

Лабораторная работа 2

Калашников Михаил, Б03-205

Стек и локальные переменные

1. Сгенерируем листинг для функции, принимающей два аргумента и возвращающей int. Как можно заметить аргументы через регистры esi и edi передаются в стек функции, а возвращаемое значение передается через регистр eax. С флагом -m32 все происходит немного иначе. Аргументы записываются в стек main'a и функция тянется за ними в чужой (mainовский) стек. Возвращаемое значение передается так же.

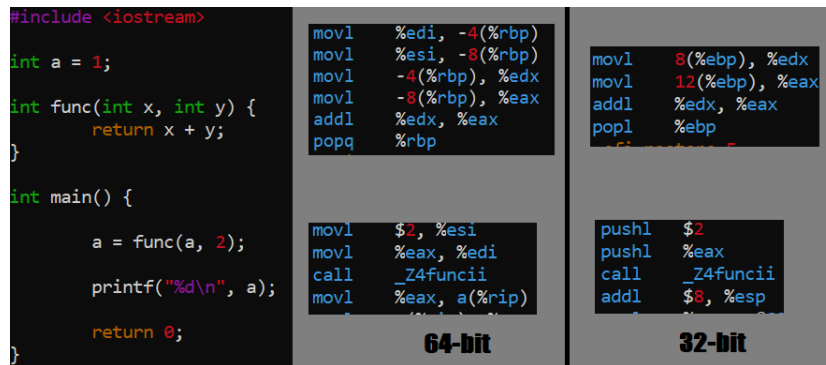


Рис. 1: К пункту 1

2. Локальные переменные и статические массивы работают аналогично. Все значения поочередно добавляются в стек. Перед этим из rsp вычитается нужное количество байтов. При создании динамического массива, в стек сохраняется указатель на него. Обращение к элементам происходит по адресу, который вычисляется несложным образом.



Рис. 2: К пункту 2

3. При создании экземпляра структуры в стек закидываются все ее поля поочередно. Если же создать экземпляр как глобальную переменную, то обращение к полям будет происходить через адрес глобального экземпляра (как на рисунке). Если полем является статический массив, то все работает аналогично. При обращении к нужному элементу массива вычисляется его адрес относительно начала структуры. Если передать структуру как аргумент, то она запишется в мейновский стек и функция будет обращаться к чужому стеку. При возвращении структуры она вроде как записывается в стек и читается оттуда `main`ом. В 32-битной системе все **гораздо** нагляднее.

<pre>#include <iostream> struct abc { int a; int b[5]; }; abc FUNC(abc t) { t.b[1]++; return t; } abc A = {34, 45}; int main() { A = FUNC(A); return 0; }</pre>	<pre>movq %rdi, -8(%rbp) movl 24(%rbp), %eax addl \$1, %eax movl %eax, 24(%rbp) movq -8(%rbp), %rcx movq 16(%rbp), %rax movq 24(%rbp), %rdx movq %rax, (%rcx) movq %rdx, 8(%rcx) movq 32(%rbp), %rax movq %rax, 16(%rcx) movq -8(%rbp), %rax popq %rbp</pre>	<div>main64-bit</div> <pre>pushq 16+A(%rip) pushq 8+A(%rip) pushq A(%rip) movq %rax, %rdi call _Z4FUNC3abc addq \$24, %rsp movq -48(%rbp), %rax movq -40(%rbp), %rdx movq %rax, A(%rip) movq %rdx, 8+A(%rip) movq -32(%rbp), %rax movq %rax, 16+A(%rip)</pre>
<div>32-bitfunction</div> <pre>movl 20(%ebp), %eax addl \$1, %eax movl %eax, 20(%ebp) movl 8(%ebp), %eax movl 12(%ebp), %edx movl %edx, (%eax) movl 16(%ebp), %edx movl %edx, 4(%eax) movl 20(%ebp), %edx movl %edx, 8(%eax) movl 24(%ebp), %edx movl %edx, 12(%eax) movl 28(%ebp), %edx movl %edx, 16(%eax) movl 32(%ebp), %edx movl %edx, 20(%eax) movl 8(%ebp), %eax popl %ebp</pre>	<pre>pushl 20+A@GOTOFF(%ebx) pushl 16+A@GOTOFF(%ebx) pushl 12+A@GOTOFF(%ebx) pushl 8+A@GOTOFF(%ebx) pushl 4+A@GOTOFF(%ebx) pushl A@GOTOFF(%ebx) pushl %eax call _Z4FUNC3abc addl \$24, %esp movl -56(%ebp), %eax movl %eax, A@GOTOFF(%ebx) movl -52(%ebp), %eax movl %eax, 4+A@GOTOFF(%ebx) movl -48(%ebp), %eax movl %eax, 8+A@GOTOFF(%ebx) movl -44(%ebp), %eax movl %eax, 12+A@GOTOFF(%ebx) movl -40(%ebp), %eax movl %eax, 16+A@GOTOFF(%ebx) movl -36(%ebp), %eax movl %eax, 20+A@GOTOFF(%ebx)</pre>	<div>main</div>

Рис. 3: К пункту 3

- При передаче структуры по указателю не происходит записывание всей структуры в стек main'a, так как функция сразу знает где структуру искать. В стек записывается только адрес ее начала. Разницу в листинге между передачей по указателю и передачей по ссылке я не нашел. Равны даже размеры файлов с листингами, так что подозреваю что и листинги не полностью идентичны.

