**实验五 LinkLab链接实验**

**一、实验目的**

1. 理解链接的作用与工作步骤
2. 掌握 ELF 结构、符号解析与重定位的工作过程
3. 熟练使用 Linux 工具完成 ELF 分析与修改

**二、实验环境**

PC机，Linux 32-bit i386，C/汇编语言

gcc、objdump、gdb、readelf、hexedit

**三、实验学时**

4学时

**四、实验内容**

4.1 静态数据与 ELF 数据节

4.1.1 实验内容

修改二进制可重定位目标文件“phase1.o”的.data 节的内容（不允许修改其它节的内容），使其如下与 main.o 模块链接后运行时输出（且仅输出）自己学号的后10位，如对“202131060407 贾\*\*”，输出“2131060407”。

linux> gcc -m32 -no-pie -o linkbomb main.o phase1.o

linux> ./linkbomb

linux> 2131060407

4.1.2 实验原理与方法

实验原理分析：

当源代码被编译时，它首先被转换为汇编语言，然后由汇编器转换为机器代码，生成的目标文件就是二进制可重定位目标文件。这些文件包含多个节（sections），每个节都有特定的用途，如代码、数据、符号表等。要输出学号的后10位，可以通过直接修改 .data 节中的值来实现。使用readelf查看目标文件的节布局。使用hexedit编辑 .data 节。链接完成，生成的可执行文件运行。

