Link 3.B Object Files - listings

Young W. Lim

2018-04-05 Fri

Outline

Based on

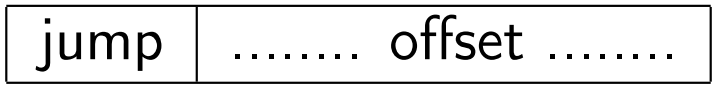
"Self-service Linux: Mastering the Art of Problem Determination",
Mark Wilding
"Computer Architecture: A Programmer's Perspective",
Bryant & O'Hallaron

Compling 32-bit program on 64-bit gcc

- gcc -v
- gcc -m32 t.c
- sudo apt-get install gcc-multilib
- sudo apt-get install g++-multilib
- gcc-multilib
- g++-multilib
- gcc -m32
- objdump -m i386

PC-relative addressing

- jump relative



- effective PC address = next instruction address + *offset*
  (offset may be negative)
- particularly useful in connection with jumps,
  because typical jumps are to nearby instructions
- most `if` or `while` statements are reasonably short
- another advantage is +position-independent_ code

`https://en.wikipedia.org/wiki/Addressing_mode#PC-relative`

an example of PC-relative addressing

```
  jle .L4              If <=, goto dest2
   .p2align 4,,7       Aligns next instruction to multpile of 8
.L5:                   dest1:
  movl %edx, %eax
  sarl $1, %eax
  subl %eax, %edx
  jg .L5               If >, goto dest1
.L4:                   dest2:
  movl %edx, %eax
```

Computer Architecture : A Programmer's Prespective

object files

disassembled version

```
1.   8: 7e 11                jle  1b <silly+0x1b>    Target = dest2
2.   a: 8d b6 00 00 00 00    lea  0x0(%esi),%esi     Added nops
3.  10: 89 d0                mov  %edx,%eax          dest1:
4.  12: c1 f8 01             sar  $0x1,%eax
5.  15: 29 c2                sub  %eax,%edx
6.  17: 85 d2                test %edx,%edx          %edx & %edx = %edx
7.  19: 7f f5                jg   10 <silly+0x10>    Target = dest1
8.  1b: 89 d0                mov  %edx,%eax          dest2:
```

disassembled version after linking

```
1.  80483c8: 7e 11                jle  79473db <silly+0x1b>  Target = dest2
2.  80483ca: 8d b6 00 00 00 00    lea  0x0(%esi),%esi        Added nops
3.  80483d0: 89 d0                mov  %edx,%eax             dest1:
4.  80483d2: c1 f8 01             sar  $0x1,%eax
5.  80483d5: 29 c2                sub  %eax,%edx
6.  80483d7: 85 d2                test %edx,%edx             %edx & %edx = %edx
7.  80483d9: 7f f5                jg   80483d0 <silly+0x10>  Target = dest1
8.  80483db: 89 d0                mov  %edx,%eax             dest2:
```

PC-relative offset

Jump Forward

```
1.   8: 7e 11                jle  1b <silly+0x1b>    Target = dest2
2.   a: 8d b6 00 00 00 00    lea  0x0(%esi),%esi     Added nops
```

- jump target : 0x1b (27)
- jump target encoding : 0x11 + 0xa = 0x1b (17 + 10 =27)
- next instruction address : 0xa (10)

Jump Backward

```
7.  19: 7f f5                jg   10 <silly+0x10>    Target = dest1
8.  1b: 89 d0                mov  %edx,%eax          dest2:
```

- jump target : 0x10 (16)
- jump target encoding : 0xf5 + 0x1b = 0xf5 (-11 + 27 =16)
- next instruction address : 0x1b (27)

disassembled version after linking notes

Jump Forward

```
1.   8: 7e 11                jle  1b <silly+0x1b>    Target = dest2

7.  19: 7f f5                jg   10 <silly+0x10>    Target = dest1
```

Jump Backward

```
1.  80483c8: 7e 11           jle  79473db <silly+0x1b>  Target = dest2

7.  80483d9: 7f f5           jg   80483d0 <silly+0x10>  Target = dest1
```

- the instruction have been relocated to different addresses, but the encondings of the jump targets in line 1 and line 7 remain unchanged
- by using PC-relative encoding of the jump targets, the instructions can be completely encoded (requiring just two bytes) and the object code can be shifted to different positions in memory without modification.

