Lista 6

tags: ASK

Zadanie 1

Zadanie 1. Poniższy wydruk otrzymano w wyniku deasemblacji rekurencyjnej procedury zadeklarowanej następująco: «`long puzzle(long n, long *p)`». Zapisz w języku `C` kod odpowiadający tej procedurze. Następnie opisz zawartość jej **rekordu aktywacji** (ang. *stack frame*). Wskaż **rejestry zapisane przez funkcję wołaną** (ang. *callee-saved registers*), zmienne lokalne i adres powrotu.

```
puzzle:
        push %rbp
        xorl %eax, %eax          //rax = 0 (:= result)
        mov %rsi, %rbp           //rbp = *p
        push %rbx
        mov %rdi, %rbx           //rbx = n
        sub $24, %rsp            //rozszerzamy stos o 3 bajty
        test %rdi, %rdi          //jeśli n <= 0
        jle .L1                  //skok do L1
        lea 8(%rsp), %rsi        //rsi jest wskaznikiem na zmienna t
        lea (%rdi,%rdi), %rdi    //n *= 2
        call puzzle              //wywołujemy puzzle(2n, &t)
        add 8(%rsp), %rax        //result += t
        add %rax, %rbx           //n += t
.L1:    mov %rbx, (%rbp)         //*p = n
        add $24, %rsp            //zwalniamy 3 bajty ze stosu
        pop %rbx                 //przywracamy rbx
        pop %rbp                 //przywracamy rbp
        ret
```

```
puzzle(long n, long *p) {
    long result = 0;
    if (n > 0) {
        long t;
        result = puzzle(2*n, &t);
        result += t;
        n += result;
    }
    *p = n;
    return result;
}
```

rekord aktywacji
adres powrotu
%rbp
%rbx
?
miejsce wskazywane przez %rsi
?

Rejestry zapisane przez funkcję wołaną: %rbp, %rbx Zmienne lokalne: 4, 5, 6

Zadanie 2

Zadanie 2. Poniżej zamieszczono kod procedury o sygnaturze «`struct T puzzle8(long *a, long n)`». Na jego podstawie podaj definicję typu «`struct T`». Przetłumacz tę procedurę na język C, po czym jednym zdaniem powiedz co ona robi. Gdyby sygnatura procedury nie była wcześniej znana to jaką należałoby wywnioskować z poniższego kodu?

MEMORY This class consists of types that will be passed and returned in memory via the stack.

Returning of Values The returning of values is done according to the following algorithm:

1. Classify the return type with the classification algorithm.
2. If the type has class MEMORY, then the caller provides space for the return value and passes the address of this storage in `%rdi` as if it were the first argument to the function. In effect, this address becomes a "hidden" first argument. This storage must not overlap any data visible to the callee through other names than this argument.

   On return `%rax` will contain the address that has been passed in by the caller in `%rdi`.
3. If the class is INTEGER, the next available register of the sequence `%rax`, `%rdx` is used.

Jako, że wartość zwracana przez puzzle8 nie mieści się w dwóch rejestrach to będzie ona zwracana przez pamięć. Funkcja wywołująca rezerwuje miejsce w pamięci i przekazuje wskaźnik na to miejsce jako pierwszy argument funkcji wywołanej.

```
puzzle8:
        movq %rdx, %r11             # r11 = n
        xorl %r10d, %r10d           # zerujemy r10 (:= i)
        xorl %eax, %eax             # zerujemy rax (:= sum)
        movq $LONG_MIN, %r8         # r8 = long_min (:= max)
        movq $LONG_MAX, %r9         # r9 = long_max (:= min)
.L2:    cmpq %r11, %r10             # jeśli i >= n
        jge .L5                     # skaczemy do L5
        movq (%rsi,%r10,8), %rcx    # pobieramy i-ty element z a
        cmpq %rcx, %r9              # porównujemy z min
        cmovg %rcx, %r9             # jeśli min większy to min = a[i]
        cmpq %rcx, %r8              # porównujemy z max
        cmovl %rcx, %r8             # jeśli max mniejszy to max = a[i]
        addq %rcx, %rax             # sum+=a[i]
        incq %r10                   # i++
        jmp .L2                     # skaczemy na początek pętli
.L5:    cqto                        # sign-extend RAX -> RDX:RAX

        #wypełnianie structa
        movq %r9, (%rdi)            # struct.min = min
        idivq %r11                  # dzielimy RDX:RAX(sum) przez n
        movq %r8, 8(%rdi)           # struct.max = max
        movq %rax, 16(%rdi)         # struct.average = sum / n
        movq %rdi, %rax             # return struct*
        ret
```

```
struct T {
    long min;
    long max;
    long average;
};

struct T puzzle8(long *a, long n){
    long sum = 0;
    long max = LONG_MIN;
    long min = LONG_MAX;

    for(int i = 0; i < n; i++){
        long elem = a[i];
        if(elem > max) {max = elem};
        if(elem < min) {min = elem};
        sum += elem;
    }
    struct T result = { min, max, sum / n }
    return result;
}
```

