Biblioteki wspólne w systemach UNIX, ładowanie dynamiczne bibliotek

Marcin Owsiany `marcin@owsiany.pl`

2003-11-17

Zakres referatu

- linkery i loadery — wprowadzenie
- format ELF
- dynamic loader
- `dlopen()` & friends
- tworzenie i zarządzanie bibliotekami

Wprowadzenie

- linkery i loadery — rys historyczny
- mechanizm COW (copy-on-write)
- kod PIC (Position-Independent Code) i non-PIC
- modele współdzielenia kodu:
  - prosty model bez systemu operacyjnego
  - model ze współdzieleniem przestrzeni adresowej
  - modele z pamięcią wirutalną: prosty (A.OUT) i złożony

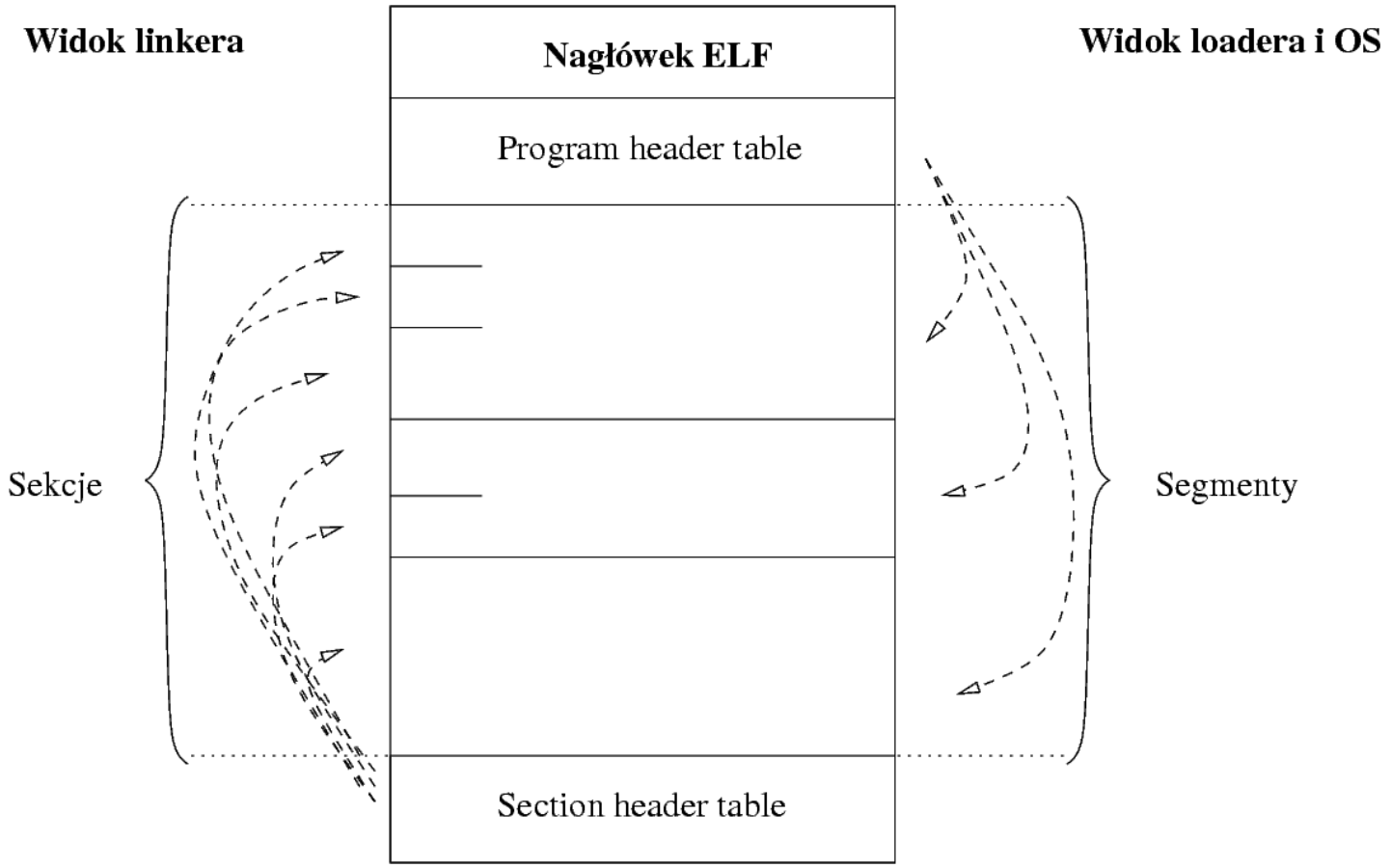
Format ELF

- cechy formatu
- nagłówek ELF

```
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:                           Intel 80386
Version:                           0x1
Entry point address:               0x30b0
Start of program headers:          52 (bytes into file)
Start of section headers:          80096 (bytes into file)
Flags:                             0x0
Size of this header:               52 (bytes)
Size of program headers:           32 (bytes)
Number of program headers:         3
Size of section headers:           40 (bytes)
Number of section headers:         22
Section header string table index: 21
```

Struktrura pliku ELF

ELFowa ,,schizofrenia'': — dwa widoki na zawartość pliku



Typy plików ELF

Relocatable (relokowalne). Tworzone przez kompilatory i asemblery. Przed dalszym wykorzystaniem muszą zostać przetworzone przez linker (konsolidator), ponieważ zawarty w nich kod nie zawiera poprawnych adresów.

