Лекция 9. Программа демонстрации соглашений о вызовах ABI

В предыдущей лекции были описаны изменения состояний стека в процессе вызова функции и возврата из нее. Рассмотрим программу демонстрации этих изменений. Позже покажем ее работу в отладчике.

Программа `func.S`

Лекционный комментарий.

Главная программа — вызывающая, она вызывает функцию `Read_Sym`, которая в цикле получает очередной байт из массива `Symbols` и вызывает функцию `Trans_Sym`, передавая ей этот байт для обработки. `Trans_Sym` согласно ABI возвращает результат в регистре `%eax`. Заметим, что структура вложенных вызовов аналогична схеме в предыдущей лекции, где `F` вызывала `f1`, а та — `f2`.

NBNB. При выполнении команд функции `Trans_Sym` в архитектурном стеке будет

NB ДВА кадра стека — функции `Read_Sym` и, «выше» его — функции `Trans_Sym`.

```
#       ABI соглашения о вызовах функций

#   Байты из массива Symbols читаются по одному.
#   Если прочтен код цифры, то он преобразуется
#   в 4-байтовое целое, иначе в значение -1.
#   Результат записывается в элементы массив Numbers.
```

```
    .include "my-macro"   # подключение макро

.data # секция данных

Symbols:
    .asciz "91A23B456C789"    # массив символьных кодов
#                               цифр и "не цифр"

#   для показа исходного состояния стека
#   и мест РОН в нем после pusha в основной программ

Ini:    .ascii "INIT"  # стек
EAXF:   .ascii "axF "  # %eax
EDIF:   .ascii "diF "  # %edi
EBPm4:  .ascii "bp-4"  # для показа %ebp через %esi
```

Лекционный комментарий.

По умолчанию в окне «Память» отладчика kdbg **показано содержание байтов в 16-ой системе и в виде кодов символов.**

Если задать в этом окне адрес $esp**, то, записывая в регистры и в стек определенные выше константы, можно будет легко идентифицировать расположение в архитектурном стеке его элементов**

При этом символ F **в имени константы указывает на значения регистров главной программы (**"axF "**для регистра** %eax*),* **а пара символов** f1 **— на значения регистров функции** Read_Sym **(см. ниже** "axf1"**).**

NBNB. Т.к. по соглашениям ABI **регистр** %ebp **является базой кадра стека, его значения изменять НЕЛЬЗЯ, в том числе в него нельзя записывать значения идентифицирующих констант.**

Однако адрес этого регистра в стеке после выполнения команды pusha **меньше адреса регистра** %esi **на** 4**, которому и будем присваивать значение** "bp-4" **константы** EBPm4 **(одинаковое для** F **и**

f1) **идентифицирующей в стеке регистр** %ebp.

