Lab1 田新韬 2012011268

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)

生成kernel：

$(call add\_files\_cc,$(call listf\_cc,$(LIBDIR)),libs,)

编译libs目录下所有的.c/.s，生成.o/.d

$(call add\_files\_cc,$(call listf\_cc,$(KSRCDIR)),kernel,$(KCFLAGS))

编译kernel目录下所有.c/.s，生成.o/.d，具体指定了编译选项，存放在kcflag中，其为：

-fno-builtin -nostdinc 关闭内建库

-fno-PIC 不生成位置无关的symbol

-Wall 开启警告

-ggdb -gstabs 开启gdb

-m32 目标平台32位

-fno-stack-protector 不生成栈保护

-o 输出

$(kernel): tools/kernel.ld

连接kernel

生成bootblock：

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

编译/boot目录下的文件，

-nostdinc 不搜索默认路径头文件

-0s 优化生成代码

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

连接bootblock

-N 设置为读写

-e 设置入口

-Ttext 0x7C00 设置起始地址

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

OBJDUMP反汇编，保存至asm文件

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

OBJCOPY，输出至out文件

-S 移除符号和重定位信息

-O binary 指定输出格式为binary

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

使用sign程序生成bootblock

$(call add\_files\_host,tools/sign.c,sign,sign

生成sign.o，用于生成bootblock依赖

$(call create\_target\_host,sign,sign)

