第 7 章家庭作业

——201808010515　计科 1805 黄茂荣

***7.6** 考虑下面的 swap.c 函数版本，它计算自己被调用的次数：

```
1   extern int buf[];
2
3   int *bufp0 = &buf[0];
4   static int *bufp1;
5
6   static void incr()
7   {
8       static int count=0;
9
10      count++;
11  }
12
13  void swap()
14  {
15      int temp;
16
17      incr();
18      bufp1 = &buf[1];
19      temp = *bufp0;
20      *bufp0 = *bufp1;
21      *bufp1 = temp;
22  }
```

对于每个 swap.o 中定义和引用的符号，请指出它是否在模块 swap.o 的 .symtab 节中有符号表条目。如果是这样，请指出定义该符号的模块（swap.o 或 main.o）、符号类型（本地、全局或外部）以及它在模块中所处的节（.text、.data 或 .bss）。

符号	swap.o.symtab 条目?	符号类型	定义符号的模块	节
buf				
bufp0				
bufp1				
swap				
temp				
incr				
count				

分析

swap.o.symtab 是存放模块 swap.o 的符号表，符号表中不包含局部变量的条目，故 temp 不在符号表中，第 2 列除了 temp 为否，其余为是。

buf 前有 extern 表示从其他文件中引入的变量，故 buf 为外部符号；带有静态属性（static）的函数和变量是本地符号，故 bufp1,incr,count 是本地符号；由 swap 模块定义的是全局符号，故 bufp0,swap 是全局符号。

除了 buf 定义在 main.o 模块，其余符号均是定义在 swap.o 模块。

.data 节是存放已经初始化的全局变量，故 buf,bufp0,count 放在.data 节;.bss 是存放未初始化的全局变量,故 bufp1 放在.bss 节;.text 是存放已编译的程序的机器代码，故 swap,incr 放在.text 节。局部变量既不放在.data 节，也不放在.bss 节，而是存放在栈中，故 temp 填“——”

符号	swap.o.symtab	符号类型	定义符号的模块	节
buf	是	External	main.o	.data

bufp0	是	Global	swap.o	.data
bufp1	是	Local	swap.o	.bss
swap	是	Global	swap.o	.text
temp	否	——	——	——
incr	是	Local	swap.o	.text
count	是	Local	swap.o	.data

**7.12 图 7-10 中的 swap 程序包含 5 个重定位的引用。对于每个重定位的引用，给出它在图 7-10 中的行号、运行时存储器地址和值。swap.o 模块中的原始代码和重定位条目如图 7-19 所示。

```
1  00000000 <swap>:
2     0:   55                      push   %ebp
3     1:   8b 15 00 00 00 00       mov    0x0,%edx        Get *bufp0=&buf[0]
4                                  3: R_386_32    bufp0   Relocation entry
5     7:   a1 04 00 00 00          mov    0x4,%eax        Get buf[1]
6                                  8: R_386_32    buf     Relocation entry
7     c:   89 e5                   mov    %esp,%ebp
8     e:   c7 05 00 00 00 00 04    movl   $0x4,0x0        bufp1 = &buf[1];
9    15:   00 00 00
10                                 10: R_386_32   bufp1   Relocation entry
11                                 14: R_386_32   buf     Relocation entry
12    18:   89 ec                  mov    %ebp,%esp
13    1a:   8b 0a                  mov    (%edx),%ecx     temp = buf[0];
14    1c:   89 02                  mov    %eax,(%edx)     buf[0]=buf[1];
15    1e:   a1 00 00 00 00         mov    0x0,%eax        Get *bufp1=&buf[1]
16                                 1f: R_386_32   bufp1   Relocation entry
17    23:   89 08                  mov    %ecx,(%eax)     buf[1]=temp;
18    25:   5d                     pop    %ebp
19    26:   c3                     ret
```

