Linking

Made By: Yang Yuxin & Wang Chang

编译链接过程

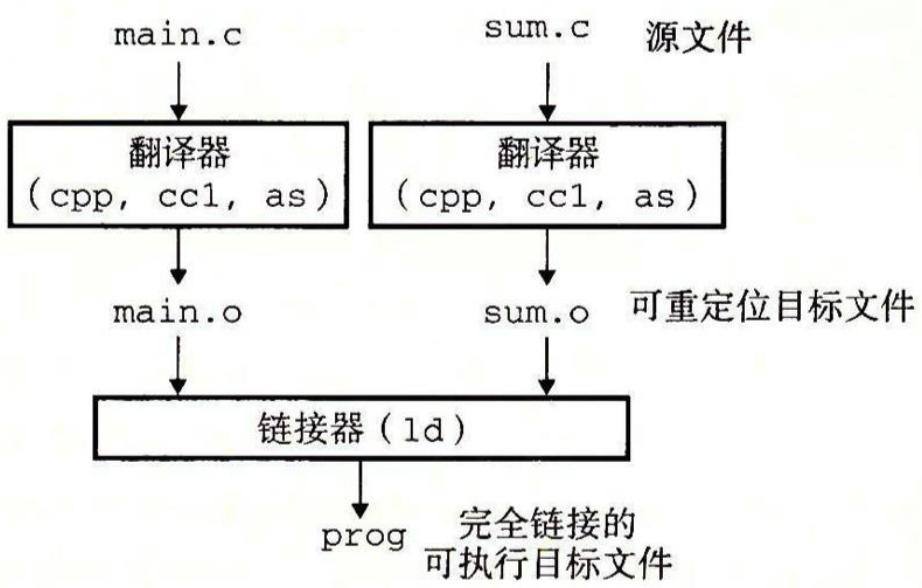


图 7-2 静态链接。链接器将可重定位目标文件组合起来，形成一个可执行目标文件 prog

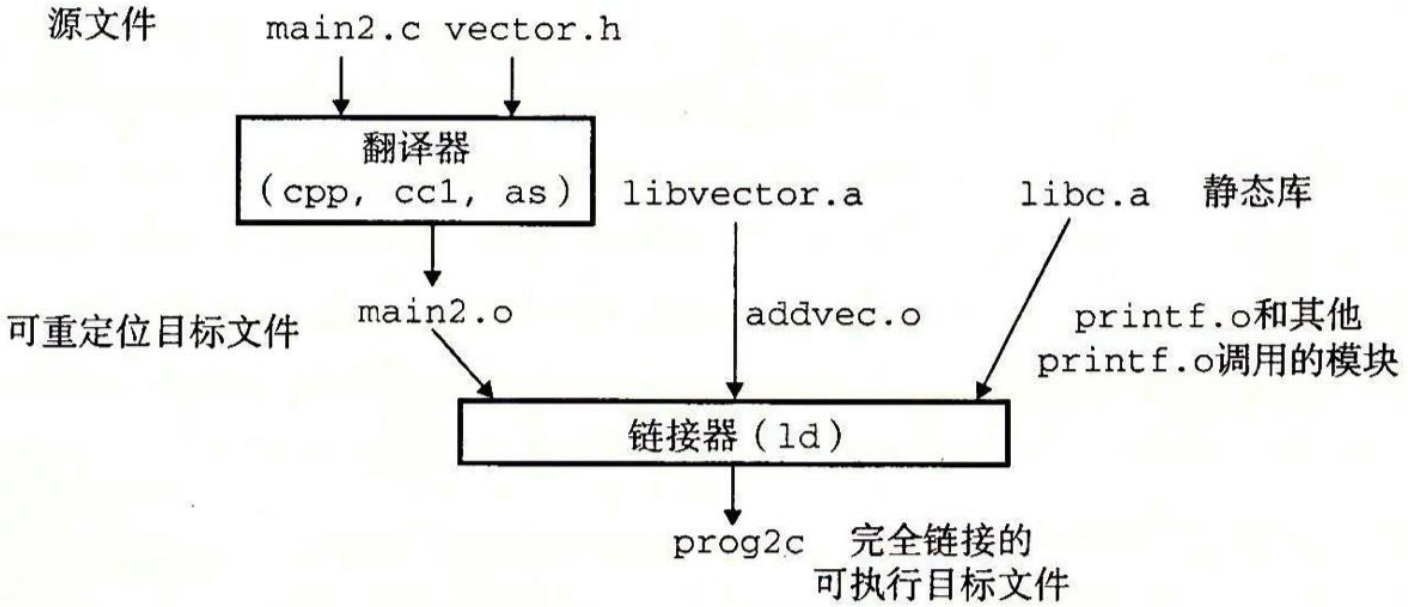


