Beijing Forest Studio
北京理工大学信息系统及安全对抗实验中心

程序的链接

硕士研究生　侯钰斌

2021年03月14日

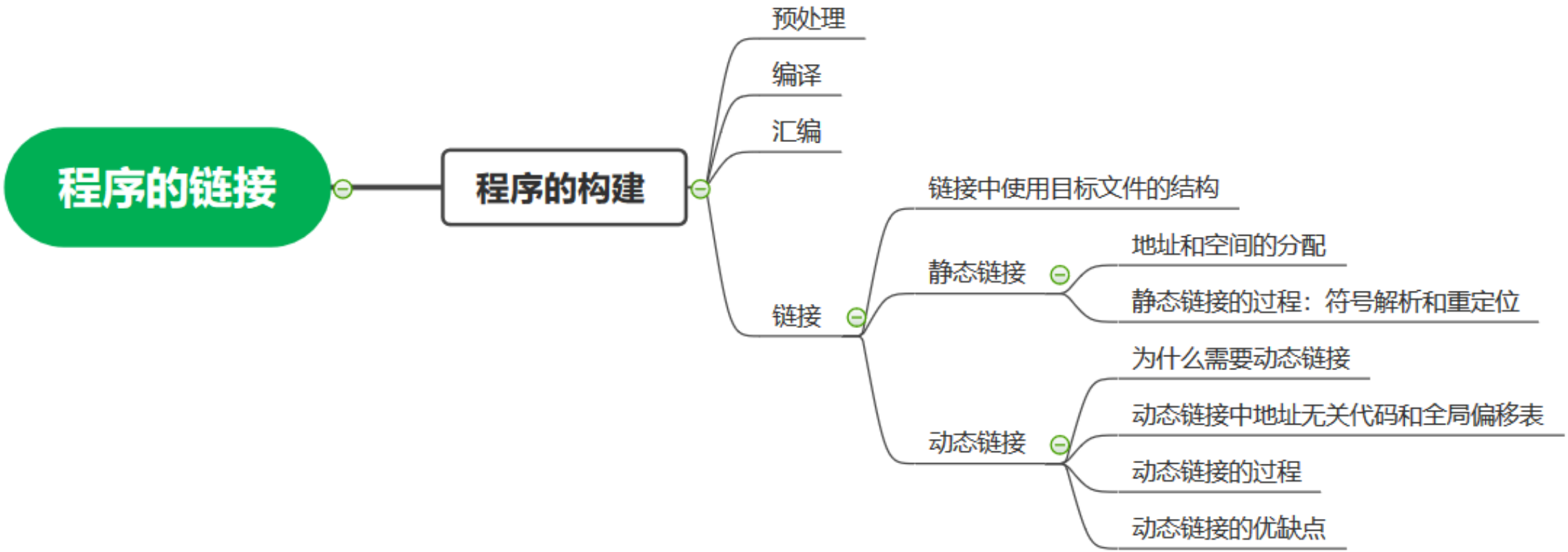
内容提要

- 概念引入
- 程序编译的过程
- 链接的历史
- 链接的基本概念
- 目标文件的结构
- 静态链接
- 动态链接
- 参考文献

预期收获

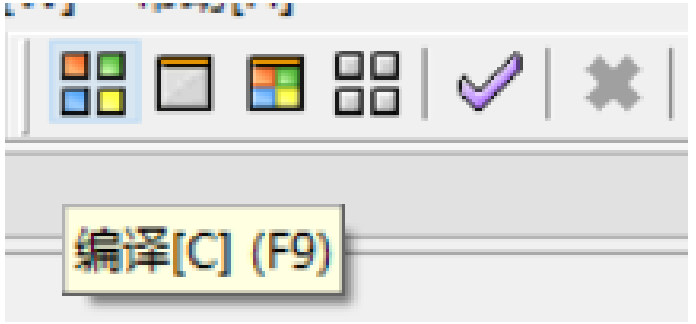
- 预期收获
  - 1. 了解程序编译和链接的基本概念和过程
  - 2.了解可执行程序中特殊意义的段
  - 3.了解静态链接和动态链接的过程

思维导图


程序的链接
程序的构建
预处理
编译
汇编
链接
链接中使用目标文件的结构
静态链接
地址和空间的分配
静态链接的过程：符号解析和重定位
动态链接
为什么需要动态链接
动态链接中地址无关代码和全局偏移表
动态链接的过程
动态链接的优缺点

概念引入

- 一个简单的操作
  - 使用集成开发环境编写C/C++程序，单击某个快捷键就可以实现从源码转化为可执行文件的过程。这个过程名为构建（Build）。
  - 使用简单的构建命令（gcc xxx.c）也可以实现从源码转化为可执行文件的过程。
  - 集成开发环境和编译器默认的构建过程参数足够满足大多数人对于程序开发的要求。
  - 集成环境提供的构建功能对于大多数开发者是透明的。



