

**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**实验名称： ELF文件与程序链接**

**院 系 ：计算机科学与技术**

**专业班级 ： 计算机202203**

**学 号 ： U202215643**

**姓 名 ： 王国豪**

**指导教师 ： 朱 虹**

**年 月 日**

**一、实验目的与要求**

通过修改给定的可重定位的目标文件（链接炸弹），加深对可重定位目标文件格式、目标文件的生成、以及链接的理论知识的理解。

实验环境：Ubuntu

工具：GCC、GDB、readelf、hexdump、hexedit、od等。

**二、实验内容**

**任务** 链接炸弹的拆除

**在二进制层面，逐步修改构成目标程序“linkbomb”的多个二进制模块（“.o文件”），然后链接生成可执行程序，要求可执行程序运行能得到指定的效果。修改目标包括可重定位目标文件中的数据、机器指令、重定位记录等。**

**1、第1关 数据节的修改**

修改二进制可重定位目标文件 phase1.o 的数据节中的内容（不允许修改其他节），使其与main.o链接后，生成的执行程序，可以输出自己的学号。

**2、第2关 简单的机器指令修改**

修改二进制可重定位目标文件 phase2.o 的代码节中的内容（不允许修改其他节），使其与main.o链接后，生成的执行程序。在phase\_2.c 中，有一个静态函数 static void myfunc( ) ，要求在 do\_phase 函数中调用myfunc( )，显示信息myfunc is called. Good!。

**3、第3关 有参数的函数调用的机器指令修改**

修改二进制可重定位目标文件 phase3.o 的代码节中的内容（不允许修改其他节），使其与main.o链接后，生成的执行程序。在phase\_3.c 中，有一个静态函数 static void myfunc(int offset) ，要求在 do\_phase函数中调用myfunc(pos )，将do\_phase的参数pos直接传递myfunc，显示相应的信息。

**4、第4关 有局部变量的机器指令修改**

修改二进制可重定位目标文件 phase4.o 的代码节中的内容（不允许修改其他节），使其与main.o链接后，生成的执行程序。在phase\_4.c 中，有一个静态函数 static void myfunc(char \*s) ，要求在 do\_phase 函数中调用myfunc(s )，显示出自己的学号。

**5、第5关 重定位表的修改**

修改二进制可重定位目标文件 phase5.o 的重定位节中的内容（不允许修改代码节和数据节），使其与main.o链接后，生成的执行程序运行时，显示Class Name : Computer Foundation. Teacher Name : Zhu Hong。

