ELF1 2B Program Headers

Young W. Lim

2022-01-10 Mon

Outline

1. Based on

2. Program Headers
   - Program Headers
   - Segment contents
   - Base address

Based on

"Study of ELF loading and relocs", 1999
http://netwinder.osuosl.org/users/p/patb/public_html/elf_relocs.html

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`

(1) ELF header and program headers

- the ELF file has an header that describes
  the overall layout of the file.
- the ELF header actually points to
  another group of headers called the program headers
  - these headers describe to the operating system anything
    that might be required for it
    to load the binary into memory and execute it.
  - segments are described by program headers,
    but so are some other things required to get the executable running.

https://www.bottomupcs.com/elf.xhtml

(2) ELF32_Ehdr

ELF File Header

```
typedef struct {
        unsigned char e_ident[EI_NIDENT];
        Elf32_Half    e_type;
        Elf32_Half    e_machine;
        Elf32_Word    e_version;
        Elf32_Addr    e_entry;
        Elf32_Off     e_phoff;  ........ for program header
        Elf32_Off     e_shoff;
        Elf32_Word    e_flags;
        Elf32_Half    e_ehsize;
        Elf32_Half    e_phentsize;  .... for program header
        Elf32_Half    e_phnum;  ........ for program header
        Elf32_Half    e_shentsize;
        Elf32_Half    e_shnum;
        Elf32_Half    e_shstrndx;
} Elf32_Ehdr;
```

https://www.bottomupcs.com/elf.xhtml

(3) e_phoff, e_phentsize, e_phnum

- in the ELF (File) header definition

e_phoff	the offset in the file where the program header table starts
e_phentsize	the size of an entry of in the program header table
e_phnum	the number of entries in the program header table

- with these three fields, the file's program headers can be located and accessed

https://www.bottomupcs.com/elf.xhtml

(4) Elf32_Phdr

Program Header

```
typedef struct {
        Elf32_Word    p_type;
        Elf32_Off     p_offset;
        Elf32_Addr    p_vaddr;
        Elf32_Addr    p_paddr;
        Elf32_Word    p_filesz;
        Elf32_Word    p_memsz;
        Elf32_Word    p_flags;
        Elf32_Word    p_align;
} ELF32_Phdr;
```

https://www.bottomupcs.com/elf.xhtml

(5) Program header

- A file's program header table is an array of structures
  - each entry describing
    - a segment or
    - other information the system needs
      to prepare the program for execution.
  - an object file segment contains one or more sections
    though this fact is transparent to the program header
- program headers are meaningful
  only for executable and shared object files.

https://man7.org/linux/man-pages/man5/elf.5.html

(6) Program header table

- the program header table
  - starts at e_phoff in the file
  - the table's total size : e_phentsize * e_phnum
  - each entry has the same size : e_phentsize (in bytes)
  - the number of entries : e_phnum

https://man7.org/linux/man-pages/man5/elf.5.html

(7) p_type

- Program headers more than just segments.

p_type	shows what the program header entry is defining
PT_INTERP	this *information* entry defines a string pointer to an interpreter for the binary file.
PT_LOAD	this *segment* entry specifies a loadable segment described by p_filesz and p_memsz

https://www.bottomupcs.com/elf.xhtml

(8) PT_INTERP

p_type	PT_INTERP	defines a string pointer to an interpreter

- this array element (program header table entry) specifies the *location* and *size* of a null-terminated path name to invoke as an interpreter
- *information* entry
- meaningful only for executable files (though it may occur for shared objects);
    - it may not occur more than once in a file.
    - if it is present, it must precede any loadable segment entry.

https://refspecs.linuxbase.org/elf/gabi4+/ch5.pheader.html#segment_contents

(9) PT_INTERP example

Program Header

```
Type           Offset             VirtAddr           PhysAddr
               FileSiz            MemSiz             Flags  Align
PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
               0x00000000000001f8 0x00000000000001f8 R E    8
INTERP         0x0000000000000238 0x0000000000400238 0x0000000000400238
               0x000000000000001c 0x000000000000001c R      1
[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
```

- *location* : `p_offset`, `p_vaddr`, `p_paddr`
- *size* : `p_filesz`, `p_memsz`

https://refspecs.linuxbase.org/elf/gabi4+/ch5.pheader.html#segment_contents

(10) Interpreter

- some changes might need to be made for the binary to execute properly at runtime.
- the usual interpreter of a binary file is the dynamic loader
- it is called because it takes the final steps to finish loading of the executable and to prepare the binary image for running.

https://www.bottomupcs.com/elf.xhtml

(11) PT_LOAD

p_type	PT_PT_LOAD	specifies a loadable segment . . . .

