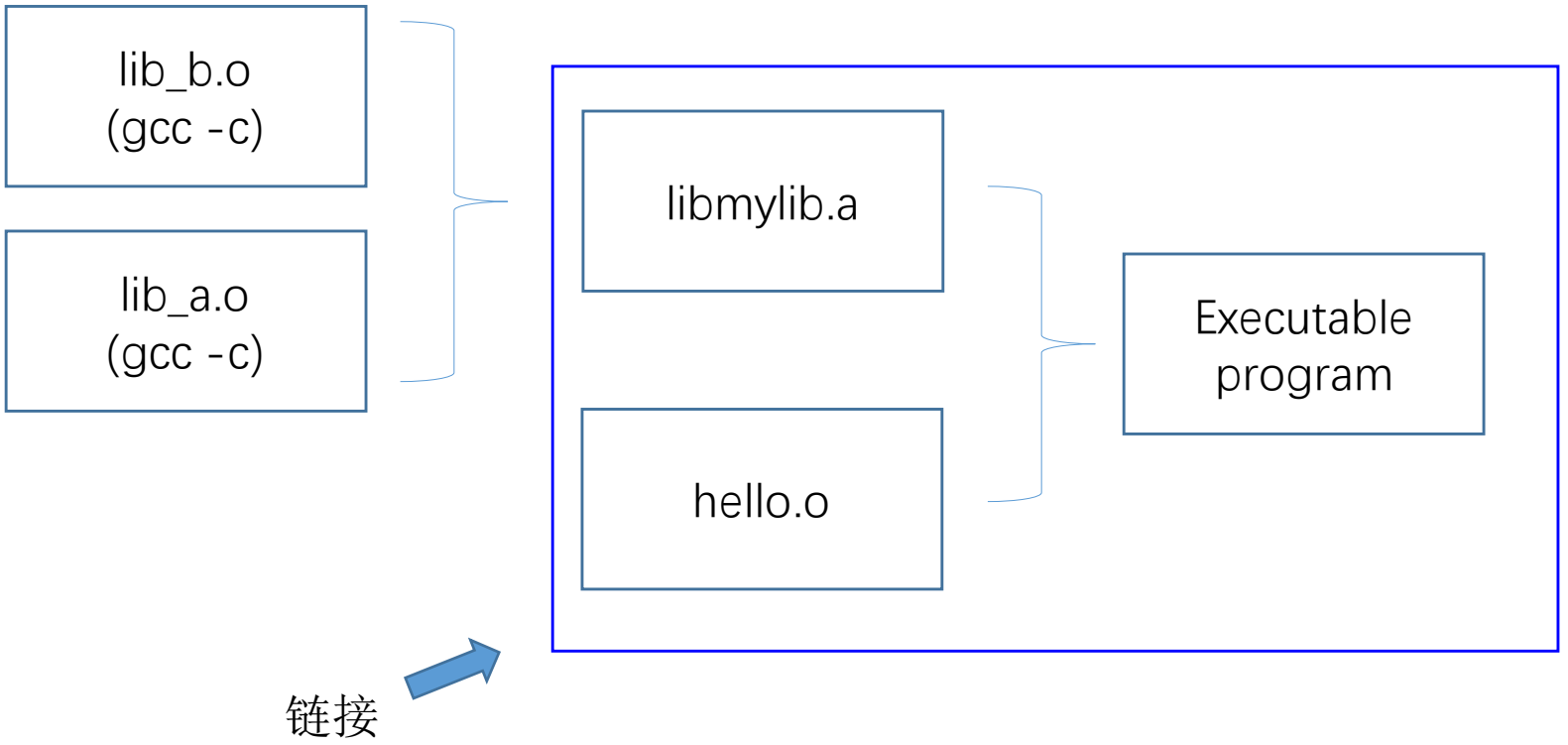
静态链接、动态链接

刘宇翔

2019年10月

静态链接



静态链接是指在编译阶段直接把静态库加入到可执行文件中去，这样可执行文件会比较大。而**动态链接**则是指链接阶段仅仅只加入一些描述信息，而程序执行时再从系统中把相应动态库加载到内存中去。

静态链接

test.c

```c
extern void hello_other_lib(int*, int*);
extern int share;
int main(){
    int a = 1;
    hello_other_lib(&a, &share);
    return 0;
}
```

my_lib.c

```c
int share = 100;
void hello_other_lib(int* a, int* b){
    int c = *a;
    *a = *b;
    *b = c;
}
```

