

华中科技大学

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 2202

学 号： U202215378

姓 名： 冯瑞琦

指导教师： 王多强

2024 年 6 月 7 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

(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。

输入 `readelf -a phase1.o` 查看 ELF 文件内容，找到输出的 .data 节中偏移量为 32 的

Section Headers:										
[Nr]	Name	Type	Addr	Off	Size	ES	Flg	Lk	Inf	Al
Terminal		NULL	00000000	000000	000000	00		0	0	0
[1]	.group	GROUP	00000000	000034	000008	04		23	13	4
[2]	.text	PROGBITS	00000000	00003c	000037	00	AX	0	0	1
[3]	.rel.text	REL	00000000	000644	000028	08	I 23	2	4	
[4]	.data	PROGBITS	00000000	000080	000028	00	WA	0	0	32

位置。

编译链接原来的 phase1.o 文件，运行生成的 linkbomb1 文件，观察输出的内容如下图。

使用 hexedit 打开 phase1.o，找到 data 节数据，如下图。

从运行 linkbomb1 观察到的 ID 的内容改成学号 U202215383，最后重新编译链接生成新的 linkbomb1，再次运行，得到的输出为自己的学号，如下图，完成任务。

```
00000080 | 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 30 31 32 33 34 35 |qrstuvwxyz012345
000000A0 | 36 37 38 39 00 00 00 00 00 00 00 00 00 79 6F 75 72 |6789          your
000000B0 | 20 49 44 20 69 73 20 3A 20 25 73 0A 00 8B 04 24 | ID is : %s < $
00000080 | 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 55 32 30 | abcdefghijklmU20
00000090 | 32 32 31 35 33 37 38 20 20 20 20 20 20 20 20 20 |2215378
000000A0 | 20 20 20 20 00 00 00 00 00 00 00 00 79 6F 75 72 |          your
000000B0 | 20 49 44 20 69 73 20 3A 20 25 73 0A 00 8B 04 24 | ID is : %s < $
```

```
qr@qr-virtual-machine:~$ gcc -no-pie -m32 -o linkbomb1 main.o phase1.o
qr@qr-virtual-machine:~$ ./linkbomb1
please input you stuid : U202215378
your ID is : U202215378
Bye Bye !
```

(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!` 想要调用 `myfunc` 函

数，首先需要找到 myfunc 函数的偏移地址，计算公式如下：偏移地址 = myfunc 的函数地址 - call 指令偏移一位的地址。

通过 readelf -a phase2.o 获得 .text 段的偏移地址 0x00003c。

[Nr]	Name	Type	Addr	Off	Size	ES	Flg	Lk	Inf	Al
[0]		NULL	00000000	000000	000000	00		0	0	0
[1]	.group	GROUP	00000000	000034	000008	04		23	11	4
[2]	.text	PROGBITS	00000000	00003c	00004d	00	AX	0	0	1

使用 disass phase2 反汇编 phase2.o 文件，可以观察得到 myfunc 的函数地址是 0x00000000，call 指令偏移一位的地址为 $0x38 + 0x1 = 0x39$ ，如下图所示。根据计算公式即可计算出偏移地址 = $0x0 - 0x39 = 0xffffffffc7$ (补码)。由 do_phase 函数反汇编结果还可得知插入的指令在 .text 段中的偏移地址为 0x00000038。

```
Dump of assembler code for function myfunc:
0x00000000 <+0>:      push    %ebp

Dump of assembler code for function do_phase:
0x0000002b <+0>:      push    %ebp
0x0000002c <+1>:      mov     %esp,%ebp
0x0000002e <+3>:      call   0x2f <do_phase+4>
0x00000033 <+8>:      add     $0x1,%eax
0x00000038 <+13>:     nop
0x00000039 <+14>:     nop
```

得到偏移地址后即可编写汇编语句，并编译反汇编得到机器码。

```
1 call 0xffffffffc7
2 mov %ebp, %esp

00000000 <.text>:
0:  e8 c3 ff ff ff      call  0xffffffffc8
5:  89 ec               mov   %ebp,%esp
```

用先前获得的 .text 段的偏移地址 0x00003c 和在 do_phase 函数中插入的指令在 .text 段中的偏移地址 0x00000038 相加即可得到插入指令的地址 0x00000074。将机器码指令 e8 c3 ff ff ff 89 ec 填入 phase2.o 文件中 0x00000074 位置，如下图所示。

