Lista 8

tags: `ASK`

Zadanie 1

Wygeneruj **plik relokowalny** «`swap.o`», po czym na podstawie wydruku polecenia «`readelf -t -s`» dla każdego elementu tablicy symboli podaj:

- adres symbolu względem początku sekcji,
- typ symbolu – tj. «`local`», «`global`», «`extern`»,
- rozmiar danych, na które wskazuje symbol,
- numer i nazwę sekcji – tj. «`.text`», «`.data`», «`.bss`» – do której odnosi się symbol.

Co przechowują sekcje «`.strtab`» i «`.shstrtab`»?

```
 extern int buf[];

int *bufp0 = &buf[0];
static int *bufp1;

static void incr() {
  static int count = 0;
  count++;
}

void swap() {
  int temp;
  incr();
  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```

:::info ***Pliki relokowalne*** Pliki generowane przez kompilator, przeznaczone do późniejszego przetworzenia przez konsolidator. :::

Wynik `readelf -t -s`

```
There are 12 section headers, starting at offset 0x2e0:

Section Headers:
  [Nr] Name
       Type              Address          Offset            Link
       Size              EntSize          Info              Align
       Flags
  [ 0]
       NULL              0000000000000000  0000000000000000  0
       0000000000000000 0000000000000000  0                 0
       [0000000000000000]:
  [ 1] .text
       PROGBITS          0000000000000000  0000000000000040  0
       000000000000002e 0000000000000000  0                 1
       [0000000000000006]: ALLOC, EXEC
  [ 2] .rela.text
       RELA              0000000000000000  0000000000000200  9
       0000000000000060 0000000000000018  1                 8
       [0000000000000040]: INFO LINK
  [ 3] .data
       PROGBITS          0000000000000000  0000000000000070  0
       0000000000000008 0000000000000000  0                 8
       [0000000000000003]: WRITE, ALLOC
  [ 4] .rela.data
       RELA              0000000000000000  0000000000000260  9
       0000000000000018 0000000000000018  3                 8
       [0000000000000040]: INFO LINK
  [ 5] .bss
       NOBITS            0000000000000000  0000000000000078  0
       0000000000000004 0000000000000000  0                 4
       [0000000000000003]: WRITE, ALLOC
  [ 6] .comment
       PROGBITS          0000000000000000  0000000000000078  0
       0000000000000025 0000000000000001  0                 1
       [0000000000000030]: MERGE, STRINGS
  [ 7] .note.GNU-stack
       PROGBITS          0000000000000000  000000000000009d  0
       0000000000000000 0000000000000000  0                 1
       [0000000000000000]:
  [ 8] .note.gnu.property
       NOTE              0000000000000000  00000000000000a0  0
       0000000000000020 0000000000000000  0                 8
       [0000000000000002]: ALLOC
  [ 9] .symtab
       SYMTAB            0000000000000000  00000000000000c0  10
       0000000000000120 0000000000000018  9                 8
       [0000000000000000]:
  [10] .strtab
       STRTAB            0000000000000000  00000000000001e0  0
       0000000000000020 0000000000000000  0                 1
       [0000000000000000]:
  [11] .shstrtab
       STRTAB            0000000000000000  0000000000000278  0
       0000000000000062 0000000000000000  0                 1
       [0000000000000000]:

Symbol table '.symtab' contains 12 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 SECTION LOCAL  DEFAULT    5
     2: 0000000000000000     8 FUNC    LOCAL  DEFAULT    1 incr
     3: 0000000000000000     4 OBJECT  LOCAL  DEFAULT    5 count.1915
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    1
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    3
     6: 0000000000000000     0 SECTION LOCAL  DEFAULT    6
     7: 0000000000000000     0 SECTION LOCAL  DEFAULT    7
     8: 0000000000000000     0 SECTION LOCAL  DEFAULT    8
     9: 0000000000000008    38 FUNC    GLOBAL DEFAULT    1 swap
    10: 0000000000000000     8 OBJECT  GLOBAL DEFAULT    3 bufp0
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND buf
```

:::info

- `Value` -- adres względem początku sekcji
- `Size` -- rozmiar danych
- `Bind` -- typ symbolu
- `Ndx` -- numer sekcji w tablicy nagłówków sekcji

- `1` -- .text
- `3` -- .data
- `5` -- .bss
- `6` -- .comment
- `7` -- .note.GNU-stack
- `8` -- .note.gnu.property :::

`shstrtab` -- nazwy sekcji `strtab` -- nazwy symboli

Zadanie 2

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 «mismatch-a.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?