Encoding format of object code

- the format of object code
- understanding how the targets of jump instructions are encoded will be important
  - when studying linking process
  - interpreting the output of a disassembler
- In assembly code, jump targets are written using symbolic labels
- the assembler, and later the linker, generate the proper encodings of the jump targets
- there are several different encodings of for jumps, but some of the most commonly used ones are PC-relative

`Computer Architecture : A Programmer's Prespective`

Encoding jump instructions

- PC-reltive
  - encodes the difference between the address of the target instruction the address of the instruction immediately following the jump
  - these offsets can be encoded using 1, 2, or 4 bytes
- Absolute
  - directly specify the target address using 4 bytes
- the assembler and linker select the appropriate encodings

Jump Instruction

EB cb	JMP rel8	Jump short, relative, disp relative to next inst
E9 cw	JMP rel16	Jump near, relative, disp relative to next inst
E9 cd	JMP rel32	Jump near, relative, disp relative to next inst
FF /4	JMP r/m16	Jump near, absolute indirect, address in r/m16.
FF /4	JMP r/m32	Jump near, absolute indirect, address in r/m32.
EA cd	JMP ptr16:16	Jump far, absolute, address given in operand.
EA cp	JMP ptr16:32	Jump far, absolute, address given in operand.
FF /5	JMP m16:16	Jump far, absolute indirect, address in m16:16.
FF /5	JMP m16:32	Jump far, absolute indirect, address in m16:32.

`https://c9x.me/x86/html/file_module_x86_id_147.html`

Call Instruction

E8 cw	CALL rel16	Call near, relative, disp relative to next inst
E8 cd	CALL rel32	Call near, relative, disp relative to next inst
FF /2	CALL r/m16	Call near, absolute indirect, address in r/m16
FF /2	CALL r/m32	Call near, absolute indirect, address in r/m32
9A cd	CALL ptr16:16	Call far, absolute, address given in operand
9A cp	CALL ptr16:32	Call far, absolute, address given in operand
FF /3	CALL m16:16	Call far, absolute indirect, address in m16:16
FF /3	CALL m16:32	Call far, absolute indirect, address in m16:32

https://c9x.me/x86/html/file_module_x86_id_147.html

absolute offset in jmp

r/m16 or r/m32

- an absolute offset is specified indirectly
  in a general-purpose register or a memory location
- the operand-size attribute determines
  the size of the target operand (16 or 32 bits).
- absolute offsets are loaded directly into the EIP register.
- If the operand-size attribute (for *near relative* jumps) is 16,
  the upper two bytes of the EIP register are cleared,
  resulting in a maximum instruction pointer size of 16 bits.

https://c9x.me/x86/html/file_module_x86_id_147.html

relative offset in jmp

rel8, rel16, or rel32

- a relative offset is generally specified
  as a label in assembly code,
- but at the machine code level,
  it is encoded as a signed 8-, 16-, or 32-bit immediate value
- this value is added to the value in the EIP register.
- the EIP contains the address of the next instruction
  following the JMP instruction
- When using relative offsets, the opcode (for short vs. near jumps)
  and the operand-size attribute (for *near relative* jumps)
  determines the size of the target operand (8, 16, or 32 bits).

https://c9x.me/x86/html/file_module_x86_id_147.html

FF /2, FF /3 : CALL, FF /4, FF /5 : JMP

- /digit — a digit between 0 and 7 indicates that
  - the ModR/M byte of the instruction uses only the R/M (register or memory) operand.
  - the REG field contains the digit that provides an extension to the instruction's opcode.
  - MOD(2 bits), REG(3 bits), R/M(3 bits)
- /r — indicates that the ModR/M byte of the instruction contains a REG operand and an R/M operand.