00000070 | 01 00 00 00 E8 C3 FF FF FF 89 89 EC

再次反汇编 phase2.o 文件观察，即可发现可以正确地调用 myfunc 函数，编译链接生成 linkbomb2 可执行文件，运行即可得到正确结果，完成任务。

```
qr@qr-virtual-machine:~$ gcc -no-pie -m32 -o linkbomb2 main.o phase2.o
qr@qr-virtual-machine:~$ ./linkbomb2
please input you stuid : U202215378
myfunc is called. Good!
Bye Bye !
```

(3) 第 3 关 有参数的函数调用的机器指令修改

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。整体思路和第二关一样，只需要在调用函数前将一个参数入栈。

按照第二关的步骤，找到 myfunc 函数地址为 0x0，call 指令偏移一位地址为 0x3f，可得到偏移地址为 $0x0 - 0x3f = 0xfffffc1$ 。

```
00000000 <myfunc>:
0: 55                                push    %ebp

0000002e <do_phase>:
2e: 55                                push    %ebp
2f: 89 e5                            mov     %esp,%ebp
31: e8 fc ff ff ff                  call    32 <do_phase+0x4>
36: 05 01 00 00 00                  add     $0x1,%eax
3b: 90                                nop
3c: 90                                nop
3d: 90                                nop
3e: 90                                nop
```

编写 ph3.s 文件，由于要将 do_phase 的参数 pos 直接传递给 myfunc，而 pos 的地址为 0x8(%ebp)，故调用 myfunc 函数前 push 0x8(%ebp)。然后根据偏移地址 0xfffffc1 编写汇编代码 ph3.s 并反汇编获得机器码，并将机器码填入 phase3.o。

```
1 push 0x8(%ebp)
2 call 0xfffffc1
3 mov %ebp, %esp

00000000 <.text>:
0: ff 75 08                        push    0x8(%ebp)
3: e8 bd ff ff ff                  call    0xfffffc5
8: 89 ec                            mov     %ebp,%esp

00000070 | FF FF 05 01 00 00 00 FF 75 08 E8 BD FF FF 89
00000080 | EC 90 90 90 90 90 90 90 90 90 90 90 5D C3 67
```

编译链接生成 linkbomb3 可执行文件，运行即可得到正确结果，完成任务。

```
qr@qr-virtual-machine:~$ gcc -no-pie -m32 -o linkbomb3 main.o phase3.o
qr@qr-virtual-machine:~$ ./linkbomb3
please input you stuid : U202215378
gate 3: offset is : 13!
Bye Bye !
```

(4) 第 4 关 有局部变量的机器指令修改

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。整体思路和第三关一样，区别是需要将传递的参数换成字符串地址，然后将字符串改成自己的学号。

首先通过反汇编了解 do_phase 函数和 myfunc 函数，在 do_phase 函数中，49-5d 这四行语句，就是一个字符串，要将其作为参数传给 myfunc 函数。而再观察 myfunc 函数，在 20 这一行处调用了一个函数，通过 gdb 调试可以发现这个是 printf 函数，1d 这一行 push %edx，正是将字符串传递给 printf 函数。且使用 gdb 调试可以发现，参数需要是地址形式，故在调用 myfunc 函数前入栈的应该是字符串的首地址-0x17(%ebp)。

```
00000000 <myfunc>:
0: 55                                push    %ebp
1: 89 e5                            mov     %esp,%ebp
3: 53                                push    %ebx
4: 83 ec 04                         sub     $0x4,%esp
7: e8 fc ff ff ff                  call    8 <myfunc+0x8>
c: 05 01 00 00 00                  add     $0x1,%eax
11: 83 ec 08                         sub     $0x8,%esp
14: ff 75 08                         push    0x8(%ebp)
17: 8d 90 00 00 00 00               lea     0x0(%eax),%edx
1d: 52                                push    %edx
1e: 89 c3                            mov     %eax,%ebx
20: e8 fc ff ff ff                  call    21 <myfunc+0x21>

0000002e <do_phase>:
2e: 55                                push    %ebp
2f: 89 e5                            mov     %esp,%ebp
31: 83 ec 18                         sub     $0x18,%esp
34: e8 fc ff ff ff                  call    35 <do_phase+0x7>
39: 05 01 00 00 00                  add     $0x1,%eax
3e: 65 a1 14 00 00 00               mov     %gs:0x14,%eax
44: 89 45 f4                         mov     %eax,-0xc(%ebp)
47: 31 c0                            xor     %eax,%eax
49: c7 45 e9 55 32 30 32            movl    $0x32303255,-0x17(%ebp)
50: c7 45 ed 32 31 32 33            movl    $0x33323132,-0x13(%ebp)
57: 66 c7 45 f1 34 35              movw    $0x3534,-0xf(%ebp)
5d: c6 45 f3 00                     movb    $0x0,-0xd(%ebp)
```

