ПРОГРАММИРОВАНИЕ В Delphi ГЛАЗАМИ ХАКЕРА

2-е издание

+CD

Михаил Фленов

Как сделать код маленьким, а программу невидимой
От простых шуточных программ до сложной манипуляции системой
Примеры работы с сетью с использованием Windows Socket API
Как работать с портами компьютера и получать информацию о системе
Интересные алгоритмы написания сетевых программ

bhv

УДК 681.3.068
ББК 32.973.26-018.1
Ф69

Фленов М. Е.

Ф69 Программирование в Delphi глазами хакера. — 2-е изд., перераб. и доп.— СПб.: БХВ-Петербург, 2007. — 480 с.: ил. + CD-ROM

ISBN 978-5-9775-0081-4

Рассмотрены нестандартные приемы программирования, а также примеры использования недокументированных функций и возможностей языка Delphi в ОС Windows при разработке шуточных программ и серьезных сетевых приложений для диагностики сетей, управления различными сетевыми устройствами и просто при повседневном использовании интернет-приложений.

Компакт-диск содержит исходные коды примеров и откомпилированные программы, а также популярные приложения компании CyD Software Labs.

Для программистов

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Инны Тачиной*
Зав. производством	*Николай Тверских*

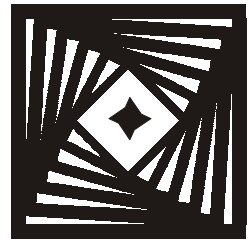
Лицензия ИД № 02429 от 24.07.00. Подписано в печать 28.09.07.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 38,7.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04
от 11.11.2004 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0081-4

Оглавление



Глава 1

Минимизация, скорость и безопасность

Что самое главное при написании шуточных программ? Ну, конечно же, невидимость. Программы, созданные в этой и следующих главах, будут незаметно "сидеть" в системе и выполнять нужные действия при наступлении определенного события. Это значит, что программа не должна появляться на панели задач или в списке запущенных программ в окне, выдаваемом при нажатии комбинации клавиш <Ctrl>+<Alt>+<Del>. Да, в Windows 2000/XP и других ОС есть возможность увидеть запущенные процессы, и от них спрятаться труднее, но это уже не такая большая проблема. Так что, прежде чем начать что-то писать, нужно узнать, как спрятать свое творение от чужого глаза.

Помимо этого, программы-приколы должны иметь маленький размер. Приложения, создаваемые Delphi, достаточно "весомые". Даже простейшая программа, выводящая одно окно, занимает более 200 Кбайт на диске. Если вы захотите отослать такую шутку по электронной почте, то отправка и получение письма с вашей программой отнимет лишнее время у вас и получателя. Это не очень приятно, поэтому в данной главе я познакомлю вас с тем, как можно уменьшить размер программ, создаваемых в Delphi.

1.1. Методы минимизации

Чтобы понять, какие методы есть для уменьшения размера программы, необходимо ответить на вопрос: "Из-за чего программа, созданная Delphi, получается большой?" Ответ очень прост, Delphi является объектным языком, а если точнее — компонентным. *Компоненты* — это следующий шаг в программировании. В Delphi каждый элемент выглядит как объект или компо-

нент, который обладает своими свойствами, методами и событиями. Любой объект вполне автономен, он многое уже умеет делать без ваших указаний и без необходимости в написании дополнительного кода. Это значит, что вам нужно только подключить его к своей форме, изменить нужным образом свойства, и приложение готово! И оно будет работать без какого-либо прописывания его деятельности.

Простой пример автономности объектов можно увидеть на примере какого-либо визуального компонента. Например, кнопка обладает множеством свойств: положение, текст, тип кнопки, рисунок и т. д. Вам не нужно заботиться о том, как будет отображаться текст или картинка на кнопке, все это уже реализовано в самом компоненте. Вам не надо заботиться о том, чтобы кнопка правильно визуально реагировала на действия пользователя при наведении фокуса или нажатии, об этом уже позаботились разработчики, которые создавали кнопку.

Но в объектном программировании есть и свои недостатки. В объектах реализовано большое количество действий, которые вы и пользователь сможете производить с ним. Но реально в любой программе мы используем два-три из всех этих свойств. Все остальное — для программы лишний груз, который никому не нужен, но от которого невозможно избавиться. Каждый объект — неделимое целое, из которого просто нельзя вырвать определенные свойства и методы.

