

# Введение во внутреннее устройство исполняемого Win32 PE файла

## Инструменты:

- Отладчик OllyDbg 1.10
- Редактор ресурсов, просмотрщик PE-заголовков PE Explorer
- Компилятор языка C/C++
- Редактор текстовых файлов Notepad++ (с HEX-Editor plugin)
- И в самом конце — интерактивный дизассемблер IDA

Для начала нам нужен опытный образчик. Желательно без лишнего «мусора» внутри. Делаем простейшее приложение под Win32. Вот его код:

```
//файл init.cpp
//
#pragma comment(linker, "/MERGE:.rdata=.text")
//в секции .data хранятся, например, данные объявленных глобальных переменных
//объединяем ее с секцией .text - с объединением надо быть осоторжным.
//У разных типов секций разные права доступа на выполнение, чтение и запись.
#pragma comment(linker, "/MERGE:.data=.text")
/*
//Раскомментируйте этот код, если пользуетесь компилятором из Visual Studio 6.0
#pragma comment(linker, "/FILEALIGN:512 /SECTION:.text,ERW /IGNORE:4078")
/**/
#pragma comment(linker, "/ENTRY:New_WinMain") //Переопределяем функцию WinMain
/*
//Если компилируете из IDE, можете раскомментировать эту строчку.
#pragma comment(linker, "/NODEFAULTLIB")
//Однако, в поздних версиях (например, 2005) Вы получите ошибку из-за
//автоматического добавления библиотеки безопасной работы со строками
//Нужно будет отключить Use Link-time code generation (/GL) или
//включить сюда данную библиотеку. Поскольку делаем минимальное приложение,
//жертвуем безопасностью
/**/
#pragma comment(lib, "user32.lib")
#pragma comment(linker, "/SUBSYSTEM:WINDOWS")

#include <windows.h>

int WINAPI New_WinMain(HINSTANCE hInstance,
                      HINSTANCE hPrevInstance,
                      LPTSTR lpCmdLine,
                      int nCmdShow)
{
    ::MessageBoxW(0, L"Text", L"Caption", MB_OK);
    // hOwner = NULL;
    // Text = "Text"
    // Title = "Caption"
    // Style = MB_OK|MB_APPLMODAL
    return 0;
}
```

Чтобы скомпилировать, создаем подобный cmd-файл

```
set PATH=%PATH%;C:\Program Files\Microsoft Visual Studio 8\VC\bin
set LIB=C:\Program Files\Microsoft Visual Studio 8\VC\lib;C:\Program
```

```
Files\Microsoft Visual Studio 8\VC\PlatformSDK\lib
set INCLUDE=C:\Program Files\Microsoft Visual Studio 8\VC\include;C:\Program
Files\Microsoft Visual Studio 8\VC\PlatformSDK\Include
::set LIBRARY_PATH=C:\Program Files\Microsoft Visual Studio 8\VC\lib;C:\Program
Files\Microsoft Visual Studio 8\VC\PlatformSDK\Lib
::set CPLUS_INCLUDE_PATH=C:\Program Files\Microsoft Visual Studio
8\VC\include;C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\Include

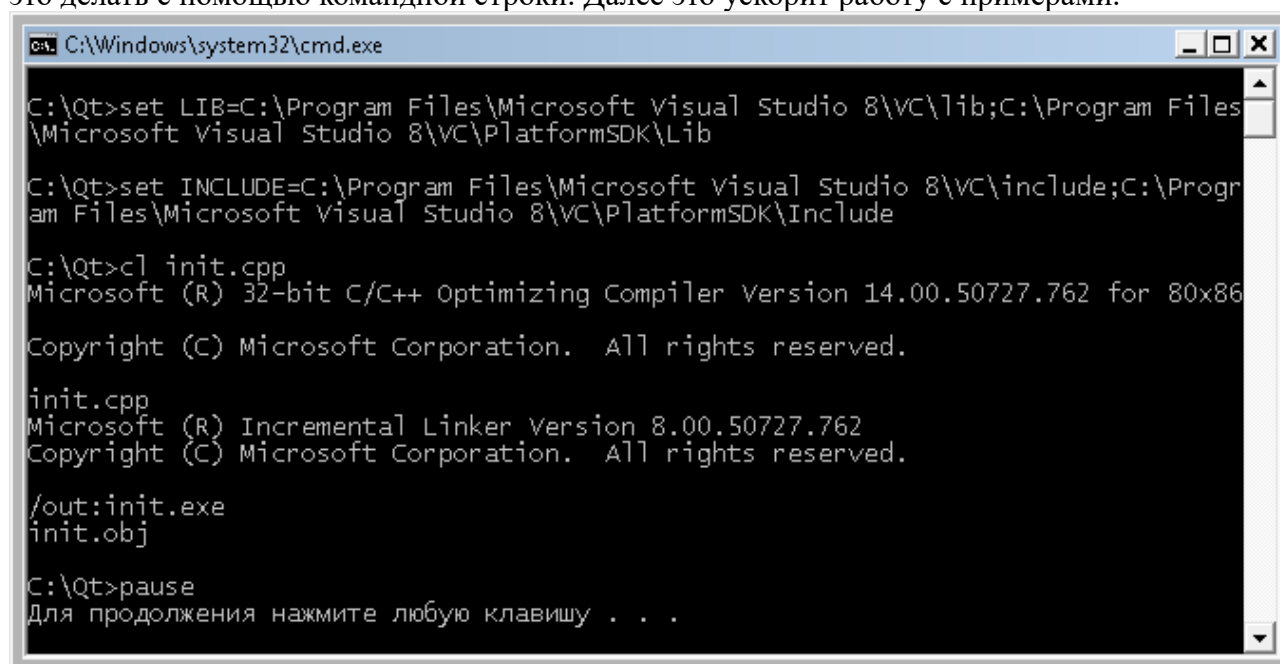
cmd
```