现在还需要找到 call 后面接的地址，按照第二关的方法计算，myfunc 的地址为 0x0，call 指令偏移一位的地址为 0x66，故偏移地址为 0x0 - 0x66 = 0xfffff9a。编写汇编代码 ph4.s，并反汇编获得机器码，并将机器码填入 phase4.o，除此之外还需要找到字符串对应的字节，将其改成自己的学号。

```
1 lea -0x17(%ebp), %eax
2 push %eax
3 call 0xfffff9a
4 mov %ebp, %esp
```

```
00000000 <.text>:
0: 8d 45 e9          lea    -0x17(%ebp),%eax
3: 50                push   %eax
4: e8 96 ff ff ff    call   0xfffff9f
9: 89 ec            mov    %ebp,%esp
```

```
00000080 | 89 45 F4 31 C0 C7 45 E9 55 32 30 32 C7 45 ED 32 | %Eô1ÀÇEéU202ÇEi2
00000090 | 31 35 33 66 C7 45 F1 37 38 C6 45 F3 00 8D 45 E9 | 153fÇEñ78ÆEó Éé
000000A0 | 50 E8 96 FF FF FF 89 EC 90 90 90 90 90 90 90 | Pè-ÿÿÿ%i
```

编译链接生成 linkbomb4 可执行文件，运行即可得到正确结果，完成任务。

```
qr@qr-virtual-machine:~$ ./linkbomb4
please input you stuid : U202215378
gate 4: your ID is : U202215378!
Bye Bye !
```

(5) 第 5 关 重定位表的修改

找到存储着目标数据的符号及其信息，在重定位节内将当前符号的信息改成目标符号信息即可。

首先打印出 data 段的数据，确定哪个是目标符号，哪个是当前符号。可以发现目标符号时 classname 和 teachername，需要用他们替换掉 originalclass 和 originalteacher 两个符号。

```
qr@qr-virtual-machine:~$ objdump -d -j .data phase5.o

phase5.o:      file format elf32-i386


Disassembly of section .data:

00000000 <classname>:
0: 43 6f 6d 70 75 74 65 72 20 46 6f 75 6e 64 61 74      Computer Foundat
10: 69 6f 6e 00                                           ion.

00000014 <teachername>:
14: 5a 68 75 20 48 6f 6e 67 00 00 00 00 00 00 00 00      Zhu Hong.....
24: 00 00 00 00                                           ....

00000028 <originalclass>:
28: 63 20 70 72 6f 67 72 61 6d 6d 69 6e 67 00 00 00      c programming...
38: 00 00 00 00                                           ....

0000003c <originalteacher>:
3c: 6d 61 00 00 00 00 00 00 00 00 00 00 00 00 00 00      ma.....
4c: 00 00 00 00                                           ....
```

找到重定位节在 phase5.o 中的偏移位置，如下图所示。

```
Relocation section '.rel.text' at offset 0x764 contains 8 entries:
Offset      Info      Type             Sym.Value      Sym. Name
00000008     00001002 R_386_PC32        00000000      __x86.get_pc_thunk.bx
0000000e     0000110a R_386_GOTPC        00000000      _GLOBAL_OFFSET_TABLE_
00000017     00000c09 R_386_GOTOFF       00000028      originalclass
0000001e     00000309 R_386_GOTOFF       00000000      .rodata
00000024     00001204 R_386_PLT32        00000000      printf
00000030     00000d09 R_386_GOTOFF       0000003c      originalteacher
00000037     00000309 R_386_GOTOFF       00000000      .rodata
0000003d     00001204 R_386_PLT32        00000000      printf
```

```
Section Headers:
```

[Nr]	Name	Type	Addr	Off	Size	ES	Flg	Lk	Inf	Al
[0]		NULL	00000000	000000	000000	00		0	0	0
[1]	.group	GROUP	00000000	000034	000008	04		23	16	4
[2]	.text	PROGBITS	00000000	00003c	00005f	00	AX	0	0	1
[3]	.rel.text	REL	00000000	000764	000040	08	I	23	2	4

