華中科技大學

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：CS2209

学　　号：U202210755

姓　　名：章宏宇

指导教师：李专

2024 年 4 月 28 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. phase1

考虑到任务描述“只能修改数据节”，所以先尝试执行程序：

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb$ gcc -no-pie -o linkbomb1 main.c phase1.
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb$ ./linkbomb1
please input you stuid : U202210755
your ID is : ijklmnopqrstuvwxyz
Bye Bye !
```

发现输出十分规则，在数据节中应该存在字母表，故直接用 hexedit 打开 phase1.o：

```
00000078   00 00 00 00  00 00 00 00  61 62 63 64  65 66 67 68  69 6A 6B 6C  6D 6E 6F 70  ........abcdefghijklmnop
00000090   71 72 73 74  75 76 77 78  79 7A 00 00  00 00 00 00  00 00 00 00  00 00 00 00  qrstuvwxyz..............
```

故直接从 i 开始修改为学号即可。通关结果如下

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb_final$ ./linkbomb1
please input you stuid : U202210755
your ID is : U202210755
Bye Bye !
```

2. phase2

首先先用 objdump -d phase2.o 来查看汇编代码

```
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
```

发现出题人已经留好一段空指令来方便修改了。故联想到上课所讲的 PC32 重定位方法，只需要插入一条 call <myfunc> 指令即可。偏移地址为下一条 RIP 与<myfunc+0>的差，即修改为如图：

```
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:   e8 d9 ff ff ff          call   0 <myfunc>
  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
```

通关结果如下：

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb_final$ ./linkbomb2
please input you stuid : U202210755
myfunc is called. Good!
Bye Bye !
```

3. phase3

首先查看 phase3.o 的汇编代码：

```
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
```

发现%edi 已经是 pos 了，于是仿照 phase2 直接 call myfunc 即可。汇编代码修改如下：

```
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:   e8 c8 ff ff ff          call   0 <myfunc>
  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
```

通关结果如下：

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb_final$ ./linkbomb3
please input you stuid : U202210755
gate 3: offset is : 8!
Bye Bye !
```

4. phase4

首先思考如何找到对应字符串指针，发现在 main 函数中有 stuid

```
int main(int argc, const char *argv[])
{
    int     cookie;
    char  stuid[12];
    printf("please input you stuid : ");
    scanf("%s",stuid);
    cookie = gencookie(stuid);

    if (phase)
        (*phase)(cookie);
    else  {
        printf("Welcome to linkbomb \n");
        printf("You should modify phase1.o,  phase2.o ....\n");
        printf("execute :  gcc -no-pie -o linkbomb1 main.o phase1.o \n");
        printf("execute :  ./linkbomb1 \n");
   }
   printf("Bye Bye !\n");
   return 0;
}
```

于是思路显而易见，找到 stuid 与 do_phase 的 rbp 的偏移量，修改 edi 为 stuid 的值即可。于是首先进入 gdb，在 scanf 那句打上断点，查看 rbp 和传参寄存器的值：

```
(gdb) i reg rbp
rbp            0x7fffffffe0e0      0x7fffffffe0e0
(gdb) i reg rsi
rsi            0x7fffffffe0cc      140737488347340
```

再进入 do_phase，查看 rbp 的值：

```
(gdb) continue
Continuing.

Breakpoint 4, 0x00000000004013e6 in do_phase ()
(gdb) i reg rbp
rbp            0x7fffffffe0a0      0x7fffffffe0a0
```

于是可得偏移量为 0x2c，于是构造攻击指令如下：

```
0000000000000000 <.text>:
   0:   48 89 ef                mov    %rbp,%rdi
   3:   48 83 c7 2c             add    $0x2c,%rdi
```