实验步骤：

（1）使用objdump工具获得目标文件的汇编代码,使用readelf

工具获得其重定位记录。

（2）结合汇编代码和重定位信息,定位输出函数的调用参数在目

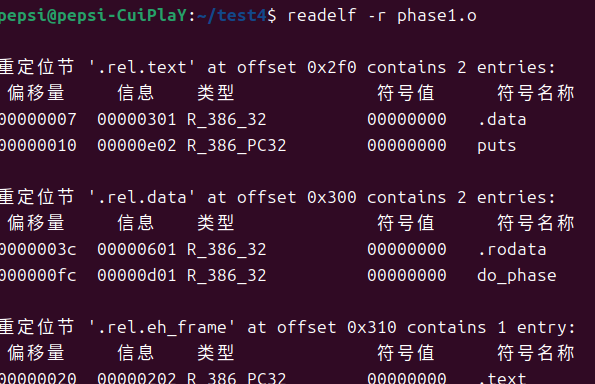
标文件中的存储地址。

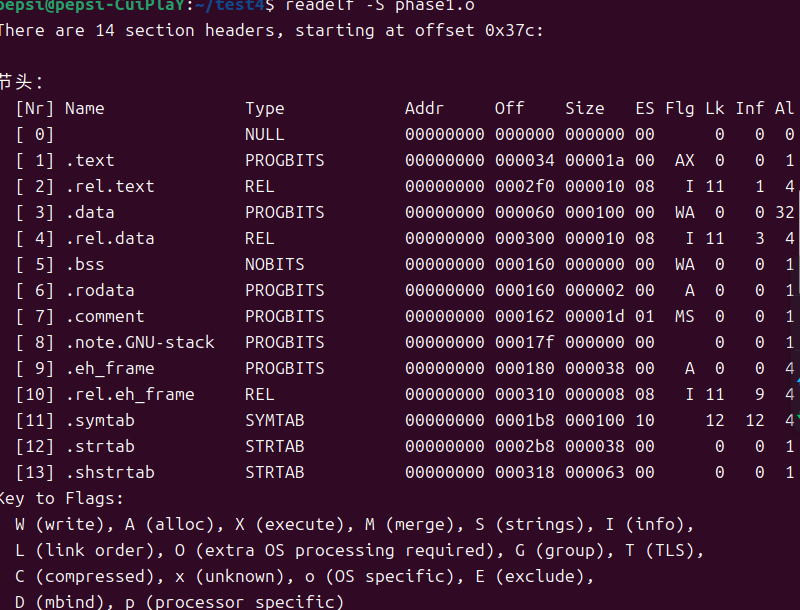
（3）使用hexedit工具,对phase1.o模块的数据节中相应字节进

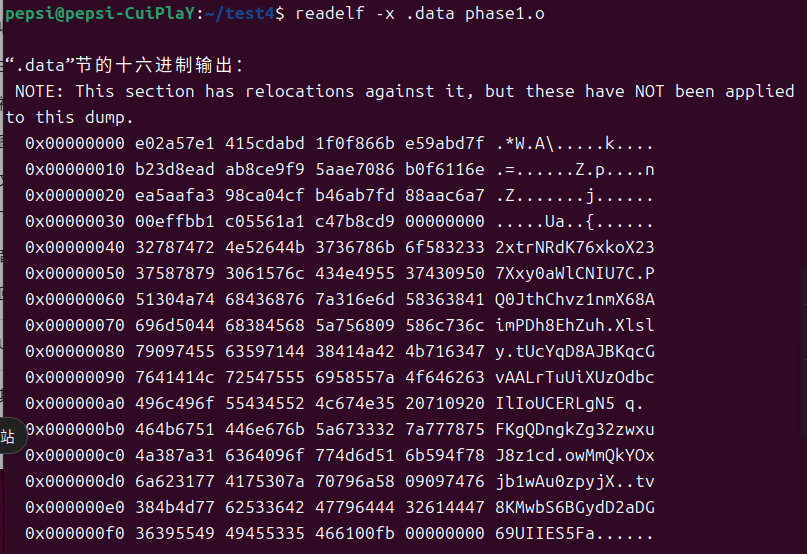
行修改。

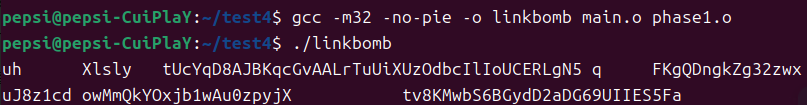
程序运行截屏：

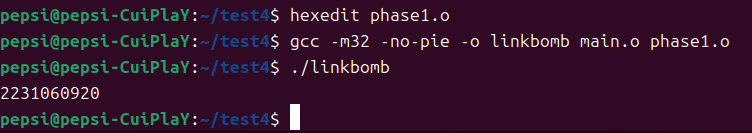












4.1.3 实验中遇到的问题及解决办法

在使用hexedit时提示如下错误：

pepsi@pepsi-CuiPlaY:~/test4$ sudo apt-get install hexedit

正在读取软件包列表... 完成

正在分析软件包的依赖关系树... 完成

正在读取状态信息... 完成

下列【新】软件包将被安装：

hexedit

升级了 0 个软件包，新安装了 1 个软件包，要卸载 0 个软件包，有 10 个软件包未被升级。

需要下载 24.8 kB 的归档。 解压缩后会消耗 70.7 kB 的额外空间。

忽略:1 http://cn.archive.ubuntu.com/ubuntu mantic/universe amd64 hexedit amd64 1.6-1

忽略:1 http://cn.archive.ubuntu.com/ubuntu mantic/universe amd64 hexedit amd64 1.6-1

忽略:1 http://cn.archive.ubuntu.com/ubuntu mantic/universe amd64 hexedit amd64 1.6-1

错误:1 http://cn.archive.ubuntu.com/ubuntu mantic/universe amd64 hexedit amd64 1.6-1

暂时不能解析域名“cn.archive.ubuntu.com”

E: 无法下载 http://cn.archive.ubuntu.com/ubuntu/pool/universe/h/hexedit/hexedit\_1.6-1\_amd64.deb

暂时不能解析域名“cn.archive.ubuntu.com”