```c
/* mismatch-a.c */
void p2(void);

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

```c
/* mismatch-b.c */
#include <stdio.h>

char main;

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

:::info `main` typu char jest niezainicjowaną zmienną, czyli jest słabym symbolem. Z kolei `main` w pliku `mismatch-a.c` jest funkcją, czyli silnym symbolem. W takim w procesie konsolidacji słaby symbol zostanie przyporządkowany do tego samego miejsca w pamięci co funkcja `main`. :::

```
0000000000000000 <main>:
   0:   f3 0f 1e fa             endbr64
   4:   48 83 ec 08             sub    $0x8,%rsp
   8:   e8 00 00 00 00          callq  d <main+0xd>
   d:   b8 00 00 00 00          mov    $0x0,%eax
  12:   48 83 c4 08             add    $0x8,%rsp
  16:   c3                      retq
```

Widać, że jeśli spróbujemy przeczytać pierwszy bajt to dostaniemy `f3`. W dalszej części programu rozmazujemy `f3` ze znakiem i zwracamy `0xfffffff3` przez printf.

Co się stanie jak przypiszemy wartość do main w p2? Otrzymamy segmentation fault, gdyż pamięć z kodem jest read-only.

Co się stanie jak zainicjujemy main w p2? Otrzymamy błąd kompilacji, bo takie same silne symbole są niedopuszczalne.

Zadanie 3

Zadanie 3. Zapoznaj się z narzędziami do analizy **plików relokowalnych** w formacie ELF i bibliotek statycznych, tj. «objdump», «readelf» i «ar»; a następnie odpowiedz na następujące pytania:

1. Ile plików zawierają biblioteki «libc.a» i «libm.a» (katalog «/usr/lib/x86_64-linux-gnu»)?
2. Czy wywołanie kompilatora z opcją «-Og» generuje inny kod wykonywalny niż «-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 podanych poleceń.

:::info

- `ar` -- zarządzanie plikami archiwalnymi
- `objdump` -- analiza plików obiektowych
- `readelf` -- analiza plików ELF
- `pliki relokowalne` -- pliki generowane przez kompilator, przeznaczone do późniejszego przetworzenia przez konsolidator. :::

1. Ile plików zawierają biblioteki `libc.a` i `libm.a`?

Wystarczy wykonać `ar t` dla bilbioteki `libc.a` oraz zgrupowanych bibliotek w skrypcie linkera `libm.a` i zliczyć wiersze.

2. Czy wywołanie kompilatora z opcją `-Og` generuje inny kod wykonywalny niż `-Og -g`?

:::info Flaga `-Og` włącza te same optymalizacje co flaga `-O1` z wyjątkiem tych, które mogą mieć wpływ na komfort debugowania. Flaga `-g` dostarcza dodatkowych informacji przydatnych przy debugowaniu. ::: Można sprawdzić przy użyciu `readelf -S` (drukowanie sekcji), że rzeczywiście pojawiają się nowe sekcje z informacjami do debugowania.

