Pliki wykonywalne ELF

Programy wykonywalne oraz biblioteki współdzielone w systemach Linux zapisywane są w plikach, których struktura zdefiniowana jest przez standard ELF (*Executable and Linkable Format*)

Nie tylko w rodzinie systemów Linux

- rodzina systemów BSD
- alternatywne systemy - AmigaOS 4, MorphOS, AROS, Plan9
- systemy konsoli gier (Sony Playstation 2/3/4, Nintendo Wii
- urządzenia podręczne - tablety, smartfony z systemami Android

Analiza struktury pliku ELF

Narzędzia

- readelf
- objdump

Wchodzą w skład pakietu *binutils* https://www.gnu.org/software/binutils/

Pełne definicje formatu ELF znajdują się w pliku `/usr/include/elf.h`

Struktury

- **ELF Header** - początkowa struktura opisująca podstawowe własności ELF-a, takie jak wymagana architektura procesora.
- **Program Headers** - tablica struktur opisująca fragmenty pliku, ich przeznaczenie, lokalizacje w pliku i w pamięci, oraz uprawnienia, z którymi powinny zostać załadowane do pamięci.
- **Section Headers** - tablica struktur podobnych do Program Headers.
- **Dynamic Headers** - tablica przechowująca informacje niezbędne do poprawnego ładowania bibliotek współdzielonych.
- **Symbols Header** - opis symboli używanych w pliku

Zrzut hex nagłówka ELF

```
# xxd -l 52 sqlite3
00000000: 7f45 4c46 0101 0100 0000 0000 0000 0000  .ELF............

00000010: 0300 2800 0100 0000 203f 0000 3400 0000  ..(..... ?..4...

00000020: 34ac 1300 0002 0005 3400 2000 0800 2800  4.......4. ...(.

00000030: 2600 2500                                &.%.
```

Adresy

Adres 0x00 zawiera 4 bajty, które zawsze przyjmują tę samą postać: 0x7F, 0x45, 0x4C oraz 0x46. W przypadku, gdy plik pod tym adresem zawiera inną treść, najprawdopodobniej nie jest to plik ELF

- **`0x04`** zawiera jeden bajt identyfikujący platformę 1 - 32 bity, 2 - 64 bity
- **`0x05`** kolejność bajtów na danej platformie (big-endian wartość 1 /little-endian wartość 2)
- **`0x07`** użyty interfejs binarny (ABI), nowe wersje specyfikacji ELF wycofują użycie tego pola i zaleca się jego wyzerowanie
- **`0x08`** historycznie zawierał informacje o wersji ABI zdefiniowanego w poprzednim polu

Adresy

- `0x10` zawiera dwa bajty określające typ pliku wykonywalnego
- `0x12` zawiera dwa bajty informujące o architekturze docelowej, może przyjmować wiele wartości, najpopularniejsze: “2” dla SPARC, “3” dla architektury 32 bitowej, “8” dla architektury MIPS, “40” dla architektury ARM, “64” dla architektury 64 bitowej

Przeczytanie nagłówka *file header* (struktura `ELF32_Ehdr`)

```
root@lukassz-host:/home/lukassz/sqliteARM/sqlite-autoconf-3160200# readelf -h sqlite3
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x3fd8
  Start of program headers:          52 (bytes into file)
  Start of section headers:          1303252 (bytes into file)
  Flags:                             0x5000200, Version5 EABI, soft-float ABI
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         8
  Size of section headers:           40 (bytes)
  Number of section headers:         38
  Section header string table index: 37
```

