**Разработка программы передачи и приёма сообщений в локальной сети Microsoft**

На сегодняшний день существует довольно большое число программ отправки и принятия сообщений для локальных сетей под управлением на платформах от компаний Microsoft, таких как семейство Windows 9x (95, 98, ME, XP) и Windows NT4, Windows2000, разнообразных как по алгоритмам, так и по используемым протоколам. Основной задачей, на которую следует ориентироваться при выборе подобных программ, является величина бюджета, и как следствие, расходы на дополнительную закупку оргтехники (серверы, дополнительные сети, концентраторы).

Поэтому такие программы разделяются на 2 типа:

* Программы клиент-сервер, то есть передача сообщений осуществляется через выделенный сервер.
* Программы для одноранговых сетей, не требующие выделенного сервера.

К первым средствам передачи сообщений можно отнести такие программы, как Microsoft Messenger, AOL ICQ, различные чаты и телеконференции.

Второй тип программ менее функционален, но экономически оправдан даже для небольшого бюджета и требует гораздо меньше системных ресурсов.

Однако существующие на данный момент программы второго типа являются незаконченными по своим первоначальным идеям их создания.

Целью настоящего проекта явилось создание многофункциональной программы получения и отправки сообщений по локальной сети. Программа разрабатывалась в локальной сети общежития МИЭТ (SWAMP), насчитывающей 400 компьютеров на сегодняшний момент. В процессе создания использовались менее функциональные, но аналогичные по разработке программы Winpopup и Traypopup.

**Эскизный проект**

*Описание протоколов, используемых при написании дипломной работы*

Стремительное развитие электронной техники и локальных сетей обусловило необходимость создания современного программного обеспечения, и как следствие, необходимость разработки быстрых и удобных в использовании средств обмена информации, в частности, протоколов.

Протоколы представляют собой набор условий (правил), которые регламентируют формат и процедуры обмена информацией между двумя или несколькими независимыми устройствами или процессами. Протокол имеет три важнейших элемента: синтаксис, семантику и синхронизацию (timing). Синтаксис протокола определяет поля, например, может быть 16-байтовое поле для адресов, 32-байтовое поле для контрольных сумм и 512 байт на пакет. Семантика протокола придает этим полям значение: например, если адресное поле состоит из всех адресов, это «широковещательный» пакет. Синхронизация – количество битов в секунду – это скорость передачи данных. Она важна не только на самых низких уровнях протокола, но и на высших.

В общих чертах, протокол – это принятый набор стандартов, устанавливающих порядок общения компьютеров. В этом разделе рассматриваются протоколы, доступные в операционных системах семейства Windows, как и когда каждый из протоколов используется, а также приведено краткое описание каждого протокола.

Встроенные основные протоколы Windows:

* Network Basic Input/Output System (NetBIOS)
* NetBIOS Extended User Interface (NetBEUI)
* Transmission Control Protocol (TCP/IP)
* NWLink Internetwork Packet Exchange/Sequenced packet Exchange (IPX/SPX)

Теперь рассмотрим каждый из них.

**NetBIOS**

NetBIOS предоставляет механизмы работы основных функций Windows, таких как просмотр и межпроцессное общение между сетевыми серверами. Данный протокол является очень быстрым, требующий совсем небольшого количества служебной информации, поэтому он используется Windows для основных операций. К сожалению, NetBIOS не имеет возможностей для маршрутизации и поэтому не может быть использован в качестве основного протокола для сетей, в которых нужны такие возможности.

**NetBEUI**

NetBEUI – простой транспортный протокол сетевого уровня, предназначенный для поддержки сетей NetBIOS. Как и NetBIOS, NETBEUI не предназначен для маршрутизации, так что он тоже не может использоваться в большой сети. Преимущества NetBEUI – это его скорость и защищённость от ошибок, простота применения и небольшие требования к ресурсам. К недостаткам можно отнести отсутствие средств маршрутизации. Почти полное отсутствие поддержки платформенно-независимых операций и слишком малое количество средств для поиска неполадок.

NetBEUI можно использовать только в небольшой сети. К примеру, в локальной сети общежития МИЭТ, где используются средства маршрутизации, применение данного протокола является нецелесообразным.

**TCP/IP**

Протокол TCP/IP на сегодняшний день чаще всего используется для работы в сети. Именно этому протоколу Интернет обязан своим быстрым ростом. TCP/IP – самый гибкий транспортный протокол, позволяющий охватывать большие сетевые пространства. Кроме того, он предоставляет возможности для поддержки платформенно-независимых приложений, маршрутизации, а также поддерживает протоколы SNMP, DHCP, службы WINS и DNS.

TCP/IP может быть использован в любой сети, но не рекомендуется для маленьких сетей из-за своих требований к ресурсам. С его помощью можно организовывать большие многоранговые сети.

**NWLink (IPX/SPX)**

NWLink – это реализация фирмой Microsoft набора протоколов IPX/SPX, используемых фирмой Novell в сетях NetWare. Этот протокол включен в NT для возможности общения с серверами NetWare. С помощью этого протокола клиенты сетей Microsoft могут пользоваться ресурсами серверов Netware и наоборот

Операционные системы семейства Windows используют NWLink для работы с клиентами и серверами NetWare. Кроме того, клиентам и серверам NetWare требуется служба File And Printer Sevice For NetWare для работы с файлами и принтерами семейства Windows.

NWLink используется в сетях, в которых находятся серверы NetWare, или там, где требуются средства маршрутизации, но не хватает ресурсов для использования протокола TCP/IP.

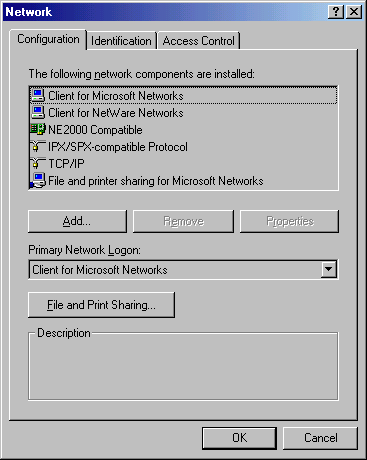


Рис. 1.1. Пример настройки сетевого окружения OC семейства Windows

*Обзор аналогичных программ принятия и отправки сообщений*

Наиболее используемыми на сегодняшний момент явлются две следующие программы, не требующие больших системных ресурсов.

1. *Winpopup.*

Входит в стандартную поставку Windows. В отличие от Winpopup, в программе, представленной в данной дипломной работе, окна отправки и принятия сообщений являются независимыми, что очень удобно для быстрых переговоров: в окне отправки сообщения пишется текст, а получение сообщений происходит в другом окне и между данными окнами можно свободно переключаться. В программе Winpopup такой независимости нет, поэтому для просмотра входящего сообщения приходится отменять отправления своего текста получателю, что является очень неудобной особенностью стандартного средства отправки и приёма сообщений Windows 9x. Что примечательно, в стандартной поставке Windows NT нет аналога Winpopup, поэтому данная программа может являться хорошим сервисом для локальных сетей, состоящих из компьютеров этой операционной системы.

1. *Traypopup.*

Основное преимущество этой программы перед прародителем (как и основное отличие) состоит в том, что в окошке посылки сообщения есть кнопка **Select**, которая показывает список всех компьютеров, присутствующих в сети. Это удобно, потому что не нужно вспоминать имя пользователя в том виде, как оно записано в настройках сети. Также при вызове программы в окне редактирования адреса получателя выставляется имя получателя предыдущего сообщения.

Второе отличие от стандартной программы – она не сидит в **Панели Задач**, а прячется в третьей, к часам, что экономит место, с одной стороны, но неопытному пользователю будет сложно отыскать её среди иконок других программ, с другой стороны. Из недостатков – часто «вылетает» при приёме сообщения, а также не может справиться с количеством сообщений, больших 30.

Другие аналогичные программы не рассматриваются в связи их громоздкостью и большими требованиями к ресурсам.

**Технический проект**

*Общая структура системы*

Программа отправки и получения текстовых сообщений в локальной сети.

Используя почтовые слоты (mailslots) и функции NetBIOS, данная программа полностью совместима с такими стандартными утилитами как WinPopup, net send. Добавлен ряд полезных новшеств, функций и возможностей, такие как история сообщений, быстрые ответы, поддержка большинства ОС Microsoft, быстрый список, поддержка сетевых групп.

**Почтовый слот (mailslot)**

Почтовые слоты – это механизм односторонней пересылки данных между процессами по сети. Почтовые слоты могут использоваться для циркулярной рассылки сообщений целой группе компьютеров одного домена, однако, с другой стороны, нет никакой гарантии доставки посланных данных по нужному адресу, а также не предусмотрено извещение отправителя об их получении.

Любой процесс, который создаёт почтовый слот для приёма данных, называется *сервером слота*. Даже если само по себе приложение является клиентом и получает данные от приложения-сервера, по отношению к созданному почтовому слоту оно представляет собой сервер. Для данного слота любой компьютер может быть сервером, но читать из слота данные (или обращаться к нему другим способом, например, через наследование) может только процесс, который его создал.

Всякий процесс, которому известно имя слота, может посылать ему данные. Процессы, которые посылают данные на почтовый слот, называются *клиентами слота.*

Рассмотрим структуру построения почтового слота, а также процедуры, используемые при написании данного дипломного проекта, и опишем их.

