

**课 程 实 验 报 告**

**课程名称： 计算机系统基础实验**

**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 ： cs2208**

**学 号 ： U202215642**

**姓 名 ： 田清林**

**指导教师 ： 班鹏新**

**2024 年 4 月 23 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**

**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。**

**第1关** **静态数据与ELF数据节**

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。

**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!。

**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。

**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。

**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 : Xu Xiangyang。

**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

**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。

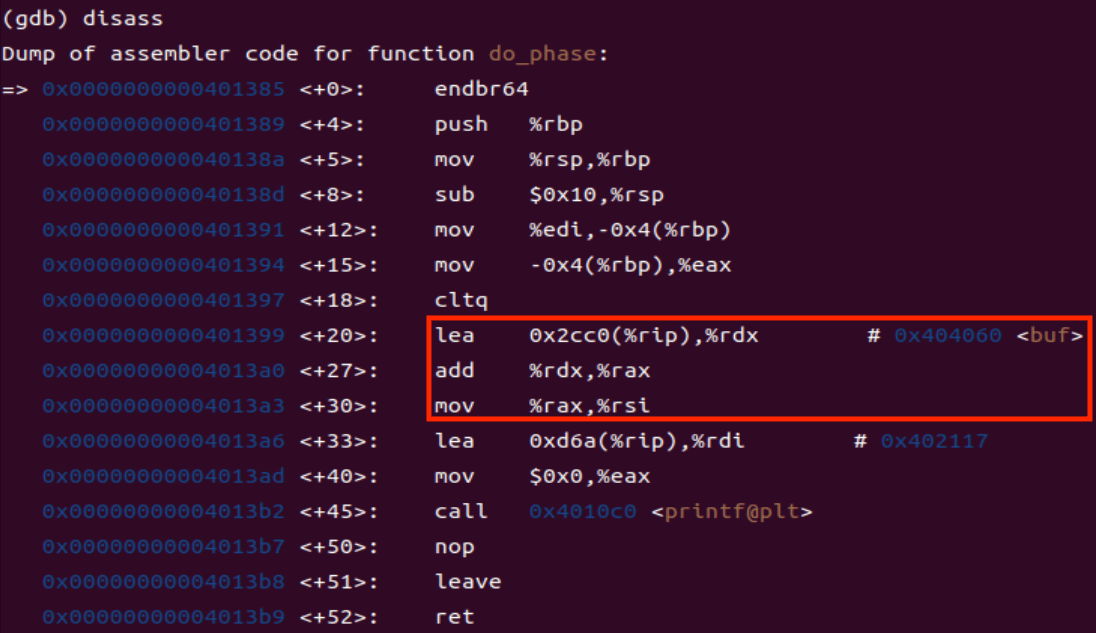
**三、实验记录及问题回答**

**（1）实验任务的实验记录**

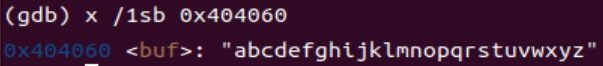
1. **第1关 静态数据与ELF数据节**

实验要求：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。

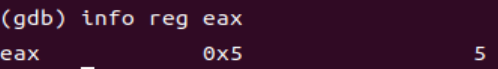
实验思想：找到.data节的位置，修改其值为学号。

 看汇编代码，可以看到传入printf的第二个参数为buf+偏移量rax，buf为全局变量，rax为do\_phase传入的参数。

查看buf位置内容如下：



单步执行至传入的参数赋值给eax，查看eax值。