Поправьте пути, если у Вас другая версия Visual Studio или она установлена в другое место. В открывшемся окне командной строки пишем

```
cl init.cpp
```

У меня в Visual Studio 2005 в директории компилятора не оказалось библиотеки mspdb80.dll, которую требует cl.exe. Надо на время скопировать ее из общей папки Visual Studio 2005 (Common7\IDE).

Можно, конечно, скомпилировать данный пример непосредственно в IDE, но лучше научиться это делать с помощью командной строки. Далее это ускорит работу с примерами.



```
cmd. C:\Windows\system32\cmd.exe

C:\Qt>set LIB=C:\Program Files\Microsoft Visual Studio 8\VC\lib;C:\Program Files
\Microsoft Visual Studio 8\VC\PlatformSDK\Lib

C:\Qt>set INCLUDE=C:\Program Files\Microsoft Visual Studio 8\VC\include;C:\Progr
am Files\Microsoft Visual Studio 8\VC\PlatformSDK\Include

C:\Qt>cl init.cpp
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 14.00.50727.762 for 80x86
Copyright (C) Microsoft Corporation. All rights reserved.

init.cpp
Microsoft (R) Incremental Linker Version 8.00.50727.762
Copyright (C) Microsoft Corporation. All rights reserved.

/out:init.exe
init.obj

C:\Qt>pause
Для продолжения нажмите любую клавишу . . .
```

Рис. 1.1.

За более подробной информацией о компиляции в Visual Studio из командной строки обратитесь к [документации](#).

Скомпилированное приложение init.exe и исходный код init.cpp находятся в архиве example\_app.zip.

Приложение выводит диалоговое окно и после нажатия кнопки ОК завершает работу.

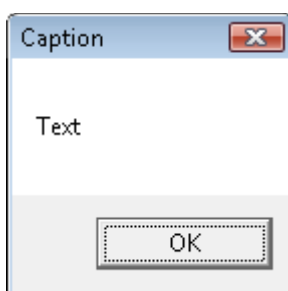


Рис. 1.2.

Загружаем файл init.exe в OllyDbg (нажимаем пункт меню File → Open и выбираем наш файл).

