操作系统实验

Linker & Loaders

(Based on practice)

刘璟 ljing12@software.nju.edu.cn

2015.4.17

Outline

- static linking
- dynamic linking
- Something about loader

Static linking

add.asm | func.asm | call_add.c | factorial.asm | a.c | b.c

```c
//to explain the process of static link
//a.c
extern int shared;
int main(){
    int a=100;
    swap(&a,&shared);
}
```

add.asm | func.asm | call_add.c | factorial.asm | a.c | b.c

```c
//to explain the process of static link
//a.c
extern int shared;
int main(){
    int a=100;
    swap(&a,&shared);
}
```

add.asm | func.asm | call_add.c | factorial.asm | a.c | b.c

```c
//to explain the process of static link
int shared=1;

void swap(int *a,int *b){
    int c=*a;
    *a=*b;
    *b=c;
}
```

Static linking

$ objdump -h a.o

```
a.o:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000027  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  0000005c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  0000005c  2**2
                  ALLOC
  3 .comment      0000002b  00000000  00000000  0000005c  2**0
                  CONTENTS, READONLY
  4 .note.GNU-stack 00000000  00000000  00000000  00000087  2**0
                  CONTENTS, READONLY
  5 .eh_frame     00000038  00000000  00000000  00000088  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
```

Static linking

$ objdump -h b.o

```
b.o:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000022  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  00000000  00000000  00000058  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  0000005c  2**2
                  ALLOC
  3 .comment      0000002b  00000000  00000000  0000005c  2**0
                  CONTENTS, READONLY
  4 .note.GNU-stack 00000000  00000000  00000000  00000087  2**0
                  CONTENTS, READONLY
  5 .eh_frame     00000038  00000000  00000000  00000088  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
```

Static linking

```
$ ld a.o b.o -e main -o ab
$ objdump -h ab
```

```
ab:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000004a  08048094  08048094  00000094  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .eh_frame     00000058  080480e0  080480e0  000000e0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .data         00000004  08049138  08049138  00000138  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .comment      0000002a  00000000  00000000  0000013c  2**0
                  CONTENTS, READONLY
```

What happened?

The similar segments merge in the output file.

Static linking

$objdump -d a.o

```
a.o:     file format elf32-i386


Disassembly of section .text:

00000000 <main>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 e4 f0                and    $0xfffffff0,%esp
   6:   83 ec 20                sub    $0x20,%esp
   9:   c7 44 24 1c 64 00 00    movl   $0x64,0x1c(%esp)
  10:   00
  11:   c7 44 24 04 00 00 00    movl   $0x0,0x4(%esp)
  18:   00
  19:   8d 44 24 1c             lea    0x1c(%esp),%eax
  1d:   89 04 24                mov    %eax,(%esp)
  20:   e8 fc ff ff ff          call   21 <main+0x21>
  25:   c9                      leave
  26:   c3                      ret
```

What happened in the two boxes?

Static linking

$objdump -d ab

```
08048094 <main>:
 8048094:       55                      push   %ebp
 8048095:       89 e5                   mov    %esp,%ebp
 8048097:       83 e4 f0                and    $0xfffffff0,%esp
 804809a:       83 ec 20                sub    $0x20,%esp
 804809d:       c7 44 24 1c 64 00 00    movl   $0x64,0x1c(%esp)
 80480a4:       00
 80480a5:       c7 44 24 04 38 91 04    movl   $0x8049138,0x4(%esp)
 80480ac:       08
 80480ad:       8d 44 24 1c             lea    0x1c(%esp),%eax
 80480b1:       89 04 24                mov    %eax,(%esp)
 80480b4:       e8 03 00 00 00          call   80480bc <swap>
 80480b9:       c9                      leave
 80480ba:       c3                      ret
 80480bb:       90                      nop

080480bc <swap>:
 80480bc:       55                      push   %ebp
 80480bd:       89 e5                   mov    %esp,%ebp
 80480bf:       83 ec 10                sub    $0x10,%esp
 80480c2:       8b 45 08                mov    0x8(%ebp),%eax
 80480c5:       8b 00                   mov    (%eax),%eax
 80480c7:       89 45 fc                mov    %eax,-0x4(%ebp)
 80480ca:       8b 45 0c                mov    0xc(%ebp),%eax
 80480cd:       8b 10                   mov    (%eax),%edx
 80480cf:       8b 45 08                mov    0x8(%ebp),%eax
 80480d2:       89 10                   mov    %edx,(%eax)
 80480d4:       8b 45 0c                mov    0xc(%ebp),%eax
 80480d7:       8b 55 fc                mov    -0x4(%ebp),%edx
 80480da:       89 10                   mov    %edx,(%eax)
 80480dc:       c9                      leave
 80480dd:       c3                      ret
```

relocation

Static linking

How to relocate? Relocation Table & Symbol Table

$objdump -r a.o

```
a.o:     file format elf32-i386

RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE              VALUE
00000015 R_386_32          shared
00000021 R_386_PC32        swap


RELOCATION RECORDS FOR [.eh_frame]:
OFFSET   TYPE              VALUE
00000020 R_386_PC32        .text
```