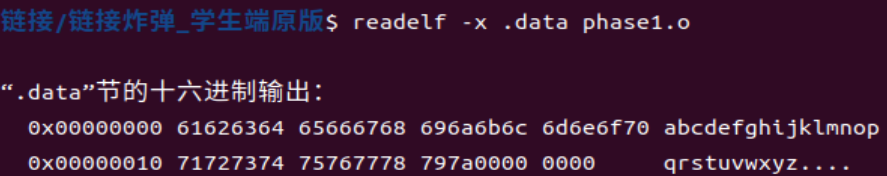
指针的偏移量为5，也就是输出的时候从字符f开始。

只需要将全局变量buf从f开始改为学号的ascii码并在最后加上’\0’作为字符串结束标志，即可输出学号。

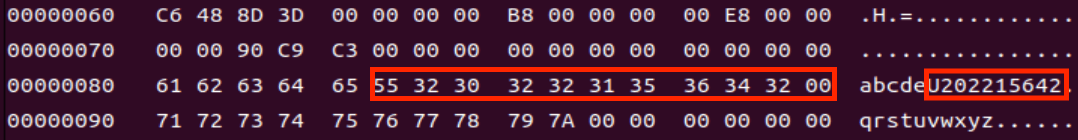
使用readelf，查看data段偏移量。

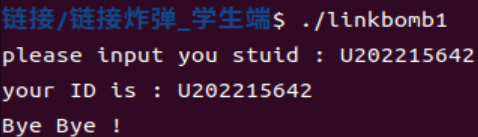


接下来查看.data内容，确定buf内容存储在.data节中。



使用hexedit修改phase1.o对应位置的内容，修改结束内容如下：





通过第一关。

1. **第2关 简单的机器指令修改**

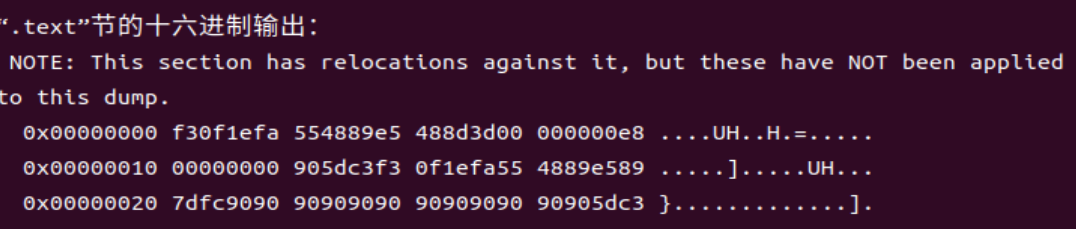
实验要求：修改二进制可重定位目标文件 phase2.o 的代码节中的内容（不允许修改其他节），使其与main.o链接后，生成的执行程序。在phase\_2.c 中，有一个静态函数 static void myfunc( ) ，要求在 do\_phase 函数中调用myfunc( )，显示信息myfunc is called. Good!。

实验：

找到.text节（代码段）位置，如下所示。



查看text节内容，注意每行前面的八位十六进制数是地址不是机器码。



使用objdump对phase2.o进行反汇编，结果如下：

phase2.o： 文件格式 elf64-x86-64

Disassembly of section .text:

0000000000000000 <myfunc>:

0: f3 0f 1e fa endbr64

4: 55 push %rbp

5: 48 89 e5 mov %rsp,%rbp

8: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # f <myfunc+0xf>

f: e8 00 00 00 00 call 14 <myfunc+0x14>

14: 90 nop

15: 5d pop %rbp

16: c3 ret

0000000000000017 <do\_phase>:

17: f3 0f 1e fa endbr64

1b: 55 push %rbp

1c: 48 89 e5 mov %rsp,%rbp

1f: 89 7d fc mov %edi,-0x4(%rbp)

