Полная перекомпиляция простых exe-программ
или тестируем демо-версию IdaPro 5.7

by **Erfaren**
http://erfaren.narod.ru
erfaren@rambler.ru

Достоинства плагина pdb.plw версии 5.7

С момента публикации первой статьи (http://erfaren.narod.ru/Asm/Erfaren001.htm) прошло совсем немного времени, как вышла уже новая демо-версия знаменитой программы **Ильфака Гильфанова** – **IdaPro 5.7** (http://hex-rays.com/idapro/idadown.htm). Поэтому, грех ее не протестировать 🙂, тем более что у нас было достаточно замечаний относительно проблем с загрузкой отладочных **pdb**-символов в предыдущей версии **5.6**.

Кстати, удалось выяснить, в чем заключается эта проблема. Как оказалось, она содержится в плагинах **pdb.plw**, версий **5.5** и **5.6**. Если вместо них положить плагин версии **5.0** или **5.2** (версии **5.3** и **5.4** найти не удалось), то при наличии в корне «Иды» трех файлов из пакета **«Debugging Tools for Windows (x86)»** (http://msdl.microsoft.com/download/symbols/debuggers/dbg_x86_6.11.1.404.msi) :

dbgeng.dll, v. 6.11.0001.404;
dbghelp.dll, v. 6.11.0001.404;
symsrv.dll, v. 6.11.0001.404,

загрузка символов происходит из Интернета практически без проблем. Однако ручная загрузка **pdb**-файлов, с помощью старых плагинов, может вызвать затруднения.

К счастью, плагин **pdb.plw**, версии **5.7** работает великолепно. Без вышеупомянутых трех файлов, он без проблем загружает существующие символы локально, с жесткого диска (через меню **File / Load file / PDB file . . .**), а при их наличии – автоматически, через Интернет, при запуске соответствующей программы для анализа ее в **IdaPro 5.7**.

Дополнительная настройка IdaPro 5.7

В новой «Иде» появились новые опции. Рядом с опцией **«Analysis options»** появилось место для галки **«Imported DLL options»**. Только описана она в другом файле – **pe.xml**. Однако идея та же, чтобы не отмечать постоянно это поле, мы заменим

```
<checkbox xpath:visible="/ida/loaders/pe/@ordinals" variable="../../page[@X='pe_import_p']/@visible" caption="Imported DLL options"/>
```

на

```
<checkbox checked="true" xpath:visible="/ida/loaders/pe/@ordinals" variable="../../page[@X='pe_import_p']/@visible" caption="Imported DLL options"/>
```

Остальные изменения те же, что описаны в первой статье.

Предлагаемые программы для перекомпиляции

В прошлой статье предлагалось исследовать на «перекомпилируемость» следующий перечень **GUI**-приложений для **XP**юши:

regedt32.exe	3584 байт
winver.exe	5632 байт
dcomcnfg.exe	6144 байт

winhlp32.exe	8192 байт
control.exe	8192 байт
eventvwr.exe	8704 байт
winmsd.exe	11776 байт

и другие.

Попробуем поработать с ними с помощью последней версии «Иды».

Перекомпиляция regedt32.exe

Программа **regedt32.exe** также является обёрткой для приложения **regedit.exe**. Размер ее меньше рассмотренного нами в прошлой статье **write.exe**. Поэтому никаких проблем с перекомпиляций не должно быть. И действительно, действуя, по уже известной схеме, мы получаем новый **exe**-модуль, который может как запуститься, так и нет. Все зависит от способа подгрузки отладочных символов. Если мы воспользуемся автоматической загрузкой символов из Интернета (при старте **IdaPro**), то получим представление части данных в виде кода (рис. 1).

```
0100101C ; Segment type: Pure code
0100101C ; Segment permissions: Read/Execute
0100101C _text           segment para public 'CODE' use32
0100101C                 assume cs:_text
0100101C                 ;org 100101Ch
0100101C                 assume es:nothing, ss:nothing, ds:_text, fs:nothing, gs:nothing
0100101C                 dd 2 dup(0)
01001024                 dd 3B7D845Bh, 0
0100102C                 dd 2, 1Dh, 1048h, 448h
0100103C ; ---------------------------------------------------------------------------
0100103C
0100103C ; const CHAR szFile
0100103C _szFile:                                ; DATA XREF: WinMain(x,x,x,x)+A↓o
0100103C                 jb      short near ptr loc_10010A2+1
0100103E                 db      65h
0100103E                 imul    esi, fs:[si+2Eh], 657865h
01001048                 dec     esi
01001049                 inc     edx
0100104A                 xor     [eax], esi
0100104A ; ---------------------------------------------------------------------------
0100104C                 dd 0
01001050                 dd 3B7D845Bh, 1, 65676572h, 32337464h, 6264702Eh
01001064                 db 2 dup(0)
```

Рис. 1. Данные, представленные как код, при автоматической загрузке символов из Интернет. Код после компиляции дает неправильную строку: **«reegdit.exe»**.

Несмотря на неправильное представление, наш ассемблерный код **regedt32.asm** скомпилируется без ошибок, но программа не будет выполнена, т.е. запуск приложения **regedit.exe** не произойдет. При анализе этой ситуации выясняется, что данные, представленные кодом, компилируются в строку «**reegdit.exe**». Естественно, приложения с таким именем на нашем компьютере нет, поэтому ничего запущено не будет.

Другое дело, если мы загрузим символы вручную, тогда данные будут представлены корректно (рис.2) и скомпилированный после этого код аккуратно вызовет нужное приложение (рис. 3). **Pdb**-файлы, для ручной подгрузки, можно взять либо из скачанных предварительно, с сайта **Microsoft**, упаковок отладочных символов, либо из временного каталога **\Temp\Ida**, куда «Ида» кладет скачанные автоматически символы из Интернета, либо утилитой **symchk.exe**, входящей в состав упоминавшегося уже пакета **«Debugging Tools for Windows (x86)»**, либо сторонними утилитами (вроде **symget.exe** из http://debuginfo.com/download/symget.zip) , предназначенными для индивидуальной загрузки из Интернета нужного нам **pdb**-файла.

```
0100101C ; ===========================================================================
0100101C
0100101C ; Segment type: Pure code
0100101C ; Segment permissions: Read/Execute
0100101C _text           segment para public 'CODE' use32
0100101C                 assume cs:_text
0100101C                 ;org 100101Ch
0100101C                 assume es:nothing, ss:nothing, ds:_text, fs:nothing, gs:nothing
0100101C                 dd 2 dup(0)
01001024                 dd 3B7D845Bh, 0
0100102C                 dd 2, 1Dh, 1048h, 448h
0100103C ; char szFile[12]
0100103C _szFile         db 'regedit.exe',0      ; DATA XREF: WinMain(x,x,x,x)+A↓o
01001048 aNb10           db 'NB10',0
0100104D                 align 10h
01001050                 dd 3B7D845Bh, 1, 65676572h, 32337464h, 6264702Eh
01001064                 db 2 dup(0)
```

Рис. 2. Данные, представленные как данные, при ручной загрузке отладочных символов.

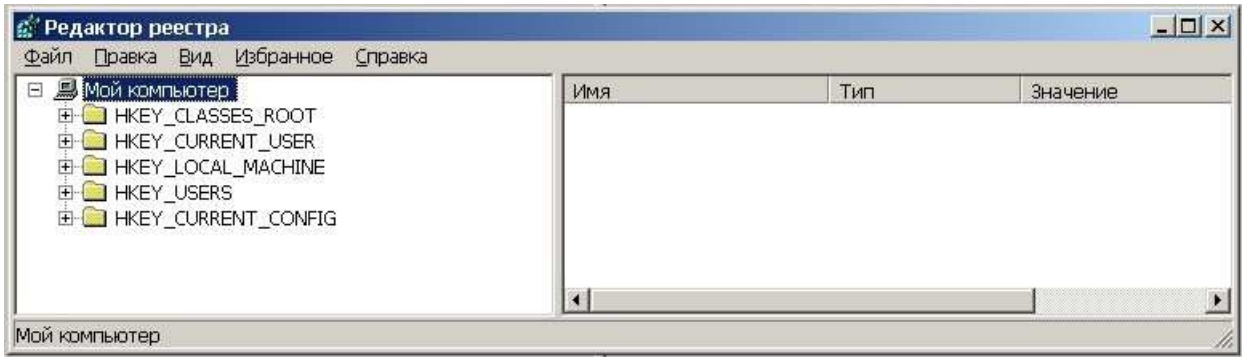

Рис. 3. Запущенное перекомпилированное приложение **regedt32.exe**.

