|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **《操作系统设计与实践》实验报告** | | | | | | | |
| 课程名称 | 操作系统设计与实践 | | | 成 绩 |  | 教师签名 |  |
| 实验名称 | 实验环境搭建 | | | 实验序号 | 1 | 实验日期 |  |
| 姓 名 | 张子航 | 学 号 | 2021302181026 | | | | 组长 |
| 姓 名 | 辜汝曦 | 学 号 | 2021302141194 | | | | 组员 |
| 姓 名 | 杨馨悦 | 学 号 | 2021302181212 | | | | 组员 |
| 姓 名 | 赵敏 | 学 号 | 2021302181215 | | | | 组员 |
| 1. **实验目的及实验内容**   （本次实验所涉及并要求掌握的知识；实验内容等） | | | | | | | |
| 【实验目的】  搭建实验的基本环境，熟悉开发与调试工具  【参考资料】   1. 《Orange’s 一个操作系统的实现》对应章节：第一、二章 2. 汇编语言快速入门   Introduction to Linux Intel Assembly Language :  <http://heather.cs.ucdavis.edu/~matloff/50/LinuxAssembly.html>  Linux 汇编语言开发指南:  <http://www.ibm.com/developerworks/cn/linux/l-assembly/>  【实验内容】   1. 认真阅读课本章节资料 2. 在实验机上安装virtualbox，并安装ubuntu 3. 安装ubuntu开发环境 4. 下载bochs源码，编译并安装bochs环境 5. 使用bochs自带工具bximage创建虚拟软驱 6. 阅读、编译boot.asm，并反汇编阅读 7. 修改bochsrc，运行调试你的第一个程序 8. 重点掌握课本14页表2.1 的bochs调试指令   【注意事项】   1. Virtualbox及其增强包下载：   https://www.virtualbox.org/wiki/Downloads   1. Ubuntu下载：   http://mirrors.163.com/ubuntu-releases/   1. 修改ubuntu源方法：   http://mirrors.163.com/.help/ubuntu.html   1. Bochs下载：   http://bochs.sourceforge.net/getcurrent.html   1. Bochs编译注意事项   可能需要安装build-essential、libx11-randr0-dev、libxrandr-dev，可在编译过程中发现   1. 修改bochsrc中部分内容：   修改romimage 和vgaromimage对应的文件位置，以你的实际安装位置为准  注释掉keyboard\_mapping一行 | | | | | | | |
| 1. **实验环境及实验步骤**   （列出本次实验所使用的软件、工具；简要概括实验步骤） | | | | | | | |
| **实验环境：**  虚拟机工具：VMWare Workstation 16  虚拟机版本：Ubuntu 14.04.6(内存4GB，硬盘40GB，双核处理器)  开发与调试工具：bochs 2.6.8  其他依赖项：g++，nasm，bison，xorg-dev，build-essential  **实验步骤：**  **首先，编译运行第一个bochs程序，实验步骤如下：**   * 安装环境，在VMWare上安装Ubuntu系统，在Ubuntu中配置实验环境； * 下载并编译bochs源码； * 使用bochs的bximage在项目文件夹中创建一个可启动的软盘； * 将boot.asm移动到项目文件夹中，使用nasm编译代码boot.asm，输出boot.bin； * 将boot.bin，即引导文件安装到软盘中； * 在项目文件夹中创建bochsrc文件； * 通过bochsrc文件启动bochs，调试运行软盘上的引导程序。   **然后，在以上步骤的基础上，完成实验练习中的题目。** | | | | | | | |
