|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 武汉大学国家网络安全学院教学实验报告 | | | | | |
| 课程名称 | 操作系统设计与实践 | | 实验日期 | |  |
| 实验名称 |  | | 实验周次 | |  |
| 姓名 | 学号 | | 专业 | | 班级 |
| 王卓 | 2021302191791 | | 网络空间安全 | | 9班 |
|  |  | |  | |  |
|  |  | |  | |  |
|  |  | |  | |  |
| 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） | | | | | |
| **本次实验内容**   1. 汇编和C的相互调用方法   --在例程基础上，在汇编与C程序中各添加一个简单带参数的函数调用，让两种语言撰写的程序实现混合调用，功能可自定义。   1. ELF文件格式   --使用xxd命令分析ELF文件中的ELF header、Program header，了解各项的作用   1. 使用Loader加载ELF文件，重新放置内核 2. 扩展内核，切换堆栈和GDT、整理文件结构、使用makefile编译程序、添加中断处理 3. 设计题：修改启动代码，在引导过程中在屏幕上画出一个你喜欢的ASCII图案，并将第三章的内存管理功能代码、你自己设计的中断代码集成到你的kernel文件目录管理中，并建立makefile文件，编译成内核并引导   **思考问题：**   1. 汇编和C内定义的函数，相互间调用的方法是怎样的？ 2. 描述ELF文件格式以及作用，和大家学习的PE相比，结构上有什么相同和差异？ 3. 如何从Loader加载ELF，如何确定ELF文件加载到内存的位置？ 4. 对照书中例程代码，这个内核扩展了哪些功能，这些功能流程是怎样的，他们都是在哪些源文件的代码中进行描述的？这些功能彼此有相互关联吗，给出说明？ 5. 书中代码内存的布局是怎样的？在这里有哪些是特权代码，哪些是非特权代码，在处理器控制权切换时，权限变化情况如何？ 6. 下载一个真正的内核源文件，分析一下是怎么在管理组织源码文件的（选做）。 7. 完成设计题并能演示。 | | | | | |
| 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：**  略  **具体实验步骤：**   1. 汇编和C的相互调用方法   --在例程基础上，在汇编与C程序中各添加一个简单带参数的函数调用，让两种语言撰写的程序实现混合调用，功能可自定义。  略。。。   1. ELF文件格式   --使用xxd命令分析ELF文件中的ELF header、Program header，了解各项的作用   1. ELF header   具体的ELF文件格式以及作用的描述在后续思考题部分讲述，这里我们直接使用xxd命令分析编译后生成的ELF文件中的ELF header  20231113145840  **图x ELF header**  开头的4字节固定不变，第一个字节值为0x7F，紧跟着的就是ELF三个字符，这4字节表明这个文件是一个ELF文件，前16个字节标识ELF文件格式，从ascii码中可以看到ELF字符。  ELF header的格式如下面代码所示：   1. typedef struct { 2. unsigned char e\_ident [ 1 6 ] ; 3. Elf32\_Half         e\_type ; 4. Elf32\_Half         e\_machine ; 5. Elf32\_Word         e\_version ; 6. Elf32\_Addr         e\_entry ; 7. Elf32\_Off          e\_phoff ; 8. Elf32\_Off          e\_shoff ; 9. Elf32\_Word         e\_flags ; 10. Elf32\_Half         e\_ehsize ; 11. Elf32\_Half         e\_phentsize ; 12. Elf32\_Half         e\_phnum; 13. Elf32\_Half         e\_shentsize ; 14. Elf32\_Half         e\_shnum; 15. Elf32\_Half         e\_shstrndx ; 16. }Elf32\_Ehdr ;   下面是ELF header中各项的含义：  ·e\_type：它标识的是该文件的类型，文件foobar的e\_type是 0x02，表明它是一个可执行文件（ExecutableFile）。  ·e\_machine：foobar中此项的值为3，表明运行该程序需要的体系结构为 Intel80386。  ·e\_version：它确定文件的版本，foobar中的版本值是1。  ·e\_entry：程序的入口地址。文件foobar的入口地址为 0x80480A0。  ·e\_phoff：Program header table在文件中的偏移量（以字节计数）。这里的值是0x34。  ·e\_shoff：Section header table在文件中的偏移量（以字节计数）。这里的值是 0x350。  ·e\_flags：对IA32而言，此项为0。  ·e\_ehsize：ELFheader大小（以字节计数）。这里值为0x34。  ·e\_phentsize：Program header table中每一个条目（一个Programheader）的大小。这里值为 0x20。  ·e\_phnum：Program header table中有多少个条目，这里有3个。  ·e\_shentsize：Section header table中每一个条目（一个Sectionheader）的大小，这里值为0x28。  ·e\_shnum：Section header table中有多少个条目，这里有7个。  ·e\_shstrndx包含节名称的字符串表是第几个节（从零开始数）。这里值为 6，表示第 6 个节包含节名称。   1. Program header   我们看到。Program header table在文件中的偏移量是0x34，而ELF header大小(e\_ehsize)也是0x34，可见ELF header后面紧接着就是Program header table。我们使用xxd来分析Program header，它描述的是系统准备程序运行所需的一个段(Segment)或其他信息。  20231113153727  **图x Program header**  程序头表中共有三项(e\_phnum=3)，偏移分别是0x34~0x53、0x54~0x73和0x74~0x93。Program header的格式如下面代码所示：   1. typedef struct { 2. Elf32\_Word p\_type ; 3. Elf32\_Off  p\_offset ; 4. Elf32\_Addr p\_vaddr ; 5. Elf32\_Addr p\_paddr ; 6. Elf32\_Word p\_filesz ; 7. Elf32\_Word p\_memsz; 8. Elf32\_Word p\_flags ; 9. Elf32\_Word p\_align ; 10. }Elf32\_Phdr ;   其中各项的含义如下：  ·p\_type：当前Program header所描述的段的类型  ·p\_offset：段的第一个字节在文件中的偏移  ·p\_vaddr：段的第一个字节在内存中的虚拟地址  ·p\_paddr：在物理地址定位相关的系统中，此项是为物理地址保留  ·p\_filesz：段在文件中的长度  ·p\_memsz：段在内存中的长度  ·p\_flags：与段相关的标志  ·p\_align：根据此项值来确定段在文件以及内存中如何对齐  在foobar中共有三个Program header，取值表如下所示：   |  |  |  |  | | --- | --- | --- | --- | | 名称 | Program0 | Program1 | Program2 | | p\_type | 0x1 | 0x1 | 0x6474E551 | | p\_offset | 0x0 | 0x164 | 0x0 | | p\_vaddr | 0x8048000 | 0x8049164 | 0x0 | | p\_paddr | 0x8048000 | 0x8049164 | 0x0 | | p\_filesz | 0x164 | 0x8 | 0x0 | | p\_memsz | 0x164 | 0x8 | 0x0 | | p\_flags | 0x5 | 0x6 | 0x7 | | p\_align | 0x1000 | 0x1000 | 0x10 |   根据上述消息，可以大致画出文件内偏移地址到虚拟地址的映射关系。  20231113213113  **图x 内存映射**  事实上，这也就是foobar在加载进内存之后的情形。   1. 使用Loader加载ELF文件，重新放置内核 2. 使用Loader加载ELF文件   Loader需要做两项工作：1.加载内核到内存；2.跳入保护模式   * 1. 加载内核到内存   我们希望使用Loader加载ELF文件，在之后肯定是要加载内核文件的。加载一个文件的步骤依旧是寻找文件、定位文件以及读入内存。我们修改loader.asm，让它把内核放进内存。  首先，我们在a盘的目录下寻找KERNEL.BIN  20231114151623  接着，我们执行以下操作去定位文件  20231114151840  可以看到，代码的大致工作和boot.asm是类似的。  加载内核的代码大致完成，这里我们使用一个最简单的kernel.asm文件作为内核（实际算不上内核）来测试，实现的功能是显示一个字符“K”  20231114152236  首先修改bochsrc，然后执行以下命令进行编译：  $ nasm -f elf -o kernel.o kernel.asm  $ld -s -o kernel.bin kernel.o  $sudo mount -o loop a.img /mnt/floppy/  $sudo cp kernel.bin /mnt/floppy/ -v  $sudo umount /mnt/floppy/  结果如下所示：  20231114154738  可以看到，Loading后面出现一个圆点，说明Loader读了一个扇区。