1. 使用gcc –c命令分别得到test.o以及my_lib.o
3. 使用ar命令生成libmylib.a或直接使用gcc生成最终可执行文件。
4. 使用ld(或gcc) –L./ -lmylib通过静态链接库mylib以及test.o生成最后的可执行文件

```
gcc -c my_lib.c -o my_lib.o
gcc -c test.c -o test.o
```

```
ar crv libmylib.a my_lib.o
ld(gcc) -o test test.o -L./ -lmylib
```

生成静态库
(库名为liba.a则在链接时为-la)

```
gcc -o test test.o my_lib.o
```

直接链接

静态链接

使用objdump –h查看test.o、my_lib.o(或libmylib.a)以及最终的可执行文件的所有节信息。

关注text字段和data字段的大小

```
my_lib.o:     file format elf64-x86-64

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text         0000002c  0000000000000000  0000000000000000  00000040  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  0000000000000000  0000000000000000  0000006c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  0000000000000000  0000000000000000  00000070  2**0
                  ALLOC
  3 .comment      0000002e  0000000000000000  0000000000000000  00000070  2**0
                  CONTENTS, READONLY
  4 .note.GNU-stack 00000000  0000000000000000  0000000000000000  0000009e  2**0
                  CONTENTS, READONLY
  5 .eh_frame     00000038  0000000000000000  0000000000000000  000000a0  2**3
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
```

```
objdump -h ${file_name}|grep -A1 -E "\.text|\.data|\.rodata"
```

```
========================test.o=========================
  0 .text         00000027  0000000000000000  0000000000000000  00000040  2**0
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  0000000000000000  0000000000000000  00000067  2**0
                  CONTENTS, ALLOC, LOAD, DATA
========================my_lib.o=======================
  0 .text         0000002c  0000000000000000  0000000000000000  00000040  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  0000000000000000  0000000000000000  0000006c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
========================test===========================
  0 .text         00000053  00000000004000e8  00000000004000e8  000000e8  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
--
  2 .data         00000004  0000000000601000  0000000000601000  00001000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
```

静态链接

链接时发生了什么？

objdump –r –d test.o

```
test.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <main>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   48 83 ec 10             sub    $0x10,%rsp
   8:   c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%rbp)
   f:   48 8d 45 fc             lea    -0x4(%rbp),%rax
  13:   be 00 00 00 00          mov    $0x0,%esi
                        14: R_X86_64_32 share
  18:   48 89 c7                mov    %rax,%rdi
  1b:   e8 00 00 00 00          callq  20 <main+0x20>
                        1c: R_X86_64_PC32       hello_other_lib-0x4
  20:   b8 00 00 00 00          mov    $0x0,%eax
  25:   c9                      leaveq
  26:   c3                      retq
```

Executable and Linkable Format

1. 空间与地址分配
2. 符号解析和重定位

objdump –r –d test

```
test:     file format elf64-x86-64


Disassembly of section .text:

00000000004000e8 <main>:
  4000e8:       55                      push   %rbp
  4000e9:       48 89 e5                mov    %rsp,%rbp
  4000ec:       48 83 ec 10             sub    $0x10,%rsp
  4000f0:       c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%rbp)
  4000f7:       48 8d 45 fc             lea    -0x4(%rbp),%rax
  4000fb:       be 00 10 60 00          mov    $0x601000,%esi
  400100:       48 89 c7                mov    %rax,%rdi
  400103:       e8 07 00 00 00          callq  40010f <hello_other_lib>
  400108:       b8 00 00 00 00          mov    $0x0,%eax
  40010d:       c9                      leaveq
  40010e:       c3                      retq

000000000040010f <hello_other_lib>:
  40010f:       55                      push   %rbp
  400110:       48 89 e5                mov    %rsp,%rbp
  400113:       48 89 7d e8             mov    %rdi,-0x18(%rbp)
  400117:       48 89 75 e0             mov    %rsi,-0x20(%rbp)
  40011b:       48 8b 45 e8             mov    -0x18(%rbp),%rax
  40011f:       8b 00                   mov    (%rax),%eax
  400121:       89 45 fc                mov    %eax,-0x4(%rbp)
  400124:       48 8b 45 e0             mov    -0x20(%rbp),%rax
  400128:       8b 10                   mov    (%rax),%edx
  40012a:       48 8b 45 e8             mov    -0x18(%rbp),%rax
  40012e:       89 10                   mov    %edx,(%rax)
  400130:       48 8b 45 e0             mov    -0x20(%rbp),%rax
  400134:       8b 55 fc                mov    -0x4(%rbp),%edx
  400137:       89 10                   mov    %edx,(%rax)
  400139:       5d                      pop    %rbp
  40013a:       c3                      retq
```

