|  |  |  |  |
| --- | --- | --- | --- |
| **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 | 2019/11/8 |
| 实验名称 | 让操作系统走进保护模式 | 实验序号 | 五 |
| 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：**  • 如何从软盘读取并加载一个Loader程序到操作  系统，然后转交系统控制权  • 对应章节：第四章  **实验内容：**  1. 向软盘镜像文件写入一个你指定的文件，手  工读取在磁盘中的信息  2. 在软盘中找到指定的文件，读取其扇区信息  3. 将指定文件装入指定内存区，并执行  4. 学会在bochs中使用xxd读取反汇编信息  **完成本次实验要思考的问题：**  1.FAT12格式是怎样的？  2.如何读取一张软盘的信息  3.如何在软盘中找到指定的文件  4.如何在系统引导过程中，从读取并加载一个可执行文件  到内存，并转交控制权？  5.为什么需要这个Loader程序不包含dos系统调用？ | | | |
| 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：**  引导扇区，loader与控制权转交。  一个操作系统从开机到开始运行，大 致经历“引导→加载内核入内存→跳入保护模式→开始执行内核”这样一个过程。也就是说，在内核开始执行之前不但要加载内 核，而且还有准备保护模式等一系列工作，如果全都交给引导扇区来做，512字节很可能是不够用的，所以，把这个过程交给另外的模块来完成，我们把这个模块叫做Loader。引导扇区负责把Loader加载入内存并且把控制权交给它，其他工作交给 Loader来做，因为它没有512字节的限制，将会灵活得多。  在这里，为了操作方便，把软盘做成FAT12格式，这样对Loader以及今后的Kernel（内核）的操作将会非常简单易行。  **实验步骤：**  1. 向软盘镜像文件写入一个你指定的文件，手工读取在磁盘中的信息   1. 修改引导扇区   增加BPB等信息以被识别。修改boot.asm，生成boot.bin，写入已引导扇区。（boot.asm在第一章出现）    把生成的Boot.bin写入磁盘引导扇区，运行的效果没有变，仍然会是图1.1的样子。但是，现在的软盘已经能够被DOS以及 Linux识别了，我们已经可以方便地往上添加或删除文件了。  修改bochsrc:  修改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  增加display\_library: sdl     1. 一个简单的loader   Loader.asm。编译为loader.bin     1. 读软盘，根目录部分得loader起始扇区号   为加载loader.bin到软盘，需要读软盘。  核心思想为修改boot.asm，引导扇区，使其功能改为读软盘，寻找loader.bin  用bios中断 int 13h读软盘。    中断需要的参数不是原来提到的从第0扇区开始的扇区号，而是柱面号、磁头号以及在当前柱面上的扇区号3个分量，所以需要我们自己来转换一下。对于1.44MB的软盘来讲，总共有两面（磁头号0和1），每面80个磁道（磁道号0～79），每个 磁道有18个扇区（扇区号1～18）。下面的公式就是软盘容量的由来： 2×80×18×512=1.44MB  于是，磁头号、柱面（磁道）号和起始扇区号可以用图所示的方法来计算。  注意如Q=0,1,2,3,4，0与1为柱面0，在两面为磁道0.0为磁头0,1为磁头1.  可知写软盘时先写一个柱面，上下磁道都写满了再切换柱面。    对应，写读软盘函数到boot.asm。    由于上述代码用到堆栈，故有初始化堆栈，初始化ss和esp    之后写查找loader.bin的函数      遍历根目录区所有的扇区，将每一个扇区加载入内存，然后从中寻找文件名为Loader.bin的条目，直到找到为止。找到的那一刻，es:di是指向条目中字母N后面的那个字符。