

Ligação estática, executável e imagem em memória

Bib: Computer Systems: A Programmer's Perspective (cap. 7, .6-.9)

Programação em Sistemas Computacionais

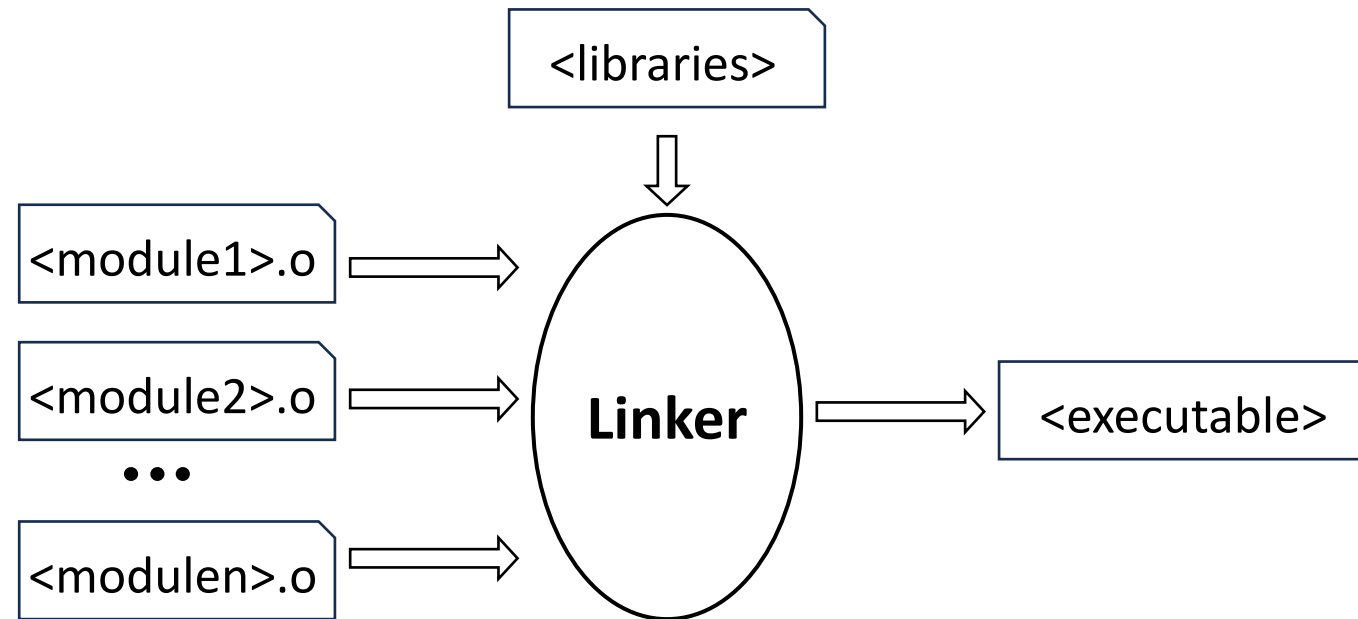
João Pedro Patriarca (joao.patriarca@isel.pt), Gabinete F.0.23 do edifício F

ISEL, ADEETC, LEIC

Agenda

- Ligação estática
 - Resolução de símbolos
 - Relocação
- Ficheiro objeto executável e imagem do executável em memória

Ligação estática – ligação de ficheiros objeto relocáveis



Ligação estática

- Baseia-se em duas tarefas principais:
 1. Resolução de símbolos: associa cada referência a um símbolo a uma única definição do símbolo
 2. Relocação (de notar que no ficheiro objeto relocável, cada secção de código e dados começa no endereço 0):
 1. Combina secções do mesmo tipo;
 2. Atribui endereços de memória no espaço de endereçamento da aplicação a cada secção e, por consequência, a cada símbolo;
 3. Modifica as referências aos símbolos no código e dados com base na nova posição de memória atribuída a cada símbolo.
- Os princípios aqui enunciados aplicam-se a qualquer ferramenta para ligação estática

Agenda

- Ligação estática
 - Resolução de símbolos
 - Relocação
- Ficheiro objeto executável e imagem do executável em memória