*Создание почтового слота.*

Для создания слота в приложении используется функция CreateMailSlot():

HANDLE CreateMailslot (

LPCTSTR lpName,

DWORD nMaxMessageSize,

DWORD lReadTimeout,

LPSECURITY\_ATTRIBUTES lpSecurityAttributes);

Первый параметр – это указатель на ASCII‑строку, содержащую имя создаваемого почтового слота. Слоты создаются только на локальном компьютере, поэтому полное имя слота выглядит следующим образом:

\\.\mailslot\[путь] имя

В именах почтовых слотов поддерживается псевдокаталоговая структура. Для лучшей организации почтовых слотов в их именах можно указывать пути в каталогах.

Функция CreateMailSlot также принимает в качестве параметров максимальный размер сообщений, время задержки для операций чтения и необязательную защитную структуру. Вместо значения задержки можно указать MAILSLOT\_WAIT\_FOREVER для блокирования операций чтения на слоте до получения сообщения без ограничения времени.

Если функция CreateMailSlot() завершается успешно, она возвращает ключ для нового слота. В случае ошибки возвращается значение INVALID\_HANDLE\_VALUE, для получения кода ошибки можно воспользоваться функцией GetLastError().

Создание типичного почтового слота может быть реализовано следующим образом:

Hslot: = CreateMailSlot(«\\\\.\mailslot\\MySlot\MsgSlot»,

300,

MAILSLOT\_WAIT\_FOREVER,

NULL);

If hSlot=INVALID\_HANDLE\_VALUE then

Begin

ShowMessage (‘CreateMailSlot Error’, GetLastError());

Halt(0);

End;

Для изменения времени задержки операций чтения после создания слота используется функция SetMailSlot(). Кроме того, для получения такой информации о слоте, как максимальный размер сообщения, размер следующего сообщения в очереди и количество сообщений в очереди, можно использовать функцию GetMailSlotInfo().

*Чтение из почтового слота*

Для чтения из слота используется функция ReadFile(), которая завершает свою работу после получения сообщения, независимо от того, сколько байт необходимо прочитать.

Кроме того, с помощью функции GetMailSlotInfo() можно узнать количество сообщений, ожидающих в очереди слота, и размер следующего сообщения.

*Запись в почтовый слот*

Для записи сообщения в слот необходимо сначала открыть его ключ с помощью функции CreateFile(), а затем воспользоваться функцией WriteFile() для посылки сообщения. Ключ слота закрывается функцией CloseHandle().

При вызове CreateFile() необходимо указать имя слота, которому посылается сообщение. Указанное имя допускает циркулярную рассылку данных всем слотам с такими же именами на компьютерах домена.

Для локального компьютера имя слота выглядит следующим образом:

\\.\mailslot\[путь] имя

Для удалённого компьютера имя слота имеет следующий вид:

\\имя\_удалённого\_компьютера\mailslot\[путь] имя

Для рассылки сообщений всем слотам с одинаковыми именами на компьютерах домена необходимо указать следующее имя:

\\имя\_домена\mailslot\[путь] имя

Для рассылки всем компьютерам в домене более высокого иерархического уровня используется следующее имя:

\\\*\mailslot\[путь] имя

Кроме того, при вызове функции CreateFile() можно выбрать режим GENERIC\_WRITE, режим совместного использования FILE\_SHARE\_READ или режим создания OPEN\_EXISTING.

После успешного завершения функции CreateFile() ключ, который она возвращает, можно использовать в функции WriteFile() для посылки сообщений на слот.

*Закрытие слота.*

По окончании работы с почтовым слотом кго необходимо закрыть вызовом функции CloseHandle() для освобождения ассоциированных с ним системных ресурсов. По завершении процесса все ключи слотов, ассоциированные с процессом, закрываются автоматически. После их закрытия почтовый слот уничтожается, а все данные в его буфере теряются.

**Протокол NetBIOS и его команды**

При написании программы с использованием протокола NetBIOS используется блок управления сетью – Net Control Block (NCB). В таблице 1 проиллюстрирована структура NCB, длина и значение каждого поля. Все поля блока NCB перенесены в WinAPI, соответственно, программирование на Delphi с использованием команд NetBIOS является достаточно тривиальной задачей.

*Визуальные компоненты Delphi, используемые при разработке данного дипломного проекта.*

**TMainMenu** позволяет поместить главное меню в программу. При помещении TMainMenu на форму это выглядит, как просто иконка. Иконки данного типа называют «невидимыми компонентом», поскольку они невидимы во время выполнения программы. Создание меню включает три шага: (1) помещение TMainMenu на форму, (2) вызов Дизайнера Меню через свойство Items в Инспекторе Объектов, (3) определение пунктов меню в Дизайнере Меню.

**TPopupMenu** позволяет создавать всплывающие меню. Этот тип меню появляется по щелчку правой кнопки мыши на объекте, к которому привязано данное меню. У всех видимых объектов имеется свойство PopupMenu, где и указывается нужное меню. Создается PopupMenu аналогично главному меню.

**TLabel** служит для отображения текста на экране. Вы можете изменить шрифт и цвет метки, если дважды щелкнете на свойство Font в Инспекторе Объектов. Видно, что это легко сделать и во время выполнения программы, написав всего одну строчку кода.

**TEdit** – стандартный управляющий элемент Windows для ввода. Он может быть использован для отображения короткого фрагмента текста и позволяет пользователю вводить текст во время выполнения программы.

**TMemo –** иная форма TEdit. Подразумевает работу с большими текстами. TMemo может переносить слова, сохранять в ClipBoard фрагменты текста и восстанавливать их, и другие основные функции редактора. TMemo имеет ограничения на объем текста в 32Кб, это составляет 10–20 страниц. (Есть VBX и «родные» компоненты Delphi, где этот предел снят).

**TButton** позволяет выполнить какие-либо действия при нажатии кнопки во время выполнения программы. В Delphi все делается очень просто. Поместив TButton на форму, Вы по двойному щелчку можете создать заготовку обработчика события нажатия кнопки. Далее нужно заполнить заготовку кодом:

procedure TForm1. Button1Click (Sender: TObject);

begin

ShowMessage(‘Test’);

end;

**TCheckBox** отображает строку текста с маленьким окошком рядом. В окошке можно поставить отметку, которая означает, что что-то выбрано.

**TListBox** нужен для показа прокручиваемого списка. Классический пример ListBox’а в среде Windows – выбор файла из списка в пункте меню File | Open многих приложений. Названия файлов или директорий и находятся в ListBox’е.

**TComboBox** во многом напоминает ListBox, за исключением того, что позволяет водить информацию в маленьком поле ввода сверху ListBox. Есть несколько типов ComboBox, но наиболее популярен спадающий вниз (drop-down combo box), который можно видеть внизу окна диалога выбора файла.

**TScrollbar –** полоса прокрутки, появляется автоматически в объектах редактирования, ListBox’ах при необходимости прокрутки текста для просмотра.

**TGroupBox** используется для визуальных целей и для указания Windows, каков порядок перемещения по компонентам на форме (при нажатии клавиши TAB).

**TBitBtn –** кнопка вроде TButton, однако на ней можно разместить картинку (glyph). TBitBtn имеет несколько предопределенных типов (bkClose, bkOK и др), при выборе которых кнопка принимает соответствующий вид. Кроме того, нажатие кнопки на модальном окне (Form2. ShowModal) приводит к закрытию окна с соответствующим модальным результатом (Form2. ModalResult).

**TSpeedButton –** кнопка для создания панели быстрого доступа к командам (SpeedBar). Пример – SpeedBar слева от Палитры Компонент в среде Delphi. Обычно на данную кнопку помещается только картинка (glyph).

**TScrollBox** – позволяет создать на форме прокручиваемую область с размерами большими, нежели экран. На этой области можно разместить свои объекты.

**TTimer –** таймер, событие OnTimer периодически вызывается через промежуток времени, указанный в свойстве Interval. Период времени может составлять от 1 до 65535 мс.

*Требования, предъявляемые к программе отправки и приёма сообщений.*

Теперь можно уточнить постановку задачи: требуется написать систему, обеспечивающую выполнение следующих функций:

* + реализация средств обмена информацией, используя вышеописанные протоколы;
  + использование функций для удобной работы приёма и отправки сообщений;
  + выдача данных пользователям по их запросам.

Система должна работать под управлением Windows 95, реализована в среде разработки Delphi 5. Система должна состоять из двух компонент – ядро реализации средств обмена информацией и пользовательская оболочка.

### *Выбор платформы проектирования, обоснование*

В качестве операционной среды, в которой реализовался данный дипломный проект, была выбрана среда *Windows9x (Windows NT).*Причины этого следующие:

1. Распространенность этих ОС, в связи с чем получается охват большого числа возможных пользователей;
2. Возможность работы с большими массивами данных, реализация чего в других средах представляет нетривиальную и трудновыполнимую задачу;
3. Удобство графического интерфейса дает возможность при минимизации затрат на его реализацию максимизировать удобство работы пользователя с программой. Интерфейс интуитивно понятен и стандартизован, соответственно пользователь не потратит много времени на освоение программы и в дальнейшем количество возможных ошибок в его действиях будет минимизировано.
4. 32-разрядность систем *Windows9x* и *Windows NT* увеличивает скорость работы с соответствующими числами.

В качестве среды программирования была выбрана среда *Borland* *Delphi 5****,*** сочетающая в себе как все преимущества всех средств, относящихся к *RAD*, так и свои собственные преимущества:

1. Простота и надежность создания и отладки программы;
2. Использование всех преимуществ операционных систем *Windows’95* и *Windows NT****,*** включая 32-разрядность, многозадачность, удобный интерфейс и прочее;
3. Использование обработки исключений (exceptions), что позволяет повысить надежность работы программного продукта;

Наличие и доступность большого количества компонент, реализующих многие стандартные функции.

Учитывая специфику задачи и возможности операционной системы, разрабатываемая система должна подчиняться следующим требованиям.

