Linking

尹嘉奕 2020.11.26

Content

01 目标文件

02 符号解析

03 重定位

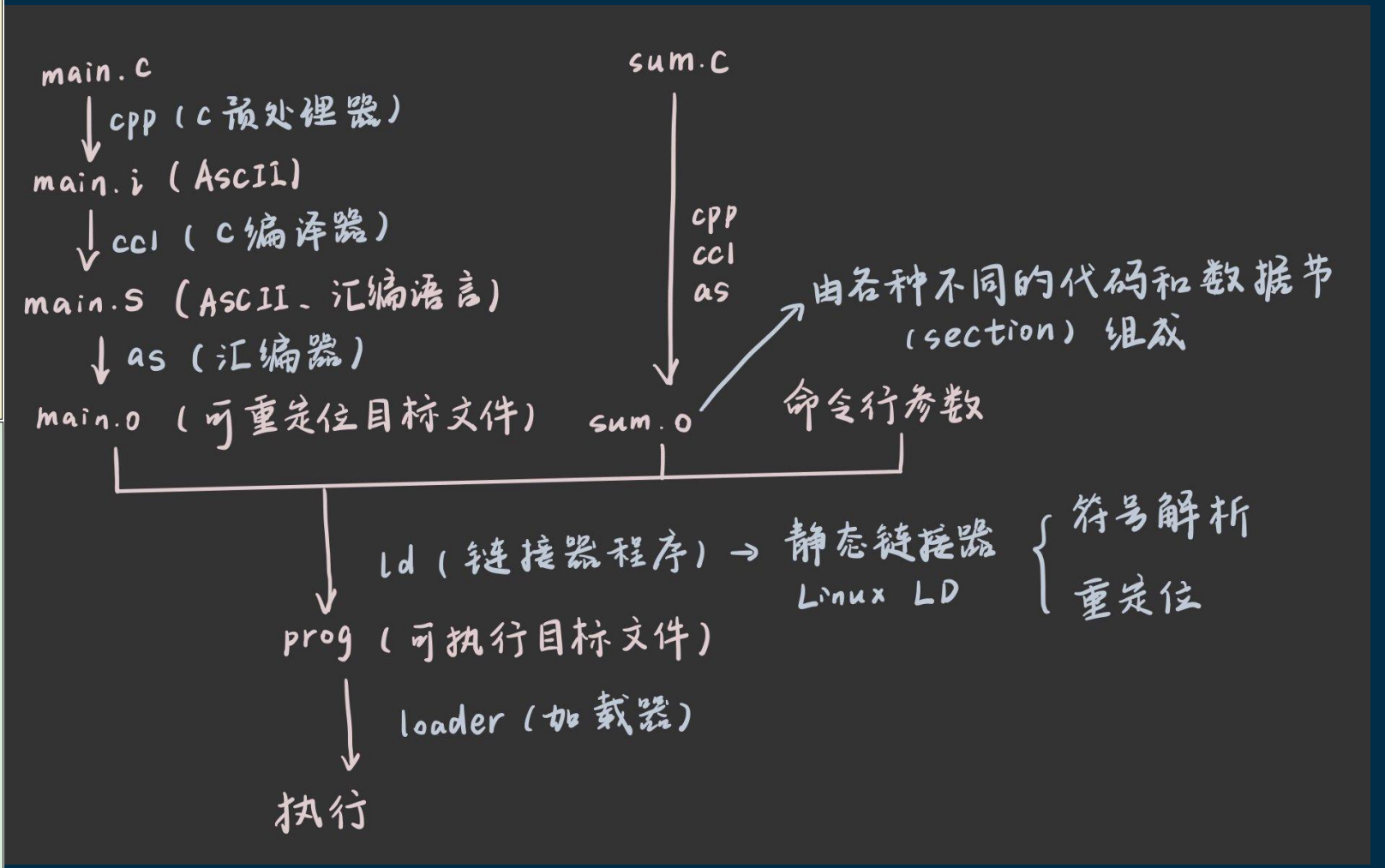
00 编译器驱动程序

```c
int sum(int *a, int n);

int array[2] = {1, 2};

int main(int argc, char** argv)
{
    int val = sum(array, 2);
    return val;
}
```

main.c

```c
int sum(int *a, int n)
{
    int i, s = 0;

    for (i = 0; i < n; i++) {
        s += a[i];
    }
    return s;
}
```