Если мы, после автоматической загрузки отладочных символов, желаем перейти к их ручному использованию, а затем наоборот, то удобно разместить нужные нам три файла в отдельном каталоге «Иды», например с именем «**!**» и создать парочку пакетных **bat**-файлов, копирующих и удаляющих искомые файлы. Например:

```
:: Файл copy.bat
copy !\dbgeng.dll dbgeng.dll
copy !\dbghelp.dll dbghelp.dll
copy !\symsrv.dll symsrv.dll
```

и

```
:: Файл del.bat
del dbgeng.dll
del dbghelp.dll
del symsrv.dll
```

В принципе, достаточно удалять и копировать только один файл, при условии, что остальные два всегда присутствуют в корне «Иды».

Перекомпиляция winver.exe

Эта программа имеет тот же размер, что и **write.exe**, исследованная в первой статье. Посмотрим, какие у нее будут сюрпризы.
IdaPro, исходя из своих собственных, по-видимому, высоких побуждений, сформировал нам описание такой структуры (рис. 4).

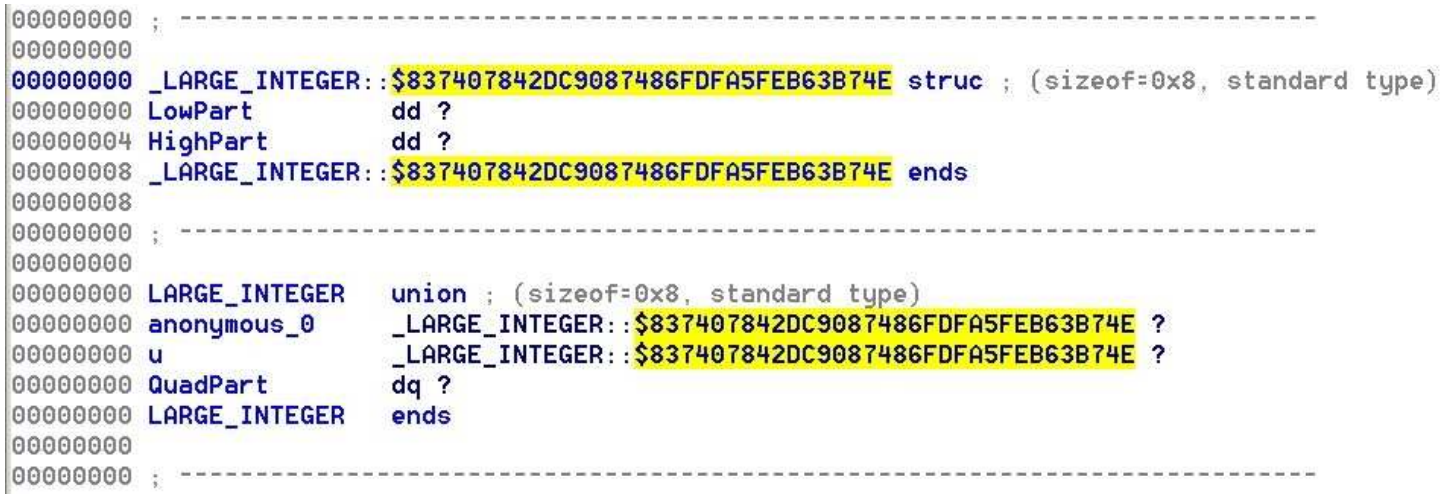

Рис. 4. Структура **_LARGE_INTEGER,** вызывающая «стоны» у компилятора .

Однако, компилятору, как и нам ничего не известно о благородных порывах «Иды» и он тупо ругается на непонятки в определении этих структур . Чтобы не раздражать лишний раз компилятор, мы поправим «Иду» и запишем, более понятные всем нам определения:

```
; ---------------------------------------------------------------------------

_LARGE_INTEGER struc ; (sizeof=0x8, standard type)
   LowPart                 dd ?
   HighPart                dd ?
_LARGE_INTEGER ends

; ---------------------------------------------------------------------------

LARGE_INTEGER   union ; (sizeof=0x8, standard type)
   anonymous_0             _LARGE_INTEGER <>
   u                       _LARGE_INTEGER <>
   QuadPart                dq ?
LARGE_INTEGER   ends

; ---------------------------------------------------------------------------
```

Это совсем другое дело, мысленно ответил нам компилятор . По крайней мере, такую запись он понимает.

Компилируем. Получаем новую ошибку в строке:

```
FileTime = FILETIME ptr -24Ch
```

Ну, это проблема с регистром символов. Решается просто. Добавляем строку

```
option casemap:none
```

в начало файла **headers.inc** и имена функций и переменных в ассемблерном коде становятся чувствительными к регистру.

Также добавляем в этот файл команду

```
includelib Lib\user32.lib
```

поскольку из этой библиотеки в нашем коде вызываются функции. Имеющуюся там строку

```
includelib Lib\msvcrt.lib
```

можно закомментировать или удалить (а можно и оставить), так как из этой библиотеки, в дизассемблерном коде **winver.exe v. 5.1**, ничего не вызывается. А вот в версии **5.2** этой программы (из **Win2003 Server**, которую мы переименуем в **winver2.exe**) эта библиотека используется. Заметим, что библиотечные ***.lib** файлы мы берем либо из пакета **MASM32 v. 10**, либо из **MS Visual Studio**.

Мы видим, что в ресурсах данной программы появился новый объект – **xml**-манифест (рис. 5).

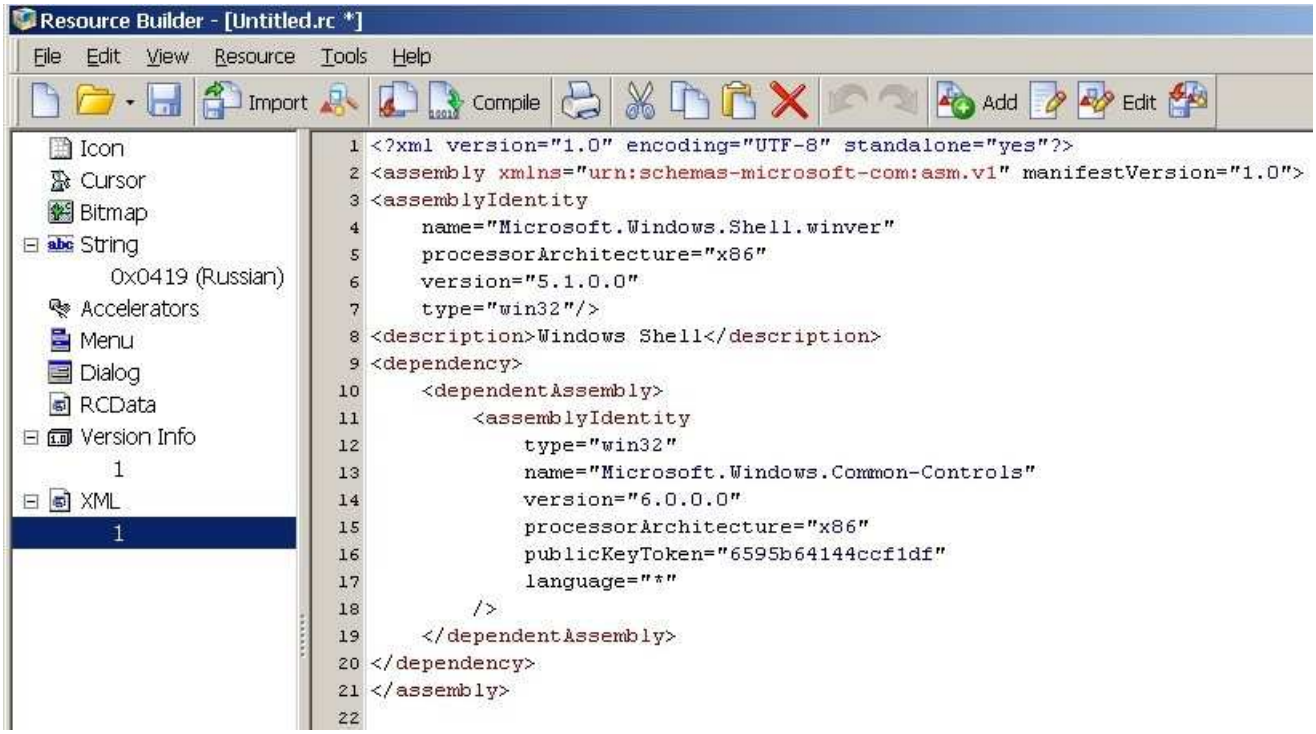

Рис. 5. Манифест приложения **winver.exe**.

