# Linking

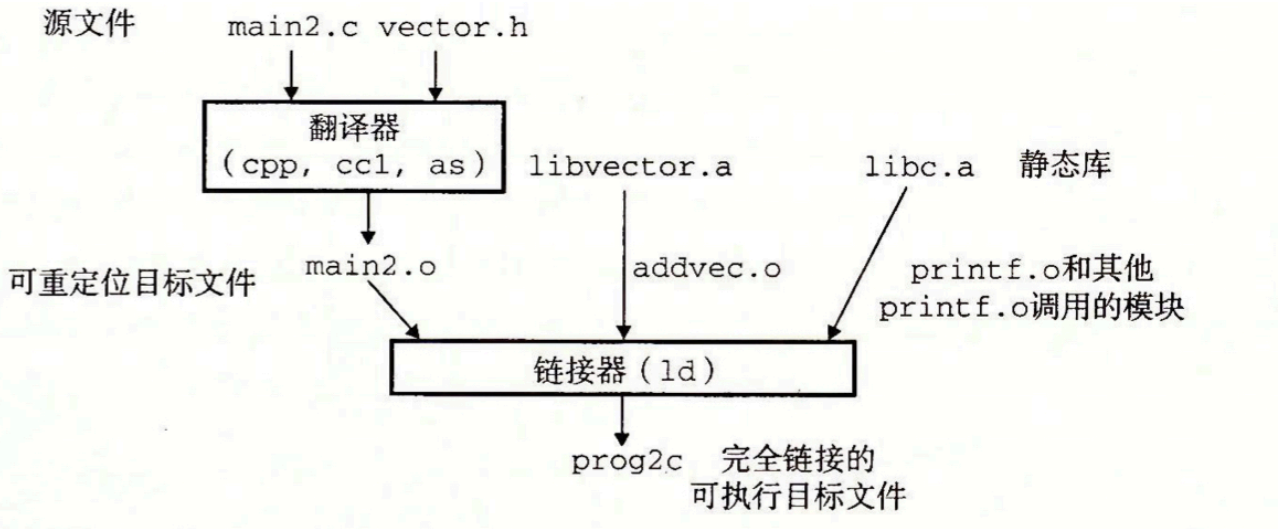
文泓宇

2019 年 11 月 14 日

- 编译器：高级语言 $\rightarrow$ 汇编语言，生成汇编代码
- 汇编器：汇编语言 $\rightarrow$ 机器语言，生成可重定位目标文件（ELF）
- 链接器（静态链接）：可执行目标文件
- 加载器

链接器需要解决什么问题？

- 合并多个文件的信息
- 计算运行时地址



编译器和汇编器做了什么？

- 机器代码

编译器和汇编器做了什么？

- 机器代码
- 数据

## 编译器和汇编器做了什么？

- 机器代码
- 数据
  - 只读数据：printf 格式串，跳转表
  - 已初始化的全局或静态变量
  - 未初始化的静态变量，或初始化为 0 的全局或静态变量
  - 未初始化的全局变量
  - 不该被重定位的符号
  - 未定义的符号
  - 局部变量

编译器和汇编器做了什么？

- 机器代码
- 数据
  - 只读数据：printf 格式串，跳转表
  - 已初始化的全局或静态变量
  - 未初始化的静态变量，或初始化为 0 的全局或静态变量
  - 未初始化的全局变量
  - 不该被重定位的符号
  - 未定义的符号
  - 局部变量
- 符号表

编译器和汇编器做了什么？

- 机器代码
- 数据
  - 只读数据：printf 格式串，跳转表
  - 已初始化的全局或静态变量
  - 未初始化的静态变量，或初始化为 0 的全局或静态变量
  - 未初始化的全局变量
  - 不该被重定位的符号
  - 未定义的符号
  - 局部变量
- 符号表
- 重定位方式

## 编译器和汇编器做了什么？

- 机器代码
- 数据
  - 只读数据：printf 格式串，跳转表
  - 已初始化的全局或静态变量
  - 未初始化的静态变量，或初始化为 0 的全局或静态变量
  - 未初始化的全局变量
  - 不该被重定位的符号
  - 未定义的符号
  - 局部变量
- 符号表
- 重定位方式
- 调试信息