А если вспомнить про такую возможность объектного программирования, как наследование, то получается, что у кнопки может быть несколько предков и весь их код, все их возможности также должны быть включены в исполняемый файл. В сложных библиотеках (а библиотека VCL, используемая в Delphi очень сложная) очень много различных объектов, и в большинстве случаев иерархия наследования в библиотеках древовидная. В этом дереве все объекты и компоненты (ветви дерева) происходят от одного объекта (ствол), что немного экономит количество исходного кода и упрощает жизнь, но сэкономить результирующий код все равно не позволяет.

Но как же тогда создать компактный код, чтобы программа занимала минимум места на винчестере и в оперативной памяти? Тут есть несколько вариантов.

- Сжимать готовые программы с помощью компрессоров. Объектный код сжимается в несколько раз, и программа, созданная с использованием VCL, может превратиться из монстра в 300 Кбайт в скромного по размерам "зверя", "весящего" всего 30—50 Кбайт. Главное преимущество состоит в том, что вы не лишаетесь возможностей объектного программирования и можете спокойно забыть про неудобства WinAPI.
- Не использовать визуальную или объектную библиотеку, которая упрощает программирование. В Delphi такой библиотекой является VCL, а в

Visual C++ — это MFC. В этом случае весь код придется набирать вручную и работать только с WinAPI, что достаточно не просто. Программа в таком случае получается очень маленькой и быстрой. Но таким образом вы лишаетесь простоты визуального программирования и можете ощутить все неудобства программирования с помощью чистого WinAPI. Небольшую утилиту написать с использованием только Windows API еще можно, но большую программу — нереально.

- Использовать визуальные библиотеки, которые не сильно влияют на размер результирующего исполняемого файла. Для Delphi есть такая библиотека — это связка MCK-KOL.
- Использовать динамическую компиляцию библиотеки. Коротко рассмотрим, как это помогает нам. В этом случае, в исполняемый файл попадают только логика работы программы и ваши собственные объекты или компоненты. Стандартная библиотека языка не попадает в исполняемый файл, а подключается динамически из dll-файлов, и должна поставляться совместно с программой или быть заранее установленной на компьютере пользователя.

В данной главе мы рассмотрим все эти методы уменьшения размера. Причем вы можете использовать даже сочетания из двух методов, например, написать программу без использования визуальной библиотеки, а потом еще и сжать ее с помощью специализированного компрессора.

1.2. Сжатие исполняемых файлов

Самый простой способ уменьшить размер приложения — использование программы для сжатия файлов. Лично я очень люблю ASPack, которую вы можете найти на компакт-диске в каталоге Программы (файл установки называется ASPack.exe). Она прекрасно сжимает исполняемые exe-файлы и динамические dll-библиотеки.

Я не буду подробно описывать процесс установки ASPack, потому что там абсолютно нет ничего сложного. Только одно нажатие на кнопке **Next**, и все готово! Теперь запустите установленную программу, и вы увидите окно, изображенное на рис. 1.1. Главное окно состоит из нескольких вкладок:

- **Open File**;
- **Compress**;
- **Options**;
- **About**;
- **Help**.

На вкладке **Open File** есть только одна кнопка — **Open**. Нажмите ее и выберите файл, который вы хотите сжать. Как только вы выберете файл, программа перейдет на вкладку **Compress** и начнет сжатие (рис. 1.2).