При компиляции этого манифеста в файл ресурсов мы получаем код вида:

```
1 24
MOVEABLE PURE LOADONCALL DISCARDABLE
LANGUAGE LANG_RUSSIAN, 1
BEGIN
'3C 3F 78 6D 6C 20 76 65 72 73 69 6F 6E 3D 22 31 '
. . .
'3C 2F 61 73 73 65 6D 62 6C 79 3E 0D 0A '
END
```

В таком виде этот код мало отличается от бинарного и неудобен для просмотра и редактирования. К тому же компилятор ресурсов ругается на подобное представление данных. Сейчас мы полностью разделяем негодование компилятора 😔 и после небольших поисков в Интернете обнаруживаем более подходящую для нас форму записи:

```
1 24
MOVEABLE PURE LOADONCALL DISCARDABLE
"winver.xml"
```

Здесь **winver.xml** – файл, содержимое которого мы скопировали непосредственно из **Resource Builder** (рис. 5). В таком виде уже значительно удобней иметь дело с ресурсом манифеста.

Другие ошибки компиляции не выходят за пределы уже известных нам. В итоге подгружаем ресурсы исходной программы к вновь создаваемой программе и окончательно собираем **exe**-шник.

Скомпилированная программа чудесным образом запускается, радуя нас своим непритязательным видом (рис. 6).

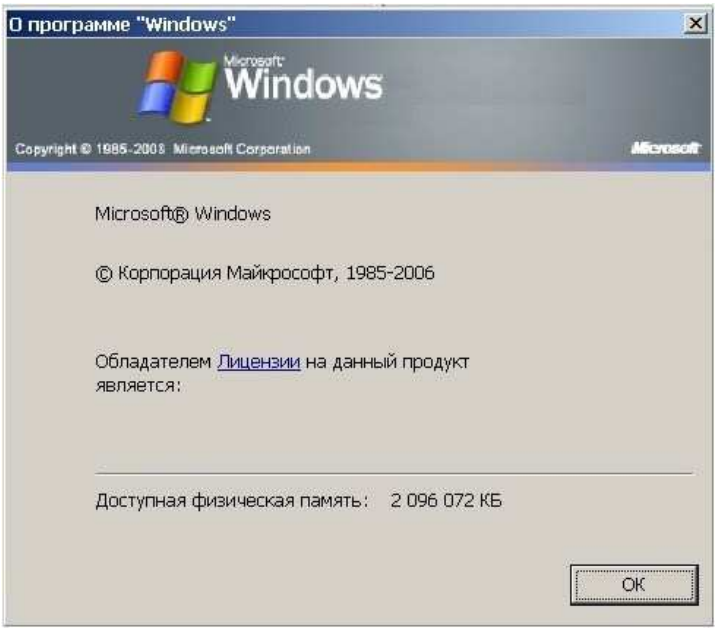

Рис. 6. Запущенное перекомпилированное приложение **winver.exe**.

Заметим еще, что возможно в скрипт ресурсов **winver.rc** придется добавить строку

```
#define LANG_RUSSIAN                0x19
```

к тем описаниям, которые мы делали в предыдущих скриптах.

Таким образом, все превосходно работает, но остается маленький нюанс. Запуск перекомпилированной программы **winver.exe** из **XP** под **Windows 2003,** при выходе из нее, не выгружает экземпляр этой программы из памяти, что легко можно определить вызвав **Диспетчер задач** по **Ctrl+Alt+Del**. В самой же **XP**юше все прекрасно выгружается. Это послужило поводом рассмотреть аналог этой программы из **Win2003** (**winver2.exe**) в обеих рассматриваемых нами системах.

Серверная версия этой программы имеет размер **6656** байт. Объем рассматриваемого нами чистого ассемблерного кода превышает предыдущий вариант в более, чем два раза. Кроме того, вызывается множество функций из **msvcrt.dll**, чего нет в **XP**-шной версии. Так что различия существенные, хотя ресурсы отличаются непринципиально (в основном, языком строковых данных). Тем не менее, перекомпиляция прошла успешно, в рамках уже известных нам исправлений. Результат работы тот же, что и на рис. 6. Однако завершение программы выполняется абсолютно корректно в обеих версиях операционных систем, что не может не радовать. Вы можете посмотреть различия в коде в прилагаемом архиве.

Перекомпиляция dcomcnfg.exe

Интересно, что каждая новая программа привносит в процесс ее декомпиляции и новой компиляции свои неповторимые нюансы. Но легче решать проблемы по одной, чем все сразу. Так что продолжим наши исследования способности **IdaPro 5.7** генерить качественный дизассемблерный код из исходных **exe**-программ. Пока это у него получается неплохо, но посмотрим, что будет дальше.

Выполнение этой программы эквивалентно командной строке (может быть с дополнительными параметрами):

```
mmc.exe C:\WINDOWS\System32\Com\comexp.msc
```

Эта команда запускает консоль **MMC (Microsoft Management Console)** с открытием сохраненной консоли управления **comexp.msc (Component Services)**. Как это типично для малых **GUI**-шных программ **Windows**, мы опять имеем дело с некой обёрткой для внешних программ. Однако поскольку мы практически не интересуемся содержимым рассматриваемых **exe**-файлов, по крайней мере, больше,

чем это нужно для целей перекомпиляции, то это не суть важно. Главное, это проблемные нюансы рекомпиляции и способы их разрешения. Ну, а уже собственно цель дизассемблирования / реассемблирования для конкретно выбранного приложения – прерогатива нашего читателя, которого мы просто обучаем работать с великолепным программным продуктом **IdaPro**, прежде всего, как дизассемблером, качество которого определяется способностью перекомпилированного кода нормально работать.

Несмотря на уже известный нам тип исследуемого **exe**-шника, данная программа также припасла свои сюрпризы. Повторяя процесс получения нужного нам ассемблерного кода **dcomcnfg.asm**, мы отметим только новые для нас аспекты.

1. Оказывается «Ида» не очень корректно представляет длинные строки комментариев. Слишком длинные строки она просто режет на куски, но символ комментария, точку с запятой, не вставляет в начало новой строки. Впрочем, это не большая проблема и легко «лечится». Просто, либо добавляем новый символ комментария, либо объединяем соответствующие строки.

Более интересными оказываются следующие нюансы дизассемблирования.

2. «Ида» ссылается на макрос, который нигде не определен. Т.е. он добавляет в листинг команду включения несуществующего в поставке «Иды» файла:

```
include uni.inc ; see unicode subdir of ida for info on unicode
```

которой должен содержать, по крайней мере, описание макроса **unicode**, получающего на вход три параметра, иначе строки вида:

```
unicode 0, <DCOMCnfg>, 0
. . .
unicode 0, <%s %s\com\comexp.msc>, 0
. . .
unicode 0, <%s\mmc.exe>, 0
```

совершенно не понравятся компилятору.

Легко предположить, что смысл действия макроса заключается в представлении **ASCII**-строки в угловых скобах как строки формата **Unicode**. А оставшиеся параметры могут обозначать кодовую страницу и символ завершения **Unicode**-строки. То, что это действительно так, подтверждает, соответствующий макрос из поставки бесплатной рабочей (**Freeware**) версии **IdaPro 4.9**, которую можно также скачать на сайте **Ильфака** (http://hex-rays.com/idapro/idadown.htm). Вот его описание:

```
;===========================================================================

;comment *
unicode macro page, string, zero
  irpc c, <string>
    db '&c', page
  endm

  ifnb <zero>
    dw zero
  endif
endm
;*

;===========================================================================
```

Этот текст можно сохранить в **uni.inc**, на который дана ссылка в листинге, либо в наш файл **headers.inc** и его подгружать, как мы это делали ранее. Кстати, туда еще следует добавить строку:

```
includelib Lib\ntdll.lib
```

так как **dcomcnfg** использует эту библиотеку.