## 编译器和汇编器做了什么？

- 机器代码 (.text)
- 数据
  - 只读数据：printf 格式串，跳转表 (.rodata)
  - 已初始化的全局或静态变量 (.data)
  - 未初始化的静态变量，或初始化为 0 的全局或静态变量 (.bss)
  - 未初始化的全局变量 (COMMON)
  - 不该被重定位的符号 (ABS)
  - 未定义的符号 (UNDEF)
  - 局部变量
- 符号表 (.symtab)
- 重定位方式 (.rel.text, .rel.data)
- 调试信息 (.debug, .line, .strtab)

# 符号表

```
Num:    Value          Size Type    Bind   Vis      Ndx Name
  8: 0000000000000000    24 FUNC    GLOBAL DEFAULT    1 main
  9: 0000000000000000     8 OBJECT  GLOBAL DEFAULT    3 array
 10: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND sum
```

指明每个符号的：

- name: 符号名字
- type: 函数或数据
- binding: 全局或局部
- reserved
- section: 在哪个节
- value: 偏移量或绝对地址
- size: 至少有多大

# 符号表

```
Num:    Value          Size Type    Bind   Vis      Ndx Name
  8: 0000000000000000    24 FUNC    GLOBAL DEFAULT    1 main
  9: 0000000000000000     8 OBJECT  GLOBAL DEFAULT    3 array
 10: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND sum
```

指明每个符号的：

- name: 符号名字
- type: 函数或数据
- binding: 全局或局部
- reserved
- section: 在哪个节
- value: 偏移量或绝对地址
- size: 至少有多大

# 符号解析

- 强符号：函数和已初始化的全局变量
- 弱符号：未初始化的全局变量

# 符号解析

- 强符号：函数和已初始化的全局变量
- 弱符号：未初始化的全局变量

- 不能有同名的强符号

# 符号解析

- 强符号：函数和已初始化的全局变量
- 弱符号：未初始化的全局变量

- 不能有同名的强符号
- 如果只有一个强符号，选择这个强符号
- 没有强符号，随机选择一个弱符号（危险！）

# 符号解析

- 强符号：函数和已初始化的全局变量
- 弱符号：未初始化的全局变量

- 不能有同名的强符号
- 如果只有一个强符号，选择这个强符号
- 没有强符号，随机选择一个弱符号（危险！）

- 谨慎使用全局变量
- 使用 static variables 代替全局变量
- 使用'extern' 关键字

# 重定位

两步：

- 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

两步：

- 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

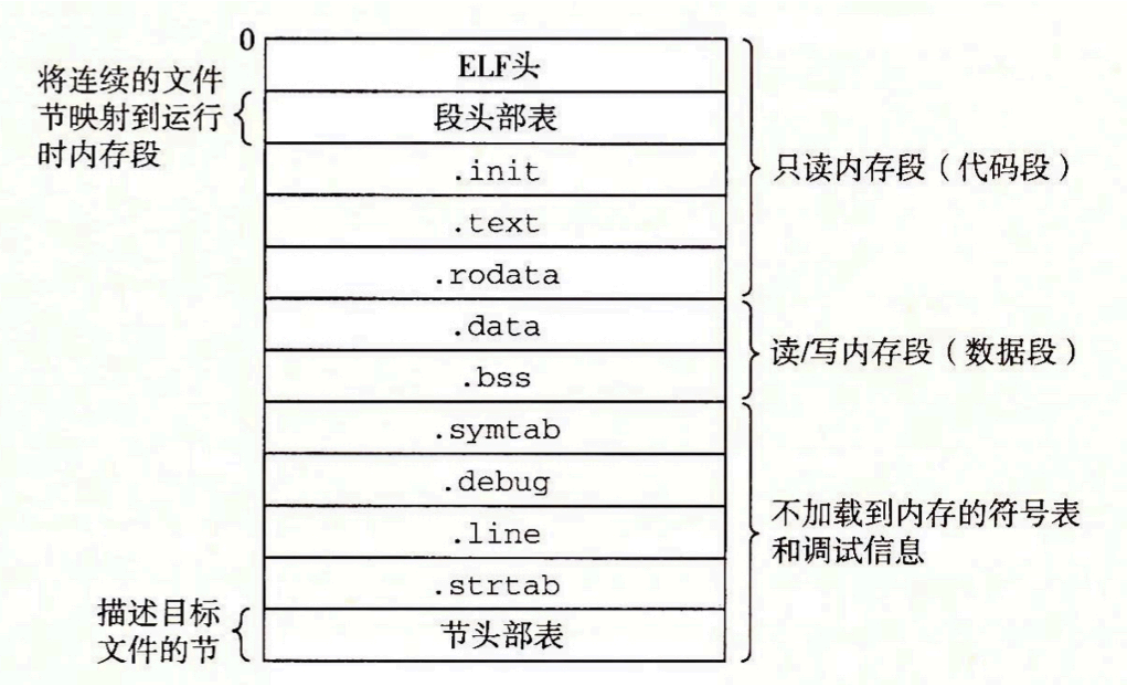
code/link/elfstructs.c

```
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;
```