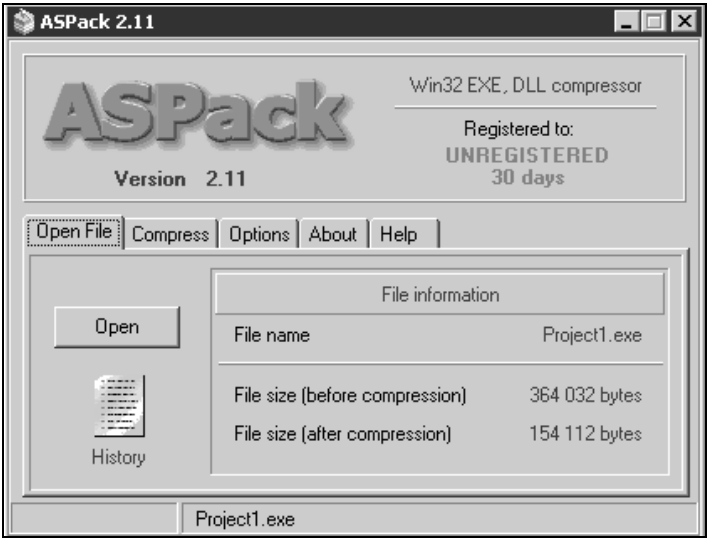


Рис. 1.1. Главное окно ASPack

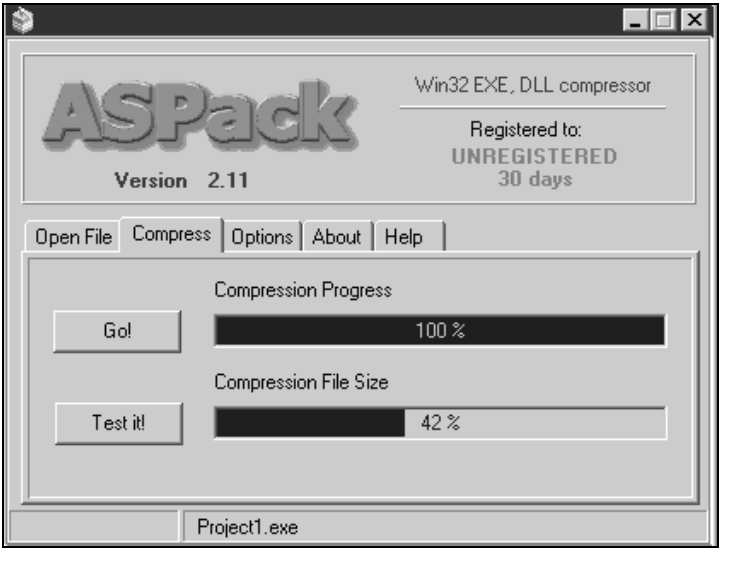


Рис. 1.2. Сжатие файла

Сжатый файл сразу перезаписывает существующий, а старая несжатая версия сохраняется на всякий случай под тем же именем, но с расширением bak. Настроек у ASPack не так уж много (рис. 1.3), и с ними вы сможете разобраться без моей подсказки. Лучше я расскажу вам, как это работает.

Рис. 1.3. Настройки ASPack

Давайте разберемся, как работает сжатие. Сначала весь код программы сжимается архиватором. Если вы думаете, что он какой-то "навороченный", то сильно ошибаетесь. Для сжатия используется обычный архиватор, только оптимизированный для сжатия двоичного кода. После этого, в конец сжатого кода добавляется код разархиватора, который будет программу распаковывать обратно. И в самом конце ASPack изменяет заголовок исполняемого файла так, чтобы при старте сначала запускался распаковщик.

Теперь, когда вы запустите сжатую программу, сначала заработает распаковщик, который "развернет" бинарный код программы и аккуратно разместит его в памяти компьютера. Как только этот процесс закончится, разархиватор передаст управление вашей программе.

Некоторые считают, что из-за расходов на распаковку программа будет работать медленней!!! Я бы сказал, что вы не заметите разницу. Даже если и будут какие-то потери, то они окажутся неощутимыми (по крайней мере, на современных компьютерах). Это происходит потому, что архивация хорошо оптимизирована под двоичный код. И по сути дела, распаковка происходит

только один раз и в дальнейшем никакого влияния на работу программы не оказывает. В результате потери в скорости из-за сжатия будут неощутимы и только на этапе запуска исполняемого файла.

При нормальном программировании с использованием всех современных возможностей типа визуальности и объектного программирования код получается большим, но его можно сжать на 60—70% специальным архиватором. А писать такой код намного легче и быстрее.

Еще один аргумент в пользу применения сжатия — заархивированный код труднее взломать, потому что не каждый дизассемблер сможет прочитать упакованные команды. Так что, помимо уменьшения размера вы получаете защиту, способную отпугнуть большинство взломщиков. Конечно же, профессионала не отпугнешь даже этим, но взломщик средней руки не будет мучиться со сжатым двоичным кодом.

