Systemy komputerowe

Lista zadań nr 5

Na zajęcia 28 marca – 1 kwietnia 2019

UWAGA! Rozwiązania poniższych zadań studenci będą prezentować przy pomocy komputera (x86-64 GNU/Linux) z projektorem dostarczonych przez prowadzącego ćwiczenia. Prezentowane rozumowania należy uzasadnić wydrukami z odpowiednich narzędzi. Można używać notatek.

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

```
1  #include "stdio.h"
2
3  static int global = 15210;
4
5  static void set_global(int val) {
6          global = val;
7  }
10 int main(void) {
11         printf("before: %d\n", global);
12         set_global(15213);
13         printf("after: %d\n", global);
14         return 0;
15 }
```

Polecenie `gcc main.c -o main` jest równoważne ciągowi poleceń
`cpp -o main_p.c main.c; gcc -S main_p.c; as -o main.o main_p.s; gcc -o main main.o`.

- Jaka jest rola poszczególnych poleceń w tym ciągu?
- Skąd pochodzi kod, który znalazł się w pliku `main_p.c`?
- Co zawiera plik `main_p.s`. Zauważ etykiety odpowiadające zmiennej `global` i obydwu funkcjom. W jaki sposób przyporządkować etykiecie jej typ?
- Poleceniem `objdump -t` wyświetl tablicę symboli pliku `main.o`. Jakie położenie wg. tej tablicy mają symbole `global` i `set_global`?
- Poleceniem `objdump -h` wyświetl informacje o sekcjach w pliku `main.o`. Dlaczego adres sekcji `.text` i `.data` to 0? Jakie są adresy tych sekcji w pliku wykonywalnym `main`?

Zadanie 2. Poniżej podano zawartość pliku `swap.c`:

```
1  extern int buf[];
2
3  int *bufp0 = &buf[0];
4  static int *bufp1;
5  int intvalue = 0x77;
6
7  static void incr() {
8    static int count = 0;
9    count++;
10 }
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.

Wskazówka: Wyprodukuj plik swap.o, następnie użyj poleceń objdump -t swap.o oraz objdump -s -j nazwa_sekcji swap.o. Zapoznaj się z opisem tych opcji w systemowym podręczniku dla objdump (man objdump)

Zadanie 3. Rozważmy program skompilowany z opcją `-Og` składający się z dwóch plików źródłowych:

```c
/* foo.c */
void p2(void);

int main() {
  p2();
  return 0;
}
```

```c
/* bar.c */
#include <stdio.h>

char main;

void p2() {
  printf("0x%x\n", main);
}
```

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`? Co by się zmieniło gdybyśmy w pliku `bar.c` zainicjowali zmienną `main` w miejscu jej definicji? Odpowiedzi uzasadnij posługując się narzędziem `objdump`.

Wskazówka: Może się przydać opcja -d polecenia objdump

Zadanie 4. Które wiersze w kodzie z zadania drugiego będą wymagać dodania wpisu do tablicy relokacji?

Wskazówka: Zastanów się jakie dodatkowe informacje należy umieścić w plikach relokowalnych, by umieć powiązać miejsca wywołania procedur z położeniem procedury w skonsolidowanym pliku wykonywalnym. Mogą przydać się opcje -d oraz -d -r narzędzia objdump.

Zadanie 5. Ponownie rozważamy program z zadania pierwszego. Zdezasembluj plik `main.o` i zidentyfikuj punkty wywołania funkcji `printf` oraz referencje do zmiennej `global`. Potwierdź poprawność tych czynności analizując wpisy w tablicy relokacji dla pliku `main.o` (polecenie `objdump -r`). Jakich obiektów dotyczą wpisy odnoszące się do wartości z sekcji `.rodata`?

Zadanie 6. Poniższy kod w języku `C` skompilowano z opcją `-Og`. Następnie sekcję `.text` otrzymanej jednostki translacji poddano deasemblacji. Kompilator umieścił tablicę skoków dla instrukcji przełączania 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 i przesunięcia względem początku symbolu.

```c
int relo3(int val) {
  switch (val) {
    case 100:
      return val;
    case 101:
      return val + 1;
    case 103:
    case 104:
      return val + 3;
    case 105:
      return val + 5;
    default:
      return val + 6;
  }
}
```

```
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
```

Wskazówka: Może pomóc narzędzie readelf