Executable (wykonywalne). Można je załadować do pamięci i wykonać. Funkcję tą wykonuje program zwany loaderem.

Shared (wspólne). Ten typ obejmuje dynamiczne biblioteki wspólne.

Core. Oznacza, że plik zawiera zrzut pamięci programu. ELF nie definiuje dla tego typu żadnych struktur poza samym nagłówkiem.

Zależności między typem obiektu a częściami pliku:

Typ obiektu	Sekcje	Segmenty
Relocatable	tak	nie
Executable	nie	tak
Shared	tak	tak

Zawartość

- Sekcje
  - tablica napisów (ang. *string table*) — *de facto* skonkatenowane napisy w formacie null-terminated,
  - tablica symboli (importowanych i eksportowanych przez dany obiekt). Używa ona offsetów tablicy napisów do określenia nazw symboli.

    Informacje te czyta na przykład loader dynamiczny w celu określenia potrzebnych obiektów.
- Segmenty — zbiory sekcji o wspólych cechach (wykonywalność, RO/RW, rodzaj danych)

  Istotny jest segment typu `PHT_DYNAMIC`:
  - wpis typu `DT_SONAME`. Nazwa ta służy do rozróżnienia między bibliotekami.
  - zero lub więcej wpisów typu `DT_NEEDED`. SONAME *wymaganej* do działania danego obiektu biblioteki dynamicznej.

    Kolejność występowania wpisów typu `DT_NEEDED` jest istotna, gdyż decyduje o zakresie widoczności symboli.

Przykładowa sekcja `.dynamic`

```
Dynamic segment at offset 0x13500 contains 24 entries:
  Tag    Type                 Name/Value
 0x00000001 (NEEDED)      Shared library: [libssl.so.0.9.7]
 0x00000001 (NEEDED)      Shared library: [libcrypto.so.0.9.7]
 0x00000001 (NEEDED)      Shared library: [libc.so.6]
 0x00000001 (NEEDED)      Shared library: [libpthread.so.0]
 0x0000000e (SONAME)      Library soname: [libgadu.so.2]
 0x0000000c (INIT)        0x2908
 0x0000000d (FINI)        0xdf50
 0x00000004 (HASH)        0x94
 0x00000005 (STRTAB)      0x1670
 0x00000006 (SYMTAB)      0x770
 [...]
 0x00000000 (NULL)        0x0
```

Ładowanie procesu

- rola jądra
- start interpretera programu: `/lib/ld-linux.so.2` lub `/usr/lib/ld.so.1`
- zadania loadera dynamicznego:
  - ewentualne załadowanie właściwego programu,
  - określenie wymaganych obiektów i załadowanie ich,
  - zrelokowanie programu i bibliotek,
  - zamknięcie deskryptora pliku przekazanego przez jądro, o ile w ten sposób właściwy program został przekazany interpreterowi,
  - zainicjalizowanie programu i bibliotek w odpowiedniej kolejności,
  - oddanie kontroli programowi w taki sposób, jakby został wywołany bezpośrednio przez `exec()`.

Ładowanie bibliotek

- preloading (`LD_PRELOAD` i `/etc/ld.so.preload`)
- czytanie wpisów `DT_NEEDED` i `DT_RPATH`, z segmentu `PHT_DYNAMIC`

  Nazwy wymienione we wpisach `DT_NEEDED` są albo równe tym wymienionym w `DT_SONAME` bibliotek, albo (jeśli nazwa zawiera conajmniej jeden znak ,,/'') — wprost ścieżkami do tych obiektów.
- Mechanizmy poszukiwania bibliotek. Jeśli nazwa nie jest ścieżką, to plik, który ma zostać załadowany określają (w systemach GNU/Linux) kolejno następujące mechanizmy:
  - ścieżka podana przez element `DT_RPATH`,
  - ścieżka podana w zmiennej środowiskowej `LD_LIBRARY_PATH`,
  - skompilowane informacje z pliku `/etc/ld.so.cache`. (Generowany przez `ldconfig`, z `/etc/ld.so.conf`.)
  - domyślny katalog `/usr/lib`
  - domyślny katalog `/lib`
- podmiana symboli — może być też zagrożeniem

Dynamic loading w SunOS

Dodatkowe istotne mechanizmy w linkerze dynamicznym systemu Solaris:

- rozwijanie zmiennych: `/usr/platform/$PLATFORM/lib/libc_psr.so.1` $\Rightarrow$ `/usr/platform/SUNW,Ultra-80/lib/libc_psr.so.1`
- alternatywne pliki konfiguracji linkera — `LD_CONFIG`
- zmienna środowiskowa `LD_LOADFLTR` umożliwia ,,podstawienie'' alternatywnej biblioteki w miejsce innej.
- zmienna `LD_NOLAZYLOAD` powoduje ignorowanie flagi ,,lazy loading'' ustawionej przez link editor, i załadowanie danej zależności tak szybko jak to możliwe.
- zmienna `LD_PROFILE` umożliwia profiling wybranej biblioteki.
- `/usr/lib/0@0.so.1` — obsługa archaicznych programów

Relokacja

- złożoność conajmniej rzędu $O(rs)$, gdzie r — liczba relokacji, a s — liczba symboli zdefiniowanych we wszystkich powiązanych obiektach.
- relokacja jest skomplikowana — to, jaki adres zostanie przyporządkowany danemu symbolowi zależy od zakresu (ang. *lookup scope*), który z kolei zależy od kolejności ładowania bibliotek
- stosowanie tablicy haszującej, w obrębie poszczególnych kubełków symbole są wyszukiwane przy pomocy pełnego porównania nazw. Stąd słaba wydajność linkowania C++.

`dlopen()` and friends

Opisane wcześniej mechanizmy są realizowane w przezroczysty sposób przez linker dynamiczny. Istnieje jednak bardziej elastyczny mechanizm: funkcje `dlopen()`, `dlsym()` i `dlclose()`.

Przykładowe użycie tych funkcji:

```
void    *handle;
int     *iptr, (*fptr)(int);

handle = dlopen("/foo/bar/libfoo.so", RTLD_LOCAL | RTLD_LAZY);

*(void **)(&fptr) = dlsym(handle, "my_function");
iptr = (int *)dlsym(handle, "my_object");

(*fptr)(*iptr);
```

`dlopen()` — resolving symboli

Można użyć wartości specjalnych pierwszego argumentu `dlsym()`:

`RTLD_DEFAULT` — szukanie symbolu wykonywane w zakresie globalnym — wynikowy obiekt ma takie samo znaczenie jakby został użyty bezpośrednio,

`RTLD_NEXT` — określa obiekt następny po tym, który jest zdefiniowany przez daną nazwę. Takie zachowanie umożliwia tworzenie hierarchii obiektów, które oddelegowują działania do swych poprzedników.

Przykład: biblioteka ,,sleepless'' (`LD_PRELOAD=sleepless.so sleep 10`):

```
#define __USE_GNU
#include <dlfcn.h>
unsigned int sleep(unsigned int seconds)
{
    unsigned int (*origsleep)(unsigned int) = dlsym(RTLD_NEXT, "sleep");
    return origsleep(seconds / 2);
}
```

Pisanie bibliotek

- pisanie kodu — w zasadzie brak różnic od zwykłego programu
- kompilacja

```
cc -fPIC -o plik1.o plik1.c
[...]
cc -fPIC -o plikm.o plikm.c
```

- linkowanie

```
cc -shared -o libfoo.so.N -Wl,-z,defs -Wl,-soname,libfoo.so.N \
  plik1.o ... plikm.o -llib1 ... -libx
```

- uwaga na kolejność plików relokowalnych i bibliotek

Przydatne narzędzia

`readelf` — wypisuje ,,niskopoziomowe'' informacje na temat zawartości pliku obiektowego

`file` — podaje m.in. informację czy plik zawiera tablice symboli i debugowania

`strip` — ,,obdziera'' plik z niepotrzebnych tablic symboli, dzięki czemu zmniejsza jego wielkosć

`ldd` — pokazuje jakie biblioteki wymagane są przez dany obiekt i które zostaną załadowane

`strings` — wypisuje napisy zawarte w pliku obiektowym

`nm` — wypisuje symbole zawarte w pliku obiektowym

`strace, ltrace` — śledzą wykonanie programu

`libtool` — nakładka na toolchain pozwalająca programiście uzyskać jednolity obraz bibliotek bez względu na platformę (symulacja bibliotek dynamicznych, wywoływanie odpowiednich komend)

Zgodność binarna bibliotek

Dany program X, zlinkowany (przy tworzeniu) z biblioteką Y może być linkowany dynamicznie z dowolną wersją tej biblioteki, pod warunkiem, że jest ona binarnie zgodna z wersją, z którą X był linkowany przy jego tworzeniu.

Za interfejsy biblioteki uznajemy:

- funkcje (i ich prototypy) — zarówno nazwy jak i typy argumentów i zwracanych wartości,
- niestatyczne zmienne globalne — nazwy i typy,
- definicje struktur, unii i typów

Biblioteki *tracą* binarną zgodność przy:

- usuwaniu interfejsów,
- zmianie typów interfejsów,
- istotnej niezgodnej zmianie semantyki działania.