Classificação de símbolos versus visibilidade dos símbolos

- Símbolos fortes: funções e variáveis globais ~~inicializadas~~ (compilador com versão ≥ 10 considera qualquer variável global, iniciada ou não, um símbolo forte, por omissão)
- ~~Símbolos fracos: variáveis globais não inicializadas~~ (compilador com versão ≥ 10 não considera símbolos fracos, por omissão)
- Símbolos globais: símbolos definidos no módulo e passíveis de serem referenciados noutro módulo (funções e variáveis globais não `static`) ou referenciados no módulo mas definidos noutros módulos (funções ou variáveis globais declaradas como externas (`extern`))
- Símbolos locais: símbolos definidos e referenciados exclusivamente dentro do módulo (funções e variáveis globais marcadas com o qualificador `static`)
 - Não confundir com variáveis locais de uma função (não geram informação simbólica, a não ser que estejam marcadas com o qualificador `static`)

Regras na definição de múltiplos símbolos globais

1. Não são permitidos múltiplos símbolos fortes;
2. Na presença de um símbolo forte e múltiplos símbolos fracos, é selecionado o símbolo forte;
3. Na presença de múltiplos símbolos fracos é selecionado qualquer símbolo, tipicamente, o símbolo que ocupa maior espaço em memória.

Resolução de símbolos

- A resolução de símbolos locais é simples porque quer as referências quer as definições estão contidas dentro do módulo
 - O compilador gera erro de compilação se encontrar múltiplas definições do mesmo símbolo
- A resolução de símbolos globais é menos simples porque podem estar definidos noutros módulos
 - O *linker* gera erros de ligação nos casos:
 - Não encontra a definição do símbolo em nenhum dos módulos sujeitos a ligação
 - Encontra múltiplas definições fortes do mesmo símbolo

Aplicação da regra 1 – dois cenários que geram erro de ligação

Resolução de símbolos

- Múltiplas definições de funções globais

rule1_f1.c

```
int main() {  
    return 0;  
}
```

rule1_f2.c

```
int main() {  
    return 0;  
}
```

```
$ gcc rule1_f1.c rule1_f2.c -c  
$ nm rule1_f1.o rule1_f2.o  
rule1_f1.o:  
0...0000 T main  
rule1_f2.o:  
0...0000 T main
```

- Múltiplas definições de variáveis globais

rule1_v1.c

```
int x = 15213;  
long y = 123;  
int main() {  
    return 0;  
}
```

rule1_v2.c

```
int x = 15213;  
int y;  
void f() {}
```

```
$ gcc rule1_v1.c rule1_v2.c -c  
$ nm rule1_v1.o rule1_v2.o  
rule1_v1.o:  
0...0000 T main  
0...0000 D x  
0...0008 D y  
rule1_v2.o:  
0...0000 T f  
0...0000 D x  
0...0000 B y
```

Aplicação da regra 2 – símbolo forte e múltiplos símbolos fracos

Resolução de símbolos

rule2_m1.c

```
#include <stdio.h>
void f(void);
int x = 15; // Strong

int main() {
    f();
    printf("x = %d\n", x);
    return 0;
}
```

rule2_m2.c

```
int x; // Weak

void f() {
    x = 15212;
}
```



```
rule2_m1.o:
    U f
0...0000 T main
    U printf
0...0000 D x

rule2_m2.o:
0...0000 T f
0...0004 C x
```

```
$ gcc -fcommon -c rule2_m1.c rule2_m2.c
$ gcc -o rule2 rule2_m1.o rule2_m2.o
$ ./rule2
x = 15212
```

Aplicação da regra 3 – múltiplos símbolos fracos

Resolução de símbolos

rule3_m1.c

```
#include <stdio.h>
void f(void);
int x; // Weak

int main() {
    f();
    printf("x = %d\n", x);
    return 0;
}
```

rule3_m2.c

```
int x; // Weak

void f() {
    x = 15212;
}
```



```
rule3_m1.o:
    U f
0...0000 T main
    U printf
0...0004 C x

rule3_m2.o:
0...0000 T f
0...0004 C x
```

```
$ gcc -fcommon -c rule3_m1.c rule3_m2.c
$ gcc -o rule3 rule3_m1.o rule3_m2.o
$ ./rule3
x = 15212
```

Erros não detetados pelo *linker* (1 de 2)