打印出重定位节的内容，如下图所示。

```
qr@qr-virtual-machine:~$ readelf -x .rel.text phase5.o

Hex dump of section '.rel.text':
0x00000000 08000000 02100000 0e000000 0a110000 .....
0x00000010 17000000 090c0000 1e000000 09030000 .....
0x00000020 24000000 04120000 30000000 090d0000 $......0.....
0x00000030 37000000 09030000 3d000000 04120000 7.....=.....
```

对比两图的数据，可以发现重定位节存储的是重定位内容的 Offset 和 Info 两个数据。Offset 是重定位的位置，不需要修改，需要修改的是 originalclass 和 originalteacher 的 Info 信息。与下图相对比，可以发现 Info 倒数第二个字节存放的数据是符号的索引数，故修改重定位节内 originalclass 和 originalteacher 的 Info 的倒数第二个字节为 classname 和 teachername 的索引即可。

```
qr@qr-virtual-machine:~$ readelf -s phase5.o

Symbol table '.symtab' contains 19 entries:
   Num:   Value          Size Type      Bind   Vis      Ndx Name
   --:   --:          --: --:      --:   --:      --: --
   0: 00000000           0 NOTYPE   LOCAL DEFAULT UND
   1: 00000000           0 FILE    LOCAL DEFAULT ABS phase5.c
   2: 00000000           0 SECTION LOCAL DEFAULT 2 .text
   3: 00000000           0 SECTION LOCAL DEFAULT 8 .rodata
   4: 00000000           0 SECTION LOCAL DEFAULT 9 .text.__x86.get_[...]
   5: 00000000           0 SECTION LOCAL DEFAULT 10 .debug_info
   6: 00000000           0 SECTION LOCAL DEFAULT 12 .debug_abbrev
   7: 00000000           0 SECTION LOCAL DEFAULT 15 .debug_line
   8: 00000000           0 SECTION LOCAL DEFAULT 17 .debug_str
   9: 00000000           0 SECTION LOCAL DEFAULT 18 .debug_line_str
  10: 00000000          20 OBJECT  GLOBAL DEFAULT 4 classname
  11: 00000014          20 OBJECT  GLOBAL DEFAULT 4 teachername
  12: 00000028          20 OBJECT  GLOBAL DEFAULT 4 originalclass
  13: 0000003c          20 OBJECT  GLOBAL DEFAULT 4 originalteacher
```

hexedit 打开 phase5.o 文件，找到并修改即可。

```
00000770 | 0A 11 00 00 17 00 00 00 | 09 0A 00 00 1E 00 00 00
00000780 | 09 03 00 00 24 00 00 00 | 04 12 00 00 30 00 00 00
00000790 | 09 0B 00 00 37 00 00 00 | 09 03 00 00 3D 00 00 00
```

编译链接生成 linkbomb5 可执行文件，运行即可得到正确结果，完成任务。

```
qr@qr-virtual-machine:~$ gcc -no-pie -m32 -o linkbomb5 main.o phase5.o
qr@qr-virtual-machine:~$ ./linkbomb5
please input you stuid : U202215378
Class Name Computer Foundation
Teacher Name Zhu Hong
Bye Bye !
```

(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，输出自己的学号。

编写 C 语言代码，将 myprint 函数指针（来自 phase6.o）赋值为自己编写的 my 函数，my 函数打印学号，达到目标输出。编写的 phase6_patch.c 文件如下图所示。

```
1 #include<stdio.h>
2 void my();
3 void (*myprint)() = my;
4 void my(){
5     printf("U202215378\n");
6 }
7
```

gcc -c -g main.c -o main.o 用 64 位重新编译 main.c 文件；

gcc -c -g phase6_patch.c -o phase6_patch.o 64 位编译 phase6_patch.c 文件；

gcc -no-pie -o linkbomb6 main.o phase6.o phase6_patch.o 以生成 364 为可执行程序 linkbomb6。最后 ./linkbomb6 编译链接生成 linkbomb6 可执行文件，运行即可得到正确结果，完成任务。

```
qr@qr-virtual-machine:~$ gcc -no-pie -o linkbomb6 main.o phase6.o phase6_patch.o
qr@qr-virtual-machine:~$ ./linkbomb6
please input you stuid : U202215378
U202215378
Bye Bye !
```