静态链接

objdump –t my_lib.o

```
my_lib.o:     file format elf64-x86-64

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 my_lib.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .note.GNU-stack        0000000000000000 .note.GNU-stack
0000000000000000 l    d  .eh_frame      0000000000000000 .eh_frame
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000000000 g     O .data  0000000000000004 share
0000000000000000 g     F .text  000000000000002c hello_other_lib
```

objdump –t test.o

```
test.o:     file format elf64-x86-64

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 test.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .note.GNU-stack        0000000000000000 .note.GNU-stack
0000000000000000 l    d  .eh_frame      0000000000000000 .eh_frame
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000000000 g     F .text  0000000000000027 main
0000000000000000         *UND*  0000000000000000 share
0000000000000000         *UND*  0000000000000000 hello_other_lib
```

动态链接

my_lib.c | my_lib.h | Makefile.dynamic | dynamic_a.c | dynamic_b.c

```c
#include<stdio.h>
#include"my_lib.h"

char* dynamic_lib_name = "MY_DYNAMIC_LIB";
int global_num = 0;
void hello_other_lib(const char* name, const char* yourname){
    printf("Hello dynamic lib, I'm %s, and your name is %s, global num is %d\n", name, yourname, global_num);
}
```

```c
#ifndef MY_LIB_H
#define MY_LIB_H
void hello_other_lib(const char* name, const char* yourname);
extern char* dynamic_lib_name;
extern int global_num;
#endif
```

```
gcc -fPIC -c -g my_lib.c -o my_lib.o
gcc -shared my_lib.o -o libd.so
gcc -c -g dynamic_a.c -o da.o
gcc -c -g dynamic_b.c -o db.o
gcc -o da da.o -L./ -ld
gcc -o db db.o -L./ -ld
```

my_lib.c | my_lib.h | Makefile.dynamic | dynamic_a.c

```c
#include"my_lib.h"

int main(){
    int a = 5;
    hello_other_lib("A", dynamic_lib_name);
    global_num = 2;
    hello_other_lib("A", dynamic_lib_name);
    getchar();
    return 0;
}
```

动态链接vs静态链接

1. “动态”是如何在库加载和函数调用时体现？

2. 动态连接器如何工作？

动态链接

```
readelf -S da|grep "bss"
  [25] .bss              NOBITS           0000000000601040  0000103c
readelf --relocs da

Relocation section '.rela.dyn' at offset 0x560 contains 3 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000600ff8  000400000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0
000000601040  000e00000005 R_X86_64_COPY     0000000000601040 dynamic_lib_name + 0
000000601048  000a00000005 R_X86_64_COPY     0000000000601048 global_num + 0

Relocation section '.rela.plt' at offset 0x5a8 contains 4 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000601018  000200000007 R_X86_64_JUMP_SLO 0000000000000000 __libc_start_main + 0
000000601020  000300000007 R_X86_64_JUMP_SLO 0000000000000000 getchar + 0
000000601028  000400000007 R_X86_64_JUMP_SLO 0000000000000000 __gmon_start__ + 0
000000601030  000500000007 R_X86_64_JUMP_SLO 0000000000000000 hello_other_lib + 0
```

```
(gdb) x/1xw 0x601030
0x601030 <hello_other_lib@got.plt>:     0x00400676
```

```
(gdb) x/dw 0x601040
0x601040 <dynamic_lib_name>:    0
(gdb) x/dw 0x601048
0x601048 <global_num>:  0
```

```
Disassembly of section .plt:

0000000000400630 <__libc_start_main@plt-0x10>:
  400630:       ff 35 d2 09 20 00       pushq  0x2009d2(%rip)        # 601008 <_GLOBAL_OFFSET_TABLE_+0x8>
  400636:       ff 25 d4 09 20 00       jmpq   *0x2009d4(%rip)        # 601010 <_GLOBAL_OFFSET_TABLE_+0x10>
  40063c:       0f 1f 40 00             nopl   0x0(%rax)

0000000000400640 <__libc_start_main@plt>:
  400640:       ff 25 d2 09 20 00       jmpq   *0x2009d2(%rip)        # 601018 <_GLOBAL_OFFSET_TABLE_+0x18>
  400646:       68 00 00 00 00          pushq  $0x0
  40064b:       e9 e0 ff ff ff          jmpq   400630 <_init+0x28>

0000000000400650 <getchar@plt>:
  400650:       ff 25 ca 09 20 00       jmpq   *0x2009ca(%rip)        # 601020 <_GLOBAL_OFFSET_TABLE_+0x20>
  400656:       68 01 00 00 00          pushq  $0x1
  40065b:       e9 d0 ff ff ff          jmpq   400630 <_init+0x28>

0000000000400660 <__gmon_start__@plt>:
  400660:       ff 25 c2 09 20 00       jmpq   *0x2009c2(%rip)        # 601028 <_GLOBAL_OFFSET_TABLE_+0x28>
  400666:       68 02 00 00 00          pushq  $0x2
  40066b:       e9 c0 ff ff ff          jmpq   400630 <_init+0x28>

0000000000400670 <hello_other_lib@plt>:
  400670:       ff 25 ba 09 20 00       jmpq   *0x2009ba(%rip)        # 601030 <_GLOBAL_OFFSET_TABLE_+0x30>
  400676:       68 03 00 00 00          pushq  $0x3
  40067b:       e9 b0 ff ff ff          jmpq   400630 <_init+0x28>
```

```
(gdb) x/4i 0x400670
   0x400670 <hello_other_lib@plt>:      jmpq   *0x2009ba(%rip)        # 0x601030 <hello_other_lib@got.plt>
   0x400676 <hello_other_lib@plt+6>:    pushq  $0x3
   0x40067b <hello_other_lib@plt+11>:   jmpq   0x400630
```