E: 有几个软件包无法下载，要不运行 apt-get update 或者加上 --fix-missing 的选项再试试？

解决办法：

出现这种问题的原因是虚拟机使用手机热点网络导致的，将虚拟机的网络设置为桥接模式。

4.2 指令与 ELF 代码节

4.2.1 实验内容

修改二进制可重定位目标文件“phase2.o”的.text 节的内容（不允许修改其它节的内容），使其如下与 main.o 模块链接后运行时输出（且仅输出）自己学号的后10位，如对“202131060407 贾\*\*”，输出“2131060407”。

linux> gcc -no-pie -o linkbomb main.o phase2.o

linux> ./linkbomb

linux> 2131060407

4.2.2 实验原理与方法

实验原理分析：

当源代码被编译时，它首先被转换为汇编语言，然后由汇编器转换为机器代码，生成的目标文件就是二进制可重定位目标文件。这些文件包含多个节（sections），每个节都有特定的用途，如代码、数据、符号表等。要输出学号的后10位，先要编写inst.s文件，同时要保证不会出现内存冲突，将inst文件使用汇编器后再反汇编得到机器码，修改 .text 节中相应的值来实现。使用readelf查看目标文件的节布局。使用hexedit编辑 .text节。链接完成，生成的可执行文件运行。

实验步骤：

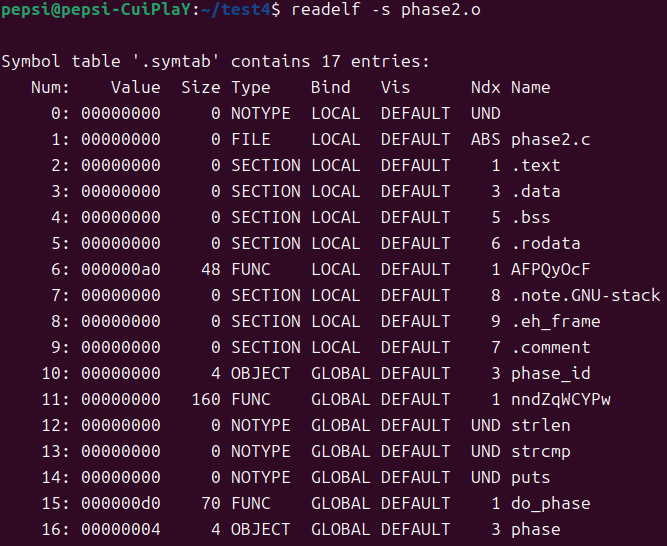
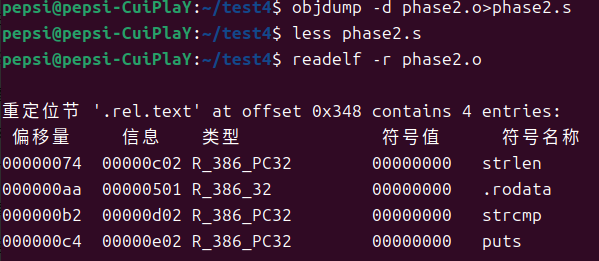
（1）使用objdump工具获得phase2.o目标模块的汇编代码,使用readelf工具获得其重定位记录和符号表。

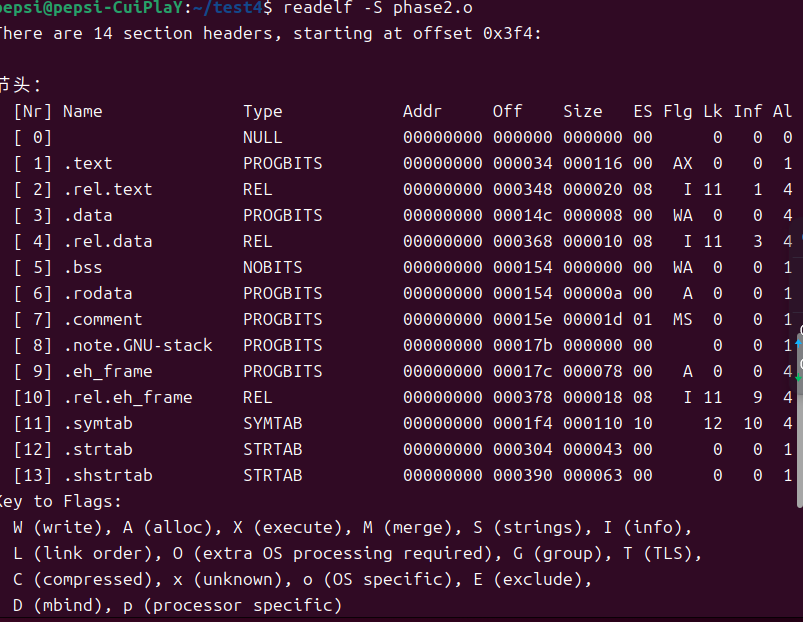
（2）分析汇编代码并结合重定位信息、符号表,推断phase2.o模块中各函数的功能作用,定位出其中负责输出的函数。