Resolução de símbolos

Função externa não declarada

m1_errf.c

```
#include <stdio.h>
int main() {
    int x = f();
    printf("x = 0x%x\n", x);
    return 0;
}
```

m2_errf.c

```
float f() {
    return -1;
}
```

```
m1_errf.o:
             U f
0...0000 T main
             U printf
```

```
m2_errf.o:
0...0000 T f
```



```
$ gcc -o errf m1_errf.c m2_errf.c
m1_errf.c: In function 'main':
m1_errf.c:4:13: warning: implicit declaration of function 'f' ...
    4 |         int x = f();
      |                   ^
$ ./errf
x = 0x0
```

Erros não detetados pelo *linker* (2 de 2)

Resolução de símbolos

Múltipla definição de símbolos com tipos diferentes

m1_errv.c

```
#include <stdio.h>
void f(void);
int x = 15213;
int y = 15212;
int main() {
    f();
    printf("x = 0x%x "
           "y = 0x%x\n",
           x, y);
    return 0;
}
```

m2_errv.c

```
double x;

void f() {
    x = -0.0;
}
```



```
m1_errv.o:
0...0000 U f
0...0000 T main
0...0000 U printf
0...0000 D x
0...0004 D y

m2_errv.o:
0...0000 T f
0...0008 C x
```

```
$ gcc -fcommon -o errv m1_errv.c m2_errv.c
/usr/bin/ld: warning: alignment 4 of symbol `x' in /tmp/ccfez85R.o is
smaller than 8 in /tmp/ccjIPcXQ.o
$ ./errv
x = 0x0 y = 0x80000000
```

