GNU binutils i wprowadzenie do linkera

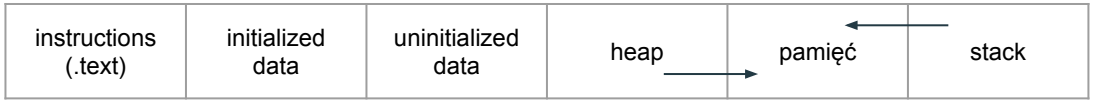
Maciej Trzciński

Pliki ELF

- Executable and Linkable File
- Uniwersalny
- “Nie-do-końca-executable”
- Operuje na względnych adresach pamięci
- Na Windowsie: PE (Portable Executable)

Struktura ELF - podstawy

- Nagłówek
- Program headers
- Section headers
- Sekcje
  - .text
  - .rodata
  - .data
  - .bss



Nagłówek ELF

```
mattcaner@mattcaner-brutus:~/Programowanie/Studia/2019-2020 lato/PN/Projekt2$ readelf -h math1.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          568 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         11
  Section header string table index: 10
```

File data w pliku ELF

- **.text**
- **.data**
- **.rodata**
- **.bss**

Pliki obiektowe w praktyce

- gcc -c tworzy plik obiektowy
- pliki obiektowe mają najczęściej rozszerzenie .o

[przykład 1.1]

Komenda readelf

- **Pozwala uzyskiwać niektóre informacje o pliku elf tak, żeby były czytelne dla człowieka**
- **readelf -h**
- **readelf -s**
- **readelf -S**

[przykład 1.2]

Komenda strings

- wyszukuje w pliku obiektowym wszystkie łańcuchy znaków o długości co najmniej 4 znaków
- strings -f
- strings -t {o,d,x}
- często pomocny bywa tutaj grep

[przykład 1.3]

Linker

- **Po co nam linker?**
- **Jak działa linker?**
- **Kiedy używamy linkera (nie wiedząc o tym)?**

Słowo o procesie kompilacji

- **wyobraźmy sobie duży projekt w C++ (wiele setek funkcji i tysiące linii kodu)**
- **chcemy zmienić pojedynczy plik**
- **co z funkcjami systemowymi, np. printf?**

“Zewnętrzny” kod, czyli biblioteki

- **Dlaczego dzisiaj nic by nie działało bez bibliotek**
- **Gdzie w korzystaniu z bibliotek jest krytyczna rola linkera**

Linkowanie statyczne

- pliki .a (archive)
- biblioteka jest włączana do pliku wykonywalnego w momencie linkowania
- zalety (brak zależności od biblioteki, szybkość)
- wady (większy rozmiar programu, przy zmianie biblioteki trzeba od nowa linkować)

Linkowanie dynamiczne

- pliki .so (shared object)
- zasada działania (biblioteki jest ładowana do programu przy jego uruchomieniu) - o czym należy w takim razie pamiętać?
- zalety (przy zmianie biblioteki program uwzględni zmiany, oszczędność miejsca na dysku)
- wady (trochę wolniejszy czas ładowania programu)

Linkowanie dynamiczne na Windowsie

- pliki .dll (dynamic-link library)
- najważniejsza różnica między .so i .dll - cały .dll nie jest ładowany do programu w momencie jego uruchomienia, poza tym idea .so i .dll jest bardzo podobna
- drobne różnice techniczne
- podejście Windowsa do dll

Symbole w plikach obiektowych

- na Unixie określone (jak wszystko) przez strukturę C
- pod niektórymi względami przypominają zmienne
- globalne i lokalne symbole
- zdefiniowane i niezdefiniowane symbole
- absolutne symbole

Komenda nm

```
mattcaner@mattcaner-brutus:~/Programowanie/Studia/2019-2020 lato/PN/Projekt2$ nm libmymath.so
0000000000201020 B __bss_start
0000000000201020 b completed.7698
                 w __cxa_finalize
0000000000000520 t deregister_tm_clones
00000000000005b0 t __do_global_dtors_aux
0000000000200e88 t __do_global_dtors_aux_fini_array_entry
0000000000201018 d __dso_handle
0000000000200e90 d _DYNAMIC
0000000000201020 D _edata
0000000000201028 B _end
00000000000005fa T factorial
0000000000000780 T _fini
000000000000062c T first_derivative
00000000000005f0 t frame_dummy
0000000000200e80 t __frame_dummy_init_array_entry
00000000000008a0 r __FRAME_END__
0000000000201000 d _GLOBAL_OFFSET_TABLE_
                 w __gmon_start__
000000000000078c r __GNU_EH_FRAME_HDR
00000000000004e8 T _init
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
0000000000000560 t register_tm_clones
0000000000000683 T second_derivative
0000000000000700 T simple_integral
0000000000201020 d __TMC_END__
```

