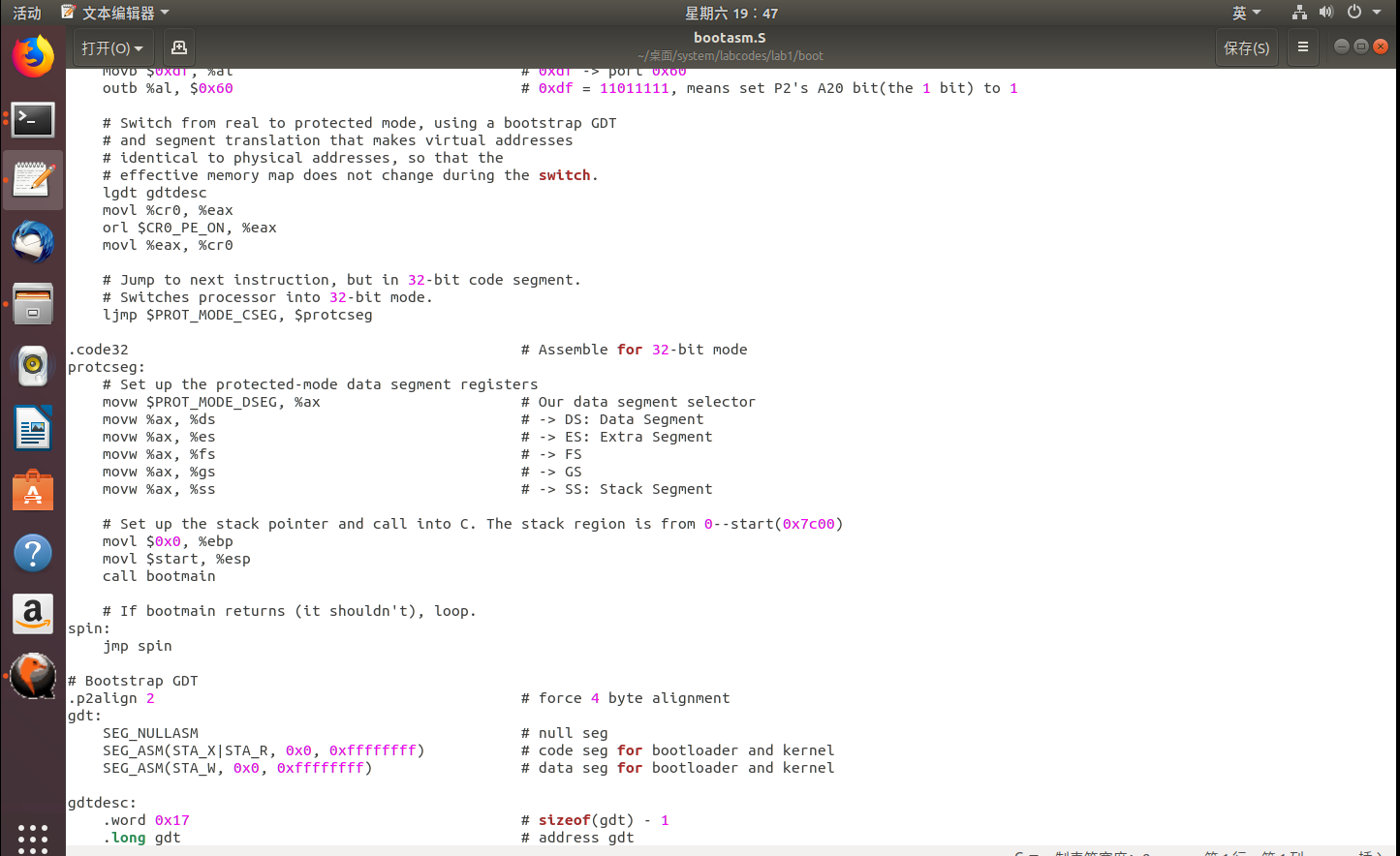
在tools/gdbinit结尾加上

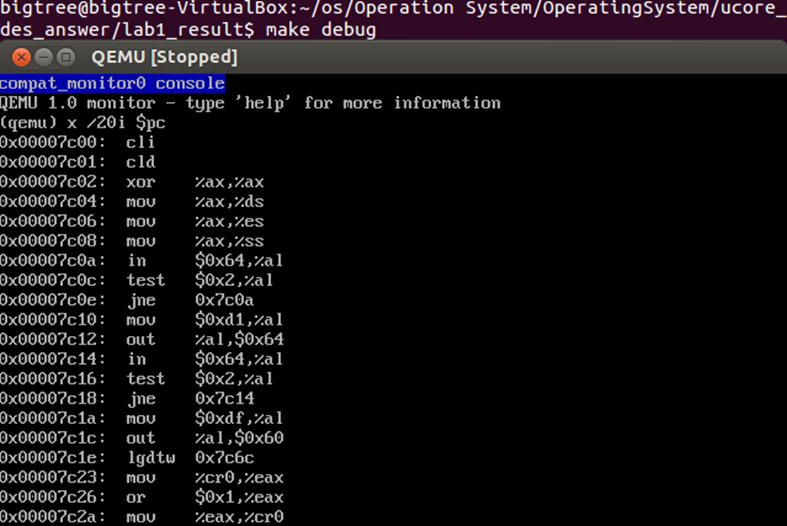


在终端中输入lab1 make-mon



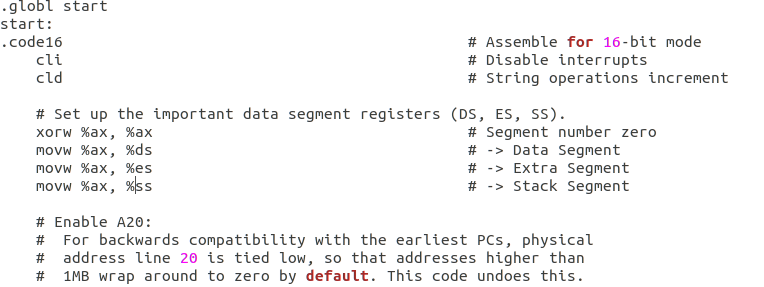
打开bootasm.S，和bootblock.asm 进行比较。



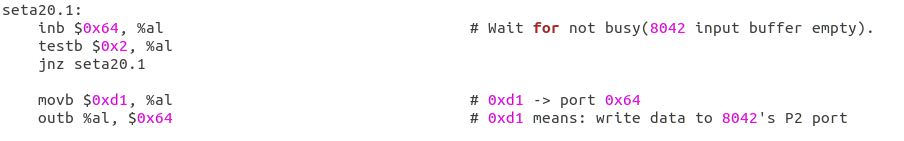


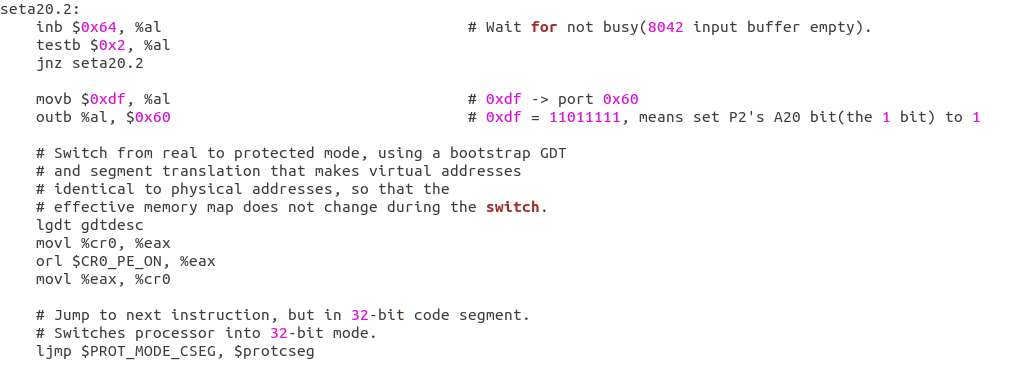