| 1. **实验过程分析**   （详细记录实验过程，通过截图展示得到的结果。特别是对于实验中发生的故障和问题，要进行故障分析，说明故障排除的过程及方法。）  **一、第一个bochs程序**   1. **安装实验环境**   首先在VMWare Station中安装Ubuntu14.04.6，然后在Ubuntu的terminal中运行如下命令：  **sudo apt-get install g++**  **sudo apt-get install nasm**  **sudo apt-get install bison**  **sudo apt-get install xorg-dev**  **sudo apt-get install build-essential**  这样，实验环境就配置好了。   1. **安装bochs**   首先，我们下载bochs的源代码压缩包；  **wget https://sourceforge.net/projects/bochs/files/bochs-2.6.8.tar.gz**  然后，我们解压该压缩包；  **sudo tar zxvf bochs-2.6.8.tar.gz**  这样，我们成功在ubuntu上获得了bochs的源码：    然后，我们进入bochs-2.6.8文件夹，在该文件夹下运行configure，配置参数；  **sudo ./configure --enable-debugger --enable-disasm**  其中，--enable-debugger打开调试功能，--enable-disaem打开反汇编功能。  最后，我们用make和make install来安装bochs。  **sudo make**  **sudo make install**  【报错】在这一步，程序报错了，是cpu的配置出现了问题。首先我们通过bochs -help cpu来得出bochs支持的类型；    然后，cpu配置相关的文件位于.bochsrc中，我们使用gedit打开它，找到cpu相关的配置文字，将原有的model类型注释掉，换成Supported CPU models中支持的model类型，其他不变。再运行一次make和make install，这一次程序顺利运行。    这样，bochs就成功安装在Ubuntu上了。在命令行中输入bochs，程序可以成功运行：     1. **创建软盘**   我们使用bximage命令来创建软盘。首先，程序会弹出选项来让用户选择接下来的操作，选择1，即create new floppy or hard disk image；然后选择image的类型，选择fd；之后选择大小、名称可以直接选择默认选项，我们名称使用了lab1.img。最后，程序生成了对应的软盘。     1. **编译boot.asm**   boot.asm的代码如下：  **org 07c00h ; 告诉编译器程序加载到7c00处**  **mov ax, cs**  **mov ds, ax**  **mov es, ax**  **call DispStr ; 调用显示字符串例程**  **jmp $ ; 无限循环**  **DispStr:**  **mov ax, BootMessage**  **mov bp, ax ; ES:BP = 串地址**  **mov cx, 16 ; CX = 串长度**  **mov ax, 01301h ; AH = 13, AL = 01h**  **mov bx, 000ch ; 页号为0(BH = 0) 黑底红字(BL = 0Ch,高亮)**  **mov dl, 0**  **int 10h ; 10h 号中断**  **ret**  **BootMessage: db "Hello, OS world!"**  **times 510-($-$$) db 0; 填充剩下的空间，使生成的二进制代码恰好为512字节**  **dw 0xaa55 ; 结束标志**  其中，真正用来执行的引导扇区都会装载到07c00h处，因为硬盘上会有一个主引导扇区，由它控制其他引导扇区。而主引导扇区位于0600h处，它会将真正的可引导扇区加载到07c00h处。因此，我们会编写org 07c00h这一段代码。  而dw 0xaa55表示引导工作已经完成，接下来的工作交给操作系统完成。0xaa55地址可以判断引导区是否合法，当不存在该代码时，操作系统不会对这一块软盘进行操作。  