Однако, все проделанные процедуры не приведут нас к нужному успеху, поскольку данный макрос имеет некоторые ограничения в работе. А именно, он не может корректно обработать спецсимволы (в приведенных примерах это знак процента "**%**"). Этот не очень очевидный факт может привести к критике как макроса **unicode**, так и автора данного дизассемблера, не поставившего файла **uni.inc**. Пример подобной дискуссии можно посмотреть на форуме: http://www.cracklab.ru/f/index.php?action=vthread&forum=1&topic=12507&page=4 , начиная с сообщения который создал **Neuch____** .

Я не знаю, можно ли написать макрос, аналогичный **unicode**, который корректно обрабатывает спецсимволы **MASM**'а в передаваемой ему строке. Однако, оказывается, можно заставить спецсимвол быть простым литералом, поставив перед ним восклицательный знак "**!**". Вот, что по этому поводу говорится в документации **Borland International "Turbo assembler. Quick Reference Guide", © 1990, 1991**:

«For **Ideal** and **MASM** modes. . .

!*character*

Treats ***character*** literally, regardless of any special meaning it might otherwise have.»

И, действительно, записав приведенные выше строки в виде:

```
unicode 0, <!%s !%s\com\comexp.msc>, 0
. . .
unicode 0, <!%s\mmc.exe>, 0
```

мы получим корректный результат работы нашего макроса, по крайней мере, компилятор не будет на него ругаться. Это, конечно, не единственное решение, как следует из обсуждения на указанном выше форуме, но достаточно приемлемое.

3. Исправив эти и другие, описанные ранее, ошибки компиляции дизассемблерного кода, мы наконец-то получим новый **exe**-файл (обратим внимание, что этот **exe**-шник собирается на стандартной базе **ImageBase = 0x400000**). Тем не менее, проблемы на этом не закончились. Наше приложение запускается, но ничего не делает. С такого рода ошибками компиляции бороться труднее всего. Универсальное решение – запускать одновременно две копии программы (новой и старой) в отладчике и пошагово наблюдать за их работой. Как отладчик **IdaPro** мне непривычен, поэтому я предпочитаю старую знакомую – **Олю Дебаговую** (**OllyDbg v. 1.10** с плагинами). В Интернете легко найти множество ее версий и различных плагинов, а также русскоязычную документацию.

Как правильно отлаживать программы – это отдельный большой разговор. Здесь мы не будем заострять на этом внимание. В принципе, информации на эту тему опубликовано достаточно. Приведем конечный результат для локализованной проблемы.

Дело в том, что **dcomcnfg**, в своей секции данных имеет значение **0x00401385**, расположенное по адресу **.data:00402004** (см., прилагаемый листинг), которое, оказывается, используется как адрес функции,

```
.data:00402000 ; ===========================================================
.data:00402000
.data:00402000 ; Segment type: Pure data
.data:00402000 ; Segment permissions: Read/Write
.data:00402000 _data          segment para public 'DATA' use32
.data:00402000                assume cs:_data
.data:00402000                ;org 402000h
.data:00402000 ___xc_a        db    0             ; DATA XREF: _mainCRTStartup+102□o
.data:00402001                db    0
.data:00402002                db    0
```

```
.data:00402003                 db    0
.data:00402004                 db  85h ; A
.data:00402005                 db  13h
.data:00402006                 db  40h ; @
.data:00402007                 db    0
.data:00402008 ___xc_z         db    0             ; DATA XREF: _mainCRTStartup+FD□o
.data:00402009                 db    0
.data:0040200A                 db    0
.data:0040200B                 db    0
.data:0040200C ___xi_a         db    0             ; DATA XREF: _mainCRTStartup+CC□o
.data:0040200D                 db    0
.data:0040200E                 db    0
.data:0040200F                 db    0
.data:00402010 ___xi_z         db    0             ; DATA XREF: _mainCRTStartup+C7□o
.data:00402011                 db    0
.data:00402012                 db    0
.data:00402013                 db    0
```

получаемой, косвенным образом, управление. Поэтому «Ида» этот «финт ушами» «честно» проморгал 🙂. Таким образом, данное **смещение** оценено «Идой», как **значение** или **константа**. Такого рода ошибки из разряда трудно обнаруживаемых. На больших программах это может быть источником постоянной головной боли 🙂.

См. для примера, статьи **Криса Касперски «Секреты ассемблирования дизассемблерных листингов»** (http://www.insidepro.com/kk/150/150r.shtml) и **«Линковка дизассемблерных файлов»** (http://www.insidepro.com/kk/151/151r.shtml)

Естественно, что в таком виде данное значение сохраниться после перекомпиляции, а адреса функций изменятся. Однако переход по фиксированному адресу может привести к непредсказуемым последствиям. В данном случае программа просто преждевременно заканчивает свою работу. Теперь, когда мы поняли причину ошибки, легко ее исправить. Нужно просто указать в нужном месте секции данных не абсолютное значение, а относительное. Для этого выясним, какая функция в исходной программе **dcomcnfg.lst** расположена по адресу **0x00401385** (см. прилагаемый листинг).

```
.text:00401385 ; =============== S U B R O U T I N E =======================================
.text:00401385
.text:00401385 ; Attributes: bp-based frame
.text:00401385
.text:00401385 ___security_init_cookie proc near
.text:00401385
.text:00401385 PerformanceCount= LARGE_INTEGER ptr -10h
.text:00401385 SystemTimeAsFileTime= _FILETIME ptr -8
.text:00401385
.text:00401385                 mov     edi, edi
.text:00401387                 push    ebp
.text:00401388                 mov     ebp, esp
```

Как видим, это функция **___security_init_cookie**. Поэтому перепишем искомый участок из секции данных в виде:

```
; ===========================================================================

; Segment type: Pure data
; Segment permissions: Read/Write
_data           segment para public 'DATA' use32
                assume cs:_data
                ;org 402000h
___xc_a         db    0             ; DATA XREF: _mainCRTStartup+102□o
                db    0
                db    0
                db    0

                dd offset ___security_init_cookie
```

```
___xc_z     db  0           ; DATA XREF: _mainCRTStartup+FD□o
            db  0
            db  0
            db  0
```

Компилируем и, к нашей радости, программа корректно запустилась (рис. 7).

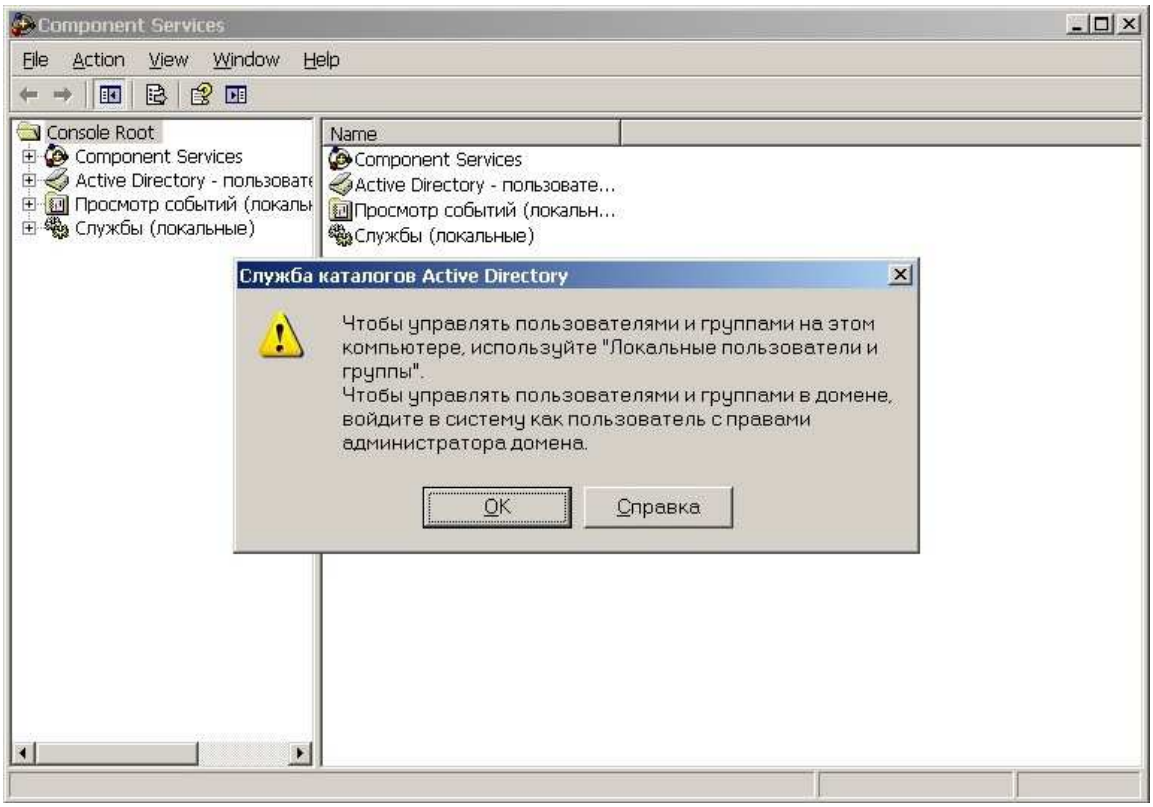

Рис. 7. Запущенное перекомпилированное приложение **dcomcnfg.exe**.