Еще одна утилита для сжатия файлов, получившая большую популярность — UPX. Ее преимуществом является то, что она абсолютно бесплатна и даже распространяется в исходных кодах. Скачать ее можно здесь: **http://upx.sourceforge.net**. Но тут есть одна проблема — утилита выполняется из командной строки, а в наши времена, когда миром правят курсор мыши и простые визуальные окна, люди отвыкли от черного экрана с курсором.

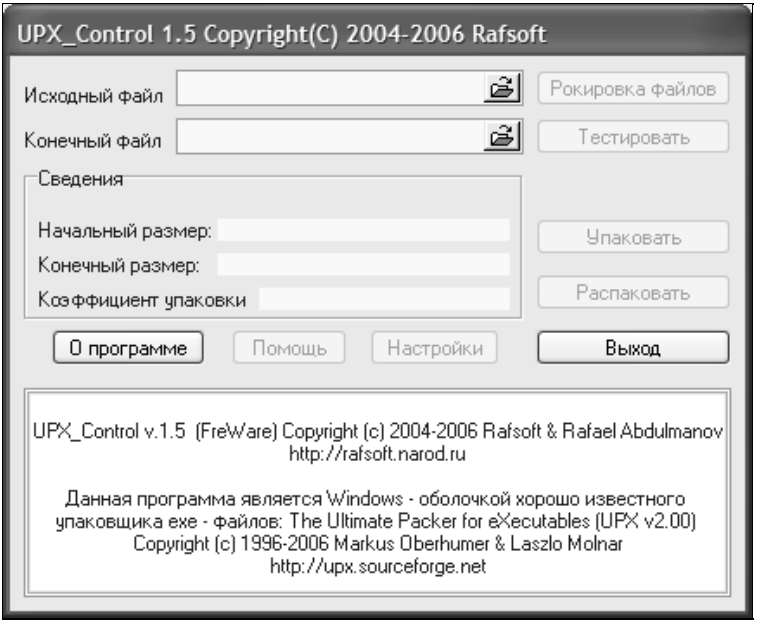


Рис. 1.4. Главное окно программы UPX_Control

Для тех, кто любит визуальность и понятность, могу посоветовать небольшую утилиту UPX_Control, которую можно скачать здесь: **http://rafsoft.narod.ru**. Главное окно еще проще, чем у UPX, а возможности не хуже (рис. 1.4).

В этой программе нужно выбрать исходный файл — исполняемый файл, который необходимо сжать, и нажать кнопку **Упаковать**.

Существует множество программ для сжатия исполняемых файлов. Они отличаются алгоритмом сжатия и возможностями. Некоторые программы позволяют еще шифровать код и предоставлять механизмы, предотвращающие взлом утилит, но это уже необходимо платным программам, а эту тему мы не рассматриваем. К тому же, для большинства универсальных утилит защиты уже давно есть универсальные программы взлома, так что платить за такую защиту деньги — смысла нет, ведь она не эффективна.

1.3. Без окон, без дверей...

Если вы хотите создать программу действительно маленького размера, то необходимо забыть обо всех удобствах. Вы не сможете подключать визуальные формы или другие удобные модули, написанные фирмой Borland для упрощения жизни программиста. Только API-функции самой Windows — и ничего больше.

Для того чтобы создать маленькую программу в Delphi, нужно создать новый проект (по умолчанию система Delphi при открытии сама создаст новый файл проекта, но вы всегда можете самостоятельно создать новое приложение, выбрав в меню **File | New | Application**) и зайти в менеджер проектов (меню **View | Project Manager**). Здесь нужно удалить все модули и формы (пункт **Unit*N***, он выделен на рис. 1.5), чтобы остался только файл самого проекта (по умолчанию его имя Project1.exe). Никаких модулей в проекте не должно быть.

Теперь нужно щелкнуть правой кнопкой мыши на имени проекта и выбрать из появившегося контекстного меню пункт **View Source** или в главном меню **Project** выбрать пункт **View Source**. В редакторе кода откроется файл проекта Project1.dpr. Если вы уже удалили все модули, то его содержимое должно быть таким:

```
program Project1;

uses
  Forms;

{$R *.res}
```

```
begin
  Application.Initialize;
  Application.Run;
end.
```