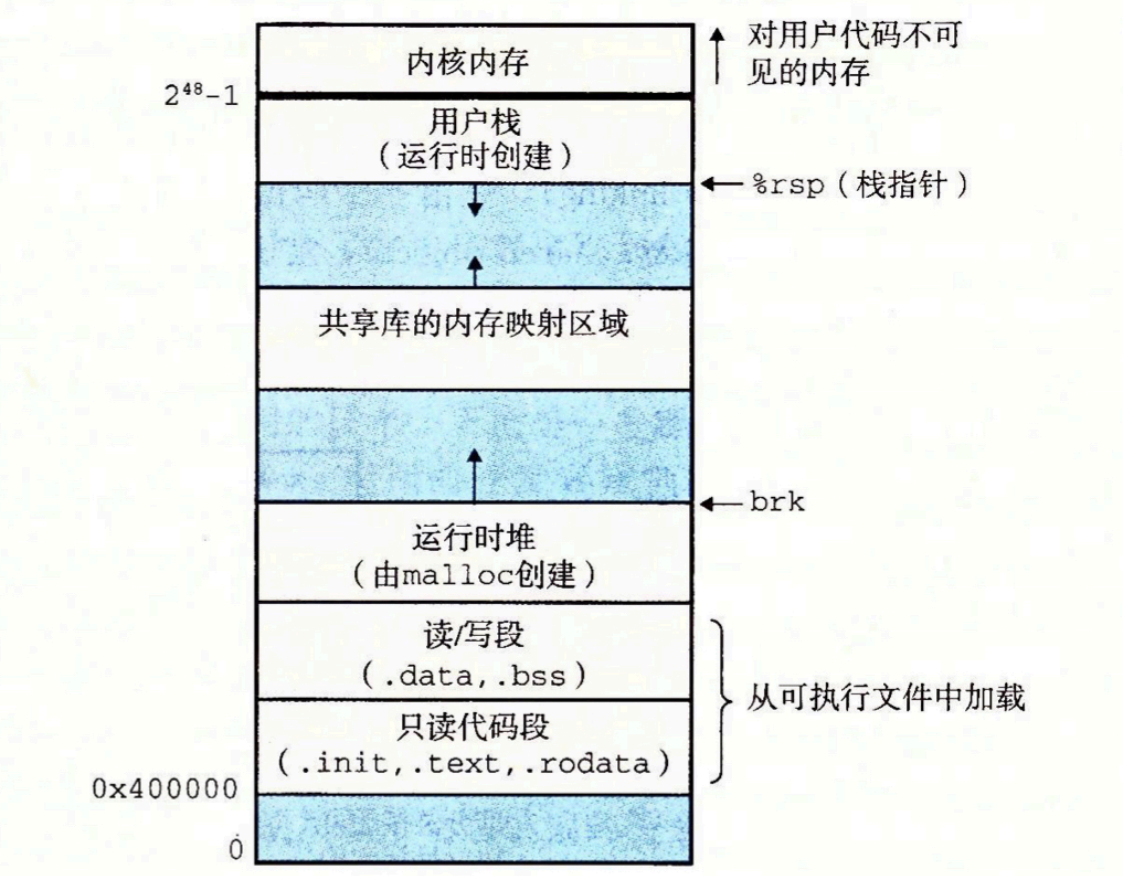
code/link/elfstructs.c

- R_X86_64_PC32: 重定位一个使用 32 位 PC 相对地址的引用
  - 计算出引用的运行时地址：refaddr = ADDR(s) + r.offset
  - 更新引用：*refptr = (unsigned)(ADDR(r.symbol) + r.addend - refaddr
- R_X86_64_32: 重定位一个使用 32 位绝对地址的引用
  - *refptr = ADDR(r.symbol) + r.addend