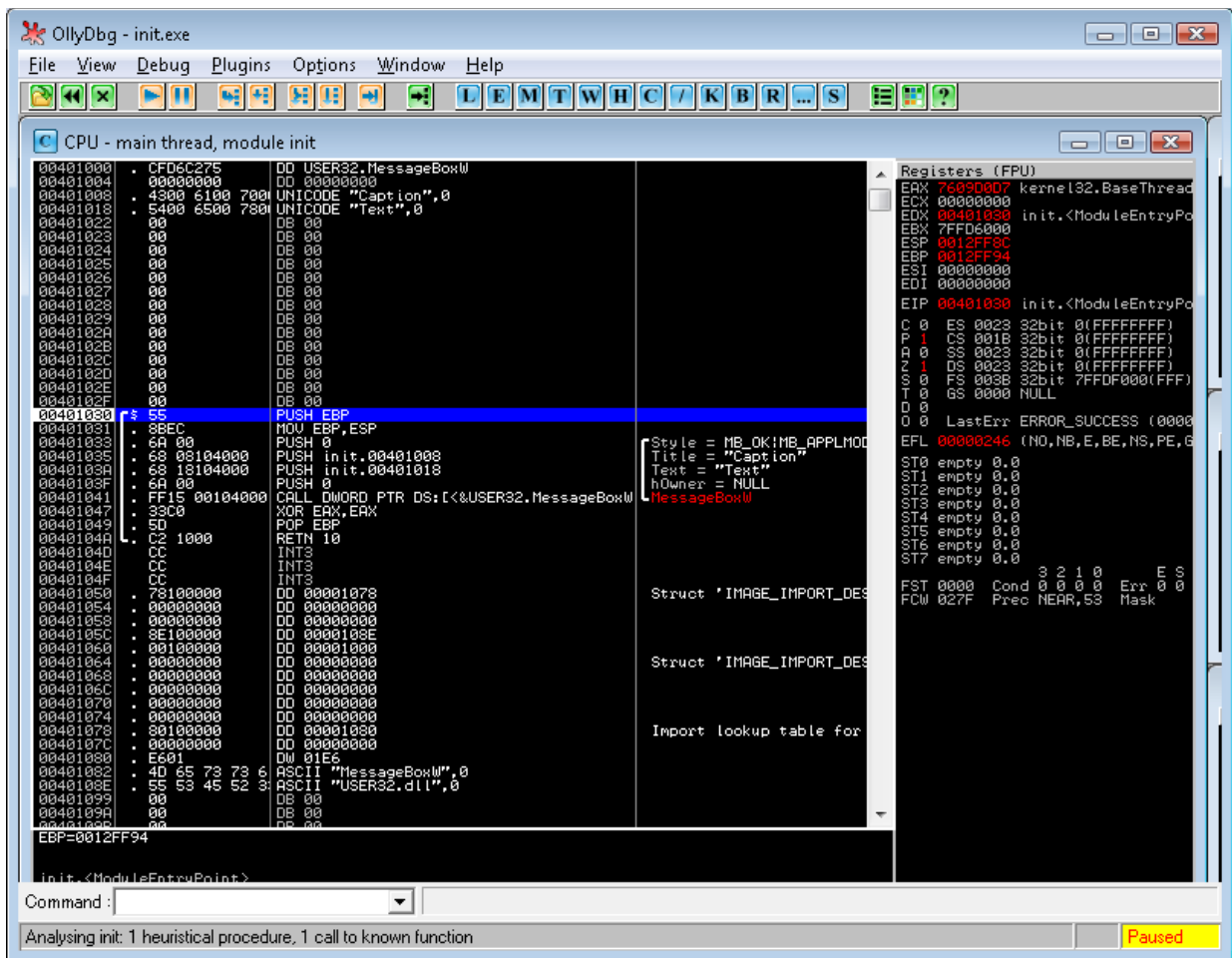


Рис. 1.3.

В окне «CPU» в левом столбце фон текста черный, кроме одной строчки. Это так называемая точка входа в программу (Entry Point) – наша функция `New_WinMain` из исходного кода. Она располагается по адресу `0x00401030`. Это не смещение в «физическом» файле, а виртуальный адрес. Также в этом окне можно посмотреть значения регистров и флагов процессора, но об этом позже, когда будем изучать условие `if-else`.

Для наглядности откроем наш файл также в PE Explorer. И перейдем во вкладку «Section Headers»

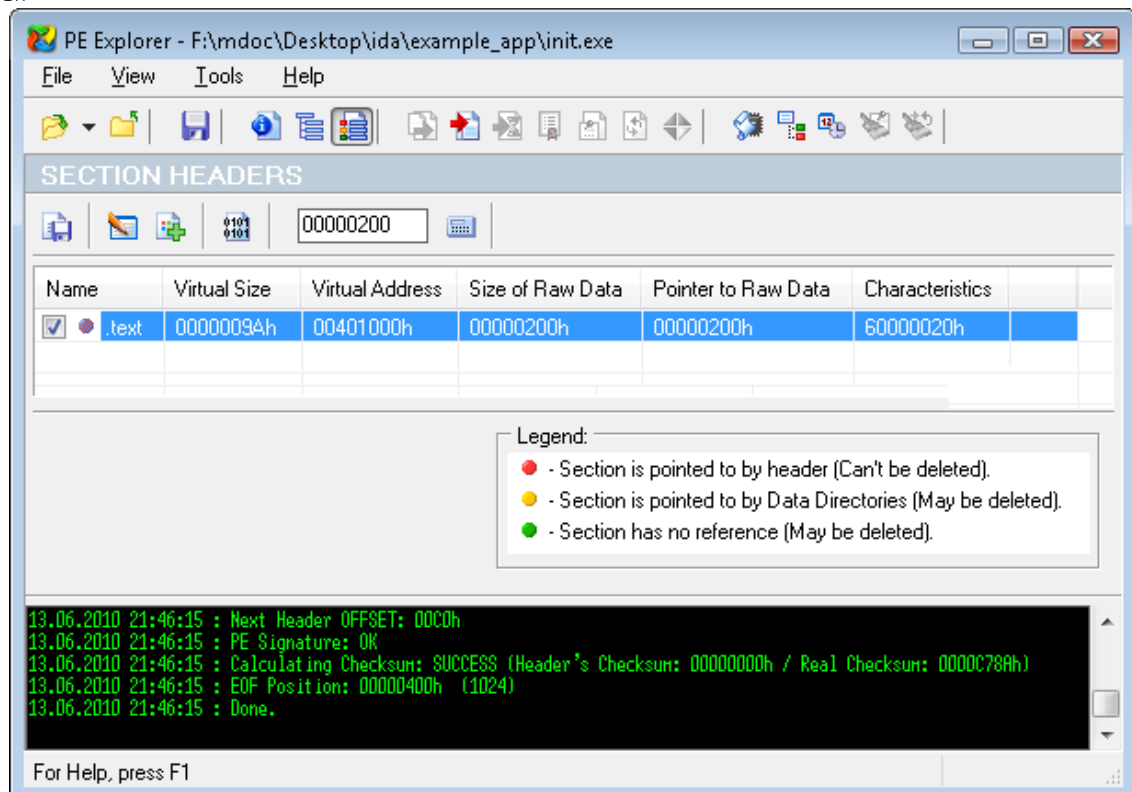


Рис. 1.4.