- the size of a loadable segment is described
  by p_filesz (file size)
  and p_memsz (memory size)
- the bytes from the file are mapped
  to the beginning of the memory segment.
- loadable segment entries in the program header table
  appear in ascending order, sorted on the p_vaddr member.

https://refspecs.linuxbase.org/elf/gabi4+/ch5.pheader.html#segment_contents

(12) `p_memsz`, `p_filesz`

- `p_memsz > p_filesz`
  the *extra bytes* are defined to hold the value 0 and to follow the segment's initialized area
- `p_memsz < p_filesz` : not possible case
  the memory size cannot be smaller than the file size

`https://docs.oracle.com/cd/E19683-01/816-1386/chapter6-83432/index.html`

(13) p_offset, p_vaddr, p_paddr

p_offset	shows the location *where* the segment starts in the file on disk
p_vaddr	shows what address that the segment resides in the *virtual memory*
p_paddr	shows the *physical address*, which is only useful for small embedded systems *without virtual memory*

https://www.bottomupcs.com/elf.xhtml

(14) p_flags

`p_flags`	shows the *premissions* on the segment excecute, read, and write permissions can be specified in any combination

- the system gives access permissions to the segment, through the `p_flags` member.
- *at least one* loadable segment (not mandated)

https://www.bottomupcs.com/elf.xhtml

(15) p_flags

p_flags	Flags relevant to the segment	
PF_X	eXecute	0x1
PF_W	Write	0x2
PF_R	Read	0x4
PF_MASKPROC	Unspecified	0xf0000000

- code segments should be marked as read and execute only,
- data sections as read and write with no execute.
- PF_MASKPROC mask are reserved for processor-specific semantics.

https://www.bottomupcs.com/elf.xhtml

(17) p_align

p_align	gives the value to which the *segments* are aligned in *memory* and in the *file*

- p_align = 0 or 1 mean no alignment is required.
- p_align should be a positive, integral power of 2
- loadable process segments must have congruent values for p_vaddr and p_offset, modulo the page size

https://www.bottomupcs.com/elf.xhtml

Segment contents (1)

- An object file segment consists of one or more sections though this fact is *transparent* to the program header
- Whether the file segment holds one or many sections also is *immaterial* to *program loading*
- Nonetheless, *various data* must be present for *program execution*, *dynamic linking*, and so on.

`https://docs.oracle.com/cd/E19683-01/816-1386/chapter6-83432/index.html`

Segment contents (2)

- text segments contain read-only instructions and data
- data segments contain writable data and instructions

https://docs.oracle.com/cd/E19683-01/816-1386/chapter6-83432/index.html

Segment contents (3)

- A PT_DYNAMIC program header element points at the .dynamic section
- The .got and .plt sections also hold information related to position-independent code and dynamic linking
- The .plt section can reside in a text or a data segment, depending on the processor.

https://docs.oracle.com/cd/E19683-01/816-1386/chapter6-83432/index.html

Segment contents (4)

- The .bss section has the type (sh_type = SHT_NOBITS)
  - Although it occupies no space in the file, it contributes to the segment's memory image
  - Normally, these uninitialized data reside at the end of the segment,
  - thereby making p_memsz larger than p_filesz in the associated program header element.

https://docs.oracle.com/cd/E19683-01/816-1386/chapter6-83432/index.html

Base address (1)

- executable and shared object files
  have a base address :
  - the lowest virtual address associated
    with the memory image of the program's object file.
- to relocate the memory image of the program
  during dynamic linking

https://docs.oracle.com/cd/E19683-01/816-1386/chapter6-83432/index.html

Base address (2)

- an executable or shared object file's
  base address is calculated during execution
  from three values:
  - the memory load address
  - the maximum page size
  - the lowest virtual address of a program's loadable segment

  - the virtual addresses in the program headers
    might not represent the actual virtual addresses
    of the program's memory image

https://docs.oracle.com/cd/E19683-01/816-1386/chapter6-83432/index.html

Base address (3)

- to compute the base address
  of an executable or shared object file
  you determine the memory addreses associated with
  the lowest `p_vaddr` value
  for a `PT_LOAD` segment.
- then obtain the base address
  by *truncating* the memory address
  to the nearest multiple of the maximum page size.
- depending on the kind of file being loaded into memory,
  the memory address might not match the `p_vaddr` values.

https://docs.oracle.com/cd/E19683-01/816-1386/chapter6-83432/index.html