|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | 史新宽 | **学号** | 29 |
| **实验题目** | Lab1练习 | | |
| **实验内容** | **练习1、**  **Makefile的主要组成：**  最开始是对各种常量的初始化，接下来部分则用于推断环境中调用所安装的gcc应当使用的命令：利用bash命令来推断qemu的命令，定义各种编译命令以及编译选项，生成ucore.img。  **make与make “V=”所生成信息的不同**  make    make “V=”    make V=会输出所有的编译过程。  **gcc编译器是如何一步步生成ucore可执行文件的**  先生成ucore.img需要先生成kernel和bootblock再创建一个大小为10000字节的块，然后再将bootblock，kernel拷贝过去。  生成kernel    编译kernel    链接生成kernel    生成bootblock    编译bootblock    链接生成bootblock    生成ucore.ing    执行make    **sign.c的作用是什么**  利用tools/sing.c生成sign.o利用sign.o生成sign，sign对bootblock进行修饰，在512个字节的最后两个字节写入了0x55AA，作为引导区的标记。  **练习2、**  **从CPU加电后执行的第一条指令开始，单步跟踪BIOS的执行**  在tools/gdbinit里进行相应设置    执行make debug    **在初始化位置0x7c00设置实地址断点**  需要的tools/gdbinit文件如下    补全代码    执行make lab1-mon      **bootasm.S和 bootblock.asm进行比较**  bootasm.S    bootblock.asm    bootblock.asm中没有指定位宽w（word）之外，其余内容完全一致  **自己设置断点并进行测试**  修改tools/gdbinit    执行make debug    **练习3、**  **分析bootloader进入保护模式的过程**  1. bootloader入口为start，bootloader会被BIOS加载到内存的0x7c00处，此时cs=0, eip=0x7c00，在刚进入bootloader的时候，最先执行的操作分别为关闭中断、清除EFLAGS的DF位以及将ax, ds, es, ss寄存器初始化为0；    2. 从0x64内存地址中（映射到8042的status register）中读取8042的状态，直到读取到的该字节第二位（input buffer是否有数据）为0，此时input buffer中无数据；    3. 往0x64写入0xd1命令，表示修改8042的P2 port    4. 继续等待input buffer为空    5. 往0x60端口写入0xDF，表示将P2 port的第二个位（A20）选通置为1    6. 设置GDT（全局描述符表）    7. 将cr0寄存器的PE位置1，即可从实模式切换到保护模式    8. 使用一个长跳转指令，将cs修改为32位段寄存器，以及跳转到protcseg这一32位代码入口处，此时CPU进入32位模式    **练习4、**  **分析bootloader加载ELF格式的OS的过程**  1. 该函数的作用是连续不断地从0x1F7地址读取磁盘的状态，直到磁盘不忙为止；    2. 等待磁盘直到其不忙；  往0x1F2到0X1F6中设置读取扇区需要的参数，包括读取扇区的数量以及LBA参数；  往0x1F7端口发送读命令0X20；  等待磁盘完成读取操作；  从数据端口0X1F0读取出数据到指定内存中；    3. 函数readseg，将readsect进行进一步封装，提供能够从磁盘第二个扇区起（kernel起始位置）offset个位置处，读取count个字节到指定内存中    4. 从磁盘的第一个扇区（第零个扇区为bootloader）中读取OS kenerl最开始的4kB代码，然后判断其最开始四个字节是否等于指定的ELF\_MAGIC，用于判断该ELF header是否合法；    5. 从ELF头文件中获取program header表的位置，以及该表的入口数目，然后遍历该表的每一项，并且从每一个program header中获取到段应该被加载到内存中的位置（Load Address，虚拟地址），以及段的大小，然后调用readseg函数将每一个段加载到内存中，至此完成了将OS加载到内存中的操作；    6. 从ELF header中查询到OS kernel的入口地址，然后使用函数调用的方式跳转到该地址上去    **练习5、**  **实现函数调用堆栈跟踪函数**  补全代码：    lab1中执行 “make qemu”后，得到如下的输出    最后一行打印出来的是bootmain这个函数对应的栈帧信息，其中ebp表示该栈帧的base pointer，eip表示在该函数内调用栈上的下一个函数指令的返回地址，而后面的args则表示传递给bootmain函数的参数。  **练习6、**  mmu.h中中断描述符定义    中断描述符一个表项占8字节，其中0~15位和48~63位分别是offset的低16位和高16位，16~31位是段选择子，段选择子获得段基址加上偏移量就是入口  **练习6、**  完善代码    extern uintptr\_t \_\_vectors[];//声明\_\_vertors[] You can use "extern uintptr\_t \_\_vectors[];" to define this extern variable which will be used later.  int i;  for(i=0;i<256;i++) {  SETGATE(idt[i],0,GD\_KTEXT,\_\_vectors[i],DPL\_KERNEL);//对整个idt数组进行初始化  }  SETGATE(idt[T\_SWITCH\_TOK],0,GD\_KTEXT,\_\_vectors[T\_SWITCH\_TOK],DPL\_USER);//在这里先把所有的中断都初始化为内核级的中断  lidt(&idt\_pd);//使用lidt指令加载中断描述符表 just google it! and check the libs/x86.h to know more.利用google找到了相关函数  }  mmu.h中的SETGATE宏    **练习6、**  代码    执行make qemu | | |
| **总结** | * 1.了解到了ucore的“项目组成。 * 2. 弄清了操作系统镜像文件ucore.img是如何一步一步生成的。 * 3.理解了Makefile的用处，理解了它的便利性掌握了其使用方法。   4. 初步掌握了如何用qemu与gdb对ucore代码进行调试。  5.了解了编译运行bootloader的过程调试，bootloader的方法，PC启动bootloader的过程，ELF执行文件的格式和加载。  6. 了解了实模式到保护模式的转换。 | | |
| **日期** | 2020.05.31 | **成绩** |  |