*Конфигурация технических средств*

Требования к составу и параметрам технических средств:

* i486 66 MHz процессор (Pentium или выше рекомендуется);
* 4MB оперативной памяти;
* 1Мб свободного пространства на жестком диске;

– Сетевой адаптер;

Необходимое программное обеспечение:

– Windows’95 или Windows NT 4 и выше;

– Поддержка протокола TCP/IP.

**Рабочий проект**

*Интерфейс программы*

Главная форма содержит следующие компоненты:

– Основное меню, состоящее из пунктов работы с сообщениями и опции для работы с программой.

– Кнопки для быстрой работы с сообщениями:

1. Кнопка «Send» предназначена для посылки сообщения на имя другого компьютера или группу.
2. Кнопка «Delete» удаляет текущее просмотренное сообщение.
3. Кнопки навигации по сообщениям:

а) Кнопка «<» осуществляет просмотр предыдущего сообщения.

б) Кнопка «>» осуществляет просмотр следующего сообщения.

– Строка состояния, показывающая текст «No Message» при отсутствии сообщений и информацию об отправителе, получателе и времени получения сообщения при их наличии.

– Окно для просмотра сообщений.

– Номер текущего сообщения и их общее количество; полезно при навигации и при получении большого количества сообщений.

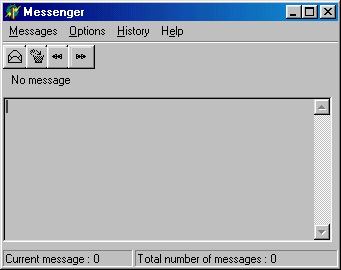


Рис. 1.6. Главное окно программы отправки и получения сообщений.

*Описание с методы работы с программой Messenger*

Основное меню состоит из следующих подменю:

– Messages. Состоит из следующих всплывающих подменю:

Send, Previous, Next, Delete, Delete All и Exit.

Первые четыре подменю вызывают процедуру обработки нажатий «быстрых» кнопок и полностью аналогичны им по своим функциям.

Подменю Delete All удаляет сразу все пришедшие сообщения.

Подменю Exit осуществляет выход из программы.

– Данное подменю содержит параметры для работы с программой.

– History. Содержит историю отправленных и полученных сообщений.

– Help. Содержит информацию о работе с программой.

Рассмотрим отдельно каждую форму.

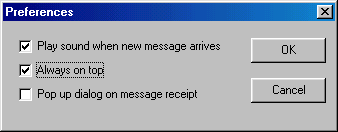


Рис. 1.7. Настройки программы

1. Play sound when new message arrives – Включение/отключение звука, подающегося на колонки через звуковую плату при получении сообщения.
2. Always on top – При включении данной опции окно программы будет всегда находиться поверх других окон
3. Pop up dialog on message receipt – При включении данного параметра окно программы будет всплывать при получении сообщения.

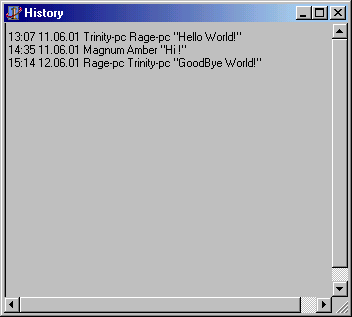


Рис. 1.8. Окно истории сообщений

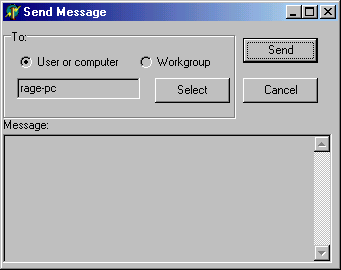


Рис. 1.9. Окно отправки сообщения

Рассмотрю более подробно визуальный процесс посылки сообщения.

При нажатии на кнопку «Send» быстрой работы с сообщениями, равно как и при вызове всплывающего подменю Send из меню Messages, вызывается новое окно, на котором находится:

1. Выбор получателя сообщений:

a) User or Computer. Им может являться имя пользователя, зашедшего в сеть Microsoft либо имя компьютера.

б) Workgroup. Сообщение можно послать на любую группу Win9x, в которую могут быть включены компьютеры.

1. Окно для написания получателя сообщений.
2. Окно сообщения.
3. Кнопка «Send» для отправления сообщения.
4. Кнопка «Cancel» для отмены отправления сообщения.
5. Кнопка «Select» для выбора адресата получателя.

Если размер отсылаемого или принятого сообщения больше размера формы для его написания, то на данной форме становится активным ScrollBar, с помощью которого можно прокручивать форму, тем самым просматривая сообщение.

При нажатии на кнопку «Select» вызывается окно поиска всех групп и компьютеров, находящихся в данной локальной сети.

Пользователю достаточно просто выбрать имя в ListBox’e и нажать кнопку «OK», после чего выбранное имя автоматически отобразиться в окне получателя сообщения.

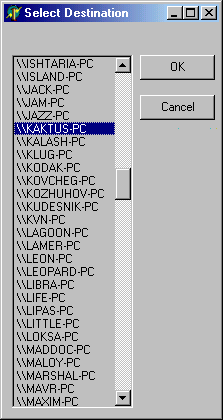


Рис. 1.10. Выбор адресата получателя

*Поиск компьютеров в локальной сети*

Приведём пример кода программы, реализующую поиск компьютеров в локальной сети Microsoft.

procedure TForm4. Button1Click (Sender: TObject);

var Q, BufferSize: DWord;

R: THandle;

Buf: ^Buffer;

P: Pointer;

NR: ^NETRESOURCE;

begin

ListBox1. Clear;

if WNetOpenEnumA (RESOURCE\_GLOBALNET, RESOURCETYPE\_ANY, 0, nil, R) = NO\_ERROR then

begin

BufferSize:= 0;

Q:= $FFFFFFFF;

if WNetEnumResourceA (R, Q, Buf, BufferSize) <> NO\_ERROR then

begin

Q:=1;

New(Buf);

while WNetEnumResourceA (R, Q, Buf, BufferSize) = NO\_ERROR do

begin

begin

P:= Buf;

NR:= P;

ListBox1. Items. Add (NR^.lpProvider);

L. Add(Buf);

end;

Q:= $FFFFFFFF;

New(Buf);

end;

Dispose(Buf);

end;

WNetCloseEnum(R);

end;

end;

procedure TForm4. FormCreate (Sender: TObject);

begin

ListBox1. Clear;

L:= TList. Create;

end;

procedure TForm4. FormClose (Sender: TObject; var Action: TCloseAction);

var I: Integer;

begin

for I:= 0 to L. Count – 1 do

begin

Dispose (L. Items[I]);

end;

L. Free;

end;

procedure TForm4. Button2Click (Sender: TObject);

var

s:string;

begin

if ListBox1. ItemIndex<>-1 then

begin

s:=ListBox1. Items [ListBox1. ItemIndex];

if s[1]='\' then Delete (s, 1, 2);

Form2. Edit1. Text:=s;

ModalResult:=mrOK;

end;

end;

procedure TForm4. ListBox1DblClick (Sender: TObject);

var Q, BufferSize: DWord;

R: THandle;

Buf: ^Buffer;

P: Pointer;

NR: ^NETRESOURCE;

NREsource: NETRESOURCE;

I: Integer;

Err: Integer;

Path: string;

Sr: TSearchRec;

begin

NR:= L. Items [ListBox1. ItemIndex];

Move (NR^, NResource, SizeOf(NETRESOURCE));

ListBox1. Clear;

Path:= NResource.lpRemoteName;

Err:=WNetOpenEnumA (RESOURCE\_GLOBALNET, RESOURCETYPE\_ANY, 0, @NResource, R);

for I:= 0 to L. Count -1 do

begin

Dispose (L. Items[I]);

end;

L. Clear;

if Err = NO\_ERROR then

begin

BufferSize:= 1024;

q:=1;

New(Buf);

while WNetEnumResourceA (R, Q, Buf, BufferSize) = NO\_ERROR do

begin

begin

P:= Buf;

NR:= P;

ListBox1. Items. Add (NR^.lpRemoteName);

L. Add(Buf);

end;

New(Buf);

end;

Dispose(Buf);

end;

WNetCloseEnum(R);

end;

*Отправление сообщений*

Приведем часть кода программы, реализующую отправление сообщений:

function TForm2.NBName (s:string):string;

const

size=1024;

var

ncb:TNCB;

buf, p:PChar;

i, k:integer;

ch:char;

res:string;

begin

NBName:='';

GetMem (buf, size);

FillChar (ncb, SizeOf(TNCB), 0);

ncb.ncb\_command:=char (NCBASTAT);

ncb.ncb\_buffer:=buf;

ncb.ncb\_length:=size;

for i:=1 to Length (s) do

ncb.ncb\_callname [i‑1]:=UpCase (s[i]);

for i:=Length (s) to NCBNAMSZ do ncb.ncb\_callname[i]:=' ';

ncb.ncb\_callname [NCBNAMSZ‑1]:=#03;

ch:=netbios (addr (ncb));

if (ch<>#0) and (ncb.ncb\_retcode<>#0) then Exit;

p:=ncb.ncb\_buffer;

with PAdapterStatus (ncb.ncb\_buffer)^ do

begin

p:=p+SizeOf (TAdapterStatus);

for i:=0 to name\_count‑1 do

begin

with PNameBuffer (p)^ do

begin

if (name [NCBNAMSZ‑1]=' ') and ((ord (name\_flags) and UNIQUE\_NAME)=UNIQUE\_NAME) then

begin

res:='';

for k:=0 to NCBNAMSZ‑2 do if name[k]<>' ' then res:=res+name[k];

