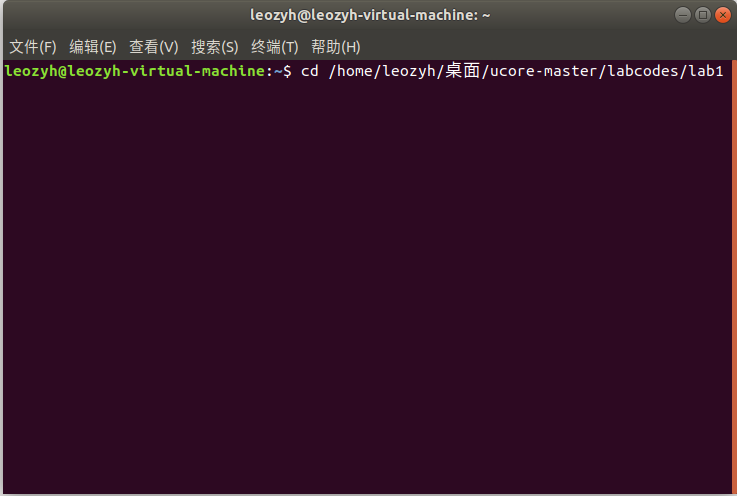
**练习一**

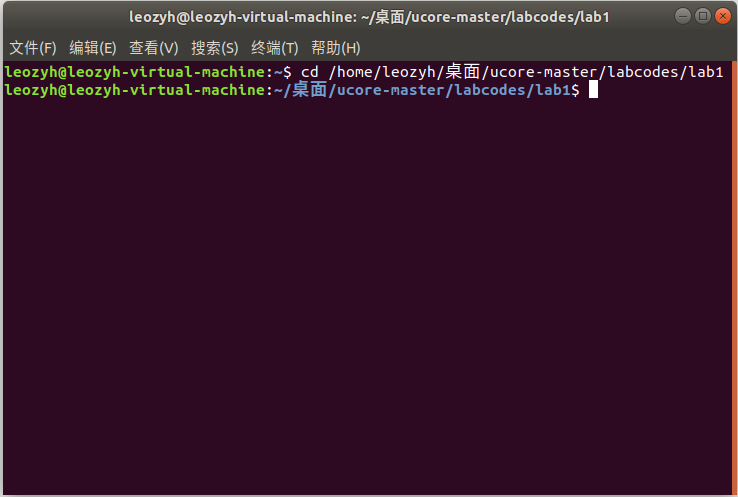
1. 练习1.1

这个步骤比较简单，主要是通过观察一些指令来了解镜像文件ucore.img是怎么生成的。

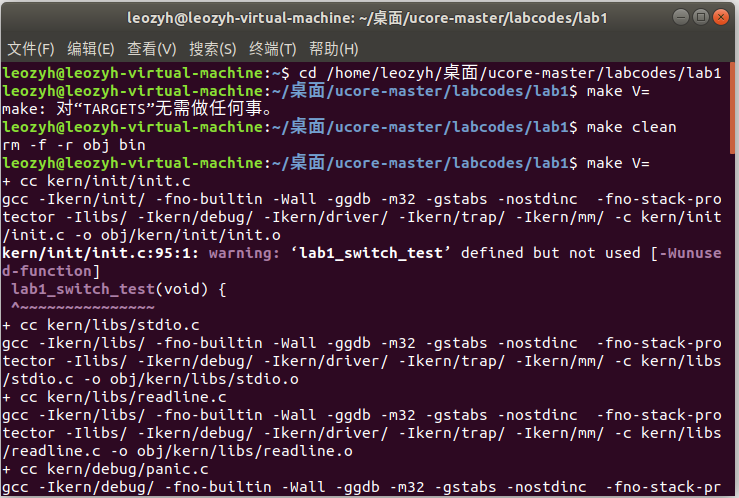
上操作：

1. 打开命令行，跳转到lab1文件夹中：

输入指令



跳转完成

1. 执行“make V=”指令：（PS：如果输入“make V=”后出现“make: 对“TARGETS”无需做任何事。”，不要慌张，输入“make clean”后再次运行就可以啦。）