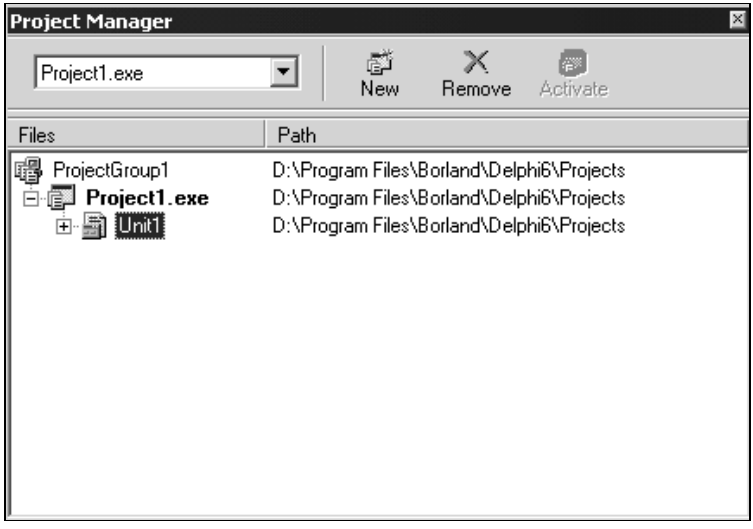


Рис. 1.5. Project Manager

Теперь можно скомпилировать абсолютно пустой проект. Для этого надо выбрать в меню **Project** пункт **Compile Project** или нажать комбинацию клавиш <Ctrl>+<F9>. После компиляции следует выбрать в меню **Project** команду **Information for Project1**. Появится окно с информацией о проекте. Окно, которое появилось передо мной, вы можете увидеть на рис. 1.6.

В правой части окна должны быть описаны используемые пакеты. Так как вы все удалили, значит, там должна красоваться надпись **(None)**. А вот с левой стороны должна быть описана информация о скомпилированном коде. Самая последняя строка показывает размер файла, и у меня он равен 370 688 байтов. Ничего себе "пустая программа"!!! Мы же ничего еще не написали. Откуда же тогда такой большой код?

Давайте разберемся, что осталось в нашем проекте, чтобы обрезать все то, что еще не обрезано. Сразу обратите внимание, что в разделе `uses` подключен модуль `Forms`. Это объектный модуль, написанный "дядей Борландом", а значит, его использовать нельзя, потому что именно он увеличивает размер нашей программы. Между командами `begin` и `end` используется объект

`Application`. Этот объект тоже указывать не надо, потому что он использует объектные возможности и не заботится о "фигуре" программы.

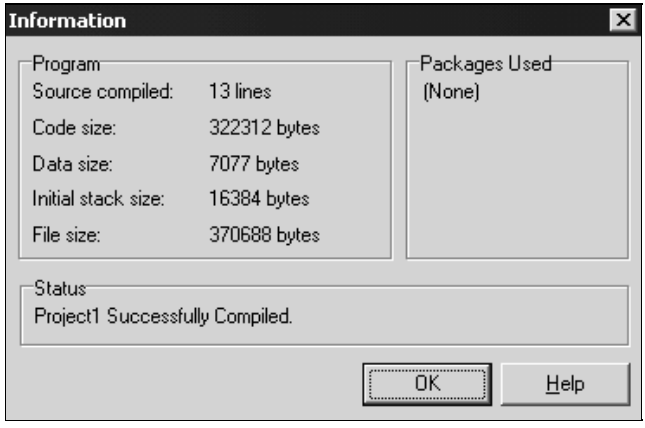


Рис. 1.6. Окно информации о проекте

Большой объем, который появляется даже у пустой программы, как раз и связан с объектом `Application`, который объявлен в модуле `Forms`. Хотя мы использовали только два метода — `Initialize` и `Run`, — при компиляции в exe-файл попадает весь объект `TApplication`, а он состоит из сотен, а может и тысяч строчек кода. Помимо этого, вместе с `TApplication` в исполняемый файл потянется целая куча и еще маленький вагончик дополнительного мусора, который реально в данном проекте не используется.