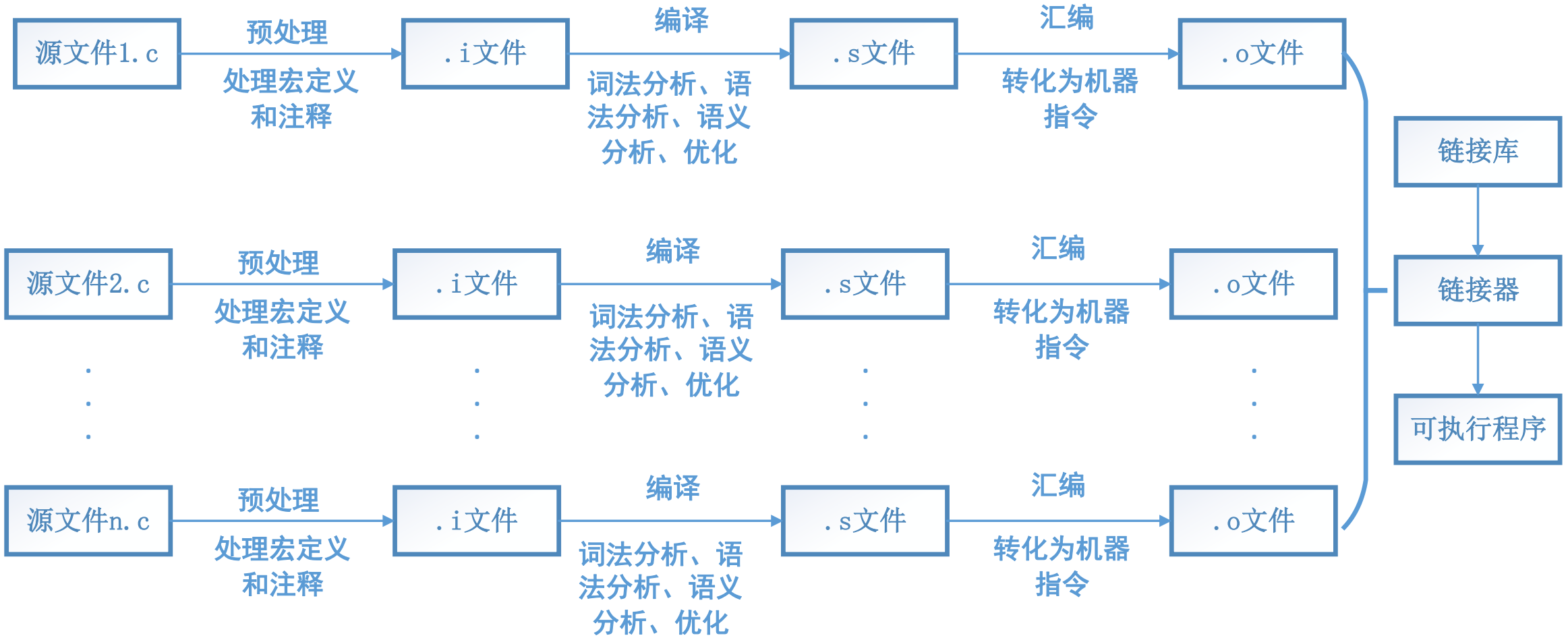
概念引入

- 各类程序语言学习入门“Hello World”
  - 编译器在构建过程中做了什么?
  - 编译得到的执行文件的结构是什么样的?
  - 执行文件中各类函数和变量是怎么调用的?

```c
#include<stdio.h>
int  main()
{
      printf("hello world\n");
      return 0;
}
```