**6、第6关 强弱符号**

不准修改 main.c 和phase6.o，通过增补一个文件，使得程序链接后，能够输出自己的学号。

#gcc -no-pie -o linkbomb6 main.o phase6.o phase6\_patch.o

**7、第7关 只读数据节的修改**

修改 phase7.o 中只读数据节（不准修改代码节），使其与main.o链接后，能够输出自己的学号。

**三、实验记录及问题回答**

**（1）实验结果及操作过程记录**

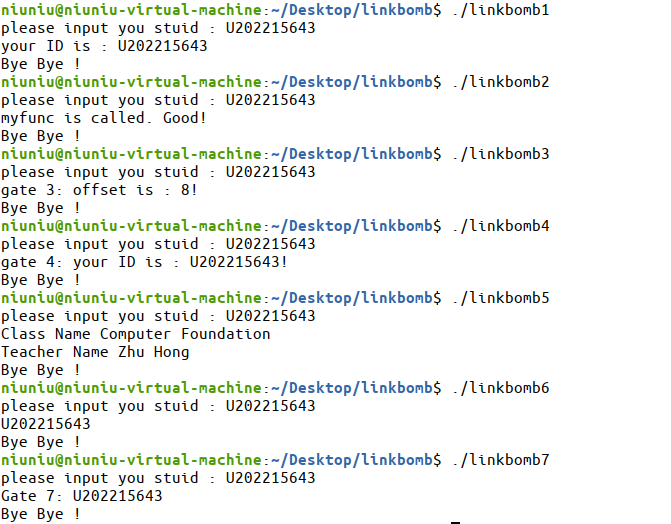


图 1 实验完成截图

下面会展示每个实验的关键修改文件的截图。

**1、第1关 数据节的修改**

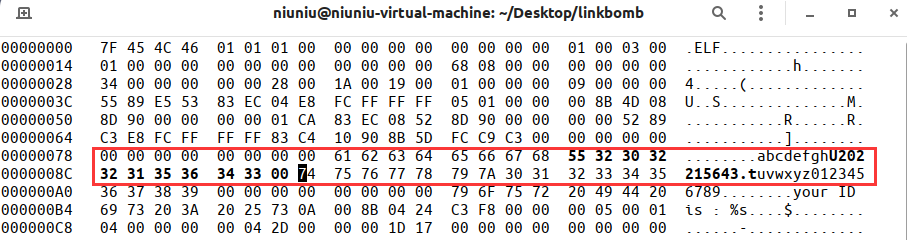


图 2 第1关实验修改文件截图

**2、第2关 简单的机器指令修改**

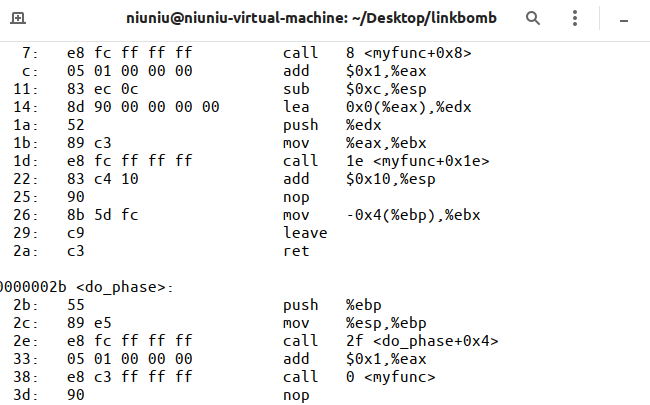


图 3 修改完的phase2.o截图

**3、第3关 有参数的函数调用的机器指令修改**

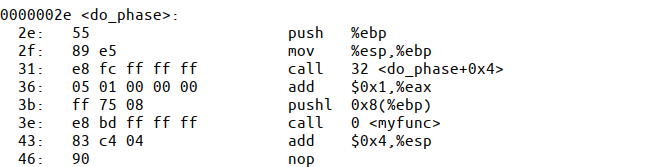


图 4 修改后的phase3.o的截图

**4、第4关 有局部变量的机器指令修改**

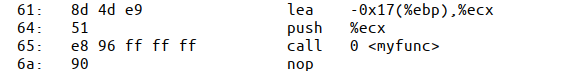


图 5 修改后的phase5.o截图

**5、第5关 重定位表的修改**

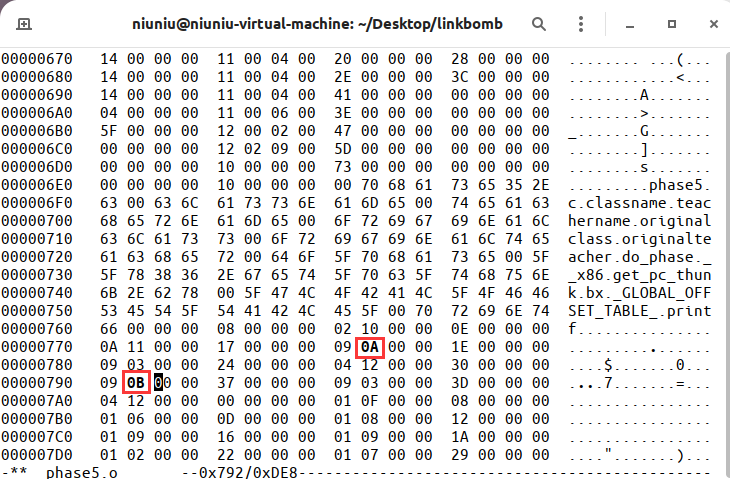