`https://stackoverflow.com/questions/24295464/what-does-the-4-mean-in-ff-4`

ModR/M : encoding x86 instruciton operands

- The MOD-REG-R/M byte specifies
  - instruction operands and their
  - addressing mod
- MOD(2 bits), REG(3 bits), R/M(3 bits)
- MOD specifies x86 addressing mode
- REG specifies source or destination register
- R/M combined with MOD specifies
  1. the 2nd operand in a two operand instruction, or
  2. the only operand in a single operand instruction (NOT, NEG)

`http://www.c-jump.com/CIS77/CPU/x86/X77_0060_mod_reg_r_m_byte.htm`

Reg : encoding x86 instruciton operands

REG value	Reg when data size is 8 bits	Reg when data size is 16 bits	Reg when data size is 32 bits
000	al	ax	eax
001	cl	cx	ecx
010	dl	dx	edx
011	bl	bx	ebx
100	ah	sp	esp
101	ch	bp	ebp
110	dh	si	esi
111	bh	di	edi

- d bit in the opcode : source/destination
  - d=0 : `MODR/M <- REG` : REG is the source
  - d=1 : `REG <- MODR/M` : REG is the destination

`http://www.c-jump.com/CIS77/CPU/x86/X77_0060_mod_reg_r_m_byte.htm`

16-bit addressing forms with the ModR/M byte

effective address	Mod	R/M	
[BX+SI]	00	000	
[BX+DI]	00	001	
[BP+SI]	00	010	/2
[BP+DI]	00	011	/3
[SI]	00	100	/4
[DI]	00	101	/5
disp16	00	110	
[BX]	00	111	

https://gerardnico.com/_detail/computer/cpu/modr_16bit_intel_addressing_form.jpg?

cb, cw, cd, cp, co, ct

cb	1-byte
cw	2-byte
cd	4-byte
co	8-byte
ct	10-byte

https://stackoverflow.com/questions/15017659/how-to-read-the-intel-opcode-notatio

rel8, rel16, rel32

- rel8 — A relative address in the range
  from 128 bytes before the end of the instruction
  to 127 bytes after the end of the instruction.
- rel16, rel32 — A relative address within the same code segment
  as the instruction assembled.
  - The rel16 symbol applies to instructions
    with an operand-size attribute of 16 bits;
  - the rel32 symbol applies to instructions
    with an operand-size attribute of 32 bits.

https://gerardnico.com/_detail/computer/cpu/modr_16bit_intel_addressing_form.jpg?

ptr16:16, ptr32:32

- ptr16:16, ptr32:32 — A far pointer,
  typically to a different code segment

- the value to the left of the colon is a 16-bit selector or
  value destined for the code segment register.
- the value to the right corresponds to
  the offset within the destination segment.

- ptr16:16 when the operand size attribute is 16 bits
- ptr16:32 when the operand size attribute is 32 bits

https://gerardnico.com/_detail/computer/cpu/modr_16bit_intel_addressing_form.jpg?

m16:16, m32:32

m16:16, m32:32, m64:64

- a memory operand containing a far pointer composed of two numbers.
- the number to the left of the colon corresponds to the pointer's segment selector.
- The number to the right corresponds to its offset.

https://gerardnico.com/_detail/computer/cpu/modr_16bit_intel_addressing_form.jpg?

r/m16, r/m32

r/m16, r/m32

- a general-purpose register or memory operand
- The contents of memory are found at the address provided by the effective address computation.
- when an instruction's operand size attribute is 16 bits
  - The word general-purpose registers are:
    AX, CX, DX, BX, SP, BP, SI, DI.
- when an instruction's operand size attribute is 32 bits
  - The doubleword registers are:
    EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI.

https://gerardnico.com/_detail/computer/cpu/modr_16bit_intel_addressing_form.jpg?