图 7-8 与静态库链接

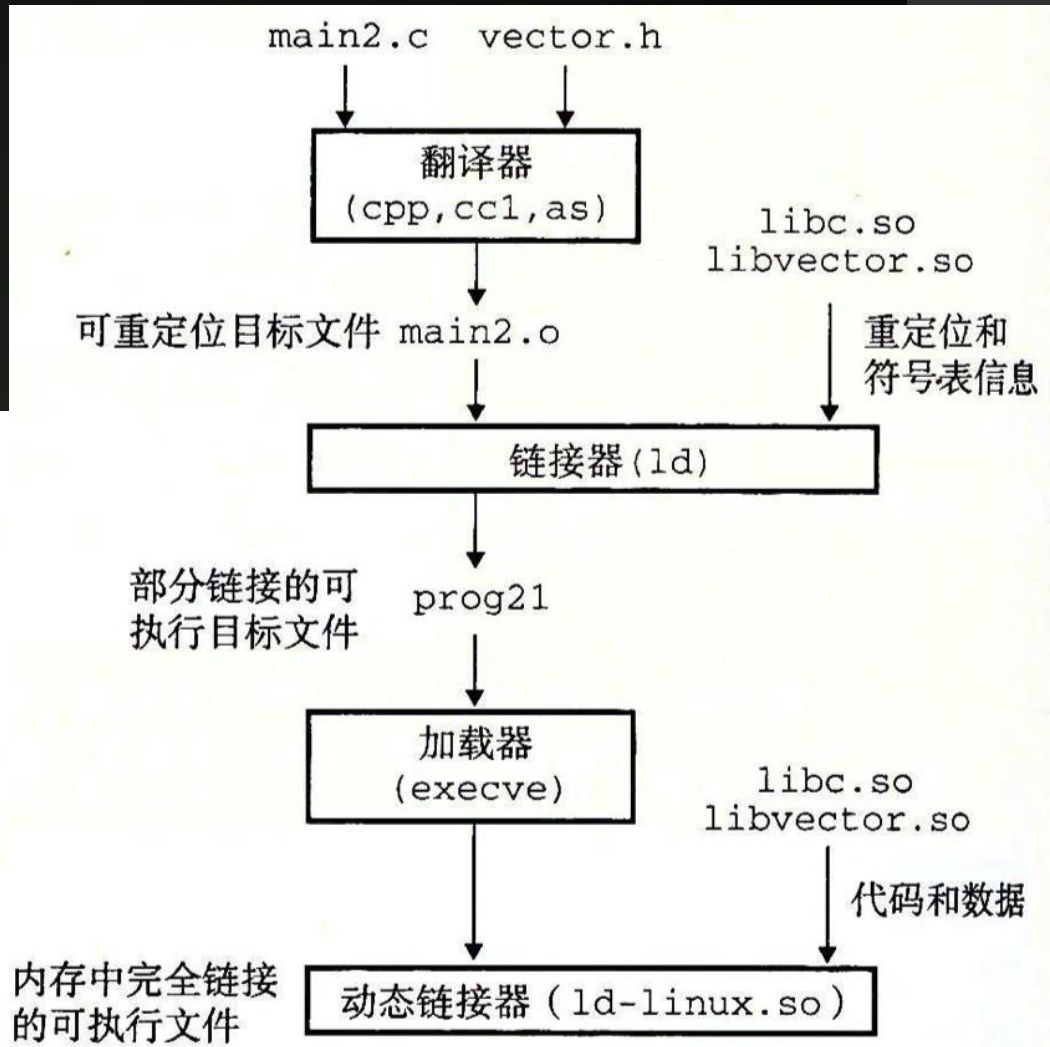


图 7-16 动态链接共享库

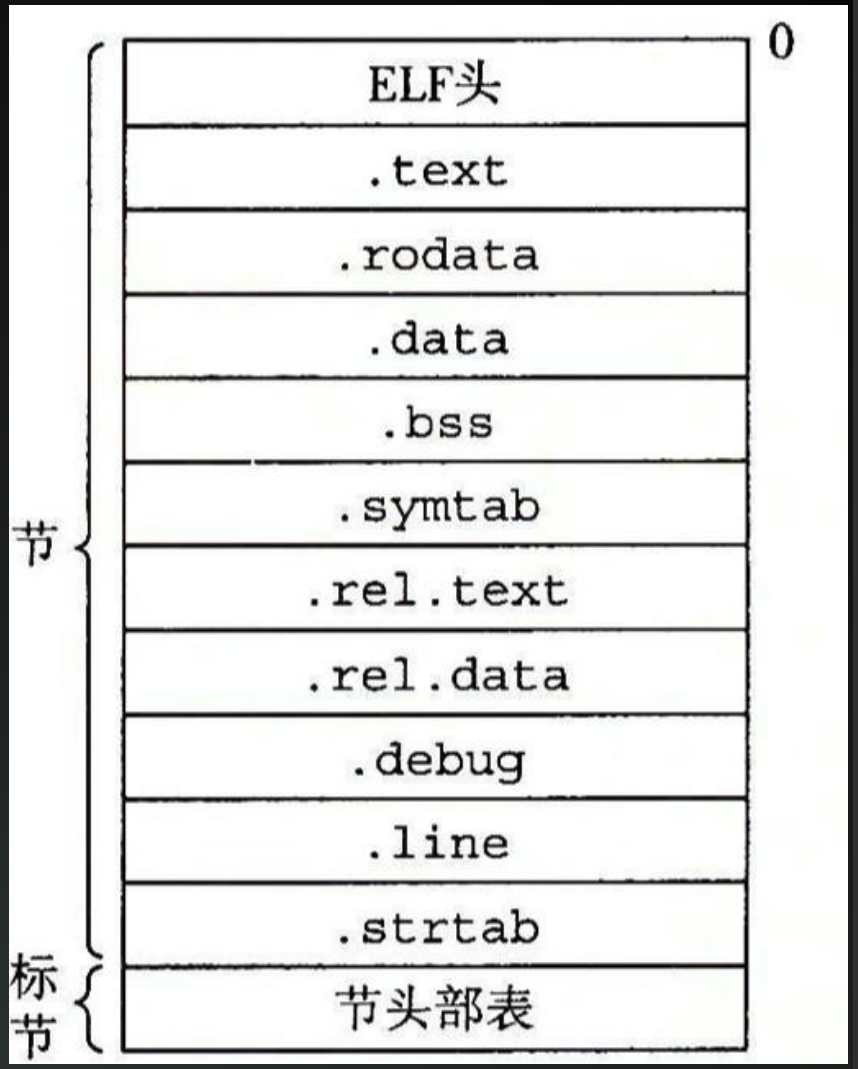
静态链接

- static linker以一组**Relocatable object file**作为输入，生成**完全链接的、可以加载和运行的可执行目标文件**。
- 两个任务：
  - 符号解析：将每个**符号引用**与一个**符号定义**关联起来。需要为符号引用选择正确的符号定义。
  - 重定位：将每个**符号定义**与一个**内存位置**关联起来。需要为符号定义选择内存位置，并修改符号引用使他们指向正确的内存位置。
- 目标文件有三种形式：
  - 可重定位目标文件
  - 可执行目标文件
  - 共享目标文件

