7-动态链接

静态链接存在**浪费空间，更新困难的问题；将程序的模块分割成独立的模块，运行时再链接**

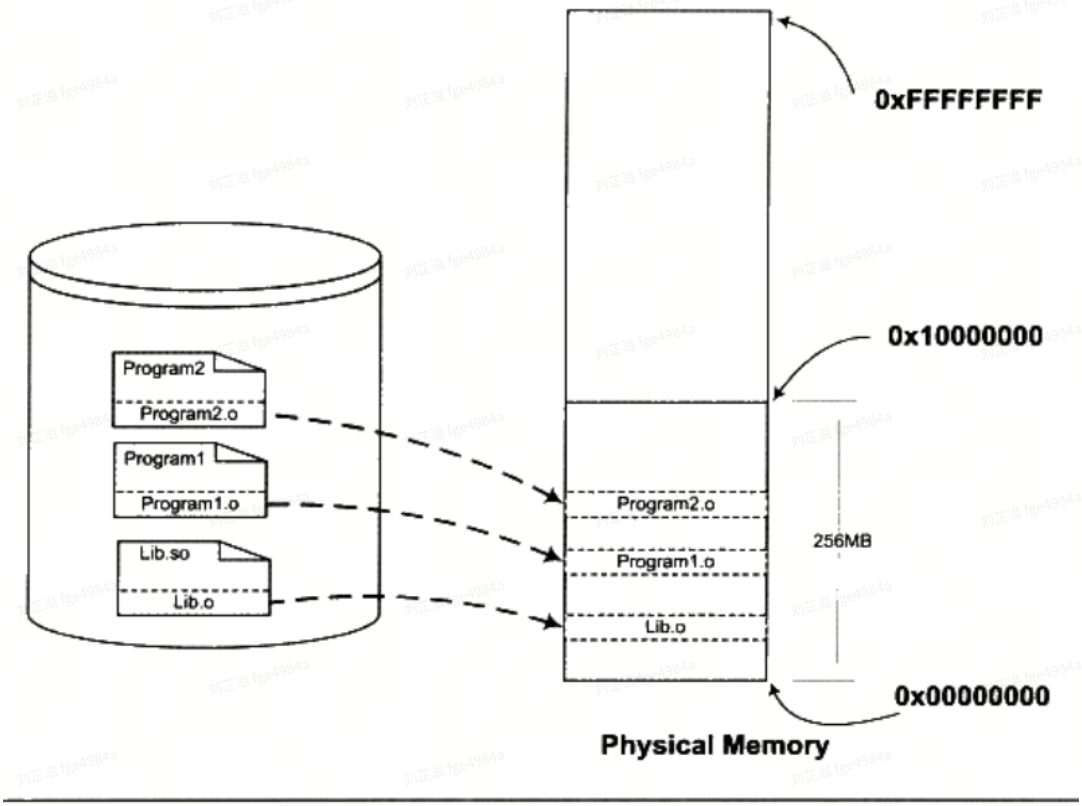


图 7-2　动态链接时文件在内存中的副本

ELF动态链接

- 链接器将动态共享对象的符号**引用标记为动态链接的符号；不进行地址重定位，把该过程留到装载时进行**

```
#ifndef LIB_H_
#define LIB_H_

void lib_func();

#endif /* LIB_H_ */

#include <stdio.h>
#include "lib.h"
void lib_func() {
  printf("lib_func() called.\n");
}
```

- Lib.so装载地址在0x0处，但实际运行不在0x0处

```
lzy@lzydesktop:~/Workspace/linklib/dynamiclinking$ readelf -l Lib.so

Elf file type is DYN (Shared object file)
Entry point 0x0
There are 11 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x00000000000004b0 0x00000000000004b0  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x000000000000011d 0x000000000000011d  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
                 0x00000000000000a4 0x00000000000000a4  R      0x1000
  LOAD           0x0000000000002e80 0x0000000000003e80 0x0000000000003e80
                 0x00000000000001a0 0x00000000000001a8  RW     0x1000
  DYNAMIC        0x0000000000002e90 0x0000000000003e90 0x0000000000003e90
                 0x0000000000000150 0x0000000000000150  RW     0x8
  NOTE           0x00000000000002a8 0x00000000000002a8 0x00000000000002a8
                 0x0000000000000020 0x0000000000000020  R      0x8
  NOTE           0x00000000000002c8 0x00000000000002c8 0x00000000000002c8
                 0x0000000000000024 0x0000000000000024  R      0x4
  GNU_PROPERTY   0x00000000000002a8 0x00000000000002a8 0x00000000000002a8
                 0x0000000000000020 0x0000000000000020  R      0x8
  GNU_EH_FRAME   0x0000000000002000 0x0000000000002000 0x0000000000002000
                 0x0000000000000024 0x0000000000000024  R      0x4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     0x10
  GNU_RELRO      0x0000000000002e80 0x0000000000003e80 0x0000000000003e80
                 0x0000000000000180 0x0000000000000180  R      0x1

 Section to Segment mapping:
  Segment Sections...
   00     .note.gnu.property .note.gnu.build-id .gnu.hash .dynsym .dynstr .rela.dyn
   01     .init .plt .plt.got .text .fini
   02     .eh_frame_hdr .eh_frame
   03     .init_array .fini_array .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.gnu.property
   06     .note.gnu.build-id
   07     .note.gnu.property
   08     .eh_frame_hdr
   09
   10     .init_array .fini_array .dynamic .got
```

```
lzy@lzydesktop:~/Workspace/linklib/dynamiclinking$ cat /proc/18984/maps
5571f803e000-5571f803f000 r--p 00000000 08:20 52558                      /home/lzy/Workspace/linklib/dynamiclinking/main
5571f803f000-5571f8040000 r-xp 00001000 08:20 52558                      /home/lzy/Workspace/linklib/dynamiclinking/main
5571f8040000-5571f8041000 r--p 00002000 08:20 52558                      /home/lzy/Workspace/linklib/dynamiclinking/main
5571f8041000-5571f8042000 r--p 00002000 08:20 52558                      /home/lzy/Workspace/linklib/dynamiclinking/main
5571f8042000-5571f8043000 rw-p 00003000 08:20 52558                      /home/lzy/Workspace/linklib/dynamiclinking/main
7fc92bee4000-7fc92bee7000 rw-p 00000000 00:00 0
7fc92bee7000-7fc92bf0f000 r--p 00000000 08:20 31365                      /usr/lib/x86_64-linux-gnu/libc.so.6
7fc92bf0f000-7fc92c0a4000 r-xp 00028000 08:20 31365                      /usr/lib/x86_64-linux-gnu/libc.so.6
7fc92c0a4000-7fc92c0fc000 r--p 001bd000 08:20 31365                      /usr/lib/x86_64-linux-gnu/libc.so.6
7fc92c0fc000-7fc92c0fd000 ---p 00215000 08:20 31365                      /usr/lib/x86_64-linux-gnu/libc.so.6
7fc92c0fd000-7fc92c101000 r--p 00215000 08:20 31365                      /usr/lib/x86_64-linux-gnu/libc.so.6
7fc92c101000-7fc92c103000 rw-p 00219000 08:20 31365                      /usr/lib/x86_64-linux-gnu/libc.so.6
7fc92c103000-7fc92c110000 rw-p 00000000 00:00 0
7fc92c117000-7fc92c118000 r--p 00000000 08:20 52557                      /home/lzy/Workspace/linklib/dynamiclinking/Lib.so
7fc92c118000-7fc92c119000 r-xp 00001000 08:20 52557                      /home/lzy/Workspace/linklib/dynamiclinking/Lib.so
7fc92c119000-7fc92c11a000 r--p 00002000 08:20 52557                      /home/lzy/Workspace/linklib/dynamiclinking/Lib.so
7fc92c11a000-7fc92c11b000 r--p 00002000 08:20 52557                      /home/lzy/Workspace/linklib/dynamiclinking/Lib.so
7fc92c11b000-7fc92c11c000 rw-p 00003000 08:20 52557                      /home/lzy/Workspace/linklib/dynamiclinking/Lib.so
7fc92c11c000-7fc92c11e000 rw-p 00000000 00:00 0
7fc92c11e000-7fc92c120000 r--p 00000000 08:20 31362                      /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7fc92c120000-7fc92c14a000 r-xp 00002000 08:20 31362                      /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7fc92c14a000-7fc92c155000 r--p 0002c000 08:20 31362                      /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7fc92c156000-7fc92c158000 r--p 00037000 08:20 31362                      /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7fc92c158000-7fc92c15a000 rw-p 00039000 08:20 31362                      /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7fff9c5a3000-7fff9c5c4000 rw-p 00000000 00:00 0                          [stack]
7fff9c5ca000-7fff9c5ce000 r--p 00000000 00:00 0                          [vvar]
7fff9c5ce000-7fff9c5cf000 r-xp 00000000 00:00 0                          [vdso]
```