NBName:=res;

end;

end;

p:=p+SizeOf (TNameBuffer);

end;

end;

FreeMem (buf);

end;

procedure TForm2. Button1Click (Sender: TObject);

var

handle:THandle;

lpFileName:PChar;

lpNumberOfBytesWritten: Cardinal;

buffer:string;

i:integer;

s:LPSTR;

comp:string;

name, name2:PChar;

name1:string;

namesize:DWORD;

begin

Button1. Enabled:=False;

if RadioButton1. Checked then

begin

comp:=NBName (Edit1. Text);

lpFileName:=PChar ('\\'+comp+'\mailslot\messngr'+#0);

handle:=CreateFile (lpFileName, GENERIC\_WRITE, 0, NIL, CREATE\_ALWAYS, 0, 0);

if handle=INVALID\_HANDLE\_VALUE then

begin

ShowMessage ('Error when executing CreateFile()');

end;

buffer:='';

for i:=0 to Memo1. Lines. Count‑1 do

buffer:=buffer+Memo1. Lines. Text+#10+#13;

name1:='';

namesize:=MAX\_COMPUTERNAME\_LENGTH + 1;

GetMem (name, MAX\_COMPUTERNAME\_LENGTH + 1);

GetComputerName (name, namesize);

name2:=name;

for i:=0 to namesize‑1 do

begin

name1:=name1+name2^;

name2:=name2+1;

end;

FreeMem (name);

buffer:=name1+#0+Edit1. Text+#0+buffer;

s:=LPSTR (buffer);

WriteFile (handle, s^, length (buffer) – 2, lpNumberOfBytesWritten, NIL);

if length (buffer) – 2<>lpNumberOfBytesWritten then

begin

ShowMessage ('Error when writing file. Number Of Bytes Written: '+IntToStr (lpNumberOfBytesWritten));

end;

CloseHandle (handle);

end;

Button1. Enabled:=True;

end;

Используя функции NetBIOS, процедура Tform2.NBName определяет. является ли адресат пользователем удалённой машины или данное имя является самим компьютером, а также, не является ли адресат рабочей группой сети Microsoft.

Также, часть данной процедуры определяет, запущена или нет на данной машине ещё одна программа «всплывающих сообщений». Так как для совместимости с аналогичными программами требуется указать одно и то же имя почтового слота (в данном случае, имя компьютера, на котором запущено данное приложение), то при наличии другой подобной программы показывается окно с кодом ошибки. Код ошибки определяется средствами NetBIOS и в данной дипломной работе реализована процедурой NBName.



Рис. 1.11. Ошибка при создании почтового слота

Коды ошибок, выдаваемые NetBIOS, указаны в техническом проекте.

*Структура приёма сообщений*

Прикладная программа, используя интерфейс WinAPI и встроенные функции Windows, постоянно опрашивает почтовый слот на наличие приходящего сообщения. В отличие от DOS, например, в котором данная реализация представляла бы бесконечный цикл и все ресурсы компьютера уходили бы на ожидание сообщения, операционная среда Windows позволяет создавать такие циклы без практической потери ресурсов, разделяя части программы на потоки (threads). При наличии сообщения в почтовом слоте оно передаётся в программу средствами NetBIOS. Почтовый слот может содержать в себе любое количество сообщений, независимо от того, когда они все будут востребованы данной программой.

Прием сообщений можно реализовать следующим образом:

Var L: Tlist;

procedure TForm1. ShowMess;

var

p:PChar;

s:string;

begin

Memo1. Clear;

p:=L. Items[Current];

s:='Message from ';

repeat

s:=s+p^;

p:=p+1;

until (p^=#0);

p:=p+1;

s:=s+' to';

repeat

s:=s+p^;

p:=p+1;

until (p^=#0);

Label1. Caption:=s;

p:=p+1;

s:='';

repeat

s:=s+p^;

p:=p+1;

until (p^=#0);

Memo1. Lines. Add (s);

StatusBar1. Panels[0].Text:='Current message: '+IntToStr (current+1);

StatusBar1. Panels[1].Text:='Total number of messages: '+IntToStr (L. Count);

end;

procedure TForm1. Timer1Timer (Sender: TObject);

var

lpNextSize:DWORD;

lpMessageCount:pointer;

buffer:PChar;

lpNumberOfBytesRead:DWORD;

MessageCount, all:DWORD;

s:string;

begin

lpMessageCount:=addr (MessageCount);

if GetMailslotInfo (h, nil, lpNextSize, lpMessageCount, nil) then

begin

if lpNextSize<>MAILSLOT\_NO\_MESSAGE then

begin

Beep;

all:=MessageCount;

while all<>0 do

begin

GetMem (buffer, lpNextSize);

if ReadFile (h, buffer^, lpNextSize, lpNumberOfBytesRead, nil) then

begin

s:='';

L. Add (buffer);

StatusBar1. Panels[1].Text:='Total number of messages: '+IntToStr (L. Count);

if L. Count=1 then begin current:=0; ShowMess; end;

GetMailslotInfo (h, nil, lpNextSize, lpMessageCount, nil);

end

else FreeMem (buffer);

all:=all‑1;

end;

end;

end;

end;

Процедура Timer1Timer является ядром получения сообщений, в то время как ShowMess – реализацией выдачи сообщения на экран в виде, понятном пользователю.

Строки

StatusBar1. Panels[0].Text:='Current message: '+IntToStr (current+1);

StatusBar1. Panels[1].Text:='Total number of messages: '+IntToStr (L. Count), входящие в состав данной процедуры, характеризуют номер сообщения по счёту всего количества сообщений и количество всех сообщений.

С учетом вышеописанного, пришедшее сообщение выглядит таким образом:

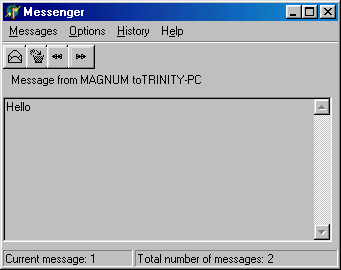


Рис. 1.12. Пример полученного сообщения

Используя кнопки навигации, можно легко просматривать все пришедшие сообщения, одновременно использую другие функции Messenger’a.

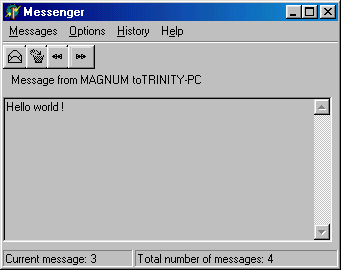


Рис. 1.13. Обзор пришедших сообщений

В данном разделе в части литературного обзора были рассмотрены общие положения, применяемые разработчиками программного обеспечения средств передачи информации в сетях Microsoft, даны основные подходы создания программ отправки и принятия сообщений; показан пример разработки данного программного обеспечения.

В разделе «Постановка задачи» были рассмотрены требования к системе.

Технический проект содержит общую структуру системы, структуру данных, связи между объектами, алгоритмические связи, простота использования подобных программ для пользователей, кратко описаны основные типы компонент и классов, используемые для написания программ отправки сообщений.

В рабочем проекте дана конфигурация технических средств, алгоритмы работы программы, структурная схема работы программы, показана иерархия форм, включая примеры диалога с пользователем и участки программного кода, описывающие важные процедуры системы отправки и принятия сообщений.

В целом представлена полная и объективная картина, отражающая содержание выполненных работ по проектированию программ работы с сетевыми протоколами операционных систем семейства Windows, созданию алгоритмов реализации аналогичного программного обеспечения и их программной реализации.

Разработка программ в среде Borland Delphi 5 относится к технологии RAD (Rapid Application Development) – быстрое создание приложений. До появления RAD‑средств делались попытки облегчить труд программиста, сделать этот труд более продуктивным, повысив этим быстродействие и удобство создания приложений. Первоначально появились интегрированные среды разработки – IDE (Integrated Development Environment), объединяющие компилятор, специальный текстовый редактор, ориентированный на работы с текстами на целевом языке программирования и средства отладки – трассировщик, дебаггер и другие. По сравнению с предыдущими средствами программирования это был большой шаг вперед, но затем появились CASE‑средства, которые позволяли максимально визуализировать процесс создания программы, но пользоваться ими было чересчур неудобно из-за их негибкости. В большинстве случаев написать достаточно сложную программу было очень трудно. Средства RAD, взяв все лучшее из интегрированных сред разработки и CASE систем, объединили гибкость работы с исходными текстами с удобством создания графического интерфейса пользователя – GUI (Graphic User Interface). К сожалению, большинство RAD‑средств не позволяют визуализировать логику самой программы, но и те методы, которые заложены в Delphi 5, позволяют поднять производительность разработки программ в 2–3 раза.

**Особенности разработки программ в среде Delphi 5**

Основными подходами к программированию на Delphi являются визуальная компонентность и многократное использование кода. Рассмотрю каждый из этих методов в отдельности.

*Визуальная компонентность*

Данный подход является логическим развитием объектно-ориентированного программирования (ООП). Каждый компонент имеет свои события (events), методы (methods) и свойства (properties). Методы и события пришли из объектно-ориентированного программирования, а свойства являются привилегией компонентного подхода. Свойства позволяют изменить состояние компонента, причем как динамически – во время работы программы, так и статически – визуально во время разработки. Естественно, что компоненты имеет все те особенности, что и объекты в объектно-ориентированном программировании – наследование, полиморфизм и инкапсуляция.