图 6 修改后的phase5.o的截图

**6、第6关 强弱符号**



图 7 编写的phase6\_path.c截图

**7、第7关 只读数据节的修改**

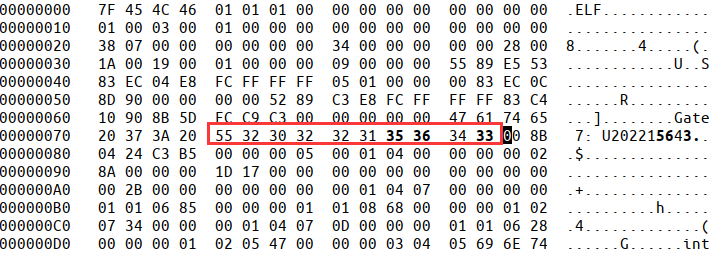
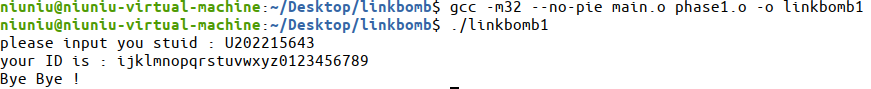


图 8 修改后的phase7.o截图

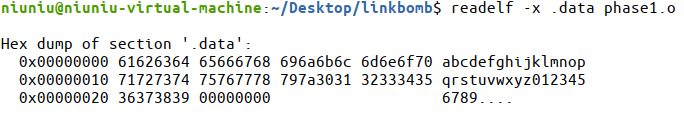
**（2） 描述修改各个文件的基本思想**

**1、第1关 数据节的修改**

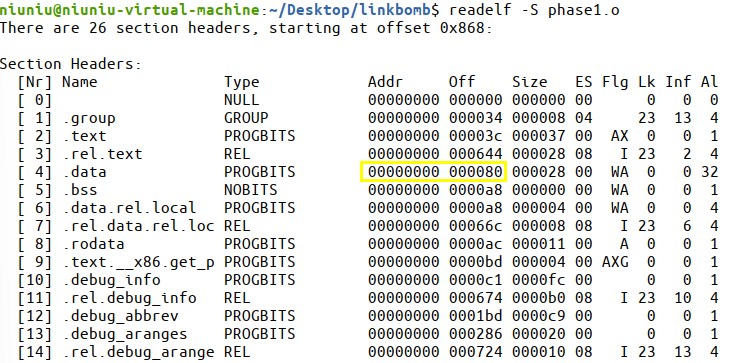
我们先编译main.c并和phase1.o链接一下。



可以看到我们需要修改phase1.o中data节中的内容，让其能够正确输出我们的学号，我们先来查看一下我们的data节中的内容。

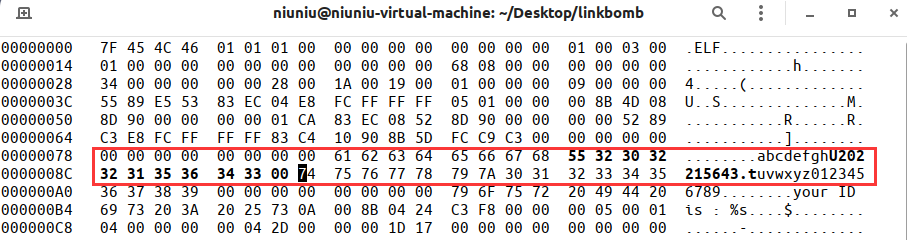


我们可以在这里看到我们刚刚输出的字符串，由此推测这就是我们需要修改的地方。我们还需要指导.data节在phase1.o中的位置



可以看到.data节在phase1.o中偏移量为0x80，但是我们学号不一定就要放在偏移为0x80的位置，我们需要根据我们第一次试探的输出ijklmn等字符的输出判断我们实际上需要修改的数据的位置，提前计算得到学号对应的十六进制表示 55 32 30 32 32 31 35 36 34 33，将指定位置的的数据替换为我们的学号，别忘记了末尾用00填充。