图 7-19 练习题 7.12 的代码和重定位条目

图 7-10 中的行号	地址	值

图 7-10 如下

```
1  080483b4 <main>:
2   80483b4:  55                      push   %ebp
3   80483b5:  89 e5                   mov    %esp,%ebp
4   80483b7:  83 ec 08                sub    $0x8,%esp
5   80483ba:  e8 09 00 00 00          call   80483c8 <swap>          swap();
6   80483bf:  31 c0                   xor    %eax,%eax
7   80483c1:  89 ec                   mov    %ebp,%esp
8   80483c3:  5d                      pop    %ebp
9   80483c4:  c3                      ret
10  80483c5:  90                      nop
11  80483c6:  90                      nop
12  80483c7:  90                      nop

13 080483c8 <swap>:
14  80483c8:  55                      push   %ebp
15  80483c9:  8b 15 5c 94 04 08       mov    0x804945c,%edx          Get *bufp0
16  80483cf:  a1 58 94 04 08          mov    0x8049458,%eax          Get buf[1]
17  80483d4:  89 e5                   mov    %esp,%ebp
18  80483d6:  c7 05 48 95 04 08 58    movl   $0x8049458,0x8049548    bufp1 = &buf[1]
19  80483dd:  94 04 08
20  80483e0:  89 ec                   mov    %ebp,%esp
21  80483e2:  8b 0a                   mov    (%edx),%ecx
22  80483e4:  89 02                   mov    %eax,(%edx)
23  80483e6:  a1 48 95 04 08          mov    0x8049548,%eax          Get *bufp1
24  80483eb:  89 08                   mov    %ecx,(%eax)
25  80483ed:  5d                      pop    %ebp
26  80483ee:  c3                      ret
```

分析：根据题中途 7-19 可以知道有 5 个需要重定位的条目，分别是 bufp0,buf[1],bufp1,buf[1],bufp1,重定位的类型均是 R_386_32（重定位绝对引用）。根据重定位的条目去 7-10 中找到对应的行号，以及值，地址的话根据 *refptr=（unsigned）(ADDR(r.symbol)+*refptr 进行计算。

bufp0:0x08048c8+0x3=0x080483cb

buf[1]: 0x08048c8+0x8=0x080483d0

bufp1: 0x08048c8+0x10=0x080483d8

buf[1]: 0x08048c8+0x14=0x080483dc

bufp1: 0x08048c8+0x1f=0x080483e7

图 7-10 中的行号	地址	值
15(bufp0)	0x080483cb	0x0804945c
16(buf[1])	0x080483d0	0x08049458
18(bufp1)	0x080483d8	0x08049548
18(buf[1])	0x080483dc	0x08049458
23(bufp1)	0x080483e7	0x08049548

7.13 考虑图 7-20 中的 C 代码和相应的可重定位目标模块。

A. 确定当模块被重定位时，链接器将修改 .text 中的哪些指令。对于每条这样的指令，列出它的重定位条目中的信息：节偏移、重定位类型和符号名字。

B. 确定当模块被重定位时，链接器将修改 .data 中的哪些数据目标。对于每条这样的指令，列出它的重定位条目中的信息：节偏移、重定位类型和符号名字。

可以随意使用诸如 OBJDUMP 之类的工具来帮助你解答这个题目。

```
1   extern int p3(void);
2   int x = 1;
3   int *xp = &x;
4
5   void p2(int y) {
6   }
7
8   void p1() {
9       p2(*xp + p3());
10  }
```

a) C 代码

```
1   00000000 <p2>:
2      0:   55                      push   %ebp
3      1:   89 e5                   mov    %esp,%ebp
4      3:   89 ec                   mov    %ebp,%esp
5      5:   5d                      pop    %ebp
6      6:   c3                      ret

7   00000008 <p1>:
8      8:   55                      push   %ebp
9      9:   89 e5                   mov    %esp,%ebp
10     b:   83 ec 08                sub    $0x8,%esp
11     e:   83 c4 f4                add    $0xfffffff4,%esp
12    11:   e8 fc ff ff ff          call   12 <p1+0xa>
13    16:   89 c2                   mov    %eax,%edx
14    18:   a1 00 00 00 00          mov    0x0,%eax
15    1d:   03 10                   add    (%eax),%edx
16    1f:   52                      push   %edx
17    20:   e8 fc ff ff ff          call   21 <p1+0x19>
18    25:   89 ec                   mov    %ebp,%esp
19    27:   5d                      pop    %ebp
20    28:   c3                      ret
```

b) 可重定位目标文件的 .text 节

```
1   00000000 <x>:
2      0:   01 00 00 00
3   00000004 <xp>:
4      4:   00 00 00 00
```

c) 可重定位目标文件的 .data 节

图 7-20 练习题 7.13 的示例代码

分析：

假设所有的符号都已经有了运行时的地址。.text 中有 3 个地方有重定位。第 12 行是调用 p3，返回值被放在了 edx 中。14 行要将 xp 的值放入 eax。这里的 0x0 其实应该重定位为 xp 的地址，所以这应该是一个绝对引用。接着 add (%eax),%edx，也就是完成了*xp + p3()。然后再 push %edx，作为参数压入栈。第 17 行是调用 p2()。最后返回。

所以.text 中有三个地方需要重定位。

在.data 节中，第一个 x 是不需要重定位的，第二个 xp 的值需要重定位为 x 的地址

解答

.text

行号	节偏移	重定位类型	符号名字
12	0x12	相对	p3
14	0x19	绝对	xp
17	0x21	相对	p2

.data

行号	节偏移	重定位类型	符号名字
12	0x04	绝对	x