Ниже аналогично будет идентифицироваться расположение в архитектурном стеке локальных переменных вызываемых функций.

```
.bss    # секция общей памяти

.lcomm  Numbers, 40    # массив 4-х байтовых значений цифр

.global _start         # точка входа - глобальная метка

.text # секция команд процессора

_start:

    nop

#   Индикаторы исходных состояний:

#   ▶ Стека

    movl Ini, %eax
    movl %eax, 0(%esp)

#   ▶ Регистров общего назначения перед pusha

    movl EAXF,  %eax   # первый
    movl EDIF,  %edi   # последний
    movl EBPm4, %esi   # следующий после %ebp
                       # !!! %ebp НЕ ТРОГАТЬ !!!

    pusha  # РОН в стек

    pushl $Symbols # Параметр-2 - адрес массива в стек
    pushl $8       # Параметр-1 в стек, цикл 0-7

    call Read_Sym      # вызов функции

    addl $8,%esp   # очистить стек от параметров Read_Sym

    popa           # восстановить РОН

    Finish # конец работы, возврат в ОС
```

```
.type  Read_Sym, @function   # читает Symbols в цикле

#   Имеет два параметра

# P1 - число байтов для чтение из массива в ОП
# P2 - адрес массива откуда читать

# Прочтенный байт передается в Trans_Sym.
# Ее результат возвращается в %eax и передается
# в элементы массиве Numbers

Read_Sym:

#   Стандартный пролог

    pushl %ebp     # %ebp  вызывающей -> стек
    movl  %esp, %ebp   # обеспечить адресный доступ к
                       # параметрам и
#             локальным переменным в стеке путем базовой
#             адресации через ebp

.data # секция данных

LVAR1:  .ascii "LFr1" # показ локальной переменной Кадра стека
1

.text # секция команд процессора

    subl  $4, %esp  # завести локальную перемен. в Кадре 1
    movl  LVAR1,%eax
    movl  %eax,-4(%ebp)

#   Собственно Код  функции

    subl %ecx, %ecx   # инициируем цикл по байтам Symbols

#   Начало цикла

NextSym:
    movl 12(%ebp), %edx  # адрес P2 - массива в %edx

#   Подготовка вызова функции Trans_Sym

#   Ее параметр - байт передадим через %bl регистра %ebx
```

```
    subl %ebx, %ebx    # все нули

#   - передадим код символа в %bl
#   %edx - базовый - взяли из P2,
#   %ecx - индексный - номер цикла, MM = 1 - один байт
#   регистровая адресация

    movb (%edx,%ecx,1), %bl

#   Параметр Trans_Sym готов, можно ее вызывать.

.data # секция данных

#   для показа мест РОН в стеке после pusha в ReadSym

EAXf1:  .ascii "axf1"  # %eax
EDIf1:  .ascii "dif1"  # %edi

.text # секция команд процессора

#   Индикаторы РОН f1 перед pusha

    movl EAXf1,  %eax      # первый
    movl EDIf1,  %edi      # последний
    movl EBPm4,  %esi      # следующий после %ebp
                           # !!! %ebp НЕ ТРОГАТЬ !!!

    pusha           # сохранить РОН текущей функции Read_Sym

    pushl %ebx      # Параметр Trans_Sym в стек

    call Trans_Sym

    addl   $4,%esp     # очистить стек от параметров Trans_Sym

#   Опять работает Read_Sym

#   В %eax 4 байтовый результат Trans_Sym

#   Запись результата в массив Numbers.

#   Базовый регистр НЕ ЗАДАН - запятая после лев. скобки
#   %ecx - индексный регистр, масштабный множитель - 4
#   т.к. элементы Numbers - 4-х байтовые слова
#   регистровая адресация
```

```
    movl   %eax, Numbers(,%ecx,4)

    popa   # восстановть регистры Read_Sym

    incl %ecx              # наращиваем счетчик цикла
    cmpl 8(%ebp), %ecx # счетчик цикла равен первому
                         параметру?

    jne NextSym    # ДА, на повтор
                   # НЕТ - выходим из цикла

#   Стандартный эпилог функции

    movl %ebp, %esp    # восстановить указатель стека
    popl %ebp          # восстановить ebp
    ret         # возврат в вызывающую

# конец Read_Sym

#    Функция преобразования кода символа в числовое значение
#    с фильтрацией кодов цифр. P1 - байт кода. Возвращается
#    значение цифры или -1 если код не символа цифры.

.type  Trans_Sym, @function

Trans_Sym:

#   Стандартный пролог фунции

    pushl %ebp     # сохранить в стеке значение, бывшее в
                     вызывающей
    movl  %esp, %ebp   # обеспечить адресный доступ к
                         параметрам и
#             локальным переменным в стеке путем базовой
#             адресации через %ebp

.data # секция данных

LVAR2:
    .ascii "LFr2"  # показ локальной переменной Кадра 2

.text # секция команд процессора

    subl  $4, %esp  # завести локальную перемен. в Кадре 2
    movl  LVAR2,%eax
    movl  %eax,-4(%ebp)
```

```
#   тело функции

    movl 8(%ebp), %eax  # первый  параметр в eax

#   Фильтр кода символа цифры<

    cmpb $'9', %al     # код больше кода символа '9' ?
    ja Ret_error       # ДА - на возврат -1
    cmpb $'0', %al     # код меньше кода сивола '0' ?
    jb Ret_error       # ДА - на возврат -1

    subl $0x30, %eax   # получение числового значения

    jmp Ret_norm   # на возврат числ. значения цифры

Ret_error:

    movl $-1,%eax  # для возврата если код не символ цифры

Ret_norm:

#   Стандартный эпилог функции

    movl %ebp, %esp    # восстановить указатель стека
    popl %ebp          # восстановить ebp
    ret                # возврат в вызывающую

#   Конец  Trans_Sym

    .end   # последняя строка исходного текста
```