其中宏定义与变量      由于在读取过程中打印一些字符串，我们需要一个函数来做这项工作。为了节省代码长度，字符串的长度都设为9字节，不够则用空格补齐，这样就相当于一个二维数组，定位的时候通过数字就可以了。显示字符串的函数DispStr，调用它的时候只要保证寄存器dh的值是字符串的序号就可以了。     1. 写入boot.bin，loader.bin到软盘并反汇编调试   写入：    但此时boot.bin只是找到了loader.bin，运行不会有效果，所以加断点反汇编调试。  b 0x7c00 是因为bios把boot sector加载到0x7c00处。见boot.asm    N 单步执行，遇到函数则跳过。这里跳过了BPB  U 反汇编。/45 为count，反汇编的指令个数。用help x可以查看信息        然后书上b 0x7cb4是在boot.bin的jmp $处下断点，根据实际情况(0x7cad处是jmp.-2,jmp $)，我在b 0x7cad处下断点，然后  **x /32xb es:di - 16** ←查看*es:di* 前后的内存  **x /13xcb es:di - 11** ←容易发现*es:di* 前乃我们要找的文件名  **sreg** ←查看*es*  r查看di      可见拷贝成功。   1. 根据（3）读根目录得到的扇区号，读FAT将loader加载到内存   继续修改boot.asm。  现在我们已经有了Loader.bin的起始扇区号，我们需要用这个扇区号来做两件事：一件是把起始扇区装入内存，另一件则是通过它找到FAT中的项，从而找到Loader占用的其余所有扇区。  在这里，我们把Loader装入内存的BaseOfLoader:OffsetOfLoader处  写一个函数来找到FAT中的项。函数的输入就是扇区号，输出则是其对应的FAT项的值    新增加了宏SectorNoOfFAT1，它与前面提到的RootDirSectors、SectorNoOfRootDirectory等宏一起，与FAT12有关的几个数字我们都定义成了宏，而不是在程序中进行计算。一方面，这是为缩小引导扇区代码考虑；  另一方面，这些数字一般情况下是不会变的，写代码计算它们其实是一种浪费。  由于一个FAT项可能跨越两个扇区，所以在代码中一次总是读两个扇区，以免在边界发生错误。  之后加载loader    新的宏DeltaSectorNo。根据下面的例子来看，文件RIVER.TXT对应的目录条目中的开始簇号是2。实际上，开始簇号是2对应的是数据区的第一个扇区。所以，我们需要有一个方法来计算簇号为X代表从引导扇区开始算起是第几个扇区。 根目录区占用RootDirSectors也即14个扇区，根目录区的开始扇区号是19，于是用“X+RootDirSectors+19-2”来算出“33”这个正确的扇区号。所以，我们又定义了一个宏DeltaSectorNo为17（即19-2）来帮助计算正确的扇区号： DeltaSectorNo **equ** 17   1. 向loader移交控制权   上面的代码调试通过后，我们就已经成功地将Loader加载入内存，下面让我们来一个跳转，开始执行Loader     1. 整理boot.asm 测试   为了在执行时实现更好的效果，增加如下代码  2首先清屏，然后显示字符串“Booting”。这样，加载Loader时打印的圆点也会出现在这个字符串的后面。 屏幕上的圆点数目表明我们读了几个扇区就把Loader加载完毕。    在加载完毕跳入loader前打印ready    更新引导扇区和loader。运行  修改bochsrc    Cp boot.bin，cp loader.bin    运行，成功       1. 总结   Loader.bin本质上是个.COM文件，最大也不可能超过64KB。但是，我们已经成功突破512字节限制，这个进步无疑是巨大的。  Linux的的引导扇区代码Boot.s比我们的代码简单，它直接把内核移动到目标内存。我们的代码之所以复杂一些，是因为我们想和MSDOS的磁盘格式兼容，以便调试的时候容易一些。比如现在，我们就完全可以把第3章中的代码pmtest9.asm编译一下，将编译后的二进制命名为Loader.bin并复制到刚刚引导过 的软盘中覆盖掉原来简陋的Loader.bin。