可重定位目标文件

- 各个节的信息
- ELF头中有什么？

```
candy@ubuntu:~/ics/linking$ readelf -h main.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          1400 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         13
  Section header string table index: 12
```



```c
int global_uninit;
int global_init_0 = 0;
int global_init_1 = 1;

static int global_static_uninit;
static int global_static_init_0 = 0;
static int global_static_init_1 = 1;

int global_func_def(int x) {
  return x + 1;
}
static int global_static_func_def(int x) {
  return x + 1;
}
int global_func_decl_with_ref(int);
int global_func_decl_without_ref(int);

int main() {
  int local_uninit;
  int local_init_0 = 0;
  int local_init_1 = 1;

  static int local_static_uninit;
  static int local_static_init_0 = 0;
  static int local_static_init_1 = 1;

  global_func_decl_with_ref(3);
}
```

```
Section Headers:
  [Nr] Name              Type            Address          Offset
       Size              EntSize         Flags  Link  Info  Align
  [ 0]                   NULL            0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS        0000000000000000  00000040
       0000000000000051  0000000000000000  AX       0     0     1
  [ 2] .rela.text        RELA            0000000000000000  000004a8
       0000000000000018  0000000000000018   I      10     1     8
  [ 3] .data             PROGBITS        0000000000000000  00000094
       000000000000000c  0000000000000000  WA       0     0     4
  [ 4] .bss              NOBITS          0000000000000000  000000a0
  [ 5] .comm
  [ 6] .note
  [ 7] .note
  [ 8] .eh_f
  [ 9] .rela
  [10] .symt
  [11] .strt
  [12] .shst

Symbol table '.symtab' contains 23 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4
     5: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    4 global_static_uninit
     6: 0000000000000008     4 OBJECT  LOCAL  DEFAULT    4 global_static_init_0
     7: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 global_static_init_1
     8: 0000000000000013    19 FUNC    LOCAL  DEFAULT    1 global_static_func_def
     9: 0000000000000008     4 OBJECT  LOCAL  DEFAULT    3 local_static_init_1.1933
    10: 000000000000000c     4 OBJECT  LOCAL  DEFAULT    4 local_static_init_0.1932
    11: 0000000000000010     4 OBJECT  LOCAL  DEFAULT    4 local_static_uninit.1931
    12: 0000000000000000     0 SECTION LOCAL  DEFAULT    6
    13: 0000000000000000     0 SECTION LOCAL  DEFAULT    7
    14: 0000000000000000     0 SECTION LOCAL  DEFAULT    8
    15: 0000000000000000     0 SECTION LOCAL  DEFAULT    5
    16: 0000000000000004     4 OBJECT  GLOBAL DEFAULT  COM global_uninit
    17: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    4 global_init_0
    18: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 global_init_1
    19: 0000000000000000    19 FUNC    GLOBAL DEFAULT    1 global_func_def
    20: 0000000000000026    43 FUNC    GLOBAL DEFAULT    1 main
    21: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND _GLOBAL_OFFSET_TABLE_
    22: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND global_func_decl_with_ref
```

符号

- 符号表中包含了可重定位目标文件中**定义和引用**的符号的信息。
- 对于C语言中的定义可以分为如下几类：
  - 函数：
    - 区分静态和非静态
    - 只有函数声明没有引用不会出现在符号表中
  - 变量：
    - 区分静态和非静态
    - 只考虑全局变量和静态局部变量，非静态局部变量不会出现在符号表中
- 三种符号：
  - 全局符号：非静态函数 + 非静态全局变量
  - 外部符号：其他模块定义的全局符号
  - 局部符号：静态函数 + 静态变量（包括静态全局和静态局部变量）