地址无关代码

- 动态链接会遇到**共享对象地址冲突问题**

装载时重定位

- gcc只使用"-shared"使用**装载时重定位**
- 程序内部相对位置不变，只是整个程序加载地址发生偏移；**直接加上基址偏移即可**
- 如果数据段中有**绝对地址引用；**编译器和链接器产生重定位表；存在R_X86_64_RELATIVE的重定位入口

```
a.so:     file format elf64-x86-64

DYNAMIC RELOCATION RECORDS
OFFSET           TYPE              VALUE
0000000000003e80 R_X86_64_RELATIVE  *ABS*+0x0000000000010f0
0000000000003e88 R_X86_64_RELATIVE  *ABS*+0x0000000000010b0
0000000000004018 R_X86_64_RELATIVE  *ABS*+0x0000000000004018
0000000000004020 R_X86_64_RELATIVE  *ABS*+0x000000000000402c
0000000000003fe0 R_X86_64_GLOB_DAT  __cxa_finalize
0000000000003fe8 R_X86_64_GLOB_DAT  _ITM_registerTMCloneTable
0000000000003ff0 R_X86_64_GLOB_DAT  _ITM_deregisterTMCloneTable
0000000000003ff8 R_X86_64_GLOB_DAT  __gmon_start__
```

PIC

- Positon-independent Code：地址无关代码

```c
static int a;
extern int b;
extern void ext();

void bar() {
  a = 1;    // 内部数据访问
  b = 2;    // 外部数据访问
}

void foo() {
  bar();    // 内部函数调用
  ext();    // 外部函数调用
}
```

