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 maina и функция тянется за ними в чужой (mainовский) стек. Возвращаемое значение передается так же.

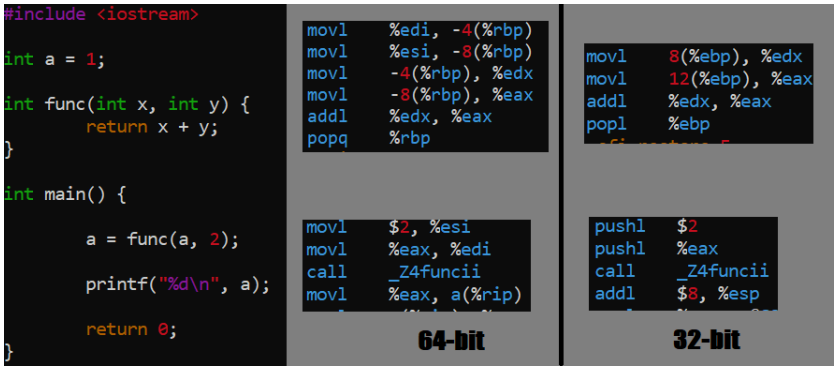

Рис. 1: К пункту 1

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 эдементам 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 mainом. В 32-битной системе все **гораздо** нагляднее.

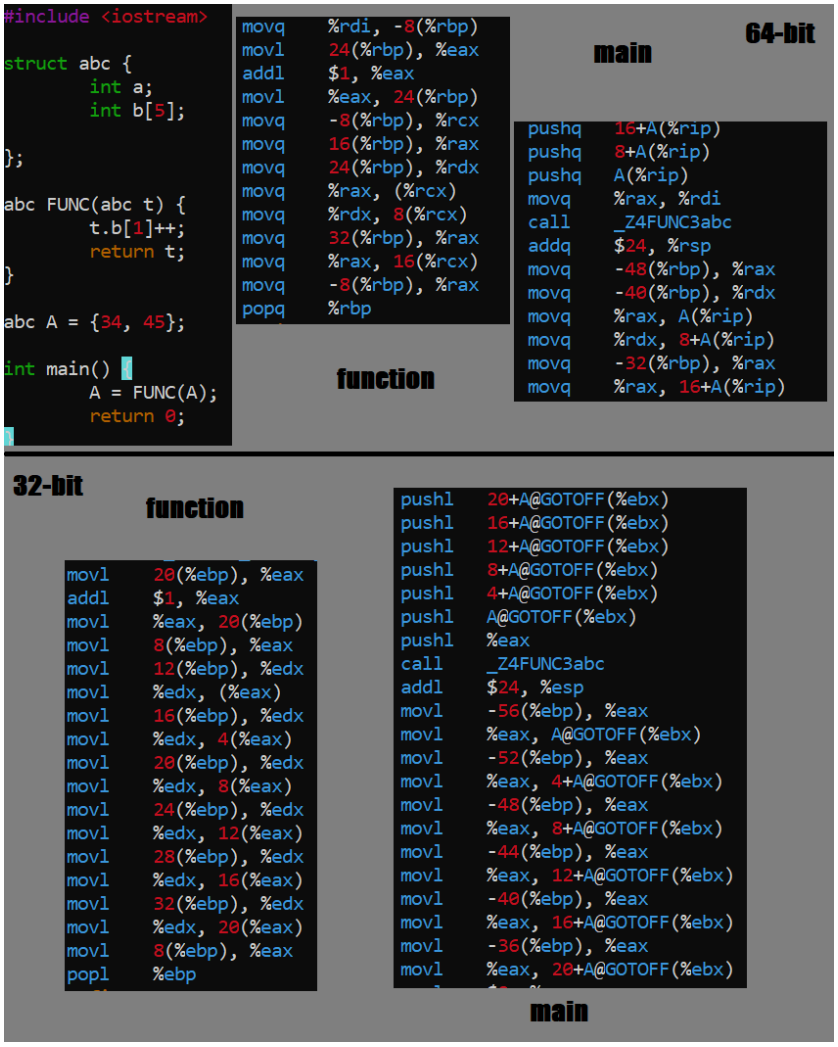

Рис. 3: К пункту 3

4. При передаче структуры по указателю не происходит записывание всей структуры в стек maina, 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.

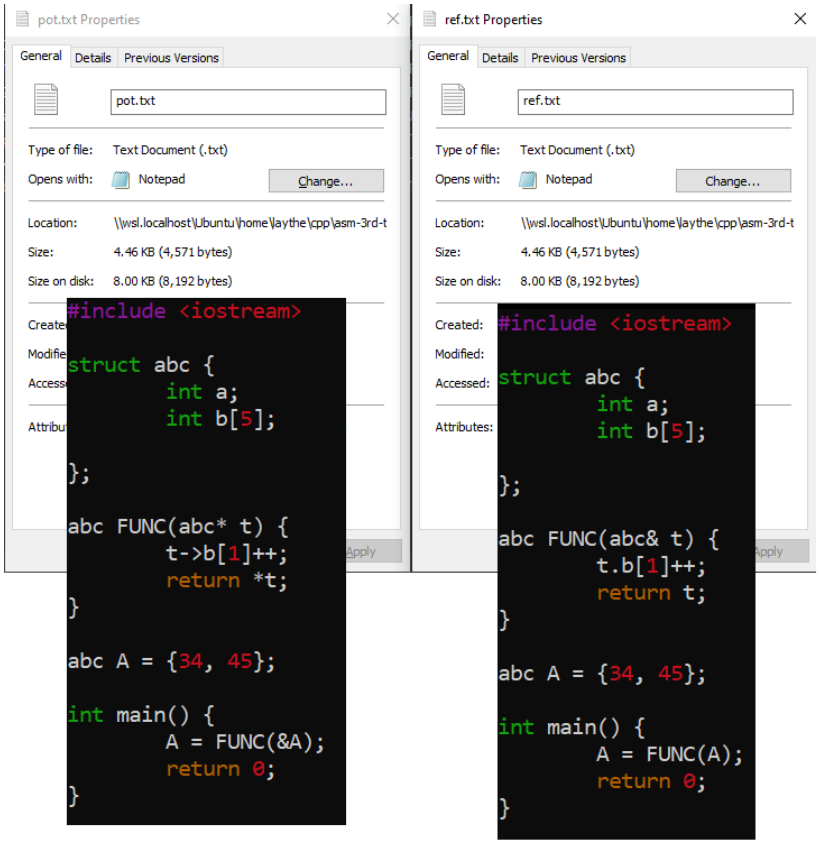

Рис. 4: К пункту 4

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 сегфолтунлась, 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. :)

Рис. 5: К пункту 6