22: 90 nop

23: 90 nop

24: 90 nop

25: 90 nop

26: 90 nop

27: 90 nop

28: 90 nop

29: 90 nop

2a: 90 nop

2b: 90 nop

2c: 90 nop

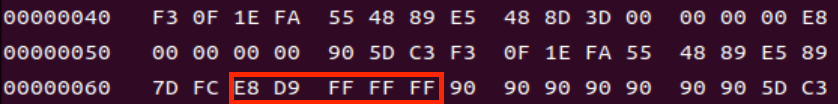
2d: 90 nop

2e: 5d pop %rbp

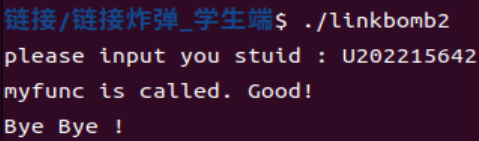
2f: c3 ret

可以看到myfunc和do\_phase函数，要求do\_phase函数跳转到myfunc函数，可以在do\_phase中将nop改为call指令，使用相对寻址，call机器码为e8，之后四位为小端法存放的四字节有符号数，需要跳转到myfunc第一条指令即0x0位置，计算得到偏移量为-0x27，转换为补码为0xFFFFFFD9。则机器码为E8 D9 FF FF FF。

修改phase\_2.o的值如下所示：



通过。



1. **第3关 有参数的函数调用的机器指令修改**

实验要求：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。

这题与上一关差异仅在于多了个参数，在64位系统中函数的第一个由rdi传递，只需在调用前将rdi设置为参数即可。

找到.text节（代码段）位置，如下所示。



反汇编代码如下：

phase3.o： 文件格式 elf64-x86-64

Disassembly of section .text:

0000000000000000 <myfunc>:

0: f3 0f 1e fa endbr64

4: 55 push %rbp

5: 48 89 e5 mov %rsp,%rbp

8: 48 83 ec 10 sub $0x10,%rsp

c: 89 7d fc mov %edi,-0x4(%rbp)

f: 8b 45 fc mov -0x4(%rbp),%eax

12: 89 c6 mov %eax,%esi

14: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # 1b <myfunc+0x1b>

1b: b8 00 00 00 00 mov $0x0,%eax

20: e8 00 00 00 00 call 25 <myfunc+0x25>

25: 90 nop

26: c9 leave

27: c3 ret

0000000000000028 <do\_phase>:

28: f3 0f 1e fa endbr64

2c: 55 push %rbp

2d: 48 89 e5 mov %rsp,%rbp

30: 89 7d fc mov %edi,-0x4(%rbp)

33: 90 nop

34: 90 nop

35: 90 nop

36: 90 nop

37: 90 nop

38: 90 nop

39: 90 nop

3a: 90 nop

3b: 90 nop

3c: 90 nop

3d: 90 nop

3e: 90 nop

3f: 90 nop

40: 90 nop

41: 90 nop

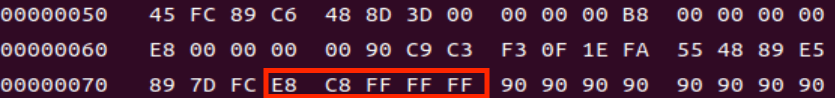
42: 5d pop %rbp

43: c3 ret

可以看到在do\_phase中edi除了传参后就没有再使用过，因此可以直接调用myfunc函数，此时的edi值刚好是do\_phase的参数，也就是应该传入的参数。

同样使用call指令并相对寻址，计算出偏移量为-38，机器码为E8 C8 FF FF FF。

修改phase\_3.o的值如下所示：



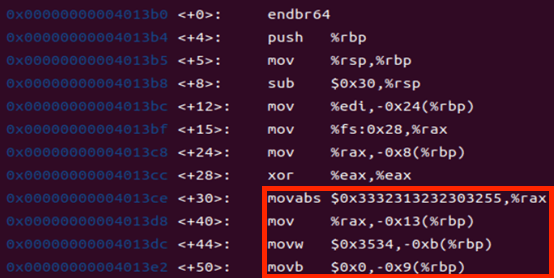
通过第三关。

1. **第4关 有局部变量的机器指令修改**

实验要求：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。

此处需要将一个局部变量作为myfunc的参数，因此首先要找到局部变量，之后在调用前将局部变量放在rdi中。

do\_phase汇编代码如下：



将一个8字节的立即数放入rax之后放入-0x13(%rbp),之后再在-0xb(%rbp)放入两字节数，最后在-0x9(%rbp)处放一个0。猜测存在一个局部变量char数组，起始位置为-0x13(%rbp)。