Procedura znajduje element największy, najmniejszy oraz średnią elementów tablicy a. Prawdopodobna sygnatura to `struct T* puzzle(struct T *s, long *a, long n)`

Zadanie 4

Zadanie 4. Poniżej widnieje kod wzajemnie rekurencyjnych procedur «M» i «F» typu «`long (*)(long)`». Programista, który je napisał, nie pamiętał wszystkich zasad **konwecji wołania procedur**. Wskaż co najmniej dwa różne problemy w poniższym kodzie i napraw je!

```
M:
    pushq %rdi              //odkładamy argument n na stos
    testq %rdi, %rdi
    je .L2
    leaq -1(%rdi), %rdi
    call M
    movq %rax, %rdi
    call F
    movq (%rsp), %rdi
    subq %rax, %rdi
    .L2: movq %rdi, %rax
    ret
F:
    testq %rdi, %rdi
    je .L3
    movq %rdi, %r12
    leaq -1(%rdi), %rdi
    call F
    movq %rax, %rdi
    call M
    subq %rax, %r12
    movq %r12, %rax
    ret
L3: movl $1, %eax
    ret
```

Procedura M ewidentnie nie sprząta po sobie stosu, przez co adres powrotu funkcji będzie nieprawidłowy. Procedurę można naprawić następująco:

```
M:      pushq %rdi
        testq %rdi, %rdi
        je .L2
        leaq -1(%rdi), %rdi
        call M
        movq %rax, %rdi
        call F
        movq (%rsp), %rdi
        subq %rax, %rdi
.L2:    movq %rdi, %rax
        popq %rdi
        ret
```

Jak wiadomo rejestr %r12 jest *calle-saved*, a więc musi być odpowiednio zabezpieczony przez funkcję wywoływaną tj. należy zachować wartość tego rejestru na stosie i następnie ją odtworzyć, gdy przestaniemy z niej korzystać (czyli z powrotem przypisać do %r12). Procedurę można naprawić następująco:

```
F:      testq %rdi, %rdi
        je .L3
        pushq %r12
        movq %rdi, %r12
        leaq -1(%rdi), %rdi
        call F
        movq %rax, %rdi
        call M
        subq %rax, %r12
        movq %r12, %rax
        popq %r12
        ret
L3:     movl $1, %eax
        ret
```

Zadanie 5

Zadanie 5. Skompiluj poniższy kod źródłowy kompilatorem gcc z opcjami «`-Og -fomit-frame-pointer -fno-stack-protector`» i wykonaj deasemblację **jednostki translacji** przy użyciu programu «`objdump`». Wytłumacz co robi procedura `alloca(3)`, a następnie wskaż w kodzie maszynowym instrukcje realizujące przydział i zwalnianie pamięci. Wyjaśnij co robi robi instrukcja «`leave`».