```
< There are 31 section headers, starting at offset 0x3970:
---
> There are 36 section headers, starting at offset 0x4268:
62,64c62,74
<   [28] .symtab           SYMTAB           0000000000000000  00003038
<        0000000000000618  0000000000000018          29    46     8
<   [29] .strtab           STRTAB           0000000000000000  00003650
---
>   [28] .debug_aranges    PROGBITS         0000000000000000  00003034
>        0000000000000030  0000000000000000           0     0     1
>   [29] .debug_info       PROGBITS         0000000000000000  00003064
>        000000000000034d  0000000000000000           0     0     1
>   [30] .debug_abbrev     PROGBITS         0000000000000000  000033b1
>        0000000000000107  0000000000000000           0     0     1
>   [31] .debug_line       PROGBITS         0000000000000000  000034b8
>        0000000000000112  0000000000000000           0     0     1
>   [32] .debug_str        PROGBITS         0000000000000000  000035ca
>        00000000000002a9  0000000000000001  MS       0     0     1
>   [33] .symtab           SYMTAB           0000000000000000  00003878
>        0000000000000690  0000000000000018          34    51     8
>   [34] .strtab           STRTAB           0000000000000000  00003f08
66,67c76,77
<   [30] .shstrtab         STRTAB           0000000000000000  00003852
<        000000000000011a  0000000000000000           0     0     1
---
>   [35] .shstrtab         STRTAB           0000000000000000  0000410a
>        000000000000015a  0000000000000000           0     0     1
```

3. Z jakich bibliotek współdzielonych korzysta interpreter języka python?

`readelf -d /usr/bin/python3` – sekcje dynamiczne

```
Dynamic section at offset 0x4f3dc0 contains 30 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x0000000000000001 (NEEDED)             Shared library: [libpthread.so.0]
 0x0000000000000001 (NEEDED)             Shared library: [libdl.so.2]
 0x0000000000000001 (NEEDED)             Shared library: [libutil.so.1]
 0x0000000000000001 (NEEDED)             Shared library: [libm.so.6]
 0x0000000000000001 (NEEDED)             Shared library: [libexpat.so.1]
 0x0000000000000001 (NEEDED)             Shared library: [libz.so.1]
```

Zadanie 4

Po uruchomieniu program kończy się 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 ten program tak, by się poprawnie zakończył. Gdzie został umieszczony ciąg znaków po poprawce? Nie wolno modyfikować sygnatury procedury «somestr» i pliku «str-a.c», ani korzystać z dodatkowych procedur.

```c
/* str-b.c */
char *somestr(void) {
  return "Hello, world!";
}
```

```c
/* str-a.c */
#include <stdio.h>

char *somestr(void);

int main(void) {
  char *s = somestr();
  s[5] = '\0';
  puts(s);
  return 0;
}
```

Przyczyną *Segmentation fault* jest próba zapisu do `s[5]`. Dzieje się tak ponieważ napis "Hello, world!" został umieszczony w sekcji tylko do odczytu (.rodata). Wystarczy zdekompilować plik `str` i sprawdzić zawartość .rodata – `readelf -p .rodata`.

Naprawa

```
char *somestr(void) {
  static char readable[] = "Hello, world!";
  return readable;
}
```

Zadanie 5

Zadanie 5. Posiłkując się narzędziem «objdump» podaj rozmiary sekcji «.data» i «.bss» plików «bar.o» i «foo.o». Wskaż rozmiar i pozycje symboli względem początków odpowiednich sekcji. Wyjaśnij znaczenie opcji «-fno-common» przekazywanej do kompilatora.
Na czym polega **częściowa konsolidacja** z użyciem opcji «-r» do polecenia «ld»? Czym różni się sposób wygenerowania plików «merge-1.o» i «merge-2.o»? Na podstawie **mapy konsolidacji** porównaj pozycje symboli i rozmiary sekcji w plikach wynikowych. Z czego wynikają różnice skoro konsolidator nie dysponuje informacjami o typach języka C?