Exercícios de teste sobre resolução de símbolos

- Usar os enunciados publicados na meta disciplina de PSC

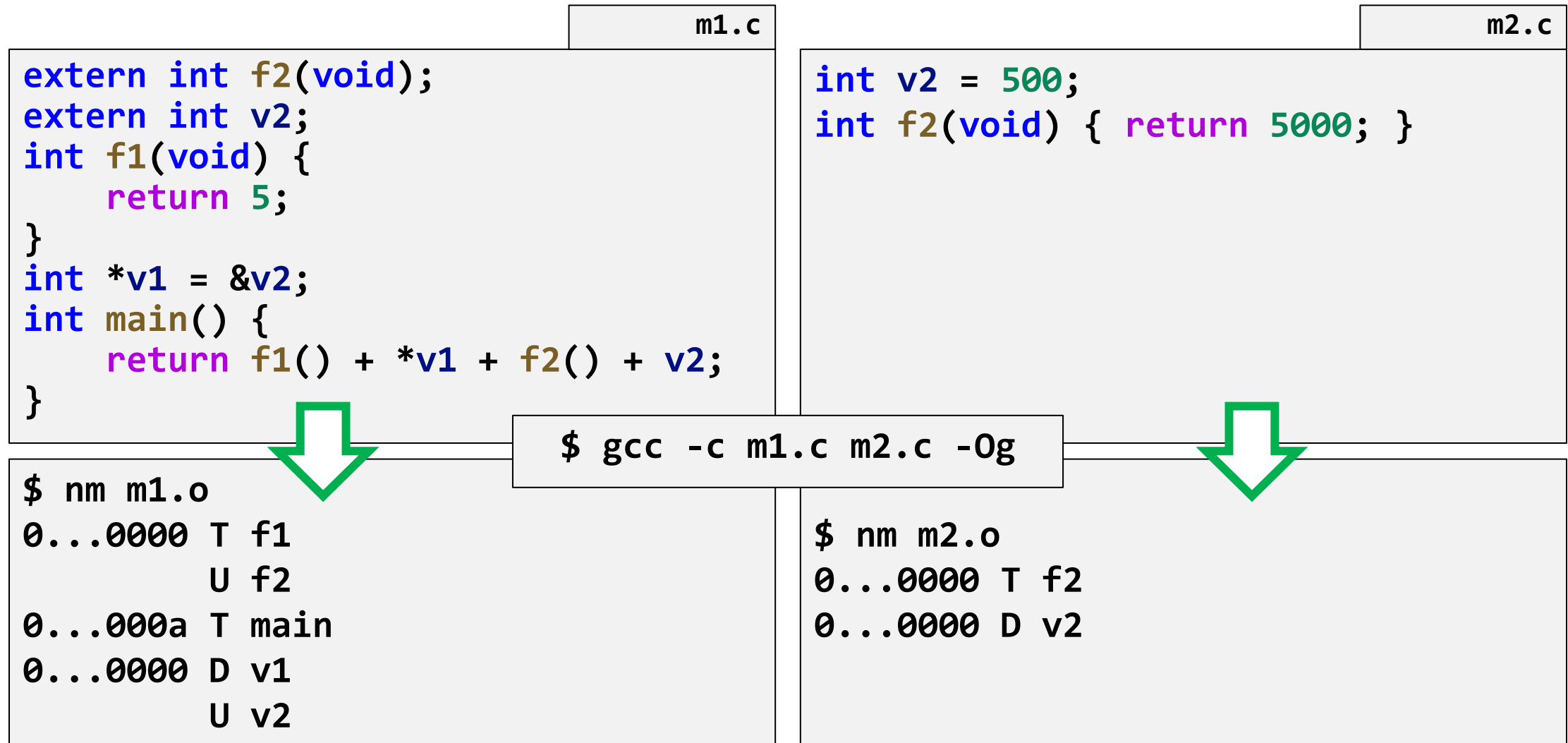
Agenda

- Ligação estática
 - Resolução de símbolos
 - Relocação
- Ficheiro objeto executável e imagem do executável em memória

Relocação

- Concluída a resolução de símbolos: cada referência tem associada uma única definição
- Relocação:
 1. Combina secções do mesmo tipo, convertendo numa única secção do tipo;
 2. Atribui endereços de memória no espaço de endereçamento da aplicação a cada secção e, por consequência, a cada símbolo definido;
 3. Modifica as referências aos símbolos no código e dados com base na nova posição de memória atribuída a cada símbolo.

Exemplo para ilustração do processo de relocação



Tabelas de relocação e *disassembly* da secção **.text** de **m1.o**

Exemplo para ilustração do processo de relocação

```
$ objdump -r m1.o
```

```
m1.o: file format elf64-x86-64
```

```
RELOCATION RECORDS FOR [.text]:
```

| OFFSET | TYPE | VALUE |
|----------|----------------|--------|
| 0...0010 | R_X86_64_PLT32 | f1-0x4 |
| 0...0017 | R_X86_64_PC32 | v1-0x4 |
| 0...0020 | R_X86_64_PLT32 | f2-0x4 |
| 0...0028 | R_X86_64_PC32 | v2-0x4 |

```
RELOCATION RECORDS FOR [.data.rel]:
```

| OFFSET | TYPE | VALUE |
|----------|-------------|-------|
| 0...0000 | R_X86_64_64 | v2 |

Relativo

Absoluto

```
$ objdump -d m1.o
```

```
0000000000000000 <f1>:
```

```
0:  f3 0f 1e fa    endbr64
4:  b8 05 00 00 00  mov    $0x5,%eax
9:  c3             retq
```

```
000000000000000a <main>:
```

```
a:  f3 0f 1e fa    endbr64
e:  53             push   %rbx
f:  e8 00 00 00 00  callq  <main+0xa>
14: 48 8b 15 00 00 00 00  mov    0x0(%rip),%rdx
1b: 03 02          add    (%rdx),%eax
1d: 89 c3          mov    %eax,%ebx
1f: e8 00 00 00 00  callq  <main+0x1a>
24: 01 d8          add    %ebx,%eax
26: 03 05 00 00 00 00  add    0x0(%rip),%eax
2c: 5b            pop    %rbx
2d: c3            retq
```

Conteúdos das secções **.text** de **m1.o** e de **m2.o**

Exemplo para ilustração do processo de relocação

```
$ objdump -s -j .text m1.o
m1.o:      file format elf64-x86-64
```

Contents of section **.text**:

```
0000 f30f1efa b8050000 00c3f30f 1efa53e8 .....S.
0010 00000000 488b1500 00000000 03 0289c3e8 ....H.....
0020 00000000 01d80305 00000000 5bc3 .....[.
```

} Size = 0x2E

```
$ objdump -s -j .text m2.o
m2.o:      file format elf64-x86-64
```