# 可执行目标文件


0
ELF头
段头部表
.init
.text
.rodata
.data
.bss
.symtab
.debug
.line
.strtab
节头部表
将连续的文件节映射到运行时内存段
描述目标文件的节
只读内存段（代码段）
读/写内存段（数据段）
不加载到内存的符号表和调试信息

# 加载

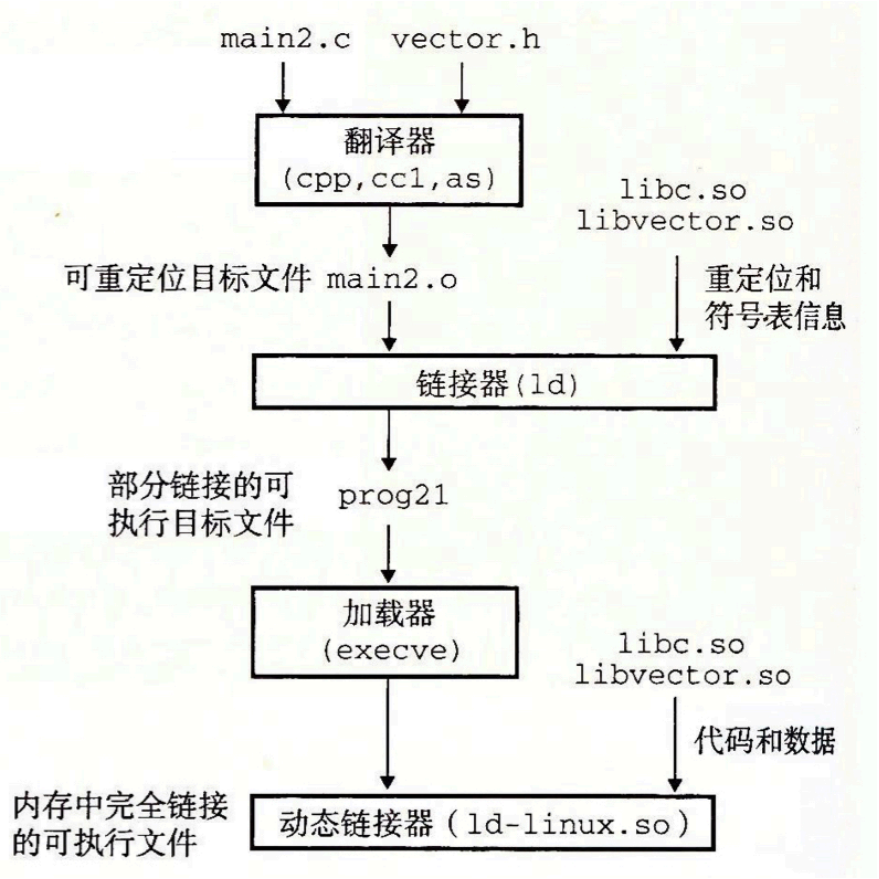

内核内存
对用户代码不可见的内存
$2^{48}-1$
用户栈
（运行时创建）
%rsp（栈指针）
共享库的内存映射区域
brk
运行时堆
（由malloc创建）
读/写段
（.data,.bss）
只读代码段
（.init,.text,.rodata）
从可执行文件中加载
0x400000
0

# 静态库

将多个相关的函数封装成一个单独的库（存档文件'.a'）
静态库的符号解析：

- 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 $m_i$ 定义了 U 中的符号，就将 $m_i$ 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。

# 共享库'.so'


main2.c
vector.h
翻译器
(cpp,cc1,as)
libc.so
libvector.so
可重定位目标文件 main2.o
重定位和
符号表信息
链接器(ld)
部分链接的可
执行目标文件
prog21
加载器
(execve)
libc.so
libvector.so
代码和数据
内存中完全链接
的可执行文件
动态链接器（ld-linux.so）