现在，内核被我们加载进内存了，不过除了能看到“Ready.”字样之外，没有其他现象出现。   * 1. 跳入保护模式   不同于之前在保护模式中学习的：大部分描述符的段基址是运行时计算后填入相应位置，因为我们不知道段地址，也就不知道程序运行时在内存中的位置。现在，我们不需要这样了，因为我们自己加载了loader，已经确定了段地址为BaseOfLoader，所以在Loader中出现的变量的物理地址可以由以下公式计算：  标号物理地址 = BaseOfLoader \* 10h + 标号的偏移  这样就导致BaseOfLoader同时在boot.asm和loader.asm中使用，于是我们将BaseOfLoader定义在一个文件load.inc中  20231114191327  直接定义了一个宏BaseOfLoaderPhyAddr来表示BaseOfLoader \* 10h  接下来进入保护模式，同之前实验操作，进入之后打印字符“P”  x  执行以下命令进行编译运行：  $ nasm loader.asm -o loader.bin  $sudo mount -o loop a.img /mnt/floppy/  $sudo cp loader.bin /mnt/floppy/  $sudo umount /mnt/floppy/  $sudo bochs -f bochsrc  20231114192001  看到字母“P”，说明成功进入保护模式。  接下来我们打开分页机制，同之前实验，我们先使用15h中断得到获取内存信息。  20231114192421  接着显示并打印内存信息。  20231114193721  得到内存信息之后，启动分页机制，这个基本类似之前实验，故不放图。  这里运行之后，结果如下所示：  20231114194145   1. 重新加载内核   前面编写了一个十分简单的kernel函数并加载到了内存当中，现在就需要对其进行整理，并移交控制权。   * 1. 整理内核程序   编写的内核程序是一个ELF程序，ELF程序的program header table字段有重要含义，整理这个内核程序，实际上是基于program header table的信息进行如下面C语言语句的内存复制：  **memcpy(p\_vaddr, BaseOfLoaderPhyAddr + p\_offset, p\_filesz)**  但是，由于ld生成的可执行文件中p\_vaddr的值较大，在这里超过我们的内存范围，所以我们需要修改一下ld指令时的参数。编译时使用以下命令：  $nasm -f elf -o kernel.o kernel.asm  $ld -s -Ttext 0x30400 -o kernel.bin kernel.o  这样就解决了内存的问题，只需要向内核交出控制权   * 1. 移交控制权   20231114195637  运行结果如下所示：  20231114200132  可以看到，第二行中央出现字符“K”，说明我们的内核开始执行了。   1. 扩展内核，切换堆栈和GDT、整理文件结构、使用makefile编译程序、添加中断处理   略。。。   1. 设计题：修改启动代码，在引导过程中在屏幕上画出一个你喜欢的ASCII图案，并将第三章的内存管理功能代码、你自己设计的中断代码集成到你的kernel文件目录管理中，并建立makefile文件，编译成内核并引导   略。。。 | | | | | |
| 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） | | | | | |
|  | | | | | |
| 1. 实验结果总结   （对实验结果进行分析，完成思考题目，并提出实验的改进意见） | | | | | |
| **思考题：**   1. 汇编和C内定义的函数，相互间调用的方法是怎样的？ 2. 描述ELF文件格式以及作用，和大家学习的PE相比，结构上有什么相同和差异？ 3. ELF文件格式     **图x ELF文件格式**  可以看出，ELF文件由4个部分组成，分别是ELF header、Program header、节和Section header table  ·ELF header  ELF header格式的代码见具体实验步骤2部分。ELF文件力求支持从8位到32位不同架构的处理器。为了使文件格式与机器无关，定义了下表中的这些数据类型。   |  |  |  |  | | --- | --- | --- | --- | | 名称 | 大小 | 对齐 | 用途 | | Elf32\_Addr | 4 | 4 | 无符号程序地址 | | Elf32\_Half | 2 | 2 | 无符号中等大小 | | Elf32\_Off | 4 | 4 | 无符号文件偏移 | | Elf32\_Sword | 4 | 4 | 有符号大整数 | | Elf32\_Word | 4 | 4 | 无符号大整数 | | unsigned char | 1 | 1 | 小符号小整数 |   ·Program header  Program header格式的代码见具体实验步骤2部分。  ·Section header  Section header的结构如下所示：   1. typedef struct { 2. Elf32\_Word  sh\_name ; 3. Elf32\_Word  sh\_type ; 4. Elf32\_Word  sh\_flags ; 5. Elf32\_Addr  sh\_addr ; 6. Elf32\_Off  sh\_offset ; 7. Elf32\_Word  sh\_size ; 8. Elf32\_Word  sh\_link ; 9. Elf32\_Word  sh\_info ; 10. Elf32\_Word  sh\_addralign ; 11. Elf32\_Word  sh\_entsize ; 12. }Elf32\_Shdr ; 13. ELF的作用   ELF的作用可以用下图来进行表示：    **图x ELF的作用**  ELF文件参与程序的连接（建立一个程序）和程序的执行（运行一个程序）。在汇编器和链接器看来，ELF文件是由Section header table描述的一系列section的集合，而执行一个ELF文件时，在加载器（Loader）看来它是由Program header table描述的一系列segment的集合。   1. ELF和PE文件的对比   简单来说，ELF对应于UNIX下的文件，而PE则是Windows的可执行文件，它们都是基于Unix的COFF（Common Object file format）格式的变种，它们都包含了整个文件的基本属性，如文件版本，目标机器型号，程序入口等等。他们都有段表，保存了各种各样段的基本属性，比如段名，段长，文件中的偏移，读写权限。他们都有字符串表，将使用的字符串统一放在那里，然后通过偏移量来引用字符串。他们描述的是相同的信息，只是在数据段的链接方式上有所区别。  然而，二者之间也存在一些差异。例如，ELF文件可以用于不同的硬件平台和操作系统，而PE文件主要使用在32位和64位的Windows操作系统上。此外，ELF文件中的段可以有多种类型，包括代码段、数据段、只读数据段、调试信息等，而PE文件中的段类型可能会有所不同。这些差异主要是由于两种文件格式分别针对不同的操作系统进行了优化。  事实上，Linux系统中有wine命令，它是一个能够在多种“POSIX-compliant”操作系统上运行Windows应用的兼容层。wine通过将Linux不能理解的指令翻译成Linux能够理解的指令，来实现Linux系统下运行Windows程序.   1. 如何从Loader加载ELF，如何确定ELF文件加载到内存的位置？ 2. 对照书中例程代码，这个内核扩展了哪些功能，这些功能流程是怎样的，他们都是在哪些源文件的代码中进行描述的？这些功能彼此有相互关联吗，给出说明？ 3. 书中代码内存的布局是怎样的？在这里有哪些是特权代码，哪些是非特权代码，在处理器控制权切换时，权限变化情况如何？ 4. 下载一个真正的内核源文件，分析一下是怎么在管理组织源码文件的（选做）。 5. 完成设计题并能演示。   **实验的改进意见：**  1.提供更详细的实验指导：在每个实验步骤中，提供更详细的指导和说明，  比如预期结果等，这样能帮助我们更好地完成实验。  2.介绍实验目的和背景：实验开始之前提供实验的目的和背景，解释一下为  什么需要进行该实验以及其与操作系统的关系，这能帮助我们更好理解实验的意  义和重要性。 | | | | | |
| 1. 各人实验贡献与体会（每人各自撰写） | | | | | |
| 王卓：  在本次实验中，本人独立完成大部分实验内容，并负责实验内容第2、3题，思考题第2题实验报告的撰写。  这次的实验总体来说，内容庞杂，涉及的新知识很多且较难理解。尤其对于我网安专业的学生，更是需要花费更多的时间去了解相关基础知识。此外，阅读教材、汇编代码以及熟悉语句和段落的功能对我来说也较为困难。  通过本次实验，我深入地分析、理解并掌握了以下的内容：保护模式的跳转(包括GDT的设置)、用Loader加载ELF文件等。我掌握了C与汇编的相互调用，这对于后续实现复杂功能是十分必要的。在Loader加载ELF的调试过程中，对 ELF的格式有了更加全面的理解。另外，我掌握了makefile的使用，能够更加高效地管理文件与工程。感谢老师和助教准备的这次实验，希望自己能从实验中学会更多的知识，掌握更多的技能。 | | | | | |
| 1. 教师评语 | | | | | |
|  | | | | | |
| **教师评分（请填写好姓名、学号）** | | | | | |
| 姓名 | | 学号 | | 分数 | |
|  | |  | |  | |
|  | |  | |  | |
|  | |  | |  | |
|  | |  | |  | |
| 教师签名：  年 月 日 | | | | | |