Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

AGH University of Science
and Technology

Binutils, biblioteki dynamiczne i statyczne

Gabriel Górski

Wydział Fizyki i Informatyki Stosowanej

10 IV 2018

Część I

Pliki obiektowe

Zarys

Wstęp

Kod maszynowy który procesor *umie ewaluować* jest fundamentalną acz nie jedyną częścią pliku obiektowego.
Plik obiektowy — aby był użyteczny — musi zawierać w sobie wszelkie informacje konieczne do tego by dało się go integrować z innymi plikami obiektowymi i ostatecznie doprowadzić do formy wykonywalnej.

Formaty plików

Konieczna jest jakaś fizyczna, udokumentowana manifestacja pliku obiektowego tj. **format**
Przykłady formatów plików obiektowych

ELF

Executable and Linkable Format (więcej przy okazji analizy)

PE

Portable Executable

Mach-O

Mach object

Rodzaje plików obiektowych

Rodzaje plików obiektowych w ramach formatu ELF:

1. Relocatable object
2. Executable object
3. Shared object
4. Core object

Relocatable object

Jest to bezpośredni efekt pracy kompilatora który musi zostać poddany dalszemu linkowaniu. Zawiera *luki* które muszą zostać poddane procesowi relokacji w czasie linkowania (stąd nazwa). Może powstać w wyniku łączenia przez linker kilku innych plików tego typu.

Przykład

plik.c $\longrightarrow$ kompilator $\longrightarrow$ plik.o

Executable object

Efekt procesu linkowania. Zawiera wszystkie konieczne informacje do wykonania programu. Wszystkie adresy są już odpowiednio zmapowane lub jest zawarta informacja o konieczności dostarczenia odpowiednich symboli poprzez spełnienie zależności wobec odpowiednich współdzielonych bibliotek **Przykład**

plik1.o, plik2.o, plik3.o ... + [-lbiblioteka1 -lbiblioteka2 ...] $\longrightarrow$ linker $\longrightarrow$ plik.out

Shared object, core object

- **Shared object**
  Efekt linkowania przy opcji *–shared*. Typ pliku wykorzystywany jako biblioteka współdzielona lub współcześnie bardzo często jak plik wykonywalny. Więcej przy porównaniu typów bibliotek
- **Core object**
  Plik obiektowy będący zrzutem obrazu procesu w momencie jego śmierci w wyniku błędu.

Część II

Biblioteki

Zarys

Typy bibliotek

Wyróżniamy dwa typy bibliotek

- Biblioteki statyczne
- Biblioteki współdzielone

Biblioteki statyczne I

Biblioteka statyczna jest archiwum plików obiektowych które możemy zlinkować z naszymi plikami źródłowymi. Tak powstały plik wykonywalny będzie już potem w pełni niezależny. **Przykład**
Działanie:

- Proces linkowania zaczyna się od podanych plików obiektowych. Powstaje w ten sposób lista brakujących symboli będących zależnościami
- Kolejno podane biblioteki statyczne są rozpatrywane od lewej do prawej
- Linkowane są jedynie te pliki obiektowe (z bibliotek) które dostarczają brakujących symboli ze wspomnianej listy

Biblioteki statyczne II

- Ma to swoje konsekwencje: jeśli w liście podanych linkerowi argumentów biblioteka B znajduje się za biblioteką A, a biblioteka B korzysta z symboli z **niedołączonego** pliku obiektowego z biblioteki A to nastąpi błąd linkowania

Wnioski:

- Kolejność bibliotek w liście argumentów ma znaczenie
- Może wystąpić cykliczna zależność

Tabela pomocnicza do przykładu

		Biblioteka A			Biblioteka B
Pliki obiektowe	staticlibraryusage.o	add2num.o	dostuff.o	uselessfunction.o	anotherfunction.o
Dostarczane referencje	main	add2num, what	dostuff	uselessfunction	anotherfunction
Brakujące referencje	anotherfunction, add2num	anotherfunction			dostuff
Lista brakujących referencji L → P	anotherfunction, add2num	anotherfunction	anotherfunction	anotherfunction	dostuff
		Biblioteka B	Biblioteka A		
Pliki obiektowe	staticlibraryusage.o	anotherfunction.o	add2num.o	dostuff.o	uselessfunction.o
Dostarczane referencje	main	anotherfunction	add2num, what	dostuff	uselessfunction
Brakujące referencje	anotherfunction, add2num	dostuff	anotherfunction		
Lista brakujących referencji L → P	anotherfunction, add2num	add2num, dostuff	dostuff		

Biblioteki współdzielone

W odróżnieniu od bibliotek statycznych będących jedynie archiwami plików obiektowych, **biblioteki współdzielone** wprowadzają zupełnie nowy mechanizm.
Linkowanie względem obiektu współdzielonego (shared object) powoduje zawarcie w wynikowym pliku wykonywalnym **informacji o odpowiedniej zależności/zależnościach**, która ma być rozwiązana **w momencie uruchomienia programu**.
Zadaniem tym para się **linker dynamiczny** który sam w sobie jest obiektem typu współdzielonego. W momencie oddania kontroli nad programem do systemu operacyjnego przestrzeń adresowa programu jest gotowa do użycia i pozbawiona luk. **Przykład**