练习3：分析bootloader进入保护模式的过程

实验过程：



关中断,并清除方向标志，清零各数据段寄存器:DS、ES、FS

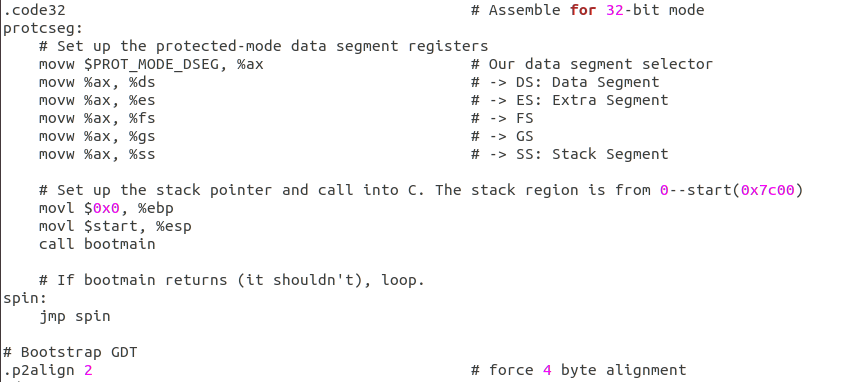




为何开启A20，以及如何开启A20

等待8042输入缓冲区空，从0x64端口中读入一个字节到al中，测试al的第2位，al的第2位为0，则跳出循环，将0xd1写入al中，等待8042输入缓冲区空，从0x64端口中读入一个字节到al中，测试al的第2位，al的第2位为0，则跳出循环，将0xdf入al中，将0xdf入到0x64端口中，打开A20