我们输入hexedit phase1.o开始修改。



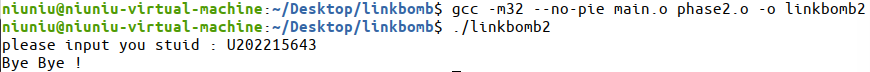
修改完后用ctrl+x退出，我们验证一下是否修改正确。



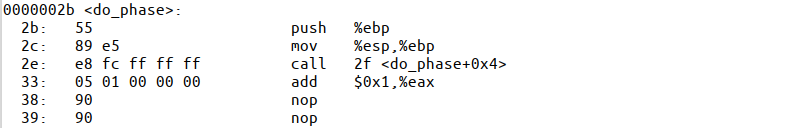
修改正确，完成！

**2、第2关 简单的机器指令修改**

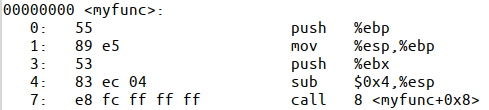
这一阶段的目标是修改phase2.o的.text节的内容，我们先试着去链接一下main.o和phase2.o。



发现啥也没输出。我们直接反汇编一下phase2.o，发现do\_phase函数里面并没有什么具体的实现。



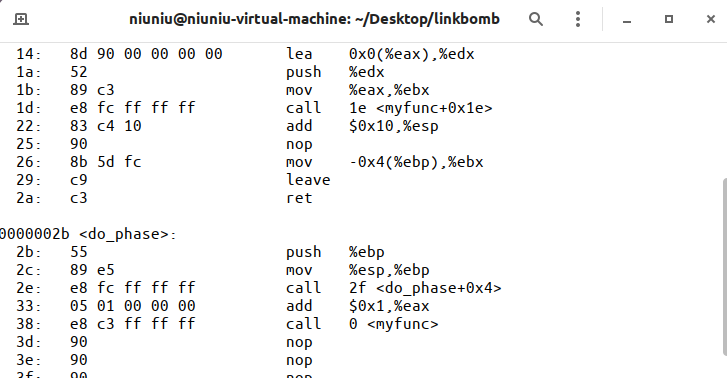
我们关注一下myfunc函数的偏移，发现是0。



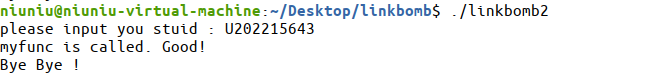
由于我们采用的是PC相对寻址的方式，我们计算新的重定位内容并在add指令后加一条call指令跳转到我们的myfunc函数，转移目标地址=PC+偏移地址，PC为0x3d，目标地址为0，所以重定位的值为0-3d=-3d,转换为十六进制为0xffffffc3。所以我们要去对应的.txt节中修改这个重定位的值。

经过查询得到我们.txt节的偏移为0x3c。我们实际需要修改的位置是0x3c+0x3d处的值。

修改完我们再次反汇编一下phase2.o代码。



我们去验证一下修改是否正确。

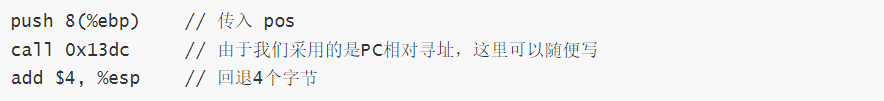


验证正确！

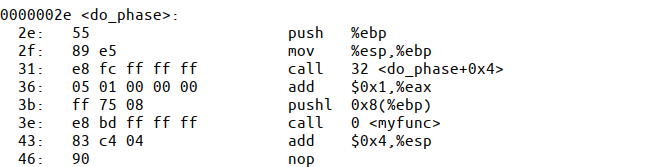
**3、第3关 有参数的函数调用的机器指令修改**

我们先查看一下phase3.o里面.txt节的位置还是在0x3c的位置，这一关我们不仅需要在do\_phase中调用myfunc函数还需要传参，经过查询，得知do\_phase中的参数保存8（%ebp）中，且在myfunc中也是调用8（%ebp）来调用参数。