运行成功！代码段依次执行

1. 指令分析：+ cc kern/init/init.c //编译init.c
2. gcc -c kern/init/init.c -o obj/kern/init/init.o
3. + cc kern/libs/readline.c //编译readline.c
4. gcc -c kern/libs/readline.c -o
5. obj/kern/libs/readline.o
6. + cc kern/libs/stdio.c //编译stdlio.c
7. gcc -c kern/libs/stdio.c -o obj/kern/libs/stdio.o
8. + cc kern/debug/kdebug.c //编译kdebug.c
9. gcc -c kern/debug/kdebug.c -o obj/kern/debug/kdebug.o
10. + cc kern/debug/kmonitor.c //编译komnitor.c
11. gcc -c kern/debug/kmonitor.c -o
12. obj/kern/debug/kmonitor.o
13. + cc kern/debug/panic.c //编译panic.c
14. gcc -c kern/debug/panic.c -o obj/kern/debug/panic.o
15. + cc kern/driver/clock.c //编译clock.c
16. gcc -c kern/driver/clock.c -o obj/kern/driver/clock.o
17. + cc kern/driver/console.c //编译console.c
18. gcc -c kern/driver/console.c -o
19. obj/kern/driver/console.o
20. + cc kern/driver/intr.c //编译intr.c
21. gcc -c kern/driver/intr.c -o obj/kern/driver/intr.o
22. + cc kern/driver/picirq.c //编译prcirq.c
23. gcc -c kern/driver/picirq.c -o
24. obj/kern/driver/picirq.o
25. + cc kern/trap/trap.c //编译trap.c
26. gcc -c kern/trap/trap.c -o obj/kern/trap/trap.o
27. + cc kern/trap/trapentry.S //编译trapentry.S
28. gcc -c kern/trap/trapentry.S -o
29. obj/kern/trap/trapentry.o
30. + cc kern/trap/vectors.S //编译vectors.S
31. gcc -c kern/trap/vectors.S -o obj/kern/trap/vectors.o
32. + cc kern/mm/pmm.c //编译pmm.c
33. gcc -c kern/mm/pmm.c -o obj/kern/mm/pmm.o
34. + cc libs/printfmt.c //编译printfmt.c
35. gcc -c libs/printfmt.c -o obj/libs/printfmt.o
36. + cc libs/string.c //编译string.c
37. gcc -c libs/string.c -o obj/libs/string.o
38. + ld bin/kernel //链接成kernel
39. ld -o bin/kernel
40. obj/kern/init/init.o obj/kern/libs/readline.o
41. obj/kern/libs/stdio.o obj/kern/debug/kdebug.o
42. obj/kern/debug/kmonitor.o obj/kern/debug/panic.o
43. obj/kern/driver/clock.o obj/kern/driver/console.o
44. obj/kern/driver/intr.o obj/kern/driver/picirq.o
45. obj/kern/trap/trap.o obj/kern/trap/trapentry.o
46. obj/kern/trap/vectors.o obj/kern/mm/pmm.o
47. obj/libs/printfmt.o obj/libs/string.o
48. + cc boot/bootasm.S //编译bootasm.c
49. gcc -c boot/bootasm.S -o obj/boot/bootasm.o
50. + cc boot/bootmain.c //编译bootmain.c
51. gcc -c boot/bootmain.c -o obj/boot/bootmain.o
52. + cc tools/sign.c //编译sign.c
53. gcc -c tools/sign.c -o obj/sign/tools/sign.o
54. gcc -O2 obj/sign/tools/sign.o -o bin/sign
55. + ld bin/bootblock //根据sign规范生成bootblock
56. ld -m elf\_i386 -nostdlib -N -e start -Ttext 0x7C00
57. obj/boot/bootasm.o obj/boot/bootmain.o
58. -o obj/bootblock.o
59. //创建大小为10000个块的ucore.img，初始化为0，每个块为512字节
60. dd if=/dev/zero of=bin/ucore.img count=10000
61. //把bootblock中的内容写到第一个块
62. dd if=bin/bootblock of=bin/ucore.img conv=notrunc
63. //从第二个块开始写kernel中的内容
64. dd if=bin/kernel of=bin/ucore.img seek=1 conv=notrunc

（3）从代码中可以看出，ucore.img的生成步骤大致如下：

1、先使用gcc命令，把kern目录下的代码都编译成obj/kern/\*\*/\*\*.o文件；2、使用gcc命令，把lib目录下的代码都编译成obj/libs/\*\*.o文件；

3、用ld命令通过/tools/kern.ls文件配置，把obj/kern/\*/\*.o文件连接成bin/kernel；

4、用gcc命令，把boot目录下的文件编译成obj/boot/\*.o文件；

5、用gcc把tools/sign.c编译成obj/sign/tools/sign.o；

6、用ld把obj/boot/\*.o连接成obj/bootblock.o；

7、用dd命令创建了一个bin/ucore.img文件；

8、用dd命令把bin/bootblock写入bin/ucore.img文件；

9、用dd命令创bin/kernel写入bin/ucore.img文件；

执行less Makefile命令可以看出Ucore.img的生成过程：

1 编译所有生成bin/kernel所需的文件

2 链接生成bin/kernel

3 编译bootasm.S bootmain.c sign.c

4 根据sign规范生成obj/bootblock.o，将obj/bootblock.o文件规范化为符合规范的硬盘主引导扇区的文件bin/bootblock

5 生成ucore.img

1. 练习1.2

截取sign.c文件中的部分源码

char buf[512]; //定义buf数组

memset(buf, 0, sizeof(buf));

// 把buf数组的最后两位置为 0x55, 0xAA

buf[510] = 0x55;

buf[511] = 0xAA;

FILE \*ofp = fopen(argv[2], "wb+");

size = fwrite(buf, 1, 512, ofp);

if (size != 512) { //大小为512字节

fprintf(stderr, "write '%s' error,

size is %d.\n", argv[2], size);

return -1;

}123456789101112

主引导扇区的规则如下：

1 大小为512字节

2 多余的空间填0

3 第510个（倒数第二个）字节是0x55，

4 第511个（倒数第一个）字节是0xAA。