程序构建的过程

- 构建过程分为四个步骤：预处理、编译、汇编和链接。

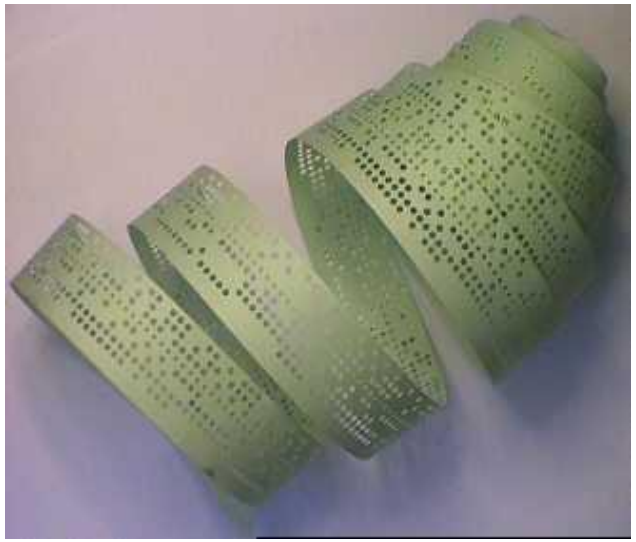


程序构建的过程

- 预处理：处理预编译指令和注释。
  - 将所有#define删除，并展开所有宏定义。
  - 处理所有条件预编译指令，如#if、#ifdef、#endif等。
  - 将导入的文件插入到预编译指令#include的位置。
  - 删除所有注释。
- 编译：将预处理后的文件进行词法语法分析、语义分析并优化生成汇编代码。
  - 将高级语言翻译为机器语言，是构建程序的关键步骤。
- 汇编：将汇编代码转化为机器指令，生成目标文件.o。
  - 根据汇编指令和机器指令的对照表一一对应进行翻译。
- 链接：连接有关的目标文件，建立不同文件的调用关系，组成一个操作系统可执行的整体。

链接的历史

- 程序的链接早在高级语言发明前就出现了。
  - 早期计算机编程指令的输入是打孔纸带，程序以0和1机器码的形式储存在纸带上。
  - 程序时常需要修改，插入指令并调整指令顺序和跳转地址非常麻烦。
  - 当程序规模变得越来越大，这种修改方法的效率越来越低。
- 汇编语言的出现改变了这种繁琐复杂的修改方式。
  - 使用符号代替二进制指令和模块起始地址。
  - 当修改指令后，汇编器会自动计算修改后函数起始地址和变量起始地址。
- 随着编程语言的发展和代码规模的扩大，程序被分为不同的模块，以便维护和重用，模块的调用和通信催生了如今的程序链接器的诞生。



链接的基本概念

- 在一个程序中，每个源代码模块独立编译，并按照实际需求组装，这个过程被称为“链接”。
- 链接的主要工作就是处理各个模块间的引用关系，使得各个模块可以正确的衔接。

T	实现程序设计的模块化，提升程序的可维护性和复用性
I	程序的目标文件
P	1. 地址和空间的分配 2. 符号决议 3. 重定位
O	操作系统可执行的文件

链接的基本概念

- 主程序模块a.c调用了另一个模块b.c中的函数swap()和变量shared。
  - 两个模块独自编译，主程序不知道调用函数swap()的地址，可以默认为0，调用代码举例为call 0x00000000。
  - 链接器根据引用的符号swap自动查找swap()的地址，假设地址为0x45000000。
  - 修改主程序中所有调用该函数的地址，如call 0x45000000。

```
/* a.c */
extern int shared;

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

```
/* b.c */
int shared = 1;