Мы видим одну секцию с именем `.text` – в исходнике мы указали линкеру оставить только ее. В ней располагается весь код вместе со значением переменных. Нам нужны 2 параметра: Virtual Address = `0x00401000` (он получился путем сложения *Image Base* + *Base of Code*, см. вкладку «*Headres Info*») и Pointer to Raw Data = `0x00000200`. Второй параметр указывает на физический адрес в нашем файле. А поскольку Entry Point = `0x00401030`, что на `0x30` больше Virtual Address, то физическое смещение Entry Point = `0x00000230` (на рис. 1.5. отмечен красным фоном).

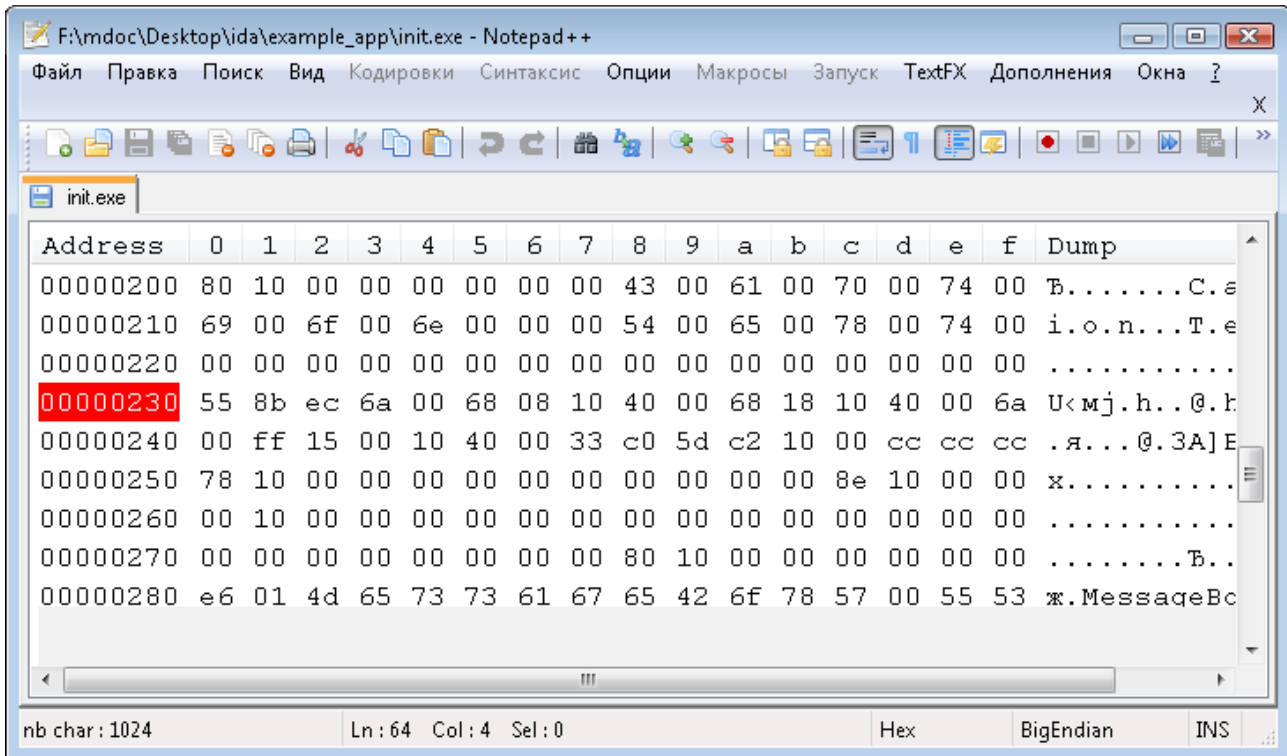


Рис. 1.5.

Возвращаясь к рис. 1.4., укажем, что параметр Size of Raw Data равен `0x200`, что соответствует команде линкеру `/FILEALIGN:512`, прописанной в исходнике — меньшее делать не желательно.

То есть, сложив Pointer to Raw Data и Size of Raw Data, получим физический адрес окончания секции `.text`. Он равен `0x00000400`, что соответствует 1024 Байт — размеру нашего файла.

Для нас представляет интерес все то, что находится по физическому смещению `0x00000200` (виртуальному `0x00401000`) и выше. До этого смещения в файле располагается PE-заголовок, с чтением которого хорошо и наглядно справляется PE Explorer.

Все, что находится до секции `.text`, нам пока не очень нужно. Тем более формат исполняемого файла — большая обширная тема. Если будет интересно, почитайте следующие статьи:

[Загрузчик PE-файлов](#) — подробно рассмотрена структура и метод загрузки.