对boot.asm进行编译的代码为：  **nasm boot.asm -o boot.bin**  对于boot.bin，可以通过ndisasm命令反汇编：     1. **安装软盘**   通过运行以下命令，可以将boot.bin装载到lab1.img中：  **dd if=boot.bin of=lab1.img bs=512 count=1 conv=notrunc**     1. **编写bochsrc**   我们在项目文件夹中创建bochsrc。下面是本实验的bochsrc与资料中bochsrc的不同：   * 将floppya: 1\_44=a.img中的a.img改为lab1.img； * rominage和vgarominage中将/usr/share部分换成$BXSHARE环境变量。   【报错】之前将bochsrc放到bochs 2.8.6文件夹下，之后程序报错表示找不到lab1.img，将它移动到项目文件夹下就可以修复问题。   1. **运行bochs**   我们通过下面的命令启动软盘：  **bochs -f bochsrc**    选择6，开始simulation；  此时，可以在终端中调试（-b）或者直接运行（-c）。直接输入c，Bochs x86 Emulator输出了正确结果：    我们也可以输入b 0x7c00在mov ax, cs处打一个断点。之后，程序停在断点位置，可以通过info cpu来查看此时cpu的信息。结果如下：    这样，我们就完成了第一个bochs程序的调试和运行。  **二、实验练习**  **1. 删除0xAA55，观察程序效果，找出原因**   1. 修改bochsrc文件：floppya: 1\_44=test1.img, status=inserted 2. 复制boot.asm的代码到test1.asm，注释掉boot.asm的最后一行代码，修改如下      1. 在项目文件夹打开终端，使用nasm编译代码，nasm test1.asm -o test1.bin 2. 使用bximage创建软盘 3. 安装软盘：dd if=test1.bin of=test1.img bs=512 count=1 conv=notrunc 4. 启动软盘，bochs -f bochsrc进入调试，输出结果如下:      1. 发现输入c后无法运行，报错并直接退出，也没有显示“Hello, OS world!”，这是因为位于1扇区的512字节且最后字节为aa55的程序是引导程序，a.img为引导盘，aa55即为标志，删除了该标志后相当于没有了引导程序，所以无法运行直接退出。   **2. 修改程序中输出为，一个包含自己名字的字符串，调试程序**   1. 修改bochsrc文件：      1. 复制boot.asm的代码到myname.asm，将BootMessage改为自己的名字，相应的，同时将串长度改为10      1. 在项目文件夹打开终端，使用nasm编译代码，nasm myname.asm -o myname.bin 2. 使用bximage创建软盘 3. 安装软盘：dd if=myname.bin of=myname.img bs=512 count=1 conv=notrunc      1. 启动软盘：bochs -f bochsrc进入调试，输出自己名字的字符串，结果如下：      1. 把生成的可执行文件反汇编，看看输出的内容是怎样的，并在虚拟机启动过程，设置断点进行调试，在实验报告中截图   可执行文件反汇编的内容是：    进行断点调试过程：  由引导程序可知，系统启动时，需要将引导扇区的引导程序加载至系统内存的  0x7c00 位置然后继续执行，设置 0x7c00 为断点, 执行代码：  **b 0x7c00**  直接跳到该断点处执行程序：  **c**  在此处查看所有寄存器当前状态：  **info cpu**  执行 9条指令后，可用r查看当前寄存器状态：  **step 4**  **step 5**  **r**         1. 为什么要jmp $？如何改造程序，输出包含名字的字符串20次？   （1）jum $可使程序在此条指令处无限循环。这是一种常见的引导扇区结束的方式，因为引导扇区的主要任务是在引导过程中执行一些初始化和引导操作，之后会传递控制给操作系统或其他引导加载程序。因此，引导扇区通常需要保持运行，直到系统引导完毕，或者在遇到错误情况时可以终止引导过程。jmp $ 是实现这种无限循环的简单方法。  （2）按照如下代码改造程序，可输出包含名字的字符串20次：    效果为：     1. 回答：为什么要对段寄存器进行赋值   在这段代码中，为了确保DS和ES寄存器的值与代码段寄存器CS的值相同，使得访问数据和堆栈时使用的内存地址正确，以便在以后进行数据操作时定位到正确的位置所以将其值设置为CS寄存器的值。  假设将这两行的代码删除：    对这段代码进行编译，并启动bochs进行测试：    最终得出的结果如下：   1. 回答：如何在该程序中调用系统中断   在该段代码中，第14行代码即为屌用系统第10号中断，该中断用于控制和操作显示屏相关的功能，如显示字符、设置显示模式、光标控制等。这个中断提供了一些基本的显示和图形操作，允许程序直接与显示硬件进行交互，从而在屏幕上显示文本和图形。 | | | | | | | |
|  | | | | | | | |
| 1. **实验结果总结**   （对实验结果进行分析。并理论联系实际，思考并列出本实验对应的OS原理的知识点，并说明本实验中的实现部分如何对应和体现了原理中的基本概念和关键知识点。） | | | | | | | |
| 安装虚拟运行环境和Ubuntu操作系统：  操作系统安装与配置对了解操作系统的安装过程和虚拟环境的搭建对深入理解操作系统的构建和运行环境是至关重要的。  而下载、编译并安装Bochs环境则体现了操作系统的虚拟性。了解虚拟化技术以及模拟运行环境对于操作系统的开发、测试和调试是非常重要的。同时使用bximage创建虚拟软驱对于我们了解存储设备的创建、配置和管理对于操作系统的存储管理和文件系统设计是至关重要的。  而运行并调试程序则体现了我们自己搭建的操作系统引导加载程序的功能。代码中的中断处理和系统调用等部分则是对于我们操作系统知识的回顾与巩固。 | | | | | | | |
| 五、个人分工及心得体会  （每个人分别填写自己在本次实验中的分工，并总结实验的心得体会。） | | | | | | | |
| **辜汝曦：**  在实验报告中负责第二部分和第三部分中“第一个bochs程序”的撰写。  心得体会：首先，这次实验在bochsrc上我犯了典型的错误，把它放在bochs 2.8.6文件夹下，导致bochs -f bochsrc时程序报错。其实当bochs一旦编译安装完成，就不用管这个文件夹，只需要调用命令行中的bochs命令。其次，实验练习帮助我回顾了汇编语言的基本知识点，譬如几种典型的中断，loop、bne、beq等指令的用法等。最后，它帮助我熟悉了OS实验常见的开发环境与工具，为接下来的实验打好基础。  杨馨悦：  独立完成本次实验，并撰写报告中实验练习的第1、2题。  通过本次实验，我熟悉了VMWare下安装Ubuntu32位开发环境的方法，了解了编译安装bochs环境的基本过程和BIOS的启动过程，复习了汇编语言的基本知识。同时学会了使用bochs自带工具bximage创建虚拟软驱，熟悉了使用bochs调试汇编程序的基本步骤及Bochs的调试基本命令。在具体的实验操作过程中，出现了一系列报错问题，但好在我通过积极查阅相关资料、询问老师同学解决了这些问题。这也提醒我：理论与实践之间存在巨大鸿沟，只有在掌握理论知识的基础上，积极投身实践，勤动手动脑才能够真正学以致用，熟能生巧。我相信，本次实验将会为后续的操作系统实验的开展打下良好基础。  赵敏：  独立完成本次实验，并撰写报告中实验练习的第3、4题。  在这次实验中，通过对bochs的安装调试配置，我熟悉了对bochs虚拟机的使用，一些低级错误使得我在配置bochsrc文件的过程中频频发生段错误，幸而得到班级群中同学们过往经验的提示和网上资料的补充，同时由于bochs版本的不同对指令和配置都做出改进，才能完美完场bochs的配置。在编写第4题的过程中对于汇编语言的知识也有了一个笼统的复习和回顾，然而未曾提前想到使用清屏功能使得屏幕看起来更加整洁是一个小小的缺憾。但是本次实验依然收益颇多，更大大增强了我的动手能力。  张子航：  独立完成本次实验，并撰写报告中实验练习的第5，6题，实验结果总结。  在本次实验中，我学会了如何搭建基本的实验环境，bochs环境的配置，也学习了bochs 的基本调试指令。于此同时，也复习了汇编语言的基础知识和引导加载程序的工作原理。我们学会了如何调用系统中断以及如何使用中断来与硬件交互。这为我们进一步学习操作系统和底层编程打下了良好的基础。此外，自己的动手能力和独立思考能力也有提升，也为接下来的实验打下来基础。 | | | | | | | |