МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Государственное образовательное учреждение
высшего профессионального образования -
«Оренбургский государственный университет»

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

А.Ю. ВЛАДОВА

ВЫЧИСЛИТЕЛЬНЫЕ СЕТИ И СЕТЕВОЕ ПРОГРАММИРОВАНИЕ

Лабораторный практикум

Рекомендовано к изданию Редакционно – издательским советом государственного образовательного учреждения высшего профессионального образования - «Оренбургский государственный университет»

Оренбург 2003

ББК 32.973.202-018.2я73
В 57
УДК 681.324(075)

Рецензенты
доктор технических наук, профессор Н. А. Соловьев,
кандидат физико-математических наук, доцент Е. А. Корнев

Владова А.Ю.

В 57 **Вычислительные сети и сетевое программирование: Лабораторный практикум. - Оренбург: ГОУ ВПО ОГУ, 2003. – 28 с.: ил.**

В лабораторном практикуме изложены основы диагностики сети, управления общими ресурсами и вопросы сетевого программирования. Практикум содержит методические указания к выполнению лабораторных и самостоятельных работ.

Лабораторный практикум подготовлен на кафедре «Программное обеспечение вычислительных средств и автоматизированных систем» и предназначен для студентов старших курсов специальности ПОВТАС по дисциплинам «Архитектуры вычислительных систем и сетей» и «Программное обеспечение сетей ЭВМ».

ББК 32.973.202-018.2я73

Введение

В лабораторном практикуме рассматривается круг вопросов, связанных с основами разработки сетевых приложений и сетевой диагностики. Целью лабораторного практикума является развитие у студентов-программистов навыков разработки прикладных программ для обмена данными. Для достижения поставленной цели рассмотрены следующие задачи:

- этапы диагностики сети;
- клиент-серверная модель обмена данными в сети;
- основные принципы сетевого программирования.

В рамках первой задачи разработана лабораторная работа №1 «Основы диагностики сети», в которой описаны основные сетевые утилиты операционных систем семейства Windows для определения IP-адреса, имени компьютера, посылка сообщений, подключения сетевых ресурсов. Приводятся примеры работы с утилитами.

Решение второй задачи раскрывается в лабораторной работе №2 «Обмен сообщениями на базе сетевых компонентов Delphi**»,** в которой, на базе компонентов TClientSocket и TServerSocket среды программирования Delphi, дан пример разработки «чата». Проведен анализ возможностей аналогичных сетевых компонентов, даны рекомендации по их применению.

Лабораторная работа №3 «Передача сообщений на базе функций библиотеки WinSock» решает задачу изучения основных принципов сетевого программирования на основе использования библиотеки WinSock, которая позволяет на низком уровне создавать приложения для обмена данными. Оформлены примеры вызова основных функций библиотеки.

В качестве языка программирования, в котором отлаживались приведенные примеры, выбран Delphi, как мощный инструмент для создания надежных серверных и клиентских приложений. Все теоретические сведения подкреплены множеством примеров, которые могут служить базой при разработке полноценного программного обеспечения в данной области.

Отчет по каждой лабораторной работе должен включать титульный лист, постановку задачи, теоретические сведения, иерархическую схему процедур, текст основных процедур, результаты работы и выводы.

1 Лабораторная работа №1. Основы диагностики сети

1.1 Постановка задачи

Используя стандартные сетевые утилиты, проанализировать конфигурацию сети на платформе ОС Windows, т.е. получить свой IP-адрес, узнать имя домена, имена компьютеров, входящих в домен, просмотреть и при необходимости подключить общие ресурсы, определить причину возможных неполадок, так же получить информацию об использовании портов.

1.2 Краткая теоретическая справка

Мониторинг и анализ сети представляют собой важные этапы контроля работы сети. Для решения этих задач регулярно производится сбор данных, который дает базу для измерения реакции сети на изменения и перегрузки. Чтобы осуществить сетевую передачу, нужно проверить корректность подключения клиента к сети, наличие у клиента хотя бы одного протокола сервера, знать IP-адрес компьютеров сети и т. д. Поэтому в сетевых операционных системах, и в частности, в Windows, существует множество мощных утилит для пересылки текстовых сообщений, управления общими ресурсами, диагностике сетевых подключений, поиска и обработки ошибок. Утилиты запускаются из командной строки или из сеанса MS DOS.

1.3 Сетевые утилиты

1.3.1 Утилита **hostname**

Выводит имя локального компьютера (хоста). Она доступна только после установки поддержки протокола TCP/IP. Пример вызова команды *hostname*[1)]:

```
G:\UTILl>hostname
14423-5
```

1.3.2 Утилита **ipconfig**

Выводит диагностическую информацию о конфигурации сети TCP/IP. Эта утилита позволяет просмотреть текущую конфигурацию IP-адресов компьютеров сети. Синтаксис утилиты *ipconfig*:

ipconfig [/*all* | */renew* [адаптер] | */release* [адаптер]],

[1)] В разработке примеров принимал участие Цыганков А. С.

где *all* - выводит сведения о имени хоста, DNS (Domain Name Service), типе узла, IP-маршрутизации и др. Без этого параметра команда *ipconfig* выводит только IP-адреса, маску подсети и основной шлюз;

/renew [адаптер] - обновляет параметры конфигурации DHCP (Dynamic Host Configuration Protocol – автоматическая настройка IP-адресов). Эта возможность доступна только на компьютерах, где запущена служба клиента DHCP. Для задания адаптера используется имя, выводимое командой *ipconfig* без параметров;

/release [адаптер] - очищает текущую конфигурацию DHCP. Эта возможность отключает TCP/IP на локальных компьютерах и доступна только на клиентах DHCP. Для задания адаптера используется имя, выводимое командой *ipconfig* без параметров. Эта команда часто используется перед перемещением компьютера в другую сеть. После использования утилиты *ipconfig /release*, IP-адрес становиться доступен для назначения другому компьютеру.

Запущенная без параметров, команда *ipconfig* выводит полную конфигурацию TCP/IP, включая IP адреса и маску подсети.

