操作系统

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)

答：

# create ucore.img//创建ucore.img

：

$(UCOREIMG): $(kernel) $(bootblock)

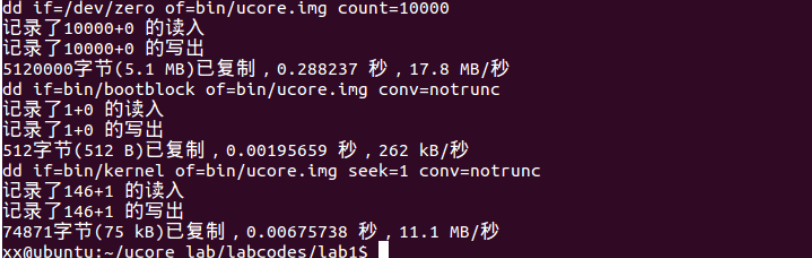
$(V)dd if=/dev/zero of=$@ count=10000

$(V)dd if=$(bootblock) of=$@ conv=notrunc

$(V)dd if=$(kernel) of=$@ seek=1 conv=notrunc

$(call create\_target,ucore.img)





生成ucore.img需要先生成kernel和bootlock

首先创建一个count的1000的块

创建bootlock代码：

# create bootblock

bootfiles = $(call listf\_cc,boot)

$(foreach f,$(bootfiles),$(call cc\_compile,$(f),$(CC),$(CFLAGS) -Os -nostdinc))

bootblock = $(call totarget,bootblock)

$(bootblock): $(call toobj,$(bootfiles)) | $(call totarget,sign)

@echo + ld $@

$(V)$(LD) $(LDFLAGS) -N -e start -Ttext 0x7C00 $^ -o $(call toobj,bootblock)

@$(OBJDUMP) -S $(call objfile,bootblock) > $(call asmfile,bootblock)

@$(OBJDUMP) -t $(call objfile,bootblock) | $(SED) '1,/SYMBOL TABLE/d; s/ .\* / /; /^$$/d' > $(call symfile,bootblock)

@$(OBJCOPY) -S -O binary $(call objfile,bootblock) $(call outfile,bootblock)

@$(call totarget,sign) $(call outfile,bootblock) $(bootblock)

$(call create\_target,bootblock)

根据make V= 查看得知要生成bootblock要先生成bootasm.o|bootmain.o|sign

创建kernel：

# create kernel target

kernel = $(call totarget,kernel)

$(kernel): tools/kernel.ld

$(kernel): $(KOBJS)

@echo + ld $@

$(V)$(LD) $(LDFLAGS) -T tools/kernel.ld -o $@ $(KOBJS)

@$(OBJDUMP) -S $@ > $(call asmfile,kernel)

@$(OBJDUMP) -t $@ | $(SED) '1,/SYMBOL TABLE/d; s/ .\* / /; /^$$/d' > $(call symfile,kernel)

$(call create\_target,kernel)

根据make V=查看得知生成kenel要kern文件下的.c文件编译的.o文件的支持

练习**2**：使用**qemu**执行并调试**lab1**中的软件。（要求在报告中简

要写出练习过程）

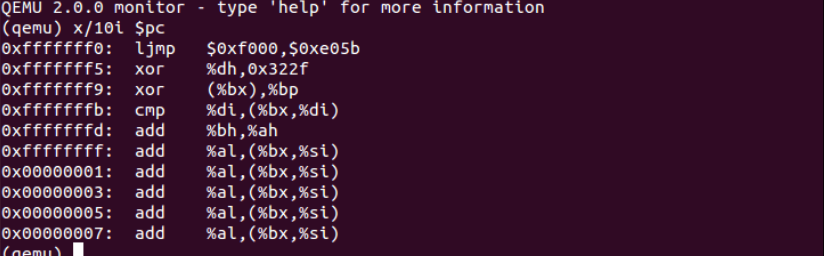
为了熟悉使用qemu和gdb进行的调试工作，我们进行如下的小练习：

1. 从CPU加电后执行的第一条指令开始，单步跟踪BIOS的执行。

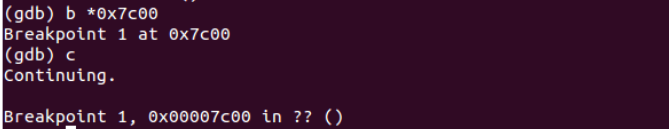
2. 在初始化位置0x7c00设置实地址断点,测试断点正常。

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。

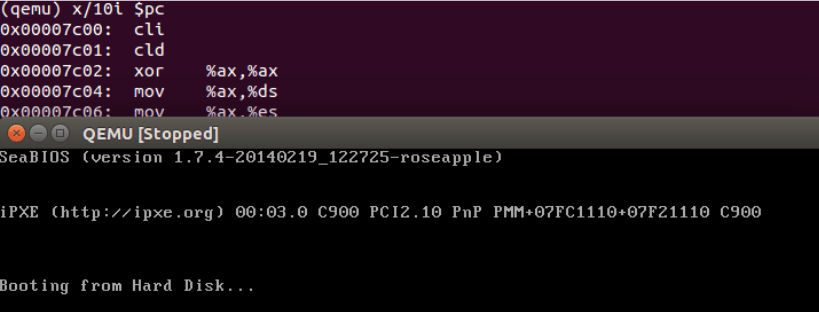
在QEMU窗口中使用 x/10i $pc 查看最近10条指令的反汇编内容。在gdb中使用si命令执行单步，显示位置



输入b \*0x7c00设置断点 输入c执行到断点



输入x/10i $pc 查看最近10条指令的反汇编内容，可以看到qemu执行到断点0x7c00。断点工作正常。



再次输入c的时候qemu正常。得到结论，断点工作正常

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。

gdb得到的反汇编代码与bootasm.S和bootblock.asm中的代码基本相一致。

练习**3**：分析**bootloader**进入保护模式的过程。（要求在报告中

写出分析）

首先从%cs=0 $,pc=0x7c00进入bootloader。

准备：将中断标志位清0，不允许中断，设置增址，将段寄存器置0。

 .code16

cli

     cld

xorw %ax, %ax

movw %ax, %ds

 movw %ax, %es

movw %ax, %ss

开启A20：通过将键盘控制器上的A20线置于高电位 seta20.1:

inb $0x64, %al

testb $0x2, %al jnz

seta20.1

movb $0xd1, %al

outb %al, $0x64                                  seta20.2:

inb $0x64, %al

testb $0x2, %al jnz seta20.2

movb $0xdf, %al        # 打开A20                  outb %al, $0x60

lgdt gdtdesc //加载gdtr

进入保护模式：

  movl %cr0, %eax

 orl $CR0\_PE\_ON, %eax

   movl %eax, %cr0

通过长跳转更新cs的基地址。

     ljmp $PROT\_MODE\_CSEG, $protcseg

.code32                                          protcseg:

设置段寄存器，并建立堆栈 。

     movw $PROT\_MODE\_DSEG, %ax                            movw %ax, %ds

movw %ax, %es

      movw %ax, %fs                                         movw %ax, %gs                                         movw %ax, %ss                                         movl $0x0, %ebp      movl $start, %esp

完成进入保护模式，进入bootmain。

call bootmain