- 反汇编

```
pic.so:     file format elf64-x86-64


Disassembly of section .init:
```

```
0000000000001000 <_init>:
    1000:       f3 0f 1e fa             endbr64
    1004:       48 83 ec 08             sub    $0x8,%rsp
    1008:       48 8b 05 e9 2f 00 00    mov    0x2fe9(%rip),%rax        # 3ff8 <__gmon_start__>
    100f:       48 85 c0                test   %rax,%rax
    1012:       74 02                   je     1016 <_init+0x16>
    1014:       ff d0                   callq  *%rax
    1016:       48 83 c4 08             add    $0x8,%rsp
    101a:       c3                      retq

Disassembly of section .plt:

0000000000001020 <.plt>:
    1020:       ff 35 e2 2f 00 00       pushq  0x2fe2(%rip)        # 4008 <_GLOBAL_OFFSET_TABLE_+0x8>
    1026:       f2 ff 25 e3 2f 00 00    bnd jmpq *0x2fe3(%rip)        # 4010 <_GLOBAL_OFFSET_TABLE_+0x10>
    102d:       0f 1f 00                nopl   (%rax)
    1030:       f3 0f 1e fa             endbr64
    1034:       68 00 00 00 00          pushq  $0x0
    1039:       f2 e9 e1 ff ff ff       bnd jmpq 1020 <.plt>
    103f:       90                      nop
    1040:       f3 0f 1e fa             endbr64
    1044:       68 01 00 00 00          pushq  $0x1
    1049:       f2 e9 d1 ff ff ff       bnd jmpq 1020 <.plt>
    104f:       90                      nop

Disassembly of section .plt.got:

0000000000001050 <__cxa_finalize@plt>:
    1050:       f3 0f 1e fa             endbr64
    1054:       f2 ff 25 85 2f 00 00    bnd jmpq *0x2f85(%rip)        # 3fe0 <__cxa_finalize>
    105b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

Disassembly of section .plt.sec:

0000000000001060 <ext@plt>:
    1060:       f3 0f 1e fa             endbr64
    1064:       f2 ff 25 ad 2f 00 00    bnd jmpq *0x2fad(%rip)        # 4018 <ext+0x2e9e>
    106b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

0000000000001070 <bar@plt>:
    1070:       f3 0f 1e fa             endbr64
```

```
    1074:       f2 ff 25 a5 2f 00 00    bnd jmpq *0x2fa5(%rip)        # 4020
<bar+0x2ee7>
    107b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

Disassembly of section .text:

0000000000001080 <deregister_tm_clones>:
    1080:       48 8d 3d a9 2f 00 00    lea    0x2fa9(%rip),%rdi        # 4030
 <completed.8061>
    1087:       48 8d 05 a2 2f 00 00    lea    0x2fa2(%rip),%rax        # 4030
 <completed.8061>
    108e:       48 39 f8                cmp    %rdi,%rax
    1091:       74 15                   je     10a8
<deregister_tm_clones+0x28>
    1093:       48 8b 05 56 2f 00 00    mov    0x2f56(%rip),%rax        #
3ff0 <_ITM_deregisterTMCloneTable>
    109a:       48 85 c0                test   %rax,%rax
    109d:       74 09                   je     10a8
<deregister_tm_clones+0x28>
    109f:       ff e0                   jmpq   *%rax
    10a1:       0f 1f 80 00 00 00 00    nopl   0x0(%rax)
    10a8:       c3                      retq
    10a9:       0f 1f 80 00 00 00 00    nopl   0x0(%rax)

00000000000010b0 <register_tm_clones>:
    10b0:       48 8d 3d 79 2f 00 00    lea    0x2f79(%rip),%rdi        # 4030
 <completed.8061>
    10b7:       48 8d 35 72 2f 00 00    lea    0x2f72(%rip),%rsi        # 4030
 <completed.8061>
    10be:       48 29 fe                sub    %rdi,%rsi
    10c1:       48 89 f0                mov    %rsi,%rax
    10c4:       48 c1 ee 3f             shr    $0x3f,%rsi
    10c8:       48 c1 f8 03             sar    $0x3,%rax
    10cc:       48 01 c6                add    %rax,%rsi
    10cf:       48 d1 fe                sar    %rsi
    10d2:       74 14                   je     10e8 <register_tm_clones+0x38>
    10d4:       48 8b 05 0d 2f 00 00    mov    0x2f0d(%rip),%rax        #
3fe8 <_ITM_registerTMCloneTable>
    10db:       48 85 c0                test   %rax,%rax
    10de:       74 08                   je     10e8 <register_tm_clones+0x38>
    10e0:       ff e0                   jmpq   *%rax
    10e2:       66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
    10e8:       c3                      retq
    10e9:       0f 1f 80 00 00 00 00    nopl   0x0(%rax)

00000000000010f0 <__do_global_dtors_aux>:
    10f0:       f3 0f 1e fa             endbr64
```

```
    10f4:       80 3d 35 2f 00 00 00    cmpb   $0x0,0x2f35(%rip)        # 4030
 <completed.8061>
    10fb:       75 2b                   jne    1128
<__do_global_dtors_aux+0x38>
    10fd:       55                      push   %rbp
    10fe:       48 83 3d da 2e 00 00    cmpq   $0x0,0x2eda(%rip)        #
3fe0 <__cxa_finalize>
    1105:       00
    1106:       48 89 e5                mov    %rsp,%rbp
    1109:       74 0c                   je     1117
<__do_global_dtors_aux+0x27>
    110b:       48 8b 3d 16 2f 00 00    mov    0x2f16(%rip),%rdi        # 4028
 <__dso_handle>
    1112:       e8 39 ff ff ff          callq  1050 <__cxa_finalize@plt>
    1117:       e8 64 ff ff ff          callq  1080 <deregister_tm_clones>
    111c:       c6 05 0d 2f 00 00 01    movb   $0x1,0x2f0d(%rip)        # 4030
 <completed.8061>
    1123:       5d                      pop    %rbp
    1124:       c3                      retq
    1125:       0f 1f 00                nopl   (%rax)
    1128:       c3                      retq
    1129:       0f 1f 80 00 00 00 00    nopl   0x0(%rax)

0000000000001130 <frame_dummy>:
    1130:       f3 0f 1e fa             endbr64
    1134:       e9 77 ff ff ff          jmpq   10b0 <register_tm_clones>

0000000000001139 <bar>:
    1139:       f3 0f 1e fa             endbr64
    113d:       55                      push   %rbp
    113e:       48 89 e5                mov    %rsp,%rbp
    1141:       c7 05 e9 2e 00 00 01    movl   $0x1,0x2ee9(%rip)        # 4034
 <a>
    1148:       00 00 00
    114b:       48 8b 05 86 2e 00 00    mov    0x2e86(%rip),%rax        #
3fd8 <b-0x60>
    1152:       c7 00 02 00 00 00       movl   $0x2,(%rax)
    1158:       90                      nop
    1159:       5d                      pop    %rbp
    115a:       c3                      retq

000000000000115b <foo>:
    115b:       f3 0f 1e fa             endbr64
    115f:       55                      push   %rbp
    1160:       48 89 e5                mov    %rsp,%rbp
    1163:       b8 00 00 00 00          mov    $0x0,%eax
    1168:       e8 03 ff ff ff          callq  1070 <bar@plt>
```

```
124      116d:       b8 00 00 00 00          mov    $0x0,%eax
125      1172:       e8 e9 fe ff ff          callq  1060 <ext@plt>
126      1177:       90                      nop
127      1178:       5d                      pop    %rbp
128      1179:       c3                      retq
129
130  000000000000117a <ext>:
131      117a:       f3 0f 1e fa             endbr64
132      117e:       55                      push   %rbp
133      117f:       48 89 e5                mov    %rsp,%rbp
134      1182:       90                      nop
135      1183:       5d                      pop    %rbp
136      1184:       c3                      retq
137
138  Disassembly of section .fini:
139
140  0000000000001188 <_fini>:
141      1188:       f3 0f 1e fa             endbr64
142      118c:       48 83 ec 08             sub    $0x8,%rsp
143      1190:       48 83 c4 08             add    $0x8,%rsp
144      1194:       c3                      retq
```

模块内部调用或跳转

- **模块内部函数相对位置固定**；直接使用call offset调用函数即可

模块内部数据访问

得到PC值+偏移量

- 首先call一个内部函数（`__i686.get_pc_thunk.cx`），这个函数将call指令压栈的地址取出保存到一个寄存器中，返回。
- 这样，就巧妙的拿到了PC指针的值。然后加上offset，就可以得到内部变量的地址。