符号表

- 各个field的含义
- 三个特殊的伪节：
  - ABS
  - UNDEF：比如只有声明和引用但没有定义的函数
  - COMMON：未初始化的全局变量会放在COMMON里而不是.bss里，为了之后的符号解析

code/link/elfstructs.c

```
1  typedef struct {
2      int   name;       /* String table offset */
3      char  type:4,     /* Function or data (4 bits) */
4            binding:4; /* Local or global (4 bits) */
5      char  reserved;   /* Unused */
6      short section;    /* Section header index */
7      long  value;      /* Section offset or absolute address */
8      long  size;       /* Object size in bytes */
9  } Elf64_Symbol;
```

code/link/elfstructs.c

练习

```c
#include <stdio.h>
int sum(int a, int b);
int foo(int c, int d);

static int g_A = 2;
int g_B = 3;
static int g_C;
int g_D;

int main() {
    static int j;
    static int v = 2;
    int i;
    printf("%d\n", sum(j, v));
}
```

名字	是否在符号表里	全局 or 局部	在.data	在.rodata	在.bss	在 COMMON	在.strtab
sum							
foo							
a							
g_A							
g_B							
g_C							
g_D							
j							
v							
i							
printf							
"%d\n"							

符号解析

- 将每个**符号引用**与一个**符号定义**关联起来。需要为符号引用选择正确的符号定义。
- **难**点是**多重定义的全局符号**。
- 将**全局符号**分为：
  - 强符号：定义的函数，初始化的全局变量
  - 弱符号：未初始化的全局变量

根据强弱符号的定义，Linux 链接器使用下面的规则来处理多重定义的符号名：

- 规则 1：不允许有多个同名的强符号。
- 规则 2：如果有一个强符号和多个弱符号同名，那么选择强符号。
- 规则 3：如果有多个弱符号同名，那么从这些弱符号中任意选择一个。

- **Q: 只有声明和引用没有定义的函数是什么？** 既不是强符号也不是弱符号，是外部符号，不需要使用上述规则

与静态库的链接

- 将多个相关的函数封装成一个单独的库（.a）
- 过程：
  - 维护三个集合：未解析的符号集合 U, 已解析的符号集合 D, 可重定位目标文件集合 E
  - 对于每个输入文件：
    - 如果是一个目标文件，加入 E, 更新 U D
    - 如果是一个存档文件，依次扫过每个成员目标文件，如果某个mi 定义了 U 中的符号，就将 mi 加入 E，并且更新 U D
    - 若最终 U 非空，则返回错误。否则构造出可执行文件。

重定位

- 重定位：将每个**符号定义**与一个**内存位置**关**联**起来。需要**为**符号定**义选择**内存位置，并修改符号引用使他**们**指向正确的内存位置。
  - 将来自不同文件的相同类型的**节**合并（**这**一步中确定每个**节**和符号的运行**时**地址）
  - 重定位**节**中的符号引用（使用.rel.text和.rel.data中的**重定位条目**）

code/link/elfstructs.c

```
1  typedef struct {
2      long offset;    /* Offset of the reference to relocate */
3      long type:32,   /* Relocation type */
4           symbol:32; /* Symbol table index */
5      long addend;    /* Constant part of relocation expression */
6  } Elf64_Rela;
```