载入GDT表，加载cro到eax，将eax的第0位置为1，将cr0的第0位置为1，$PROT\_MODE\_CSEG的值为0x80



使用32位模式编译，赋0x80，赋0x80，赋0x80，赋0x80，gs赋0x80， ss赋0x80，设置帧指针和栈指针

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

1、等待磁盘准备好；

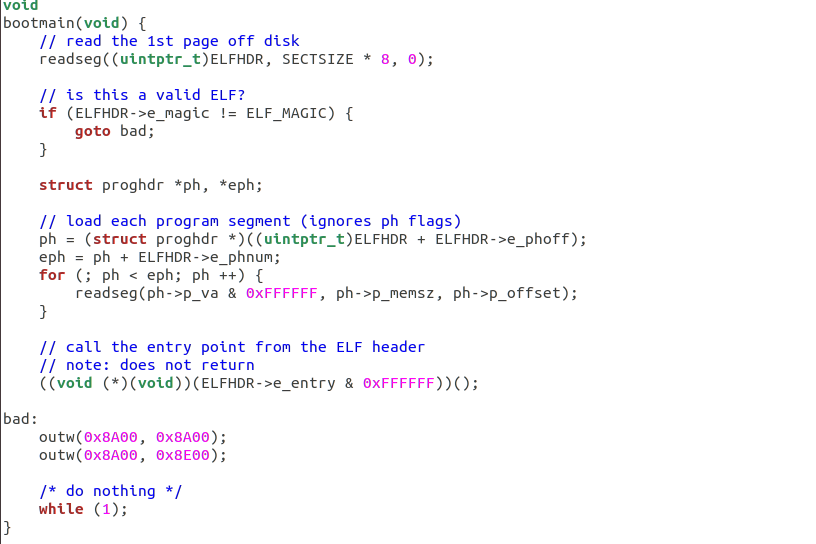
2、发出读取扇区的命令；

3、等待磁盘准备好；

4、把磁盘扇区数据读到指定内存。

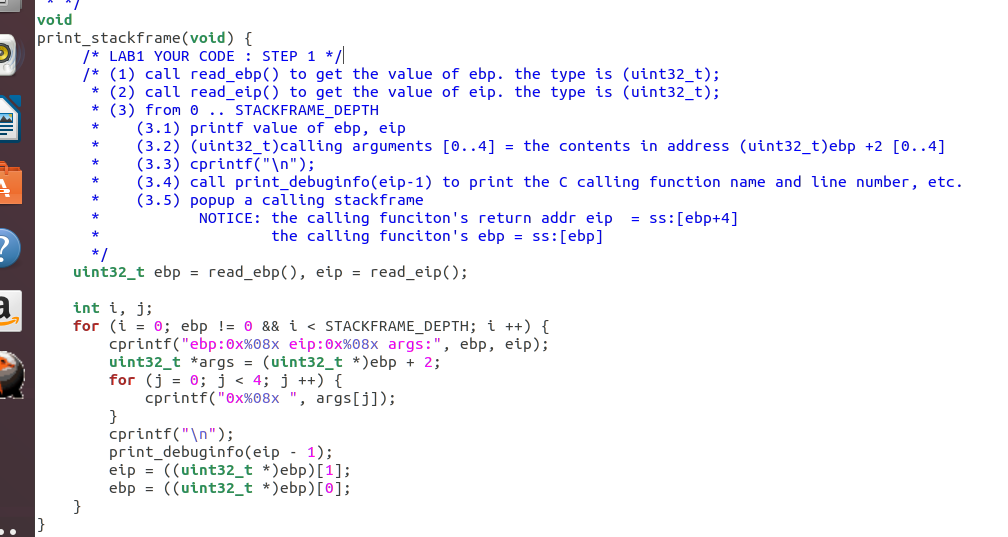
实验过程：



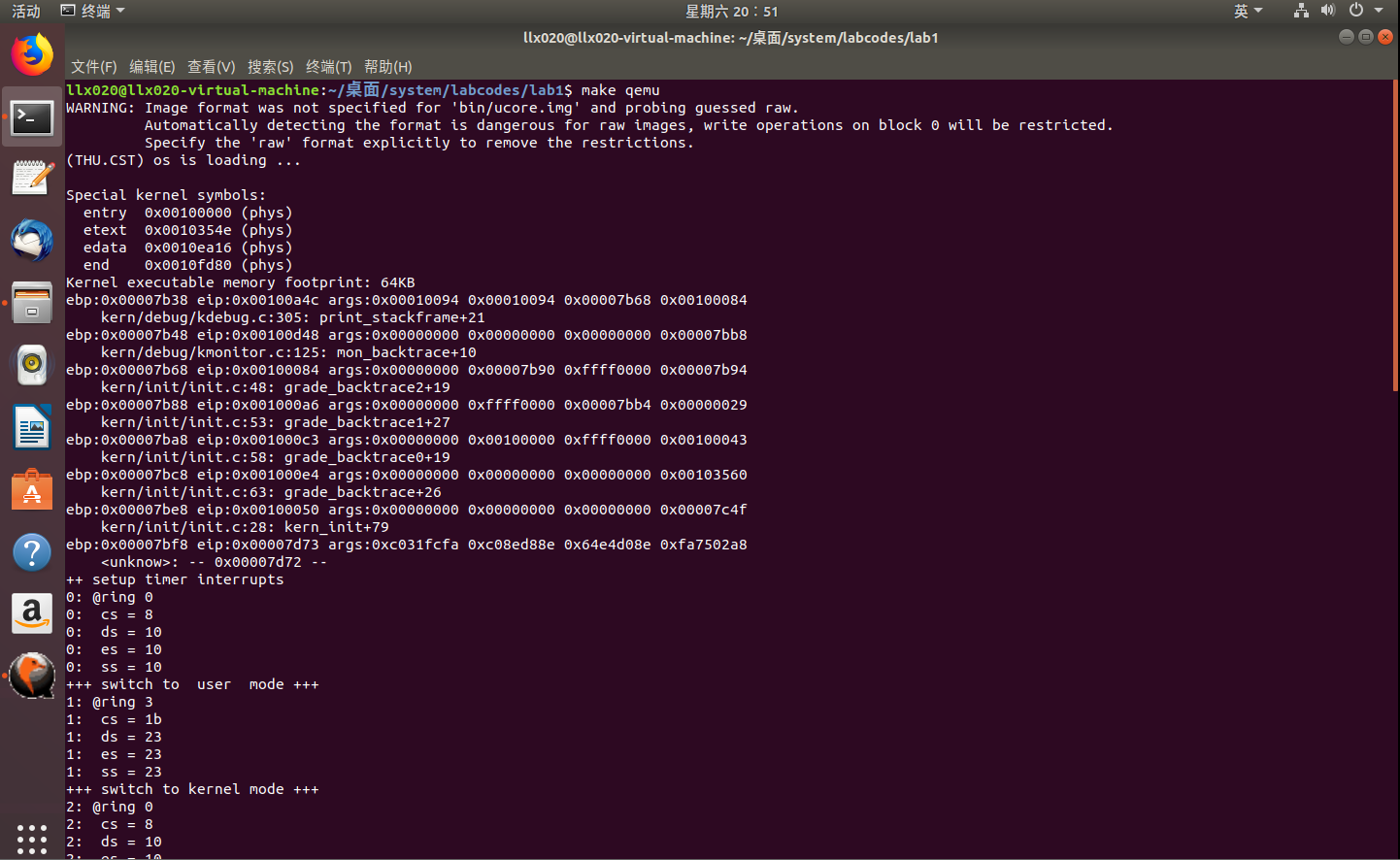


从磁盘上读取第一页，检验ELF是否有效吗，通过储存在头部的幻数判断是否是合法的ELF文件，加载每个程序段(忽略ph标志)，先将描述表的头地址存在ph， 按照描述表将ELF文件中数据载入内存， ELF文件0x1000位置后面的0xd1ec比特被载入内存0x00100000，ELF文件0xf000位置后面的0x1d20比特被载入内存0x0010e000， 从ELF报头调用入口点，不返回。