sum.c



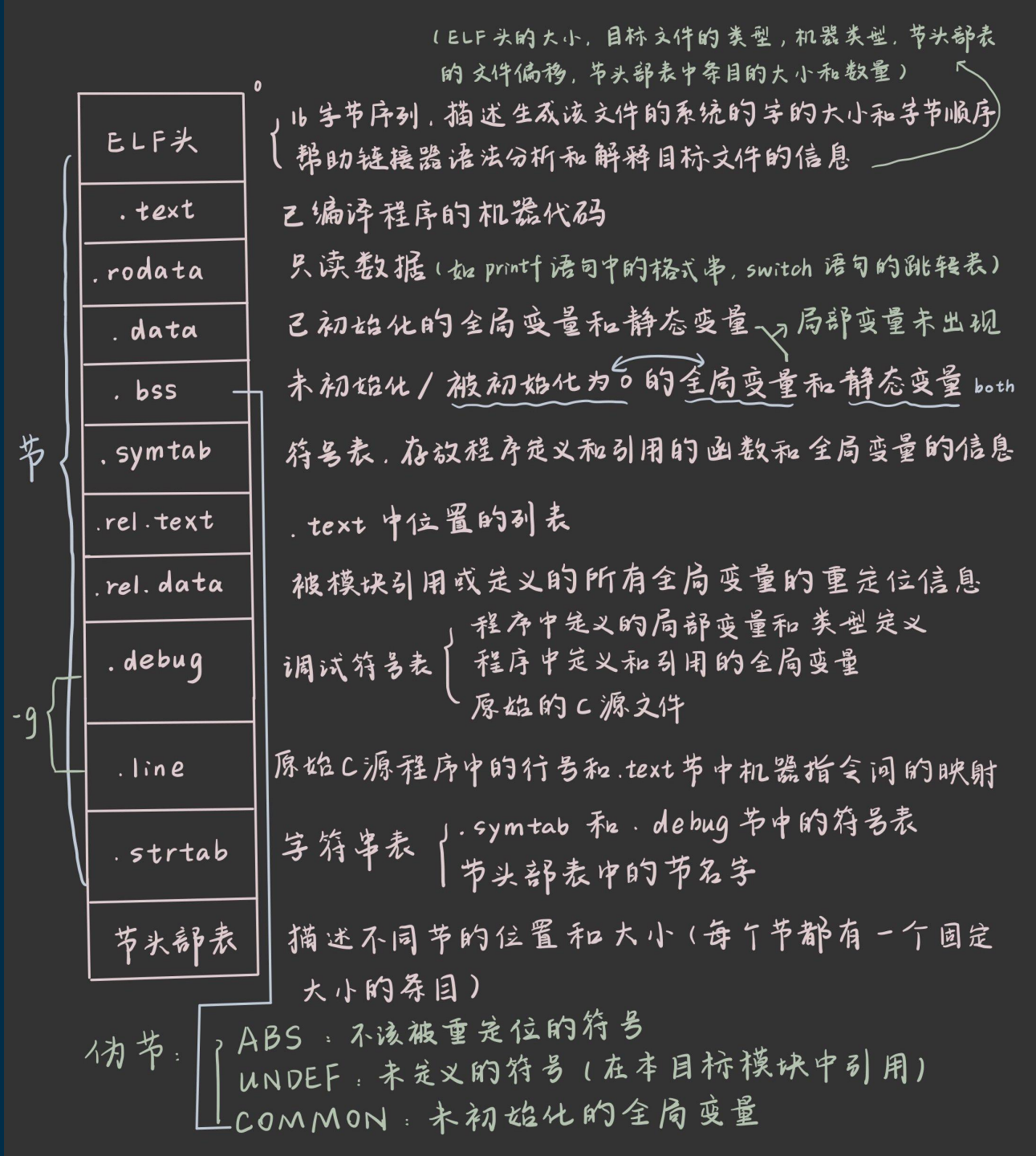
01 目标文件

可重定位目标文件

目标文件的三种形式：

- 可重定位目标文件 .o
- 可执行目标文件 a.out
- 共享目标文件 .so

→ **ELF：可执行可链接格式**



01 目标文件

可执行目标文件

目标文件的三种形式：

- 可重定位目标文件 .o
- 可执行目标文件 a.out
- 共享目标文件 .so

→ **ELF**：可执行可链接格式

01 目标文件

可重定位目标文件 to 可执行目标文件：链接器的两个主要任务

- 符号解析

  工作：把每个符号引用正好和一个符号定义关联起来

  目的：找到所有模块里的符号引用（ref），并为这些引用找到定义（def）

- 重定位

  工作：把每个符号定义与一个内存位置关联起来，并修改所有对这些符号的引用，使它们指向这个内存位置

  目的：确定模块中所有引用（ref）指向的地址

02 符号解析

符号和符号表

每个可重定位目标文件都有符号表 .symtab，它是一个条目的数组，每个条目包含符号的类型（变量or函数，局部or全局），符号的字符串名字的位置，符号的地址、目标的大小等。

- 全局符号：由该模块定义并能被其他模块引用的符号
  （对应非静态的C函数和全局变量）
- 外部符号：由其他模块定义并被该模块引用的符号
  （对应其他模块中定义的非静态C函数和全局变量）
- 局部符号：只被该模块定义和引用的符号
  （对应带static属性的C函数和全局变量）

Highlight：.symtab不包含对应非静态程序变量的任何变量！

02 符号解析

链接器如何解析多重定义的全局符号?