[Исследование переносимого формата исполнимых файлов](#) — структура файла

И сугубо практика — а именно код:

[IMAGE\\_DOS\\_HEADER](#), [PIMAGE\\_NT\\_HEADERS](#), [IMAGE\\_FILE\\_HEADER](#), [IMAGE\\_OPTIONAL\\_HEADER32](#), [IMAGE\\_DIRECTORY\\_ENTRY\\_EXPORT](#), [Список функций из DLL AddressOfNames](#)

В PE Explorer есть встроенный дизассемблер, но в OllyDbg он намного лучше, более того Olly является дебаггером. Как им пользоваться, отлично рассказано в статьях Рикарда Нарвахо, переводы которых доступны на [www.wasm.ru](http://www.wasm.ru).

Далее посмотрим листинг дизассемблированного кода нашего приложения. Для этого откроем окно «CPU» в OllyDbg (см. рис. 1.3.).

```

; Листниг 1
; резюмируя вышесказанное, если нужно получить физический адрес, вычитаем из
0x00401200 виртуальный
00401000 > . CFD6C275 DD USER32.MessageBoxW; см. 00401080 - импорт функции
00401004 00000000 DD 00000000
; наши строки, 1 символ занимает 2 байта т.к. Unicode-символы. Латинские коды
расположены в младших байтах и аналогичны кодам ASCII. Старшие байты равны 00
00401008 . 4300 6100 7000>UNICODE "Caption",0
00401018 . 5400 6500 7800>UNICODE "Text",0 ;
; следующие нули появились от того, что адрес Entry Point должен быть кратен 0xF,
а последнее слово неудачно «залезло» на пару соседних байт. Мы можем в этом
убедиться, если перекомпилируем пример, убрав одну букву у слова Text
00401022 00 DB 00
00401023 00 DB 00
00401024 00 DB 00
00401025 00 DB 00
00401026 00 DB 00
00401027 00 DB 00
00401028 00 DB 00
00401029 00 DB 00
0040102A 00 DB 00
0040102B 00 DB 00
0040102C 00 DB 00
0040102D 00 DB 00
0040102E 00 DB 00
0040102F 00 DB 00
; наконец-то добрались до точки входа в приложение. Сейчас будет выполняться
основная функция New_WinMain, в которой вызовется написанная нами в исходнике
функция MessageBoxW
; для доступа к элементам стека используют регистр EBP
; следующие 2 строчки – стандартный вход в функцию, используемый языками высокого
уровня
00401030 >/$ 55 PUSH EBP ; открываем кадр стека
00401031 |. 8BEC MOV EBP,ESP ; ESP – указывает на верхнее зн-е стека
; кладем в| стек (PUSH) тип кнопок – последний параметр функции
00401033 |. 6A 00 PUSH 0 ; /Style = MB_OK|MB_APPLMODAL
; кладем в| стек текст сообщения путем указания адреса, по которому он хранится
00401035 |. 68 08104000 PUSH init.00401008 ; |Title = "Caption"
; кладем в| стек его заголовок путем указания адреса, по которому он хранится
0040103A |. 68 18104000 PUSH init.00401018 ; |Text = "Text"
; кладем в| стек хэндл
0040103F |. 6A 00 PUSH 0 ; |hOwner = NULL
; вызываем| нашу функцию, которая возьмет из стека наши параметры
00401041 |. FF15 00104000 CALL DWORD PTR DS:[<&USER32.MessageBoxW>>
; | ; \MessageBoxW
00401047 |. 33C0 XOR EAX,EAX ; обнуляем EAX т.к. ф-я return 0)
; берем из стека EBP, в котором хранится прежний ESP. Закрываем кадр стека
00401049 |. 5D POP EBP
; RETN ставится в конце процедуры. Она извлекает из стека адрес возврата и
осуществляет переход по этому адресу, при этом стек возвращается в свое исходное
состояние (до команды CALL). 10 – объем параметров, взятый у стека (2 значения
типа int по 4 байта + 2 указателя на строка по 4 байта = 16 Байт = 0x10)
0040104A \. C2 1000 RETN 10 ; выход из процедуры
0040104D CC INT3
0040104E CC INT3
0040104F CC INT3
; начало структуры таблицы импорта
00401050 . 78100000 DD 00001078 ; Struct 'IMAGE_IMPORT_DESCRIPTOR'
00401054 . 00000000 DD 00000000
00401058 . 00000000 DD 00000000
0040105C . 8E100000 DD 0000108E
00401060 . 00100000 DD 00001000
00401064 . 00000000 DD 00000000 ; Struct 'IMAGE_IMPORT_DESCRIPTOR'
00401068 . 00000000 DD 00000000
0040106C . 00000000 DD 00000000
00401070 . 00000000 DD 00000000
00401074 . 00000000 DD 00000000