查看-0x13(%rbp)，内容如下：



因此只需要将-0x13(%rbp)放入rdi之后调用代码即可。

产生代码如下：

0: 48 8d 7d ed lea -0x13(%rbp),%rdi

将phase4.o反汇编，部分代码如下：

57: 66 c7 45 f5 34 35 movw $0x3534,-0xb(%rbp)

5d: c6 45 f7 00 movb $0x0,-0x9(%rbp)

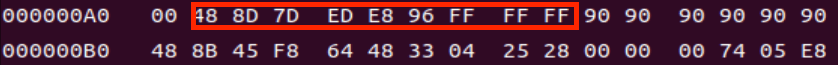
61: 90 nop

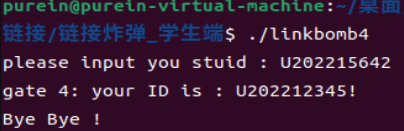
从61处开始修改机器代码，分别写入lea和call的代码，长度分别为4字节和5字节，call下一条指令的地址为，myfunc位置为0，偏移量为-0x6a，因此call指令机器码为E8 96 FF FF FF。

找到.text节（代码段）位置，如下所示。



修改phase4.o如下：





1. **第5关 重定位表的修改**

实验要求：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 : Xu Xiangyang。

重定位条目结构如下：

typedef struct {

long offset; /\* Offset of the reference to relocate \*/

long type:32, /\* Relocation type \*/

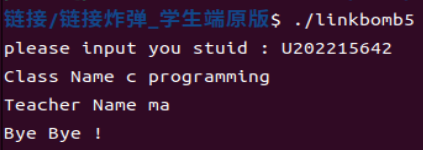
symbol:32; /\* Symbol table index \*/

long addend; /\* Constant part of relocation expression \*/

} Elf64\_Rela;

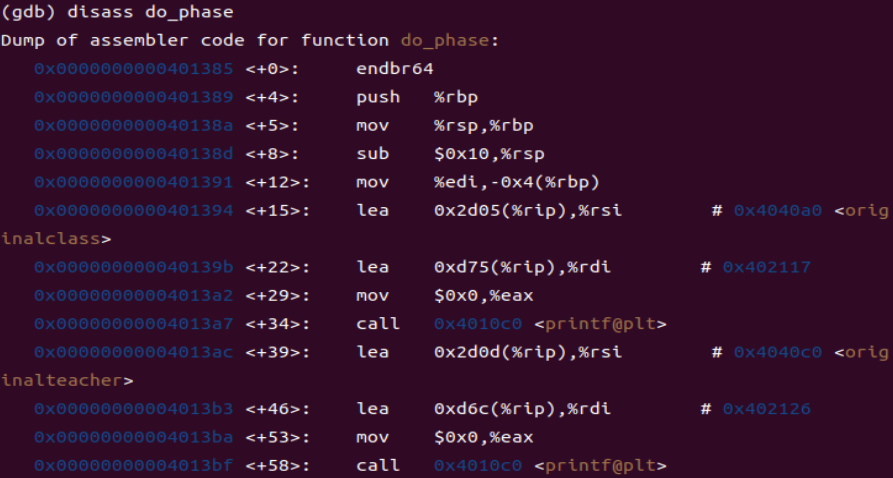
通过改变重定位条目中的符号表索引，可以使其重定位后引用不同的变量，从而改变程序的输出。

先不做更改尝试运行。



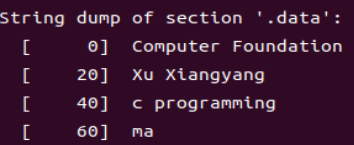
发现Class Name 和Teacher Name后的字符串内容不合要求。

查看do\_phase汇编代码：



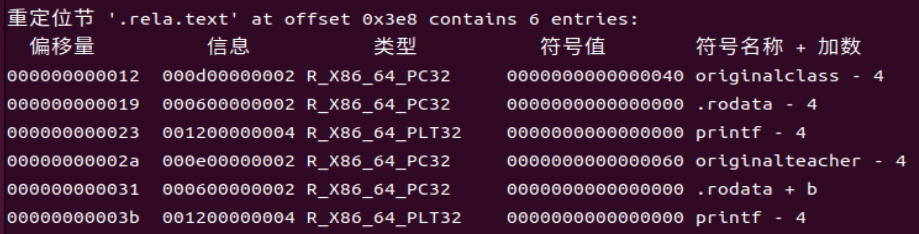
发现它用originalclass和originalteacher分别调用printf函数进行输出。