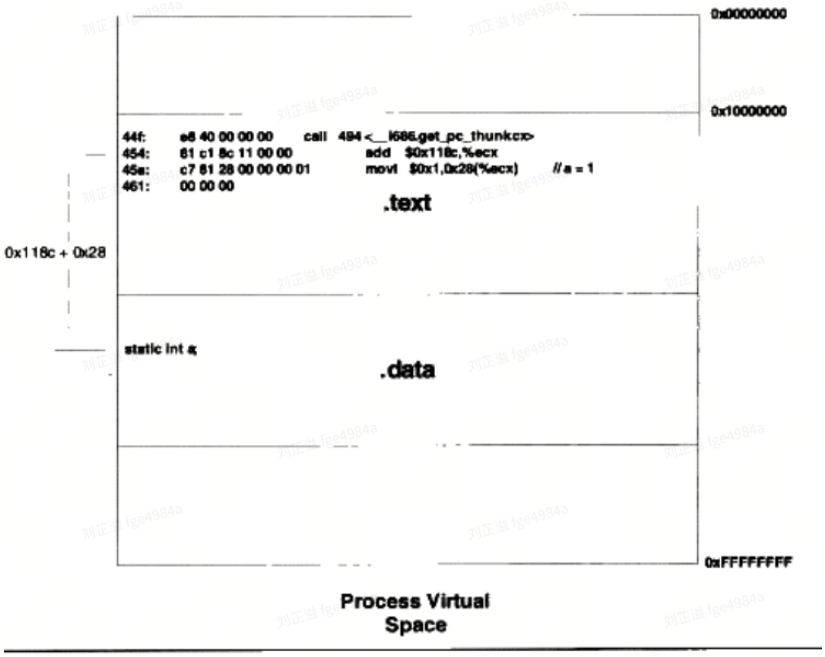


图 7-6　模块内部数据访问示意

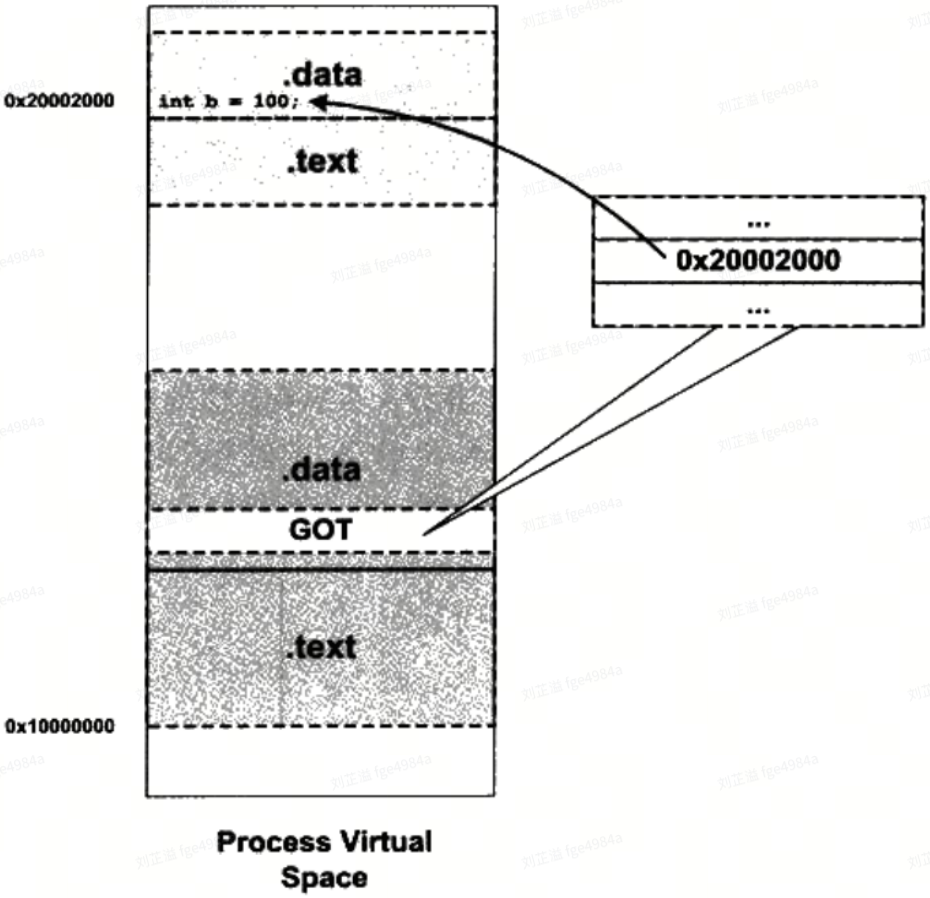
注意：x86_64位系统下直接用%rip来找PC（RIP相对寻址）

- 字符串等不再是绝对地址；使用相对地址
- 寻找位置变成%rip+relative offset
- 下图中a, b均是RIP相对寻址

```
0000000000001139 <bar>:
    1139:       f3 0f 1e fa             endbr64
    113d:       55                      push   %rbp
    113e:       48 89 e5                mov    %rsp,%rbp
    1141:       c7 05 e9 2e 00 00 01    movl   $0x1,0x2ee9(%rip)        # 4034
 <a>
    1148:       00 00 00
    114b:       48 8b 05 86 2e 00 00    mov    0x2e86(%rip),%rax        #
3fd8 <b-0x60>
    1152:       c7 00 02 00 00 00       movl   $0x2,(%rax)
    1158:       90                      nop
    1159:       5d                      pop    %rbp
    115a:       c3                      retq
```

模块间数据访问

- 模块间数据访问目标地址再装载时决定
- **GOT全局偏移表：指向其他模块定义变量的指针**
- 代码需要引用全局变量时，通过GOT中对应的项简介引用



- 读取动态链接重定位项

```
114b:       48 8b 05 86 2e 00 00    mov    0x2e86(%rip),%rax        # 3fd8 <b-0x60>
 # plt段
0000000000001060 <ext@plt>:
    1060:       f3 0f 1e fa             endbr64
    1064:       f2 ff 25 ad 2f 00 00    bnd jmpq *0x2fad(%rip)        # 4018 <ext+0x2e9e>
    106b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

0000000000001070 <bar@plt>:
    1070:       f3 0f 1e fa             endbr64
    1074:       f2 ff 25 a5 2f 00 00    bnd jmpq *0x2fa5(%rip)        # 4020 <bar+0x2ee7>
    107b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

```

```
lzy@lzy:~/Workspace/Linker_Lib/fPIC$ objdump -R pic.so

pic.so:     file format elf64-x86-64

DYNAMIC RELOCATION RECORDS
OFFSET           TYPE              VALUE
0000000000003e48 R_X86_64_RELATIVE  *ABS*+0x0000000000001130
0000000000003e50 R_X86_64_RELATIVE  *ABS*+0x00000000000010f0
0000000000004028 R_X86_64_RELATIVE  *ABS*+0x0000000000004028
0000000000003fd8 R_X86_64_GLOB_DAT  b
0000000000003fe0 R_X86_64_GLOB_DAT  __cxa_finalize
0000000000003fe8 R_X86_64_GLOB_DAT  _ITM_registerTMCloneTable
0000000000003ff0 R_X86_64_GLOB_DAT  _ITM_deregisterTMCloneTable
0000000000003ff8 R_X86_64_GLOB_DAT  __gmon_start__
0000000000004018 R_X86_64_JUMP_SLOT  ext
0000000000004020 R_X86_64_JUMP_SLOT  bar
```