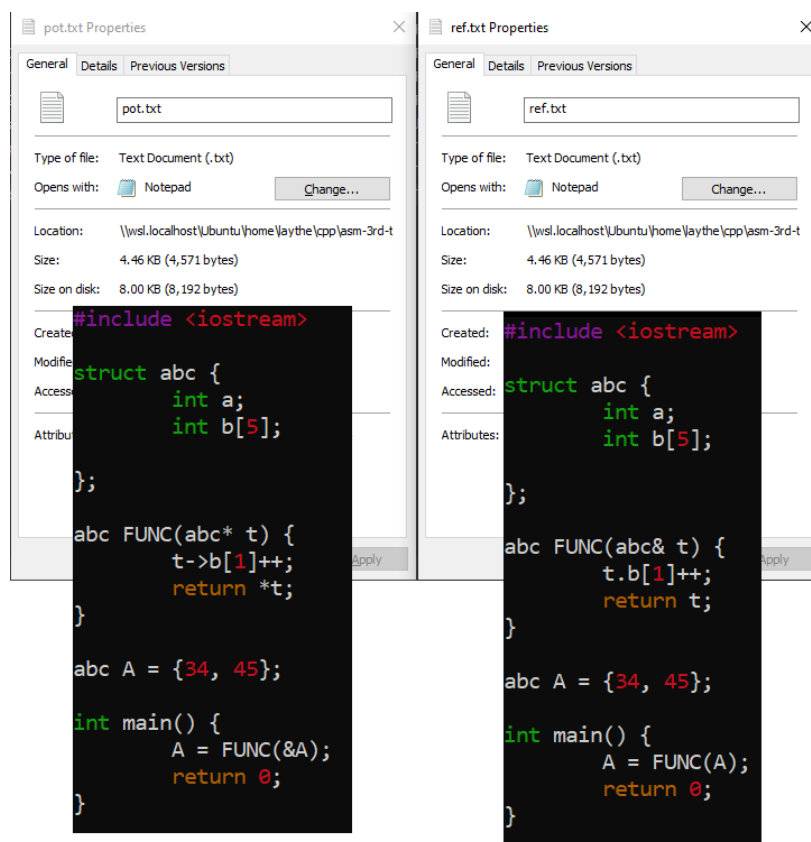


Рис. 4: К пункту 4

5. С жирными структурами работа происходит очень аккуратно. В стек добавляются только необходимые значения. Такой режим работы начинается когда размер структуры превышает 64 байта.
6. Создадим бесконечно-рекурсивную функцию. С помощью ассемблерных вставок реализуем вывод глубины рекурсии в терминал. При такой реализации функция не будет забирать у стека место под работу с printf. То есть на каждом шаге рекурсии будут выполняться два pushq (один от call, второй явный при начале работы функции), а у стека будет съедаться соответственно 16 байт. Зная после какой итерации программа сегфолтнулась, можно найти размер стека:

$$S = 523664 * 16 \text{ байт} = 8378624 \text{ байт} \approx 8 \text{ мегабайт}$$

Это значение совпадает с ulimit -s, которое тоже равно 8 мегабайтам. :)

```

#include <iostream>
void rec() {
    asm(
        "addl $1, %r15d\n\t"
        "movl %r15d, %esi\n\t"
        "leaq .LC0(%rip), %rax\n\t"
        "movq %rax, %rdi\n\t"
        "movl $0, %eax\n\t"
        "call printf@PLT"
    );
    rec();
}

int main() {
    asm("movl $0, %r15d");
    printf("%d\n", 0);
    rec();

    return 0;
}

```

```

523660
523661
523662
523663
523664
523665
Segmentation fault

```

```

_Z3recv:
.LFB1731:
.cfi_startproc
endbr64
pushq   %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq    %rsp, %rbp
.cfi_def_cfa_register 6

#APP
# 3 "break.c" 1
    addl $1, %r15d
    movl %r15d, %esi
    leaq .LC0(%rip), %rax
    movq %rax, %rdi
    movl $0, %eax
    call printf@PLT

# 0 "" 2
#NO_APP
    call    _Z3recv
    nop
    popq    %rbp

```

C, 64-bit

Рис. 5: К пункту 6