PB173 – Binární programování Linux

VII. Linker skripty

Jiri Slaby

Fakulta informatiky
Masarykova univerzita

12. 11. 2015

Linker skripty

- `ldscripts`
- Linker se jimi řídí při linkování objektů
  - Tj. vždy se nějaký použije
- Udávají
  - Kde která sekce leží
    - Např. pro embedded, kód musí být na adrese X a data na adrese Y
  - Výstupní formát
  - Kam skočit po načtení
  - A další
- Může přidávat i symboly
  - Např. „toto” je adresa začátku sekce Z
- Dokumentace
  - `pinfo ld`
  - (`http://sourceware.org/binutils/`)

Formát skriptu

- Textové soubory
- Příkazy
  - Pozice první instrukce, kterou spustit: `ENTRY`
  - Výstupní formát: `OUTPUT_FORMAT`
  - Seznam sekcí a jejich obsah: `SECTIONS`
  - Verzování: `VERSION`
  - ...
- Komentáře
  - `/**/` jako v C
- Linkeru se předává při linkování
  - `gcc -T skript.x ...`
  - `ld -T skript.x ...`
- Demo: výchozí skript

Základní příkazy

- `ENTRY`
  - Udává, kam se má skočit po zavedení
  - Např. `ENTRY(main)`
- `OUTPUT_FORMAT`
  - Výstupní BFD formát
  - `OUTPUT_FORMAT("elf64-x86-64")` (popř. `elf32-i386`)

Úkol

Základní skript

1. Vytvořte si linker skript
2. Bude obsahovat pouze `ENTRY` a `OUTPUT_FORMAT`
3. Vytvořte zdrojový soubor s jednou funkcí
   - Bude obsahovat nekonečný cyklus
   - Jméno bude stejné jako v `ENTRY`
4. Přeložte a slinkujte skriptem
   - `gcc -Wl,--build-id=none -nostdlib -T skript.x ...`
5. Podívejte se na rozložení sekcí
   - Měly by začínat okolo adresy 0
6. Spusťte program
   - Bude fungovat jen pokud `sysctl vm.mmap_min_addr` je 0
   - Pokud není, změňte

Sekce

- Blok `SECTIONS { ... }`
- Záznamy typu `.vystupni_sekce : { soubor(.vstupni_sekce)}`
  - `soubor` může být `*`
- `/DISCARD/` je speciální výstupní sekce
  - Obsah se zahodí

Příklad

```
SECTIONS
{
  .text     : { *(.text) }
  .data     : { *(.data) soubor.o(.data.special) }
  .bss      : { *(.bss) }
  /DISCARD/ : { *(.debug*) *(.comment*) *(.eh_frame*) }
}
```

Úkol

Skript se sekcemi

1. Rozšiřte skript, aby výsledek obsahoval pouze sekci `.text`
2. Přeložte
3. Ověřte pomocí `objdump -h`
4. Spusťte

Sekce – adresování

- Každému bodu ve skriptu lze přiřadit adresu
  - Aktuální adresa je `.` (tečka)
  - `. = 0x10000;`
- Každou sekci lze:
  - Vložit na nějakou adresu: `.sekce 0x10000 : { ... }`
  - Zarovnat: `.sekce ALIGN(...): { ... }`
- Pro každou adresu lze vložit symbol
  - `symbol = .;`
  - V kódu pak `extern unsigned long symbol;`

Příklad

```
SECTIONS
{
  . = 0x40000;
  my_start = .;
  .text : { *(.text) }
  .data 0x50000 : { *(.data) }
}
```

Úkol

Skript se sekcemi a adresami

1. Rozšiřte zdrojový soubor o další 2 funkce, tedy:
   - 1 (původní): s nekonečným cyklem, dejte ji do sekce `.text.t2`
   - 2: bude volat `extern void my_fun(void)`, do sekce `.text.t1`
   - 3: bude volat adresu `0x401000`, nastavte `ENTRY` na její jméno
2. Rozšiřte skript
   - Aby všechny sekce začínaly za hranicí `0x400000`
     - `. = ...` na začátku bloku `SECTIONS`
   - Přidejte novou výstupní sekci `.text.t` na adresu `0x401000`
     - Obsah `.text.t1`
     - Nový symbol `my_fun` na aktuální adrese
     - Obsah `.text.t2` (tedy `my_fun` ukazuje na začátek `.text.t2`)
3. Přeložte
4. Ověřte pomocí `objdump -h` a `objdump -t`
5. Spusťte

Sekce 2

Verzování funkcí

Verzování funkcí

- Pro zachování zpětné kompatibility knihovních funkcí
  - V knihovnách se v průběhu času opravují chyby
    - Ale někteří „uživatelé” knihovny spoléhali na chyby
  - Jak zachovat zpětnou kompatibilitu nějaké funkce?
- Chceme 2 funkce s týmž názvem, ale s jinou funkcionalitou
  - Podle verze použité při linkování

Před opravou chyby	Po opravě chyby
`void fun() { puts("OLD"); }`	`void fun() { puts("NEW"); }`
libX v1	libX v2

Řešením jsou linker skripty s verzováním

Verzování funkcí

- Každé funkci lze ve skriptu:
  - Přiřadit verzi (`fun` v1, `fun` v2 nebo výchozí)
  - Určit její viditelnost
- Blok `VERSION { ... }`
- Obsah `VERSION` lze předat i samostatně na příkazové řádce
  - Odpadá nutnost psaní celého linker skriptu
  - `ld --version-script=verze.x ...`
  - `gcc -Wl,--version-script=verze.x ...`
- Zobrazení pomocí `readelf -V`
- Dokumentace
  - `pinfo ld`
  - U. Drepper: How To Write Shared Libraries, kap. 3

Verze ve skriptu

Obsah sekce `VERSION` ve skriptech

- Strom verzí – dědičnost
- Uzly obsahují funkce
- Uzly mohou obsahovat i označení viditelnosti
  - `global:`
  - `local:`

Příklad

```
VERS_1.1 {
  global:
    fun;
  local:
    old_fun*;
};

VERS_1.2 {
  new_functionality_fun;
} VERS_1.1; /* inheritance */
```

Verze ve zdrojovém souboru

- Ve zdrojových souborech je třeba přiřadit verze
- Globální `asm` direktiva
  - Neverzovaná: `asm(".symver fun_old1,fun@");`
  - Pro určitou verzi: `asm(".symver fun_old2,fun@VERS_1");`
  - Výchozí: `asm(".symver fun_new,fun@@VERS_2");`

Příklad

```
void fun_old()  {
   puts("OLD");
}
asm(".symver fun_old,fun@");

void fun_new() {
   puts("NEW");
}
asm(".symver fun_new,fun@@VERS_2");
```

Úkol

Verzování

1. Otevřete si `pb173-bin/07/`
2. Projděte 3 verze knihovny `vers{1,2,3}.c`
   - Definuje funkci `bubak`
   - `bubak` v nich má 3 různé sémantiky podle verze
3. Projděte soubor, který ji používá `x.c`
   - Volá funkci `bubak`
   - (Pro 3. verzi předává i parametr)
4. Přeložte
   - Standardně pomocí `make`
5. Spusťte `x{1,2,3}`
6. Projděte verze v `x{1,2,3}` za pomoci `readelf -V`
7. Pokračujte domácím úkolem, částí 2