Zalety i wady

Biblioteki statyczne:

+ Archiwum biblioteki konieczne jest jedynie na etapie linkowania (statycznego)
+ Dystrybucja oprogramowania jest wygodniejsza — Wszystkie brakujące zależności dostarczane bezpośrednio
– Ostateczny plik wykonywalny waży więcej
– Potencjalne problemy licencyjne

Biblioteki współdzielone

+ Współdzielony blok z kodem który jest read-only między programami korzystającymi z tej samej biblioteki (PIC)
+ Łatwa propagacja aktualizacji kodu biblioteki
+ Możliwość podmieniania kodu w czasie działania programu korzystając z funkcjonalności udostępnianej przez linker dynamiczny
– Plusy biblioteki statycznej (generalnie)

Wstęp

Przed użyciem pozycja biblioteki współdzielonej nie jest znana, a jej kod potrzebuje odpowiednich adresów do funkcji i danych. Gdy linker dynamiczny wczytuje z dysku bibliotekę problem może zostać rozwiązany na dwa sposoby

- *load-time relocation*
- *PIC - Position Independent Code*

Load-time relocation

Jest to relokacja wykonywana (jak nazwa wskazuje) w czasie ładowania programu do pamięci. Adresy w sekcji kodu biblioteki zostają wtedy zamienione na właściwe adresy wykorzystując dane z wpisów w tabeli relokacji.

Ta technika jest zupełnie wyłączona z użytku (pomijając hacki z wykorzystaniem flag) na architekturze 64 bitowej.

Zalety: (x86)

- Mniejszy *overhead* operacji w porównaniu do PICa

Wady:

- *Współdzielone* biblioteki — *.text* musi być RW, więc konieczna jest kopia całej biblioteki
- Potencjalnie mniej bezpieczne przez zupełny brak randomizacji adresowania

PIC

Technika polegająca na tym, że znany jest offset między daną instrukcją, a tablicą GOT z sekcji *.data*. Pozostaje on stały przed jak i po załadowaniu biblioteki do pamięci.
W czasie działania programu (x86-64) za pomocą **licznika programu** oraz tablic PLT i GOT uzyskiwana jest faktyczna pozycja symbolu gdzie po skoku może być wykonywana dalsza ewaluacja kodu.

Zalety: (x86)

- Większe bezpieczeństwo (przy włączonym ASLR)
- Współdzielony blok kodu ze względu na jego charakter RO — mniej miejsca zużytego w pamięci

Wady:

- Niewielki *overhead* związany z niebezpośrednimi zawołaniami funkcji itd.

Część III

Manipulacja i interakcja z kodem

Zarys

Binutils I

GNU Binutils jest zestawem narzędzi do manipulacji, modyfikacji i ogólnej interakcji z plikami obiektowymi

- ld - GNU linker.
- as - GNU asembler.
- addr2line - Konwertuje adresy na nazwy plików i numery lini
- ar - Narzędzie do tworzenia, modyfikowania i ekstrakcji archiwów - biblioteki statyczne, pakiety DEB
- c++filt - *Demangler* symboli C++.
- dlltool - Narzędzie do tworzenia plików koniecznych do budowania plików DLL pod format PE
- gold - Nowy linker, stworzony we współpracy z Google
- gprof - Profiler plików wykonywalnych
- nlmconv - Konwerter plików *relocatable* do formatu NLM

Binutils II

- nm - Listowanie symboli z pliku obiektowego
- objcopy - Kopiuje i tłumaczy pliki obiektowe
- objdump - Wyświetla w czytelny sposób informacje z pliku obiektowego
- ranlib - Dodaje indeks do zawartości archiwum
- readelf - Wyświetla informacje o pliku obiektowym ELF
- size - Wyświetla rozmiary sekcji w pliku obiektowym lub archiwum plików obiektowych
- strings - Wyświetla wszystkie stringi znajdujące się w pliku obiektowym (binarnym)
- strip - *Okraja* plik obiektowy z symboli
- windmc - Kompatybilny z Windowsem kompilator plików typu *message*
- windres - Kompatybilny z Windowsem kompilator plików typu *resource*

gcc

Programy takie jak *ld* czy *as* są narzędziami filozofii UNIXowej, specjalizują się w konkretnych zadaniach.
By osiągnąć nimi sukces musimy bardzo często podać bardzo dużą ilość parametrów, by sprecyzować nasze wymagania.
Dobrze znany nam kompilator/linker **gcc** jest pewnego rodzaju wrapperem na te narzędzia. **Przykład**

Przykłady wykorzystania niektórych narzędzi binutils

`elf.png` [1]