Rodzaje symboli przy użyciu komendy nm

- A (absolute)
- B, b (.bss)
- T, t (.text)
- D, d (.data)
- C (Common, unitialized data)
- R, r (readonly czyli .rodata)
- U (undefined)
- W, V, w (weak)
- u (unique)
- N (debugging)

[przykład 2.1]

Opcje nm

- nm -g
- nm --defined-only
- nm -u
- nm -n
- nm --size-sort

[przykład 2.2]

Komenda strip

- **Usuwa z pliku obiektowego lub wykonywalnego symbole**
- **Po co?**

  [przykład 2.3]

ld

- GNU linker (ld = loader)
- Najczęściej jest używany za naszymi plecami
- Jest całkowicie darmowy, (dość) dobrze udokumentowany i elastyczny
- Ważna część gcc
- Występuje również w wersji na Windowsa
- Udostępnia funkcjonalności do pracy na mikrokontrolerach (np. na ARMie, który już znamy)

Podstawowe użycie

- **Wyciągnięte z man:**

  ld -o <output> /lib/crt0.o hello.o -lc

- **ld [opcje] <pliki do zlinkowania> [opcje]**
- **opcji jest dużo i mogą mieć bardzo złożone działanie**
- **Możemy równie dobrze manipulować działaniem ld przy okazji użycia gcc, dzięki opcji -Wl**

Opcje linkera ld

- ld -r
- ld -o
- ld -L<katalog bibliotek>
- ld -l<biblioteka>
- ld -so<nazwa_biblioteki>
- ld -rpath<ścieżka>
- ld -T <plik skryptowy>

[przykład 3.1]

Skrypty linkera ld

- **Pozwalają na dużą kontrolę nad końcowym wyglądem pliku**
- **Składnia przypomina C**

Przykłady skryptów linkera

- definiowanie symboli: symbol = wartość
- HIDDEN (...)
- ENTRY (...)
- OUTPUT_FORMAT (...)

Skrypty linkera: SECTIONS

- **location counter (kropka)**
- **sekcja : pozycja**
- **wyrażenie *(...)**
- **ALIGN(...)**

Przykład z dokumentacji

```
SECTIONS
{
  . = 0x10000;
  .text : { *(.text) }
  . = 0x8000000;
  .data : { *(.data) }
  .bss : { *(.bss) }
}
```

Przykład użycia skryptu

[przykład 3.2]

Jak gcc korzysta z linkera?

- **Możemy to łatwo sprawdzić, używając gcc -v (verbose)**
- **Możemy wstawiać komendy linkera do gcc (!), składnia ma wtedy postać gcc [opcje] -Wl,<opcje ld> [pozostałe opcje gcc]**
- **Używając gcc -Wl, możemy korzystać ze skryptów linkera w gcc**

[przykład 3.3]

Tworzenie biblioteki współdzielonej

[przykład 4.1]

Tworzenie bibliotek DLL

- **Jak tworzyć własne biblioteki .dll na Windowsie?**

  [przykład 5.1]

