|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | 葛桦玮 | **学号** | 08 |
| **实验题目** | 第二次实验  Lab1 | | |
| **实验内容** | **1.（练习1）了解ucore的“项目组成”。**  bootloader部分   * boot/bootasm.S ：定义并实现了bootloader最先执行的函数start，此函数进行了一定的初始化，完成了从实模式到保护模式的转换，并调用bootmain.c中的bootmain函数。 * boot/bootmain.c：定义并实现了bootmain函数实现了通过屏幕、串口和并口显示字符串。bootmain函数加载ucore操作系统到内存，然后跳转到ucore的入口处执行。 * boot/asm.h：是bootasm.S汇编文件所需要的头文件，主要是一些与X86保护模式的段访问方式相关的宏定义。   ucore操作系统部分  系统初始化部分：   * kern/init/init.c：ucore操作系统的初始化启动代码   内存管理部分：   * kern/mm/memlayout.h：ucore操作系统有关段管理（段描述符编号、段号等）的一些宏定义 * kern/mm/mmu.h：ucore操作系统有关X86 MMU等硬件相关的定义，包括EFLAGS寄存器中各位的含义，应用/系统段类型，中断门描述符定义，段描述符定义，任务状态段定义，NULL段声明的宏SEG\_NULL, 特定段声明的宏SEG，设置中 断门描述符的宏SETGATE（在练习6中会用到） * kern/mm/pmm.[ch]：设定了ucore操作系统在段机制中要用到的全局变量：任务状态段ts，全局描述符表 gdt[]，加载全局描述符表寄存器的函数lgdt，临时的内核栈stack0；以及对全局描述符表和任务状态段的初始化函数gdt\_init   外设驱动部分：   * kern/driver/intr.[ch]：实现了通过设置CPU的eflags来屏蔽和使能中断的函数； * kern/driver/picirq.[ch]：实现了对中断控制器8259A的初始化和使能操作； * kern/driver/clock.[ch]：实现了对时钟控制器8253的初始化操作；- kern/driver/console.[ch]：实现了对串口和键盘的中断方式的处理操作；   中断处理部分：   * kern/trap/vectors.S：包括256个中断服务例程的入口地址和第一步初步处理实现。注意，此文件是由tools/vector.c在编译ucore期间动态生成的； * kern/trap/trapentry.S：紧接着第一步初步处理后，进一步完成第二步初步处理；并且有恢复中断上下文的处理，即中断处理完毕后的返回准备工作； * kern/trap/trap.[ch]：紧接着第二步初步处理后，继续完成具体的各种中断处理操作；   内核调试部分：   * kern/debug/kdebug.[ch]：提供源码和二进制对应关系的查询功能，用于显示调用栈关系。其中补全print\_stackframe函数是需要完成的练习。其他实现部分不必深究。 * kern/debug/kmonitor.[ch]：实现提供动态分析命令的kernel monitor，便于在ucore出现bug或问题后，能够进入kernel monitor中，查看当前调用关系。实现部分不必深究。 * kern/debug/panic.c | assert.h：提供了panic函数和assert宏，便于在发现错误后，调用kernel monitor。大家可在编程实验中充分利用assert宏和panic函数，提高查找错误的效率。   公共库部分   * libs/defs.h：包含一些无符号整型的缩写定义。 * Libs/x86.h：一些用GNU C嵌入式汇编实现的C函数（由于使用了inline关键字，所以可以理解为宏）。   工具部分   * Makefile和function.mk：指导make完成整个软件项目的编译，清除等工作。 * sign.c：一个C语言小程序，是辅助工具，用于生成一个符合规范的硬盘主引导扇区。 * tools/vector.c：生成vectors.S，此文件包含了中断向量处理的统一实现。   2**.（练习1）会使用make编译文件，了解Makefile中的主要组成，实验报告中要有简单介绍。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    make V=      如何一步步生成ucore可执行文件的  （1）编译libs和kern目录下所有的.c和.S文件，生成.o文件，并链接得到bin/kernel文件  （2）编译boot目录下所有的.c和.S文件，生成.o文件，并链接得到bin/bootblock.out文件  （3）编译tools/sign.c文件，得到bin/sign文件  利用bin/sign工具将bin/bootblock.out文件转化为512字节的bin/bootblock文件，并将bin/bootblock的最后两个字节设置为0x55AA  （4）为bin/ucore.img分配5000MB的内存空间，并将bin/bootblock复制到bin/ucore.img的第一个block，紧接着将bin/kernel复制到bin/ucore.img第二个block开始的位置  **3.