所以我们的汇编文件可以这么写



编译成机器码放入phase3.o中，如下图：



注意我们call指令中的偏移要修改为-0x43,也就是0xffffffbd,方法和上一关一样。注意，如果汇编代码中没有add $4,%ebp的话会导致堆栈错乱。

最终运行截图如下，正确！



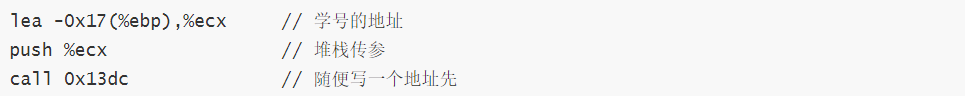
**4、第4关 有局部变量的机器指令修改**

这一关我们需要修改局部变量并把这个局部变量传入myfunc函数，查看do\_phase函数，发现了局部变量存储的是一个字符串，里面是我们的学号，我们需要修改为自己的学号并调用函数输出。

我们先修改我们的学号信息。

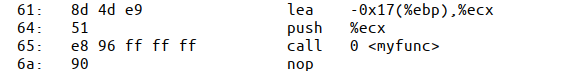


接着我们需要编写我们的汇编代码。



将上面的汇编代码编译成机器代码写入phase4.o中，我们再计算一下重定位的内容为

-6a,十六进制表示为0xffffff96,最终植入的代码如下：

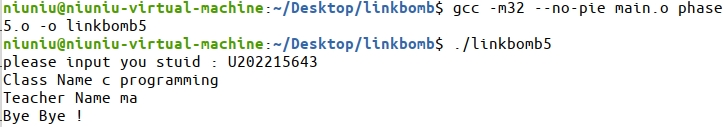


注意这里没有和上一关一样有add $4,%esp,这是因为这一关中do\_phase中末尾有leave指令，而上一关没有。最终运行结果如下，正确！

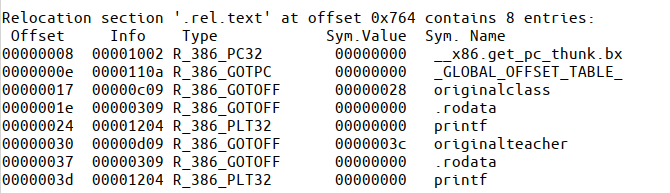


**5、第5关 重定位表的修改**

我们先链接看一下输出是什么，如下图所示，



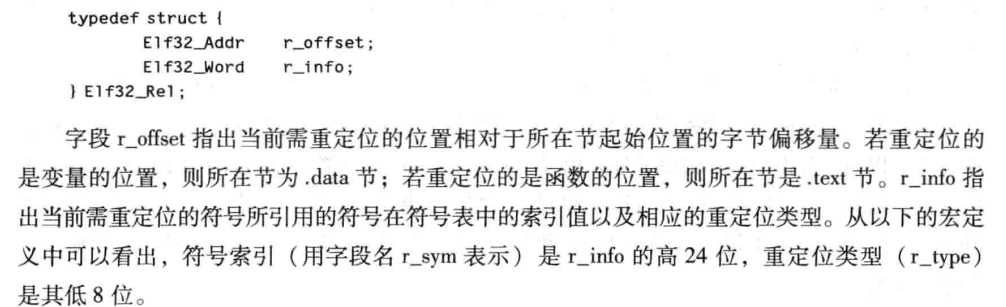
先查看一下重定位节的信息，发现没什么头绪。



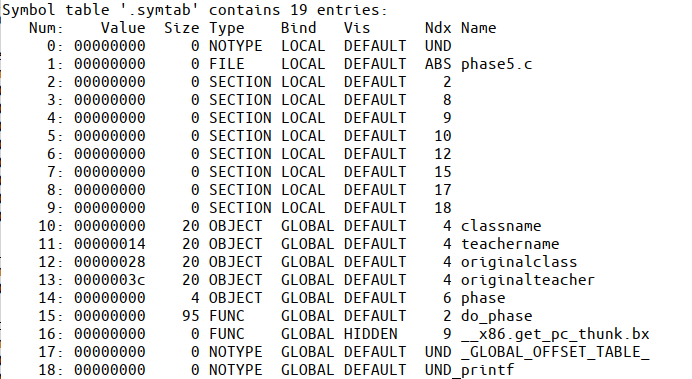