之后再仿照 phase2，call myfunc 即可。通关结果如下：

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb_final$ ./linkbomb4
please input you stuid : U202210755
gate 4: your ID is : U202210755!
Bye Bye !
```

5. phase5

首先，先用 readelf -r phase5.o 来看重定位节中的内容：

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb$ readelf -r phase5.o

Relocation section '.rela.text' at offset 0x3e8 contains 6 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000000012  000d00000002 R_X86_64_PC32     0000000000000040 originalclass - 4
000000000019  000600000002 R_X86_64_PC32     0000000000000000 .rodata - 4
000000000023  001200000004 R_X86_64_PLT32    0000000000000000 printf - 4
00000000002a  000e00000002 R_X86_64_PC32     0000000000000060 originalteacher - 4
000000000031  000600000002 R_X86_64_PC32     0000000000000000 .rodata + b
00000000003b  001200000004 R_X86_64_PLT32    0000000000000000 printf - 4

Relocation section '.rela.data.rel.local' at offset 0x478 contains 1 entry:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000000000  001000000001 R_X86_64_64       0000000000000000 do_phase + 0

Relocation section '.rela.eh_frame' at offset 0x490 contains 1 entry:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000000020  000200000002 R_X86_64_PC32     0000000000000000 .text + 0
```

可以发现出现了 originalclass 和 originalteacher。但很明显，在重定位节中不可能存下完整的 name，应该是存下了对应在符号表中的索引。查看符号表可知：

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb$ readelf -s phase5.o

Symbol table '.symtab' contains 19 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS phase5.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .data.rel.local
     6: 0000000000000000     0 SECTION LOCAL  DEFAULT    7 .rodata
     7: 0000000000000000     0 SECTION LOCAL  DEFAULT    9 .note.GNU-stack
     8: 0000000000000000     0 SECTION LOCAL  DEFAULT   10 .note.gnu.property
     9: 0000000000000000     0 SECTION LOCAL  DEFAULT   11 .eh_frame
    10: 0000000000000000     0 SECTION LOCAL  DEFAULT    8 .comment
    11: 0000000000000000    20 OBJECT  GLOBAL DEFAULT    3 classname
    12: 0000000000000020    20 OBJECT  GLOBAL DEFAULT    3 teachername
    13: 0000000000000040    20 OBJECT  GLOBAL DEFAULT    3 originalclass
    14: 0000000000000060    20 OBJECT  GLOBAL DEFAULT    3 originalteacher
    15: 0000000000000000     8 OBJECT  GLOBAL DEFAULT    5 phase
    16: 0000000000000000    80 FUNC    GLOBAL DEFAULT    1 do_phase
    17: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND _GLOBAL_OFFSET_TABLE_
    18: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
```

Originalclass 的索引是 0xd，originalteacher 的索引是 0xe，接着我们用十六进制查看重定位节：

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb$ readelf -x .rela.text phase5.o

Hex dump of section '.rela.text':
  0x00000000 12000000 00000000 02000000 0d000000 ................
  0x00000010 fcffffff ffffffff 19000000 00000000 ................
  0x00000020 02000000 06000000 fcffffff ffffffff ................
  0x00000030 23000000 00000000 04000000 12000000 #...............
  0x00000040 fcffffff ffffffff 2a000000 00000000 ........*.......
  0x00000050 02000000 0e000000 fcffffff ffffffff ................
  0x00000060 31000000 00000000 02000000 06000000 1...............
  0x00000070 0b000000 00000000 3b000000 00000000 ........;.......
  0x00000080 04000000 12000000 fcffffff ffffffff ................
```

可以发现，出现了 0d 和 0e。故大胆猜测，只需要把原来的索引改成 classname 和 teachername 的索引即可。最后成功通关，结果如下：

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb_final$ ./linkbomb5
please input you stuid : U202210755
Class Name Computer Foundation
Teacher Name Xu Xiangyang
Bye Bye !
```

6. phase6

首先，用 objdump -d -r phase6.o 来查看汇编代码和重定位信息：

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb$ objdump -d  -r phase6.o

phase6.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <do_phase>:
   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:   48 8b 05 00 00 00 00    mov    0x0(%rip),%rax        # 16 <do_phase+0x16>
                        12: R_X86_64_PC32       myprint-0x4
  16:   48 85 c0                test   %rax,%rax
  19:   74 10                   je     2b <do_phase+0x2b>
  1b:   48 8b 15 00 00 00 00    mov    0x0(%rip),%rdx        # 22 <do_phase+0x22>
                        1e: R_X86_64_PC32       myprint-0x4
  22:   b8 00 00 00 00          mov    $0x0,%eax
  27:   ff d2                   call   *%rdx
  29:   eb 0c                   jmp    37 <do_phase+0x37>
  2b:   48 8d 3d 00 00 00 00    lea    0x0(%rip),%rdi        # 32 <do_phase+0x32>
                        2e: R_X86_64_PC32       .rodata-0x4
  32:   e8 00 00 00 00          call   37 <do_phase+0x37>
                        33: R_X86_64_PLT32      puts-0x4
  37:   90                      nop
  38:   c9                      leave
  39:   c3                      ret
```