(7) 第 7 关 只读数据节的修改

找到 rodata 数据节的偏移地址，将数据改成自己的学号即可。

首先 objdump -s phase7.o 反汇编查看 phase7.o 的 .rodata 的内容，找到需要修改的数据（U202212345）相对于 .rodata 的偏移地址 0x8，如下图所示。

```
Contents of section .rodata:
0000 47617465 20373a20 55323032 32313233  Gate 7: U2022123
0010 343500                                45.
```

然后查看 .rodata 数据节在 phase7.o 中的偏移地址，发现是 0x6c，Size 为 0x13，如下图所示。

Section Headers:										
[Nr]	Name	Type	Addr	Off	Size	ES	Flg	Lk	Inf	Al
[0]		NULL	00000000	000000	000000	00		0	0	0
[1]	.group	GROUP	00000000	000034	000008	04		23	12	4
[2]	.text	PROGBITS	00000000	00003c	00002b	00	AX	0	0	1
[3]	.rel.text	REL	00000000	00053c	000020	08	I 23		2	4
[4]	.data	PROGBITS	00000000	000067	000000	00	WA	0	0	1
[5]	.bss	NOBITS	00000000	000067	000000	00	WA	0	0	1
[6]	.data.rel.local	PROGBITS	00000000	000068	000004	00	WA	0	0	4
[7]	.rel.data.re[...]	REL	00000000	00055c	000008	08	I 23		6	4
[8]	.rodata	PROGBITS	00000000	00006c	000013	00	A	0	0	1

由上面两个数据即可算出需要修改的内容的偏移地址为 0x6c+0x8=0x74，hexedit 打开 phase7.o 将数据修改成自己学号即可。

```
00000060 | 10 90 8B 5D FC C9 C3 00 00 00 00 00 47 61 74 65 | <]üÉÃ Gate
00000070 | 20 37 3A 20 55 32 30 32 32 31 35 33 37 38 00 8B | 7: U202215378 <
00000080 | 04 24 C3 B5 00 00 00 05 00 01 04 00 00 00 00 02 | $Åµ
```


要用 32 位重新编译 main.c 生成 main.o，编译链接生成 linkbomb7 可执行文件，运行即可得到正确结果，完成任务。

```
qr@qr-virtual-machine:~$ ./linkbomb7
please input you stuid : U202215378
Gate 7: U202215378
Bye Bye !
```

2. 描述修改各个文件的基本思想

详见 1。

四、体会

通过本次链接炸弹实验，我在二进制层面深入理解和操作可重定位目标文件的过程中，收获颇丰。实验涉及多个工具和技术，包括 GCC、GDB、readelf、hexdump、hexedit、od 等，这些工具的使用进一步加深了我对编译器、链接器和可执行文件格式的理解。

首先，安装和使用 hexedit 工具让我体验到在二进制层面直接修改目标文件的过程。通过 hexedit 修改 .o 文件中的数据节和代码节，不仅要求我们准确识别要修改的位置，还要掌握如何保存和退出等基本操作。通过该工具，我了解到如何在二进制文件中定位并修改特定的字节，从而实现实验中的各种目标。

实验中的几个任务设计得非常巧妙，从简单的数据节修改到复杂的重定位表修改，每一步都循序渐进地加深了我对可重定位目标文件的理解。例如，在第一关，我通过修改 phase1.o 的数据节使得生成的可执行程序能够输出我的学号，这让我理解了数据节在目标文件中的存储和引用方式。接下来的几关中，通过修改代码节中的机器指令，实现了函数调用和参数传递，这让我对汇编指令和函数调用的机制有了更深入的理解。特别是在第四关，通过修改有局部变量的机器指令，我学会了如何在二进制层面处理字符串和变量。第五关中重定位表的修改，更是让我对重定位表在链接过程中的作用有了深入的理解。我学会了如何通过 readelf 查看符号表和重定位表的信息，并通过修改重定位表中的符号编号，实现了程序的预期输出。

第六关的实验要求我们在 64 位环境下进行操作，并且通过增补一个文件实现预期功能，这让我理解了强弱符号的概念和在实际中的应用。通过对 main.o 和 phase6.o 的对比分析，我学会了如何定义和声明函数，从而实现程序的正常运行。

通过此次实验，我不仅掌握了多种工具的使用，还深入理解了编译器和链接器的工作原理。我学会了如何查看和修改目标文件中的各种节，如数据节、代码节、符号表、重定位表等，并理解了这些节在目标文件中的作用和相互关系。总的来说，本次实验不仅提升了我的实践能力，更加深了我对系统底层工作机制的理解，对我以后的学习和工作都有很大的帮助。