```

```

00401078 . 80100000 DD 00001080 ; Import lookup table for 'USER32.dll'
0040107C . 00000000 DD 00000000
00401080 . E601 DW 01E6
00401082 . 4D 65 73 73 61>ASCII "MessageBoxW",0 ; импортируем функцию
0040108E . 55 53 45 52 33>ASCII "USER32.dll",0 ; из библиотеки
00401099 00 DB 00
; ... и далее до конца секции .text сплошные нули, так как определено выравнивание
0x200 (512 Байт), а меньшее делать не рекомендуется.
00401FFF 00 DB 00

```

Делаем программу немногим более сложной. А именно вводим глобальную переменную. Код будет выглядеть так:

```

...
int i = 0x0C;
int WINAPI New_WinMain(HINSTANCE hInstance,
                       HINSTANCE hPrevInstance,
                       LPTSTR lpCmdLine,
                       int nCmdShow)
{
    ++i;
    ::MessageBoxW(0, L"Text", L"Caption", MB_OK);
    return 0;
}

```

Компилируем и полученный файл загружаем в OllyDbg. Исходник и уже скомпилированное приложение находятся в архиве под именами init\_global\_0x0C.cpp и init\_global\_0x0C.exe соответственно.

Листинг дизассемблерного кода немного изменился. Для экономии места приведем и откомментируем только измененные и важные участки:

```

; Листинг 2
00401000 > . CFD6C275 DD USER32.MessageBoxW
00401004 00000000 DD 00000000
; по адресу 00401008 находится уже не строка "Caption", а значение нашей новой
глобальной переменной 0x0C. Если бы мы не указали компилятору произвести слияние
секции .data с секцией .text, этой переменной здесь бы не было (она находилась бы
в секции .data, а мы рассматриваем секцию .text).
00401008 . 0C000000 DD 0000000C
; адреса остальных данных сместились
0040100C . 4300 6100 7000>UNICODE "Caption",0
0040101C . 5400 6500 7800>UNICODE "Text",0
00401026 00 DB 00
; ... опустили выравнивание нулями. Кстати, их на 4 байта меньше из-за введения
переменной, т.к. тип int в нашем случае занимает 4 байта
0040102F 00 DB 00
00401030 >/$ 55 PUSH EBP
00401031 |. 8BEC MOV EBP,ESP
; записываем EAX значение из ячейки памяти 00401008 (DWORD Pointer) — число 0x0C.
00401033 |. A1 08104000 MOV EAX,DWORD PTR DS:[401008]
; команда ADD увеличивает значение регистра на 1
00401038 |. 83C0 01 ADD EAX,1
; записываем в адрес нашей переменной значение регистра EAX
0040103B |. A3 08104000 MOV DWORD PTR DS:[401008],EAX
; кладем в стек (PUSH) тип кнопок — последний параметр функции
00401040 |. 6A 00 PUSH 0 ; /Style = MB_OK|MB_APPLMODAL
00401042 |. 68 0C104000 PUSH init.0040100C ; |Title = "Caption"
00401047 |. 68 1C104000 PUSH init.0040101C ; |Text = "Text"
0040104C |. 6A 00 PUSH 0 ; |hOwner = NULL
0040104E |. FF15 00104000 CALL DWORD PTR DS:[<USER32.MessageBoxW>]
; \MessageBoxW
00401054 |. 33C0 XOR EAX,EAX
00401056 |. 5D POP EBP
; взятый у стека объем параметров не изменился, так как введенная переменная на
стек не влияла
00401057 \. C2 1000 RETN 10

```

## А теперь посмотрим на устройство указателей и ссылок.

Меняем исходный код (см. файл `init_global_ptr.cpp`):

```
...
//Закомментируем строчку, где указали линкеру произвести слияние секций
//Размер программы увеличится. Можно этот участок раскомментировать,
//но в этом случае нужно отказаться от изменения зн-й глобальных переменных
//#pragma comment(linker, "/MERGE:.data=.text")
...
int i = 0x0C;
int *j = &i;
int WINAPI New_WinMain(HINSTANCE hInstance,
                       HINSTANCE hPrevInstance,
                       LPTSTR lpCmdLine,
                       int nCmdShow)
{
    *j = 0x0D;
    ::MessageBoxW(0, L"Text", L"Caption", MB_OK);
    return 0;
}
```

## Компилируем и получаем

```
; Листинг 3
; наша новая секция .data
;-----
; Name: .data (Data Section)
; Virtual Address: 00402000h Virtual Size: 00000008h
; Pointer To RawData: 00000600h Size Of RawData: 00000200h
;
; теперь наша глобальная переменная int i = 0x0C располагается в секции .data
00402000 0C DB 0Ch ; | переменная i = 0x0C
00402001 00 DB 00h ; | типа int, который
00402002 00 DB 00h ; | занимает
00402003 00 DB 00h ; | 4 байта
; там же и наш указатель j на переменную i
00402004 00204000 DD 00402000 ; | хранит адрес переменной
;-----

; а в дизассемблированном коде будут следующие изменения:
...
00401030 >/$ 55 PUSH EBP
00401031 |. 8BEC MOV EBP,ESP
; занести в регистр EAX значение по адресу 00402004, то есть
; фактически там адрес нашей переменной (00402000)
00401033 |. A1 04204000 MOV EAX,DWORD PTR DS:[402004]
; записать по адресу, хранящемуся в регистре EAX (00402000), значение 0x0D
00401038 |. C700 0D000000 MOV DWORD PTR DS:[EAX],0D
0040103E |. 6A 00 PUSH 0 ; /Style = MB_OK|MB_APPLMODAL
...
```