Заметим, что из ресурсов присутствовала только версионная информация, которая содержала новые определения, взятые нами из **MS VC++**:

```
#define VOS__WINDOWS32          0x00000004L
#define VFT_DLL                 0x00000002L
```

Кстати, маскируя смещения под константы можно весьма существенно усложнить процесс перекомпиляции программ, тем самым повысив их устойчивость к полному восстановлению. Понятно, что это далеко не единственный возможный способ защиты машинного кода.

Перекомпиляция winhlp32.exe

Эту программу перекомпилировать было одно удовольствие. Новых нюансов практически не было, если не считать переопределение переменной **String1** в секции кода

```
; =============== S U B R O U T I N E =======================================
; Attributes: noreturn bp-based frame fpd=6Ch
_main proc near             ; CODE XREF: _mainCRTStartup+120□p
String1 = word ptr -6ACh
```

и в секции данных

```
; ===========================================================================
; Segment type: Pure data
; Segment permissions: Read/Write
_data       segment para public 'DATA' use32
            assume cs:_data
```

```
            ;org 1002000h
. . .
; WCHAR String1
String1         dw 0                    ; DATA XREF: GetVersionDatum(ushort *)+18□o
```

Эта фишка уже из разряда приколов «Иды» 🙂. В самом деле, неужели трудно отслеживать уникальность используемых переменных? В данном случае, проблема легко решается, однако при декомпиляции больших проектов это может вызвать явно лишние затруднения.

Здесь мы можем различить сдвоенную переменную по типу обращения к ней (если не вникать в суть алгоритма). Когда обращение идет к переменной из секции данных, то должен применяться, как правило, оператор **offset**. По нему то мы и попробуем различить одинаково обзываемые переменные. И, действительно, это нам помогло сейчас. Довольно быстро находим единственную подобную строку

```
push    offset String1  ; lpString1
```

в которой делаем замену на

```
push    offset String1_  ; lpString1
```

и добавляем символ нижнего подчеркивания справа, в соответствующие переменные из секции данных

```
; WCHAR String1_
String1_        dw 0                    ; DATA XREF: GetVersionDatum(ushort *)+18□o
```

Теперь все Ok и наша перекомпилированная программы прекрасно запускается (рис. 8).

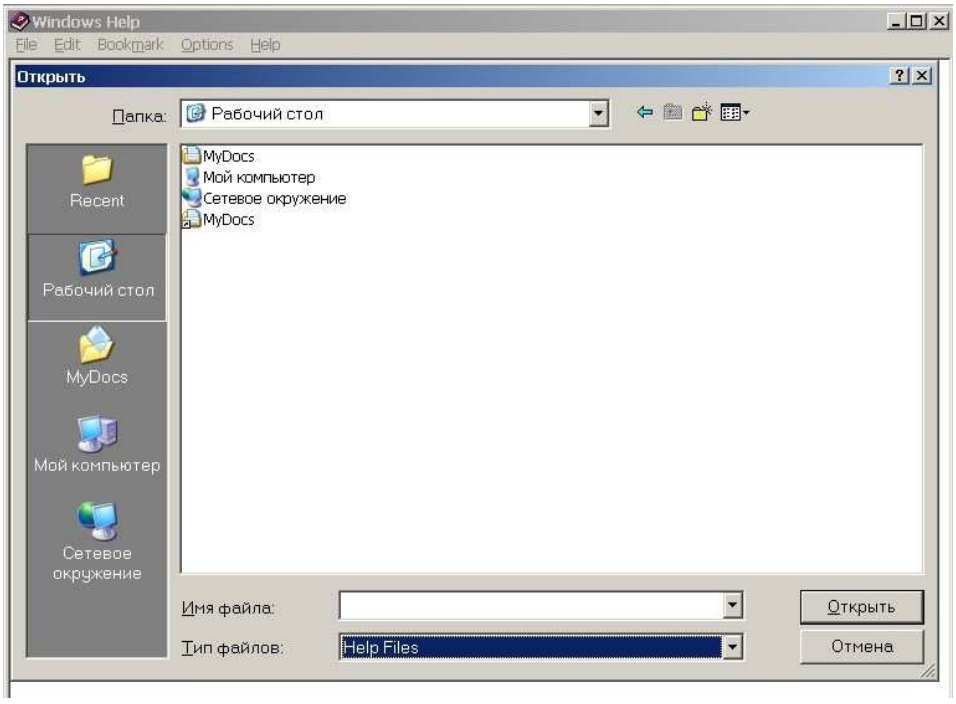

Рис. 8. Запущенное перекомпилированное приложение **winhlp32.exe**.

Перекомпиляция control.exe

В области структур данных мы исправили дубликаты имен и заменили инициализацию вида

```
anonymous_0     _SHELLEXECUTEINFOW_    ?
```

на

```
anonymous_0     _SHELLEXECUTEINFOW_    <>
```

Макрос **unicode** также «спотыкается» на символе кавычек <">, который мы тоже «экранировали» восклицательным знаком <!"> .

«Ида» любит использовать в качестве имен переменных служебные слова, в данном случае: **Type**. Заменяя его на слово **Type_**, сразу устраняем часть проблем.

Далее «Ида» также практикует общие имена в полях структур и в параметрах функций. Хотя эти имена формально не пересекаются, компилятору они не нравятся. Например, поля **wParam** и **lParam** в глобальной структуре:

```
MSG             struc ; (sizeof=0x1C, standard type)
    Hwnd            dd ?                    ; offset
    Message         dd ?
    wParam          dd ?
    lParam          dd ?
    time            dd ?
    pt              POINT <>
MSG             ends
```

и аналогичные переменные в теле функции:

```
; int __stdcall DummyControlPanelProc(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)
_DummyControlPanelProc@16 proc near     ; DATA XREF: _CreateDummyControlPanel(x)+1A□o

ExecInfo        = _SHELLEXECUTEINFOW ptr -3Ch
hWnd            = dword ptr  8
Msg             = dword ptr  0Ch
wParam          = dword ptr  10h
lParam          = dword ptr  14h
```

Однако ничего не поделаешь, имена эти нужно разделять, что мы и сделаем.

Далее, после удаления ненужной нам секции импорта, мы столкнемся с такой проблемой, как неопределенный символ **dword_100185C**. Смотрим исходный листинг программы и видим следующую картину:

```
.text:01001858 _c_aCompatCpls  dd offset aDesktop      ; "DESKTOP"
.text:0100185C                 dd 0FFFFFFFFh, 1001C1Ch, 0 ; DATA XREF: WinMainT(x,x,x,x)+71□o
.text:01001868                 dd offset aColor        ; "COLOR"
```

Ну, раз «Ида» не захотела вставить данную метку в положенное ей место, то придется нам выполнить эту работу за нее  . Получаем следующий результат.

```
_c_aCompatCpls  dd offset aDesktop      ; "DESKTOP"
dword_100185C   dd 0FFFFFFFFh, 1001C1Ch, 0 ; DATA XREF: WinMainT(x,x,x,x)+71□o
                dd offset aColor        ; "COLOR"
```

Ну вот, почти все хорошо, только линковщик жалуется, что не находит символ **__imp__IsOS@4**. И действительно, в библиотеке **shlwapi.dll** (откуда импортируется «Идой» эта функция) нет даже подстроки **IsOS**. Как выяснилось **shlwapi.dll** экспортирует эту функцию только по номеру, ординал которой равен **437**. В отладочных символах, которые использует «Ида», имя этой функции определенно, но в самой **dll**-ке этого имени нет, только ординал. Так что у нас есть два варианта. Либо в ассемблерном коде делать вызов функции **__imp__IsOS@4** по ординалу (фактически по явному адресу, который соответствует **437**-й функции), либо изменить файл **shlwapi.lib** так, чтобы данному имени соответствовал **437**-й ординал. Заметим, что, в отличие от экспортируемых имен, экспортируемые ординалы редко соответствуют одной и той же функции в разных версиях **dll**-ных библиотек. Впрочем, деваться нам некуда, придется работать с ординалами, в расчете на то, что **Микрософт** для важных функций не будет менять ординалы в различных версиях своей ОСи, что впрочем, всегда надо проверять.