模块间调用、跳转

- GOT中保存目标函数的地址；调用函数通过GOT中的项间接跳转

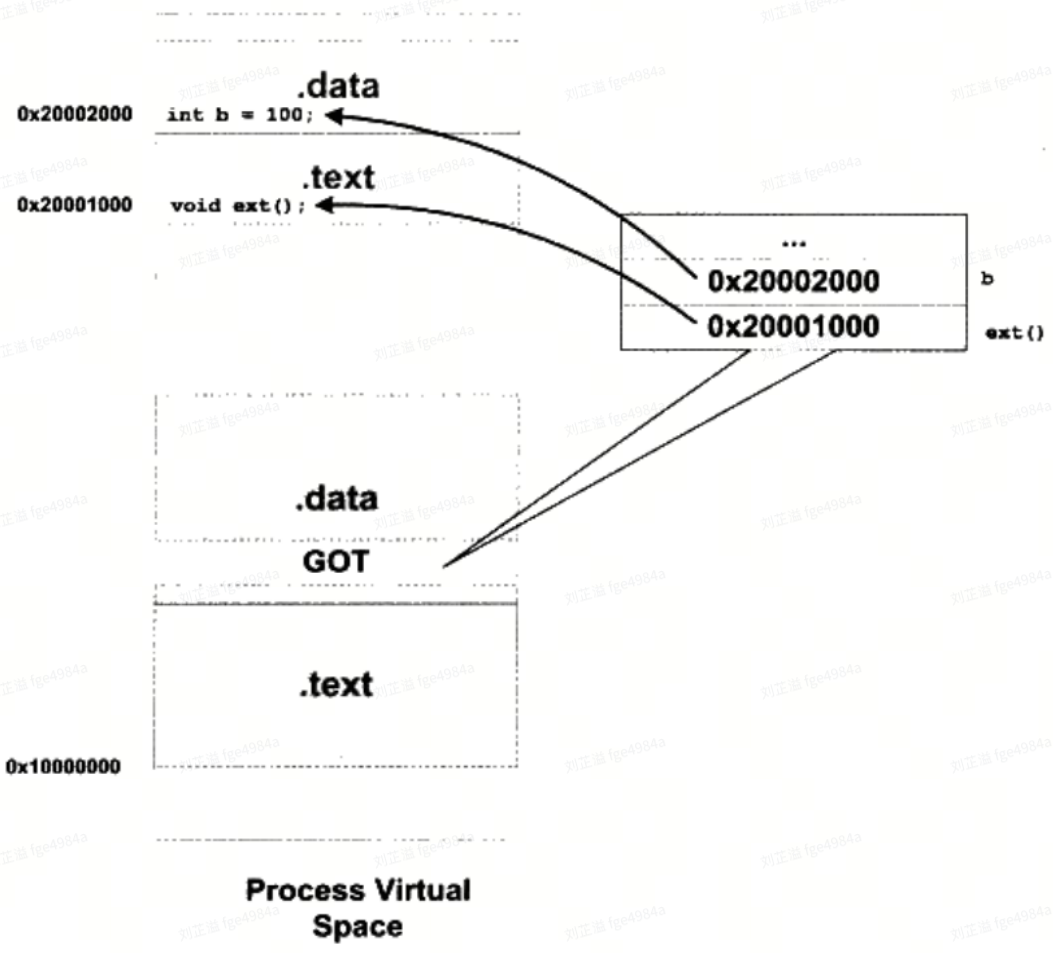


图 7-8　模块间调用、跳转

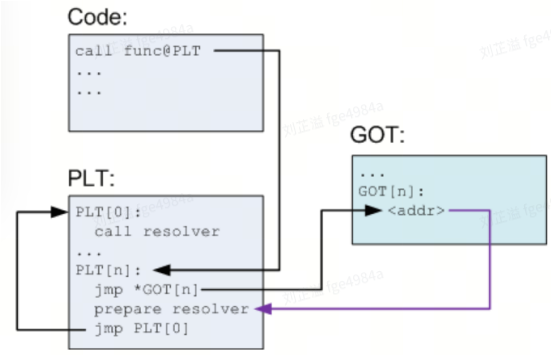
共享模块的全局变量问题

- ELF共享库默认把定义在模块内部的全局变量当作定义在其他模块的全局变量；通过GOT访问
- **如果global var在可执行文件中有副本；GOT中地址改为副本地址**
- **共享库被多个进程加载；数据段在每个进程都有独立的副本；任何一个进程只访问自己的副本**

延迟绑定（PLT，Procedure Linkage Table）

- 函数第一次被用到时才进行绑定
- 通过GOT间接跳转
  - 此时bar@GOT没有填入；直接执行下面的指令
  - **压入plt的索引**
  - **压入模块的ID**
  - **_dl_runtime_resolve_avx查找函数在.so中的地址填入offset中**
- 解析后通过第一条指令直接跳转

```
bar@plt:
jmp *(bar@GOT)
push n
push moduleID
jump _dl_runtime_resolve
```



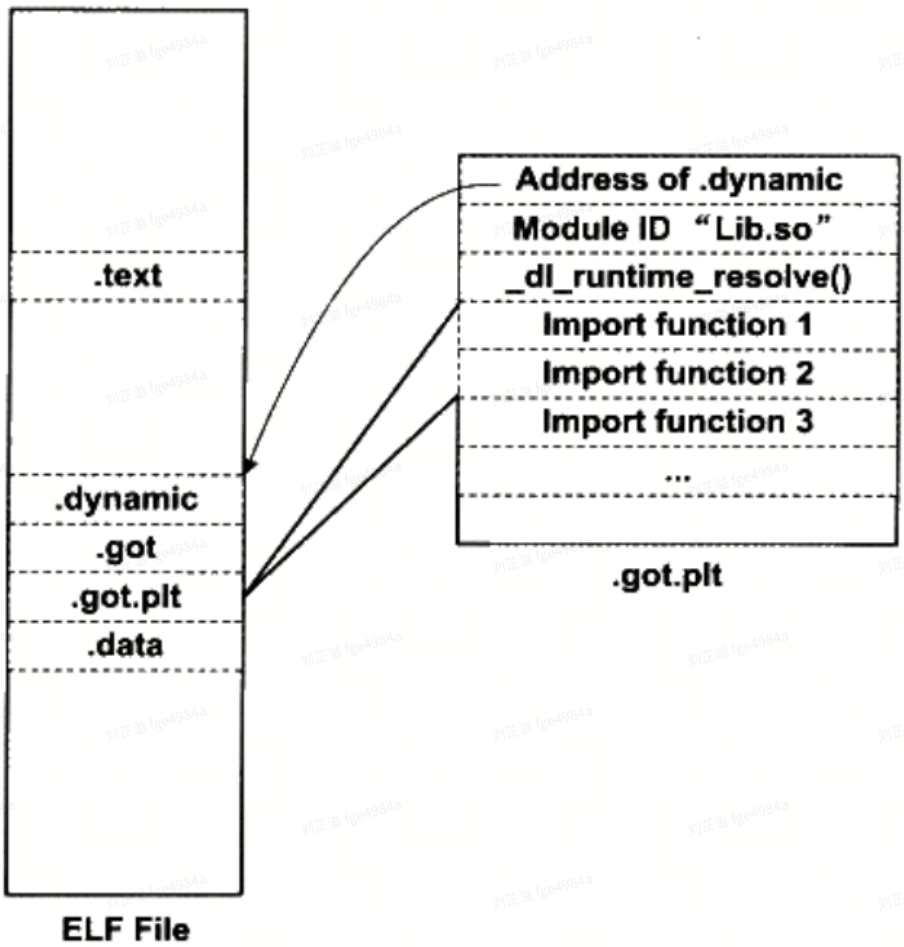

```
Relocation section '.rela.plt' at offset 0x5e8 contains 1 entry:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000003fd0  000200000007 R_X86_64_JUMP_SLO 0000000000000000 printf@GLIBC_2.2.5 +
 0
```

```c
#include <stdio.h>

void testprintf() { printf("hello\n"); }

int main() {
  char acTemp[100] = {0};
  printf("begin\n");
  testprintf();
  return 0;
}
```

```
(gdb) x /5i $pc
=> 0x55555555516d <testprintf+4>:        push   %rbp
   0x55555555516e <testprintf+5>:        mov    %rsp,%rbp
   0x555555555171 <testprintf+8>:        lea    0xe8c(%rip),%rax        # 0x555555556004
   0x555555555178 <testprintf+15>:       mov    %rax,%rdi
   0x55555555517b <testprintf+18>:       call   0x555555555060 <puts@plt>
(gdb) b *0x555555555060
Breakpoint 3 at 0x555555555060
(gdb) c
Continuing.

Breakpoint 3, 0x0000555555555060 in puts@plt ()
(gdb) x /5i $pc
=> 0x555555555060 <puts@plt>:   endbr64
   0x555555555064 <puts@plt+4>: bnd jmp *0x2f5d(%rip)        # 0x555555557fc8 <puts@got.plt>
   0x55555555506b <puts@plt+11>:        nopl   0x0(%rax,%rax,1)
   0x555555555070 <__stack_chk_fail@plt>:       endbr64
   0x555555555074 <__stack_chk_fail@plt+4>:     bnd jmp *0x2f55(%rip)       # 0x555555557fd0 <__stack_chk_fail@got.plt>
```