# PIC 数据引用

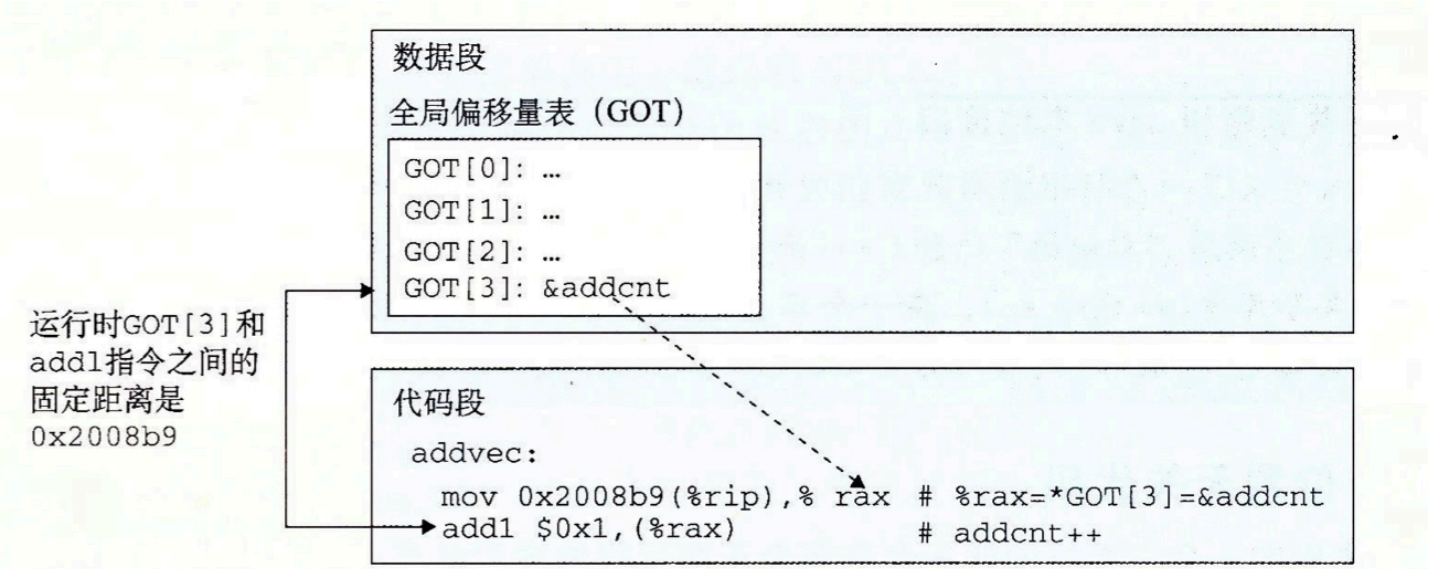
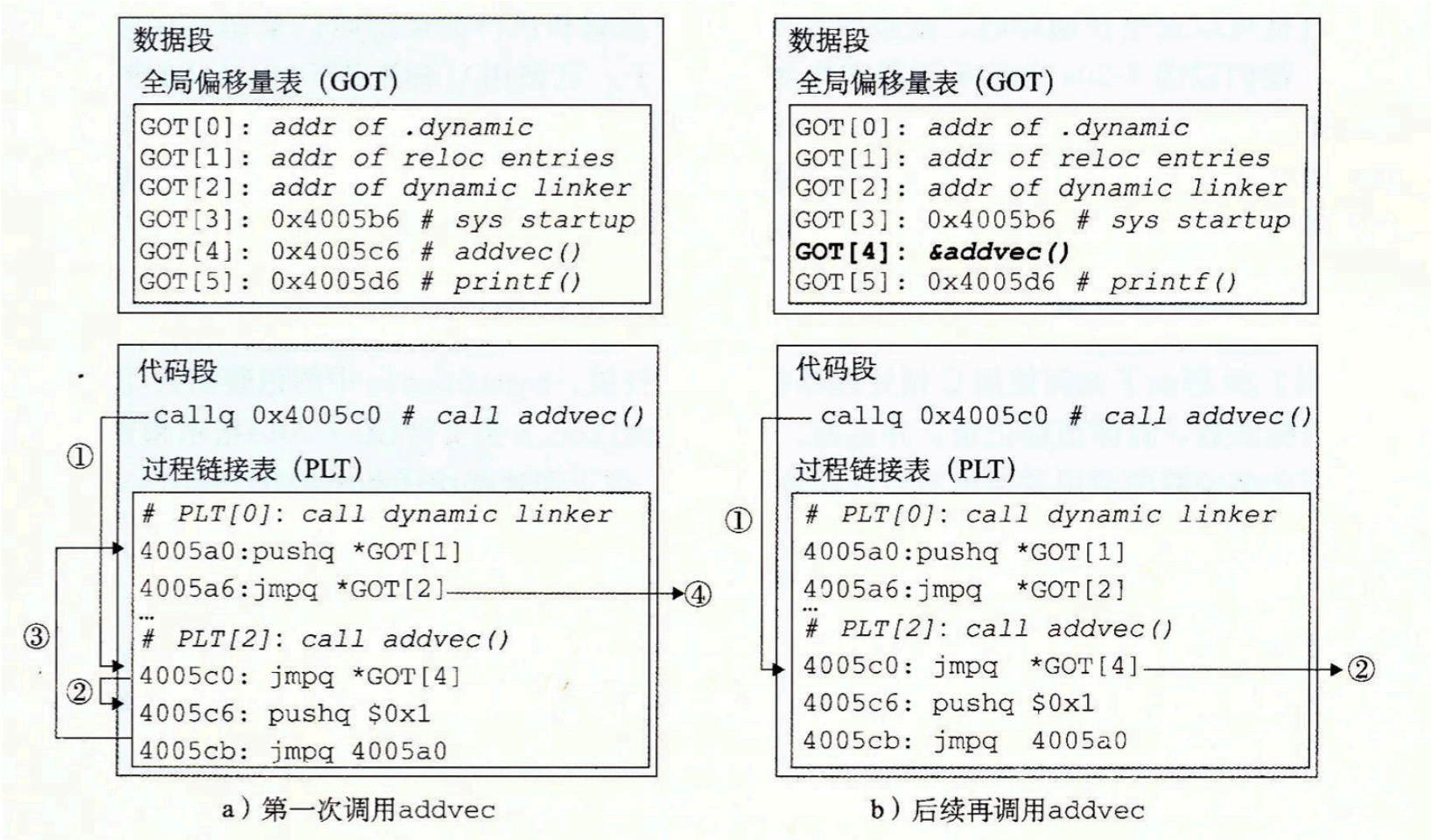