Примеры использования *ipconfig* без параметров и с ключом */all*:

- без параметров:

```
C:\Program Files>ipconfig

Настройка протокола IP для Windows 2000

Адаптер Ethernet  Подключение по локальной сети:

        DNS суффикс этого подключения  :
        IP – адрес                     : 192.168.144.235
        Маска подсети                  : 255.255.248.0
        Основной шлюз                  : 192.168.144.211
```

- с ключом */all*

```
Настройка протокола IP для Windows 2000

        Имя компьютера                 : 14423-5
        Основной DNS суффикс           : FIT.local
        Тип узла                       : Широковещательный
        Включена IP-маршрутизация      : Нет
        Доверенный WINS-сервер         : Нет
        Порядок просмотра суффиксов DNS : FIT.local

Адаптер Ethernet  Подключение по локальной сети:

        DNS суффикс этого подключения
        Описания
        Физический адрес               : 00-02-44-12-07-A7
        DHCP разрешен                  : Нет
        IP-адрес                       : 192.168.144.235
        Маска подсети                  : 255.255.248.0
        Основной шлюз                  : 192.168.144.211
```

DNS-серверы : 192.168.144.211

1.3.3 Утилита net view

Просматривает список доменов, компьютеров или общих ресурсов на данном компьютере. Синтаксис утилиты *net view*:

net view [*компьютер* | */domain*[:домен]];
net view /network:nw [\\компьютер] – используется в сетях Novell NetWare,

где *компьютер* - задает имя компьютера для просмотра общих ресурсов;
/domain[:домен] - задает домен, для которого выводится список компьютеров. Если параметр не указан, выводятся сведения обо всех доменах в сети;
/network:nw - выводит все доступные серверы в сети Novell NetWare. Если указано имя компьютера, выводится список его ресурсов в сети NetWare. С помощью этого ключа могут быть просмотрены ресурсы и в других локальных сетях.
Вызванная без параметров, утилита выводит список компьютеров в текущем домене.
Пример использования утилиты *net view*:

- без параметров:

```
C:\Program Files\Far>net view
Имя сервера                     Заметки

-------------------------------------------------------
\\14404-1
\\14422-1
\\14422-2
\14422-3
\\14423-10
\\14423-2
\\14423-3
\\14423-4
\\14423-5
\\14423-6
\\14423-7
\\14423-8
\\14423-9
\\14424-2                       USER
\\FIT-S1
Команда выполнена успешно
```

- с параметром *компьютер*:

```
C:\program Files\Far>net view 14423-8
Общие ресурсы на 14423-8
```

```
Сетевое имя          Тип          Использовать как          Комментарий
-------------------------------------------------------------------------------------------------
hdd_c                Диск
hdd_d                Диск
Команда выполнена успешно.
```

1.3.4 Утилита **ping**

Проверяет соединения с удаленным компьютером или компьютерами. Эта команда доступна только после установки поддержки протокола TCP/IP. Синтаксис утилиты *ping*:

> *ping* [*-t*] [*-a*] [*-n счетчик*] [*-l длина*] [*-f*] [*-i ttl*] [*-v тип*] [*-r счетчик*] [*-s число*] [[*-j список_комп*] | [*-k список_комп*]] [*-w интервал*] *список_назн,*

где *-t* - повторяет запросы к удаленному компьютеру, пока программа не будет остановлена;
-a - разрешает имя компьютера в адрес;
-n счетчик - передается число пакетов ECHO, заданное параметром. По умолчанию – 4;
-l длина - отправляются пакеты типа ECHO, содержащие порцию данных заданной длины. По умолчанию - 32 байта, максимум – 65527;
-f - отправляет пакеты с флагом запрещения фрагментации (Do not Fragment). Пакеты не будут разрываться при прохождении шлюзов на своем маршруте;
-i ttl - устанавливает время жизни пакетов TTL (Time To Live);
-v тип - устанавливает тип службы (Type Of Service) пакетов;
-r счетчик - записывает маршрут отправленных и возвращенных пакетов в поле записи маршрута Record Route. Параметр счетчик задает число компьютеров в интервале от 1 до 9;
-s число - задает число ретрансляций на маршруте, где делается отметка времени;
-j список_комп - направляет пакеты по маршруту, задаваемому параметром список_комп. Компьютеры в списке могут быть разделены промежуточными шлюзами (свободная маршрутизация). Максимальное количество, разрешаемое протоколом IP, равно 9;
-k список_комп - направляет пакеты по маршруту, задаваемому параметром список_комп. Компьютеры в списке не могут быть разделены промежуточными шлюзами (ограниченная маршрутизация). Максимальное количество, разрешаемое протоколом IP, равно 9;
-w интервал - указывает промежуток времени ожидания (в миллисекундах);

список_назн - указывает список компьютеров, которым направляются запросы;

Пример использования утилиты *ping* с параметром *список_назн*:

```
C:\Program Files\Far>ping 14423-8

Обмен пакетами с  14423-8.FIT.local [192.168.144.238] по 32 байт

Ответ  от 192.168.144.238: число байт=32 время<10мс  TTL=128
Ответ  от 192.168.144.238: число байт=32 время<10мс  TTL=128
Ответ  от 192.168.144.238: число байт=32 время<10мс  TTL=128
Ответ  от 192.168.144.238: число байт=32 время<10мс  TTL=128

Статистика Ping для  192.168.144.238:
    Пакетов: отправлено =4 , получено = 4, потеряно = 0(0% потерь)
Приблизительное время передачи и приема:
    наименьшее = 0мс,  наибольшее= 0мс, среднее = 0мс
```

1.3.5 Утилита netstat

Выводит статистику протокола и текущих подключений сети TCP/IP. Эта команда доступна только после установки поддержки протокола TCP/IP. Синтаксис утилиты *netstat*:

netstat [*-a*] [*-e*] [*-n*] [*-s*] [*-p протокол*] [*-r*] [*интервал*],

где *-a* - выводит все подключения и сетевые порты. Подключения сервера обычно не выводятся;

-e - выводит статистику Ethernet. Возможна комбинация с ключом *–s*;

-n - выводит адреса и номера портов в шестнадцатеричном формате (а не имена);

-s - выводит статистику для каждого протокола. По умолчанию выводится статистика для TCP, UDP, ICMP (Internet Control Message Protocol) и IP. Ключ *-p* может быть использован для указания подмножества стандартных протоколов;

-p протокол - выводит соединения для протокола, заданного параметром. Параметр может иметь значения *tcp* или *udp*. Если используется с ключом *-s* для вывода статистики по отдельным протоколам, то параметр может принимать значения *tcp*, *udp*, *icmp* или *ip*;

-r - выводит таблицу маршрутизации;

интервал - обновляет выведенную статистику с заданным в секундах интервалом. Нажатие клавиш CTRL+B останавливает обновление статистики. Если этот параметр пропущен, *netstat* выводит сведения о текущей конфигурации один раз.

