Architektury systemów komputerowych 2017

Lista zadań nr 6

Na zajęcia 5 i 6 kwietnia 2017

UWAGA! Zadania z tej listy należy rozwiązywać na komputerze z systemem operacyjnym `Linux` dla platformy `x86-64`. Prowadzący zakłada, że zainstalowana dystrybucja będzie bazowała na `Debianie` (np. `Ubuntu` lub `Mint`).

W trakcie prezentacji rozwiązań należy zdefiniować i wyjaśnić pojęcia, które zostały oznaczone **wytłuszczoną** czcionką.

Zadanie 1. Poniżej podano zawartość pliku `swap.c`:

```
1  extern int buf[];
2
3  int *bufp0 = &buf[0];
4  static int *bufp1;
5
6  static void incr() {
7    static int count = 0;
8    count++;
9  }
10 void swap() {
11   int temp;
12   incr();
13   bufp1 = &buf[1];
14   temp = *bufp0;
15   *bufp0 = *bufp1;
16   *bufp1 = temp;
17 }
```

Dla każdego elementu tablicy symboli `.symtab` zdefiniowanych lub używanych w `swap.o` podaj:

- typ symbolu (`local`, `global`, `extern`),
- rozmiar danych, na które wskazuje symbol,
- numer i reprezentację tekstową (`.text`, `.data`, `.bss`, itd.) sekcji, do której odnosi się symbol.

Które linie w powyższym kodzie będą wymagać dodania wpisu do tablicy relokacji?

Zadanie 2. Rozważmy program skompilowany z opcją `-Og` składający się z dwóch plików źródłowych:

```
1 /* foo.c */
2 void p2(void);
3
4 int main() {
5   p2();
6   return 0;
7 }
```

```
1 /* bar.c */
2 #include <stdio.h>
3
4 char main;
5
6 void p2() {
7   printf("0x%x\n", main);
8 }
```

Po uruchomieniu program drukuje pewien ciąg znaków i kończy działanie bez zgłoszenia błędu. Czemu tak się dzieje? Skąd pochodzi wydrukowana wartość? Zauważ, że zmienna `main` w pliku `bar.c` jest niezainicjowana. Co by się stało, gdybyśmy w funkcji `p2` przypisali wartość pod zmienną `main`?

Zadanie 3. Poniższy kod w języku `C` skompilowano z opcją `-Og`. Następnie sekcję `.text` otrzymanej jednostki translacji poddano deasemblacji. Kompilator umieścił tablicę skoków dla instrukcji wyboru w sekcji `.rodata` i wypełnił zerami. Dla obydwu sekcji określ pod jakimi miejscami znajdują się relokacje, a następnie podaj zawartość tablicy relokacji `.rela.text` i `.rela.rodata`, tj. listę rekordów składających się z:

- przesunięcia relokacji względem początku sekcji,
- typu relokacji,
- nazwy symbolu.

```
1  int relo3(int val) {
2    switch (val) {
3      case 100:
4        return val;
5      case 101:
6        return val + 1;
7      case 103:
8      case 104:
9        return val + 3;
10     case 105:
11       return val + 5;
12     default:
13       return val + 6;
14   }
15 }
```

```
0000000000000000 <relo3>:
 0: 8d 47 9c              lea    -0x64(%rdi),%eax
 3: 83 f8 05              cmp    $0x5,%eax
 6: 77 15                 ja     1d <relo3+0x1d>
 8: 89 c0                 mov    %eax,%eax
 a: ff 24 c5 00 00 00 00  jmpq   *0x0(,%rax,8)
11: 8d 47 01              lea    0x1(%rdi),%eax
14: c3                    retq
15: 8d 47 03              lea    0x3(%rdi),%eax
18: c3                    retq
19: 8d 47 05              lea    0x5(%rdi),%eax
1c: c3                    retq
1d: 8d 47 06              lea    0x6(%rdi),%eax
20: c3                    retq
21: 89 f8                 mov    %edi,%eax
23: c3                    retq
```

Zadanie 4. Zapoznaj się z narzędziami do analizy **plików relokowalnych** w formacie `ELF` i bibliotek statycznych, tj. `objdump`, `readelf` i `nm`; a następnie odpowiedz na następujące pytania:

1. Ile modułów translacji zawierają biblioteki `libc.a` i `libm.a` (katalog `/usr/lib/x86_64-linux-gnu`)?
2. Czy polecenie «`gcc -Og`» generuje inny kod wykonywalny niż «`gcc -Og -g`»?
3. Z jakich bibliotek współdzielonych korzysta interpreter języka `Python` (plik `/usr/bin/python`)?

Zaprezentuj w jaki sposób można dojść do odpowiedzi korzystając z powyższych poleceń.

Zadanie 5. Korzystając z **dyrektyw asemblera** opisanych w GNU as: Assembler Directives[1] pokaż jak:

1. zdefiniować globalną funkcję `foobar`,
2. zdefiniować lokalną strukturę podaną niżej:

```
static const struct {
  char a[3]; int b; long c; float pi;
} baz = { "abc", 42, -3, 1.4142 };
```

3. zarezerwować miejsce dla tablicy `long array[100]`?

Pamiętaj, że dla każdego zdefiniowanego symbolu należy uzupełnić odpowiednio tablicę `.symtab` o typ symbolu i rozmiar danych, do których odnosi się symbol.

Zadanie 6. Język `C++` pozwala na przeciążanie funkcji, tj. dopuszcza stosowanie wielu funkcji o tej samej nazwie, ale różnej sygnaturze. Wiemy, że symbole na których operuje konsolidator są beztypowe. Powstaje zatem problem unikalnej reprezentacji symboli funkcji przeciążonych. Wytłumacz na czym polega **dekorowanie nazw** (ang. *name mangling*)? Które elementy składni podlegają dekorowaniu?

Przy pomocy narzędzia `c++filt` przekształć poniższe nazwy na sygnatury funkcji języka `C++` i omów znaczenie poszczególnych fragmentów symbolu. Czy funkcja dekorująca nazwy jest różnowartościowa?

1. `_Z4funcPKcRi`
2. `_ZN3Bar3bazEPc`
3. `_ZN3BarC1ERKS_`
4. `_ZN3foo6strlenER6string`

Zadanie 7 (2). Na podstawie rozdziału §7.12 podręcznika *„Computer Systems: A Programmer's Perspective"* opisz proces **leniwego wiązania** (ang. *lazy binding*) symboli i odpowiedz na następujące pytania:

- Czym charakteryzuje się **kod umieszczalny pod dowolnym adresem** (`PIC`)?
- Do czego służą sekcje `PLT` i `GOT` – jakie dane są tam przechowywane?
- Czemu sekcja `PLT` jest modyfikowalna, a sekcje kodu i `GOT` są tylko do odczytu?
- Co znajduje się w sekcji `.dynamic`?

Wskazówka: Wykorzystaj rysunek 7.19.

[1] `https://sourceware.org/binutils/docs-2.26/as/Pseudo-Ops.html`