## Переносим объявления всех переменных в тело функции

В этом случае секция `.data` создаваться не будет. Исходный код примера находится в файле `init_dyn_ptr.cpp`.

```
...
int i = 0x0C;
int *j = &i;
*j = 0x0D;
::MessageBoxW(0, L"Text", L"Caption", MB_OK);
...
```

## Компилируем и получаем

```
; Листинг 4
00401000 > . CFD6C275 DD USER32.MessageBoxW
00401004 00000000 DD 00000000
```

```

; наши строки снова на прежних адресах, как в листинге 1
00401008 . 4300 6100 7000>UNICODE "Caption",0
00401018 . 5400 6500 7800>UNICODE "Text",0 ;

00401026 00 DB 00
; ... опустили выравнивание нулями. Их количество такое же, ка и в листинге 1
0040102F 00 DB 00
00401030 >/$ 55 PUSH EBP
00401031 |. 8BEC MOV EBP,ESP
; наши объявленные переменные в теле функции
; размер int 4 байта + размер указателя 4 байта
; команда SUB, как Вы понимаете, осуществляет вычитание двух операндов,
; помещая результат в первый операнд
00401033 |. 83EC 08 SUB ESP,8 ; кол-во байт стека для локальных переменных
; EBP-4 — адрес в стеке первой локальной переменной i.
; пока абсолютный адрес не знаем. В него заносим значение 0x0C
00401036 |. C745 FC 0C0000>MOV DWORD PTR SS:[EBP-4],0C
; запись в регистр EAX адреса переменной i, расположенной в EBP-4
; теперь в EAX адрес, где лежит наше число 0x0C
0040103D |. 8D45 FC LEA EAX,DWORD PTR SS:[EBP-4]
; объявление указателя j и занесение в него адреса i (int *j = &i)
00401040 |. 8945 F8 MOV DWORD PTR SS:[EBP-8],EAX
; запись в регистр ECX адреса указателя j, расположенного в EBP-8
; теперь в ECX адрес указателя, в котором записан адрес нашего числа 0x0C
00401043 |. 8B4D F8 MOV ECX,DWORD PTR SS:[EBP-8]
; получения значения указателя (в нем адрес переменной i)
; и запись по адресу переменной i (в переменную i) значения 0x0D
00401046 |. C701 0D000000 MOV DWORD PTR DS:[ECX],0D
; с локальными переменными дизассемблированный листинг читается сложнее
; далее код не изменился
; кладем в стек (PUSH) тип кнопок — последний параметр функции
00401040 |. 6A 00 PUSH 0 ; /Style = MB_OK|MB_APPLMODAL
00401042 |. 68 0C104000 PUSH init.0040100C ; |Title = "Caption"
00401047 |. 68 1C104000 PUSH init.0040101C ; |Text = "Text"
0040104C |. 6A 00 PUSH 0 ; |hOwner = NULL
0040104E |. FF15 00104000 CALL DWORD PTR DS:[<&USER32.MessageBoxW>> ; \MessageBoxW
|
00401054 |. 33C0 XOR EAX,EAX
00401056 |. 5D POP EBP
; взятый у стека объем параметров не изменился, так как введенная переменная на
; стек не влияла
00401057 \. C2 1000 RETN 10

```

## Условие if – else

Введем в программу условие if-else (см. файл `init_cycle.cpp`). Для легкости чтения кода сделаем переменную `i` глобальной, уберем указатель `j` и укажем линкеру произвести слияние секций.

```

...
if(i == 0x01)
{
    ::MessageBoxW(0, L"Text", L"Caption", MB_OK);
}
else
{
    i = 0x02; //вызовет исключение, если i не будет в секции .data
}

```

Далее приведем кусок дисассемблированного листинга, где идет проверка условия:

```

00401008 . 01000000 DD 00000001 ; наша переменная i = 0x01
0040100C . 4300 6100 7000>UNICODE "Caption",0
0040101C . 5400 6500 7800>UNICODE "Text",0
...
; сравнение путем вычитания значения по адресу 00401008 со значением 0x01.

