Systemy komputerowe

Lista zadań nr 5

Na zajęcia zdalne 6 – 9 kwietnia 2020

Zadanie 1. Zdefiniuj pojęcie **wyrównania danych** w pamięci (ang. *data alignment*). Dlaczego dane typów prostych (np. `short`, `int`, `double`) powinny być w pamięci odpowiednio wyrównane? Jaki związek z wyrównywaniem danych mają **wypełnienia** (ang. *padding*) w danych typu strukturalnego. Odpowiadając na powyższe pytanie podaj przykład struktury, której rozmiar w bajtach (wyliczony przez operator `sizeof`) jest większy niż suma rozmiaru pól składowych. Czemu służy **wypełnienie wewnętrzne** (ang. *internal padding*) a czemu **wypełnienie zewnętrzne** (ang. *external padding*)?

Zadanie 2. Dana jest funkcja o sygnaturze «`int16_t you(int8_t index)`» i fragmencie kodu podanym poniżej. Funkcja ta została skompilowana z flagą `-O0`, a jej kod asemblerowy również jest podany. Nieznana jest natomiast funkcja «`int16_t set_secret(void)`». Jaki argument należy podać wywołując `you`, by odkryć wartość sekretu?

```
1  int16_t you(int8_t index) {
2  struct {
3     int16_t tbl[5];
4     int8_t tmp;
5     int16_t secret;
6  } str;
7
8  str.secret = set_secret();
9  ...
10 return str.tbl[index];
11 }
```

```
4  you:    pushq   %rbp
5          movq    %rsp, %rbp
6          subq    $24, %rsp
7          movl    %edi, %eax
8          movb    %al, -20(%rbp)
9          call    set_secret
10         movw    %ax, -2(%rbp)
11         ...
12         movsbl  -20(%rbp), %eax
13         cltq
14         movzwl  -14(%rbp,%rax,2), %eax
15         leave
16         ret
```

Wskazówka: Instrukcja cltq rozszerza rejestr %eax do %rax zachowując znak. Pamiętaj, że zadeklarowane zmienne muszą być odpowiednio wyrównane.

Zadanie 3 (2pkty). Przeczytaj poniższy kod w języku `C` i odpowiadający mu kod w asemblerze, a następnie wywnioskuj jakie są wartości stałych «`A`» i «`B`».

```c
typedef struct {
  int32_t x[A][B];
  int64_t y;
} str1;

typedef struct {
  int8_t array[B];
  int32_t t;
  int16_t s[A];
  int64_t u;
} str2;

void set_val(str1 *p, str2 *q) {
  int64_t v1 = q->t;
  int64_t v2 = q->u;
  p->y = v1 + v2;
}
```

```
set_val:
  movslq 8(%rsi),%rax
  addq   32(%rsi),%rax
  movq   %rax,184(%rdi)
  ret
```

Wskazówka: Deklaracja int32_t x[A][B] powoduje, że x będzie A-elementową tablicą wartości typu int32_t [B]. Pamiętaj o wyrównaniu pól w strukturach.

Zadanie 4 (2pkty). Przeczytaj poniższy kod w języku C i odpowiadający mu kod w asemblerze, a następnie wywnioskuj jaka jest wartość stałej «CNT» i jak wygląda definicja struktury «a_struct».

```c
typedef struct {
  int32_t first;
  a_struct a[CNT];
  int32_t last;
} b_struct;

void test (int64_t i, b_struct *bp) {
  int32_t n = bp->first + bp->last;
  a_struct *ap = &bp->a[i];
  ap->x[ap->idx] = n;
}
```

```
test:
movl   0x120(%rsi),%ecx
addl   (%rsi),%ecx
leaq   (%rdi,%rdi,4),%rax
leaq   (%rsi,%rax,8),%rax
movq   0x8(%rax),%rdx
movslq %ecx,%rcx
movq   %rcx,0x10(%rax,%rdx,8)
retq
```

Wiadomo, że jedynymi polami w strukturze «a_struct» są «idx» oraz «x», i że obydwa te pola są typu numerycznego ze znakiem.