Пример использования *netstant*:

- без параметров:

C:\Program Files\Far\netstat

Активные подключения

Имя	Локальный адрес	Внешний адрес	Состояние
TCP	14423-5:1245	14423-3:netbios-ssn	ESTABLISHED

- с параметром *-a*:

C:\Program Files\Far>netstat –a

Активные подключения

Имя	Локальный адрес	Внешний адрес	Состояние
TCP	14423-5:epmap	14423-5:FIT.local:0	LISTENING
TCP	14423-5:microsoft-ds	14423-5:FIT.local:0	LISTENING
TCP	14423-5:1049	14423-5:FIT.local:0	LISTENING
TCP	14423-5:1072	14423-5:FIT.local:0	LISTENING
TCP	14423-5:netbios-ssn	14423-5:FIT.local:0	LISTENING
TCP	14423-5:427	14423-5:FIT.local:0	LISTENING
TCP	14423-5:1245	14423-3:netbios-ssn	TIME_WAIT
UDP	14423-5:epmap	*:*	
UDP	14423-5:microsoft-ds	*:*	
UDP	14423-5:1026	*:*	
UDP	14423-5:1043	*:*	
UDP	14423-5:1051	*:*	
UDP	14423-5:netbios-ssn	*:*	
UDP	14423-5:netbios-dgm	*:*	
UDP	14423-5: 427	*:*	
UDP	14423-5:isakmp	*:*	
UDP	14423-5:1048	*:*	

1.3.6 Утилита tracert

Диагностическая утилита, предназначенная для определения маршрута до точки назначения с помощью посылки эхо-пакетов протокола ICMP с различными значениями срока жизни (TTL, Time-To-Live). При этом требуется, чтобы каждый маршрутизатор на пути следования пакетов уменьшал эту величину по крайней мере на 1 перед дальнейшей пересылкой пакета. Это делает параметр TTL эффективным счетчиком числа ретрансляций. Предполагается, что когда параметр TTL становится равен 0, маршрутизатор посылает системе-источнику сообщение ICMP «Time Exceeded». Утилита *tracert* определяет маршрут путем посылки первого эхо-пакета с параметром TTL, равным 1, и с последующим увеличением этого параметра на единицу до тех пор, пока не будет получен ответ из точки назначения или не будет достигнуто максимальное допустимое значение TTL. Маршрут определяется проверкой сообщений ICMP «Time Exceeded», полученных от промежуточных маршрутизаторов. Однако некоторые маршрутизаторы сбрасывают пакеты с истекшим временем жизни без отправки соответствующего сообщения. Эти маршрутизаторы невидимы для утилиты *tracert*. Синтаксис утилиты *tracert*:

tracert [*-d*] [*-h макс_узл*] [*-j список_компьютеров*] [*-w интервал*] *точка_назн,*

где *-d* - отменяет разрешение имен компьютеров в их адреса;
-h макс_узл - задает максимальное количество ретрансляций, используемых при поиске точки назначения;
-j список_компьютеров - задает список_компьютеров для свободной маршрутизации;
-w интервал - задает интервал в миллисекундах, в течение которого будет ожидаться ответ;
точка_назн - указывает имя конечного компьютера.

Пример использования утилиты *tracert*:

```
T:\>tracert 14423-7
Трассировка маршрута к 14423-7.FIT.local [192.168.144.237]
С максимальным числом прыжков 30:
      1     <10мс <10мс <10мс 14423-7                [192.168.144.237]
Трассировка завершена
```

1.3.7 Утилита net use

Подключает общие сетевые ресурсы или выводит информацию о подключениях компьютера. Команда также управляет постоянными сетевыми соединениями. Синтаксис утилиты *net use*:

net use [*устройство* | *] [*компьютер**ресурс*[*\том*]] [*пароль* | *]] [*/user:*[*домен*\]*имя_пользователя*] [[*/delete*] | [*/persistent:{yes* | *no}*]]
net use устройство [*/home*[*пароль* | *]] [*/delete:{yes* | *no}*]
net use [*/persistent:{yes* | *no}*],

где *устройство* - задает имя ресурса при подключении/отключении. Существует два типа имен устройств: дисководы (от D: до Z:) и принтеры (от LPT1: до LPT3:). Ввод символа звездочки обеспечит подключение к следующему доступному имени устройства;
*компьютер**ресурс* - указывает имя сервера и общего ресурса. Если параметр компьютер содержит пробелы, все имя компьютера от двойной обратной черты (\\) до конца должно быть заключено в кавычки (" "). Имя компьютера может иметь длину от 1 до 15 символов;
\том - задает имя тома системы Novell NetWare. Для подключения к серверам Novell NetWare должна быть запущена служба клиента сети Novell NetWare (для Windows 2000 Professional) или служба шлюза сети Novell NetWare (для Windows 2000 Server);
пароль - задает пароль, необходимый для подключения к общему ресурсу;

* - выводит приглашение для ввода пароля. При вводе с клавиатуры символы пароля не выводятся на экран;
/user - задает другое имя пользователя для подключения к общему ресурсу;
домен - задает имя другого домена. Если домен не указан, используется текущий домен;
имя_пользователя - указывает имя пользователя для подключения;
/delete - отменяет указанное сетевое подключение. Если подключение задано с символом звездочки, будут отменены все сетевые подключения;
/home - подключает пользователя к его основному каталогу;
/persistent - управляет постоянными сетевыми подключениями. По умолчанию берется последнее использованное значение. Подключения без устройства не являются постоянными;
yes - сохраняет все существующие соединения и восстанавливает их при следующем подключении;
no - не сохраняет выполняемые и последующие подключения. Существующие подключения восстанавливаются при следующем входе в систему. Для удаления постоянных подключений используется ключ */delete*.
Вызванная без параметров утилита *net use* извлекает список сетевых подключений.