Мы предпочтем второй вариант действий, так как менять существенно гигантский ассемблерный код от «Иды» нас, в общем случае, не прикалывает ☺. Следует только помнить, что генерить собственный **lib**-файл придется для всех функций, соответствующей библиотеки, используемых в проекте. В нашем случае, **shlwapi.dll** использует только одну единственную функцию, поэтому построить вручную нужный **lib**-файл для нее одной труда не составит. Однако, если проект будет содержать сотни импортируемых функций, то придется уже искать какие-то средства автоматизации построения собственных **lib**-файлов. Но об этом речь будет идти, скорее всего, в следующей статье. Сейчас мы просто построим **lib**-файл для одной функции **_IsOS@4** (парную ей функцию **__imp__IsOS@4** компилятор создает автоматически). Помним, что при использовании экспортируемой функции в ассемблерном коде, лидирующий символ подчеркивания нужно убирать.

Итак, в нашем случае, мы должны создать текстовый файл **shlwapi.def** с содержимым:

```
LIBRARY "shlwapi.dll"
EXPORTS
_IsOS@4      @437
```

и затем скомпилировать его с помощью утилиты **lib.exe** или **polib.exe**, входящих в состав **MASM32**. Например, с помощью командного файла **def2lib.bat**:

```
SET FILENAME=shlwapi

:: Создание lib файла из def файла
:: Bin\lib /DEF:%FILENAME%.def /OUT:%FILENAME%.lib /MACHINE:X86 > a.a
Bin\polib /DEF:%FILENAME%.def /OUT:%FILENAME%.lib /MACHINE:X86 > a.a

del *.exp
```

Только при использовании **lib.exe** надо писать в **shlwapi.def**:

```
IsOS@4      @437
```

Полученный файл **shlwapi.lib** используем вместо оригинального и таким образом, преодолеваем последнее препятствие для успешного запуска **control.exe** (рис. 9).

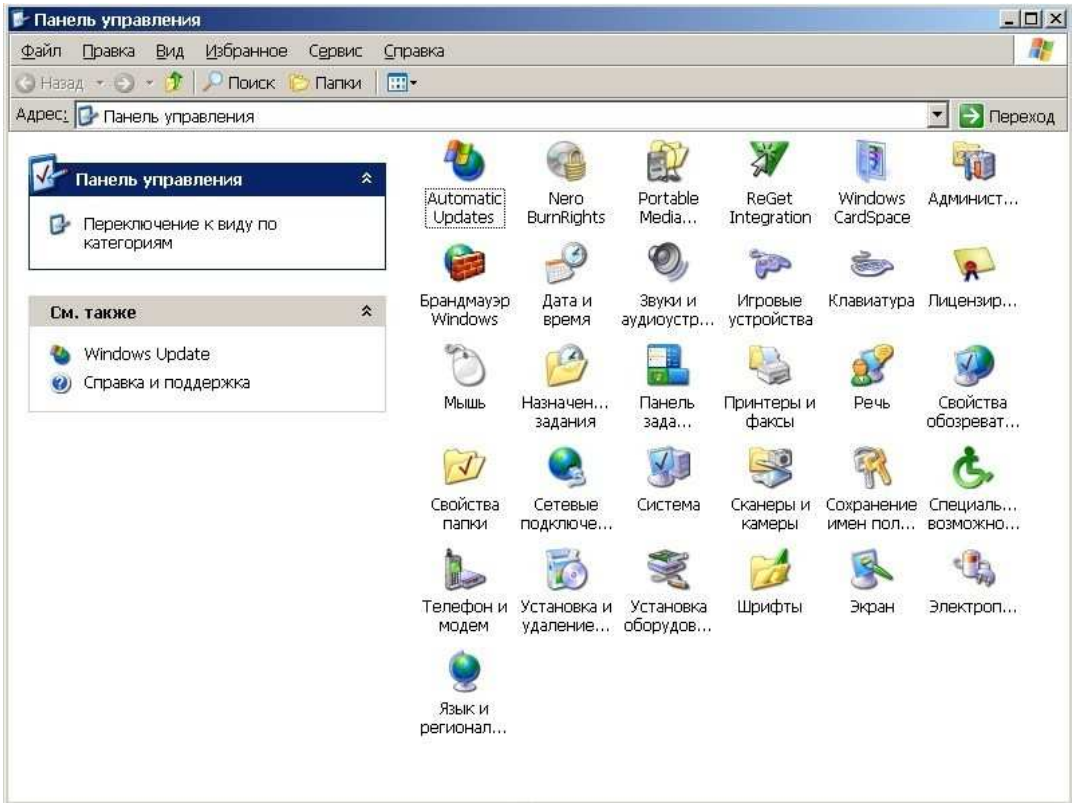

Рис. 9. Запущенное перекомпилированное приложение **control.exe**.

Перекомпиляция eventvwr.exe

Среди ресурсов данной программы появился диалог с контролами, что потребовало огромного количества символьных констант, которые пришлось взять из **MS VS C++**. Поскольку отслеживать индивидуально каждую константу было утомительно, то мы просто вставили целый блок этих констант, с запасом, так сказать, на будущее .

Другой, более существенный, нюанс оказался связанным с файлом **msvcrt.lib**. «Ида» нам выдала следующие определения:

```
; __declspec(dllimport) void * __cdecl operator new(unsigned int)
extrn _imp_??2@YAPAXI@Z:dword  ; DATA XREF: operator new(uint)r
; __declspec(dllimport) void __cdecl operator delete(void *)
extrn _imp_??3@YAXPAX@Z:dword  ; DATA XREF: operator delete(void *)r
```

Эти имена есть в файле **msvcrt.dll** (без авто префикса **_imp_**). Действительно, утилита **dumpbin.exe** из **MS VS C++** выдает нам:

```
ordinal hint      RVA              name
   18    12    0001A971    ??2@YAPAXI@Z
   19    13    0001A9A7    ??3@YAXPAX@Z
```

Но эти имена не описаны в **msvcrt.lib**. Конечно, мы можем повторить процедуру из предыдущего случая и самостоятельно создать соответствующий **def**-файл и скомпилировать собственный **lib**-файл. Хотя работы нам будет уже побольше, так как **eventvwr.exe** импортирует **17** функций из **msvcrt.dll**.

Сейчас мы вынуждены проделать эту работу вручную, но, очевидно, следует задуматься об автоматизации процесса построения собственных (универсальных) **def**-файлов и соответствующих им файлов библиотек. Для это возьмем секцию импорта (без комментариев) для **msvcrt.dll** из **eventvwr.lst**

```
;
; Imports from msvcrt.dll
;
extrn _imp___except_handler3:dword
extrn _imp___controlfp:dword
extrn _imp____set_app_type:dword
extrn _imp____p__fmode:dword
extrn _imp____p__commode:dword
extrn _imp___adjust_fdiv:dword
extrn _imp____setusermatherr:dword
extrn _imp___initterm:dword
extrn _imp____getmainargs:dword
extrn _imp____initenv:dword
extrn _imp__exit:dword
extrn _imp___cexit:dword
extrn _imp___XcptFilter:dword
extrn _imp___exit:dword
extrn _imp___c_exit:dword
extrn _imp_??2@YAPAXI@Z:dword
extrn _imp_??3@YAXPAX@Z:dword
```

Из этого листинга получим следующий код для msvcrt.def:

```
LIBRARY "msvcrt.dll"
EXPORTS
_except_handler3
_controlfp
__set_app_type
__p__fmode
__p__commode
_adjust_fdiv
__setusermatherr
_initterm
```

```
__getmainargs
__initenv
exit
_cexit
_XcptFilter
_exit
_c_exit
??2@YAPAXI@Z
??3@YAXPAX@Z
```

Эти определения корректно компилируется в **lib**-файл как с помощью **lib.exe**, так и **polib.exe**. Заменяя исходный файл **msvcrt.lib** вновь созданным, добиваемся успешной компиляции **msvcrt.asm** в **msvcrt.exe**. Результат работы программы показан на рис. 10.

