|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | 夏国圣 | **学号** | 33 |
| **实验题目** | Lab1 | | |
| **实验内容** | **练习一：了解ucore的“项目组成”理解通过make生成执行文件的过程**  在makefile中主要有很多指令组成，里面我们主要了解一下ucore.img文件时如何形成的。    从上面可以看出，UCOREIMG是由kernel和boot组成，这里的dd不是GCC的语言指令。其意思是当后面的条件满足时，前面的文件填充到$@里面去。  终端先cd命令进入到lab1\_result目录里面，输入指令make clean，将之前编译的内容删除掉。  在输入make指令，得到的编译结果是    在比较一下指令make V=输出的更详细的过程结果：    问题：简要说明gcc编译器是如何一步步生成ucore可执行文件   1. 首先把C的源代码编译为目标文件（.o文件）      1. ld命令将这些目标文件转变成可执行文件，比如下面的kernel      1. dd命令把bootloader放到ucore.img.count的虚拟硬盘中     Sign.c的作用时检查作用，从sign.c的代码中能看出，引导扇区的大小为512字节，最后两个字节为标志性结束字节0x55，0xAA，做完这样的检查才能认为是符合规范的磁盘主引导扇区。  **练习二：使用qemu执行并调试lab1中的软件**  本练习要在labcodes里面的lab1中完成，需要将answers里面makefiles的lab1-mon代码填在lab1中的makefiles里面。此步骤不在截图。  在终端用cd命令进入labcodes/lab1目录  输入 make lab1-mon      按照实验步骤，进行在初始化位置0x7c00设置实地址断点,测试断点正常    对比发现，上述反汇编指令与bootasm.S和bootblock.asm相同。  **练习3：分析bootloader进入保护模式的过程**  lab1/boot/bootasm.S源码就完成了bootloader从实模式切换到保护模式。  此段注释说明了要完成的目的：启动保护模式，转入C函数。    bootasm.S文件的作用。计算机加电后，由BIOS将bootasm.S生成的可执行代码从硬盘的第一个扇区复制到内存中的物理地址0x7c00处,并开始执行。  此时系统处于实模式。可用内存不多于1M。    箭头往上都是准备工作，下面开始激活A20地址位。先翻译注释：由于需要兼容早期pc，物理地址的第20位绑定为0，所以高于1MB的地址又回到了0x00000.  好了，激活A20后，就可以访问所有4G内存了，就可以使用保护模式了。    这个代码之后打开保护模式的标志位为1，实现了实模式切换到保护模式。  **练习4：根据代码，通过阅读bootmain.c，了解bootloader如何加载ELF文件。**  下面是bootmain.c源代码    根据上述bootmain函数分析，首先是由readseg函数读取硬盘扇区，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    读取完磁盘之后，开始加载ELF格式的文件。    **练习五：要求完成函数kern/debug/kdebug.c::print\_stackframe的实现.完成kdebug.c中函数print\_stackframe的实现。**  代码补全如图所示：    在终端进入lab1输入make qemu得到如图所示结果    ebp和eip的调用关系是首先将ebp寄存器入栈，然后将栈顶指针eip赋值给ebp。“mov ebp eip”这条指令表面上看是用eip覆盖ebp原来的值，其实不然。因为给ebp赋值之前，原ebp值已经被压栈（位于栈顶），而新的ebp又恰恰指向栈顶。此时ebp寄存器就已经处于一个非常重要的地位，该寄存器中存储着栈中的一个地址（原ebp入栈后的栈顶），从该地址为基准，向上（栈底方向）能获取返回地址、参数值，向下（栈顶方向）能获取函数局部变量值，而该地址处又存储着上一层函数调用时的ebp值。  **练习六：（1）从代码找出中断描述符表（也可简称为保护模式下的中断向量表）的定义，并简要说明中断描述符表中一个表项占多少字节？各分别表示什么？其中哪几位代表中断处理代码的入口？**    一个表项占八个字节。入口地址为：gd\_off\_31\_16 << 16 + gd\_off\_15\_0。  其中2-3字节是段选择子，0-1字节和6-7字节拼成位移 。  **联系六：（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。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。**  代码如图所示：    重点就是两步  第一步，声明\_\_vertors[],其中存放着中断服务程序的入口地址。这个数组生成于vertor.S中。  第二步，填充中断描述符表IDT。  第三步，加载中断描述符表。  这里的SETGATE在mmu.h中有定义，  **练习六：（3）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”。**  这里根据指导书查看函数trap\_dispatch，发现print\_ticks()子程序已经被实现了，所以我们直接进行判断输出即可，如下    输入make qemu得到如图所示 | | |
| **总结** | 1. 了解ucore的“项目组成” 2. 根据make编译信息，gcc编译器是如何一步步生成ucore可执行文件的 3. 学会使用qemu与gdb协作进行调试ucore代码 4. C语言编程能力提高 5. 了解理解ucore中断机制的实现流程 6. 再次熟悉掌握了Linux系统的指令，为下次的实验打下基础 | | |
| **日期** | 2020.06.01 | **成绩** |  |