ОНЛАЙН-ОБРАЗОВАНИЕ

Меня хорошо слышно && видно?

Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

OTUS

Подпрограммы

Пакулов Артур

1

X86

Подпрограммы

OTUS

Соглашения вызовов

- **Stdcall**
- **Fastcall**
- **cdecl**

Подпрограммы

Синтаксис на ЯА:

```
<имя процедуры> PROC <параметр>
                <тело процедуры>
<имя процедуры> ENDP
```

Подпрограммы

Действия при вызове

- Передать параметры
- Вызвать подпрограмму
- Вернуться в место вызова

Рассмотрим псевдокод

Подпрограмма имеет два аргумента типа DWORD и две локальные переменные такого же типа

```
void test(DWORD arg1,DWORD arg2)
{
DWORD var1 = 0x11111111;
DWORD var2 = 0x22222222;
        Eax = var1;
        Ebx = var2;
        Push data
        Pop ecx
}
//main
Test(0x77777777, 0x99999999);

push arg2;      //arg2 = 0x99999999;
push arg1;      //arg1 = 0x77777777;
100: call test
105: xor eax,eax
```

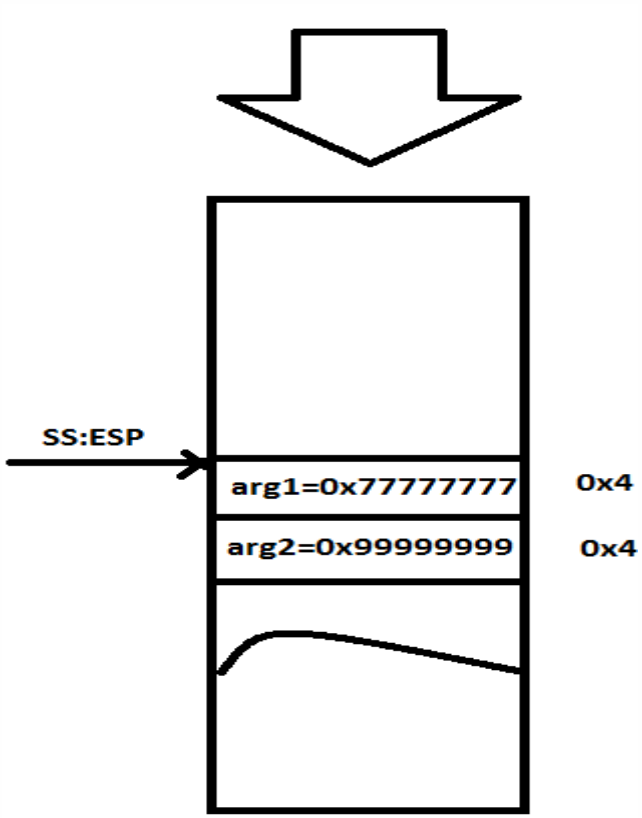
Почему локальные переменные - локальные?

Подпрограмма имеет два аргумента типа DWORD и две локальные переменные такого же типа

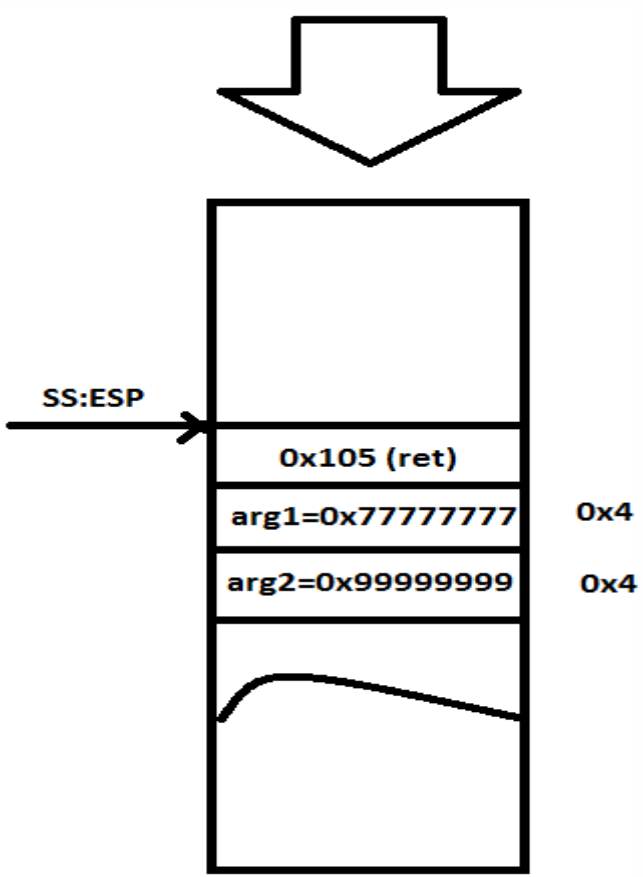
```
void test(DWORD arg1,DWORD arg2)
{
DWORD var1 = 0x11111111;
DWORD var2 = 0x22222222;
        Eax = var1;
        Ebx = var2;
        Push data
        Pop ecx
}
//main
Test(0x77777777, 0x99999999);

push arg2;      //arg2 = 0x99999999;
push arg1;      //arg1 = 0x77777777;
100: call test
105: xor eax,eax
```