查看.data内容：



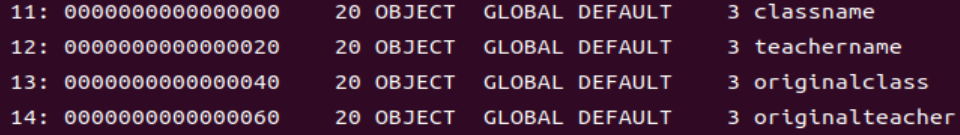
发现错误的输出和目标输出，说明需要输出的字符串都是已初始化的全局变量。

查看重定位节内容：



可以看到重定位条目按偏移量升序排列，第一条和第四条分别为originalclass和originalteacher，正是需要修改的部分。

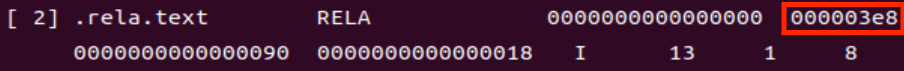
查看符号表内容：



可以看到classname和teachername的符号表索引，分别为11(0xb)，12(0xc)，originalclass和originalteacher的符号表索引分别为13(0xd)，14(0xe)。

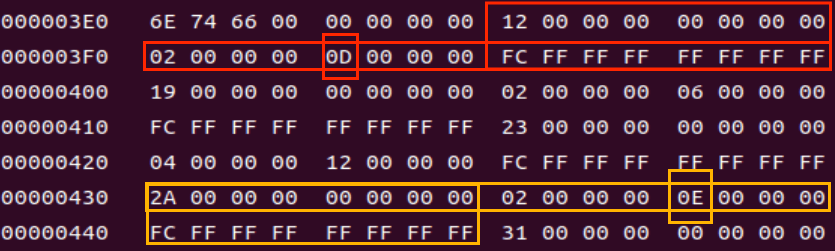
分析结构可知，只需将originalclass和originalteacher的重定位条目中符号表索引改为对应的classname和teachername的符号表索引，即可使其输出classname和teachername。

查看.rela.text位置如下：

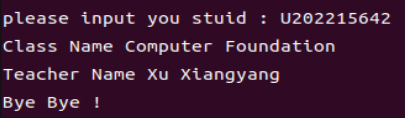


可知偏移量为0x3e8。

使用hexedit通过偏移量找到重定位表的位置，按照顺序找到originalclass 和originalteacher 的重定位条目位置，分别使用红色和黄色框出，如下图所示：