发现出现了可重定位的 myprint，于是想到直接实现一个函数

```c
#include<stdio.h>
void myprint(){
    printf("U202210755\n");
}
```

结果一编译，发现 warning 了：

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb$ gcc -no-pie -o linkbomb6 main.c phase6.o phase6_patch.c
/usr/bin/ld: warning: alignment 1 of symbol `myprint' in /tmp/ccrK88fw.o is smaller than 8 in phase6.o
/usr/bin/ld: warning: size of symbol `myprint' changed from 8 in phase6.o to 26 in /tmp/ccrK88fw.o
/usr/bin/ld: warning: type of symbol `myprint' changed from 1 to 2 in /tmp/ccrK88fw.o
```

这可不太对，于是我查看了一下 phase6 的符号表

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb$ readelf -s phase6.o

Symbol table '.symtab' contains 16 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS phase6.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .data.rel.local
     6: 0000000000000000     0 SECTION LOCAL  DEFAULT    7 .rodata
     7: 0000000000000000     0 SECTION LOCAL  DEFAULT    9 .note.GNU-stack
     8: 0000000000000000     0 SECTION LOCAL  DEFAULT   10 .note.gnu.property
     9: 0000000000000000     0 SECTION LOCAL  DEFAULT   11 .eh_frame
    10: 0000000000000000     0 SECTION LOCAL  DEFAULT    8 .comment
    11: 0000000000000000     8 OBJECT  GLOBAL DEFAULT    5 phase
    12: 0000000000000000    58 FUNC    GLOBAL DEFAULT    1 do_phase
    13: 0000000000000008     8 OBJECT  GLOBAL DEFAULT  COM myprint
    14: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND _GLOBAL_OFFSET_TABLE_
    15: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND puts
```

仔细一看，发现 myprint 是一个变量。那就说明 myprint 应该存的是 realprint 的地址，于是修改 phase6_patch.c 如下：

```c
#include<stdio.h>
long long myprint = 0;
void realprint(){
    printf("U202210755\n");
}
```

然后对 linkbomb6 查看 realprint 对应的汇编代码来找到首地址：

```
00000000004013bf <realprint>:
  4013bf:       f3 0f 1e fa             endbr64
  4013c3:       55                      push   %rbp
  4013c4:       48 89 e5                mov    %rsp,%rbp
  4013c7:       48 8d 05 66 0d 00 00    lea    0xd66(%rip),%rax        # 402134 <_IO_stdin_used+0x134>
  4013ce:       48 89 c7                mov    %rax,%rdi
  4013d1:       e8 ba fc ff ff          call   401090 <puts@plt>
  4013d6:       90                      nop
  4013d7:       5d                      pop    %rbp
  4013d8:       c3                      ret
```

修改 myprint 为对应首地址

```c
#include<stdio.h>
long long myprint = 0x4013bf;
void realprint(){
    printf("U202210755\n");
}
```

轻松通过：

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb$ ./linkbomb6
please input you stuid : U202210755
U202210755
Bye Bye !
```

7. phase7

首先先编译执行一遍，看看结果：

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb$ gcc -no-pie -o linkbomb7 main.c phase7.o
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb$ ./linkbomb7
please input you stuid : U202210755
U202212345
Bye Bye !
```