Стадии стека

1. После передачи всех аргументов



2. После выполнения инструкции «Call»



Пролог функции стандартного вызова

```
55                          push        ebp
8B EC                       mov         ebp,esp
```

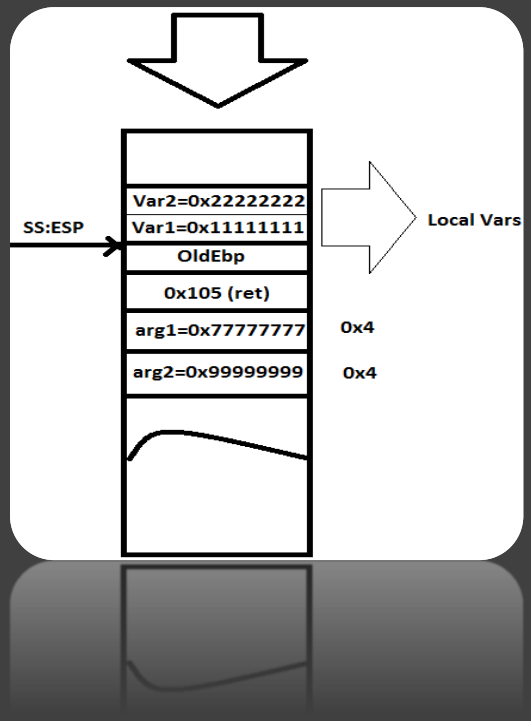
```
push ebp;           //сохраняю содержимое ebp
Mov ebp, esp        //запоминаю вершину стека
Sub esp, 8          //резервирую место в стеке
                    // для лок-х переменных
                    //размером 8 байт
```

Инициализация локальных переменных

.....

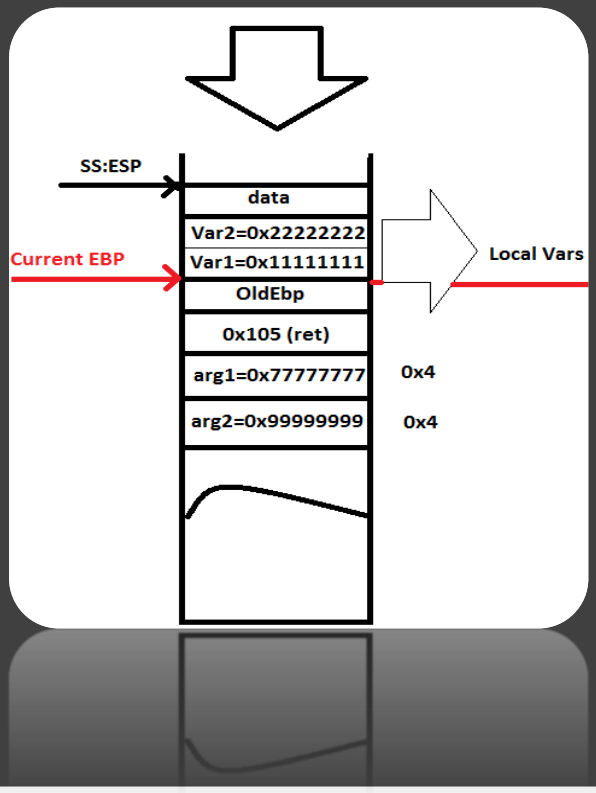
Стадии стека

1) Backup EBP
2) сохранение текущей вершины стека в EBP
3) резервироование места под локальные переменные и их инициализация



Стадии стека

1) Backup EBP
2) сохранение текущей вершины стека в EBP
3) резервироование места под локальные переменные и их инициализация



Доступ к переменным через EBP

Самый распространённый случай

```
void test(DWORD arg1,DWORD arg2)
{
DWORD var1 = 0x11111111;
DWORD var2 = 0x22222222;
        Eax = var1;
        Ebx = var2;
        Push data
        Pop ecx
}
//main
Test(0x77777777, 0x99999999);

push arg2;      //arg2 = 0x99999999;
push arg1;      //arg1 = 0x77777777;
100: call test
105: xor eax,eax
```

Eax = Var1 → mov eax, DWORD ptr[ebp-4];
Ebx = Var2 → mov ebx, DWORD ptr[ebp-8];

Eax = arg1 → mov eax, DWORD ptr[ebp+8];
Ebx = arg2 → mov ebx, DWORD ptr[ebp+12];

Доступ к переменным через ESP

Не самый распространённый случай

```
void test(DWORD arg1,DWORD arg2)
{
DWORD var1 = 0x11111111;
DWORD var2 = 0x22222222;
        Eax = var1;
        Ebx = var2;
        Push data
        Pop ecx
}
//main
Test(0x77777777, 0x99999999);

push arg2;      //arg2 = 0x99999999;
push arg1;      //arg1 = 0x77777777;
100: call test
105: xor eax,eax
```

