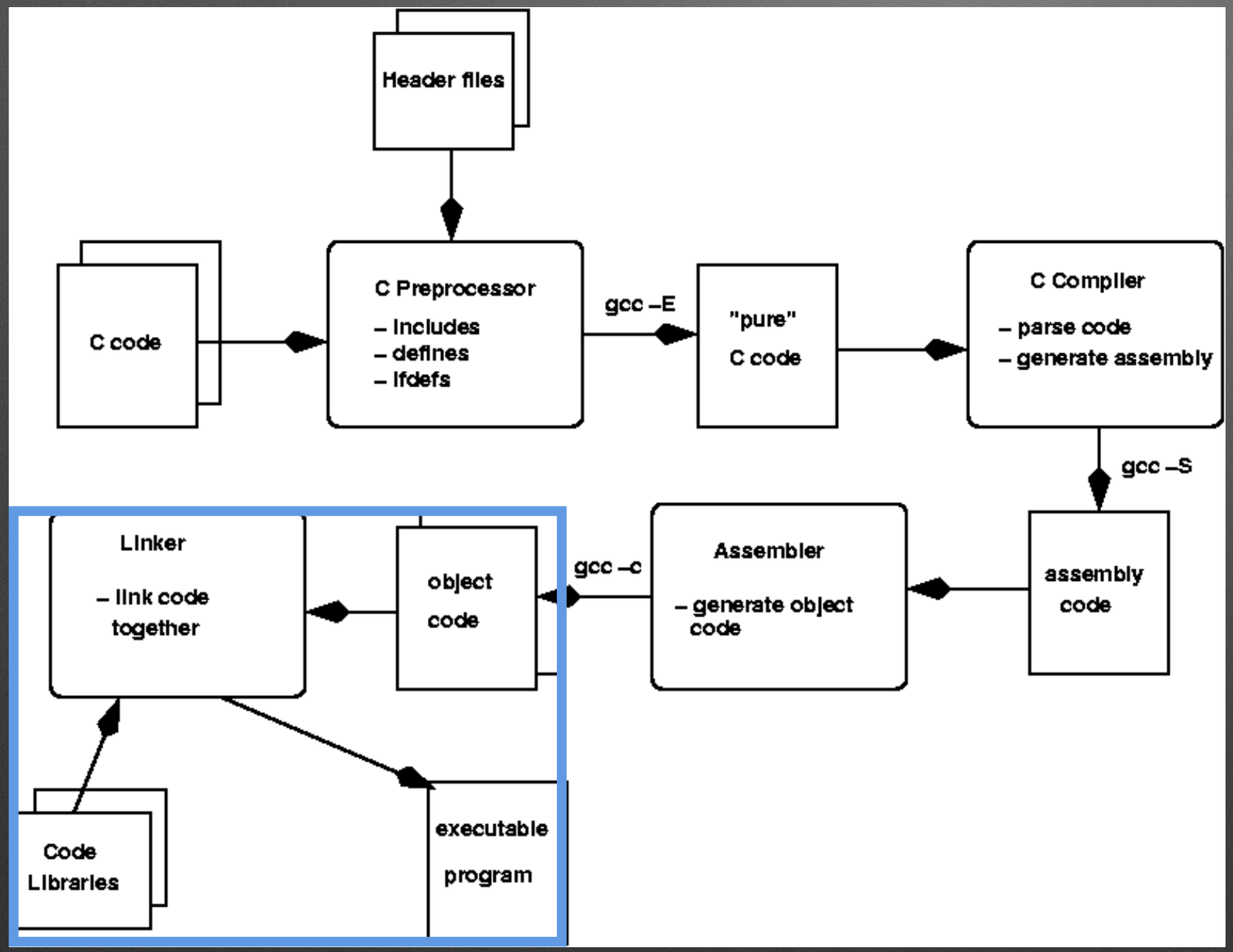
Linker & Loader

static VS dynamic linking

gcc编译过程



编译参数

-E: 仅预处理
-S: 预处理+编译
-c: 预处理+编译+汇编

Static Linking

a.c

```c
extern int shared;
int main(){
    int a = 100;
    swap(&a, &shared);
}
```