生成sign，用于生成bootblock依赖

生成ucore.img：

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

从/dev/zero中获取10000个block，其为空字符

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

从bootblock中获取数据，输出至ucore.img

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

从kernel中获取数据，跳过第一个block，输出至ucore.img

至此，ucore.img生成完毕。

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？

大小为512字节，且以0x55AA结束。

练习2

1 从CPU加电后执行的第一条指令开始，单步跟踪BIOS的执行：

修改lab1/tools/gbdinit：

set architecture i8086

target remote :1234

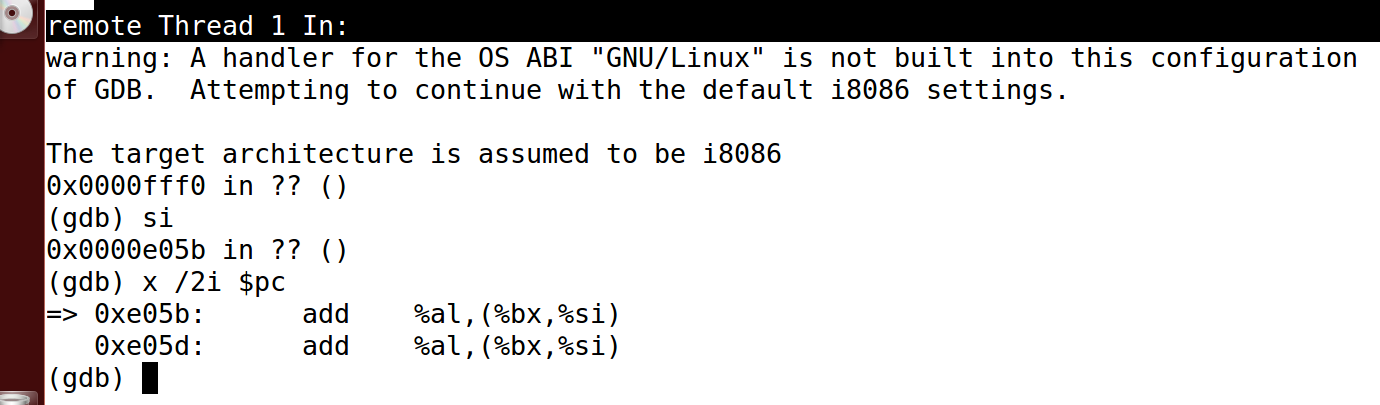
在lab1执行

make debug

在gdb界面执行

si //单步跟踪

x /2i $pc //显示当前eip处的汇编



2 在初始化位置0x7c00设置实地址断点，测试断点正常：

修改gbdinit：

set architecture i8086

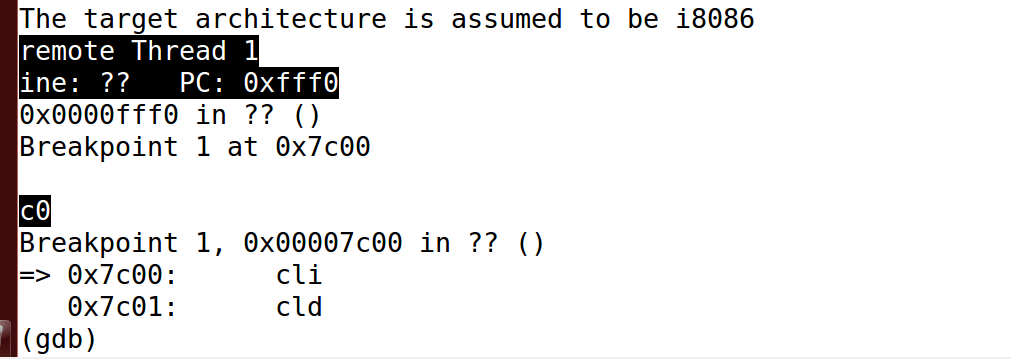
target remote :1234

b \*0x7c00

c

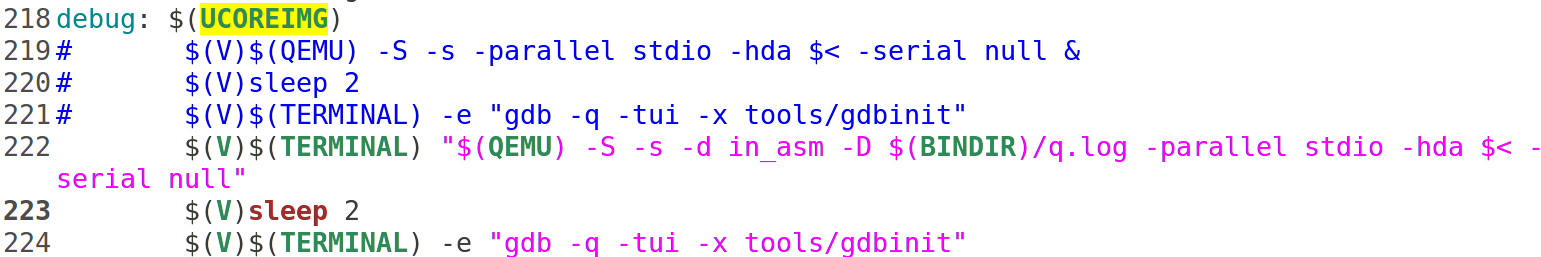
x/2i $pc

断点正常：

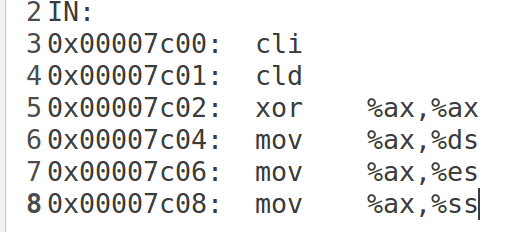


3 从0x7c00开始跟踪代码运行，将单步跟踪反汇编得到的代码与bootasm.s和bootblock.asm进行比较

改写makefile：



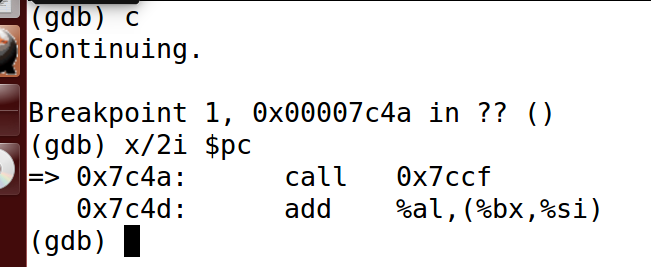
make debug得到q.log：



对比bootasm.s，bootblock.asm可以发现，bootasm.s与bootblock.s一致，与q.log中代码也一致。

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。

设置断点0x7c4a，在gdb中调试：



练习3 分析bootloader进入保护模式的过程

查看bootasm.S，逐段分析：

在0x7c00加载bootloader，关闭中断，初始化寄存器

# start address should be 0:7c00, in real mode, the beginning address of the running bootloader

.globl start

start:

