|  |  |  |  |
| --- | --- | --- | --- |
| **武汉大学国家网络安全学院教学实验报告** | | | |
| 课程名称 | 操作系统及安全设计 | 实验日期 | 2019/10/11 |
| 实验名称 | 实验环境搭建 | 实验序号 | 一 |
| 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；必要的原理分析） | | | |
| **实验目的：**  搭建基本实验环境，熟悉基本开发与调试工具  对应章节：第一、二章  **实验内容：**  1.认真阅读章节资料  2.在实验机上安装virtualbox，并安装ubuntu  3.安装ubuntu开发环境，32位环境  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，运行并调试你的第一个程序：  删除AA55，观察程序效果，找出原因  修改程序中输出为，一个任意长度的打印字符串，调试程序  把生成的可执行文件反汇编，并设置断点进行调试，打印过程  回答：为什么要对段寄存器进行赋值  回答：如何在该程序中调用系统中断 | | | |
| 1. 实验环境及实验步骤   （本次实验所使用的器件、仪器设备等的情况；具体的实验步骤） | | | |
| 实验环境：  VMwareWorkstationPro 15.5.0  Ubuntu 12.04.5 desktop i386 32位  bochs 2.6.9  关键技术：  Bochs的使用，主要在于反汇编与单步调试  汇编代码的理解  Makefile的使用  实验步骤：  1.在实验机上安装virtualbox，并安装ubuntu开发环境，32位环境  在官网下载VMware Workstation Pro 15.5.0并安装  在http://mirrors.163.com/ubuntu-releases/，下载：32位Ubuntu，12.04LTS  之后在VMware Workstation Pro中利用ubuntu iso文件安装虚拟机  并安装vmwaretools工具  结果如图：       1. 下载bochs源码，编译并安装bochs环境   （1）首先Bochs下载：bochs 2.6.9  <http://bochs.sourceforge.net/getcurrent.html>  下载完毕后复制到ubuntu   1. 安装必要环境   先修改ubuntu源  修改ubuntu源方法：  <http://mirrors.163.com/.help/ubuntu.html>  deb IMG_256http://mirrors.163.com/ubuntu/ precise main universe restricted multiverse  deb-src IMG_257http://mirrors.163.com/ubuntu/ precise main universe restricted multiverse deb IMG_258http://mirrors.163.com/ubuntu/ precise-security universe main multiverse restricted deb-src IMG_259http://mirrors.163.com/ubuntu/ precise-security universe main multiverse restricted deb IMG_260http://mirrors.163.com/ubuntu/ precise-updates universe main multiverse restricted deb IMG_261http://mirrors.163.com/ubuntu/ precise-proposed universe main multiverse restricted deb-src IMG_262http://mirrors.163.com/ubuntu/ precise-proposed universe main multiverse restricted deb IMG_263http://mirrors.163.com/ubuntu/ precise-backports universe main multiverse restricted deb-src IMG_264http://mirrors.163.com/ubuntu/ precise-backports universe main multiverse restricted deb-src IMG_265http://mirrors.163.com/ubuntu/ precise-updates universe main multiverse restricted  之后安装必要环境   sudo apt-get install g++  sudo apt-get install build-essential  Sudo apt-get install libgtk2.0-dev  Sudo apt-get install bison  Sudo apt-get install libx11-dev  libxrandr-dev  libsdl1.2-dev  vgabios  /\*bximage\*/  （3）之后安装bochs  $tar vxzf bochs-2.6.9.tar.gz    $cd bochs-2.6.9    $./configure --enable-debugger --with-sdl --enable-disasm  --enable-readline LIBS='-lX11'    $make    $sudo make  install  输入命令bochs  如下图，说明安装成功     1. 使用bochs自带工具bximage创建虚拟软驱   控制台输入bximage即可    注意创建的a.img位于Home     1. 阅读、编译boot.asm，并反汇编阅读 2. 阅读boot.asm     Boot.asm相当于写了一个引导程序  CPU通过CS：IP寄存器中的内容找到要执行的代码片段所在的实际物理内存地址  Mov ax ,cs  Mov ds ，ax  Mov es，ax  相当于给定代码段和数据段，和附加段基地址。  之后DispStr调用10h 13号中断展示BootMessage字符串，规定的展示位置，字体颜色等等。  之后BootMessage db “...”