你会发现程序马上可以执行，结果如图所示。    现在的Loader仅仅是个Loader，它不是操作系统内核，也不能当做操作系统内核。我们希望自己的操作系统内核至少应该可以在Linux下用GCC编译链接，要不然，永远用汇编一点一点地写下去实在是太痛苦了。  那么，现在我们假设已经有了一个内核，Loader肯定要加载它入内存，而且内核开始执行的时候肯定已经在保护模式下了，所以，Loader要做的事情至少有两件：  加载内核入内存。 跳入保护模式。   1. 在软盘中找到指定的文件，读取其扇区信息   见1.（3）读软盘，根目录部分得loader起始扇区号   1. 将指定文件装入指定内存区，并执行   见1.（5）根据（3）读根目录得到的扇区号，读FAT将loader加载到内存   1. 学会在bochs中使用xxd读取反汇编信息   见1.（4）写入boot.bin，loader.bin到软盘并反汇编调试  **完成本次实验要思考的问题：**   1. FAT12格式是怎样的？   FAT12 是DOS时代就开始使用的文件系统（File System），直到现在仍然在软盘上使用。  几乎所有的文件系统都会把磁盘划分为若干层次以方便组织和管理，这些层次包括：  扇区（Sector）：磁盘上的最小数据单元。  簇（Cluster）：一个或多个扇区。  分区（Partition）：通常指整个文件系统。  引导扇区是整个软盘的第0个扇区，在这个扇区中有一个很重要的数据结构  叫做BPB（BIOS ParameterBlock），引导扇区的格式如表所示，其中名称以BPB\_开头的域属于BPB，以BS\_开头的域不属于BPB， 只是引导扇区（Boot Sector）的一部分。    紧接着引导扇区的是两个完全相同的FAT表，每个占用9个扇区。第二个FAT之后是根目录区的第一个扇区。根目录区的后面是数据区，如图所示。    要把Loader复制到软盘上并让引导扇区找到并加载它，来看一下引导扇区通过怎样的步骤才能找到文件，以及如何能够把文件内容全都读出来并放进内存里。 为简单起见，我们规定Loader只能放在根目录中，而根目录信息存放在FAT2后面的根目录区中。  根目录区。 根目录区位于第二个FAT表之后，开始的扇区号为19，它由若干个目录条目（Directory Entry）组成，条目最多有BPB\_RootEntCnt个。由于根目录区的大小是依赖于BPB\_RootEntCnt的，所以长度不固定。  根目录区中的每一个条目占用32字节，格式如表所示。  主要定义了文件的名称、属性、大小、日期以及在磁盘中的位置。    举例：  创建一个虚拟软盘，假设是x.img，把它作为FreeDos的B盘，格式化后就可以往其中添加文件和目录了（比如使用FreeDos 里的edit.exe）。这样，当我们想查看它的格式时，只需用二进制查看器打开x.img就可以。  通过FreeDos在这张虚拟软盘中添加以下几个文本文件：  RIVER.TXT，内容为riverriverriver。  FLOWER.TXT，内容为300个单词flower，用来测试文件跨越扇区的情况。（可以先建一个小文件，最后再把它改长，这 样可以让它对应的簇不连续，便于观察和理解。）  TREE.TXT，内容为treetreetree。  再添加一个HOUSE目录，然后在目录nHOUSE下添加两个文本文件：  179CAT.TXT，内容为catcatcat。  DOG.TXT，内容为dogdogdog。  由于根目录区从第19扇区开始，每个扇区512字节，所以其第一个字节位于偏移19\*512=9728=0x2600处。用二 进制查看器来看看x.img的偏移0x2600处是什么，如下    以RIVER.TXT为例，它的各项值如表所示。    当我们寻找Loader时，只要发现文件名正确就认为它是我们要找的那一个文件。最后剩下最重要的信息DIR\_FstClus，即文件开始簇号，它告诉我们文件存  放在磁盘的什么位置，从而让我们可以找到它。由于一簇只包含一个扇区，所以简化了计算过程，而且下文中说到“簇”的地方， 你也可以将它替换成“扇区”。  需要注意的是，**数据区的第一个簇的簇号是2，而不是0或者1。