Contents of section **.text**:

```
0000 f30f1efa b8881300 00c3 .....
```

} Size = 0x0A

```
$ gcc -o m m1.o m2.o
```

Gera o executável

Disassembly da secção **.text** de **m (\$objdump -d m)**

Exemplo para ilustração do processo de relocação

```
0000000000001129 <f1>:
  1129:      f3 0f 1e fa      endbr64
  112d:      b8 05 00 00 00    mov     $0x5,%eax
  1132:      c3              retq

0000000000001133 <main>:
  1133:      f3 0f 1e fa      endbr64
  1137:      53              push    %rbx
  1138:      e8 [redacted]      callq   1129 <f1>
  113d:      48 8b 15 [blue]     mov     [blue](&rip),%rdx    # 4010 <v1>
  1144:      03 02              add     (%rdx),%eax
  1146:      89 c3              mov     %eax,%ebx
  1148:      e8 [green]         callq   1157 <f2>
  114d:      01 d8              add     %ebx,%eax
  114f:      03 05 [magenta]     add     [magenta](&rip),%eax # 4018 <v2>
  1155:      5b              pop     %rbx
  1156:      c3              retq

0000000000001157 <f2>:
  1157:      f3 0f 1e fa      endbr64
  115b:      b8 88 13 00 00    mov     $0x1388,%eax
  1160:      c3              retq
```

Disassembly da secção **.text** de **m (\$objdump -d m)**

Exemplo para ilustração do processo de relocação

```
0000000000001129 <f1>:
    1129:      f3 0f 1e fa                endbr64
    112d:      b8 05 00 00 00          mov     $0x5,%eax
    1132:      c3                      retq
0000000000001133 <main>:
    1133:      f3 0f 1e fa                endbr64
    1137:      53                      push    %rbx
    1138:      e8 ec ff ff ff          callq   1129 <f1>
    113d:      48 8b 15 cc 2e 00 00      mov     0x2ecc(%rip),%rdx # 4010 <v1>
    1144:      03 02                      add     (%rdx),%eax
    1146:      89 c3                      mov     %eax,%ebx
    1148:      e8 0a 00 00 00          callq   1157 <f2>
    114d:      01 d8                      add     %ebx,%eax
    114f:      03 05 c3 2e 00 00      add     0x2ec3(%rip),%eax # 4018 <v2>
    1155:      5b                      pop     %rbx
    1156:      c3                      retq
0000000000001157 <f2>:
    1157:      f3 0f 1e fa                endbr64
    115b:      b8 88 13 00 00          mov     $0x1388,%eax
    1160:      c3                      retq
```

Conteúdo da secção **.data** de **m**

Exemplo para ilustração do processo de relocação

```
$ objdump -j .data -s m
```

```
m:      file format elf64-x86-64
```