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

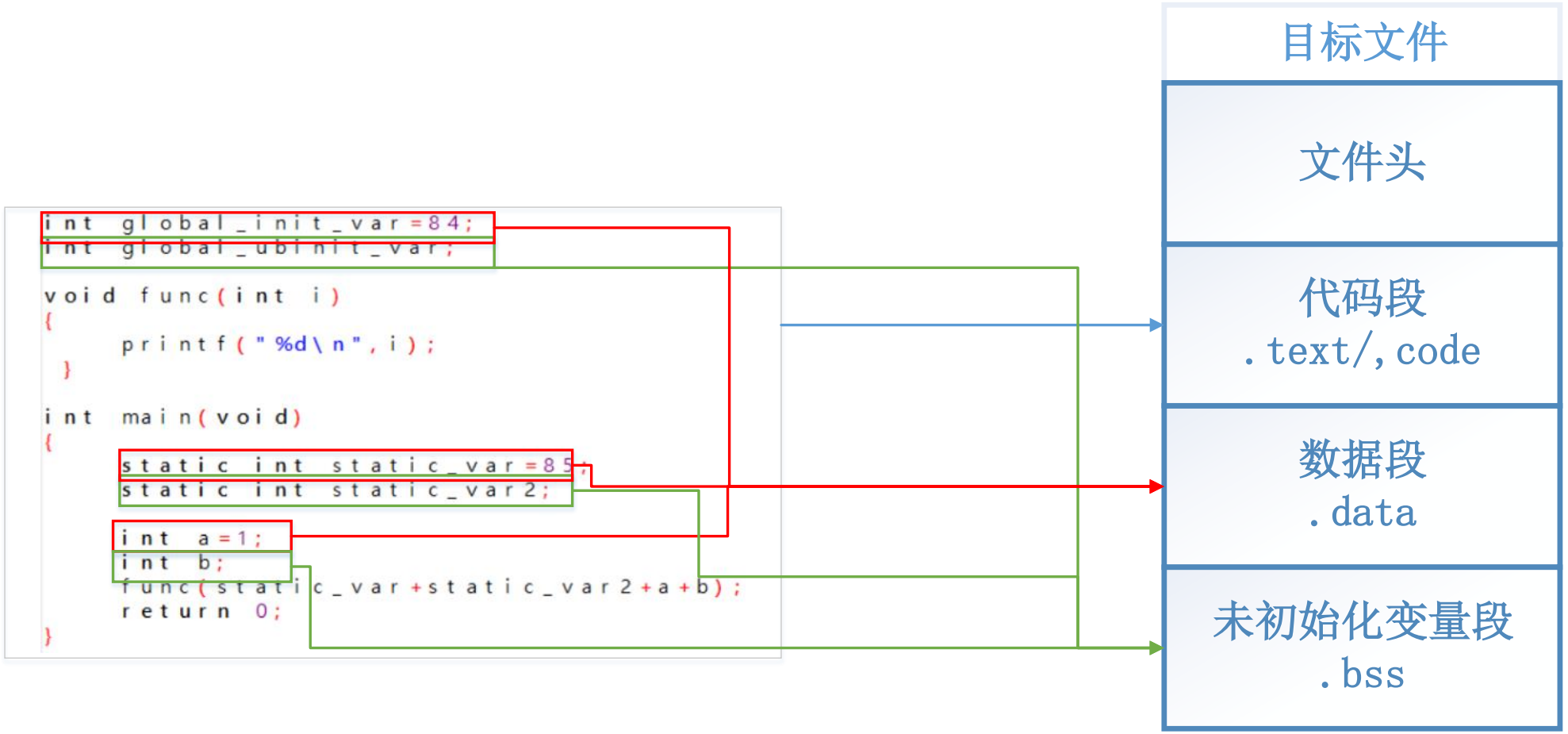
目标文件的结构

- 编译器编译源代码后，尚未进行链接的中间文件是目标文件。
  - 目标文件和可执行文件的结构和格式相似，但是某些符号和地址尚未被确定。
- 相同类型的文件：

文件类型	实例	文件说明
可重定位文件	Linux: .o/.a Windows: .obj/.lib	包含了代码和数据，可被链接为可执行文件和共享目标文件。静态链接库是一个包含多个该文件和文件索引的集合。
可执行文件	Windows: .exe	可以直接执行的程序。
共享目标文件	Linux: .so Windows: .dll	1. 使用该文件与其他可重定位文件或共享目标文件链接，生成新的目标文件。 2. 动态链接器将多个该文件与可执行文件结合并运行。
核心转储文件	Linux: .core dump	当进程意外终止时，系统将进程运行信息储存到该文件中。

目标文件的结构

- 目标文件包含编译后的机器指令代码、数据、符号表等结构，被称为“段”。



目标文件的结构

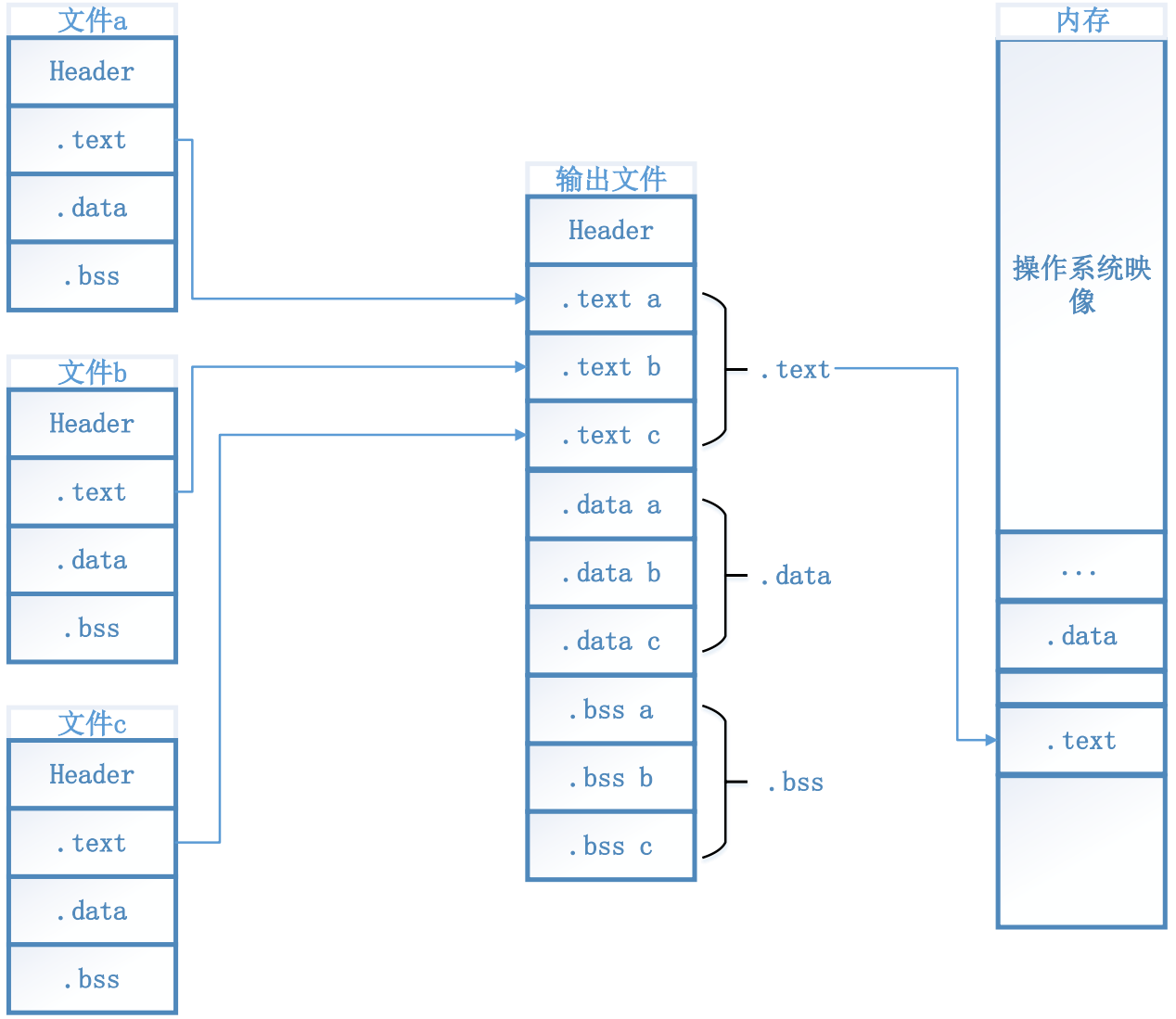
- .init 程序初始化代码段
- .finit 程序终结代码段
- .rodata 只读数据段，存放程序中的常量
- .rel.XXX 重定位表，储存重定位后模块中函数和变量的调用关系
- .symtab 符号表，存放函数和变量的符号
- .shstrtab 段名表，标识文件中段的名称和属性
- .dynamic 动态链接信息段
- .got 全局入口表，用于动态链接