Sekcja .dymanic

Każdy program, który nie jest zbudowany statycznie, zawiera sekcję o nazwie .dynamic. W przypadku jej istnienia program ładujący musi przejść dodatkowy proces inicjalizacji polegający na załadowaniu każdej biblioteki zależnej oraz - rekurencyjnie - każdej zależności bibliotek zależnych. Jeśli ładowanie jakiejś zależności nie powiedzie się, proces ładowania całego programu kończy się błędem.

```
# readelf -d sqlite3
Dynamic section at offset 0x10dd00 contains 28 entries:
  Tag        Type                         Name/Value
 0x00000003 (PLTGOT)                     0x10ee64
 0x00000002 (PLTRELSZ)                   800 (bytes)
 0x00000017 (JMPREL)                     0x37e8
 0x00000014 (PLTREL)                     REL
 0x00000011 (REL)                        0x1020
 0x00000012 (RELSZ)                      10184 (bytes)
 0x00000013 (RELENT)                     8 (bytes)
 0x6ffffffa (RELCOUNT)                   1247
 0x00000015 (DEBUG)                      0x0
 0x00000006 (SYMTAB)                     0x148
 0x0000000b (SYMENT)                     16 (bytes)
 0x00000005 (STRTAB)                     0x7f8
 0x0000000a (STRSZ)                      983 (bytes)
 0x00000004 (HASH)                       0xbd0
 0x00000001 (NEEDED)                     Shared library: [libc.so]
 0x00000001 (NEEDED)                     Shared library: [libdl.so]
 0x0000001a (FINI_ARRAY)                 0x10e980
 0x0000001c (FINI_ARRAYSZ)               8 (bytes)
 0x00000019 (INIT_ARRAY)                 0x10e988
 0x0000001b (INIT_ARRAYSZ)               16 (bytes)
 0x00000020 (PREINIT_ARRAY)              0x10e998
 0x00000021 (PREINIT_ARRAYSZ)            0x8
 0x0000001e (FLAGS)                      BIND_NOW
 0x6ffffffb (FLAGS_1)                    Flags: NOW
 0x6ffffff0 (VERSYM)                     0xf08
 0x6ffffffe (VERNEED)                    0xfe0
 0x6fffffff (VERNEEDNUM)                 2
 0x00000000 (NULL)                       0x0
```

Biblioteki zależne jakich wymaga program

ELF - dynamiczne biblioteki

```
root@lukassz-host:/home/lukassz/sqliteARM/sqlite-autoconf-3160200# readelf -l sqlite3

Elf file type is DYN (Shared object file)
Entry point 0x3fd8
There are 8 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x00100 0x00100 R   0x4
  INTERP         0x000134 0x00000134 0x00000134 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /system/bin/linker]
  LOAD           0x000000 0x00000000 0x00000000 0x10bdb0 0x10bdb0 R E 0x1000
  LOAD           0x10cb30 0x0010db30 0x0010db30 0x0348c 0x038f0 RW  0x1000
  DYNAMIC        0x10dd00 0x0010ed00 0x0010ed00 0x00108 0x00108 RW  0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0
  EXIDX          0x10bc28 0x0010bc28 0x0010bc28 0x00188 0x00188 R   0x4
  GNU_RELRO      0x10cb30 0x0010db30 0x0010db30 0x014d0 0x014d0 RW  0x8
```

```
#!/bin/sh

export ANDROID_NDK=/home/lukassz/android-ndk
export PREFIX=/opt

if [ -z ${ANDROID_NDK} ]; then
        echo "Please set ANDROID_NDK environment variable to the root directory of the Android NDK"
        read ndk
        export ANDROID_NDK=$ndk
        exit 1
fi

if [ -z ${PREFIX} ]; then
        echo "Please set PREFIX environment variable to the output directory"
        exit 1
fi

export COMPILE_TARGET=arm-linux-androideabi
export ANDROID_API=16

export ANDROID_PREFIX=${ANDROID_NDK}/toolchains/${COMPILE_TARGET}-4.9/prebuilt/linux-x86_64
export SYSROOT=${ANDROID_NDK}/platforms/android-${ANDROID_API}/arch-arm

export TOOLCHAIN_PATH=${ANDROID_PREFIX}/bin

export CPP=${TOOLCHAIN_PATH}/${COMPILE_TARGET}-cpp
export CC=${TOOLCHAIN_PATH}/${COMPILE_TARGET}-gcc
export LD=${TOOLCHAIN_PATH}/${COMPILE_TARGET}-ld
export PKG_CONFIG_PATH=${PREFIX}/lib/pkgconfig

export CFLAGS="${CFLAGS} --sysroot=${SYSROOT} -FPIE -pie -I${SYSROOT}/usr/include -I${ANDROID_PREFIX}/include"
export CPPFLAGS="${CFLAGS}"
export LDFLAGS="${LDFLAGS} -L${SYSROOT}/usr/lib -L${ANDROID_PREFIX}/lib"

./configure --host=${COMPILE_TARGET} --prefix=${PREFIX}
```

Bibliografia

1. Praktyczna inżynieria wsteczna: Metody, techniki i narzędzia
2. Zrozumieć programowanie
3. Wikipedia
4. Manual