Чтобы избавиться от накладных расходов, нужно заменить модуль `Forms` на `Windows`, который описывает только функции Windows API и не связан с объектами Delphi. Его подключение является обязательным, иначе вы не сможете вызвать ни одной функции из набора WinAPI. А между `begin` и `end` вообще все можно удалить. В итоге, самый минимальный (с учетом использования модуля `Windows`) код программы будет выглядеть так:

```
program Project1;

uses Windows;

begin

end.
```

Снова откомпилируйте проект. Откройте окно информации и посмотрите на размер получившегося файла. У меня получилось 8192 байта (рис. 1.7). Вот это уже по-человечески. Стоило только отказаться от библиотеки VCL, как размер файла сократился в несколько раз. Есть еще приемы, которые позволят сократить размер исполняемого файла до 6 Кбайт или даже менее, но это уже не важно. Я не вижу смысла мучаться из-за 2—4 Кбайт, которые не сыграют большой роли.

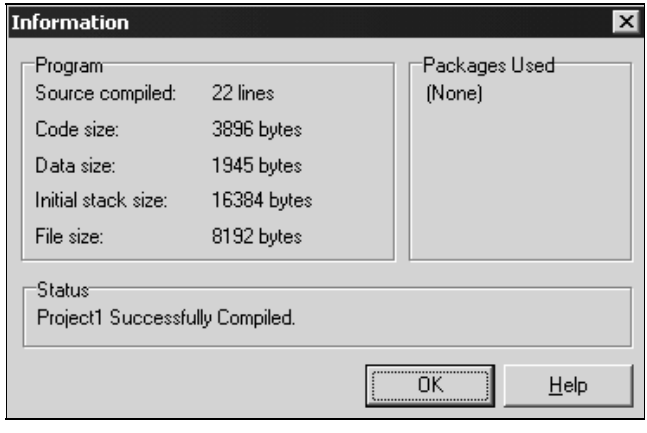


Рис. 1.7. Информации о проекте после удаления ненужного кода

Заготовка минимальной программы с использованием WinAPI готова. Теперь вы можете смело добавлять свой код. Мне нужно только объяснить вам, какие модули можно подключать к своему проекту в раздел `uses`, а какие — не стоит, дабы "жирность" исполняемого файла резко не увеличилась. Тут все очень просто и не займет много времени.

Если при установке Delphi вы не отключали копирование исходных кодов библиотек, то перейдите в каталог, куда вы установили Delphi. Здесь перейдите в папку Source, затем в Rtl и, наконец, в Win. Если вы отключили копирование исходников, то вставьте компакт-диск с Delphi и поищите эти каталоги там. В них расположены исходные коды модулей, в которых описаны все API-функции Windows. Именно эти модули вы можете подключать к своим проектам, если хотите получить маленький код. Если вы подключите что-то другое, то я уже не гарантирую минимум размера вашей программы (хотя, есть и исключения). Самое опасное — это подключение модулей из каталога Delphi7\Source\Vcl\.

Сразу же рассмотрим пример. Если вы хотите, чтобы в вашей программе были возможности работы с сетью, то вам нужно подключить к нему биб-

лиотеку Window Socket. Среди модулей WinAPI есть файл с именем winsock.pas. Значит, вы должны в разделе `uses` написать `Winsock` (расширение писать не надо), и ваша программа сможет работать с сетью.

Пока что я описал минимальный проект, в который можно добавлять свой код. Но код, который вы вставите, выполнится один раз, и программа выгрузится из памяти. А что если вам надо, чтобы программа постоянно "висела" в памяти и что-то делала? Для этого нужно использовать следующий шаблон для своих программ:

```
program Project1;

uses Windows;

var
  Msg: TMsg;
Begin

// Сюда можно добавлять свой код

// Дальше идет код, который заставит программу "висеть"
// в памяти вечно и не будет сильно загружать систему
     while GetMessage(Msg, HInstance, 0, 0) do
      begin
        TranslateMessage(Msg);
        DispatchMessage(Msg);
      end;
end.
```

Сейчас я не буду описывать этот шаблон, потому что дальше мы подробно обсудим написание полноценного шаблона минимального приложения. Там и будут описаны функции, которые используются в этом примере.

Самое интересное, что такое минимальное приложение будет не видно в системе. Мы не создавали никаких окон, значит, на экране ничего отображаться не будет. Программа не будет иметь фокуса ввода, значит, в панели задач тоже незачем что-то отображать.