Zadanie 5. Zdefiniuj semantykę operatora «?:» z języka C. Jakie zastosowanie ma poniższa funkcja.

```c
int32_t cread(int32_t *xp) {
  return (xp ? *xp : 0);
}
```

Używając serwisu godbolt.org (kompilator x86-64 gcc 8.2) sprawdź, czy istnieje taki poziom optymalizacji (-O0, -O1, -O2 lub -O3), że wygenerowany dla cread kod asemblerowy nie używa instrukcji skoku. Jeśli nie, to zmodyfikuj funkcję cread tak, by jej tłumaczenie na asembler spełniało powyższy warunek.

Wskazówka: Dążysz do wygenerowania kodu używającego instrukcji cmov. Końcowej instrukcji ret nie uważamy w tym zadaniu za instrukcję skoku.

Zadanie 6. W języku C struktury mogą być zarówno argumentami funkcji, jak i wartościami zwracanymi przez funkcje[1]. Za pomocą serwisu godbolt.org zapoznaj się z tłumaczeniem do asemblera funkcji przyjmujących pojedynczy argument każdego z poniższych typów strukturalnych. Następnie zapoznaj się z tłumaczeniem funkcji zwracających wartość każdego z tych typów. Jakie reguły dostrzegasz?

[1] Konwencja przekazywania/zwracania argumentów takich typów jest częścią ABI: https://github.com/hjl-tools/x86-psABI/wiki/x86-64-psABI-1.0.pdf, strony 19–25

```c
struct first {
int32_t val1;
};
struct second {
int32_t val2;
int32_t val1;
};
struct third {
int32_t val3;
int32_t val2;
int32_t val1;
};
```

```c
struct fourth {
int32_t val4;
int32_t val3;
int32_t val2;
int32_t val1;
};
struct fifth {
int16_t val6;
int16_t val5;
int32_t val3;
int32_t val2;
int32_t val1;
};
```

```c
struct sixth {
int32_t val7;
int32_t val4;
int32_t val3;
int32_t val2;
int32_t val1;
};
struct seventh {
int32_t val8[10000];
int32_t val4;
int32_t val3;
int32_t val2;
int32_t val1;
};
```

Czy przekazywanie/zwracanie dużych struktur funkcjom jest dobrym pomysłem?

Zadanie 7. W poniższej funkcji zmienna `field` jest polem bitowym typu `int32_t` o rozmiarze 4. Jaką wartość wypisze ta funkcja na ekranie i dlaczego? Co się stanie, gdy zmienimy typ pola `field` na `uint32_t`? Na obydwa te pytania odpowiedz analizując tłumaczenia tej funkcji na język asemblera.

```c
void set_field(void) {
struct {
  int32_t field : 4;
} s;
  s.field = 10;
  printf("Field value is: %d\n", s.field);
}
```

Wskazówka: Użyj poziomu optymalizacji «`-O0`». Dla wyższych poziomów optymalizacji kompilator zauważy, że deklaracja zmiennej «s» jest niepotrzebna i obliczy wartość wypisywaną przez «`printf`» podczas kompilacji.

Zadanie 8. Język `C` dopuszcza deklaracje tablic wielowymiarowych z opuszczonym rozmiarem pierwszego wymiaru. Taka deklaracja może wystąpić w nagłówku funkcji, np. «`void process(int32_t A[][77], size_t len)`». Nie można natomiast opuszczać rozmiarów innych wymiarów, np. «`void bad(int32_t A[77][], size_t len)`» nie jest poprawną deklaracją. Wyjaśnij, dlaczego tak jest odwołując się do sposobu, w jaki kompilator tłumaczy odwołania do tablic z `C` na asembler.