Architektury systemów komputerowych

Lista zadań nr 9

Na zajęcia 14 maja 2020

Zadanie 1. W trakcie tłumaczenia poniższego kodu na asembler kompilator umieścił tablicę skoków dla instrukcji przełączania w sekcji «.rodata».

```
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
```

Dla sekcji «.text» i «.rodata» 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 i przesunięcia względem początku symbolu.

W wyniku konsolidacji pliku wykonywalnego zawierającego procedurę «relo3», została ona umieszczona pod adresem 0x1000, a tablica skoków pod 0x2000. Oblicz wartości, które należy wstawić w miejsca relokacji.

Zadanie 2. Język C++ pozwala na przeciążanie funkcji (ang. *function overloading*), tj. dopuszcza stosowanie wielu funkcji o tej samej nazwie, ale różnej sygnaturze. Wiemy, że konsolidator nie przechowuje w tabeli symboli informacji o typach z języka programowania. Powstaje zatem problem unikatowej reprezentacji nazw używanych przez język C++. 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

Wskazówka: Szczegółowe informacje na temat kodowania nazw można znaleźć w C++ ABI: External Names[1].

[1] https://itanium-cxx-abi.github.io/cxx-abi/abi.html#mangling

Zadanie 3. Podglądając wyjście z kompilatora języka `C` pokaż jak korzystając **dyrektyw asemblera** opisanych w GNU as: Assembler Directives[2]:

- zdefiniować globalną funkcję `foobar`,
- zdefiniować lokalną strukturę podaną niżej:

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

- zarezerwować miejsce dla tablicy `long array[100]`.

Wyjaśnij znaczenie poszczególnych dyrektyw. Pamiętaj, że dla każdego zdefiniowanego symbolu należy uzupełnić odpowiednio tabelę «`.symtab`» o typ symbolu i rozmiar danych, do których odnosi się symbol.

Zadanie 4. Na podstawie [1, §7.12] opisz proces **leniwego wiązania** (ang. *lazy binding*) symboli i odpowiedz na następujące pytania:

- Czym charakteryzuje się **kod relokowalny** (ang. *Position Independent Code*)?
- Do czego służą sekcje «`.plt`» i «`.got`» – jakie dane są tam przechowywane?
- Czemu sekcja «`.got`» jest modyfikowalna, a sekcje kodu i «`.plt`» są tylko do odczytu?
- Co znajduje się w sekcji «`.dynamic`»?

Zaprezentuj leniwe wiązanie na podstawie programu «`lazy`». Uruchom go pod kontrolą debuggera `gdb`, ustaw punkty wstrzymań (ang. *breakpoint*) na linię 4 i 5. Po czym wykonując krokowo program (`stepi`) pokaż, że gdy procesor skacze do adresu procedury «`puts`» zapisanego w «`.got`» – za pierwszym wywołaniem jest tam przechowywany inny adres niż za drugim.

Wskazówka: Wykorzystaj rysunek 7.19. Dobrze jest zainstalować sobie `GDB dashboard`[3].

Literatura

[1] „Computer Systems: A Programmer's Perspective”
Randal E. Bryant, David R. O'Hallaron; Pearson; 3rd edition, 2016

[2] „System V Application Binary Interface AMD64 Architecture Processor Supplement”
H.J. Lu, Michael Matz, Milind Girkar, Jan Hubička, Andreas Jaeger, Mark Mitchell;
https://github.com/hjl-tools/x86-psABI/wiki/x86-64-psABI-1.0.pdf

[2] `https://sourceware.org/binutils/docs-2.26/as/Pseudo-Ops.html`
[3] `https://github.com/cyrus-and/gdb-dashboard`