Пример использования *net use*:
C:\Program Files>net use t: \\fit-s1\install
Команда выполнена успешно

1.3.8 Утилита net send

Отправка сообщения другому пользователю, компьютеру или псевдониму в сети. Служба сообщений должна быть запущена на компьютере для получения сообщений. Синтаксис утилиты *net send*:

*net send {имя | * | /domain*[*:имя*] *| /users} сообщение,*

где *имя* - указывает имя пользователя, имя компьютера или псевдоним, которому будет отправлено сообщение. Если имя компьютера содержит пробелы, оно должно быть заключено в кавычки (" "). Длинные имена пользователей, введенные в формате NetBIOS, могут привести к возникновению исключительных ситуаций. Имена NetBIOS ограничены 16 символами, но Windows 2000 резервирует 16-ый символ;
* - отправляет сообщение всем членам группы;
/domain[*:имя*] - отправляет сообщение всем именам в домене компьютера. Если параметр имя указан, сообщение будет отправлено всем именам заданного домена или рабочей группы;
/users - отправляет сообщение всем пользователям, подключенным к серверу;

сообщение - указывает текст сообщения.

Пример использования *net send*:
T:\netsend 14423-8 проверка связи
Сообщение успешно отправлено 14423-8

1.4 Рекомендации и замечания

На основе рассмотренных сетевых утилит ОС Windows разрабатываются пользовательские приложения, реализующие мониторинг и диагностику локальных сетей. Они позволяют минимизировать усилия по поиску и исправлению ошибок в конфигурации сети и помогают системному администратору контролировать трафик. В настоящее время создано большое количество программ этого направления: Monitor It, Nautilus NetRanger, CiscoWorks 2000, ServiceSentinel и д.р. Они распространяются через Internet на условиях freeware. Windows NT Server обладает встроенными инструментами мониторинга: Event Viewer, Performance Monitor, Network Monitor.

1.5 Контрольные вопросы

1.5.1 Исправить синтаксис утилиты. C:\Program Files\Far\>net view all.
1.5.2 Для чего нужна утилита net send? Описать ее синтаксис.
1.5.3 Укажите неверный параметр C:\net use B:\\fit-s1\:install.
1.5.4 Можно ли утилитой tracert задать максимальное число ретрансляций?
1.5.5 Какой протокол необходим для работы с утилитой ping?
1.5.6 Какой результат выдаст утилита net stat с параметрами –a s –r?
1.5.7 Для чего необходима утилита hostname?
1.5.8 Зачем используется параметр all в утилите ipconfig?

2 Лабораторная работа №2. Обмен сообщениями на базе сетевых компонентов Delphi

2.1 Постановка задачи

Используя стандартные компоненты среды Delphi TClientSocket, TServerSocket, создать клиент-серверное приложение, реализующее обмен сообщениями между компьютерами, объединенными в локальную сеть, по протоколу TCP/IP в операционной системе Windows 9x/NT/2000.

В отчете по каждой лабораторной работе студентом представляются: название, постановка задачи, иерархическая схема процедур, текст программы, результаты работы со скриншотами программы, вывод.

2.2 Краткая теоретическая справка

Выполнение данной лабораторной работы основано на технологии сокетов (sockets). Сокет – это интерфейс прикладного программирования для сетевых приложений в операционной системе Unix. Позже интерфейс был переведен в Windows. Его основным назначением является предоставление возможности обмена данными по сети между различными узлами, используя протоколы. Существует три основных типа сокетов:

– *клиентские сокеты* инициализируются со стороны клиента. Для того, чтобы открыть соединение, клиентский сокет должен «знать» имя или IP-адрес сервера и номер порта, используемый серверным сокетом. Клиент посылает серверу запрос на соединение. Сервер ставит эти запросы в очередь и обслуживает их по мере поступления;

– *серверные сокеты* устанавливают соединение с клиентским сокетом в ответ на его запрос, полученный слушающим сокетом. При этом клиентский сокет получает описание серверного сокета, после чего соединение считается установленным;

– *слушающие сокеты* создаются сервером и принимают сообщения после запроса на соединение.

Процесс обмена данными между сокетами может происходить в двух режимах: в блокирующем и неблокирующем. При блокирующем режиме программа ждет выполнения какого-либо события. При неблокирующем – все действия выполняются параллельно.

Сокету для работы необходимо указать три параметра: IP-адрес, связанный с сокетом; номер порта, для которого будут выполняться операции обмена данными; протокол, по которому будет работать созданный сокет.

IP-адрес – это 32-битный адрес, используемый для идентификации узла в сети. Каждый узел сети должен иметь уникальный IP-адрес, состоящий из идентификаторов сети и обслуживающего компьютера. Этот адрес записывается в точечно-десятичном формате (например, 192.168.144.232).

Порты, используемые сокетами, являются программными и применяются в Windows на сетевом уровне. В компьютере десятки тысяч портов из них несколько сотен используются системой, остальные – как правило свободны и могут использоваться по желанию программистов. Они необходимы для обмена информацией между клиентом и сервером. Клиенту и серверу необходимо указать свободный порт для корректной работы. Данные на порт сервера могут приходить разными порциями от разных клиентов.

Протокол – это набор правил и соглашений для передачи данных по сети. Такие правила определяют формат, содержание, параметры времени, последовательность и проверку в сообщениях, которыми обмениваются сетевые устройства. Существует множество протоколов: TCP/IP (Transmission Control Protocol/Internet Protocol), UDP (User Datagramm Protocol), IPX/SPX (Internetwork Packet Exchange/Sequenced Packet Exchange) и т.д. Стек TCP/IP содержит набор сетевых протоколов Интернета, поддерживающих связь между объединенными сетями, состоящими из компьютеров различной архитектуры и разными операционными системами. Также он включает в себя стандарты для связи между локальными компьютерами, которым назначаются IP-адреса, и соглашения о соединении сетей и правилах маршрутизации сообщений. В состав стека TCP/IP входит не требующий соединений транспортный протокол UDP. Он является ненадежным, но широко используется в клиент-серверных запросах и приложениях, в которых важна скорость обмена данными, например при передаче информации в интерактивном режиме. В сетях Novell NetWare используется стек протоколов IPX/SPX, который управляет адресацией и направлением передачи пакетов. Его основным недостатком является невозможность объединения сетей с разными протоколами.

2.3 Разработка интерфейса

В качестве примера программы для обмена сообщениями в лабораторных работах №2 и №3 рассмотрим приложение, представляющее собой чат.