У нас получилась не программа, а процесс. Именно поэтому его можно заметить в Windows 2000/XP только с помощью программ, которые умеют отображать все процессы, запущенные в системе. В Windows 2000/XP при

нажатии комбинации клавиш <Ctrl>+<Alt>+<Del> появляется окно **Диспетчер задач Windows** (рис. 1.8). В этом окне есть вкладка **Процессы**, где и можно найти такую программу. Вообще-то, на этой вкладке можно найти любую работающую программу, и от инспекторского взора Диспетчера задач спрятаться очень тяжело. Так что, достаточно только назвать исполняемый файл не сильно вызывающе, и простой пользователь ничего не заподозрит, потому что список процессов достаточно большой, и я не думаю, что кто-то знает хотя бы половину из них. Большинство пользователей в этот список просто не заглядывают.

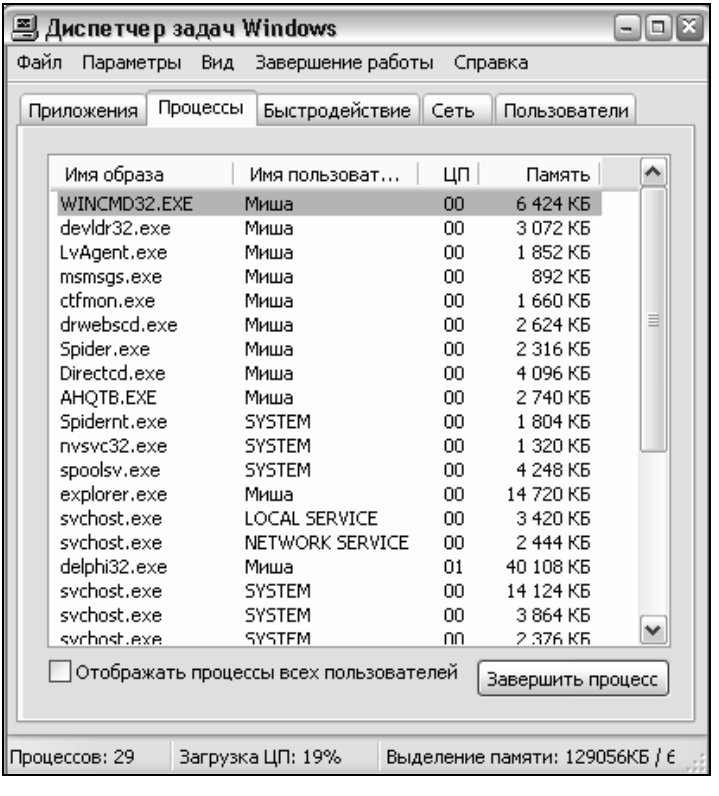


Рис. 1.8. Вкладка **Процессы** в Диспетчере задач Windows XP

В Windows 9*x* при нажатии комбинации клавиш <Ctrl>+<Alt>+<Del> такая программа пока еще будет видна. Чтобы окончательно скрыть ее из виду, нужно добавить следующий код:

```
procedure RegisterServiceProcess; external 'kernel32.dll' name 'Register-
ServiceProcess';
```

Здесь объявлена недокументированная процедура `RegisterServiceProcess`, которая есть в ядре kernel32.dll. Эта процедура делает нашу программу процессом в системе Windows 9*x*. Данную строку с описанием процедуры нужно добавить сразу после раздела `uses`.

Чтобы вызвать эту процедуру, нужно написать следующий код в любом месте программы (желательно в начале):

```
asm
 push 1
 push 0
 call RegisterServiceProcess;
end;
```

Этот код использует код на языке ассемблера, чтобы вызвать WinAPI-функцию `RegisterServiceProcess`. Да, в Delphi есть встроенный ассемблер. Вы можете прямо среди кода на Delphi писать код на ассемблере. Инструкции языка ассемблера нужно заключать между словами `asm` и `and;`. В примере используются три инструкции. В первой в стек помещается число `1` с помощью операции `push`. Во второй строке в стек отправляется значение `0`. Эти два числа, находящиеся в стеке, являются параметрами для функции `RegisterServiceProcess`. То же самое можно было бы сделать и с помощью вызова функции на Delphi: `RegisterServiceProcess(0, 1)`, но мне захотелось показать вам, как работает встроенный ассемблер. Да и для этого нужно изменить описание процедуры `RegisterServiceProcess`, которая регистрирует программу как процесс и в Windows 9*x* делает программу невидимой в списке запущенных программ, который вы вызываете нажатием комбинации клавиш <Ctrl>+<Alt>+<Del>. В последней инструкции ассемблера я вызываю процедуру `RegisterServiceProcess` с помощью оператора `call`.