接着利用hexedit输出phase5.o的对应所有的数据，我们在查看对应右侧的常量的值的时候发现了有趣的一件事 ，就是我们的originalteacher和oiginalclass是错误的值，并且刚好是我们刚刚的输出，而teacher和class是我们所需要的答案，那么这个题应该就是让我们修改重定位表中的内容将原本应该定位到originalteacher和originalclass部分的位置定位到我们的teacher和calss中。



那么我们应该怎么去实现修改呢？我们引用书本的一段话，介绍一下这个重定位节中存储的到底是什么信息。



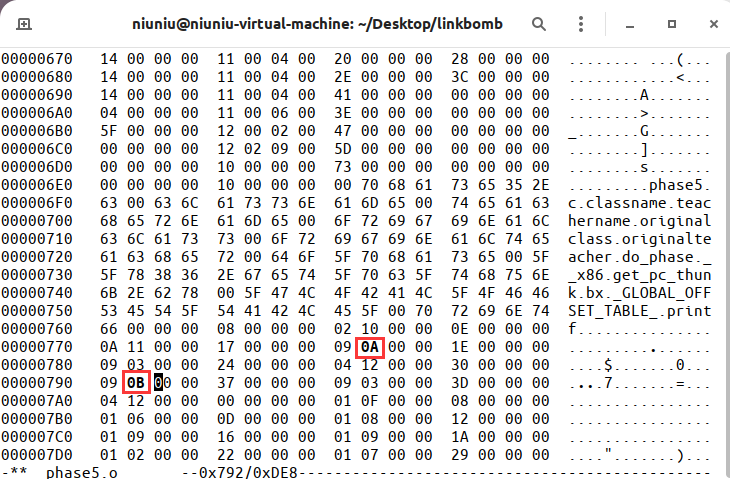
可以知道我们需要重定位的地方的偏移量在offset中，对应的符号编号在info的高24位中，那么符号编号在哪里看呢，是在符号表节中查看。



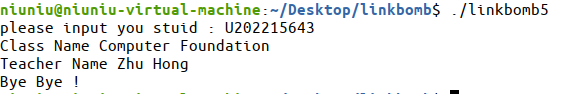
可以看到我们需要的classname和teachernme的编号分别是10和11，对应十六进制就是a和b。接着我们开始去修改。查询得知重定位节的偏移为0x764。



找到正确的位置后我们开始修改成正确的符号编号。



运行截图如下，正确！



**6、第6关 强弱符号**

这一关我们需要自己写一个程序，这一关我注意到标题是强弱符号，那么什么是强弱符号，“函数名和已初始化的全局变量名是强符号，未初始化的全局变量名是弱符号”。我们分别反汇编main.o和phase6.o，查看phase变量和myprint变量。

main.o中的符号表中的phase信息如下，说明phase是一个变量(OBJECT)，且是全局变量(GLOBAL)，且还没有初始化(COM)，这是一个弱符号。



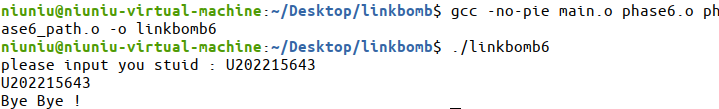
Phase6.o中的符号表中的信息如下图，这里说明phase在phase6.o中初始化，且在这里我们可以看到我们的myprint变量，myprint变量是COM，说明myprint只是在phase6.o中声明但是没有初始化，这是一个弱变量，所以我们可以在phase6\_path.c中定义myprint并且初始化。