Из проделанной работы можно сделать вывод, что надеяться на чужие **lib**-файлы особенно не стоит. В мало-мальски серьезном проекте вполне может возникнуть ситуация, когда определения из стандартного библиотечного файла будут неполными. Это потребует необходимости самостоятельного изготовления более подходящих файлов библиотек. Очевидно, что большую помощь в этом нам может оказать **IdaPro**, особенно с загруженными отладочными символами.

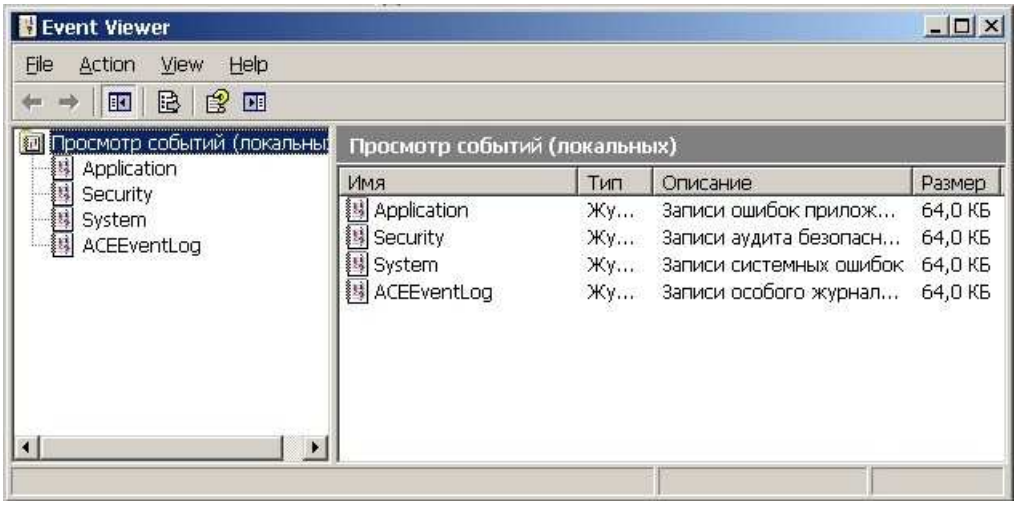

Рис. 10. Запущенное перекомпилированное приложение **eventvwr.exe**.

Перекомпиляция winmsd.exe

Наконец-то мы подошли к последнему файлу из определенного нами списка простейших **GUI**-шных **exe**-программ, взятых в основном из **Windows XP sp. 3**.

winmsd.exe это облегченная версия программы **msinfo32.exe**, которая выдает сведения о системе. Трудно понять, зачем нужно такое «разделение труда» между этими программами, но да Майкрософту виднее .

Файл ресурсов в данной программе достаточно прост, только компилятор ресурсов не понял строку:

```
12, "Информация предоставлявшаяся ранее программой WinMSD теперь отображается программой \"Сведения о системе\" (msinfo32.exe). Отобразить информацию по использованию msinfo32.exe?"
```

Очень похоже, что ему не понравились внутренние кавычки. Сейчас нет особого желания выяснять, как показывать внутренние двойные кавычки в строке ресурсов, поэтому мы просто заменим их на одинарные:

```
12, "Информация предоставлявшаяся ранее программой WinMSD теперь отображается программой 'Сведения о системе' (msinfo32.exe). Отобразить информацию по использованию msinfo32.exe?"
```

В коде «Ида» опять использовала служебное слово **Type** для локальной переменой в функции, а в качестве имени поля структуры **_msExcInfo** служебное слово **Proc**. Поэтому заменим их на слова **Type_** и **Proc_** соответственно чтобы избежать недовольства компилятора  .

Поле

```
Info          _msExcInfo 0 dup(?)
```

структуры **_msExcept** исправим на

```
Info          _msExcInfo 0 dup(<>)
```

что для компилятора также более предпочтительно.

Далее, «Ида» определила структуру

```
_msExcept7     struc  ; (sizeof=0x1C)
   Magic          dd ?                ; base 16
   Count          dd ?                ; base 10
   InfoPtr        dd ?                ; offset
   CountDtr       dd ?                ; base 10
   DtrPtr         dd ?                ; offset
   _unk           dd 2 dup(?)
_msExcept7     ends
```

в которой два последних поля не распознаны. Затем она инициализирует эти два поля как одно, типа

```
stru_1002684    _msExcept7 <19930520h, 1, offset stru_100267C, 0, 0, 0>
```

что естественно не нравится компилятору. Выход прост. Инициализируем последнее поле как структуру

```
stru_1002684    _msExcept7 <19930520h, 1, offset stru_100267C, 0, 0, <0>>
```

везде, где встречаются подобные конструкции. Это избавит нас от нескольких ошибок.

Одна ошибка компилятора очень интересна. Ему неизвестна инструкция **setalc** в коде

```
dd offset ?Dump@CObject@@UBEXAAVCDumpContext@@@Z  ; CObject::Dump(CDumpContext &)
align    10h
xchg     eax, esp
adc      eax, 1FE20100h
add      [ecx], al
push     16h
add      [ecx], al
db       3Eh
push     ss
add      [ecx], al
cmp      eax, 3E010016h
push     ss
add      [ecx], al
fcomp    qword ptr [edi]
add      [ecx], al
setalc
pop      ds
add      [ecx], al
rcr      byte ptr [edi], 1
add      [ecx], al
retf     1Fh
; ---------------------------------------------------------------------------
db       1
dd offset ?GetTypeInfoCount@CCmdTarget@@UAEIXZ  ; CCmdTarget::GetTypeInfoCount(void)
```

Однако код этот весьма подозрителен, на него нет перехода, он лежит среди данных и достаточно бессмыслен сам по себе. Так что делаем вполне законное предположение, что это не код а данные, не распознанные «Идой». И, действительно, преобразуя в «Иде» (клавишей – командой **«D»**) этот псевдокод в данные, получаем более осмысленные выражения:

```
dd offset ?Dump@CObject@@UBEXAAVCDumpContext@@@Z ; CObject::Dump(CDumpContext &)
align 10h
dd offset ??_R4CMSInfoApp@@6B@ ; const CMSInfoApp::`RTTI Complete Object Locator'  ; const
CMSInfoApp::`vftable'
??_7CMSInfoApp@@6B@ dd offset ?GetRuntimeClass@CWinApp@@UBEPAUCRuntimeClass@@XZ
    ; DATA XREF: CMSInfoApp::CMSInfoApp(void)+A□o
    ; CWinApp::GetRuntimeClass(void)
dd offset ??_GCMSInfoApp@@UAEPAXI@Z ; CMSInfoApp::`scalar deleting destructor'(uint)
dd offset ?Dump@CObject@@UBEXAAVCDumpContext@@@Z ; CObject::Dump(CDumpContext &)
dd offset ?AssertValid@CObject@@UBEXXZ ; CObject::AssertValid(void)
dd offset ?Dump@CObject@@UBEXAAVCDumpContext@@@Z ; CObject::Dump(CDumpContext &)
dd offset ?OnCmdMsg@CCmdTarget@@UAEHIHPAXPAUAFX_CMDHANDLERINFO@@@Z
    ; CCmdTarget::OnCmdMsg(uint,int,void *,AFX_CMDHANDLERINFO *)
dd offset ?OnFinalRelease@CCmdTarget@@UAEXXZ ; CCmdTarget::OnFinalRelease(void)
dd offset ?IsInvokeAllowed@CCmdTarget@@UAEHJ@Z ; CCmdTarget::IsInvokeAllowed(long)
dd offset ?GetDispatchIID@CCmdTarget@@UAEHPAU_GUID@@@Z ; CCmdTarget::GetDispatchIID(_GUID *)
dd offset ?GetTypeInfoCount@CCmdTarget@@UAEIXZ ; CCmdTarget::GetTypeInfoCount(void)
```