Relocatable Object Files

1. relocatable object dump (main.o)
2. relocatable object dump (swap.o)

relocatable objdump -d main.o

```
main.o:     formato del fichero elf32-i386


Desensamblado de la sección .text.startup:

00000000 <main>:
   0:   8d 4c 24 04             lea    0x4(%esp),%ecx
   4:   83 e4 f0                and    $0xfffffff0,%esp
   7:   ff 71 fc                pushl  -0x4(%ecx)
   a:   55                      push   %ebp
   b:   89 e5                   mov    %esp,%ebp
   d:   51                      push   %ecx
   e:   83 ec 04                sub    $0x4,%esp
  11:   e8 fc ff ff ff          call   12 <main+0x12>
  16:   83 c4 04                add    $0x4,%esp
  19:   31 c0                   xor    %eax,%eax
  1b:   59                      pop    %ecx
  1c:   5d                      pop    %ebp
  1d:   8d 61 fc                lea    -0x4(%ecx),%esp
  20:   c3                      ret
```

relocatable objdump -d swap.o

```
swap.o:     formato del fichero elf32-i386


Desensamblado de la sección .text:

00000000 <swap>:
   0:   a1 00 00 00 00          mov    0x0,%eax
   5:   8b 0d 04 00 00 00       mov    0x4,%ecx
   b:   c7 05 00 00 00 00 04    movl   $0x4,0x0
  12:   00 00 00
  15:   8b 10                   mov    (%eax),%edx
  17:   89 08                   mov    %ecx,(%eax)
  19:   89 15 04 00 00 00       mov    %edx,0x4
  1f:   c3                      ret
```

Executable Object Files

1. executable object dump (main)
2. executable object dump (swap)
3. executable object dump (section summary)

executable objdump -d p (disassemble) - main

```
080482e0 <main>:
 80482e0:       8d 4c 24 04             lea    0x4(%esp),%ecx
 80482e4:       83 e4 f0                and    $0xfffffff0,%esp
 80482e7:       ff 71 fc                pushl  -0x4(%ecx)
 80482ea:       55                      push   %ebp
 80482eb:       89 e5                   mov    %esp,%ebp
 80482ed:       51                      push   %ecx
 80482ee:       83 ec 04                sub    $0x4,%esp
 80482f1:       e8 0a 01 00 00          call   8048400 <swap>
 80482f6:       83 c4 04                add    $0x4,%esp
 80482f9:       31 c0                   xor    %eax,%eax
 80482fb:       59                      pop    %ecx
 80482fc:       5d                      pop    %ebp
 80482fd:       8d 61 fc                lea    -0x4(%ecx),%esp
 8048300:       c3                      ret
```

executable objdump -d p (disassemble) - swap

```
08048400 <swap>:
 8048400:       a1 20 a0 04 08          mov    0x804a020,%eax
 8048405:       8b 0d 1c a0 04 08       mov    0x804a01c,%ecx
 804840b:       c7 05 28 a0 04 08 1c    movl   $0x804a01c,0x804a028
 8048412:       a0 04 08
 8048415:       8b 10                   mov    (%eax),%edx
 8048417:       89 08                   mov    %ecx,(%eax)
 8048419:       89 15 1c a0 04 08       mov    %edx,0x804a01c
 804841f:       c3                      ret
```

executable objdump -d p (disassemble) - section summary

```
./p:     formato del fichero elf32-i386
Desensamblado de la sección .init: .........................
0804828c <_init>:
Desensamblado de la sección .plt: ..........................
080482b0 <__libc_start_main@plt-0x10>:
080482c0 <__libc_start_main@plt>:
Desensamblado de la sección .plt.got: ......................
080482d0 <.plt.got>:
Desensamblado de la sección .text: .........................
080482e0 <main>:
08048301 <_start>:
08048330 <__x86.get_pc_thunk.bx>:
08048340 <deregister_tm_clones>:
08048370 <register_tm_clones>:
080483b0 <__do_global_dtors_aux>:
080483d0 <frame_dummy>:
08048400 <swap>:
08048420 <__libc_csu_init>:
08048480 <__libc_csu_fini>:
Desensamblado de la sección .fini: .........................
08048484 <_fini>:
```