.code16

cli

cld

# Set up the important data segment registers (DS, ES, SS).

xorw %ax, %ax # Segment number zero

movw %ax, %ds # -> Data Segment

movw %ax, %es # -> Extra Segment

movw %ax, %ss # -> Stack Segment

使能A20：等待8042输入缓冲区为空，往0x64写入0xd1，修改8042的P2端口，等待输入缓冲区为空，往0x60写入0xDF，将A20置位为1，开启A20，利用32位寻址能力。

# Enable A20:

# For backwards compatibility with the earliest PCs, physical

# address line 20 is tied low, so that addresses higher than

# 1MB wrap around to zero by default. This code undoes this.

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

outb %al, $0x60

使用GDT从实模式切换到保护模式：载入GDT表，加载cr0到eax，将eax的第0位置1，将cr0的第0位置1，长跳转更新cs基址。至此切换至保护模式。

# Switch from real to protected mode, using a bootstrap GDT

# and segment translation that makes virtual addresses

# identical to physical addresses, so that the

# effective memory map does not change during the switch.

lgdt gdtdesc

movl %cr0, %eax

orl $CR0\_PE\_ON, %eax

movl %eax, %cr0

# Jump to next instruction, but in 32-bit code segment.

# Switches processor into 32-bit mode.

ljmp $PROT\_MODE\_CSEG, $protcseg

练习4 分析bootloader加载ELF格式OS的过程

1 读取扇区(readsect)

#等待IO不忙（等待磁盘准备好）

#向IO 0x1f2/0x1f3/0x1f4/0x1f5/0x1f6/0x1f7中写入参数（发出读取扇区命令）

#等待IO不忙（等待磁盘准备好）

#使用insl命令，读取硬盘数据至内存

2 加载ELF格式的OS

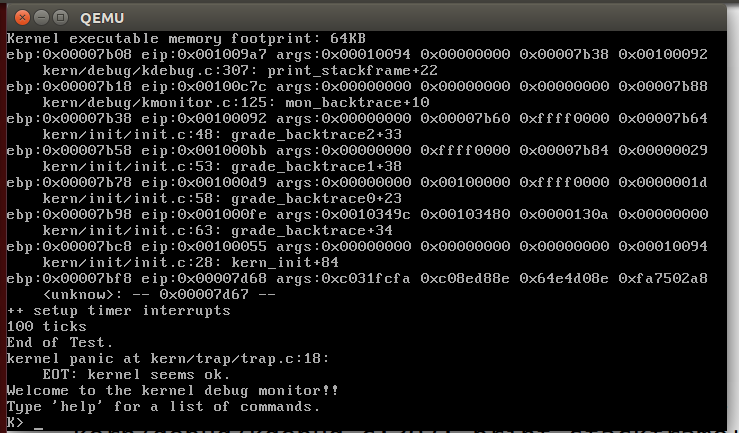
#读取8个扇区数据至0x10000，转化成elfhdr

#校验e\_magic

#根据offset将程序段数据读取至内存

练习5 实现print\_stackframe

代码另附，翻译code中的注释实现，make qemu得到：



简要过程：

#读取ebp,eip

#打印相关信息

#查找调用者，更新ebp,eip

#ebp非0且小于STACKFRAME\_DEPTH则循环

最后一行输出为第一个使用堆栈的函数(bootmain)。

练习6 完善中断初始化和处理

1. 一个表项8字节，0-15位为段偏移的0-15位，16-31位为段选择子，48-63位为段偏移的16-31位。这些位代表了中断处理代码的入口。
2. 完成代码后，运行程序如下图：

