1. **实验环境配置**

Qemu硬件模拟器安装

安装前可以进行检测本虚拟机是否已有qemu

检测：命令行 q (按TAB键)

若已存在qemu则跳过安装步骤

安装 命令行：sudo apt-get install qemu-system（安装文件较大 要等待）

完成后再进行检测是否已有

1. **lab1 练习1：理解通过make生成执行文件的过程。**

前言：由于make生成文件的过程只会显示输出，若想详细了解其执行了什么命令可在命令行中执行：make“V=”

①操作系统镜像文件ucore.img是如何一步一步生成的

1、在Makefile文件中生成ucore.img的代码为：

$(UCOREIMG): $(kernel) $(bootblock)====》由此需要kernel与booklock

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

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

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

1. kernel生成代码：

$(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)

1. Booklock生成代码

$(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)

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

$(call outfile,bootblock)

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

1. **Lab1 练习2：使用qemu动态调试，理解计算机加电后 BIOS执行过程**

1 修改 目录下labcodes/lab1/tools/gdbinit的内容为:

set architecture i8086

target remote :1234（添加）

------------------------------------------------------------------------------------------------

2 cd到 lab1目录下，执行

命令行执行：make debug

------------------------------------------------------------------------------------------------

3 在看到gdb的调试界面(gdb)后，在gdb调试界面下执行如下命令

命令行：si

即可单步跟踪BIOS了。

------------------------------------------------------------------------------------------------

4 在gdb界面下，可通过如下命令来看BIOS的代码

gdb调试界面: x /2i $pc //显示当前eip处的汇编指令

------------------------------------------------------------------------------------------------

Eg:①在gdb下输入next进行动态调试一条next一动一次动态调试（若输入x /2i $pc 即可显示当前汇编指令）

1. **lab1练习3：分析bootloader进入保护模式的过程**

①初始化环境

②将A20置为高电位

保护模式下,A20 地址线控制是要打开的，所以需要通过将键盘控制器上的A20线置于高电位，使得全部32条地址线可用。

seta20.2: // 等待8042键盘控制器不忙

inb $0x64, %al // 读取状态寄存器,等待8042键盘控制器闲置

testb $0x2, %al // 测试al的第2位是否为0

jnz seta20.2 // if al[2]=为0，就不执行该指令;否则就循环检查

movb $0xdf, %al // 将0xdf写入到al中

outb %al, $0x60 // 通过0x60写入数据11011111 即将A20置1

③加载GDT的基地址

④进入bootmain函数

此时转到保护模式完成：

1)将cr0寄存器PE位置1便开启了保护模式

ljmp $PROT\_MODE\_CSEG, $protcseg

.code32

protcseg:

2) 进入bootmain函数

call bootmain

1. **Lab1练习4：分析bootloader加载ELF格式的os的过程**

（一）bootloader是如何读取硬盘扇区的

首先就是从硬盘读取第一页(读到内存的位置，大小，ELF文件偏移)

引用到了函数readseg(uintptr\_t va, uint32\_t count, uint32\_t offset)，而readseg函数则循环调用了真正读取硬盘扇区的函数readsect来每次读出一个扇区:

1. **Lab1练习5：完成kdebug.c中函数print\_stackframe的实现**

可以通过函数>print\_stackframe来跟踪函数调用堆栈中记录的返回地址。

ss:ebp指向的堆栈位置储存着caller的ebp，以此为线索可以得到所有使用堆栈的函数ebp。

ss:ebp+4指向caller调用时的eip，ss:ebp+8等是（可能的）参数。

输出中，堆栈最深一层为

```

ebp:0x00007bf8 eip:0x00007d68 \

args:0x00000000 0x00000000 0x00000000 0x00007c4f

<unknow>: -- 0x00007d67 --

```

其对应的是第一个使用堆栈的函数，bootmain.c中的bootmain。

bootloader设置的堆栈从0x7c00开始，使用"call bootmain"转入bootmain函数。

call指令压栈，所以bootmain中ebp为0x7bf8。

1. **Lab1练习6：完成中断初始化和处理**
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。

在 idt\_init 函数中,依次对所有中断入口进行初始化。使用 mmu.h 中的 SETGATE 宏,填充 idt 数组内容。注意除了系统调用中断(T\_SYSCALL)以外,其它中断均使用中断门描述符,权限为内核态权限;而系统调用中断使用 异常,权限为陷阱门描述符。每个 中断的入口由tools/vectors.c 生成,使用 trap.c 中声明的 vectors 数组即可。

填充的代码为

-----------------------------------------------------------------------------------------------------------------

void idt\_init(void) {

extern uintptr\_t \_\_vectors[];//声明vertors[],其中存放着中断服务程序的入口地址

int i;

for(i=0;i<256;i++) {

SETGATE(idt[i],0,GD\_KTEXT,\_\_vectors[i],DPL\_KERNEL);

}

SETGATE(idt[T\_SWITCH\_TOK],0,GD\_KTEXT,\_\_vectors[T\_SWITCH\_TOK],DPL\_USER); //填充中断描述符表IDT

lidt(&idt\_pd); //使用lidt指令加载中断描述符表

}

-----------------------------------------------------------------------------------------------------------------

（二）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”。

代码：

-----------------------------------------------------------------------------------------------------------------

Case IRQ\_OFFSET + IRQ\_TIMER:

Ticks ++;

If(ticks % TICK\_NUM == 0)

{

Printf\_ticks();

}

Break;

-----------------------------------------------------------------------------------------------------------------