PLT结构

- 第一项.dynamic段地址
- 第二项是MoudelID



动态链接相关结构

.interp段

- 保存一个字符串；可执行文件需要的动态链接器的路径

```
Contents of section .interp:
 0318 2f6c6962 36342f6c 642d6c69 6e75782d  /lib64/ld-linux-
 0328 7838362d 36342e73 6f2e3200           x86-64.so.2.
```

.dynamic段

- 保存了动态链接需要的基本信息：共享对象，动态链接符号表位置，动态链接重定位表位置，共享对象初始化代码地址

```
lzy@lzy:~/Workspace/Linker_Lib/fPIC$ readelf -d pic.so

Dynamic section at offset 0x2150 contains 20 entries:
  Tag        Type                         Name/Value
 0x000000000000000c (INIT)               0x1000
 0x000000000000000d (FINI)               0x11c0
 0x0000000000000019 (INIT_ARRAY)         0x3140
 0x000000000000001b (INIT_ARRAYSZ)       8 (bytes)
 0x000000000000001a (FINI_ARRAY)         0x3148
 0x000000000000001c (FINI_ARRAYSZ)       8 (bytes)
 0x000000006ffffef5 (GNU_HASH)           0x2b8
 0x0000000000000005 (STRTAB)             0x3e0
 0x0000000000000006 (SYMTAB)             0x2f0
 0x000000000000000a (STRSZ)              104 (bytes)
 0x000000000000000b (SYMENT)             24 (bytes)
 0x0000000000000003 (PLTGOT)             0x32f8
 0x0000000000000002 (PLTRELSZ)           72 (bytes)
 0x0000000000000014 (PLTREL)             RELA
 0x0000000000000017 (JMPREL)             0x508
 0x0000000000000007 (RELA)               0x448
 0x0000000000000008 (RELASZ)             192 (bytes)
 0x0000000000000009 (RELAENT)            24 (bytes)
 0x000000006ffffff9 (RELACOUNT)          3
 0x0000000000000000 (NULL)               0x0
```

动态符号表

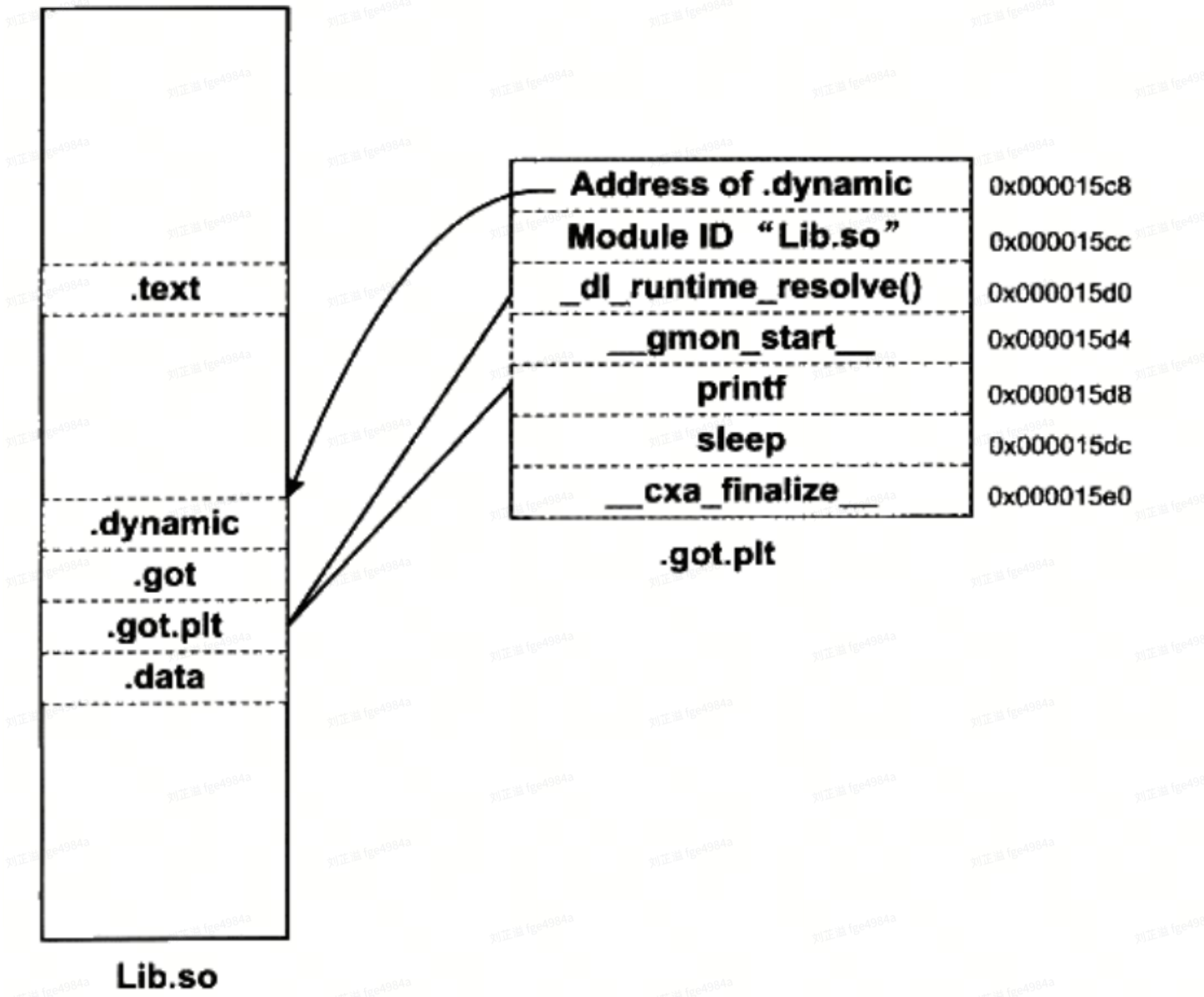
- .dynsym只保存与动态链接相关的符号
- .symtal保存所有符号
- Readelf -sD：查看动态符号表和符号哈希表