**  RIVER.TXT的开始簇号就是2，也就是说，此文件的数据开始于数据区第一个簇。  计算根目录区所占的扇区数:根目录区条目最多有BPB\_RootEntCnt个，扇区数假设根目录区共占用RootDirSectors个扇区，则有：  之所以分子要加上(BPB\_BytsPerSec—1)，是为了保证此公式在根目录区无法填满整数个扇区时仍然成立。    在本例中，容易算出RootDirSectors=14。所以：数据区开始扇区号=根目录区开始扇区号+14=19+14=33  第33扇区的偏移量是0x4200（512×33），让我们看一下这里的内容，如下：    对于小于512字节的文件来说，FAT表用处不大，但如果文件大于512字节，我们需要FAT表来找到所有的簇（扇区）。FAT表有两个，FAT2可看做是FAT1的备份，它们通常是一样的。FAT1的开始扇区号是1，偏移为512字节（0x200），如下：    每12位称为一个FAT项（FATEntry），代表一个簇。第 0个和第1个FAT项始终不使用，从第2个FAT项开始表示数据区的每一个簇，也就是说，第2个FAT项表示数据区第一个簇，依此类推。前文说过，数据区的第一个簇的簇号是2，和这里是相呼应的。  由于每个FAT项占12位，包含一个字节和另一个字节的一半。假设连续3个字节分别如图所示，那么灰色框表示的是前一个FAT项（FATEntry1），BYTE1是FATEntry1的低8位，BYTE2的低4位是 FATEntry1的高4位；白色框表示的是后一个FAT项（FATEntry2），BYTE2的高4位是FATEntry2的低4位，BYTE3是FATEntry2的高8 位。    通常，FAT项的值代表的是文件下一个簇号，但如果值大于或等于0xFF8，则表示当前簇已经是本文件的最后一个簇。如果值为0xFF7，表示它是一个坏簇。  文件RIVER.TXT的开始簇号是2，对应FAT表中的值为0xFFF，表示这个簇已经是最后一个。 **(FF 8F 00 注意是低地址，取ff和8F中的F)**  我们来看一个长一点的文件FLOWER.TXT，它的DIR\_FstClus值为3，对应第3个FAT项。结合我们打印出的FAT表内容我们知道，此FAT项值为0x008，也就是说，这个簇不是文件的最后一个簇，下一个簇号为8。我们再找到第8个FAT项，发现值为0x009， 接下来第9个FAT项值为0x00A，第0xA个FAT项值为0xFFF。所以，FLOWER.TXT占用了第3、8、9、10，共计4个簇。  这里需要注意一点，一个FAT项可能会跨越两个扇区，这种情况在编码实现的过程中要考虑在内。   1. 如何读取一张软盘的信息   使用bios中断 int 13h     1. 如何在软盘中找到指定的文件   先根据文件名遍历根目录区，找到起始扇区，然后根据起始扇区号查找对应FAT项   1. 如何在系统引导过程中，从读取并加载一个可执行文件   到内存，并转交控制权？  在引导扇区boot sector中编写函数，实现先根据文件名遍历根目录区，查找文件起始扇区号，然后根据起始扇区号找到对应起始FAT项，接着根据FAT项加载对应扇区到内存，直到加载完毕。（FAT值为FFF）  之后跳转到加载内存的起始位置，开始执行，就移交了控制权。   1. 为什么需要这个Loader程序不包含dos系统调用？   Loader调用的是bios的中断。Dos是操作系统，要用loader装入，装入前无法调用。因为要用loader加载内核，跳入保护模式。 | | | |
| 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. **对更新引导扇区和loader的操作不熟悉**   复习第二章第三章内容后解决。  **实验结果记录：**   1. **修改引导扇区增加BPB**      1. 写入boot.bin，loader.bin到软盘并反汇编调试                  1. 据（3）读根目录得到的扇区号，读FAT将loader加载到内存，移交控制权，运行   修改bochsrc    Cp boot.bin，cp loader.bin    运行，成功 | | | |