b.c

```c
int shared = 1;

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

编译链接

1. 使用gcc -c命令分别得到a.o, b.o (-fno-stack-protector)
2. 使用ld命令链接得到可执行文件

Static Linking

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000034  00000000  00000000  00000034  2**0
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000068  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000068  2**0
                  ALLOC
  3 .comment      00000025  00000000  00000000  00000068  2**0
                  CONTENTS, READONLY
  4 .note.GNU-stack 00000000  00000000  00000000  0000008d  2**0
                  CONTENTS, READONLY
  5 .eh_frame     00000044  00000000  00000000  00000090  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
```

发生了什么?

1. objdump -h a.o

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000022  00000000  00000000  00000034  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  00000000  00000000  00000058  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  0000005c  2**0
                  ALLOC
  3 .comment      00000025  00000000  00000000  0000005c  2**0
                  CONTENTS, READONLY
  4 .note.GNU-stack 00000000  00000000  00000000  00000081  2**0
                  CONTENTS, READONLY
  5 .eh_frame     00000038  00000000  00000000  00000084  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
yongshangwu@ubuntu:~/Documents/lab2/link/sl$
```

2. objdump -h b.o

3. objdump -h ab

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000056  08048094  08048094  00000094  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .eh_frame     00000064  080480ec  080480ec  000000ec  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .data         00000004  08049150  08049150  00000150  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .comment      00000024  00000000  00000000  00000154  2**0
                  CONTENTS, READONLY
yongshangwu@ubuntu:~/Documents/lab2/link/sl$
```

Static Linking

发生了什么?

1. objdump -d a.o
2. objdump -d ab

```
1  extern int shared;
2
3  int main(){
4      int a = 100;
5      swap(&a, &shared);
6  }
```

```
Disassembly of section .text:

08048094 <main>:
 8048094:       8d 4c 24 04             lea    0x4(%esp),%ecx
 8048098:       83 e4 f0                and    $0xfffffff0,%esp
 804809b:       ff 71 fc                pushl  -0x4(%ecx)
 804809e:       55                      push   %ebp
 804809f:       89 e5                   mov    %esp,%ebp
 80480a1:       51                      push   %ecx
 80480a2:       83 ec 14                sub    $0x14,%esp
 80480a5:       c7 45 f4 64 00 00 00    movl   $0x64,-0xc(%ebp)
 80480ac:       83 ec 08                sub    $0x8,%esp
 80480af:       68 50 91 04 08          push   $0x8049150
 80480b4:       8d 45 f4                lea    -0xc(%ebp),%eax
 80480b7:       50                      push   %eax
 80480b8:       e8 0b 00 00 00          call   80480c8 <swap>
 80480bd:       83 c4 10                add    $0x10,%esp
 80480c0:       8b 4d fc                mov    -0x4(%ebp),%ecx
 80480c3:       c9                      leave
 80480c4:       8d 61 fc                lea    -0x4(%ecx),%esp
 80480c7:       c3                      ret

080480c8 <swap>:
 80480c8:       55                      push   %ebp
 80480c9:       89 e5                   mov    %esp,%ebp
 80480cb:       83 ec 10                sub    $0x10,%esp
 80480ce:       8b 45 08                mov    0x8(%ebp),%eax
 80480d1:       8b 00                   mov    (%eax),%eax
 80480d3:       89 45 fc                mov    %eax,-0x4(%ebp)
 80480d6:       8b 45 0c                mov    0xc(%ebp),%eax
 80480d9:       8b 10                   mov    (%eax),%edx
 80480db:       8b 45 08                mov    0x8(%ebp),%eax
 80480de:       89 10                   mov    %edx,(%eax)
 80480e0:       8b 45 0c                mov    0xc(%ebp),%eax
 80480e3:       8b 55 fc                mov    -0x4(%ebp),%edx
 80480e6:       89 10                   mov    %edx,(%eax)
 80480e8:       c9                      leave
 80480e9:       c3                      ret
```

```
00000000 <main>:
 0:   8d 4c 24 04             lea    0x4(%esp),%ecx
 4:   83 e4 f0                and    $0xfffffff0,%esp
 7:   ff 71 fc                pushl  -0x4(%ecx)
 a:   55                      push   %ebp
 b:   89 e5                   mov    %esp,%ebp
 d:   51                      push   %ecx
 e:   83 ec 14                sub    $0x14,%esp
11:   c7 45 f4 64 00 00 00    movl   $0x64,-0xc(%ebp)
18:   83 ec 08                sub    $0x8,%esp
1b:   68 00 00 00 00          push   $0x0
20:   8d 45 f4                lea    -0xc(%ebp),%eax
23:   50                      push   %eax
24:   e8 fc ff ff ff          call   25 <main+0x25>
29:   83 c4 10                add    $0x10,%esp
2c:   8b 4d fc                mov    -0x4(%ebp),%ecx
2f:   c9                      leave
30:   8d 61 fc                lea    -0x4(%ecx),%esp
33:   c3                      ret
```