code/link/elfstructs.c

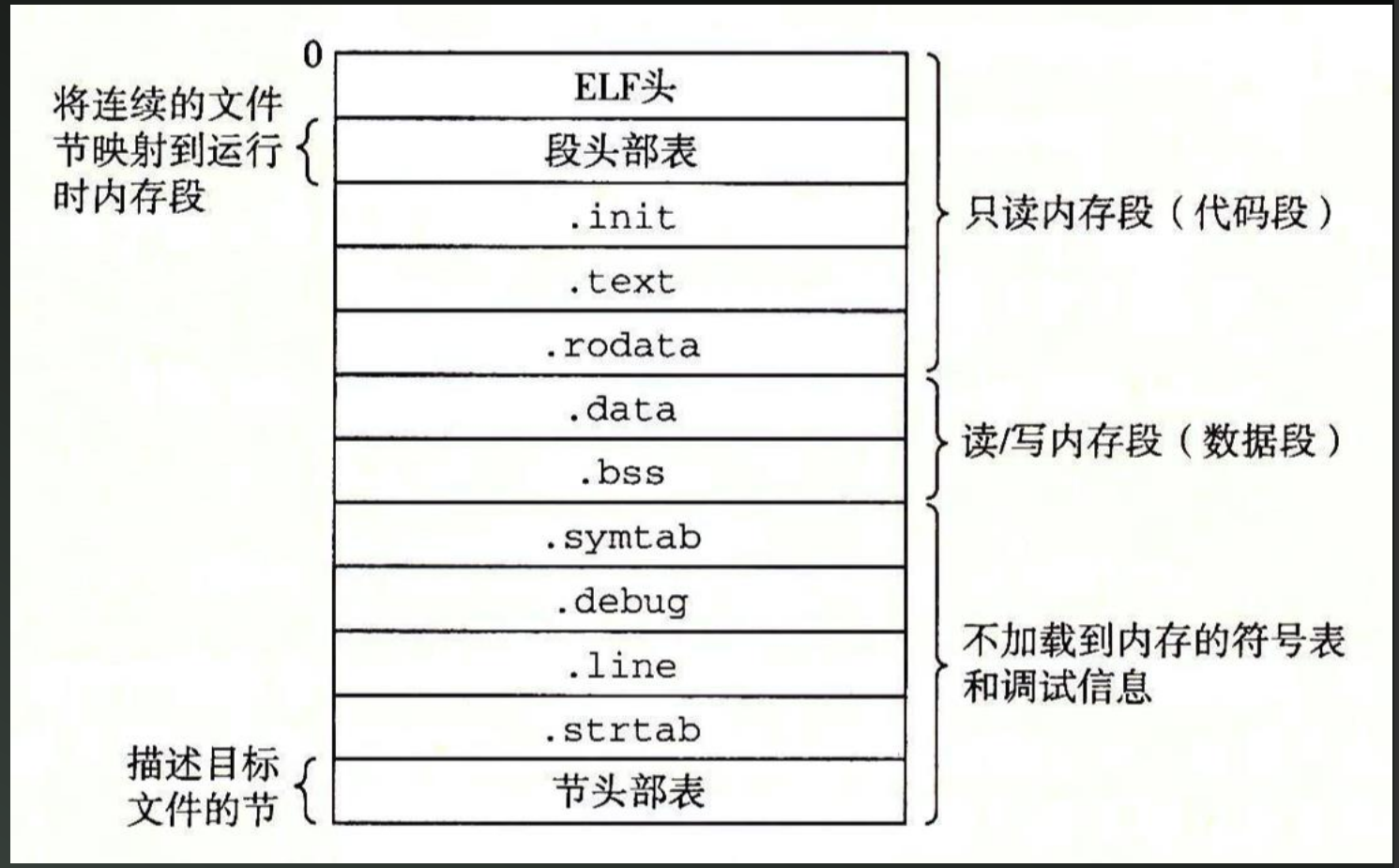
重定位符号引用的过程

- 注意掌握书上展示的例子

```
1   foreach section s {
2       foreach relocation entry r {
3           refptr = s + r.offset;  /* ptr to reference to be relocated */
4
5           /* Relocate a PC-relative reference */
6           if (r.type == R_X86_64_PC32) {
7               refaddr = ADDR(s) + r.offset; /* ref's run-time address */
8               *refptr = (unsigned) (ADDR(r.symbol) + r.addend - refaddr);
9           }
10
11          /* Relocate an absolute reference */
12          if (r.type == R_X86_64_32)
13              *refptr = (unsigned) (ADDR(r.symbol) + r.addend);
14      }
15  }
```