1. 强符号和弱符号

- 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（或加了extern声明的变量）

2. 三条处理规则

- 不允许有多个重名的强符号（否则链接器会报错）
- 如果有一个强符号和多个弱符号重名，那么选择强符号
- 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

- 符号解析的结果：

  代码中的每个符号引用正好和一个符号定义（一个输入模块中的符号表条目）关联起来

  此时，链接器知道了它的输入目标模块中的代码节和数据节的确切大小

03 重定位

重定位的两个步骤

- 重定位节和符号定义

  链接器将所有相同类型的节合并为同一类型的新的聚合节（合并输入模块）

  将运行时内存地址赋给新的聚合节、输入模块定义的每个节、输入模块定义的每个符号（为每个符号分配运行时地址）

  Result：程序中的每条指令和全局变量都有唯一的运行时内存地址

- 重定位节中的符号引用

  修改代码节和数据节中对每个符号的引用，使得它们指向正确的运行时地址（为每个引用找到它应该指向的运行时地址）

03 重定位

重定位条目

生成时间：　当汇编器遇到对最终位置未知的目标引用时

作用：　告诉链接器在将目标文件合并成可执行文件时如何修改这个引用

位置：　代码的重定位条目在 .rel.text 中

　　已初始化数据的重定位条目在 .rel.data 中

```
2   typedef struct{
3       long offset; /* 需要被修改的引用的节偏移（引用的位置离它所在的节的起始位置有多远）*/
4       long type:32, /* 如何修改新的引用：PC相对寻址 or 绝对寻址 */
5           symbol:32; /* 被修改引用应该指向的符号 */
6       long append; /* 对被修改引用的值做偏移调整 */
7   }
```

两种最基本的重定位类型：

- R_X86_64_PC32：PC相对寻址，在指令中编码的32位值 + PC当前值
- R_X86_64_32：绝对寻址，直接使用在指令中编码的32位值作为有效地址

03 重定位

重定位算法

```
2   foreach section s{
3       foreach relocation entry r{
4           refptr = s + r.offset; /* 一个指针，指向符号被引用的地址，地址里存放的是符号的值 */
5
6           /* PC相对寻址引用的重定位 */
7           if (r.type == R_X_86_64_PC32){
8               refaddr = ADDR(s) + r.offset; /* 引用的运行时地址，即 refptr 的值 */
9               /* 更新该引用，让它在运行时指向sum程序 */
10              *refptr = (unsigned) (ADDR(r.symbol) + r.addend - refaddr);
11          }
12
13          /* 绝对寻址引用的重定位 */
14          if (r.type == R_X86_64_32)
15              *refptr = (unsigned) (ADDR(r.symbol) + r.addend);
16      }
17  }
```

```
0000000000000000 <main>:
   0:   48 83 ec 08             sub    $0x8,%rsp
   4:   be 02 00 00 00          mov    $0x2,%esi
   9:   bf 00 00 00 00          mov    $0x0,%edi       # %edi = &array
                        a: R_X86_64_32 array           # Relocation entry

   e:   e8 00 00 00 00          callq  13 <main+0x13> # sum()
                        f: R_X86_64_PC32 sum-0x4       # Relocation entry
  13:   48 83 c4 08             add    $0x8,%rsp
  17:   c3                      retq
```

main.o

03 重定位

重定位的结果

- 修改后的指令放在最终可执行目标文件的 .text 节中

```
0000000000000000 <main>:
   0:   48 83 ec 08             sub    $0x8,%rsp
   4:   be 02 00 00 00          mov    $0x2,%esi
   9:   bf 00 00 00 00          mov    $0x0,%edi        # %edi = &array
                        a: R_X86_64_32 array            # Relocation entry

   e:   e8 00 00 00 00          callq  13 <main+0x13> # sum()
                        f: R_X86_64_PC32 sum-0x4        # Relocation entry
  13:   48 83 c4 08             add    $0x8,%rsp
  17:   c3                      retq
```

main.o

```
00000000004004d0 <main>:
  4004d0:       48 83 ec 08         sub    $0x8,%rsp
  4004d4:       be 02 00 00 00      mov    $0x2,%esi
  4004d9:       bf 18 10 60 00      mov    $0x601018,%edi   # %edi = &array
  4004de:       e8 05 00 00 00      callq  4004e8 <sum>     # sum()
  4004e3:       48 83 c4 08         add    $0x8,%rsp
  4004e7:       c3                  retq

00000000004004e8 <sum>:
  4004e8:       b8 00 00 00 00          mov    $0x0,%eax
  4004ed:       ba 00 00 00 00          mov    $0x0,%edx
  4004f2:       eb 09                   jmp    4004fd <sum+0x15>
  4004f4:       48 63 ca                movslq %edx,%rcx
  4004f7:       03 04 8f                add    (%rdi,%rcx,4),%eax
  4004fa:       83 c2 01                add    $0x1,%edx
  4004fd:       39 f2                   cmp    %esi,%edx
  4004ff:       7c f3                   jl     4004f4 <sum+0xc>
  400501:       f3 c3                   repz retq
```

Thanks for listening !