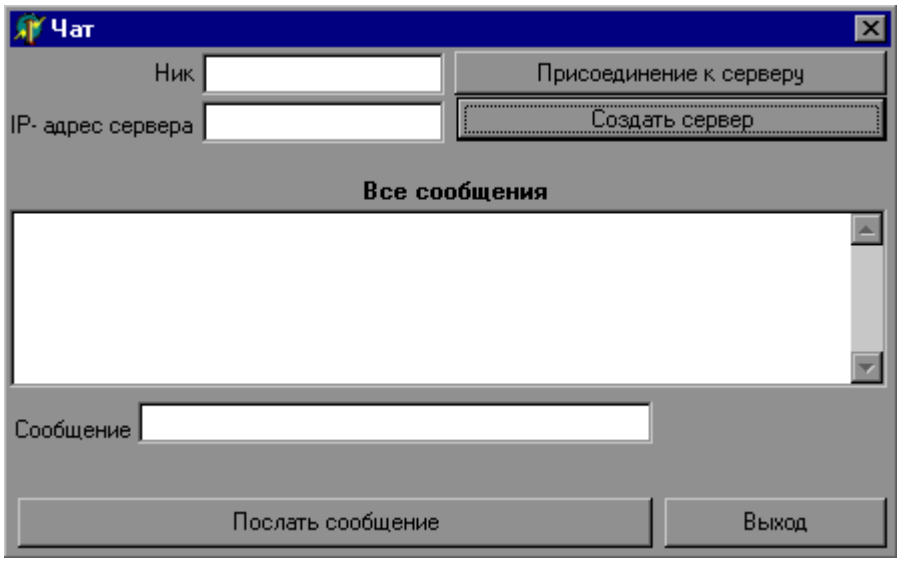

Рисунок 1 – Интерфейс программы «Чат»

Дополнительным требованием при его разработке является простой и удобный интерфейс пользователя, представленный на рисунке 1. Для реализации работы чата необходимо поместить на форму следующие интерфейсные компоненты: поля для ввода посылаемых сообщений и их отображения; строки для ввода имени пользователя и IP-адреса сервера; кнопки «Создать сервер», «Присоединение к серверу», «Послать сообщение» и «Выход».

На втором этапе происходит обработка событий на нажатие пользователем каких-либо кнопок. В нашем примере при нажатии на кнопку «Создать сервер» приложение будет проверять ввод имени. Если оно не введено, то выдается сообщение об ошибке и программа прекращает дальнейшую обработку события, иначе выполняются действия для создания сервера. При нажатии на кнопку «Присоединение к серверу» происходит проверка на ввод IP-адреса сервера и имени пользователя. Если проверка прошла удачно, то приложение использует функции и процедуры для подключения к серверу по заданному IP-адресу, иначе выдает соответствующее сообщение. Кнопкой «Послать сообщение», набранный текст отправляется серверу. При нажатии кнопки «Выход» приложение завершает работу.

2.4 Клиент-серверная модель передачи данных

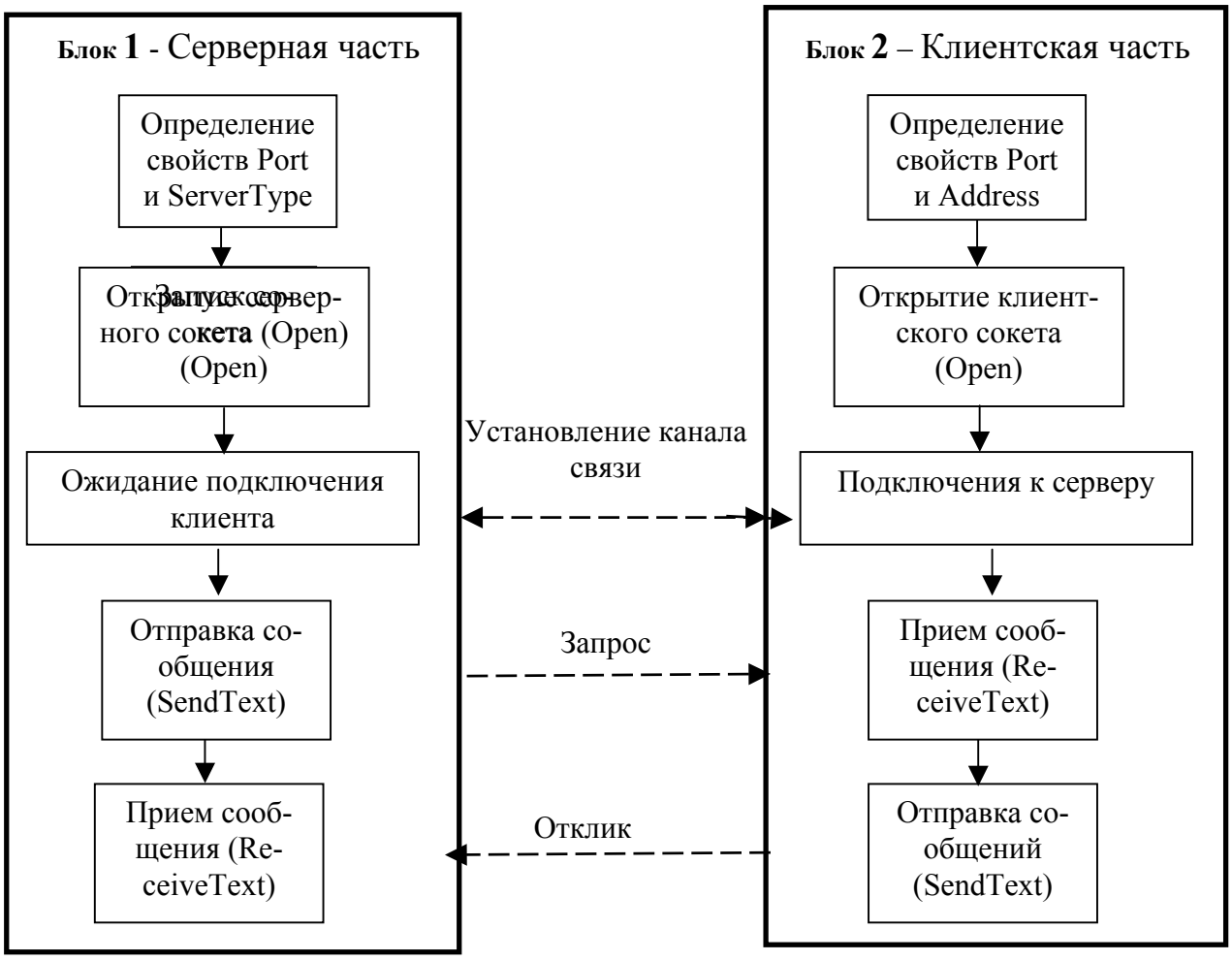

Рисунок 2 – Структура клиент-серверного приложения

Для реализации поставленной задачи необходимо создать клиентское и серверное приложение. Их структура на основе стандартных компонент TClientSocket и TServerSocket представлена на рисунке 2.