Теперь наша программа не будет видна и в Windows 9*x*. Только вы должны учитывать, что этот код прекрасно работает в Windows 9*x*, но выдаст ошибку в Windows 2000/XP, потому что в его ядре нет функции `RegisterServiceProcess`. Поэтому вы должны знать, в какой системе будет запускаться программа, и можно ли использовать процедуру `RegisterServiceProcess`.

1.4. Шаблон минимального приложения

Теперь нам пора написать хороший шаблон минимального приложения, который мы будем использовать в этой книге для написания наших программ-приколов. Для этого нам надо познакомиться с "внутренностями" Windows и его WinAPI.

Запустите уже полюбившийся Delphi. Как всегда, сразу откроется новый проект. Так как мы очень часто будем делать минимальные программы, то нам абсолютно не нужны никакие формы, их надо удалить. В меню выберите **View | Project Manager** (рис. 1.9). Перед вами появится окно менеджера проектов. Выделите **Unit1** и нажмите кнопку **Remove** для удаления лишней формы (рис. 1.10).

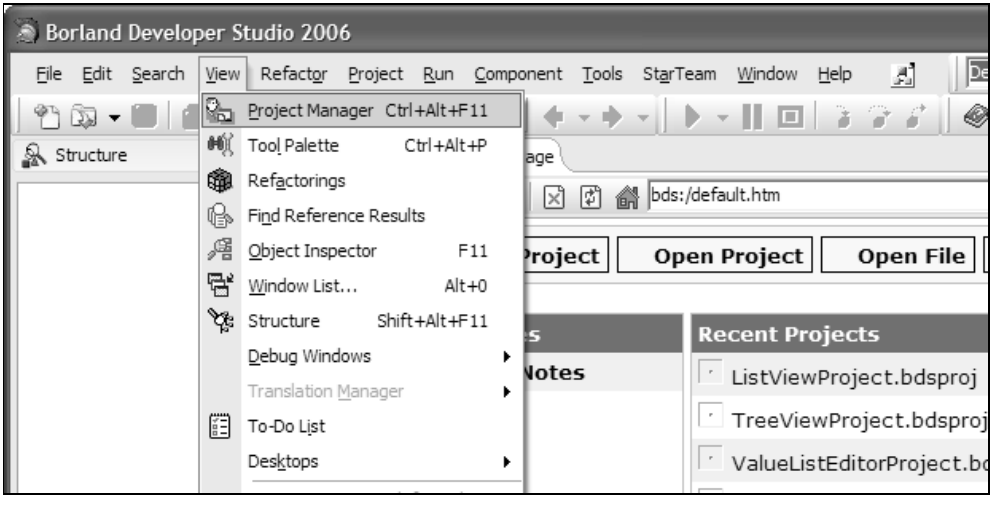


Рис. 1.9. Вызов менеджера проектов

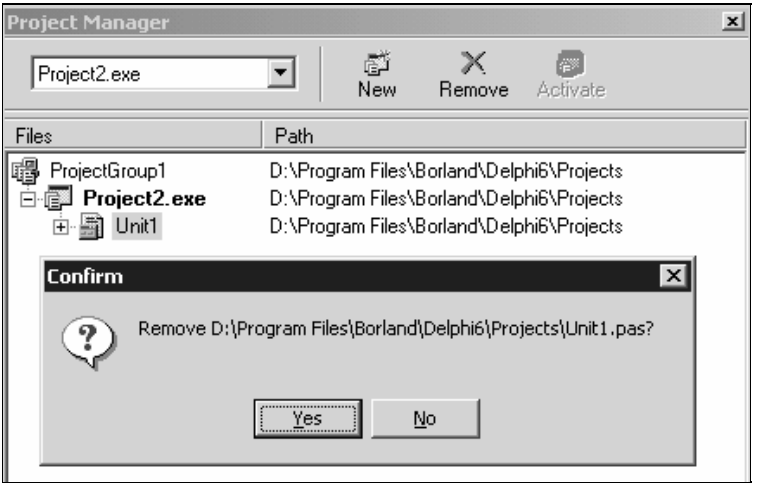


Рис. 1.10. Удаление ненужной формы