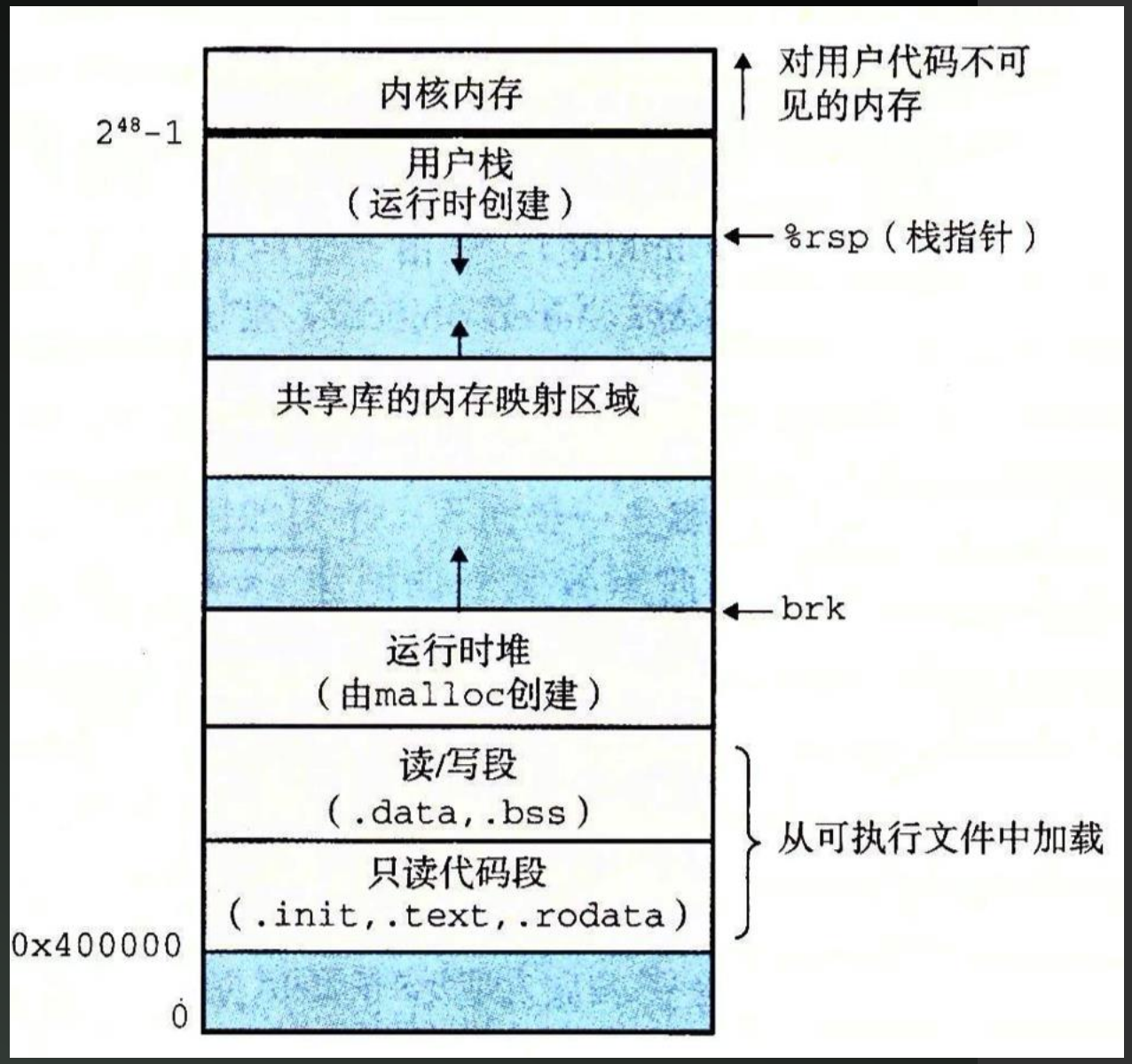
可执行目标文件

- 增加了segment header table (program header table)
- 不再需要.rel节



加载可执行目标文件

- 调用加载器（loader）