2.4.1 Реализация клиентской части

Для того, чтобы создать сетевое клиентское приложение, работающее по протоколу TCP/IP, необходимо использовать компонент TClientSocket. Каждый такой компонент работает с объектом TClientWinSocket и использует его события и методы. Основные этапы работы клиентского сокета заключаются в следующем: определение нужного сервера; установление связи с сервером; получение информации о соединении; обмен данными с сервером; закрытие соединения.

Для успешного открытия соединения, клиентскому сокету нужно знать имя или адрес сервера и номер порта. Имя задается в свойстве *Host*, а IP-адрес в свойстве *Address*. Если заданы оба значения, то предпочтение отдается имени компьютера. Для того, чтобы клиент-серверное приложение работало на локальной машине необходимо присвоить IP-адрес 127.0.0.1. Или свойству *Host* присвоить значение LocalHost. Адрес порта задается в свойстве *Port* и неявным образом в свойстве *Service*. Если заданы оба значения, используется имя службы. После задания этих параметров, нужно открыть соединение, вызвав метод *Open*. Чтобы соединение устанавливалось автоматически, при загрузке программы, необходимо в свойстве *Active* задать значение True. После соединения, свойство *Socket* объекта TClientWinSocket сервера хранит информацию о клиенте.

Соединение со стороны клиента завершается после вызова метода *Close*. Сервер также может закрыть соединение, в этом случае в клиентском сокете инициируется событие *OnDisconnect*.

2.4.2 Реализация серверной части

Сервер выполняет следующие задачи: слушает указанный порт; обрабатывает запросы на подключение; обменивается данными с клиентом; закрывает соединение.

Для создания сервера выбираем с панели Internet компонент TServerSocket и помещаем на форму приложения. В процессе работы будет использоваться один экземпляр объекта TServerWinSocket, необходимый для прослушивания порта и по одному экземпляру TServerClientWinSocket для каждого открытого соединения с сокетом клиента. Перед началом работы, серверному сокету необходимо задать номер порта, который он будет прослушивать. Номер задается в свойстве *Port* или неявным образом, при выборе службы в свойстве *Service.* Начать прослушивание порта во время исполнения программы можно, вызвав метод *Open*. Если в свойстве *Active* указать значение true, то при запуске программы слушающий сокет активизируется автоматически. Данные о слушающих и активных соединениях получаем из свойства *TServerSocket.Socket*.

Объекты, обслуживающие активные соединения, размещаются в свойстве *Connections*. Используя свойство *ActiveConnections*, в котором указывается общее число открытых соединений, можно обращаться к нужному клиенту в массиве *Connections* по индексу. Значение индекса изменяется от 0 до *ActiveConnections*–1.

Серверные соединения закрываются методом *Close*. При этом завершаются все открытые и слушающие соединения. Если клиент самостоятельно прекращает связь с сервером, то в серверном сокете произойдет событие *OnClientDisconnect*.

2.4.3 Передача и прием сообщений

Для решения поставленной задачи рекомендуется использовать неблокирующие сокеты. При этом операции приема/передачи сообщений будут происходить асинхронно и не останавливать выполнение программного кода. У клиентского сокета свойство *ClientType* должно иметь значение ctNonBlocking, а у серверного сокета свойству *ServerType* нужно присвоить значение stNonBlocking.

Для передачи сообщений служат функции *SendText* и *SendBuf*:

Функция *SendText* посылает текстовое сообщение *s* через сокет:

function SendText (**const** *s*: **string)**: **integer**,

где s - передаваемое сообщение.

Возвращаемый параметр – количество отосланных байт, в случае ошибки результат равен –1.

Функция *SendBuf* посылает буфер через сокет. Буфером может являться любой тип, например структура (**record**) или **integer**:

function SendBuf (**var** *buf*; *bufsize*: **integer**; *flags*: **integer**): **integer**,

где *buf* - передаваемые данные;

bufsize – размер данных, предназначенных для передачи;

flags – параметр передачи данных (рекомендуется приравнять к нулю).

Возвращаемое значение – количество отосланных байт, в случае ошибки результат равен –1.

Пример передачи данных[1)]:

Для клиентской части

```
Var  s:string;
Begin
      clientsocket1.Socket.SendText(s);
```

[1)] В разработке примеров принимал участие Идгеев К. М.

```
End;
Для серверной части
Var  s:string;
     i:integer;
Begin
     for i:=0 to serversocket1.socket.activeConnections -1 do
          serversocket1.socket.connections[i].SendText (s);
End;
```

Для получения сообщений используются функции *ReceiveText* и *ReceiveBuf*:

Функция *ReceiveText* получает сообщения в виде текстовой строки и вызывается без параметров:

function ReceiveText: **string**.

Эта функция возвращает полученную строку.

Функция *ReceiveBuf* получает сообщения через буфер. Как и у функции *SendBuf*, буфером функции *ReceiveBuf* может являться любой тип:

function ReceiveBuf(**var** *buf*; *bufsize*: **integer**; *flags*: **integer**): **integer**,

где *buf* – полученные данные;
bufsize – размер принятых данных;
flags – параметр передачи данных (рекомендуется принять равным 0).
Возвращаемое значение – количество полученных байт, в случае ошибки результат равен –1.

Узнать размер полученных данных можно, вызвав следующую функцию:

function ReceiveLength: **integer**.

Возвращаемое значение – количество полученных байт.

Когда на прослушиваемый сокетом порт приходит информация, переданная удаленным компьютером, вызывается событие *OnClientRead*, поэтому функции, предназначенные для получения данных, необходимо располагать в обработчике этого события.

Пример приема данных:

```
Для клиентской части
procedure TForm1.ClientSocket1Read(Sender: TObject;
  Socket: TCustomWinSocket);
Var s:string;
Begin
```

```
        s:=socket.Receivetext;
        //Вывод текста на экран
End;
Для серверной части
procedure TForm1.ServerSocket1ClientRead(Sender: TObject;
  Socket: TCustomWinSocket);
Var s:string;
Begin
        s:=socket.receivetext;
        //Пересылка данных остальным клиентам
End;
```

2.5 Рекомендации и замечания

Методов организации работы сокетов достаточно, например, файловые потоки (TFileStream) или потоки в памяти (TMemoryStream). Потоки – это канал для обмена данными, работа с которым аналогична работе с обычным файлом. Поэтому этот механизм удобно использовать при пересылке файлов больших размеров. При этом файл разбивается и пересылается по частям, контролировать правильность передачи не нужно.