*Многократное использование кода*

Термин «многократное использование кода» (code reuse) говорит сам за себя. Данная технология подразумевает возможность написания таких участков программ, которые затем с небольшими изменениями (или совсем без изменений) могут быть использованы в другой программе. Нельзя сказать, что такой подход является привилегией RAD-средств: еще раньше программисты поняли преимущества возможности выделения универсальных и часто используемых подпрограмм в отдельные библиотеки (модули). Таким образом, появлялись библиотеки подпрограмм для различных языков программирования, работающие с графикой, базами данных, реализующие многие математические функции и прочие универсальные задачи. Но именно в RAD‑средствах технология многократного использования кода приобрела законченный и целостный вид – существует широкий набор стандартных компонентов, которые могут использоваться во многих программах различного типа.

Сложность данного подхода в изменении идеологии программирования – разработчик должен изначально ориентироваться на создания кода, который затем можно будет многократно использовать. Особенно актуальна эта проблема для групп разработчиков. В этом случае необходимо мыслить глобально, с учетом требований всех членов группы и перспективы дальнейших разработок.

Кроме описанных выше средств, существует также некоторый набор инструментов, назначение которого состоит в автоматизации рутинной работы по написанию операционных сред, графических интерфейсов и т.д.

**Cоздание программ в среде Delphi**

Cоздание программ в среде *Delphi* является удобным и простым делом. Для генерации программы первоначально необходимо создать файл *проекта*, имя которого будет совпадать с именем будущей программы. Причем *Delphi* сама сделает каркас программы – пользователю необходимо только выбрать соответствующий пункт в диалоговом окне ***New****,*которое показано на рис. 1:

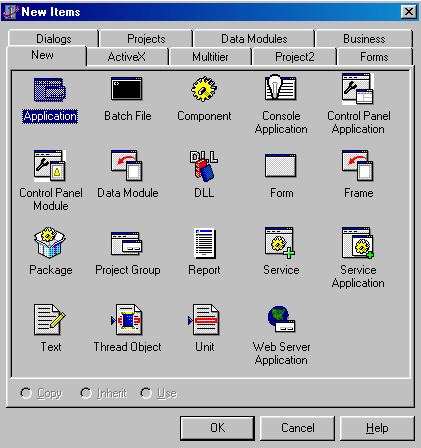


Рис. 2.1. Окно выбора объекта создания

Таким образом, выбрав тип создаваемой программы, программист получит готовый шаблон, который затем необходимо будет заполнить содержанием.

Каждое окно в *Delphi* называется формой. На форме располагаются различные визуальные и не визуальные компоненты. Весь набор компонент, доступных в системе, хранится в файле *complib.dcl* или в файле с другим именем и отображается в специальном окне среды *Delphi,* называемом «палитрой компонент» (component palette).

Палитра разбивается на страницы, группирующие компоненты по каким-либо общим признакам.



Рис. 2.2. Палитра компонент

Опишу основные компоненты палитры.

**Standard.** Большинство компонентов на этой странице являются аналогами экранных элементов самой Windows. Но компоненты Delphi обладают также некоторыми удобными дополнительными встроенными возможностям.

**Additional.** Эта страница содержит более развитые компоненты. Например, компонент Outline удобен для отображения информации с иерархической структурой, а MediaPlayer позволит вашим программам воспроизводить звук, музыку и видео. Данная страница также содержит компоненты, главное назначение которых – отображение графической информации. Компонент Image загружает и отображает растровые изображения, а компонент Shape, позволяет разместить на форме окружности, квадраты и т.д.   
**System.** Поскольку не каждая потребность, связанная с обработкой файлов, может быть удовлетворена с помощью стандартных диалоговых окон, страница System предоставляет возможность комбинировать отдельные элементы, такие как списки дисков, каталогов и файлов. Страница System также содержит компоненты, обрабатывающие обмен высокого уровня между программами посредством OLE (Object Linking and Embedding). А компонент Timer может генерировать события через определенные, заранее установленные промежутки времени.

**Win32.** Эта страница содержит компоненты, позволяющие созданным с помощью Delphi программам использовать такие нововведения в пользовательском интерфейсе 32‑разрядной Windows, как просмотр древовидных структур, просмотр списков, панель состояния, присутствующая в интерфейсе программы Windows Explorer (Проводник), расширенный текстовый редактор и др.

**Dialogs.** Windows 3.1 ввела в употребление стандартные диалоговые окна для операций над файлами, выбора шрифтов, цветов и т.д. Однако для использования их в обычной программе Windows может потребоваться написать немало вспомогательного кода. Страница Dialogs предоставляет программам Delphi простой доступ к этим стандартным диалоговым окнам.   
**Data Access и Data Controls.** Delphi использует механизм баз данных компании Borland (Borland Database Engine, BDE) для организации доступа к файлам баз данных различных форматов. Компоненты этих двух страниц облегчают программам Delphi использование сервиса баз данных, предоставляемого BDE, например многопользовательского считывания, записи, индексации и выдачи запросов для таблиц dBASE и Paradox. С использованием этих компонентов создание программы просмотра почти не требует программирования.   
**Internet.** Эта страница предоставляет компоненты для разработки приложений, позволяющих создавать HTML‑файлы непосредственно из файлов баз данных и других типов, взаимодействующих с другими приложениями для Internet. Delphi 4 дает возможность создавать приложения для Web‑сервера в виде DLL‑файлов; (Dynamic Link Library – Динамически компонуемая библиотека), способных содержать невизуальные компоненты. С помощью компонентов страницы Internet довольно просто создавать обработчики событий для обращения к определенному URL (Uniform Resource Locator – Унифицированный локатор ресурса), представлению документов в HTML‑формате и пересылки их клиент-программе.   
**ActiveX.** Эта страница содержит компоненты ActiveX, разработанные независимыми производителями программного обеспечения: сетка, диаграмма, средство проверки правописания.   
**Midas и Decision Cube.** Здесь собраны компоненты для доступа к удаленным серверам и осуществления SQL – запросов.

Гибкость среды *Delphi* позволяет настраивать палитру по своему усмотрению – объединять, разбивать страницы, переносить компоненты с одной страницы на другую и т.п. Для того, чтобы поместить компонент на форму, необходимо нажать на его изображении на палитре компонент кнопку «мыши», затем переместить указатель «мыши» на нужное место и там отпустить. Это процесс приведет к тому, что у класса формы появится поле типа этого компонента. Собрав на одну форму несколько компонентов, можно получит сколь угодно сложные диалоговые окна с полями ввода, мемо-полями и прочими компонентами. Простой пример создаваемого окна (в режиме разработки).

Разрабатываемая форма с несколькими визуальными компонентами. Хотя количество визуальных компонент на форме практически не ограничено, здравый смысл подсказывает, что перегружать форму ими не целесообразно, поскольку пользоваться программой с большим количеством кнопок, полей ввода и т.д. будет трудно из-за информационной перенасыщенности. Кроме того, рекомендуется каждый визуальный компонент снабжать строкой описания, чтобы пользователь мог догадаться, что же от него требуется.

Кроме визуальных, существуют также невизуальные компоненты. Поместив такой компонент на форму и запустив программу, мы ничего не увидим. Но некоторые компоненты во время работы себя отображают, правда для этого необходимо вызвать их методы. К таким компонентам относится всевозможные диалоговые компоненты, например *OpenDialog,* который является стандартным окном для открытия файла. Другие невизуальные компоненты внешне никак себя не проявляют, например компонент *DataSet.*

Свойства каждого компонента и формы в целом, а также события, на которые они откликаются, отображаются в окне инспектора объектов (object inspector). Инспектор объектов имеет две страницы – страницу со свойствами и страницу с событиями. Общий вид обеих страниц указан на рис. 2.4:

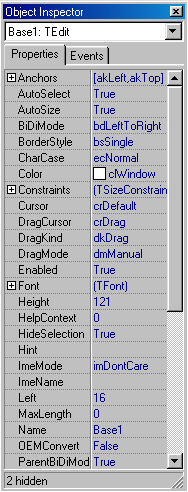


Рис. 2.4. Инспектор объектов

При редактировании свойства в инспекторе объектов учитывается тип этого свойства. Например, если свойство логического типа, то возможен будет выбор только лишь между значениями *True* и *False*. Для более сложных свойств, например, для списков строк, существуют свои редакторы свойств. Например, компонент *ComboBox* с рис. 8 имеет свойство *Items* типа списка строк – *TStrings*. В этом случае нажатие на кнопку с многоточием, расположенную рядом с именем свойства, приведет к активизации редактора списка строк. Его вид показан на рис. 5

Для нестандартных свойств, описанных программистом, существует возможность создания своего редактора свойств методами *Delphi.*

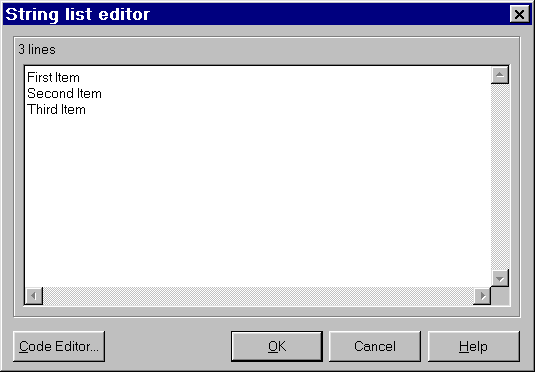


Рис. 2.5. Редактор списка строк

Кроме задания значений свойств, инспектор помогает задавать реакцию на всевозможные события, которые могут происходить в программе – перемещение указателя «мыши», нажатие и отпускание клавиш клавиатуры и т.п. На соответствующей страничке инспектора есть список множества стандартных для компонента (или формы) событий. Каждому событию соответствует своя процедура обработки этого события. Причем одна процедура может обрабатывать несколько событий. Все это указывается с помощью инспектора событий.