- readelf **(Przykład)**;
- nm **(Przykład nm $1)**;
- objdump **(Przykład objdump -D $1)**;
- c++filt **(Przykład)**

Modyfikowanie ścieżek

Platformy UNIXowe skupiają się na wykorzystaniu bibliotek współdzielonych które umieszczone są w globalnie znanych lokalizacjach takich jak np
“*/usr/lib/*”.
Systemowe pliki nagłówkowe znajdują się m.in. w “*/usr/include/*”

Linker statyczny: Jeśli Pliki *.so/*.a znajdują się w znanym katalogu możemu podać ścieżkę absolutną lub względną z wykorzystaniem flagi **-L** (Dla *.h flaga **-I**). **Przykład**

Linker dynamiczny: Sam plik wykonywalny musi jakoś dostarczać informację linkerowi dynamicznemu o pozycji bibliotek których potrzebuje. Do tego służą zmienne środowiskowe

Wybrane zmienne środowiskowe linkera dynamicznego

- **LD_LIBRARY_PATH**
  Dodaje (z pierwszeństwem) ścieżkę przeszukiwań bibliotek współdzielonych dla linkera dynamicznego (jeśli ścieżki to przedzielone ":") **Przykład**
- **LD_PRELOAD**
  Dodaje (z pierwszeństwem) ścieżkę do konkretnej biblioteki współdzielonej która ma być załadowana przed wszystkimi innymi przez linker dynamiczny (jeśli ścieżki to przedzielone ":")
  Przykład
- **LD_DEBUG**
  Zwraca na strumień błędów różnorakie informacje związane z procesowaniem wykonywanym przez linker dynamiczny (w jaki sposób procesuje relokacje, jakie biblioteki ładuje, jakie symbole ładuje)
  Przykład: LD_DEBUG=all ls

LD_PRELOAD

Zmienna ta jest szczególnie ciekawa ze względu na to, że wraz z mechanizmem *dynamic loading*-u umożliwa podmienianie, a w szczególności *wrapping* symboli dynamicznie dołączanych do jakiejś aplikacji **Przykład**

Część IV

Hotpatching

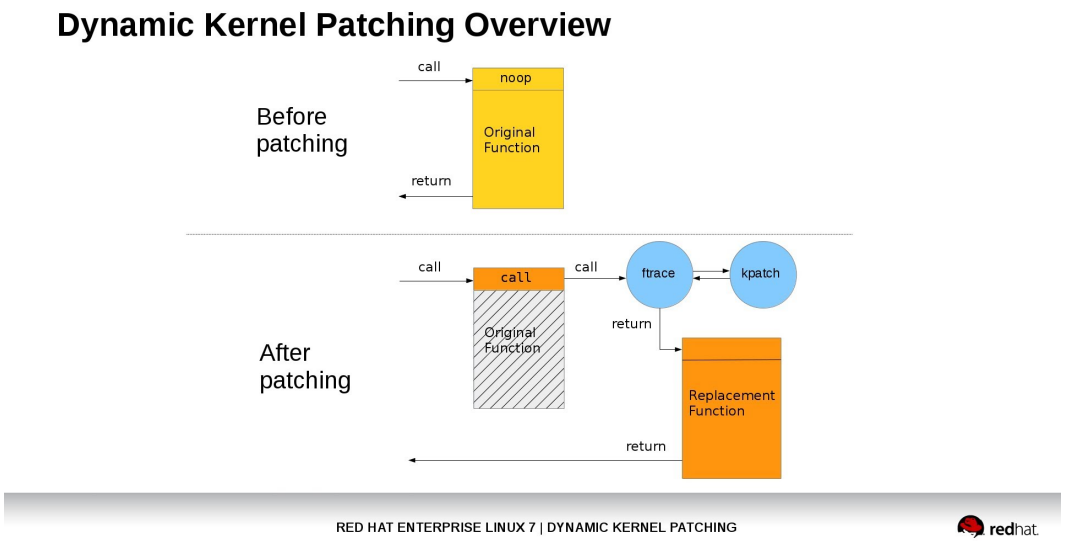
Hotpatching

Technika aktualizacji kodu programu w czasie jego działania. Wykorzystywany przy programach których ciągłość pracy jest priorytetem. Do kategorii tego typu programów należy **kernel**.

Wiele firm oferuje swoje rozwiązania

- Ksplice (Oracle)
- kpatch (Red Hat)
- kGraft (SUSE)

kpatch



Rysunek: Kpatch [2]

Bibliografia I


http://www.cirosantilli.com/elf101.png


https://github.com/dynup/kpatch


http://www.lurklurk.org/linkers/linkers.html


https://carsontang.github.io/unix/2013/06/01/guide-to-object-file-linking/


http://www.skyfree.org/linux/references/ELF_Format.pdf


https://www.symantec.com/connect/articles/dynamic-linking-linux-and-windows-part-one


https://en.wikipedia.org/wiki/Executable_and_Linkable_Format