动态链接重定位表

- .rel.dyn是对数据引用的修正（.got和.data）和.rel.plt对函数引用进行修正(.got.plt)

```
lzy@lzy:~/Workspace/Linker_Lib/fPIC$ readelf -r pic.so

Relocation section '.rela.dyn' at offset 0x448 contains 8 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000003140  000000000008 R_X86_64_RELATIVE                    1150
000000003148  000000000008 R_X86_64_RELATIVE                    1110
000000003328  000000000008 R_X86_64_RELATIVE                    3328
0000000032d0  000900000006 R_X86_64_GLOB_DAT 0000000000003338 b + 0
0000000032d8  000100000006 R_X86_64_GLOB_DAT 0000000000000000 __cxa_finalize + 0
0000000032e0  000200000006 R_X86_64_GLOB_DAT 0000000000000000 _ITM_registerTMClone
Ta + 0
0000000032e8  000300000006 R_X86_64_GLOB_DAT 0000000000000000 _ITM_deregisterTMClo
ne + 0
0000000032f0  000400000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0

Relocation section '.rela.plt' at offset 0x508 contains 3 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000003310  000800000007 R_X86_64_JUMP_SLO 00000000000011b3 ext + 0
000000003318  000500000007 R_X86_64_JUMP_SLO 000000000000117b foo + 0
000000003320  000600000007 R_X86_64_JUMP_SLO 0000000000001159 bar + 0
```

- GLOB_DAT和JUMP_SLOT被修正的位置只需要直接填入符号即可



- R_X86_64_RELATIVE：公式:B+A **B:.so文件加载到内存中的基地址 A:被重定位处原值，表示引用符号在.so文件中的偏移**

动态链接的步骤和实现

动态链接器自举

- 动态链接器入口地址是自举代码；
- 自举代码找到自己的GOT；找到.dynamic段中的重定位表和符号表
- 得到动态链接器本身的重定位入口，将其全部重定位

装载共享对象

- 全局符号介入
  - **一个符号需要被加入全局符号表时；如果相同的符号已经存在，后被加入的符号被忽略**
  - 将函数变为模块私有：使用static，可以变为模块内部调用指令

重定位和初始化

- **重新遍历可执行文件和每个共享对象的重定位表；修正GOT/PLT需要重定位的位置**
- 执行.init段的代码
  - **如果可执行文件中有.init，不执行；由程序初始化程序执行**
- **动态链接库本身是静态链接的；不能依赖于其他共享对象**

动态链接器可以被当作可执行文件运行，那么的装载地址应该是多少？

ld.so 的装载地址跟一般的共享对象没区别，即为 0x00000000。这个装载地址是一个无效的装载地址，作为一个共享库，内核在装载它时会为其选择一个合适的装载地址。

显示运行时链接

- 运行时加载；程序运行时控制加载指定的模块；不需要该模块时卸载
- **共享对象由动态链接器装载并链接；动态库装载由动态链接器提供的API进行操作**

dlopen

- 绝对路径尝试直接打开动态库
- 相对路径
  - 查找LD_LIBRARY_PATH
  - 查找/etc/ld.so.cache的共享库路径
  - /lib, /usr/lib

```
1   void *dlopen(const char *filename, int flag);
```

