|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | 张彬 | **学号** | 38 |
| **实验题目** | Lab1实验 | | |
| **实验内容** | 练习一：make 执行效果：  只有结果信息    make V= 执行效果：  所有的编译执行信息及结果信息都显示出来了    GCC的执行过程：  先将C语言的源代码编译成以.o为后缀的目标文件，然后通过ld将目标文件转换成执行程序；以同样方法再将bootblock编译链接后生成可执行文件bootblock.out；将填充ucore.img至512字节；将kernel和 bootblock放到ucore.img中。  sign.c完成了特征的标记。  练习二：1.从 CPU 加电后执行的第一条指令开始,单步跟踪 BIOS 的执行。  单步跟踪方法如下：  （1）修改 lab1/tools/gdbinit,内容为:    （2）在 lab1目录下，执行make debug进入调试界面  （3）在gdb调试界面下执行命令：si 即可单步跟踪BIOS了    （4）在gdb界面下，可通过命令：x /2i $pc 来看BIOS的代码  2.在初始化位置0x7c00 设置实地址断点,测试断点正常。  3.在调用qemu 时增加-d in\_asm -D q.log 参数，便可以将运行的汇编指令保存在q.log 中。 将执行的汇编代码与bootasm.S 和 bootblock.asm 进行比较，看看二者是否一致。  在tools/gdbinit结尾加上  b \*0x7c00  c  x /10i $pc  便可以在q.log中读到"call bootmain"前执行的命令  可以发现其与bootasm.S和bootblock.asm中的代码相同。  练习三：  首先清理环境：包括将flag置0和将段寄存器置0    开启A20：通过将键盘控制器上的A20线置于高电位，全部32条地址线可用， 可以访问4G的内存空间。    初始化GDT表：一个简单的GDT表和其描述符已经静态储存在引导区中，载入即可    进入保护模式：通过将cr0寄存器PE位置1便开启了保护模式    通过长跳转更新cs的基地址    设置段寄存器，并建立堆栈    转到保护模式完成，进入boot主方法    练习四：  首先看readsect函数， readsect从设备的第secno扇区读取数据到dst位置    readseg简单包装了readsect，可以从设备读取任意长度的内容。    在bootmain函数中，    练习五：  完成kern/debug/kdebug.c中的print\_stackframe函数    打印结果    表示从bootmain.c中的bootmain开始使用"call bootmain"转入bootmain函数，bootloader设置的堆栈从0x7c00开始。  练习六：  （1）中断向量表一个表项占用8字节，其中2-3字节是段选择子，0-1字节和6-7字节拼成位移， 两者联合便是中断处理程序的入口地址。    （2）编程完善trap.c中的idt\_init函数    （3）编程完善trap.c中的中断处理函数trap，向屏幕上打印一行文字”100 ticks”  trap函数    打印结果函数    打印结果    扩展练习：  设计函数如下  lab1\_switch\_test()  lab1\_switch\_to\_user()  lab1\_switch\_to\_kernel()  都在kern/init.c中，函数执行顺序为lab1\_switch\_test -> lab1\_switch\_to\_user -> lab1\_switch\_to\_kernel  lab1\_switch\_test函数：    lab1\_switch\_to\_user函数：    lab1\_switch\_to\_kernel函数：    添加用户调用 | | |
| **总结** | （1）熟悉ucore项目组织形式和调试方法  （2）理解ucore启动流程实现方式  （3）理解ucore中断机制的实现流程以及内核态与用户态的切换流程  （4）知道了make的用法和作用  （5）了解了Makefile文件的组成  （6）明白了os的启动过程  （7）清楚了os中断过程及产生的作用 | | |
| **日期** | 2020.06.01 | **成绩** |  |