发现输出了一个字符串，结合任务要求，充分怀疑这个字符串就放在.data 节。先查看 phase7.o 的节头：

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb$ readelf -S phase7.o
There are 16 section headers, starting at offset 0x380:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS         0000000000000000  00000040
       000000000000001e  0000000000000000  AX       0     0     1
  [ 2] .rela.text        RELA             0000000000000000  00000290
       0000000000000030  0000000000000018   I      13     1     8
  [ 3] .data             PROGBITS         0000000000000000  0000005e
       0000000000000000  0000000000000000  WA       0     0     1
  [ 4] .bss              NOBITS           0000000000000000  0000005e
       0000000000000000  0000000000000000  WA       0     0     1
  [ 5] .data.rel.local   PROGBITS         0000000000000000  00000060
       0000000000000008  0000000000000000  WA       0     0     8
  [ 6] .rela.data.r[...] RELA             0000000000000000  000002c0
       0000000000000018  0000000000000018   I      13     5     8
  [ 7] .rodata           PROGBITS         0000000000000000  00000068
       000000000000000b  0000000000000000   A       0     0     1
  [ 8] .comment          PROGBITS         0000000000000000  00000073
       000000000000002c  0000000000000001  MS       0     0     1
  [ 9] .note.GNU-stack   PROGBITS         0000000000000000  0000009f
       0000000000000000  0000000000000000           0     0     1
  [10] .note.gnu.pr[...] NOTE             0000000000000000  000000a0
       0000000000000020  0000000000000000   A       0     0     8
  [11] .eh_frame         PROGBITS         0000000000000000  000000c0
       0000000000000038  0000000000000000   A       0     0     8
  [12] .rela.eh_frame    RELA             0000000000000000  000002d8
       0000000000000018  0000000000000018   I      13    11     8
  [13] .symtab           SYMTAB           0000000000000000  000000f8
       0000000000000168  0000000000000018          14    11     8
  [14] .strtab           STRTAB           0000000000000000  00000260
       000000000000002e  0000000000000000           0     0     1
  [15] .shstrtab         STRTAB           0000000000000000  000002f0
       0000000000000089  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  D (mbind), l (large), p (processor specific)
```

再用 hexedit 打开 phase7.o：

```
00000030   00 00 00 00  40 00 00 00  00 00 40 00  10 00 0F 00  F3 0F 1E FA  55 48 89 E5  ....@.....@.........UH..
00000048   48 83 EC 10  89 7D FC 48  8D 3D 00 00  00 00 E8 00  00 00 00 90  C9 C3 00 00  H....}.H.=..............
00000060   00 00 00 00  00 00 00 00  55 32 30 32  32 31 32 33  34 35 00 00  47 43 43 3A  ........U202212345..GCC:
00000078   20 28 55 62  75 6E 74 75  20 39 2E 34  2E 30 2D 31  75 62 75 6E  74 75 31 7E   (Ubuntu 9.4.0-1ubuntu1~
```

发现“U202212345”在 0x68 处，刚好是.rodata 的开头，于是直接修改只读数据节即可，通关结果如下：

```
mystle@Lenovo16plus:~/csappexp/lab4/linkbomb_final$ ./linkbomb7
please input you stuid : U202210755
U202210755
Bye Bye !
```

（2）描述修改各个文件的基本思想

基本思路大多很统一：

1. 先编译并执行一遍，看看修改前的输出结果如何。
2. 如果有输出，就去找到输出对应的位置，直接修改
3. 如果没有输出，或者不能修改数据节，那就得结合手写汇编和重定位的相关知识，来实现修改。

四、体会

这次实验相比前几次简单太多，我也就周二晚上加上周三早上的一节课就做完了（这才正常吧）。主要是之前的实验摸爬滚打后，对于汇编的相关知识掌握地还不错，加上上课还算认真听了一会，对于 elf 文件的相关节有一定认识，不至于两眼一抹黑，连查资料都不知道翻哪一页 ppt。

但课上老师讲得挺快，来不及实操和练习，大部分都左耳进右耳出。我还真以为自己懂了，但直到做实验时才发觉一窍不通。很多都是找规律，采用了一些奇技淫巧过的（。

做完实验后，我对于 call/jmp 的相对地址、重定位的流程有了一个大致的认识，收获颇丰。也复习了一下 lab3 的相关知识，手写汇编更加熟练了（。

总而言之，这次的实验体验良好，题目质量很高，解题思路巧妙，下次还来！