ELF[101] a Linux executable walkthrough

Ange Albertini
corkami.com

Executable & Linkable Format

Dissected file

```
~$ uname -m
i686
~$ ./simple.elf
Hello World!
```

simple.elf

SHA-1: 6725968fb87b749b13fd59b0344a5890ebcadc3b
download @ elf101.corkami.com

Header 1/2
technical details for identification and execution

ELF header
identify as an ELF type
specify the architecture

Program Header table
execution information

Sections
contents of the executable

Code
executable information

Data
information used by the code

Sections' names

Header 2/2
technical details for linking
(ignored for execution)

Section Header table
linking (connecting program objects) information

Hexadecimal dump / ASCII dump

```
7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00   .ELF............
02 00 03 00 01 00 00 00 60 00 00 08 40 00 00 00   ........`...@...
C0 00 00 00 00 00 00 00 34 00 20 00 01 00 28 00   ........4.....(.
04 00 03 00                                       ....
```

Fields / Values / Explanation

1	e_ident		
	EI_MAG	0x7F, "ELF"	constant signature
	EI_CLASS, EI_DATA	1, 1	32 bits, Little-Endian
	EI_VERSION	1	Always 1
	e_type	2	Executable
	e_machine	3	Intel 386 (and later)
	e_version	1	Always 1
	e_entry	0x8000060	Address where execution starts
3	e_phoff	0x40	Program Headers' offset
	e_shoff	0xC0	Section Headers' offset
	e_ehsize	0x34	Elf header's size
	e_phentsize	0x20	Size of a single Program Header
	e_phnum	1	Count of Program Headers
	e_shentsize	0x28	Size of a single Section Header
	e_shnum	4	Count of Section Headers
	e_shstrndx	3*	Index of the names' section in the table

```
Offset:0x40/Address:0x8000040
01 00 00 00 00 00 00 00 60 00 00 08 00 00 00 08   ........`.......
A0 00 00 00 A0 00 00 00 05 00 00 00 00 10 00 00   ................
```

2	p_type	1	The segment should be loaded in memory
	p_offset	0	Offset where it should be read
	p_vaddr	0x8000000	Virtual address where it should be loaded
	p_paddr	0x8000000	Physical address where it should be loaded
	p_filesz	0xA0	Size on file
	p_memsz	0xA0	Size in memory
	p_flags	5	Readable and eXecutable

```
Offset:0x60/Address:0x8000060
B9 90 00 00 08 BA 0D 00 00 00 BB 01 00 00 00 B8   ................
04 00 00 00 CD 80 BB 01 00 00 00 B8 01 00 00 00   ................
CD 80                                             ..
```

3 — x86 assembly / Equivalent C code

```
mov ecx, 0x8000090         msg
mov edx, 0xD              msg len
mov ebx, 1                STDOUT_FILENO
mov eax, 4                SYS_WRITE
int 0x80                  system call    →write(STDOUT_FILENO, "Hello world!\n", len("Hello world!\n"));

mov ebx, 1                return code
mov eax, 1                SYS_EXIT
int 0x80                  system call    →exit(1);
```

```
Offset:0x90/Address:0x8000090
48 65 6C 6C 6F 20 57 6F 72 6C 64 21 0A 00         Hello.World!..
```

Strings
"Hello World!\n", 0

```
Offset:0xA0
00 2E 73 68 73 74 72 74 61 62 00 2E 74 65 78 74   ..shstrtab..text
00 2E 72 6F 64 61 74 61 00                         .rodata.
```

Section names
"" .shstrtab .text .rodata

```
Offset:0xC0
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0B 00 00 00 01 00 00 00   .......
06 00 00 00 60 00 00 08 60 00 00 00 22 00 00 00   ....`...`..."...
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
11 00 00 00 01 00 00 00 02 00 00 00 90 00 00 08   ................
90 00 00 00 0D 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 01 00 00 00 03 00 00 00   ................
00 00 00 00 00 00 00 00 A0 00 00 00 19 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

Section Header table

Index	Name	Type	Flags	Address	Offset	Size
0	<null>	0 SHT_NULL (inactive)				
1	.text	1 SHT_PROGBITS (program)	6 SHF_ALLOC (allocated) SHF_EXECINSTR (executable)	0x8000060	0x60	0x22
2	.rodata	1 SHT_PROGBITS (program)	2 SHF_ALLOC (allocated)	0x8000090	0x90	0x0D
3*	.shstrtab	3 SHT_STRTAB (string table)			0xA0	0x19

relative offsets in names' section

This is the whole file, however, most ELF files contain many more elements.
Explanations are simplified, for conciseness.

Loading process

1 Header
the ELF header is parsed
the Program Header is parsed
(Sections are not used)

2 Mapping
the file is mapped in memory
according to its segment(s)

LOAD Segment

Offset / Virtual Address
p_offset 0x00   0x8000000 p_vaddr
p_filesz   p_memsz
          0xA0   0x80000A0

3 Execution
Entry is called
Syscalls *kernel services* are accessed via:
 - Syscall number in the EAX register
 - calling Interrupt 0x80

Trivia

The ELF was first specified by U.S.L. and U.I. for UNIX System V, in 1989

The ELF is used, among others, in:
 - Linux, Android, *BSD, Solaris, BeOS
 - PSP, Playstation 2-4, Dreamcast, GameCube, Wii
 - various OSes made by Samsung, Ericsson, Nokia,
 - microcontrollers from Atmel, Texas Instruments