|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 李旭 | 学号 | 16 |
| 实验题目 | 1、掌握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。 | | |
| 实验内容 | 1、了解ucore的“项目组成”。  ①bootloader部分（bootasm.S、bootmain.c、asm.h）  ②ucore操作系统部分（系统初始化部分、内存管理部分、外设驱动部分、中断处理部分、内核调试部分、公共库部分、工具部分）  2、学会使用make编译文件，了解Makefile中的主要组成。查看使用make与make “V=”所生成信息的不同。 根据make编译信息，简要说明gcc编译器是如何一步步生成ucore可执行  文件的。sign.c的作用是什么。  Make：  238a733982494523573414d2b8d08fc  Make V=  3d47d08443a2259c17ad5e9ee1f966b  gcc编译器生成ucore可执行文件：  需要生成ucore.img首先需要生成bootblock，而生成bootblock需要先生成 bootmain.o和bootasm.o还有sign，这三个文件又分别由bootmain.c、bootasm.S、sigh.c来生成。  63d7944e1c68ca9c5c2cdf36d69fa79  sign.c是一个工具，作用是做一个引导扇区的特征标志   1. 学会使用qemu与gdb协作进行调试ucore代码；在初始化位置0x7c00设置实地址断点,测试断点正常。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。自己找一个bootloader或内核中的代码位置，设置断点并进行测试。（截图并简要说明实验过程）   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tools/gdbinit，在lab1下执行make debug  1  使用si命令可对BIOS进行单步跟踪  2  ②在初始化位置0x7c00设置实地址断点,测试断点正常。 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。  修改tools/gdbinit，在lab1下执行make debug  34  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5  4了解如何在bootloader中完成由实模式到保护模式的转换的。（要有过程或对应代码的截图）  ①首先清理环境：包括将flag置0和将段寄存器置0  练习三1  ②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。  练习三2  ③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  练习三3  通过长跳转更新cs的基地址；设置段寄存器，并建立堆栈  练习三4  转到保护模式完成。  5.根据代码，通过阅读bootmain.c，了解bootloader如何加载ELF文件。  执行完bootasm.S后，系统进入保护模式，进行bootmain.c开始加载OS：  ①定义ELF头指针，指向0x10000  ②读取8个扇区大小的ELF头到内存地址0x10000  ③校验ELF header中的魔数，判断是否为0x464C457FU  ④读取ELF header中的程序段到内存中  ⑤跳转到操作系统入口  6.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。  ①kern/debug/kdebug.c中的print\_stackframe函数  练习五1  ②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。   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？   中断向量表：是一个8字节的描述符数组，  2-3字节是段选择子，0-1字节和6-7字节是offset的高低位地址，  入口地址=段选择子+段内偏移量  练习六1  8.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。    9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”。  每TICK\_NUM=100，产生中断。  练习六3  打印函数如下：    打印结果：    10.参考答案labcodes\_answer/lab1,并在labcodes/lab1中完成challenge1内容，并简要说明实现的过程。  在kern/init.c中，函数执行lab1\_switch\_test -> lab1\_switch\_to\_user -> lab1\_switch\_to\_kernel | | |
| 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熟悉make及Makefile的用法和作用  ⑤理解os的启动过程  ⑥理解os中断过程及产生的作用 | | |
| 日期 | 2020.06.02 | 成绩 |  |