PB173 – Binární programování Linux

XII. Knihovny

Jiri Slaby

Fakulta informatiky
Masarykova univerzita

6. 12. 2016

Obsah cvičení

Sekce 1

Knihovny

Knihovny

- Statické knihovny
  - `ar` archiv
  - Jejích *obsah* je v každé binárce, kam se linkovaly
  - Zvyšují velikost na disku
  - Binárka ale nemá žádné externí závislosti
  - Výroba: `gcc -c` a `ar rcs lib.a *.o`
- Dynamické knihovny
  - ELF, který se linkuje za chodu
    - Podpora pluginů
    - Ale zpomaluje spuštění
    - Dynamický linker `ld.so` (`man ld.so`)
  - Výroba: `gcc -shared`

Úkol

Výroba statické knihovny

1. Proveďte `make` v `pb173-bin/12`
2. Z `lib.o` vyrobte statickou knihovnu `libX.a`
   - `ar rcs ...`
3. Knihovnu přilinkujte k `main.o`
   - `gcc -L. -lX`
4. Spusťte výsledek

Použití dynamické knihovny

1. Přilinkování pomocí linkeru
   - `gcc -lknihovna`
   - `ld.so` před spuštěním načte a slinkuje potřebné symboly
   - **Demo:** `ldd` k ověření
2. Načtení explicitně až za běhu
   - Viz dále
3. Proměnnými prostředí
   - `LD_PRELOAD` nahraje knihovnu před spuštěním

Načtení dynamické knihovny

- Knihovna `dl` (linkujte s `-ldl`) (`dlfcn.h`)
- Otevření: `void *dlopen(const char *filename, int flags)`
  - `filename`: jméno knihovny, nebo cesta
  - `flags`: `RTLD_LAZY` nebo `RTLD_NOW` spolu s dalšími
  - Návratová hodnota: držátko knihovny nebo `NULL` při chybě
- Zavření: `int dlclose(void *handle)`
- Symbol z knihovny: `void *dlsym(void *handle, const char *symbol)`
  - `handle`: držátko z `dlopen`
  - `symbol`: název symbolu (funkce, proměnné, ...)
  - Návratová hodnota: ukazatel na začátek symbolu
- Řetězec chyby: `char *dlerror(void)`

Načtení dynamické knihovny – příklad

```
#include <dlfcn.h>
#include <err.h>
#include <stdio.h>

int main(void) {
  void *h = dlopen("libm.so.6", RTLD_LAZY);
  if (!h)
    errx(1, "dlopen: %s", dlerror());

  double (*kos)(double) = dlsym(h, "cos");
  if (!kos)
    errx(1, "dlsym(cos): %s", dlerror());

  printf("%f\n", kos(0));

  dlclose(h);

  return 0;
}
```

Úkol

Načtení dynamické knihovny

1. Načtěte dynamickou knihovnu
   - Zadaná na příkazové řádce
2. Načtěte symbol
   - Zadaný také na příkazové řádce
3. Zavolejte symbol
   - S řetězcem jako parametr
4. Vyzkoušejte
   - Např. `./main libc.so.6 puts`
   - A `printf`
   - A `fclose`?

Sekce 2

Vytváření dynamických knihoven

Pojmenování dynamických knihoven

- Jméno knihovny („soname”)
  - `libXXX.so.maj_verze`
  - Např. `libelf.so.0`
- Reálné jméno (soubor)
  - `libXXX.so.maj_verze.min_verze.release`
  - Např. `libelf.so.0.8.13`
- Jméno pro linker
  - `libXXX.so`
  - Např. `libelf.so`

Vytváření dynamických knihoven

- Všechen kód musí být nezávislý na pozici, kde bude nahrán
  - ⇒ nutno překládat i linkovat s `-fPIC`, popř. lépe s `-fpic`
- Linker by měl znát soname
  - ⇒ nutno linkovat s `-Wl,-soname,jmeno`
- Linkovat sdíleně
  - ⇒ nutno linkovat s `-shared`

Příklad vytvoření knihovny

```
gcc -fpic -g -c -Wall a.c
gcc -fpic -g -c -Wall b.c
gcc -fpic -shared -Wl,-soname,libmylib.so.1 -o libmylib.so.1.2.3 a.o b.o -lc
```

Úkol

Vytvoření dynamické knihovny

1. Z `lib.o` vytvořte dynamickou knihovnu
   - `gcc -shared -Wl,-soname,...`
2. Knihovnu přilinkujte k `main.o` namísto statické
   - `gcc -L. -lX`
3. Ověte pomocí `ldd`
4. Spusťte výsledek

Přesměrování funkcí

- Knihovny mohou podvrhnout své implementace funkcí
  - Např. `malloc` pro sledování paměti
- Podmínka: funkce musí být z dynamické knihovny
- `LD_PRELOAD` nahraje knihovnu i se zvláštní implementací
- `dlsym` lze použít k nalezení originálu
  - `void *dlsym(void *handle, const char *symbol)`
  - Speciální `handle` – `RTLD_NEXT`

Příklad

```
void *malloc(size_t size)
{
  static void *(*c_malloc)(size_t);
  if (!c_malloc)
    c_malloc = dlsym(RTLD_NEXT, "malloc");

  write(1, "Somebody called malloc!\n", 24);
  return c_malloc(size);
}
```

Úkol

Přesměrování `malloc` (domácí)

1. Vytvořte novou knihovnu
2. Přesměrujte volání `malloc` a `free` do své knihovny
   - Volejte originály
   - Originály zjistěte v konstruktoru
3. Spočítejte, kolikrát se oba zavolají
4. Spočítejte, kolik MiB se alokovalo a uvolnilo
5. Na konci vypište statistiku
   - A tu vepište do commitlogu
   - Můžete použít např. `atexit` nebo `destructor`
6. Spusťte na libovolném programu, který používá alokace
7. Zkuste z `malloc` občas vrátit `NULL`
   - Co se stane?