静态链接

- 第一步：空间和地址的分配
  - 扫描所有输入的目标文件，并获得各段长度、属性和位置。计算输出文件中合并段的长度和位置，并建立与内存的映射关系。
  - 收集所有输入文件的符号定义和引用关系，统一存放到输出文件的全局符号表。
  - 全局符号表是指所有被调用的目标文件的符号表组成的结构。
- 第二步：符号解析与重定位
  - 利用第一步的信息，读取输入文件中段的数据和重定位信息，进行符号解析和重定位，调整代码中引用和转跳地址。
  - 这是链接过程的核心。

静态链接

- 空间和地址的分配
  - 相同段的内容合并在一起，有利于处理器连续执行命令或读取数据，提升了程序运行效率。
  - 相同段的合并能有效减小零散段的数量，从而减少内存碎片，减小程序装载的大小。
  - 链接器会计算输出文件中合并段的长度和位置，并建立与内存的映射关系。



静态链接

- 空间和地址的分配
  - 代码段.text在内存上的地址为0x08048094。
  - 数据段.code在内存上的地址为0x08049108。

```
/* a.c */
extern int shared;

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

```
/* b.c */
int shared = 1;

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

文件	代码段大小	数据段大小
a.o	0x34	0x0
b.o	0x3e	0x4
ab（a与b链接）	0x72	0x4

静态链接

- 符号解析
  - 直接查看a.o文件可以发现文件从外部调用的函数和变量均是未定义。
  - 如果未定义的符号在主模块的全局符号表中没有找到，则链接器会报错，程序也无法运行。

```
Symbol table '.symtab' contains 10 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    5
     7: 00000000    52 FUNC    GLOBAL DEFAULT    1 main
     8: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND shared
     9: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND swap
```

静态链接

- 符号解析
  - 链接器在空间和地址分配完成后，就会确定所有符号的目标地址。
  - 链接器检索所有输入的目标文件的符号表，并将其存放在主模块的全局符号表。
  - 链接器更新全局符号表的目标地址。

符号	类型	目标地址
main	函数	0x08048094
swap	函数	0x080480c8
shared	变量	0x08049108

- main函数是代码段最开始的模块，在代码段中偏移为0，其目标地址就是代码段的起始地址0x08048094。
- swap函数是main函数的调用模块，紧跟在main函数之后，在代码段中偏移为0x34，其目标地址是代码段起始地址0x08048094+0x34=0x080480c8。