```



```

; CMP устанавливает флаг Z=1, если разность операторов равна 0 (см. рис 1.3)
00401033 |. 833D 08104000 >CMP DWORD PTR DS:[401008],1
; Jmp if Not Zero – условный переход «прыгнуть, если не ноль».
; Если Z=0, то действие похоже на JMP (смена EIP на указанный адрес,
; то есть прыгнуть, куда сказали – 00401052).
; Если Z=1, то действие как у NOP (смена EIP на адрес следующей команды, то есть
; ничего не происходит) – выполняется инструкция, следующая за инструкцией JNZ
0040103A |. 75 16 JNZ SHORT init_cyc.00401052
; поскольку у нас i=0x01 и сравнивается с 0x01, то выполняется следующая инструкция
; кладем в стек (PUSH) тип кнопок – последний параметр функции
0040103C |. 6A 00 PUSH 0 ; /Style = MB_OK|MB_APPLMODAL
0040103E |. 68 0C104000 PUSH init_cyc.0040100C ; |Title = "Caption"
00401043 |. 68 1C104000 PUSH init_cyc.0040101C ; |Text = "Text"
00401048 |. 6A 00 PUSH 0 ; |hOwner = NULL
0040104A |. FF15 00104000 CALL DWORD PTR DS:[<USER32.MessageBoxW>
| ; \MessageBoxW
; безусловный переход на адрес 0040105C нужен для того, чтобы при
; выполнении условия при if (i = 0x01) не выполнилось условие else
00401050 |. EB 0A JMP SHORT init_cyc.0040105C
; следующая инструкция выполняется, если i != 0x01
; она записывает в ячейку переменной i значение 0x02
00401052 |> C705 08104000 >MOV DWORD PTR DS:[401008],2
0040105C |> 33C0 XOR EAX,EAX ; обнуляем EAX

```

### Описание флагов процессора

- Z** – флаг нуля. Устанавливается в 1, если результат предыдущей операции – ноль
- S** – флаг знака. Он всегда равен старшему биту результата
- C** – флаг переноса. Устанавливается в 1, если результат предыдущей операции над беззнаковыми числами не уместился в приёмнике и произошёл перенос из старшего бита, или если требуется заём (при вычитании), иначе 0
- O** – флаг переполнения. Устанавливается в 1, если результат предыдущей арифметической операции над числами со знаком выходит за допустимые для них пределы
- A** – флаг полупереноса. Устанавливается в 1, если в результате предыдущей операции произошёл перенос или заём из третьего бита в четвёртый. Этот флаг используется автоматически командами двоично-десятичной коррекции. Например, после двух команд: `mov eax, 15; inc eax;` флаг AF будет равен единице. После последовательности команд: `mov eax, 16; dec eax;` флаг AF также будет равен единице
- P** – флаг чётности. Устанавливается в 1, если младший байт результата предыдущей команды содержит чётное число битов, равных единице, иначе 0. Например, после двух команд: `mov al, 2; inc al;` флаг PF установится в 1
- I** – флаг прерываний. 1 – прерывания разрешены, 0 – прерывания запрещены
- D** – флаг направления. 0 – строки обрабатываются в сторону увеличения адресов, 1 – в сторону уменьшения адресов

### Стандартные команды условного перехода:

Код команды	Условие перехода	Переход, если аргументы CMP	Код команды	Условие перехода	Переход, если аргументы CMP
JA	C=0 и Z=0	если выше	JG	Z=0 и S=0	если больше
JAЕ JNC	C=0	если выше или равно если нет переноса	JGE	S=0	если больше или равно
JB JC	C=1	если ниже если перенос	JL	S<>0	если меньше
JBE	C=1   Z=1	если ниже или равно	JLE	Z=1   S<>0	если меньше или равно
JE JZ	Z=1	если равно если ноль	JNE JNZ	Z=0	если не равно если не ноль
JO	O=1	если есть переполнение	JNO	O=0	если нет переполнения

JS	S=1	если есть знак	JNS	S=0	если нет знака
JP	P=1	если есть четность	JNP	P=0	если нет четности

Также вместо команды `cmp` может быть команда `test` – логическое битовое И.

Проводя аналогию с высокоуровневой конструкцией, имеем

C/C++	Assembler
<pre>// ---      if(i == 0x01)     { }      else     { }  // ---</pre>	<pre>; --- CMP DWORD PTR DS:[i], 0x01 ;далее сверка условия if== JNZ 0xADDRESS else ; код, если i==0x01 (if) JMP to "остальной код программы" 0xADDRESS else ; здесь код, если i!=0x01 (else) ; --- ; остальной код программы</pre>
<pre>// ---      if(i != 0x01)     { }      else     { }  // ---</pre>	<pre>; --- CMP DWORD PTR DS:[i], 0x01 ;далее сверка условия if== JE to 0xADDRESS else ; код, если i!=0x01 (if) JMP to "остальной код программы" 0xADDRESS else ; здесь код, если i==0x01 (else) ; --- ; остальной код программы</pre>
<pre>// ---      while(i == 0x01)     { }  // ---</pre>	<pre>; --- CMP DWORD PTR DS:[i],0x01 JNZ to "остальной код программы" ; код, если i==0x01 JMP to 0xADDRESS CMP ; --- ; остальной код программы</pre>
<pre>// ---      while(i != 0x01)     { }  // ---</pre>	<pre>; --- CMP DWORD PTR DS:[i],0x01 JE to "остальной код программы" ; код, если i==0x01 JMP to 0xADDRESS CMP ; --- ; остальной код программы</pre>