Кроме компонент TClientSocket и TServerSocket для создания клиент-серверного приложения можно в качестве альтернативного варианта использовать стандартные компоненты с панели FastNet (TNMMsg и TNMMsgServ), IndyClients (ItTCPClient), IndyServers (ItTCPServer). Две последние компоненты поставляются с Delphi 6.

2.6 Контрольные вопросы

2.6.1 Что такое сокет, и какие существуют типы сокетов?
2.6.2 Для чего необходим IP-адрес?
2.6.3 Какой главный недостаток протокола IPX/SPX?
2.6.4 Описать структуру клиент-серверного приложения?
2.6.5 Какой компонент используется для создания клиента?
2.6.6 Какой протокол может использоваться для обмена данными без подтверждения о приеме?
2.6.7 Проверить правильность присвоения значения свойству объекта:
ServerSocket1.Address := ‘14423-2’;
2.6.8 Какое свойство серверного сокета получает данные об активных и слушающих соединениях?
2.6.9 Что означает следующая запись: ClientSocket1.ClientType := ctBlocking?

3 Лабораторная работа №3. Передача сообщений на базе библиотеки WinSock

3.1 Постановка задачи

Реализовать клиент – серверный программный продукт в среде Delphi, для обмена текстовыми сообщениями в рамках протокола TCP/IP под ОС Windows 9x/NT/2000 на базе функций библиотеки WinSock.

3.2 Краткая теоретическая справка

WinSock или Windows socket - это интерфейс прикладного программирования (API), созданный для реализации приложений в сети на основе протокола TCP/IP. Для работы используется WSOCK32.DLL. Эта библиотека находится в папке \System32 системного каталога Windows.

Существуют две версии WinSock:

WinSock 1.1 - поддерживает только протокол TCP/IP;

WinSock 2.0 - поддерживает дополнительное программное обеспечение.

WinSock 1.1 дал толчок к развитию World Wide Web и позволил получить доступ в Internet обычному пользователю ПК под Windows. Если цель версии 1.1 состояла в решении проблемы, то цель WinSock 2.0 - сделать сетевую среду лучше, быстрее и надежнее. В WinSock 2.0 добавлена поддержка других транспортных протоколов и новые функциональные возможности обеспечения надежности сетевого обмена информацией. WinSock 2.0 позволяет создавать независящие от транспортных протоколов приложения, работающие с TCP/IP (Transmission Control Protocol/Internet Protocol), UDP (User Datagram Protocol), IPX/SPX (Internetwork Packet Exchange/Sequenced Packet Exchange), NetBEUI (NetBios Extended User Interface). Большая эффективность таких приложений достигается за счет совмещенного ввода/вывода и разделяемых сокетов.

Спецификация WinSock разделяет функции на три типа:

- блокирующие и неблокирующие (функции Беркли);
- информационные (получение информации о наименовании доменов, службах, протоколах Internet);
- инициализации и деинициализации библиотеки.

Блокирующая – это функция, которая останавливает работу программы до своего завершения; неблокирующая – это функция, которая выполняется параллельно с программой. Список основных функций, необходимых для создания приложения, приведен в таблицах 1, 2, 3. Все описания функций WinSock даны в формате языка С, а примеры их вызова – на Delphi.

Таблица 1 – Блокирующие функции (функции Беркли)

Название функции	Назначение
1	2
Accept	Создает новый сокет и подключает его к удаленному компьютеру
Closesocket	Закрывает одну из сторон соединения

Продолжение таблицы 1

1	2
Connect	Инициализирует соединение со стороны указанного сокета
Recv	Принимает данные от подключенного сокета
Recfrom	Принимает данные от подключенного или неподключенного сокета
Send	Посылает данные подключенному сокету
Sendto	Посылает данные подключенному или неподключенному сокету

Таблица 2 – Неблокирующие функции (функции Беркли)

Название функции	Назначение
Bind	Связывает виртуальный сокет с физическим
Inet_addr	Конвертирует строку в значение, которое можно использовать в структуре in_addr:
Ioctlsocket	Управляет параметрами сокета
Listen	Переводит сокет в режим прослушивания порта.
Socket	Создает точку соединения

Таблица 3 – Функции инициализации и деинициализации библиотеки WinSock

Название функции	Назначение
WSACleanup	Прекращает работу с WinSock DLL
WSAGetLastError	Получает информацию о последней ошибке
WSASetLastError	Устанавливает возврат после ошибки
WSAStartup	Инициализирует WinSock DLL