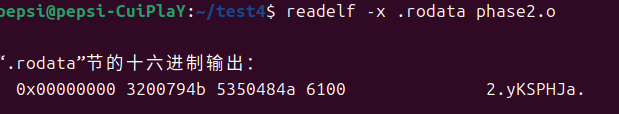
（3）构造调用输出函数的指令代码。

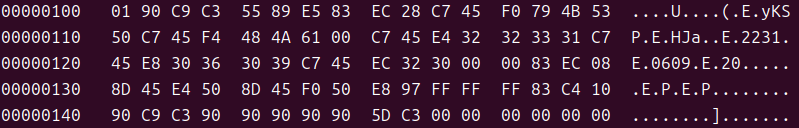
（4）使用上步构造的调用输出函数的指令代码,替换do\_phase()函数体中的nop指令,以实现期望输出。

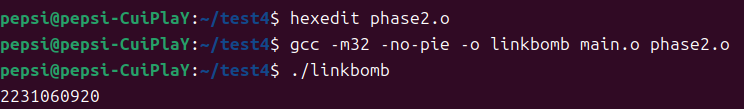
程序运行截图：











4.2.3 实验中遇到的问题及解决办法

实验中会遇到输出结果为学号后10位加上判断的字符串，要解决这种问题需要修改汇编代码，并将得到的相应的机器码仔细认真填在phase2.o文件中对应nop语句中，注意避免内存冲突问题。修改后得到的最终汇编代码如下：

.code32  
sub $0x28, %esp  
movl $0x50534b79,-0x10(%ebp)  
movl $0x614a48,-0xc(%ebp)  
movl $0x31333232,-0x1c(%ebp)  
movl $0x39303630,-0x18(%ebp)  
movl $0x3032,-0x14(%ebp)  
sub $0x8,%esp  
lea -0x1c(%ebp),%eax  
push %eax  
lea -0x10(%ebp),%eax  
push %eax  
call 0x00  
add $0x10,%esp  
nop  
leave  
ret

**五、实验收获**

吐槽一下实验过程中的心路，如：通过实验，提醒自己以后编程要注意什么？

在做这个实验的过程中，我深刻体会到了链接器的重要性和复杂性。以前编程时，经常就是写代码、编译、运行，很少去关注编译之后的过程。这次实验让我发现，原来从源代码到可执行文件，中间还有这么多细节需要处理。通过这次实验，我意识到编程时要注意模块化，尽量做到模块间低耦合、高内聚，这样在链接时才能更顺利。

ELF文件格式的复杂性也让我大开眼界。之前从未深入探究过可执行文件的内部结构，这次不得不深入了解.text、.data等节的结构，以及它们在程序运行中的作用。这让我认识到，了解底层细节对于程序员来说是非常重要的，它能帮助我们更好地理解程序的运行机制，从而写出更高效的代码。

在学习使用Linux工具进行ELF分析与修改时，我的感受是：这些工具真是太强大了！objdump、readelf能帮助我查看ELF文件的内部结构，gdb能让我动态调试程序，而hexedit则能直接修改二进制文件。这些工具的使用经验，无疑增强了我的调试能力，以后遇到问题时，我会更有信心去深入排查。

最后，我想说，这次实验虽然挺折腾人的，但收获满满。它不仅让我了解了链接器的工作原理，还让我学会了如何手动修改ELF文件，这在平时的开发中可能不常用，但作为一个计算机专业的学生，这种底层知识是非常重要的。但最最重要的是要感谢老师在第二个实验对我的帮助，谢谢巧哥！