静态链接

- 重定位
  - 链接器根据重定位表和全局符号表更新调用和赋值的地址。
  - 重定位表中偏移是代码中调用和赋值需要重定位的位置，即入口地址。

```
00000000 <main>:
   0:   8d 4c 24 04             lea    0x4(%esp),%ecx
   4:   83 e4 f0                and    $0xfffffff0,%esp
   7:   ff 71 fc                pushl  0xfffffffc(%ecx)
   a:   55                      push   %ebp
   b:   89 e5                   mov    %esp,%ebp
   d:   51                      push   %ecx
   e:   83 ec 24                sub    $0x24,%esp
  11:   c7 45 f8 64 00 00 00    movl   $0x64,0xfffffff8(%ebp)
  18:   c7 44 24 04 00 00 00    movl   $0x0,0x4(%esp)
  1f:   00
  20:   8d 45 f8                lea    0xfffffff8(%ebp),%eax
  23:   89 04 24                mov    %eax,(%esp)
  26:   e8 fc ff ff ff          call   27 <main+0x27>
  2b:   83 c4 24                add    $0x24,%esp
  2e:   59                      pop    %ecx
  2f:   5d                      pop    %ebp
  30:   8d 61 fc                lea    0xfffffffc(%ecx),%esp
  33:   c3                      ret
```

a.o的代码

```
RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE              VALUE
0000001c R_386_32          shared
00000027 R_386_PC32        swap
```

a.o的代码重定位表

静态链接

- 重定位
  - 修改程序的调用地址和赋值地址。

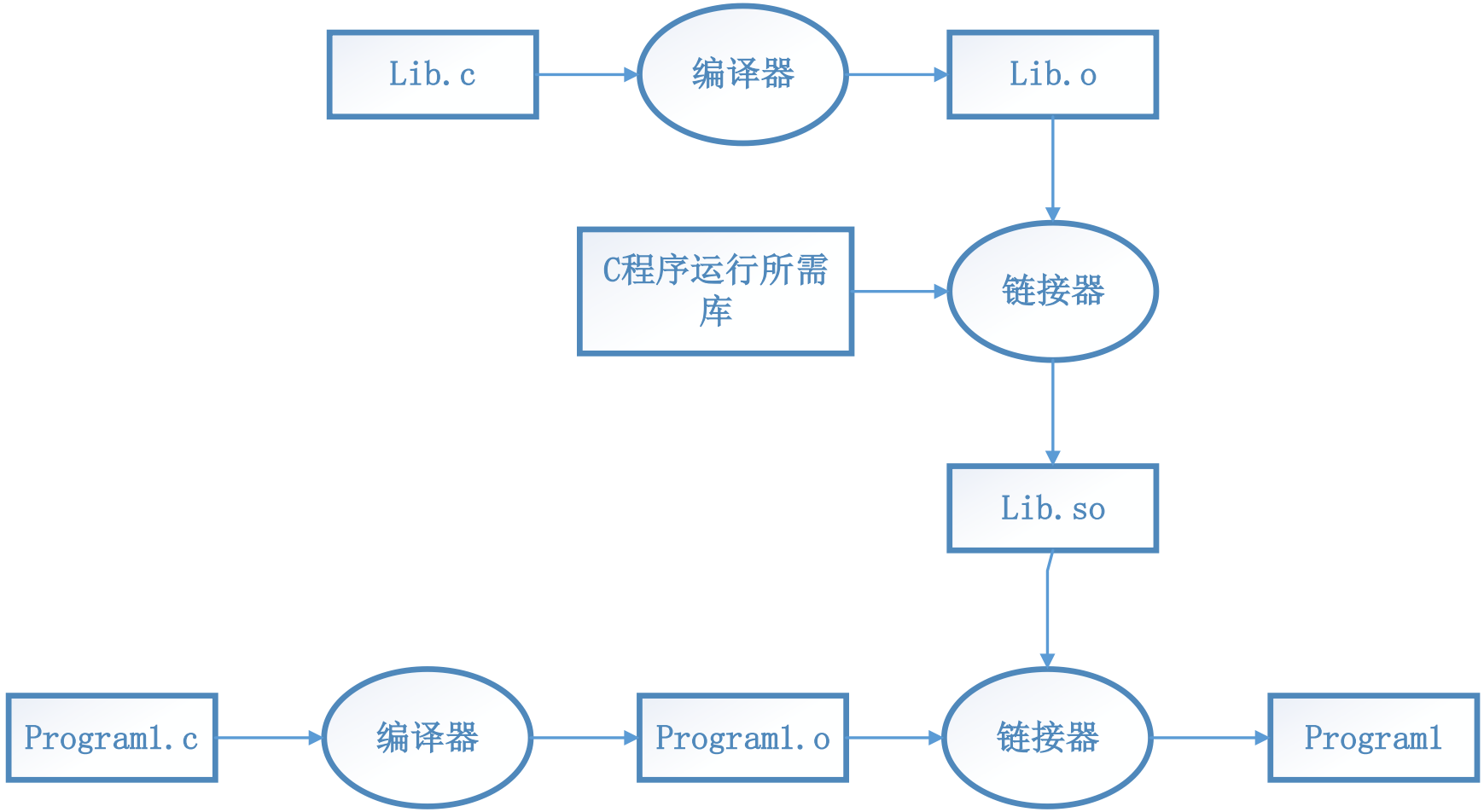
```
08048094 <main>:
 8048094:    8d 4c 24 04                 lea     0x4(%esp),%ecx
 8048098:    83 e4 f0                    and     $0xfffffff0,%esp
 804809b:    ff 71 fc                    pushl   0xfffffffc(%ecx)
 804809e:    55                          push    %ebp
 804809f:    89 e5                       mov     %esp,%ebp
 80480a1:    51                          push    %ecx
 80480a2:    83 ec 24                    sub     $0x24,%esp
 80480a5:    c7 45 f8 64 00 00 00        movl    $0x64,0xfffffff8(%ebp)
 80480ac:    c7 44 24 04 08 91 04        movl    $0x8049108,0x4(%esp)
 80480b3:    08
 80480b4:    8d 45 f8                    lea     0xfffffff8(%ebp),%eax
 80480b7:    89 04 24                    mov     %eax,(%esp)
 80480ba:    e8 09 00 00 00              call    80480c8 <swap>
 80480bf:    83 c4 24                    add     $0x24,%esp
 80480c2:    59                          pop     %ecx
 80480c3:    5d                          pop     %ebp
 80480c4:    8d 61 fc                    lea     0xfffffffc(%ecx),%esp
 80480c7:    c3                          ret

080480c8 <swap>:
 80480c8:    55                          push    %ebp
```