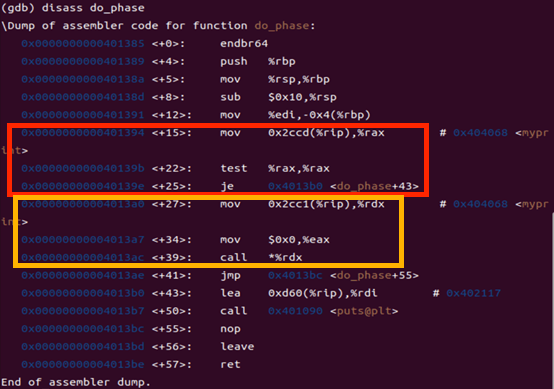
额外框出的是存放符号表索引的位置，需要对应改变为0B(11)，0C(12)。



1. **第6关 强弱符号**

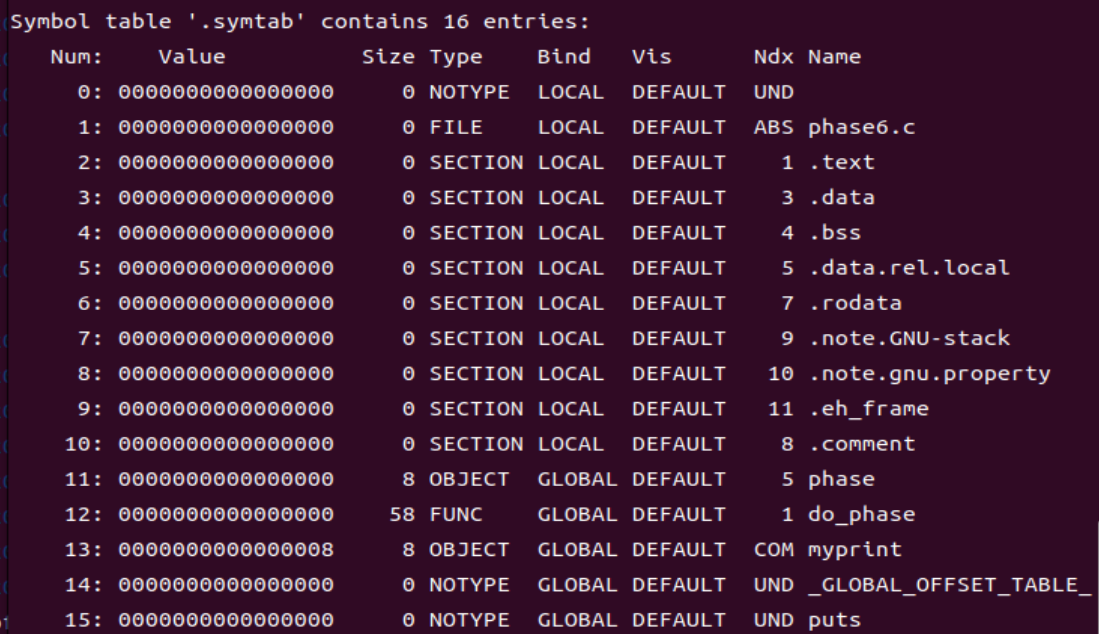
实验要求：不准修改 main.c 和phase6.o，通过增补一个文件，使得程序链接后，能够输出自己的学号。

查看do\_phase的汇编代码：



红色方框中汇编代码检查myprint是否为0，黄色方框将myprint放进rdx，同时call \*rdx，由此可看出myprint是一个函数指针。

查看符号表：



myprint是COMMON，属于弱符号，可用强符号覆盖。

编写phase6\_patch.c如下所示：

#include <stdio.h>

void print\_U202215642()

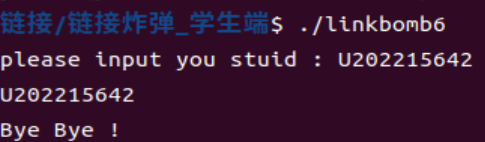
{

printf("U202215642\n");

}

void (\*myprint) = print\_U202215642;

使用gcc产生phase6\_patch.o并与main.o和phase6.o进行链接。运行

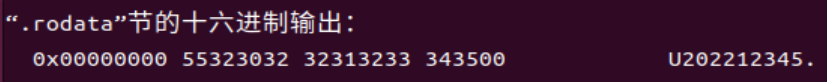


通过第六关。

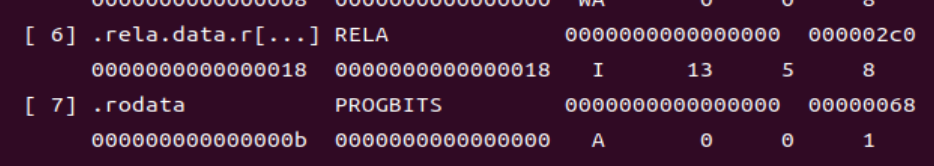
1. **第7关 只读数据节的修改**

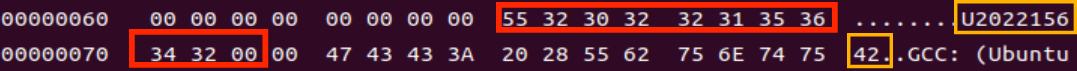
实验要求：修改 phase7.o 中只读数据节（不准修改代码节），使其与main.o链接后，能够输出自己的学号。