- 跳转到ELF头中记录的entry point（_start函数的地址，不是main函数）



动态链接

- Loader发现可执行目标文件中有.interp节，不会将控制给应用，而是加载和运行动态链接器。

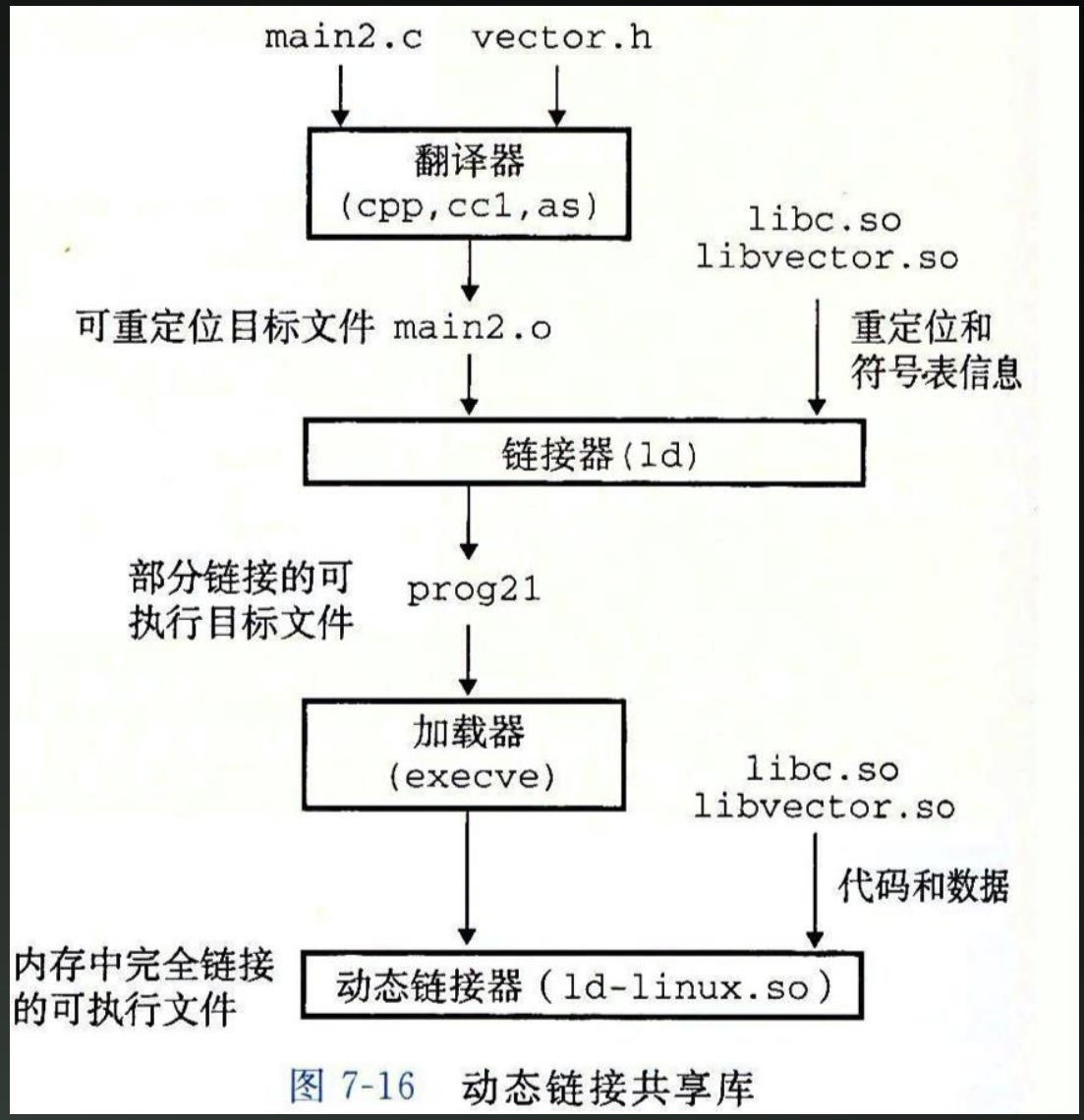


图 7-16　动态链接共享库