Static Linking

发生了什么?

1. 相同段内容合并

2. 代码重定位, VMA & LMA

Static Linking

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

RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE              VALUE
0000001c R_386_32          shared
00000025 R_386_PC32        swap


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

如何实现?

1. objdump -r a.o

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

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 a.c
00000000 l    d  .text  00000000 .text
00000000 l    d  .data  00000000 .data
00000000 l    d  .bss   00000000 .bss
00000000 l    d  .note.GNU-stack        00000000 .note.GNU-stack
00000000 l    d  .eh_frame      00000000 .eh_frame
00000000 l    d  .comment       00000000 .comment
00000000 g     F .text  00000034 main
00000000         *UND*  00000000 shared
00000000         *UND*  00000000 swap
```

2. objdump -t a.o

3. objdump -t b.o

```
SYMBOL TABLE:
00000000 l    df *ABS*  00000000 b.c
00000000 l    d  .text  00000000 .text
00000000 l    d  .data  00000000 .data
00000000 l    d  .bss   00000000 .bss
00000000 l    d  .note.GNU-stack        00000000 .note.GNU-stack
00000000 l    d  .eh_frame      00000000 .eh_frame
00000000 l    d  .comment       00000000 .comment
00000000 g     O .data  00000004 shared
00000000 g     F .text  00000022 swap
```

Dynamic Linking

优点

1. 内存和磁盘空间

2. 程序开发和发布

3. 可扩展性和可兼容性

Dynamic Linking

Lib.h

```h
#ifndef LIB_H
#define LIB_H
void func(int i);
#endif
```

Lib.c

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

pa.c

```c
#include "Lib.h"

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

pb.c

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

编译+动态链接

1. gcc -fPIC -shared -o Lib.so Lib.c
2. gcc -o pa pa.c ./Lib.so -g -F dwarf
3. gcc -o pb pb.c ./Lib.so -g -F dwarf

Dynamic Linking

你要做什么?

编写代码, 通过实验说明动态链接的过程

1. 动态链接中的段合并与重定位如何实现
2. “动态”是如何在库加载和函数调用时体现
   (延迟绑定、第一次调用、第二次调用的区别)
3. 动态链接器如何工作
   (自举、装载共享对象)

Dynamic Linking

可能会用到的命令

1. ldd: 查看引用的动态链接库
2. objdump: 查看目标代码
3. gdb: 调试
4. ps: 查看进程
5. cat: 查看内存映像

Dynamic Linking

可能有帮助的书

程序员的自我修养
——装载、链接与库

Have fun :)

contact me: wuys@smail.nju.edu.cn