定义了字符串  Times 510 -（$-$$） db 0 计算需要多少字节0才能填充剩余0，使得从开头到当前，再加填充的0构成510字节  最后dw 0Xaa55 定义最后两字节为aa55。共512字节,最后两字节为aa55，固为引导程序。  org 规定程序起始地址为07c00h  cs 代码段寄存器，  ds 数据段寄存器  es 附加段寄存器  bp 基址  ES 段地址 BP偏移地址  cx，ax，bx 数据寄存器  ah ax高8位，al ax低8位  dl dx 低8位  db 定义字节变量  Int interrupt  **dw**定义字类型变量  mov a，b 将b值赋给a  Jmp $ 表示死循环  $表示当前地址，表示当前行被汇编后的地址。  $$表示节的开始处被汇编后的地址  引导程序 BIOS将所检查启动磁盘的第一个扇区512字节载入内存，放于内存0x0000:0x07c00处。 如果第一扇区最后两个字节是AA55，那么它就是一个引导程序。  引导程序特点 大小是512字节，不能多也不能少，因为BIOS只读取512B到内存中。 它结尾必须是55AA，这是引导扇区标志。 它总是放在磁盘第一个扇区上（0磁头0此道1扇区）因为BIOS只读取第一个扇区。  Int 10h  10H中断的13号中断用于显示字符串，参数为：  AH＝13H  AL＝显示方式  BH表示视频区页数  如果AL的BIT1为0，则BL表示显示属性。  CX为字符串长度  DH表示在第几行显示（0为第一行）  DL表示在第几列显示（0为第一列）  ES：BP指向字符串   1. 编译boot.asm   nasm boot.asm -o boot.bin     1. 修改bochsrc，运行并调试你的第一个程序： 2. 修改bochsrc   修改vgaromimage和romomage对应的文件位置，注意是bochs安装后的位置，而不是tar解压的bochs文件夹中的文件  注释掉keyboard\_mapping一行  增加display\_library: sdl    修改后相当于规定了启动软盘为a.img   1. Makefile   先看Makefile文件    意为：先解压a.img.gz，然后汇编编译boot.asm,然后用boot.asm填充a.img,然后运行bochs  Clean rm -f 为删除文件  ./configure 是用来检测你的安装平台的目标特征的。比如它会检测你是不是有CC或GCC，并不是需要CC或GCC，它是个shell脚本。  make 是用来编译的，它从Makefile中读取指令，然后编译。  make install是用来安装的，它也从Makefile中读取指令，安装到指定的位置。  Bochs从当前目录寻找bochsrc作为配置文件   1. 运行   Cd 到目标文件夹  Make  就先后完成解压，编译，打开了bochs    然后输入6开始simulation  记得要输入c continue         1. 调试删除AA55,查看效果   修改后的boot.asm    **Nasm编译为b2.bin**  **Bximage 创建新的floppy disk a.img**  **写入**  **Bochs运行**    **发现效果为:无法运行，直接退出**  **原因：因为位于1扇区的512字节，且最后字节为aa55的程序为引导程序，a.img作为引导盘，aa55为标志，删除后相当于没有引导程序，故无法运行直接退出**   1. 修改输出字符串   只需要根据想要输出的字符串修改BootMessage，根据BootMessage的长度修改cx即可  举例，输出just monika  修改后的asm    运行效果     1. 把生成的可执行文件反汇编，并设置断点进行调试，打印过程   用nasm，ndisam可以反汇编  用bochs也可以反汇编，并实现单步调试  具体指令参考教材第二章，或输入 help查看命令  就对写入了原始boot.asm编译产生的boot.bin的a.img用bochs调试    b 0x7a00 设置断点  C 运行到断点  Info cpu查看cpu寄存器      x /64xb 0x7c00 查看内存 64为sequence长度，x为16进制显示，b为长度为byte  Help x 可以具体查看参数意义。    **n 单步运行 （遇到函数会跳过）**    trace-reg on 让bochs每一步都显示主要寄存器的值  U 为反汇编 disasm      可见在调用DispStr函数后，程序一直循环    6.回答相关问题  （1）回答：为什么要对段寄存器进行赋值  cs为代码段基址，赋给ds，es，确定程序运行的基址，比如数据段基址，之后加上bp，ES:BP就构成了BootMessage的地址  （2）回答：如何在该程序中调用系统中断  使用汇编int 10h 调用中断，同时设置好其参数，比如cx为字符串长度，ah=13表示输出字符串，AL=01h为显示器模式，bx 000c为颜色，dl 0 表示在第一行输出，ES:BP存储字符串地址 | | | |
| 1. 实验过程分析   （实验分工，详细记录实验过程中发生的故障和问题，进行故障分析，说明故障排除的过程及方法。根据具体实验，记录、整理相应的数据表格等） | | | |
| **遇到的问题及解决：**  **1.报错**  **IMG_256**   可以考虑在执行configure时添加参数LIBS='-lX11'  ./configure --enable-debugger --enable-disasm --enable-readline LIBS='-lX11'  2.报错make: \*\*\* [Makefile:429: install\_bin] Error 1  解决 加上sudo权限即可  IMG_256  **实验结果记录：**  1.安装好bochs：    2.bximage 创建img：    3.原始的boot.asm编译运行：    4.删除aa55后编译运行：  直接退出，无法运行  5.修改，输出任意字符串    6.反汇编与单步调试：  b 0x7a00 设置断点  C 运行到断点  Info cpu查看cpu寄存器      x /64xb 0x7c00 查看内存    **n 单步运行 （遇到函数会跳过）**    trace-reg on 让bochs每一步都显示主要寄存器的值  U 为反汇编 disasm      可见在调用DispStr函数后，程序一直循环 | | | |