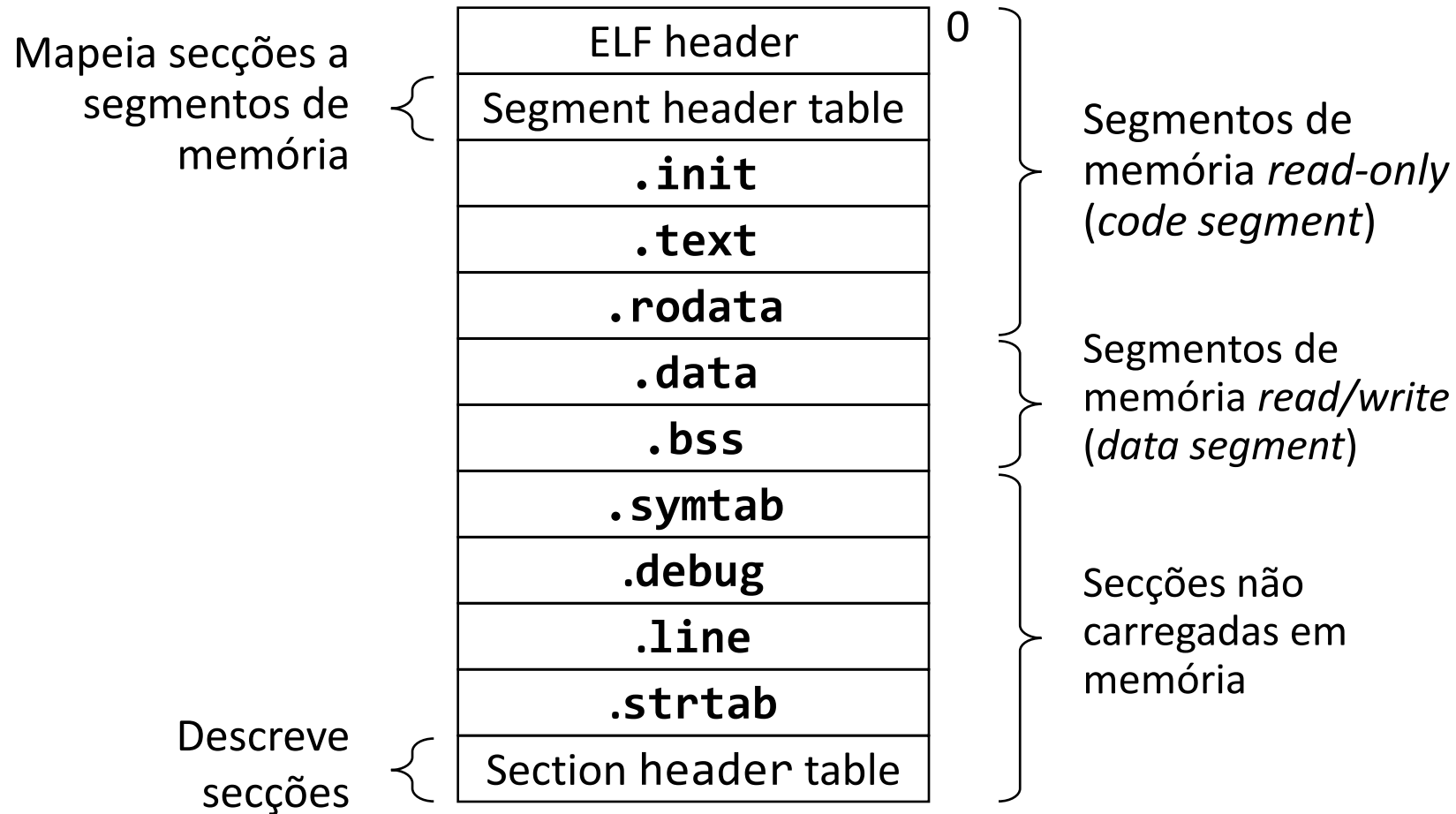
```
Contents of section .data:
```

```
4000 00000000 00000000 08400000 00000000 .....@.....
4010 18400000 00000000 f4010000 .@.....
```

Agenda

- Ligação estática
 - Resolução de símbolos
 - Relocação
- Ficheiro objeto executável e imagem do executável em memória

Estrutura típica de um ficheiro objeto executável



Impressão da tabela de símbolos

```
$ nm m
0...0038c r __abi_tag
0...0401c B __bss_start
0...0401c b completed.0
           w __cxa_finalize@GLIBC_2.2.5
0...04000 D __data_start
0...04000 W data_start
0...01070 t deregister_tm_clones
0...010e0 t __do_global_dtors_aux
0...03df8 d __do_global_dtors_aux...
0...04008 D __dso_handle
0...03e00 d _DYNAMIC
0...0401c D _edata
0...04020 B _end
0...01129 T f1
0...01157 T f2
```

```
0...01164 T _fini
0...01120 t frame_dummy
0...03df0 d __frame_dummy_init...
0...020f4 r __FRAME_END__
0...03fc0 d _GLOBAL_OFFSET_TABLE_
           w __gmon_start__
0...02004 r __GNU_EH_FRAME_HDR
0...01000 T _init
0...02000 R _IO_stdin_used
           w _ITM_deregisterTMCloneTable
           w _ITM_registerTMCloneTable
           U __libc_start_main@GLIBC_2.34
0...01133 T main
0...010a0 t register_tm_clones
0...01040 T _start
0...04020 D __TMC_END__
0...04010 D v1
0...04018 D v2
```

Impressão dos *headers* das secções

```
$ objdump -h m
m:      file format elf64-x86-64
```