ELF Tables of Example Codes

- readelf -h main.o
- readelf -h swap.o
- readelf -h p
- readelf -S main.o
- readelf -S swap.o
- readelf -S p

readelf -h main.o

```
Encabezado ELF:
  Mágico:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Clase:                             ELF32
  Datos:                             complemento a 2, little endian
  Versión:                           1 (current)
  OS/ABI:                            UNIX - System V
  Versión ABI:                       0
  Tipo:                              REL (Fichero reubicable)
  Máquina:                           Intel 80386
  Versión:                           0x1
  Dirección del punto de entrada:               0x0
  Inicio de encabezados de programa:          0 (bytes en el fichero)
  Inicio de encabezados de sección:         524 (bytes en el fichero)
  Opciones:                          0x0
  Tamaño de este encabezado:         52 (bytes)
  Tamaño de encabezados de programa: 0 (bytes)
  Número de encabezados de programa: 0
  Tamaño de encabezados de sección:  40 (bytes)
  Número de encabezados de sección:  12
  Índice de tabla de cadenas de sección de encabezado: 9
```

readelf -h swap.o

```
Encabezado ELF:
  Mágico:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Clase:                             ELF32
  Datos:                             complemento a 2, little endian
  Versión:                           1 (current)
  OS/ABI:                            UNIX - System V
  Versión ABI:                       0
  Tipo:                              REL (Fichero reubicable)
  Máquina:                           Intel 80386
  Versión:                           0x1
  Dirección del punto de entrada:               0x0
  Inicio de encabezados de programa:          0 (bytes en el fichero)
  Inicio de encabezados de sección:         596 (bytes en el fichero)
  Opciones:                          0x0
  Tamaño de este encabezado:         52 (bytes)
  Tamaño de encabezados de programa: 0 (bytes)
  Número de encabezados de programa: 0
  Tamaño de encabezados de sección:  40 (bytes)
  Número de encabezados de sección:  13
  Índice de tabla de cadenas de sección de encabezado: 10
```

readelf -h p

```
Mágico:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
Clase:                             ELF32
Datos:                             complemento a 2, little endian
Versión:                           1 (current)
OS/ABI:                            UNIX - System V
Versión ABI:                       0
Tipo:                              EXEC (Fichero ejecutable)
Máquina:                           Intel 80386
Versión:                           0x1
Dirección del punto de entrada:               0x8048301
Inicio de encabezados de programa:          52 (bytes en el fichero)
Inicio de encabezados de sección:          6180 (bytes en el fichero)
Opciones:                          0x0
Tamaño de este encabezado:         52 (bytes)
Tamaño de encabezados de programa: 32 (bytes)
Número de encabezados de programa: 9
Tamaño de encabezados de sección:  40 (bytes)
Número de encabezados de sección:  31
Índice de tabla de cadenas de sección de encabezado: 28
```