动态链接

```
(gdb) c
Continuing.
Hardware watchpoint 4: -location *0x601048

Old value = 0
New value = 10
0x00007ffff7df5fd9 in memcpy () from /lib64/ld-linux-x86-64.so.2
(gdb) where
#0  0x00007ffff7df5fd9 in memcpy () from /lib64/ld-linux-x86-64.so.2
#1  0x00007ffff7de7f45 in _dl_relocate_object () from /lib64/ld-linux-x86-64.so.2
#2  0x00007ffff7ddfdea in dl_main () from /lib64/ld-linux-x86-64.so.2
#3  0x00007ffff7df31a6 in _dl_sysdep_start () from /lib64/ld-linux-x86-64.so.2
#4  0x00007ffff7dddbc1 in _dl_start () from /lib64/ld-linux-x86-64.so.2
#5  0x00007ffff7ddd178 in _start () from /lib64/ld-linux-x86-64.so.2
#6  0x0000000000000001 in ?? ()
#7  0x00007fffffffe827 in ?? ()
#8  0x0000000000000000 in ?? ()
```

```
Breakpoint 2, 0x0000000000400676 in hello_other_lib@plt ()
(gdb) c
Continuing.
Hardware watchpoint 1: -location *0x601030

Old value = 4195958
New value = -138565803
0x00007ffff7dea982 in _dl_fixup () from /lib64/ld-linux-x86-64.so.2
(gdb) where
#0  0x00007ffff7dea982 in _dl_fixup () from /lib64/ld-linux-x86-64.so.2
#1  0x00007ffff7df1430 in _dl_runtime_resolve () from /lib64/ld-linux-x86-64.so.2
#2  0x0000000000400790 in main () at dynamic_a.c:5
```

```
(gdb) x/1xw 0x601030
0x601030 <hello_other_lib@got.plt>:     0xf7bda755
```

动态链接

	--->	_start() 进程启动例程	--->	main() 开始执行	--->	d_func() 第一次调用	--->
Dynamic var	赋初值0	加载动态库中变量	动态库中设置的初始值	/	/	/	/
Dynamic func	未载入内存表中地址指向下一条指令	/	/	/	/	执行下一条指令即加载该动态库函数的地址并更改.got.plt表中的值	通过修正后的.got.plt表中的值跳转到对应的地址执行函数

动态链接

可能会用到的命令

ldd: 查看引用的动态库的链接和名字

objdump和readelf：查看目标代码，查看各节地址和符号表等信息

gdb:调试，查看运行时地址等信息

cat /proc/pid/maps: 查看内存映像，其中pid为进程id。可以看到是否正确加载到所需要的动态库以及程序的内存分布。

Thanks !

Concat:784980667@qq.com