linker

Compiler Drivers

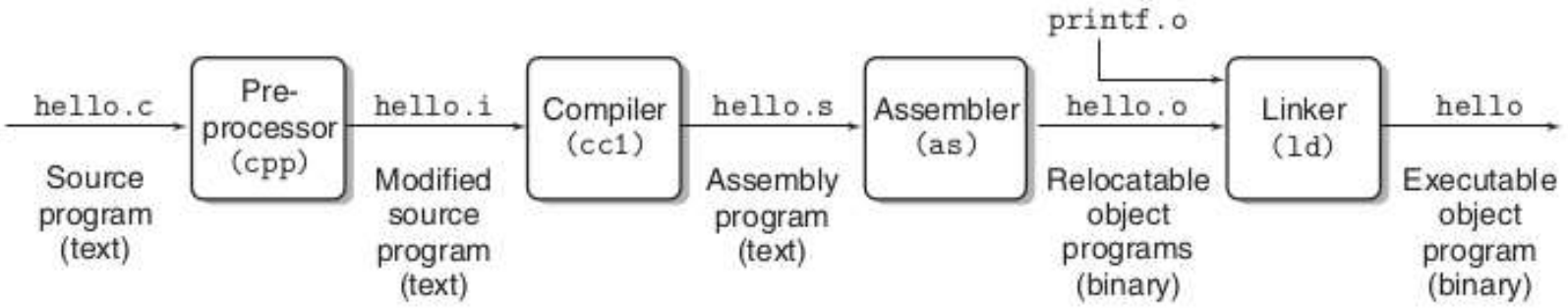


Figure 1.3 **The compilation system.**

Object Files

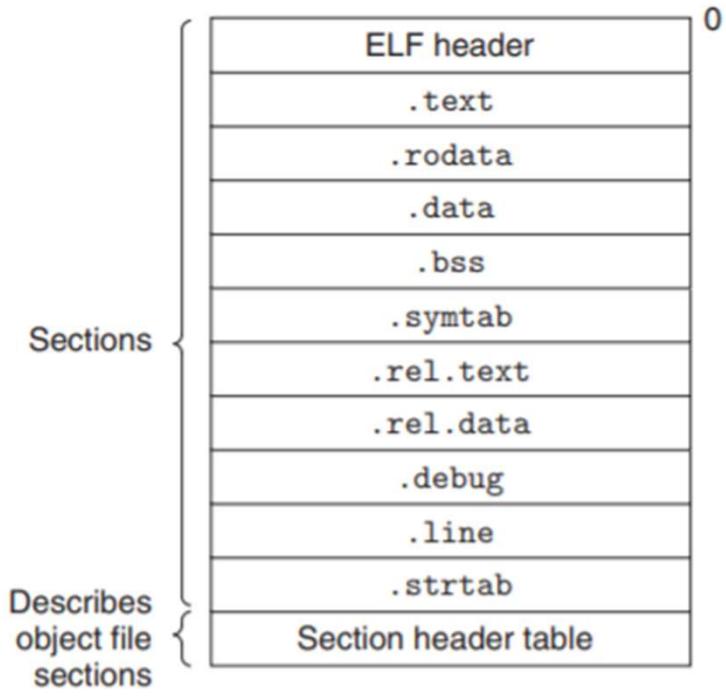
- 一種檔案類型
- Relocatable object file
  - 餵給linker的東西
  - 有binary code 跟 data
  - 可以跟其他的Relocatable object file組合成executable object file
  - Compiler 跟 assembler 生出來的
- Executable object file
  - 有binary code 跟 data
  - 可執行檔
  - Linker生出來的
- Shared object file
  - 在 load 跟 run time時可以被load 到 memory 跟 動態link 的 Relocatable object file
  - 之後應該會介紹

Object Files Format

- 每個系統不同
- Modern x86-64 Linux and Unix systems use Executable and Linkable Format (ELF)