Linker gold

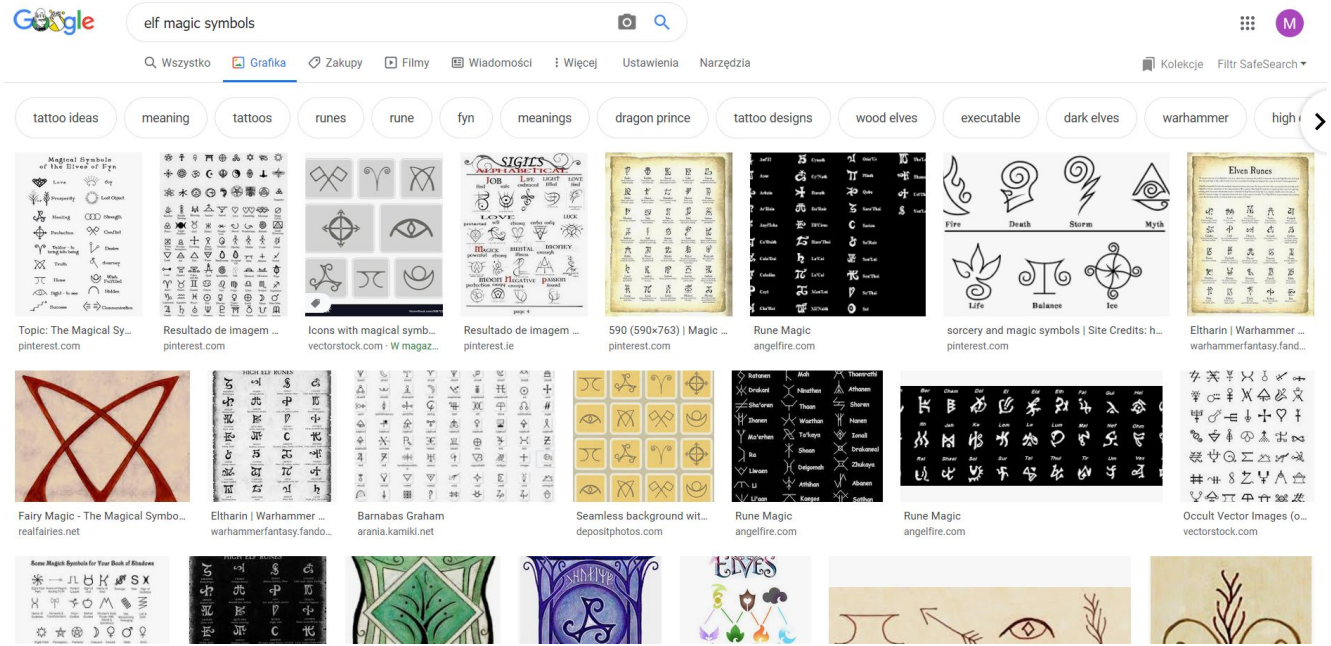
- Szybki linker, wciąż jeszcze w wersji beta
- Żeby użyć w gcc: opcja -fuse-ld=gold
- Opinie użytkowników są mieszane

[przykład 6.1]

Podsumowanie

- Pliki obiektowe umożliwiają etapy pośrednie w kompilacji
- Linker jest krytyczny dla działania większości dzisiejszych systemów, mało który program nie wymaga do działania jakiejkolwiek biblioteki
- GNU binutils umożliwiają stosunkowo łatwe tworzenie własnych bibliotek .so oraz .dll oraz dają narzędzia do pracy na nich oraz na plikach ELF, podając nam wiele informacji w sposób wygodniejszy niż gdybyśmy robili hexdump
- Oczywiście istnieją narzędzia, które zrobią część z tych rzeczy za nas, ale szczególnie, jeśli pojawiłby się w nich bug albo nietypowe działanie, warto znać mechanizmy “od podstaw”

Na koniec - uważajcie na szukanie “elf magic symbols” w google



Przydatne materiały dla tych, którzy chcą (lub potrzebują) wiedzieć więcej

Dokumentacja GNU Binutils:
http://sourceware.org/binutils/docs-2.34/
Opis standardu ELF, z rzeczami, na które tutaj nie mieliśmy czasu, napisany dość prostym językiem:
https://linux-audit.com/elf-binaries-on-linux-understanding-and-analysis/
Dokładniejszy opis standardu ELF, stworzony przez UNIX System Laboratories:
http://www.skyfree.org/linux/references/ELF_Format.pdf
Nagłówek ELF - dokładne wyjaśnienie, co jest czym, bajt po bajcie:
https://refspecs.linuxfoundation.org/elf/gabi4+/ch4.eheader.html
122 strony porządnego opracowania o linkerze ld:
https://www.eecs.umich.edu/courses/eecs373/readings/Linker.pdf
Tworzenie bibliotek DLL w Visual Studio:
https://github.com/MicrosoftDocs/cpp-docs/blob/master/docs/build/walkthrough-creating-and-using-a-dynamic-link-library-cpp.md
Opinie o gold - czy warto?:
https://lwn.net/Articles/274859/

A w razie czego polecam strony manuala

Dziękuję za uwagę