

Architektury systemów komputerowych

Lista zadań nr 8

Na zajęcia 4 maja 2022

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 11*. Do poniższej listy załączono na stronie przedmiotu pliki źródłowe wraz z plikiem *Makefile*.

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

Zadanie 1. Poniżej podano zawartość pliku «*swap.c*». Wskaż w nim wszystkie wystąpienia definicji i referencji do **symboli** [1, §7.5]. Dla każdego symbolu wskaż jego **zasięg widoczności** (tj. lokalny, globalny, zewnętrzny) oraz nazwę **sekcji**, w której go umieszczono (tj. «*.text*», «*.data*», «*.rodata*», «*.bss*»). Wydając polecenie «*make swap.o*» wygeneruj **plik relokowalny** i zweryfikuj swoje odpowiedzi na podstawie wydruku z polecenia ***nm***¹. Do czego *konsolidator* wykorzystuje tablicę symboli?

```
1 extern int printf(          7 static void incr() {          16 void swap(int i) {
2     const char *, ...);      8     static int count = 0;      17     incr();
3 extern long buf[];          9     count++;                  18     long temp = *bufp0;
4                               10 }                             19     *bufp0 = buf[i];
5 long *bufp0 = &buf[0];      11                               20     buf[i] = temp;
6 static double sum = 0.0;    12 void addf(void) {            21 }
                               13     sum += 3.14;
                               14     printf("sum = %f\n", sum);
                               15 }
```

Zadanie 2. Co przechowują sekcje «*.strtab*» i «*.shstrtab*» [2, 4-17]? Opisz znaczenie pól rekordu symbolu «*Elf64_Sym*» [2, 4-18] przechowywanego w **tablicy symboli** [2, 4], oraz pól rekordu **nagłówka sekcji** «*Elf64_Shdr*» [2, 4-8]. Na podstawie zdobytej wiedzy omów wydruk polecenia ***readelf***² z opcjami «*-t -s*» na pliku «*swap.o*». Skąd wiadomo gdzie w *pliku relokowalnym* znajdują się nagłówki sekcji oraz zawartość poszczególnych sekcji? Jaka jest pozycja danego symbolu względem początku sekcji?

Zadanie 3. Rozważmy program składający się z dwóch plików źródłowych:

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

Po uruchomieniu program drukuje pewien ciąg znaków i kończy działanie bez zgłoszenia błędu. Cemu tak się dzieje? Skąd pochodzi wydrukowana wartość? Czym różni się **symbol silny** od **słabego**? Zauważ, że zmienna «*main*» w pliku «*mismatch-b.c*» jest niezainicjowana. Co by się stało, gdybyśmy w funkcji «*p2*» przypisali wartość pod zmienną «*main*»? Co by się zmieniło gdybyśmy w pliku «*mismatch-b.c*» zainicjowali zmienną «*main*» w miejscu jej definicji? Cemu dobrym pomysłem jest przekazywanie opcji «*-fno-common*» do kompilatora?

Zadanie 4. Opisz znaczenie pól **nagłówka ELF** «*Elf64_Ehdr*» [2, 4-3] oraz tablicy rekordów **nagłówków programu** «*Elf64_Phdr*» [2, 5-1], zwanych również *segmentami programu*. Na podstawie zdobytej wiedzy omów wydruk polecenia «*readelf -h -l*» na pliku «*mismatch*». Skąd wiadomo gdzie znajduje się pierwsza instrukcja **pliku wykonywalnego** [1, §7.8]? Czy to będzie początek funkcji «*main*»? Pod jakie adresy wirtualne zostaną załadowane poszczególne segmenty programu? Które z sekcji nie zostaną załadowane do pamięci? Z których sekcji procesor będzie mógł wyłącznie czytać?

¹<https://sourceware.org/binutils/docs/binutils/nm.html>

²<https://sourceware.org/binutils/docs/binutils/readelf.html>

Zadanie 5. Rozważmy poniższy program składający się z dwóch *jednostek translacji*. Po uruchomieniu kończy się on z błędem dostępu do pamięci. Przy pomocy debuggera gdb zatrzymaj się w miejscu wystąpienia awarii i wyjaśnij jej przyczynę. Gdzie została umieszczona stała znakowa «Hello, world!»? Popraw program tak, by zakończył się poprawnie. Nie wolno modyfikować sygnatury procedury «sometr» i pliku «str-a.c», ani korzystać z dodatkowych procedur. Gdzie umieszczono ciąg znaków po poprawce?

```

1 /* str-a.c */
2 #include <stdio.h>
3
4 char *sometr(void);
5
6 int main(void) {
7     char *s = sometr();
8     puts(s);
9     return 0;
10 }
11
12 /* str-b.c */
13 char *sometr(void) {
14     return "Hello, world!";
15 }

```

Zadanie 6. Wydrukuj tablice **rekordów relokacji** z sekcji «.rel.text» i «.rel.data» pliku «swap.o» przy pomocy polecenia «readelf -r». Na podstawie [1, §7.7.1] wytłumacz uczestnikom zajęć składowe **rekordów relokacji** «Elf64_Rela» [2, 4-23]. Wyjaśnij jak na podstawie *tablicy rekordów relokacji* polecenie «objdump -d -r swap.o» identyfikuje w zdeasemblowanym kodzie miejsca, które konsolidator będzie musiał uzupełnić w trakcie generowania pliku wykonywalnego. Czy możliwe jest by asembler utworzył sekcję «.rel.bss»?