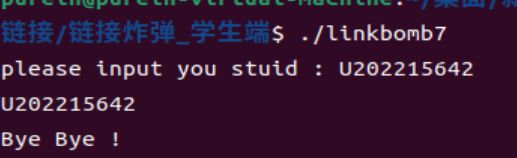
查看.rodata内容，如下所示：



查看.rodata位置：



使用hexedit修改位置，修改后如下图所示：



通过第七关。

**（2）描述修改各个文件的基本思想**

1. **第1关 静态数据与ELF数据节**

分析汇编代码，发现输出的字符串为已初始化的全局变量buf数组首地址加上某偏移量为起始的字符串，查看偏移量，在elf文件中找到.data节的位置，修改其值为学号。

1. **第2关 简单的机器指令修改**

调用myfunc函数只需要插入一条合适的call指令即可分析汇编代码，修改代码节内容，将nop替换为调用myfunc的call指令，采用PC相对寻址，根据相对位置计算出相对寻址的值。

1. **第3关 有参数的函数调用的机器指令修改**

与上一关相比多了一个参数，实验环境为64位linux系统，当只有一个参数时借助寄存器rdi来传递。只要在调用前把需要的参数提前赋值给rdi即可，call指令仍需要PC相对寻址。依此修改代码节内容。

1. **第4关 有局部变量的机器指令修改**

与上一关相比参数成了一个局部变量，分析汇编代码，将局部变量提前放入rdi即可， call指令仍需要PC相对寻址。依此修改代码节内容。

1. **第5关 重定位表的修改**

重定位过程中，链接器会修改代码节和数据节中对每个符号的引用，使得他们指向正确的运行时地址。修改重定位表可以改变引用应该指向的符号从而改变程序的输出。

ELF重定位节结构如下：

typedef struct {

long offset; /\* Offset of the reference to relocate \*/

long type:32, /\* Relocation type \*/

symbol:32; /\* Symbol table index \*/

long addend; /\* Constant part of relocation expression \*/

} Elf64\_Rela;

查看Symbol table，了解各符号在符号表的索引，改变符号表索引，使其指向另一个正确的符号。

1. **第6关 强弱符号**

符号分为强符号和弱符号。函数和已初始化的全局变量是强符号，未初始化的全局变量是弱符号。查看汇编代码，发现使用到一个名为“myprint”的弱符号，可以通过自己编写程序定义一个同名的强符号从而覆盖掉这个弱符号。写一个函数并定义函数指针myprint指向它，编译为可重定位目标文件后与其他目标文件进行链接，便可以成功替换掉myprint。

1. **第7关 只读数据节的修改**

符号本关与第一关类似，只是修改对象从.data变成了.rodata， 找到输出字符串的位置并改为学号即可。

**四、体会**

**本次实验加深了我对程序链接方面知识的理解。**

**整体实验的过程较为顺利。刚开始用hexedit打开可重定位目标文件时，看到漫长的二进制串我无从下手，只能根据课上讲到的ELF文件的结构猜测各个位置大概是什么内容。但后来在熟悉新的工具readelf后，与hexedit对照着看，既能看到ELF的结构和各部分的解释，也能看到底层的二进制表示，实验也就几乎没有再遇到什么困难，轻松解决。当然这也离不开我对理论知识的掌握。**

**程序的链接是一个复杂的过程，链接器通过将多个可重定位目标文件进行符号解析和重定位等操作，输出一个完全链接的、可加载和运行的可执行目标文件。通过了解链接过程，我理解了语言的作用域规则是如何实现的，了解静态库、共享库的原理，这可以帮助我避免一些隐蔽的编程错误，为将来构建大型程序奠定了理论基础，**