$objdump -s a.o

```
Symbol table '.symtab' contains 11 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS a.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    4
     5: 00000000     0 SECTION LOCAL  DEFAULT    6
     6: 00000000     0 SECTION LOCAL  DEFAULT    7
     7: 00000000     0 SECTION LOCAL  DEFAULT    5
     8: 00000000    39 FUNC    GLOBAL DEFAULT    1 main
     9: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND shared
    10: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND swap
```

Dynamic linking:

The advantage of dynamic linking:

- Dynamically linked shared libraries are easier to create than static linked shared libraries.
- Dynamically linked shared libraries are easier to update than static linked shared libraries.
- The semantics of dynamically linked shared libraries can be much closer to those of unshared libraries.
- Dynamic linking permits a program to load and unload routines at runtime, a facility that can otherwise be very difficult to provide.

Dynamic linking:

Steps:

- Starting the dynamic linker
  - bootstrap
- Finding the libraries
- Shared library initialization
  - relocation & initialization

Dynamic linking:

add.asm | func.asm | call_add.c | factorial.asm | a.c | b.c | pa.c

```c
//explain dynamic linking
//pa.c
#include "Lib.h"

int main(){
    func(1);
    return 0;
}
```

add.asm | func.asm | call_add.c | factorial.asm | a.c | b.c | pa.c | pb.c

```c
//explain the dynamic linking
//pb.c

int main(){
    func(2);
    return 0;
}
```

add.asm | func.asm | call_add.c | factorial.asm | a.c | b.c | pa.c | pb.c

```c
//explain dynamic linking
//Lib.h
#ifndef LIB_H
#define LIB_H
void func(int i);
#endif
```

add.asm | func.asm | call_add.c | factorial.asm | a.c | b.c | pa.c | pb

```c
//explain the dynamic linking
//Lib.c

#include <stdio.h>
void func(int i){
    printf("printing from Lib.so%d\n",i);
    sleep(-1);
}
```

Dynamic linking:

```
$ gcc -fPIC -shared -o Lib.so lib.c
$ gcc -o pa pa.c ./Lib.so
$ gcc -o pb pb.c ./Lib.so
```

What if run pa and pb simultaneously?

Lib.so will only be loaded to memory once

```
$ ./pa
$ ./pb
$ ps -e
```

Dynamic linking:

```
[redacted]~$ cat /proc/5846/maps
08048000-08049000 r-xp 00000000 08:01 134551    [redacted]/workspace
/asm/prac1/pa
08049000-0804a000 r--p 00000000 08:01 134551    [redacted]/workspace
/asm/prac1/pa
0804a000-0804b000 rw-p 00001000 08:01 134551    [redacted]/workspace
/asm/prac1/pa
b7579000-b757b000 rw-p 00000000 00:00 0
b757b000-b771f000 r-xp 00000000 08:01 1055162    /lib/i386-linux-gnu/libc-2.15.s
o
b771f000-b7721000 r--p 001a4000 08:01 1055162    /lib/i386-linux-gnu/libc-2.15.s
o
b7721000-b7722000 rw-p 001a6000 08:01 1055162    /lib/i386-linux-gnu/libc-2.15.s
o
b7722000-b7725000 rw-p 00000000 00:00 0
b7734000-b7735000 rw-p 00000000 00:00 0
b7735000-b7736000 r-xp 00000000 08:01 134548    [redacted]/workspace
/asm/prac1/Lib.so
b7736000-b7737000 r--p 00000000 08:01 134548    [redacted]/workspace
/asm/prac1/Lib.so
b7737000-b7738000 rw-p 00001000 08:01 134548    [redacted]/workspace
/asm/prac1/Lib.so
b7738000-b773a000 rw-p 00000000 00:00 0
b773a000-b773b000 r-xp 00000000 00:00 0          [vdso]
b773b000-b775b000 r-xp 00000000 08:01 1055152    /lib/i386-linux-gnu/ld-2.15.so
b775b000-b775c000 r--p 0001f000 08:01 1055152    /lib/i386-linux-gnu/ld-2.15.so
b775c000-b775d000 rw-p 00020000 08:01 1055152    /lib/i386-linux-gnu/ld-2.15.so
bf9ad000-bf9ce000 rw-p 00000000 00:00 0          [stack]
```

Dynamic linking:

How to use your codes and experimental results to understand the process of dynamic linking?

- Waiting for you!
- Bonus!
- Good luck!

Hints: use command 'readelf', 'objdump', 'ps' and 'cat' properly.

Thanks~ O(∩_∩)O !