Sections:

| Idx | Name | Size | VMA | LMA | File off | Algn | |
|-----|--------------------|----------|-----------|-----------|----------|------|---------------------------------------|
| 0 | .interp | 0000001c | 0...00318 | 0...00318 | 00000318 | 2**0 | CONTENTS, ALLOC, LOAD, READONLY, DATA |
| 1 | .note.gnu.property | 030 | 0...00338 | 0...00338 | 00000338 | 2**3 | CONTENTS, ALLOC, LOAD, READONLY, DATA |
| 2 | .note.gnu.build-id | 024 | 0...00368 | 0...00368 | 00000368 | 2**2 | CONTENTS, ALLOC, LOAD, READONLY, DATA |
| 3 | .note.ABI-tag | 00000020 | 0...0038c | 0...0038c | 0000038c | 2**2 | CONTENTS, ALLOC, LOAD, READONLY, DATA |
| 4 | .gnu.hash | 00000024 | 0...003b0 | 0...003b0 | 000003b0 | 2**3 | CONTENTS, ALLOC, LOAD, READONLY, DATA |
| 5 | .dynsym | 00000090 | 0...003d8 | 0...003d8 | 000003d8 | 2**3 | CONTENTS, ALLOC, LOAD, READONLY, DATA |
| 6 | .dynstr | 00000088 | 0...00468 | 0...00468 | 00000468 | 2**0 | CONTENTS, ALLOC, LOAD, READONLY, DATA |
| 7 | .gnu.version | 0000000c | 0...004f0 | 0...004f0 | 000004f0 | 2**1 | CONTENTS, ALLOC, LOAD, READONLY, DATA |
| 8 | .gnu.version_r | 00000030 | 0...00500 | 0...00500 | 00000500 | 2**3 | CONTENTS, ALLOC, LOAD, READONLY, DATA |
| 9 | .rela.dyn | 000000d8 | 0...00530 | 0...00530 | 00000530 | 2**3 | CONTENTS, ALLOC, LOAD, READONLY, DATA |
| 10 | .init | 0000001b | 0...01000 | 0...01000 | 00001000 | 2**2 | CONTENTS, ALLOC, LOAD, READONLY, CODE |
| 11 | .plt | 00000010 | 0...01020 | 0...01020 | 00001020 | 2**4 | CONTENTS, ALLOC, LOAD, READONLY, CODE |
| 12 | .plt.got | 00000010 | 0...01030 | 0...01030 | 00001030 | 2**4 | CONTENTS, ALLOC, LOAD, READONLY, CODE |
| 13 | .text | 00000121 | 0...01040 | 0...01040 | 00001040 | 2**4 | CONTENTS, ALLOC, LOAD, READONLY, CODE |
| 14 | .fini | 0000000d | 0...01164 | 0...01164 | 00001164 | 2**2 | CONTENTS, ALLOC, LOAD, READONLY, CODE |
| 15 | .rodata | 00000004 | 0...02000 | 0...02000 | 00002000 | 2**2 | CONTENTS, ALLOC, LOAD, READONLY, DATA |
| 16 | .eh_frame_hdr | 0000003c | 0...02004 | 0...02004 | 00002004 | 2**2 | CONTENTS, ALLOC, LOAD, READONLY, DATA |
| 17 | .eh_frame | 000000b8 | 0...02040 | 0...02040 | 00002040 | 2**3 | CONTENTS, ALLOC, LOAD, READONLY, DATA |
| 18 | .init_array | 00000008 | 0...03df0 | 0...03df0 | 00002df0 | 2**3 | CONTENTS, ALLOC, LOAD, DATA |
| 19 | .fini_array | 00000008 | 0...03df8 | 0...03df8 | 00002df8 | 2**3 | CONTENTS, ALLOC, LOAD, DATA |
| 20 | .dynamic | 000001c0 | 0...03e00 | 0...03e00 | 00002e00 | 2**3 | CONTENTS, ALLOC, LOAD, DATA |
| 21 | .got | 00000040 | 0...03fc0 | 0...03fc0 | 00002fc0 | 2**3 | CONTENTS, ALLOC, LOAD, DATA |
| 22 | .data | 0000001c | 0...04000 | 0...04000 | 00003000 | 2**3 | CONTENTS, ALLOC, LOAD, DATA |
| 23 | .bss | 00000004 | 0...0401c | 0...0401c | 0000301c | 2**0 | ALLOC |
| 24 | .comment | 00000026 | 0...00000 | 0...00000 | 0000301c | 2**0 | |

Imagem do executável em memória