Далее, как и в прошлом случае, не все функции **msvcrt.dll** определены в стандартном **msvcrt.lib**. Т.е. нам опять нужно создавать собственный файл **msvcrt.def** и делать на его основе **lib**-файл. Предыдущий **def**-файл годится не полностью, так как количество функций импортируемых из **msvcrt.dll** возрастает до **24**. Приведем новую версию **msvcrt.def**:

```
LIBRARY "msvcrt.dll"
EXPORTS
??1type_info@@UAE@XZ
__dllonexit
_onexit
_controlfp
_except_handler3
?terminate@@YAXXZ
__set_app_type
__p__fmode
__p__commode
_adjust_fdiv
__setusermatherr
exit
_cexit
_XcptFilter
_exit
_c_exit
wcstombs
_execlp
_waccess
_CxxThrowException
__CxxThrowException@8 = _CxxThrowException
__CxxFrameHandler
_initterm
__wgetmainargs
_wcmdln
```

Заметим, что «Ида» использует вызов функции **__CxxThrowException@8** (с авто префиксом **_imp_**), тогда как в **dll**-ке присутствует только имя **_CxxThrowException**. Конечно, мы можем определить в своем **inc**-файле выражение вида:

```
__CxxThrowException@8 equ _CxxThrowException
```

как это делает знаменитый **Iczelion**, но мы предпочитаем отказаться от идеи **inc**-файлов параллельных **lib**-файлам и делать все необходимые доопределения в **def**-файлах или, фактически, в **lib**-файлах, что

уменьшает количество используемых служебных файлов. В нашем случае, таковыми являются строки:

```
_CxxThrowException
__CxxThrowException@8 = _CxxThrowException
```

В принципе, подобные универсальные определения можно автоматически генерировать на основе листингов «Иды» для системных **dll**-ек **Windows** (с подгруженными отладочными символами). Для это нужно только написать собственный скрипт, что мы намерены более подробно продемонстрировать в следующей статье.

Отметим еще тот факт, что данная программа использует библиотеку **mfc42u.dll**, соответствующего либа которой нет в поставке **MASM32**, но есть в **MS VS C++**. Так что его можно позаимствовать оттуда, либо скомпилировать самим, приблизительно мы уже знаем, как это делать :).

Завершая эти и другие стандартные уже исправления ассемблерного кода, компилируем **exe**-файл, который на этот раз не захотел нормально выполниться, выдав в окне сообщения информацию о безуспешной попытки записи по указанному адресу.

Ну, что ж, лезем в отладчик по приведенному адресу, ищем аналогичный участок кода в «Иде» и сравниваем. После небольшого анализа видно, что проблемный код в «Иде» представлен в виде:

```
.text:01001E03              mov     ecx, offset unk_1003268
.text:01001E08              jmp     $+5
.text:01001E0D
.text:01001E0D ; =============== S U B R O U T I N E =======================================
.text:01001E0D ; public: __thiscall _AFX_DLL_MODULE_STATE::_AFX_DLL_MODULE_STATE(void)
.text:01001E0D ??0_AFX_DLL_MODULE_STATE@@QAE@XZ proc near
.text:01001E0D              push    esi
. . .
.text:01001E2A              retn
.text:01001E2A ??0_AFX_DLL_MODULE_STATE@@QAE@XZ endp
```

Однако компилятор скомпилирован не длинную пятибайтную инструкцию **jmp $+5**, а короткую двухбайтную, что в итоге привело к переходу не на начало функции **??0_AFX_DLL_MODULE_STATE@@QAE@XZ**, а внутрь нее (со смещением в три байта). Понятно, что ни к чему хорошему это не могло привести. Теперь, когда мы познакомились поближе с проблемой :), легко устранить ее. Можно написать либо **jmp $+2**, либо, что более надежно,

```
jmp    ??0_AFX_DLL_MODULE_STATE@@QAE@XZ
```

Таким образом, эту ошибку мы устранили, но появилась следующая, аналогичная

```
.text:010024AA sub_10024AA     proc near              ; DATA XREF: .text:010011B4□o
.text:010024AA              jmp     $+5
.text:010024AA sub_10024AA     endp ; sp-analysis failed
.text:010024AF
.text:010024AF ; =============== S U B R O U T I N E =======================================
.text:010024AF sub_10024AF     proc near
.text:010024AF              push    423h           ; unsigned __int32
. . .
.text:010024C0              retn
.text:010024C0 sub_10024AF     endp
```

Здесь мы поступаем также, вместо **jmp $+5** пишем

```
jmp    sub_10024AF
```

К счастью, других ошибок времени выполнения нет и мы, после некоторого напряженного ожидания, с удовлетворением наблюдаем наш рукотворный шедевр :) (рис. 12).

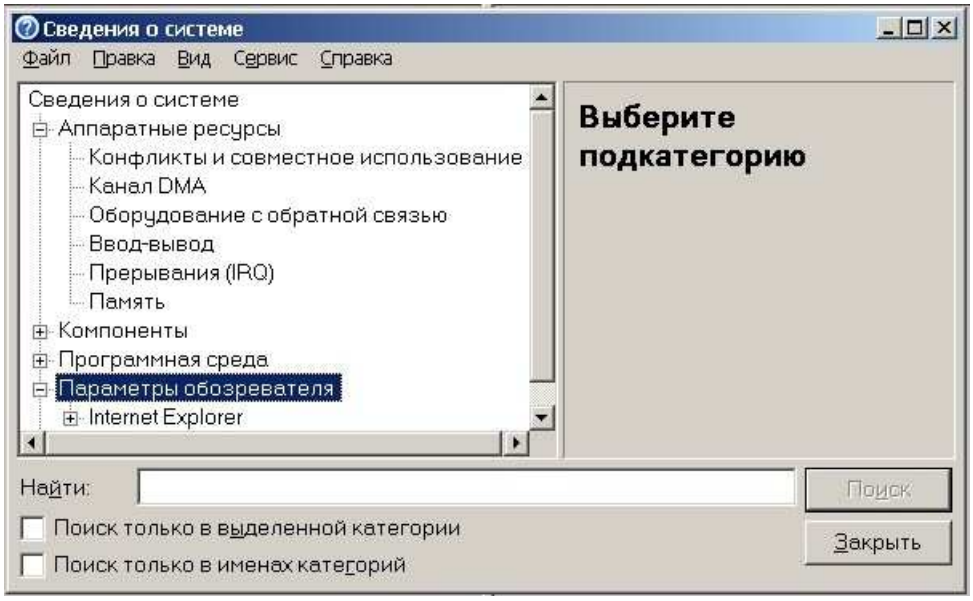

Рис. 11. Запущенное перекомпилированное приложение **winmsd.exe**.

Результаты работы по этой программе мы не будем выкладывать, а оставим в качестве «домашнего задания». Для полноты картины можете еще поработать с файлом программы **msinfo32.exe**.
Итак, намеченную нами программу действий мы выполнили и, следовательно, можем теперь подвести некоторые итоги.

Итоги

Мы лишний раз убедились насколько хорош замечательный инструмент **Ильфака Гильфанова** **IdaPro v. 5.7 demo**. Тем более, можно рекомендовать приобрести его коммерческую версию. Но даже в демо варианте это исключительно мощный инструмент. Тем не менее, хотелось бы опробовать это средство на чем-то по-настоящему реальном, на какой-нибудь достаточно большой серьезной программе. Вопрос только в том, какая это должна быть программа, чтобы с одной стороны, она была достаточно интересной и полезной, а с другой, чтобы ее исследование не ущемляло ничьих прав. Вы можете высказать по этому поводу свое мнение.

Примечание

К данному тексту приложен файл **IdaPro57Test.zip** (http://erfaren.narod.ru/Asm/IdaPro57Test.002 - измените расширение в **zip**), с результатами тестирования. Можно также посмотреть **html** версию этой статьи (http://erfaren.narod.ru/Asm/Erfaren002.htm) либо ее **pdf**-файл (http://erfaren.narod.ru/Asm/Erfaren-002-Recompilation-exe-files.pdf).