readelf -S main.o

```
Hay 12 encabezados de sección, comenzando en el desplazamiento: 0x20c:

Encabezados de Sección:
  [Nr] Nombre              Tipo            Direc    Desp   Tam    ES Opt En Inf Al
  [ 0]                     NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text               PROGBITS        00000000 000034 000024 00  AX  0   0  1
  [ 2] .rel.text           REL             00000000 0001a4 000008 08   I 10   1  4
  [ 3] .data               PROGBITS        00000000 000058 000008 00  WA  0   0  4
  [ 4] .bss                NOBITS          00000000 000060 000000 00  WA  0   0  1
  [ 5] .comment            PROGBITS        00000000 000060 000035 01  MS  0   0  1
  [ 6] .note.GNU-stack     PROGBITS        00000000 000095 000000 00      0   0  1
  [ 7] .eh_frame           PROGBITS        00000000 000098 000044 00   A  0   0  4
  [ 8] .rel.eh_frame       REL             00000000 0001ac 000008 08   I 10   7  4
  [ 9] .shstrtab           STRTAB          00000000 0001b4 000057 00      0   0  1
  [10] .symtab             SYMTAB          00000000 0000dc 0000b0 10     11   8  4
  [11] .strtab             STRTAB          00000000 00018c 000016 00      0   0  1
Clave para Opciones:
  W (escribir), A (asignar), X (ejecutar), M (mezclar), S (cadenas)
  I (info), L (orden enlazado), G (grupo), T (TLS), E (excluir)
  x (desconocido), O (se requiere procesamiento extra del SO)
  o (específico del SO), p (específico del procesador)
```

readelf -S swap.o

```
Hay 13 encabezados de sección, comenzando en el desplazamiento: 0x254:

Encabezados de Sección:
  [Nr] Nombre              Tipo            Direc    Desp   Tam    ES Opt En Inf Al
  [ 0]                     NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text               PROGBITS        00000000 000034 000036 00  AX  0   0  1
  [ 2] .rel.text           REL             00000000 0001b8 000030 08   I 11   1  4
  [ 3] .data               PROGBITS        00000000 00006c 000004 00  WA  0   0  4
  [ 4] .rel.data           REL             00000000 0001e8 000008 08   I 11   3  4
  [ 5] .bss                NOBITS          00000000 000070 000000 00  WA  0   0  1
  [ 6] .comment            PROGBITS        00000000 000070 000035 01  MS  0   0  1
  [ 7] .note.GNU-stack     PROGBITS        00000000 0000a5 000000 00      0   0  1
  [ 8] .eh_frame           PROGBITS        00000000 0000a8 000038 00   A  0   0  4
  [ 9] .rel.eh_frame       REL             00000000 0001f0 000008 08   I 11   8  4
  [10] .shstrtab           STRTAB          00000000 0001f8 00005b 00      0   0  1
  [11] .symtab             SYMTAB          00000000 0000e0 0000c0 10     12   8  4
  [12] .strtab             STRTAB          00000000 0001a0 000017 00      0   0  1
Clave para Opciones:
  W (escribir), A (asignar), X (ejecutar), M (mezclar), S (cadenas)
  I (info), L (orden enlazado), G (grupo), T (TLS), E (excluir)
  x (desconocido), O (se requiere procesamiento extra del SO)
  o (específico del SO), p (específico del procesador)
```

readelf -S p (1)

```
Hay 31 encabezados de sección, comenzando en el desplazamiento: 0x1824:

Encabezados de Sección:
  [Nr] Nombre              Tipo            Direc    Desp   Tam    ES Opt En Inf Al
  [ 0]                     NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp             PROGBITS        08048154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag       NOTE            08048168 000168 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i   NOTE            08048188 000188 000024 00   A  0   0  4
  [ 4] .gnu.hash           GNU_HASH        080481ac 0001ac 000020 04   A  5   0  4
  [ 5] .dynsym             DYNSYM          080481cc 0001cc 000040 10   A  6   1  4
  [ 6] .dynstr             STRTAB          0804820c 00020c 000045 00   A  0   0  1
  [ 7] .gnu.version        VERSYM          08048252 000252 000008 02   A  5   0  2
  [ 8] .gnu.version_r      VERNEED         0804825c 00025c 000020 00   A  6   1  4
  [ 9] .rel.dyn            REL             0804827c 00027c 000008 08   A  5   0  4
  [10] .rel.plt            REL             08048284 000284 000008 08  AI  5  24  4
  [11] .init               PROGBITS        0804828c 00028c 000023 00  AX  0   0  4
  [12] .plt                PROGBITS        080482b0 0002b0 000020 04  AX  0   0 16
  [13] .plt.got            PROGBITS        080482d0 0002d0 000008 00  AX  0   0  8
  [14] .text               PROGBITS        080482e0 0002e0 0001a2 00  AX  0   0 16
  [15] .fini               PROGBITS        08048484 000484 000014 00  AX  0   0  4
```