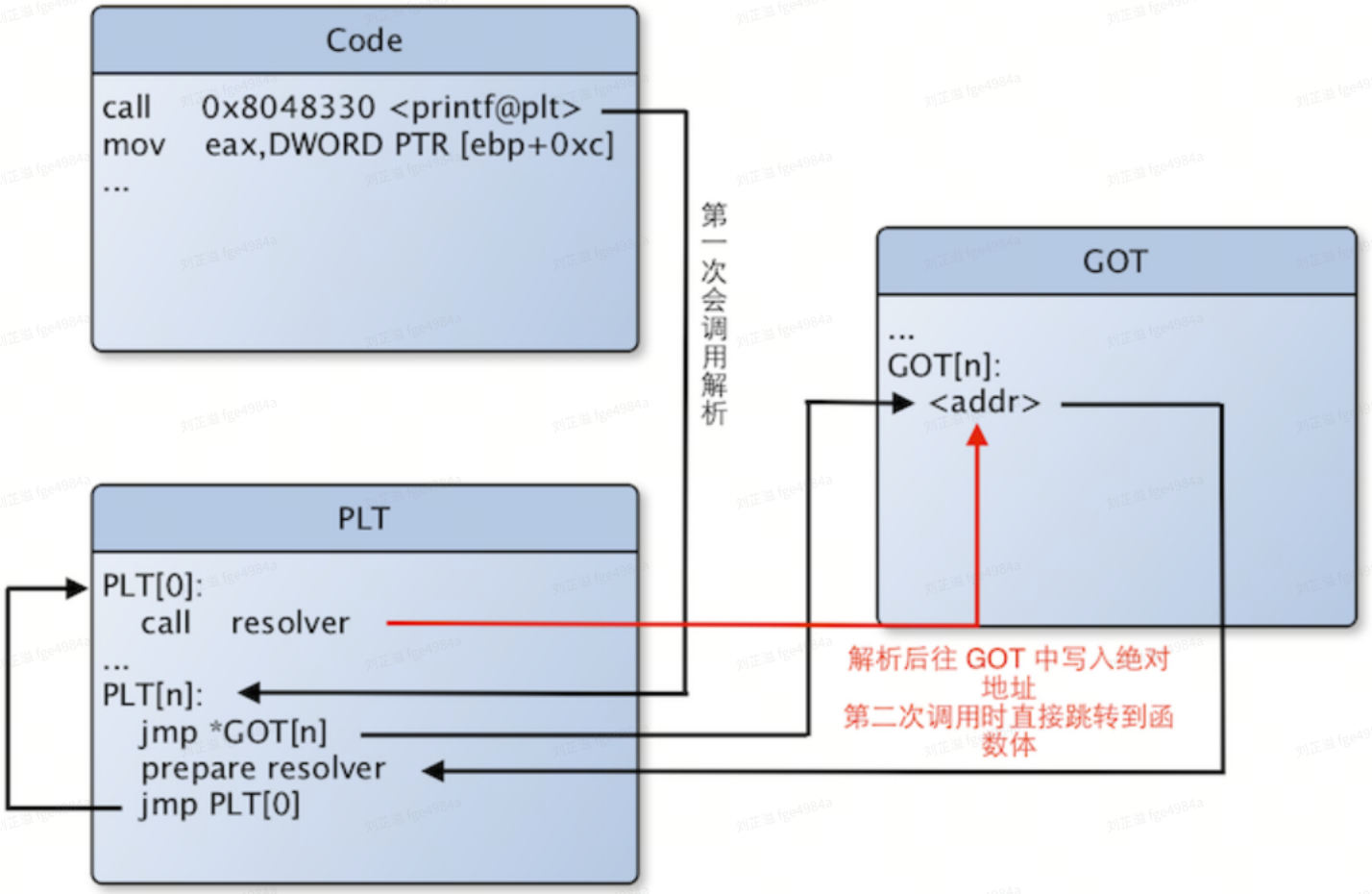
dlsym

- 找到需要的符号
- 第一个参数是动态库句柄；第二个参数是要查找的符号名字

```
1   void *dlsym(void *handle, char *symbol);
```

- 使用dlerror判断上一次调用是否成功；返回NULL，表示成功

printf的例子




```c
#include <stdio.h>

int main()
{
    printf("hello");
    printf("hello again");
    return 0;
}
```

- objdump查看反汇编
  - printf在plt中

```
Disassembly of section .plt:

0000000000001020 <.plt>:
    1020:       ff 35 9a 2f 00 00       pushq  0x2f9a(%rip)        # 3fc0
<_GLOBAL_OFFSET_TABLE_+0x8>
    1026:       f2 ff 25 9b 2f 00 00    bnd jmpq *0x2f9b(%rip)        # 3fc8
<_GLOBAL_OFFSET_TABLE_+0x10>
    102d:       0f 1f 00                nopl   (%rax)
    1030:       f3 0f 1e fa             endbr64
```

```
    1034:       68 00 00 00 00          pushq  $0x0
    1039:       f2 e9 e1 ff ff ff       bnd jmpq 1020 <.plt>
    103f:       90                      nop

Disassembly of section .plt.got:

0000000000001040 <__cxa_finalize@plt>:
    1040:       f3 0f 1e fa             endbr64
    1044:       f2 ff 25 ad 2f 00 00    bnd jmpq *0x2fad(%rip)        # 3ff8
<__cxa_finalize@GLIBC_2.2.5>
    104b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

Disassembly of section .plt.sec:

0000000000001050 <printf@plt>:
    1050:       f3 0f 1e fa             endbr64
    1054:       f2 ff 25 75 2f 00 00    bnd jmpq *0x2f75(%rip)        # 3fd0
<printf@GLIBC_2.2.5>
    105b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
```

- 运行后存储空间布局

```
(gdb) shell cat /proc/12700/maps
555555554000-555555555000 r--p 00000000 08:20 282658
/home/lzy/Workspace/Linker_Lib/PLT/a.out
555555555000-555555556000 r-xp 00001000 08:20 282658
/home/lzy/Workspace/Linker_Lib/PLT/a.out
555555556000-555555557000 r--p 00002000 08:20 282658
/home/lzy/Workspace/Linker_Lib/PLT/a.out
555555557000-555555558000 r--p 00002000 08:20 282658
/home/lzy/Workspace/Linker_Lib/PLT/a.out
555555558000-555555559000 rw-p 00003000 08:20 282658
/home/lzy/Workspace/Linker_Lib/PLT/a.out
7ffff7dbe000-7ffff7de0000 r--p 00000000 08:20 29539
/usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7de0000-7ffff7f58000 r-xp 00022000 08:20 29539
/usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7f58000-7ffff7fa6000 r--p 0019a000 08:20 29539
/usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7fa6000-7ffff7faa000 r--p 001e7000 08:20 29539
/usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7faa000-7ffff7fac000 rw-p 001eb000 08:20 29539
/usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7fac000-7ffff7fb2000 rw-p 00000000 00:00 0
```

```
7ffff7fc9000-7ffff7fcd000 r--p 00000000 00:00 0
[vvar]
7ffff7fcd000-7ffff7fcf000 r-xp 00000000 00:00 0
[vdso]
7ffff7fcf000-7ffff7fd0000 r--p 00000000 08:20 29534
/usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7fd0000-7ffff7ff3000 r-xp 00001000 08:20 29534
/usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ff3000-7ffff7ffb000 r--p 00024000 08:20 29534
/usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ffc000-7ffff7ffd000 r--p 0002c000 08:20 29534
/usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ffd000-7ffff7ffe000 rw-p 0002d000 08:20 29534
/usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ffe000-7ffff7fff000 rw-p 00000000 00:00 0
7ffffffdd000-7ffffffff000 rw-p 00000000 00:00 0
[stack]
```

- 运行后main

```
Dump of assembler code for function main:
   0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
=> 0x0000555555555151 <+8>:     lea    0xeac(%rip),%rdi        #
0x555555556004
   0x0000555555555158 <+15>:    mov    $0x0,%eax
   0x000055555555515d <+20>:    callq  0x555555555050 <printf@plt>
   0x0000555555555162 <+25>:    lea    0xea1(%rip),%rdi        #
0x55555555600a
   0x0000555555555169 <+32>:    mov    $0x0,%eax
   0x000055555555516e <+37>:    callq  0x555555555050 <printf@plt>
   0x0000555555555173 <+42>:    mov    $0x0,%eax
   0x0000555555555178 <+47>:    pop    %rbp
   0x0000555555555179 <+48>:    retq
End of assembler dump.
```

- 反汇编调用的地址

```
(gdb) disas 0x555555555050
Dump of assembler code for function printf@plt:
   0x0000555555555050 <+0>:     endbr64
```

```
4      0x0000555555555054 <+4>:     bnd jmpq *0x2f75(%rip)        #
   0x555555557fd0 <printf@got.plt>
5      0x000055555555505b <+11>:    nopl   0x0(%rax,%rax,1)
6   End of assembler dump.
```

- GOT数组

Address	Entry	Contents	Description
08049674	GOT[0]	0804969c	address of `.dynamic` section
08049678	GOT[1]	4000a9f8	identifying info for the linker
0804967c	GOT[2]	4000596f	entry point in dynamic linker
08049680	GOT[3]	0804845a	address of `pushl` in PLT[1] (`printf`)
08049684	GOT[4]	0804846a	address of `pushl` in PLT[2] (`addvec`)

Figure 7.17 **The global offset table (GOT) for executable** p2. The original code is in Figures 7.5 and 7.6.

```
1   (gdb) x/4xw 0x555555557fd0
2   0x555555557fd0 <printf@got.plt>:        0xf7e1fc90      0x00007fff 0x00000000
        0x00000000
```