练习5：实现函数调用堆栈跟踪函数

实验过程：

在终端中输入make qemu



练习6：完善中断初始化和处理

实验过程：

1、中断描述符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？

struct gatedesc {

unsigned gd\_off\_15\_0 : 16; // low 16 bits of offset in segment

unsigned gd\_ss : 16; // segment selector

unsigned gd\_args : 5; // # args, 0 for interrupt/trap gates

unsigned gd\_rsv1 : 3; // reserved(should be zero I guess)

unsigned gd\_type : 4; // type(STS\_{TG,IG32,TG32})

unsigned gd\_s : 1; // must be 0 (system)

unsigned gd\_dpl : 2; // descriptor(meaning new) privilege level

unsigned gd\_p : 1; // Present

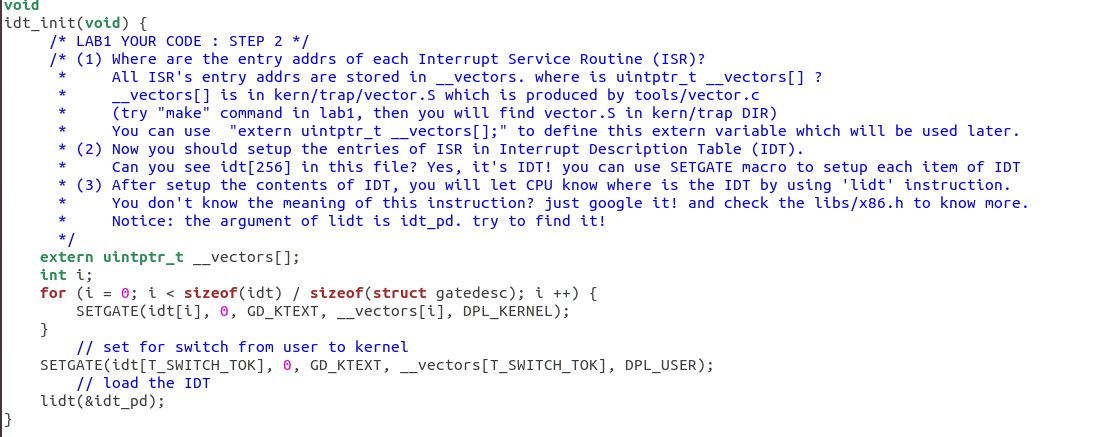
unsigned gd\_off\_31\_16 : 16; // high bits of offset in segment

};

由各项相加可知，一个表项的占64bit，即8字节，其中0 ~ 15位和48 ~ 63位分别为偏移量的低16位和高16位，两者拼接为偏移量，16~31位为段选择器。

由gd\_ss与gd\_off构成中断处理代码入口。入口地址=段选择子+段内偏移量。

2、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。



3、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”。