图 7-18 用 GOT 引用全局变量。libvector.so 中的 addvec 例程通过 libvector.so 的 GOT 间接引用了 addcnt

# PIC 函数调用


数据段
全局偏移量表（GOT）
GOT[0]: addr of .dynamic
GOT[1]: addr of reloc entries
GOT[2]: addr of dynamic linker
GOT[3]: 0x4005b6 # sys startup
GOT[4]: 0x4005c6 # addvec()
GOT[5]: 0x4005d6 # printf()
代码段
callq 0x4005c0 # call addvec()
过程链接表（PLT）
# PLT[0]: call dynamic linker
4005a0:pushq *GOT[1]
4005a6:jmpq *GOT[2]
...
# PLT[2]: call addvec()
4005c0: jmpq *GOT[4]
4005c6: pushq $0x1
4005cb: jmpq 4005a0
①
②
③
④
a）第一次调用addvec
数据段
全局偏移量表（GOT）
GOT[0]: addr of .dynamic
GOT[1]: addr of reloc entries
GOT[2]: addr of dynamic linker
GOT[3]: 0x4005b6 # sys startup
GOT[4]: &addvec()
GOT[5]: 0x4005d6 # printf()
代码段
callq 0x4005c0 # call addvec()
过程链接表（PLT）
# PLT[0]: call dynamic linker
4005a0:pushq *GOT[1]
4005a6:jmpq *GOT[2]
...
# PLT[2]: call addvec()
4005c0: jmpq *GOT[4]
4005c6: pushq $0x1
4005cb: jmpq 4005a0
①
②
b）后续再调用addvec