Zadanie 7. Na podstawie [1, §7.7.2] zreferuj **proces relokowania** referencji do symboli, dla których asembler wygenerował wpisy relokacji typu «R_X86_64_64» i «R_X86_64_32S». W trakcie tłumaczenia poniższego kodu na asembler kompilator umieścił tablicę skoków dla instrukcji wyboru switch w sekcji «.rodata». W wyniku konsolidacji pliku wykonywalnego zawierającego procedurę «relo3», została ona umieszczona pod adresem 0x1000, a tablica skoków pod 0x2000.

```

1 int relo3(int val) {
2     switch (val) {
3         case 100:
4             return val + 1;
5         case 101:
6             return val + 3;
7         case 103 ... 104:
8             return val + 5;
9         case 105:
10            return val + 7;
11        case 107:
12            return val + 11;
13        default:
14            return val + 11;
15    }
16 }

```

0000000000000000 <relo3>:		
0: 8d 47 9c	lea	-0x64(%rdi),%eax
3: 83 f8 07	cmp	\$0x7,%eax
6: 77 19	ja	21 <relo3+0x21>
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 07	lea	0x7(%rdi),%eax
20: c3	retq	
21: 8d 47 0b	lea	0xb(%rdi),%eax
24: c3	retq	

Oblicz wartości, które należy wstawić w miejsca referencji, do których odnoszą się poniższe rekordy relokacji otrzymane poleceniem «objdump -r».

```

1 RELOCATION RECORDS FOR [.text]:
2 OFFSET          TYPE          VALUE
3 000000000000000d R_X86_64_32S      .rodata
4
5
6 RELOCATION RECORDS FOR [.rodata]:
7 OFFSET          TYPE          VALUE
8 0000000000000000 R_X86_64_64      .text+0x0000000000000011
9 0000000000000008 R_X86_64_64      .text+0x0000000000000015
10 0000000000000010 R_X86_64_64      .text+0x0000000000000021
11 0000000000000018 R_X86_64_64      .text+0x0000000000000015
12 0000000000000020 R_X86_64_64      .text+0x0000000000000015
13 0000000000000028 R_X86_64_64      .text+0x0000000000000019
14 0000000000000030 R_X86_64_64      .text+0x0000000000000021
15 0000000000000038 R_X86_64_64      .text+0x000000000000001d

```

Zadanie 8. Na podstawie [1, §7.7.2] zreferuj *proces relokowania* referencji do symboli, dla których assembler wygenerował wpisy relokacji typu «R_X86_64_PC32» i «R_X86_64_PLT32». Zakładamy, że sekcja «.text» pliku «swap.o» po konsolidacji zostanie umieszczona pod adresem 0x1000, tablica «buf» pod adresem 0x2000, a *trampolina* procedury printf pod adresem 0x3000 w sekcji «.plt». Wyznacz wartości, które konsolidator wstawi w miejsca referencji symboli «printf» i «bufp0». Cemu wartość addend wynosi -4?

Wskazówka: Wszystkie typy relokacji zostały opisane w [3, §4.4.1].

Zadanie 9. Posiłkując się narzędziem **objdump**³ podaj rozmiary sekcji «.data» i «.bss» plików «bar.o» i «foo.o». Wskaż rozmiar i pozycje symboli względem początków odpowiednich sekcji.

```
1 /* bar.c */                1 /* foo.c */
2 int bar = 42;               2 long foo = 19;
3 short dead[15];            3 char code[17];
```

Na czym polega proces **częściowej konsolidacji** (ang. *partial linking*), którą można osiągnąć wywołując polecenie **ld**⁴ z opcji «-r»? Czym różni się sposób wygenerowania plików «merge-1.o» i «merge-2.o»? Na podstawie **mapy konsolidacji** (pliki «*.map») porównaj pozycje symboli i rozmiary sekcji w plikach wynikowych. Z czego wynikają różnice skoro konsolidator nie dysponuje informacjami o typach języka C?

Zadanie 10 (bonus). *Plik wykonywalny* powstały w wyniku kompilacji poniższych plików źródłowych powinien być nie dłuższy niż 1KiB. Na podstawie *nagłówka pliku ELF* wskaż w zdeasemblowanym pierwszej instrukcję, którą wykona procesor po wejściu do programu. Na podstawie *nagłówków programu* wskaż pod jaki adres wirtualny zostanie załadowany *segment* z sekcją «.text».

```
1 /* start.c */              1 /* even.c */              1 /* odd.c */
2 int is_even(long);         2 int is_odd(long n);       2 int is_even(long n);
3                             3                             3
4 void _start(void) {        4 int is_even(long n) {     4 int is_odd(long n) {
5     asm volatile(          5     if (n == 0)           5     if (n == 0)
6         "syscall"         6     return 1;            6     return 0;
7         : /* no output */  7     else                 7     else
8         : "a" (0x3c /* exit */), 8     return is_odd(n - 1); 8     return is_even(n - 1);
9         "D" (is_even(42))); 9 }                          9 }
10 }
```

Zapoznaj się z uproszczonym **skryptem konsolidatora** w pliku «main.lds». Na podstawie **dokumentacji**⁵ wyjaśnij jak skrypt kieruje procesem konsolidacji poszczególnych sekcji i tworzeniem nagłówków programu.

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”

<http://www.sco.com/developers/gabi/latest/contents.html>

[3] „System V Application Binary Interface: AMD64 Architecture Processor Supplement”

<https://raw.githubusercontent.com/hjl-tools/x86-psABI/x86-64-psABI-1.0.pdf>

³<https://sourceware.org/binutils/docs/binutils/objdump.html>

⁴<https://sourceware.org/binutils/docs/ld/index.html>

⁵<https://sourceware.org/binutils/docs/ld/Scripts.html>