readelf -S p (2)

```
  [16] .rodata           PROGBITS        08048498 000498 000008 00   A  0   0  4
  [17] .eh_frame_hdr     PROGBITS        080484a0 0004a0 000034 00   A  0   0  4
  [18] .eh_frame         PROGBITS        080484d4 0004d4 0000e0 00   A  0   0  4
  [19] .init_array       INIT_ARRAY      08049f08 000f08 000004 00  WA  0   0  4
  [20] .fini_array       FINI_ARRAY      08049f0c 000f0c 000004 00  WA  0   0  4
  [21] .jcr              PROGBITS        08049f10 000f10 000004 00  WA  0   0  4
  [22] .dynamic          DYNAMIC         08049f14 000f14 0000e8 08  WA  6   0  4
  [23] .got              PROGBITS        08049ffc 000ffc 000004 04  WA  0   0  4
  [24] .got.plt          PROGBITS        0804a000 001000 000010 04  WA  0   0  4
  [25] .data             PROGBITS        0804a010 001010 000014 00  WA  0   0  4
  [26] .bss              NOBITS          0804a024 001024 000008 00  WA  0   0  4
  [27] .comment          PROGBITS        00000000 001024 000034 01  MS  0   0  1
  [28] .shstrtab         STRTAB          00000000 00171a 00010a 00      0   0  1
  [29] .symtab           SYMTAB          00000000 001058 000490 10     30  48  4
  [30] .strtab           STRTAB          00000000 0014e8 000232 00      0   0  1
Clave para Opciones:
  W (escribir), A (asignar), X (ejecutar), M (mezclar), S (cadenas)
  I (info), L (orden enlazado), G (grupo), T (TLS), E (excluir)
  x (desconocido), O (se requiere procesamiento extra del SO)
  o (específico del SO), p (específico del procesador)
```

readelf -l p (1)

```
El tipo del fichero elf es EXEC (Fichero ejecutable)
Punto de entrada 0x8048301
Hay 9 encabezados de programa, empezando en el desplazamiento 52

Encabezados de Programa:
  Tipo           Desplaz  DirVirt    DirFísica  TamFich TamMem  Opt Alin
  PHDR           0x000034 0x08048034 0x08048034 0x00120 0x00120 R E 0x4
  INTERP         0x000154 0x08048154 0x08048154 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x005b4 0x005b4 R E 0x1000
  LOAD           0x000f08 0x08049f08 0x08049f08 0x0011c 0x00124 RW  0x1000
  DYNAMIC        0x000f14 0x08049f14 0x08049f14 0x000e8 0x000e8 RW  0x4
  NOTE           0x000168 0x08048168 0x08048168 0x00044 0x00044 R   0x4
  GNU_EH_FRAME   0x0004a0 0x080484a0 0x080484a0 0x00034 0x00034 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x000f08 0x08049f08 0x08049f08 0x000f8 0x000f8 R   0x1
```

readelf -l p (2)

```
mapeo de Sección a Segmento:
 Segmento Secciones...
  00
  01     .interp
  02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym
         .dynstr .gnu.version .gnu.version_r .rel.dyn .rel.plt .init
         .plt .plt.got .text .fini .rodata .eh_frame_hdr .eh_frame
  03     .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
  04     .dynamic
  05     .note.ABI-tag .note.gnu.build-id
  06     .eh_frame_hdr
  07
  08     .init_array .fini_array .jcr .dynamic .got
```