（练习2）学会使用qemu与gdb协作进行调试ucore代码；从CPU加电后执行的第一条指令开始，单步跟踪BIOS的执行。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）**  vi makefile    vi tools/labinit    反汇编x 10i $pc    bootasm.S和 bootblock.asm进行比较  (1)反汇编的代码中的指令不带指示长度的后缀，而bootasm.S的指令则有。比如，反汇编 的代码是xor %eax, %eax，而bootasm.S的代码为xorw %ax, %ax  (2)反汇编的代码中的通用寄存器是32位（带有e前缀），而bootasm.S的代码中的通用寄存器是16位（不带e前缀）。  **4.（练习3）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）** bootloader从实模式切换到保护模式，需要做以下事情：  （1）开启A20门    （2）在内存中建立GDT表并初始化    （3）设置cr0寄存器的PE位为1，表示从实模式切换到保护模式  将cr0寄存器的PE位（cr0寄存器的最低位）设置为1，便使能和进入保护模式了    **5.（练习4）根据代码，通过阅读bootmain.c，了解bootloader如何加载ELF文件。（需要截图，并简要说明）**  bootloader要加载的是bin/kernel文件，这是一个ELF文件。其开头是ELF header，ELF Header里面含有phoff字段，用于记录program header表在文件中的偏移，由该字段可以找到程序头表的起始地址。程序头表是一个结构体数组，其元素数目记录在ELF Header的phnum字段中  根据ELF Header和Program Header表的信息，我们便可以将ELF文件中的所有Segment逐个加载到内存中    (1)首先从硬盘中将bin/kernel文件的第一页内容加载到内存地址为0x10000的位置，目的是读取kernel文件的ELF Header信息。  (2)校验ELF Header的e\_magic字段，以确保这是一个ELF文件  (3)读取ELF Header的e\_phoff字段，得到Program Header表的起始地址；读取ELF Header的e\_phnum字段，得到Program Header表的元素数目。  (4)遍历Program Header表中的每个元素，得到每个Segment在文件中的偏移、要加载到内存中的位置（虚拟地址）及Segment的长度等信息，并通过磁盘I/O进行加载  (5)加载完毕，通过ELF Header的e\_entry得到内核的入口地址，并跳转到该地址开始执行内核代码  **6.（练习5）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。（需要make qemu 后的结果等截图，简要说明打印信息，说明调用关系。）**  **7.（练习6）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？（截图并简要说明）**  中断描述符表一个表项占8个字节，其结构如下：  bit 63..48: offset 31..16  bit 47..32: 属性信息，包括DPL、P flag等  bit 31..16: Segment selector  bit 15..0: offset 15..0  其中最开始2个字节和最末尾2个字节定义了offset，第16-31位定义了处理代码入口地址的段选择子，使用其在GDT中查找到相应段的base address，加上offset就是中断处理代码的入口。  **8.（练习6）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。（截图并简要说明）**  **9.（练习6）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”。 （截图并简要说明）**  **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（设计到哪些函数，分别在哪个文件中，执行中断的过程中，先后使用了哪些函数。）。** | | |
| **总结** | 基于分段的内存管理机制  CPU的中断机制  x86 CPU的保护模式  计算机系统的启动过程  ELF文件格式  读取LBA模式硬盘的方法  编译ucore OS和bootloader的过程  GDB的使用方法  物理内存的管理  外存的访问  OS的启动过程  OS中对中断机制的支持  OS使用保护机制对关键代码和数据的保护 | | |
| **日期** | 2020.05.26 | **成绩** |  |