动态链接

- 为什么需要动态链接
  - 静态链接需要一次性链接所有模块，如果某个模块使用频率较低，那么这个模块会占用内存资源。
  - 如果链接的模块数量多或体积大，那么链接得到的文件体积会更大，不方便载入内存并运行。
  - 如果更新模块，那么需要将所有模块重新编译和链接，这个过程十分麻烦。
- 动态链接的基本思想
  - 当程序运行时对各个模块进行链接形成一个完整的程序。
  - 共享目标文件（.so/.dll）

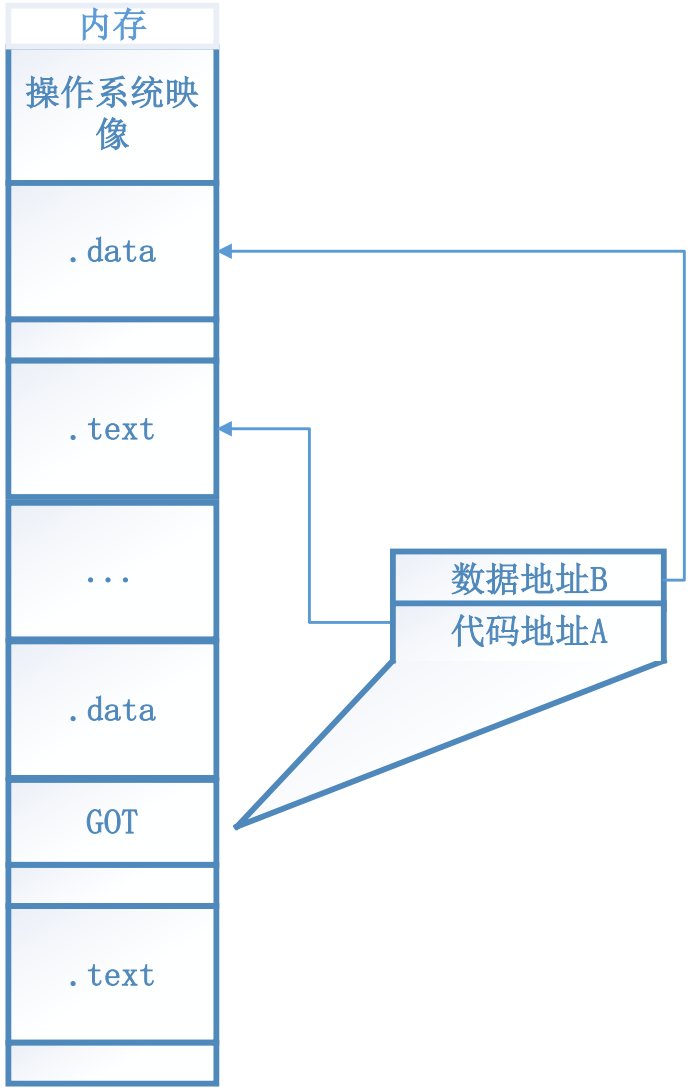
动态链接

- 动态链接过程（program1是主模块，Lib是被调用的模块）



动态链接

- 共享代码装载的问题
  - 使用动态链接，模块载入内存的地址是动态变化的，不能使用绝对地址。
  - 希望程序模块中共享的指令代码在载入内存时不会因载入地址改变而改变。
- 地址无关代码（PIC）
  - 数据部分在每个进程中都存在一个副本，便于进程运行时访问和修改数据。
  - 分离共享部分的代码，将其与数据部分合并。
  - 在数据段中建立指向共享代码部分的指针数组，即全局偏移表（GOT）