Замечу, что визуальными средствами среды *Delphi* построить законченное приложение практически невозможно. Такими средствами можно только облегчить создание программы. Всю логику придется писать старым методом – для каждой процедуры обработки событий вручную вписывать код.

Однако, существует определённая методика написания программы, призванная облегчить собственно разработку и сэкономить время на представление готового продукта и его последующую модернизацию.

**Основные этапы решения задач на ЭВМ**

*Составление проекта*

На основании анализа технического задания программист выбирает основной метод решения задачи, составляет общий проект программы. Выбранный подход к решению задачи должен обеспечивать правильные результаты для тех условий функционирования программы, которые определены ТЗ, гарантировать требуемую скорость работы, предусматривать удобство использования программы и т.п.

В проекте также характеризуются основные части проектируемой программы, их функции, взаимосвязь и последовательность выполнения, а также точно определяются входные данные и выдаваемые результаты как всей программы, так и основных ее частей.

*Алгоритмизация*

При разработке алгоритма необходимо учитывать ресурсы используемой ЭВМ (ее скорость, память) и возможности применяемой для решения задачи операционной системы. Алгоритмы для несложных задач, требования которых к ресурсам невелики, являются обычно машинно-независимыми. Алгоритм удобно записывать в виде блок-схем.

*Программирование*

В случае, когда на предыдущем этапе был получен детально разработанный алгоритм, составление программы на выбранном для программирования языке (алгоритмическом языке высокого уровня, автокоде, языке ассемблера или машинном языке) сводится к переводу этого алгоритма на язык программирования. Основная трудность при написании заключается в доскональном знании требований языка и постоянном внимании к деталям программы. Если этап *2.4.2.* был выполнен некачественно, программисту придётся дорабатывать программу «на ходу», во время написания.

*Трансляция*

Транслятор в ходе осуществления трансляции производит поиск синтаксических ошибок в программе и, в случае их обнаружения, выводит диагностику, помогающую последующей локализации ошибок. Трансляция, а вместе с ней и поиск синтаксических ошибок, могут быть прекращены, если найдена очень грубая ошибка.

*Отладка*

На этапе отладки производится обнаружение с помощью ЭВМ ошибок в программе и их исправление. Этап отладки можно разделить на три подэтапа:

6.1. Контроль правильности программы.

6.2. Локализация ошибок.

6.3. Исправление ошибок.

Подробно об отладке будет сказано в главе 2.5.

*Оформление программы*

Для возможности эксплуатации программы кем-либо кроме автора она должна быть оформлена: составлено ее описание, изготовлены машинные носители для передачи программы пользователям.

*Отчет о работе*

На основании результатов, полученных в ходе эксплуатации программы, составляется отчет о проделанной работе, оценивается выбранный метод решения задачи и эффективность программы; публикуются научные выводы.

*Модернизация*

При активной эксплуатации программы рано или поздно встаёт вопрос о модернизации старой программы или составлении новой, развивающей идеи, реализованные в прежней программе. Поэтому для уменьшения времени модернизации рекомендуется применять стуктурированное написание исходной программы.

В этой параграфе была рассмотрена упрощенная схема разработки программы в среде *Delphi.* Но кроме создания исходного текста необходимо помнить и об отладке программ.

**Методы и средства отладки**

Оказывается, практически невозможно составить реальную программу без ошибок, и почти невозможно для достаточно сложной программы быстро найти и устранить все имеющиеся в ней ошибки.

Таким образом, можно сказать, что наличие ошибок в только что разработанной программе это вполне нормальное и закономерное явление. А совсем ненормальным, из ряда вон выходящим фактом является отсутствие ошибок в программе, которая не была еще подвергнута тщательной отладке. Конечно, речь здесь идет о реальных, достаточно сложных программах. Учитывая этот печальный закон, разумно уже при разработке программы на этапах алгоритмизации и программирования готовиться к обнаружению ошибок на стадии отладки принимать профилактические меры по их предупреждению. Например, уже на первом этапе, когда выбирается общий метод решения задачи, следует разработать и основную стратегию отладки, общий план ее проведения. Необходимо уже на этой стадии разработки программы решить, каким путем можно будет в дальнейшем убедиться, что выбранный метод решения задачи обеспечивает правильные результаты для указанных в ТЗ диапазонов изменения величин и условий функционирования программы.

При разработке алгоритма программы решаются тактические вопросы проведения отладки, намечаются способы контроля отдельных блоков и приемы предстоящей локализации ошибок в них. Для этого проектируются контрольные примеры, по алгоритмам (блок-схемам) намечаются места и моменты необходимой отладочной печати и выбираются выводимые на печать данные, которые должны обеспечить возможность быстрой локализации ошибок при отладке. Разрабатывая алгоритм, следует, таким образом, учитывать, можно ли будет достаточно просто проконтролировать программу, составленную по выбранному алгоритму, и в случае, когда предвидятся большие затруднения, нужно отдать предпочтение другому, более выгодному для этапа отладки, алгоритму. Нужно всегда помнить, что главным критерием ценности программы является ее правильность, и для гарантирования такого свойства программы следует жертвовать другими показателями, такими, например, как скорость работы или требуемый объем памяти. Давно ушли в прошлое те времена, когда программу оценивали только по количеству команд в ней.

Вообще, можно считать, что именно на этапах алгоритмизации и программирования закладывается фундамент быстрой и успешной отладки. Чем более тщательно проведены этапы постановки задачи, и в частности, чем более детально разработан план отладки, чем больше внимания было уделено проверке составленного алгоритма и программы, а также применению отладочных средств для облегчения предстоящей отладки, тем меньше времени потребуется на проведение самой отладки и тем скорее будут получены на ЭВМ результаты, которым можно будет доверять. Наоборот, стремление к максимально быстрому составлению программы с целью скорейшего начала (и окончания) ее отладки, приводит к обратному результату: отладка затягивается надолго, и получение достоверных результатов откладывается на длительный срок. Такая задержка вызывается тем, что уже на поздних этапах отладки (а иногда – и счета) вскрываются все новые и новые ошибки, допущенные на этапах алгоритмизации и программирования, и приходится тратить много времени на их обнаружение и исправление. Кроме того, каждое выполнение внутреннего цикла требует обращения к ЭВМ за получением новых результатов, на ожидание которых тратится обычно от нескольких часов до нескольких дней. Напрашивается вывод, что для ускорения отладки нужно сократить кратности указанных циклов, выходя на машину с программой уже тщательно проверенной и подготовленной заранее к отладке.

Перечислю содержание этапов 1–6, имеющих целью облегчить предстоящую отладку и сократить время ее проведения.

1. Разработка общего плана проведения отладки, общей методики проверки правильности составленной программы, а также системы необходимых для отладки контрольных примеров.

2. Проверка разработанных алгоритмов, выбор отладочных средств и определение контролируемых ими мест, участков, величин.

3. Проверка составленной программы, реализация намеченного ранее плана использования отладочных средств для получения на ЭВМ необходимых при локализации ошибок тестовых результатов; изготовление эталонных результатов для тестов.

4. Ввод, печать и сверка текста программы, перенесенной на внешние носители.

5. Получение с помощью транслятора вспомогательных таблиц (например, таблица перекрестных ссылок) и проверка их.

6. Тщательный контроль первых результатов, получаемых по новой программе.

У начинающих программистов изложенный плановый подход к проведению отладки (этапы 1 и 2) вызывает вначале трудности, поскольку им приходится разрабатывать план отладки для несуществующей пока программы. Но нет другого пути освоить этот эффективный способ, кроме как развивать в себе навыки планирования своей работы и предвидения особенностей предстоящей отладки программы по ее проекту и общим алгоритмам. Чем на более ранней стадии разработки программист начинает заниматься вопросами отладки программы, тем меньше неприятных неожиданностей ожидает его в будущем. Надежды на то, что устранение ошибок из программы и получение правильных результатов произойдет как-то само собой, без затраты особых усилий, никогда не оправдываются. Вообще, оптимизм и самоуверенность для программиста на стадии разработки противопоказаны; они могут являться полезными только на стадии отладки при затяжной борьбе с очень глубоко скрытыми ошибками.

Примерное распределение между этапами общего времени, необходимого для разработки достаточно сложных программ, выглядит следующим образом [1, 6]:

1. Получение задания, составление проекта программы и общего плана отладки 10%.

2. Разработка алгоритма (15%) и детального плана отладки 20%.

3. Программирование (5%) и изготовление тестов 15%.

4. Подготовка и первая трансляция 5%.

5. Отладка 40%.

6. Оформление программы 10%.

Приведенные цифры отражают тот факт, что в процессе разработки программы работы по доказательству (демонстрации) правильности разрабатываемой программы равнозначны работам по ее изготовлению (проектированию, алгоритмизации и написанию), что можно выразить следующей формулой:

разработка программы = изготовление + доказательство.

Поэтому программой следовало бы называть только такую программу, которая выдает правильные результаты, а то, что еще не прошло стадию доказательства правильности, является не программой, а ее полуфабрикатом.

Конечно, для простых задач распределение времени между этапами будет несколько другим, за счет увеличения доли программирования по отношению к алгоритмизации и отладке. Но, как правило, время, затрачиваемое на работы, связанные с отладкой, составляет около половины всего времени, необходимого на разработку программы. Поэтому вопрос минимизации времени, необходимого на отладку, имеет особое значение. К его решению можно подойти с двух сторон:

а) путем ускорения поиска и исправления ошибок, имеющихся в программе;

б) путем уменьшения количества ошибок, допускаемых при разработке алгоритма и составлении программы.