```
/* foo.c */
long foo = 19;
char code[17];
```

```
/* bar.c */
int bar = 42;
short dead[15];
```

objdump -h foo.o

```
start address 0x0000000000000000

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text         00000000  0000000000000000  0000000000000000  00000040  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000008  0000000000000000  0000000000000000  00000040  2**3
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000011  0000000000000000  0000000000000000  00000050  2**4
                  ALLOC
  3 .comment      00000025  0000000000000000  0000000000000000  00000050  2**0
                  CONTENTS, READONLY
  4 .note.GNU-stack 00000000  0000000000000000  0000000000000000  00000075  2**0
                  CONTENTS, READONLY
  5 .note.gnu.property 00000020  0000000000000000  0000000000000000  00000078  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
```

readelf -s foo.o

```
Symbol table '.symtab' contains 9 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 SECTION LOCAL  DEFAULT    1
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    2
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5
     6: 0000000000000000     0 SECTION LOCAL  DEFAULT    6
     7: 0000000000000000    17 OBJECT  GLOBAL DEFAULT    3 code
     8: 0000000000000000     8 OBJECT  GLOBAL DEFAULT    2 foo
```

objdump -h bar.o

```
start address 0x0000000000000000

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text         00000000  0000000000000000  0000000000000000  00000040  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  0000000000000000  0000000000000000  00000040  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          0000001e  0000000000000000  0000000000000000  00000050  2**4
                  ALLOC
  3 .comment      00000025  0000000000000000  0000000000000000  00000050  2**0
                  CONTENTS, READONLY
  4 .note.GNU-stack 00000000  0000000000000000  0000000000000000  00000075  2**0
                  CONTENTS, READONLY
  5 .note.gnu.property 00000020  0000000000000000  0000000000000000  00000078  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
```

`readelf -s bar.o`

```
Symbol table '.symtab' contains 9 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 SECTION LOCAL  DEFAULT    1
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    2
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5
     6: 0000000000000000     0 SECTION LOCAL  DEFAULT    6
     7: 0000000000000000    30 OBJECT  GLOBAL DEFAULT    3 dead
     8: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    2 bar
```

Rozmiary sekcji

	.data	.bss
foo.o	8	17
bar.o	4	30

Rozmiary i pozycje symboli

Symbol	Rozmiar
code	17
foo	8
dead	30
bar	4
Symbole w obu tabelach mają pozycję względną równą 0.	

-fno-common - powoduje, że kompilator wstawia niezainicjowane zmienne do .bss zamiast sekcji COMMON.

Częściowa konsolidacja - polega na łączeniu plików relokowanych w jeden przy użyciu `ld -r`.

Mapa konsolidacji - plik zawierający informacje o tablicy symboli po skonsolidowaniu plików relokowalnych. Tworzona przy podaniu flagi `-M` do konsolidatora.

Makefile

```
merge-1.map: foo.o bar.o
    $(LD) -M=$@ -r -o merge-1.o $^

merge-2.map: bar.o foo.o
    $(LD) -M=$@ -r -o merge-2.o $^
```

Jak widać `merge-1.map` powstaje poprzez zamianę kolejności plików relokowalnych w `merge-2.map`.

`merge-1.map`

```
 ...

.data           0x0000000000000000        0xc
 *(.data)
 .data          0x0000000000000000        0x8 foo.o
                0x0000000000000000                foo
 .data          0x0000000000000008        0x4 bar.o
                0x0000000000000008                bar

.data1
 *(.data1)

.bss            0x0000000000000000       0x3e
 *(.bss)
 .bss           0x0000000000000000       0x11 foo.o
                0x0000000000000000                code
 *fill*         0x0000000000000011        0xf
 .bss           0x0000000000000020       0x1e bar.o
                0x0000000000000020                dead

...
```

merge-2.map

```
 ...

.data           0x0000000000000000       0x10
 *(.data)
 .data          0x0000000000000000        0x4 bar.o
                0x0000000000000000                bar
 *fill*         0x0000000000000004        0x4
 .data          0x0000000000000008        0x8 foo.o
                0x0000000000000008                foo

.data1
 *(.data1)

.bss            0x0000000000000000       0x31
 *(.bss)
 .bss           0x0000000000000000       0x1e bar.o
                0x0000000000000000                dead
 *fill*         0x000000000000001e        0x2
 .bss           0x0000000000000020       0x11 foo.o
                0x0000000000000020                code

...
```