动态链接

- 全局偏移表（GOT）工作原理
  - 当指令需要访问共享文件中的数据或者代码，动态链接器会先访问主模块中的GOT段。
  - 根据中GOT的符号，找到数据或代码在主模块中的入口地址。此时数据和模块的目标地址还未被确定。
  - 动态链接器在装载共享文件时会计算载入内存的地址。
  - 更新GOT中对应符号的目标地址，之后主模块可以根据GOT间接索引数据或代码。

动态链接

- 共享目标文件中的特殊段
  - .interp
    - 保存动态链接器的路径。
  - .dynamic
    - 保存动态链接的基本信息，包括动态链接符号表位置、动态链接重定位表的位置、共享文件初始化代码的地址、依赖的共享文件名等。可以认为是动态链接中的文件头。
  - .dynsym
    - 保存在动态链接中模块间符号的调用关系。
  - .rel.dyn .rel.plt .got 等是用于动态链接重定位的特殊段

动态链接

- 启动动态链接器
  - 动态链接器本身不依赖其他共享文件，自身载入内存的过程必须由自己完成。
  - 当操作系统将控制权交给动态链接器时，动态链接器开始“自举”。
    - 自举代码找到自己的全局偏移表（ GOT ），在GOT中保存了自身的.dynamic段的偏移地址，由此找到了动态链接器.dynamic段的信息。
    - 通过.dynamic段，可以获取动态链接器本身的重定位表和符号表等，从而对链接器自身进行重定位，之后链接器才能使用自身的各种变量和函数。
    - 使用地址无关代码模式（PIC）编译的模块，模块内部函数调用也是通过全局偏移表间接寻址，所以在自举完成之后链接器才能使用自身的各种变量和函数。

动态链接

- 装载所需共享目标文件
  - 链接器根据主模块中.dynamic段，寻找主模块所依赖的全部共享目标文件的名称。
  - 根据名称读取对应的共享目标文件，读取文件头和.dynamic段，将相应的文件内容载入内存。如果该目标文件还有依赖其他的共享目标文件，则将依赖的文件载入内存，直到所有的依赖的共享目标文件被载入内存。
  - 当一个新的共享目标文件被载入，它的符号表会合并到全局符号表中。
  - 需要注意的是，当一个符号被并入全局符号表时，如果同名符号已经出现，则后并入的符号被忽略。这有可能导致程序错误。

动态链接

- 重定位和初始化
  - 动态链接器遍历主模块和每个共享目标文件的重定位表。
  - 动态链接器已经拥有所有模块的符号表，修正所有文件和模块的全局偏移表（ GOT ）。
  - 重定位完成后，如果某个共享目标文件中存在.init 段，则动态链接器会执行该段中的代码，从而初始化共享目标文件，类似C++中类的初始化。
  - 如果共享目标文件中存在.finit 段，当进程结束时会执行该代码，类似C++中类的析构。
  - 完成重定位和初始化，动态链接器将控制权转交给程序入口，主模块开始执行。

动态链接

- 优点
  - 减少了程序运行时在内存中相同模块的副本数量，节省了内存空间，提升程序运行效率。
  - 只要设计好更新的模块，在下次运行程序时，新的模块就会被链接到程序中，发布和更新模块的过程会更加快速和简便。
  - 各个模块之间更加独立，方便模块的开发和测试，提升了程序的扩展性和兼容性。
- 缺点
  - 必须考虑程序接口兼容问题，否则会导致程序无法运行。
  - 如果动态链接的文件过多，又缺少有效的共享文件管理机制，会出现“DLL Hell”的问题，导致程序整体的崩溃。

参考文献

[1] 俞甲子,石凡. 程序员的自我修养--链接、装载与库[M]. 北京: 电子工业出版社, 2009.

[2] CSDN. 程序编译和链接原理理解[EB/OL]. https://blog.csdn.net/caogenwangbaoqiang/article/details/79678095, 2018-03-24.

[3] CSDN. 程序链接链的是什么[EB/OL]. https://blog.csdn.net/jmw1407/article/details/108193061, 2020-08-26.

知人者智，自知者明。
胜人者有力，自胜者
强。知足者富。强行
者有志。不失其所者
久。死而不亡者，寿。

谢谢！