*Контроль программы*

Подэтап контроля программы характеризуется как этап решения задачи, целью которого является установление наличия ошибок в составленной программе или убедительная демонстрация их отсутствия. Если будет установлено, что ошибки в программе имеются, то на следующем этапе (этап локализации) будет производиться их поиск. Поэтому в задачу контроля входит также получение еще и такой информации о характере работы программы, которая могла бы помочь в дальнейшем при поиске ошибок.

# ***Контроль текста***

Контроль текста программы можно производить как «вручную», так и с применением ЭВМ. Сначала рассмотрим «ручные» методы контроля текста программ (алгоритмов).

Можно различать три способов контроля текста без применения ЭВМ: просмотр, проверка и прокрутка.

***Просмотр***

Текст составленной программы внимательно просматривается на предмет обнаружения ошибок, описок и смысловых расхождений с текстом алгоритма, по которому производилось программирование. Помимо сплошного просмотра применяется еще и выборочный просмотр некоторых фрагментов программы.

***Проверка***

При проверке программы программист по тексту программы мысленно старается восстановить тот вычислительный процесс, который определяет программа, после чего сверяет его с требуемым процессом, т.е. ТЗ, определенном в проекте.

***Прокрутка***

Другим способ контроля программ и алгоритмов за столом является прокрутка (иногда ее называют «сухой» прокруткой – dry running – для отличия от метода прокрутки, применяемого на этапе локализации и использующего ЭВМ.) основой прокрутки является имитация программистом выполнения программы на машине, с целью более конкретного и наглядного представления о процессе, определяемом текстом проверяемой программы. Прокрутка дает возможность приблизить последовательность проверки программы к последовательности ее выполнения, что позволяет проверять программу как бы в динамике её работы, проверять элементы вычислительного процесса, задаваемого проверяемой программой, а не только статичный текст программы. Для выполнения прокрутки обычно приходится задаваться какими-то исходными данными и производить над ними необходимые вычисления.

***Печать текста***

Для проверки правильности препарации программы (переноса текста программы на какие-либо машинные носители) после ввода в машину производится на печать текста программы для последующей его сверки с исходным текстом.

*Контроль результатов*

Как бы ни была тщательно проверена и прокручена программа за столом, решающим этапом, устанавливающим ее пригодность для работы, является контроль программы по результатам ее выполнения на ЭВМ. Наиболее универсальным методом проверки для всех классов задач является метод контрольных тестов или тестирование. Тестом будем называть информацию, состоящую из исходных данных, специально подобранных для отлаживаемой программы, и из соответствующих им эталонных результатов (не только окончательных, но и промежуточных), используемых в дальнейшем для контроля правильности работы программы.

*Тестирование*

Под тестированием следует понимать процесс исполнения программы с целью обнаружения ошибок, в качестве которых принимается любое отклонение от эталонов. Хорошим считается тест, который имеет высокую вероятность обнаружения еще не выявленных ошибок.

Под отладкой понимается процесс, позволяющий получить программу, функционирующую с требуемыми характеристиками в заданной области входных данных. Таким образом, в результате отладки программа должна соответствовать некоторой фиксированной совокупности правил и показателей качества, принимаемой за эталонную для данной программы.

Существует 3 основных способа тестирования: алгоритмическое, аналитическое, содержательное.

***Алгоритмическое тестирование***

Алгоритмическое тестирование применяется программистом для контроля этапов алгоритмизации и программирования. Программисты проектируют тесты и начинают готовить эталонные результаты на этапе алгоритмизации, а используют их на этапе отладки.

***Функциональное или аналитическое тестирование***

Аналитическое тестирование служит для контроля выбранного метода решения задачи, правильности его работы в выбранных режимах и с установленными диапазонами данных. Тесты проектируют и начинают готовить сразу после выбора метода, а используют их на последнем этапе отладки или для анализа результатов пробного счета; в ходе тестирования, наряду со сверкой на совпадение, применяются и качественные оценки результатов.

***Содержательное тестирование***

Содержательное тестирование служит для проверки правильности постановки задачи. Для контроля при этом используются, как правило, качественные оценки и статистические характеристики программы, физический смысл полученных результатов и т.п. в проведении содержательного тестирования, принципы которого формулируются в техническом задании, самое активное участие должны принимать заказчики или идущие пользователи программы.

Содержательные и аналитические тесты проверяют правильность работы программы в целом или крупных ее частей, в то время, как алгоритмические тесты в первую очередь должны проверять работу отдельных блоков или операторов программы.

## **Типы тестов**

Тот вид контроля, который рассматривался выше, можно назвать тестированием основных функциональных возможностей программы – основной тест.

*Вырожденный тест*

Этот тест затрагивает работу программы в самой минимальной степени. Обычно тест служит для проверки правильности выполнения самых внешних функций программы, например, обращения к ней и выхода из нее.

*Тест граничных значений*

Тест проверяет работу программы для граничных значений параметров, определяющих вычислительный процесс. Часто для граничных значений параметра работа программы носит особый характер, который, тем самым, требует и особого контроля.

*Аварийный тест*

Тест проверяет реакцию программы на возникновение разного рода аварийных ситуаций в программе, в частности, вызванных неправильными исходными данными. Другими словами, проверяется диагностика, выдаваемая программой, а также окончание ее работы или, может быть, попытка исправления неверных исходных данных.

Помимо автономных тестов, предназначенных для контроля отдельных блоков программы, можно выделить стыковочные и комплексные тесты.

*Стыковочные тесты*

Предназначаются для проверки взаимосвязи (стыковки) уже отлаженных частей программы.

*Комплексные тесты*

Проверяют правильность работы всех или большинства частей программы после их объединения.

**Локализация ошибок**

*Способы локализации*

После того, как с помощью контрольных тестов (или каким либо другим путем) установлено, что в программе или в конкретном ее блоке имеется ошибка, возникает задача ее локализации, то есть установления точного места в программе, где находится ошибка.

Процесс локализации ошибок состоит из следующих трех компонент:

1. Получение на машине тестовых результатов.

2. Анализ тестовых результатов и сверка их с эталонными.

3. Выявление ошибки или формулировка предположения о характере и месте ошибки в программе.

По принципам работы средства локализации разделяются на 4 типа:

1. Аварийная печать.

2. Печать в узлах.

3. Слежение.

4. Прокрутка.

*АВАРИЙНАЯ ПЕЧАТЬ* осуществляется один раз при работе отлаживаемой программы, в момент возникновения аварийной ситуации в программе, препятствующей ее нормальному выполнению. Тем самым, конкретное место включения в работу аварийной печати определяется автоматически без использования информации от программиста, который должен только определить список выдаваемых на печать переменных.

*ПЕЧАТЬ В УЗЛАХ* включается в работу в выбранных программистом местах программы; после осуществления печати значений данных переменных продолжается выполнение отлаживаемой программы.

*СЛЕЖЕНИЕ* производится или по всей программе, или на заданном программистом участке. Причем слежение может осуществляться как за переменными (арифметическое слежение), так и за операторами (логическое слежение). Если обнаруживается, что происходит присваивание заданной переменной или выполнение оператора с заданной меткой, то производится печать имени переменной или метки и выполнение программы продолжается. Отличием от печати в узлах является то, что место печати может точно и не определяться программистом (для арифметического слежения); отличается также и содержание печати.

*ПРОКРУТКА* производится на заданных участках программы, и после выполнения каждого оператора заданного типа (например, присваивания или помеченного) происходит отладочная печать.

По типам печатаемых значений (числовые и текстовые или меточные) средства разделяются на арифметические и логические.

**Интегрированный отладчик Delphi**

Для работы со встроенным отладчиком Delphi 5 его интегрированная среда разработки (IDE) предлагает целую серию установок, которых практически хватит для тестирования и отладки любой задачи.

Сначала рассмотрю опции отладки:

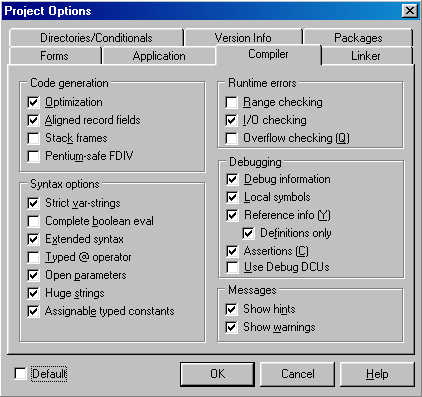


Рис. 2.6. Опции отладки проекта

Включение отладочной информации регулируется следующими установками:

* Debug Information. Опция контролирует включение отладочной информации. При отключении этой опции будет невозможно трассировать код или ставить точки прерывания в любом модуле. Опция эквивалентна директивам компилятора $D и $DEBUGINFO
* Local Symbols. Опция контролирует включение информации о локальных переменных, декларированных, например, внутри функций, процедур и раздела implementation. Эквивалентные директивы компилятора – $L и $LOCALSYMBOLS.
* Symbol Info. Эту опцию нельзя целиком отнести к разряду отладочных, так как ее действие направлено на броузер объектов, а не на встроенный отладчик. Если опция включена, броузер объектов сможет выводить информацию для объектов, определенных в модулях. Опция игнорируется при выключенных предыдущих двух опциях Эквивалентные директивы компилятора – $Y и $REFERENCEINFO.

Если модуль Delphi распространяется в виде DCU‑файлов (например, VCL) необходимо позаботиться о том, чтобы в скомпилированных модулях не содержалась отладочная информация.