我们编写的phase6\_path.c文件如下

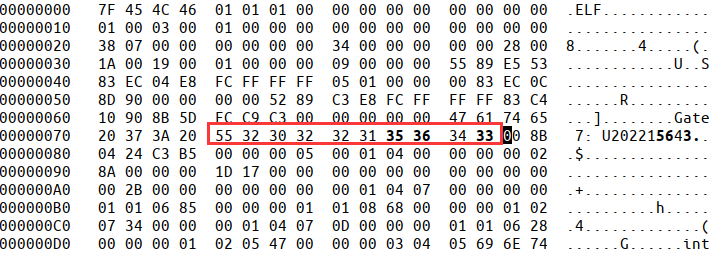


编译成phase6\_path.o后链接，运行结果如下，正确！

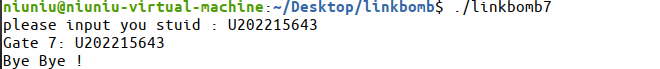


**7、第7关 只读数据节的修改**

有了前几关的基础，第七关就比较简单了。我们先尝试直接输出，发现最后输出的最后五位与我们的学号不同。然后用readelf指令查看我们phase7.o里面各个节对应的位置，找到我们的.rodata只读数据节对应的偏移，再利用hexedit指令在该数据节中依据右侧对应的值找到我们最终输出答案的存放位置，将错误的位改成我们的学号即可。



运行结果如下：



至此全部通关！

**四、体会**

在进行ELF文件与程序链接的实验后，我获得了深刻的见解和宝贵的经验。以下是我对这次实验的心得体会：

首先，我深入了解了ELF文件的结构和原理，这使我对程序的编译和链接过程有了全新的认识。ELF文件作为可执行文件的一种标准格式，其内部复杂的组织结构，包括文件头、节表和段表等，为我揭示了程序如何被计算机所识别和执行。通过亲手操作和分析ELF文件，我能够更加清晰地理解程序的构成和运行机制。

其次，在实验中，我掌握了多种强大的工具来解析和编辑ELF文件，如objdump、readelf和hexedit等。这些工具不仅帮助我直观地查看了ELF文件的汇编代码、头部信息和节表内容，还让我能够亲手尝试修改文件内容，从而更深入地了解程序的内部结构。特别是将源代码编译成可重定位文件（.o文件），并进一步链接成可执行文件的过程，让我对C语言程序的编译链接流程有了更加全面的认识。

同时，我也深刻体会到了修改ELF文件可能带来的风险。在实验过程中，我尝试修改了节表的某些属性或大小，结果导致程序无法正常运行。这让我意识到，在修改ELF文件时需要格外小心，必须确保对修改的后果有充分的认识和准备。这也让我更加明白，在编程和软件开发中，每一个细节都可能影响到程序的稳定性和正确性。

此外，与同学们的交流讨论让我受益匪浅。通过互相学习和分享经验，我们共同解决了实验中遇到的难题，也从彼此的观点中获得了新的启示。这种团队合作和互助精神不仅让我感受到了学习的乐趣，也让我更加珍惜与同学们共度的时光。

总的来说，这次ELF文件与程序链接的实验不仅加深了我对计算机系统基础知识的理解，还提高了我的实践能力和问题解决能力。我相信这些宝贵的经验和收获将对我未来的学习和工作产生深远的影响。