Eax = Var1 → mov eax, DWORD ptr[esp-4+8];
Ebx = Var2 → mov ebx, DWORD ptr[esp-8+8];

Eax = arg1 → mov eax, DWORD ptr[esp+8+8];
Ebx = arg2 → mov ebx, DWORD ptr[esp+12+8];

Пример

```
format PE GUI 4.0
entry start

include '..\include\win32a.inc'
include '..\include\api\kernel32.inc'

section '.data' data readable writeable
        MyText db 'Test',0
section '.text' code readable executable

start:
        stdcall Test1
        stdcall Test2, 0x11223344
        invoke ExitProcess, 0


proc Test1
        locals
                x dd 0
                str db 0x10 dup(0)
        endl

        mov [x], 0x99887766
        lea eax, [x]
        invoke lstrcpy, eax, MyText
        ret
endp

proc Test2
        push ebp
        mov ebp, esp

        mov ecx, [ebp + 8]

        mov esp, ebp
        pop ebp
        ret
endp

section '.idata' import data readable writeable
        library kernel32, 'KERNEL32.DLL'
```

Пример

```
format PE GUI 4.0
entry start

include '..\include\win32a.inc'
include '..\include\api\kernel32.inc'

section '.data' data readable writeab
        MyText db 'Test',0
section '.text' code readable executa

start:
        stdcall Test1
        stdcall Test2, 0x11223344
        invoke ExitProcess, 0


proc Test1
        locals
                x dd 0
                str db 0x10 dup(0)
        endl

        mov [x], 0x99887766
        lea eax, [x]
        invoke lstrcpy, eax, MyText
        ret
endp

proc Test2
        push ebp
        mov ebp, esp

        mov ecx, [ebp + 8]

        mov esp, ebp
        pop ebp
        ret
endp

section '.idata' import data readable writeable
        library kernel32, 'KERNEL32.DLL'
```

```
.00403017: 55                  push        ebp
.00403018: 89E5                mov         ebp,esp
.0040301A: 83EC14              sub         esp,014
.0040301D: C745EC00000000      mov         d,[ebp][-014],0
.00403024: C745F000000000      mov         d,[ebp][-010],0
.0040302B: C745F400000000      mov         d,[ebp][-00C],0
.00403032: C745F800000000      mov         d,[ebp][-8],0
.00403039: C745FC00000000      mov         d,[ebp][-4],0
.00403040: C745EC66778899      mov         d,[ebp][-014],099887766 ;'ЩИwf'
.00403047: 8D45EC              lea         eax,[ebp][-014]
.0040304A: 6800204000          push        000402000 ;'Test' --↑1
.0040304F: 50                  push        eax
.00403050: FF1510104000        call        lstrcpyA
.00403056: C9                  leave
.00403057: C3                  retn ; -^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-
.00403058: 55                  push        ebp
.00403059: 89E5                mov         ebp,esp
.0040305B: 8B4D08              mov         ecx,[ebp][8]
.0040305E: 89EC                mov         esp,ebp
.00403060: 5D                  pop         ebp
.00403061: C3                  retn ; -^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-
```

2

X64

Подпрограммы

Соглашение вызовов – только модифицированный fastcall

- Первые 4 аргумента передаются через RCX, RDX, R8, R9
- Остальные – через стек, в обратном порядке
- Если в первых 4 параметрах есть вещественные числа, то они дублируются через XMM регистры: XMM0…XMM3

Подпрограммы

Соглашение вызовов – только модифицированный fastcall

- Первые 4 аргумента передаются через RCX, RDX, R8, R9
- Остальные – через стек, в обратном порядке
- Если в первых 4 параметрах есть вещественные числа, то они дублируются через XMM регистры: XMM0…XMM3

Всегда резервируется место в стеке под 4 локальных переменных!
Это место – теневая часть

Подпрограммы

Результат возвращается через RAX или XMM0

- Первые 4 аргумента передаются через RCX, RDX, R8, R9
- Остальные – через стек, в обратном порядке
- Если в первых 4 параметрах есть вещественные числа, то они дублируются через XMM регистры: XMM0…XMM3

Всегда резервируется место в стеке под 4 локальных переменных!
Стек должен быть выровнен на 16 байт

Пример

```
void test(int  int1, float float1, int int2, float float2, int int3, float float3  )
```

1. Push float3
2. Push int3
3. R9 = XMM3 = float2
4. R8 = int2
5. RDX = XMM1 = float1
6. RCX = int1
7. Call test

RCX	XMM0
RDX	XMM1
D8	XMM2
R9	XMM3

Внимание!

RBX, RSI, RDI, RSP, RBP, R12-R15, XMM6, XMM15

Не должны измениться после вызова fastcall функции

RAX, RDX, R8-R16, XMM0-XMM5

Могут изменяться! Если нужны, бэкапьте перед вызовом!

OTUS

Вопросы???

Пакулов Артур

A.Pakulov.Otus@Gmail.com

Спасибо
за внимание!