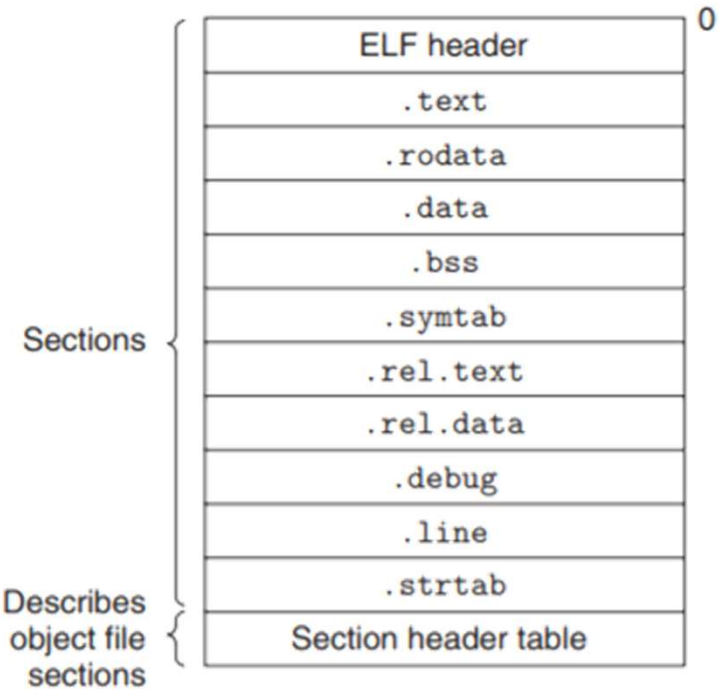
ELF

- header
  - 生成檔案的系統的word size
  - 生成檔案的系統的byte ordering
  - header 的 size
  - 檔案的type
  - machine type(ex:x86-64)
  - …



ELF

- .text
  - 程式碼
- .rodata
  - Read-only data
- .data
  - 有初始化的 global and static C variables
- .bss
  - 初始化為0或是沒有初始化的 global and static C variables
  - 這些變數在run time 才會占空間
- .symtab
  - A symbol table

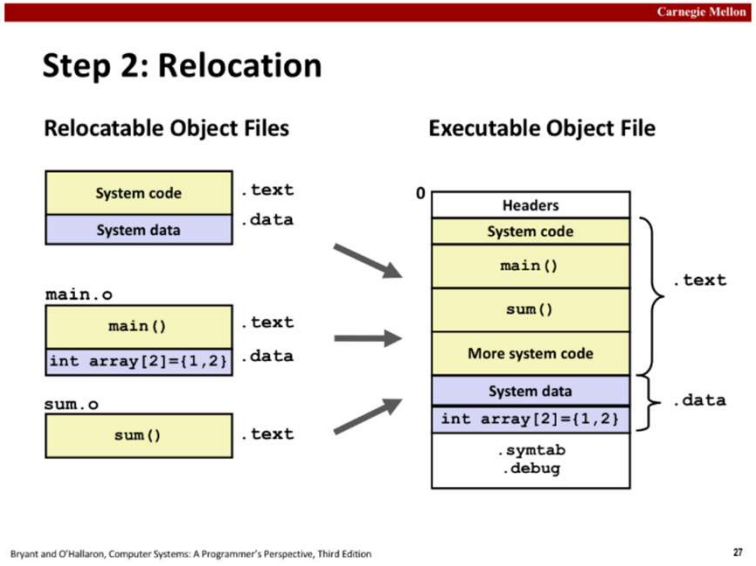


Symbol

- 變數或是函數
- 種類
  - *Global symbols :* 由 m 這個module定義，且可以被其他module引用的 symbol，像nonstatic 函數 and 全域變數
  - Global symbols : 由 m 這個module引用，但由其他module定義的 symbol 像別人定義的nonstatic 函數 and 全域變數
  - Local symbols : 只有m這個module可以用的，像是static的函數或是變數
  - Local symbols 不等於 local variables

Linker做的事

- **Symbol resolution**
  - 把每個symbol 連結到一個定義
  - 定義從symbol table找
- **Relocation**



Symbol Table

- Assembler產生
- 每一個ELF都有，包含在 .symtab section
- 一個型別長這樣的array

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

Figure 7.4 **ELF symbol table entry.** The `type` and `binding` fields are 4 bits each.

Symbol Resolution

- Foo.c

How Linkers Resolve Duplicate Symbol Names

- 在編譯階段， Compiler 會將每個 symbol 分類為 strong 或 weak 。
  - Strong Symbol ： 函數和被初始化過的全域變數。
  - Weak Symbol ： 未被初始化的全域變數。
- Assembler 會將這個資訊紀錄在 relocatable object file 的 symbol table 中。

How Linkers Resolve Duplicate Symbol Names

- Linker 利用以下列規則來決定如何做 Linking :
  1. 同時存在多個 strong symbol 是不允許的。
  2. 假設有一個 strong symbol 與多個 weak symbol ， Linker 應選擇 strong symbol 。
  3. 如果只有多個 weak symbol ，任意選擇其中一個 。

How Linkers Resolve Duplicate Symbol Names

- Example1:2個 strong symbol
- Example2:1個 strong symbol 1個 weak symbol
- Example3:型別不同
- Example4:2個 weak symbol

Linking with Static Libraries

- 編譯時執行
- 比較占空間
- 有用到的才會link

Link的過程

- set E:要被link的relocatable object files
- set U:還沒被定義的symbols
- set D:已經被定義的symbols

- link順序有影響