**Пошаговая отладка**

Одна из самых распространенных задач отладки – выполнение программы шаг за шагом, по одной строке за раз для проверки правильности выполнения. При пошаговом прохождении кода отладчик выводит окно редактирования с выполняемой программой. Точка выполнения, показывающая следующую выполняемую строку программы, представляется в виде зеленой стрелки, расположенной слева от области исходного текста в окне редактирования.

После успешной компиляции модуля на полосе отладочной информации каждая строка кода, внесшая свой вклад в модуль, будет отмечена синим кружком.

Интегрированная среда Delphi предоставляет пользователю несколько команд пошаговой отладки доступных в меню Run:

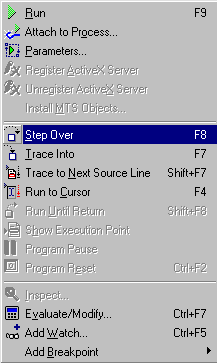


Рис. 2.7. Средства отладки программы

Ниже перечислю команды отладчика:

* Run. Выбор этой команды запускает приложение на выполнение в обычном режиме. Эту команду можно использовать как для запуска приложения, так и для продолжения его работы после какого-либо прерывания выполнения (например, по точке останова). Если включена опция Break on Exception, необходимо использовать команду для продолжения работы после получения сообщения об исключительной ситуации.
* Step Over. Когда точка выполнения находится на строке содержащей вызов процедуры или функции, необходимо использовать эту команду для выполнения строки, включая вызовы в один шаг, без прохождения отдельных строк вызываемых функций. Точка выполнения перемещается при выполнении на следующую строку.
* Trace Into. В отличие от предыдущей команды, эта опция отработает пошаговую отладку вызываемых процедур и функций. Другими словами, если, например, в строке вызывается некая процедура, то при выполнении этой команды точка выполнения перейдет на первую строку процедуры. Однако если в строке нет таких вызовов, значит, последние две команды идентичны. Но, необходимо соблюдать осторожность – при пошаговой трассировке обработчика события OnPaint требуется перерисовка окна приложения, для чего вызывается обработчик события OnPaint. Получается бесконечный цикл вызовов одного и того же обработчика. Тем не менее, стоит лишь проследить, чтобы окна приложения и редактора не перекрывались.
* Trace to Next Source Line. Иногда код вызывает другой код косвенно, например, при вызове функции, которая запускает обработчик события, или при вызове функции Windows API, которая, в свою очередь, запускает функцию косвенного вызова. Поскольку такие вызовы косвенные, отладчик не видит вызова и не отслеживает пошагового выполнения таких вызовов**.** Однако использование описываемой команды приводит к отслеживанию таких вызовов и останову отладчика на первой строке вызываемой таким образом функции или процедуры.
* Run to Cursor. Очень часто невозможно в поисках ошибки, местоположение которой с какой-то точностью известно, пошагово добираться до нужного места через сотни, а то и тысячи строк кода. В таком случае можно просто поместить курсор на нужную строку программы в окне редактирования и используйте команду Run to Cursor. Эти действия эквивалентны временному помещению точки останова в необходимую строку программы, и после выполнения предшествующего строке кода работа программы приостанавливается.
* Show Execution Point. Эта команда заставляет среду разработки открыть окно редактора и показать выполняемую в настоящее время строку программы.
* Program Pause. Выбор этой команды немедленно останавливает выполнение программы. Команда особенно полезна при зацикливании программы.
* Program Reset. Команда немедленно прекратит выполнение программы и произойдет возврат в среду разработчика.

Залогом успешного продвижения на рынке нового товара или новой услуги является всестороннее ознакомление со структурой данного рынка, от его (рынка) требований и возможностей. Исследуемая информация касается спроса и предложения конкурентов, а также ожиданий конкретных потребителей. При этом реализация направленности товара на весь рынок в целом зачастую представляется нереальной, что можно объяснить заметной неоднородностью рынка, подразумевающей различные потребности отдельных слоев потребителей. Разница в потребностях может быть вызвана различными причинами – социальными, финансовыми, географическими и многими другими. Применительно к программному обеспечению можно сказать, что различные организации и частные лица используют в своей работе различные компьютеры и на данный момент, когда существует более десятка устоявшихся компьютерных платформ, было бы не разумным пытаться охватить весь рынок целиком и поэтому рынок должен быть *сегментирован*, то есть поделен на однородные части-сегменты, в рамках которых потребитель имеет сходные возможности и потребности.

При разумном делении рынка на сегменты все инструменты маркетинга внутри него могут быть оптимально скоординированы. Именно поэтому сегментация рынка считается очень важным аспектом деятельности предприятия.

### *Методика определения сегментов рынка*

При первичной сегментации рынка целесообразно выделить сегменты товаров потребительского спроса и производственного назначения. Такая классификация важна, поскольку подчеркивает различия в характеристиках продуктов и последствиях для маркетолога.

Для дальнейшего деления рынка на сегменты можно воспользоваться различными критериями в зависимости от следующих факторов:

географического положения потребителей (регион, страна);

типа потребителя (величина предприятия, интенсивность потребления, отрасль, место в производственном процессе);

типа процесса, для которого приобретается продукция (административная деятельность, движение товара, производственный процесс);

покупательского спроса (клиент / потенциальный клиент, связь с поставщиком, частота и величина закупок);

На рынках сбыта товаров широкого потребления используют другие критерии. Классическими являются следующие показатели:

социально-экономические (образования, доходы);

демографические (возраст, пол, состав семьи);

географические

Однако следует учитывать, что всех потребителей на рынке не так-то легко разделить по категориям. Поведение потребителя в последнее время становиться все более дифференцированным, возникают различные «стили жизни» внутри общества.

Для формирования сегментации рынка используются элементы таксономического анализа – построение диаграмм Чекановского [1]. Исходным шагом, предопределяющим правильность конечных результатов, является оформление матрицы наблюдений. Эта матрица содержит наиболее полную характеристику изучаемого множества объектов и имеет вид:

X=



где w – число объектов; n – число признаков; ik – значение признака k для объекта i.

Признаки, включенные в матрицу, могут быть неоднородны, поскольку описывают разные свойства объектов. Кроме того, различаются единицы их измерения. Поэтому надлежит выполнить предварительное преобразование, которое заключается в стандартизации признаков.

Таблица 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 |  | W |
| 1 | X | - | X | - |
| 2 | - | - | - | X |
| … | … | … | … | … |
| W | - | X | - | X |

В приведенной неупорядоченной диаграмме очередность записи случайна. На это указывает явственный разброс символов, обозначающих разницу между изучаемыми элементами: наименьшее численное расстояние – «Χ»; наибольшее расстояние, т.е. пары элементов, наиболее разнящиеся между собой, – «–». Для их линейного упорядочения следует произвести перегруппировку знаков «Χ» и «–». Перегруппировка должна выполняться таким образом, чтобы знаки «X» оказались как можно ближе к главной диагонали диаграммы. С этой целью строки и столбцы таблицы переставляются до тех пор, пока не получится упорядоченная диаграмма.

### *Поиск сегментов рынка для программы посылки сообщений*

Рынок данных программ достаточно узок и специализирован. Программа предназначена для облегчения работы Интернет-компаний, людей, которые работают в локальных сетях.

Для проведения сегментации можно выделить следующие группы потенциальных потребителей программ подобного уровня:

Средние и крупные промышленные предприятия;

Научные центры;

Интернет-компании;

Учащиеся и студенты;

Небольшая группа пользователей;

Теперь рассмотрим характеристики данного программного обеспечения:

Распространенность аппаратно-программной платформы;

Интерфейс;

Удобство работы;

Цена.

Теперь составим таблицу наблюдений, оценив в баллах каждый признак для каждого потребителя по десятибалльной шкале.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Параметры  Потребители | Платформа | Интерфейс | Удобство работы | Цена |
| Промышленные предприятия | 1 | 5 | 9 | 1 |
| Научные центры | 7 | 4 | 8 | 2 |
| Интернет-компании | 5 | 7 | 10 | 10 |
| Учащиеся и студенты | 1 | 10 | 6 | 10 |
| Группа пользователей | 2 | 9 | 10 | 10 |

X=



;



; ;



; ;



; ;



;



Итак, мы получили матрицу расстояний:

***С***=



Для получения неупорядоченной диаграммы Чекановского разбиваем полученные значения на классы по правилу: если элемент меньше 2.5, то на его место ставится «X», иначе ничего не ставим.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 |
| 1 | X | X |  |  |  |
| 2 | X | X |  |  |  |
| 3 |  |  | X |  | X |
| 4 |  |  |  | X | X |
| 5 |  |  | X | X | X |

Проведя перегруппировку столбцов и строк путем перестановки сначала 4-го и 5-го столбцов, а затем таких же строчек, получаем упорядоченную диаграмму Чекановского:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 5 | 4 |
| 1 | X | X |  |  |  |
| 2 | X | X |  |  |  |
| 3 |  |  | X | X |  |
| 5 |  |  | X | X | X |
| 4 |  |  |  | X | X |

В результате мы выделили три сегмента – в одном из них располагаются потребители 1 и 2, в другом – 3 и 5, в третьем – 5 и 4.

Сегментация рынка привела к следующим результатам – были выделены три сегмента:

Промышленные предприятия и научные центры;

Интернет-компании и группы пользователей.

Учащиеся и студенты и группы пользователей.

Для первой сегмента более важны требования к удобству работы потому что данный сегмент занимается наукоемким исследованиями.

Для второй группы более важными оказываются цена, удобство интерфейса, а также доступность и распространенность программно-аппаратной платформы, потому что для мелких малобюджетных организаций и групп пользователей возможность использования имеющегося парка машин класса IBM PC с установленными на них операционной системой Windows'9x, является хорошим стимулом для приобретения данного программного продукта.