Pierwotny kod

```
#include <alloca.h>

long aframe(long n, long idx, long *q) {
    long i;
    long **p = alloca(n * sizeof(long *));
    p[n-1] = &i;
    for (i = 0; i < n; i++) {
        p[i] = q;
    }
    return *p[idx];
}
```

Jednostka translacji - pliki źródłowe wraz z plikami dołączonymi przez `#include`.

alloca() - Przyjmuje za argument rozmiar pamięci, którą następnie alokuje na stosie. Pamięć jest automatycznie zwalniana po wykonaniu funkcji wywołującej. Zwraca wskaźnik na początek zarezerwowanego miejsca.

Deasemblacja

```
n   -> RDI
idx -> RSI
*q  -> RDX
```

```
00000000000005fa <aframe>:


#alloca
 5fa:   push   %rbp                     # zachowujemy wartość rbp
 5fb:   mov    %rsp,%rbp                # rsp przejmuje rolę rbp
 5fe:   sub    $0x10,%rsp               # rezerwujemy 2 bajty na stosie
 602:   lea    0x0(,%rdi,8),%r9         # r9 = n * sizeof(long*)
 60a:   lea    0x1e(%r9),%rax           # obliczanie rezerwowanej
 60e:   and    $0xfffffffffffffff0,%rax # liczby bajtów
 612:   sub    %rax,%rsp                # rezerwacja pamięci

 615:   lea    0xf(%rsp),%r8            # obliczanie wskaźnika na
 61a:   and    $0xfffffffffffffff0,%r8  # zarezerwowane miejsce

 61e:   mov    %r8,%rcx                 # wynik(wskaźnik) zapisujemy w p


#reszta kodu
 621:   lea    -0x8(%rbp),%rax          # &i
 625:   mov    %rax,-0x8(%r8,%r9,1)     # p[n-1] = &i
 62a:   mov    $0x0,%eax                # i = 0
 62f:   jmp    639 <aframe+0x3f>        # skocz do warunku w 639
 631:   mov    %rdx,(%rcx,%rax,8)       # p[i] = q
 635:   add    $0x1,%rax                # i++
 639:   cmp    %rdi,%rax                # i < n
 63c:   jl     631 <aframe+0x37>        # pętlenie się
 63e:   mov    (%r8,%rsi,8),%rax        # p[idx]
 642:   mov    (%rax),%rax              # *p[idx]

 645:   leaveq                          # zwolnienie miejsca ze stosu
 646:   retq
```

Procedura $leave$

```
movq %rbp, %rsp
popq %rbp
```

Używana w połączeniu z $enter$

```
push %rbp
movq  %rsp, %rbp
```

Zadanie 6

Zadanie 6. Poniżej widnieje kod procedury o sygnaturze «`long puzzle5(void)`». Podaj rozmiar i składowe rekordu aktywacji procedury «`puzzle5`». Procedura «`readlong`», która wczytuje ze standardowego wejścia liczbę całkowitą, została zdefiniowana w innej jednostce translacji. Jaka jest jej sygnatura? Przetłumacz procedurę «`puzzle5`» na język `C` i wytłumacz jednym zdaniem co ona robi.

```
puzzle5:
    subq $24, %rsp        //rezerwujemy 3 bajty pamięci
    movq %rsp, %rdi       //rdi wskazuje na zmienną x w pamięci
    call readlong         //readlong(x?)
    leaq 8(%rsp), %rdi    //rdi wskazuje na kolejną zmienną y w pamięci
    call readlong         //readlong(y?)
    movq (%rsp), %rax     //rax = x
    cqto                  //rozmazujemy (znakiem) rax na rdx:rax
    idivq 8(%rsp)         //rax = x/y, rdx = x%y
    xorl %eax, %eax       //rax = 0
    testq %rdx, %rdx      //jeśli rdx == 0
    sete %al              //to rax = 1 wpp. rax = 0
    addq $24, %rsp        //sprzątamy stos
    ret
```

rekord aktywacji
adres powrotu
?
y
x

Rozmiar rekordu to 32 bajty Sygnatura readlong to najprawdopodobniej void readlong(long *x) Procedura puzzle sprawdza, czy pierwsza wczytana liczba jest podzielna przez drugą

```
long puzzle5(){
    long x,y;

    readlong(&x);
    readlong(&y);

    return x % y == 0;
}
```