3.3 Схема взаимодействия функций WinSock

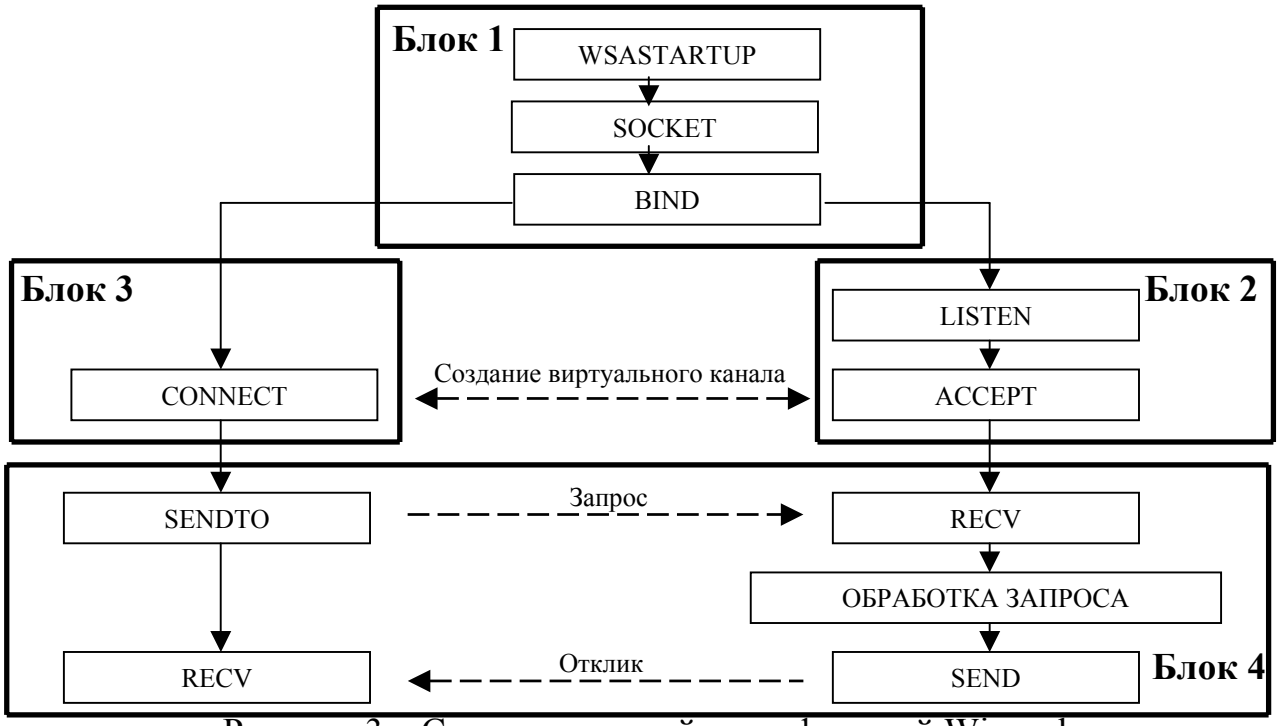

Рисунок 3 – Схема взаимодействия функций Winsock

Для реализации поставленной задачи необходимо создать клиентское и серверное приложение. Они различны по организации, но есть общие действия,

необходимые как для клиентской, так и для серверной части. Схема взаимодействия функций WinSock отражена на рисунке 3. В блоке 1 рисунка отображены общие действия сервера и клиента, в блоках 2 и 3, соответственно, действия сервера и клиента, а в блоке 4 – их взаимодействие.

Для того, чтобы использовать функции WinSock, необходимо загрузить библиотеку WSock32.dll. Как видно из блока 1 рисунка 3, это осуществляется функцей *WSAStartUp* [см. 3.7.1]. При удачной загрузке библиотеки, нужно создать сокет, используя функцию *Socket* [см. 3.7.2] и ассоциировать сокет с адресной структурой *SockAddr_In* [см. 3.7.3], которая содержит информацию о протоколе соединения, IP-адрес и порт ПК.

3.4 Реализация клиентской части

После создания клиентского сокета, он посылает запрос на подключение к серверу, используя функцию *connect* [см. 3.7.5], указав в качестве одного из параметров IP-адрес сервера. Эта функция является блокирующей, т.е. выполнение программы приостановится до тех пор, пока не придет ответ от сервера. При положительном ответе сокет подключается к серверу и может с ним взаимодействовать. Реализация клиентской части представлена в блоке 3 рисунка 1.

3.5 Реализация серверной части

После общих действий, в соответствии с блоком 2 рисунка 1, сервер прослушивает порт функцией *listen* [см. 3.7.4], т.е. проверяет его на предмет запроса от клиента. После поступления запроса, сервер обрабатывает его функцией *accept* [см. 3.7.4], т.е. сервер присоединяет клиента, создавая новый сокет. Созданный сокет выступает в качестве посредника между клиентом и сервером, т.е. “общение” происходит через новый сокет и наоборот.

3.6 Реализация обмена данными

В блоке 4 рисунка 1 отражен процесс взаимодействия между клиентом и сервером, который сводится к отправке и приема сообщений. Для отправки сообщения используется функция *send* или sendto [см. 3.7.6]. Их отличие состоит в том, что для функции *send* необходимо соединение (*connect*, *accept*), для *sendto* оно необязательно. Для приема сообщений применяют функции *recv* или *recvfrom* [см. 3.7.6]. Для функции *recvfrom* соединение (*connect*, *accept*) также необязательно.

3.7 Библиотека WinSock и ее функции

3.7.1 Инициализация WinSock

Функция *WSAStartup* инициализирует библиотеку WinSock. Она всегда стоит первой при начале работы с WinSock. Приведем ее описание:

int WSAStartup (**WORD** *wVersionRequested*, **LPWSADATA** *lpWSAData*).

Первый параметр - это версия, которая будет использоваться. Младший байт основная версия, старший байт расширение версии. Если инициализация состоялась, то вернется нулевое значение. Инициализация заключается в сопоставлении номера версии и реально существующей DLL в системе.

Второй параметр - это указатель на структуру WSADATA, в которую возвратятся параметры инициализации:

```
typedef struct WSAData {
        WORD            wVersion;
        WORD            wHighVersion;
        char            szDescription[WSADESCRIPTION_LEN+1];
        char            szSystemStatus[WSASYS_STATUS_LEN+1];
        unsigned short  iMaxSockets;
        unsigned short  iMaxUdpDg;
        char FAR *      lpVendorInfo;
} WSADATA, FAR * LPWSADATA;
```

WSACleanup завершает использование данного DLL и прерывает обращение к функциям WinSock. При удачном выполнении вернется нуль.

Пример инициализации библиотеки WinSock:

```
Var
    MyWSAData : WSADATA;
    ErrWSADATA : integer;
Begin
ErrWSAData := WSAStartUp(MakeWord(2, 0), MyWSAData);
If (ErrWSAData <> 0) then
    ShowMessage('Библиотека не была инициализирована')
Else
// Ваши дальнейшие действия
End;
```

3.7.2 Создание и удаление сокета

Функция создания сокета имеет следующий вид:

SOCKET socket(**int** *af*, **int** *type*, **int** *protocol*),

где *af* - характеризует набор протоколов, в рамках которого будут взаимодействовать клиент и сервер (это может быть TCP/IP, UDP, IPX и т.д.). Для протокола TCP/IP параметр *af* должен быть равен AF_INET, что соответствует формату адреса, принятому в Internet.

type - определяет тип коммуникаций (SOCK_STREAM, и SOCK_DGRAM). Если данный параметр равен SOCK_STREAM, то со-

кет будет использован для передачи данных через канал связи с использованием протокола TCP/IP. Если же используется SOCK_DGRAM, то передача данных будет выполняться без создания каналов связи через датаграммный протокол UDP,

protocol - задает код конкретного протокола из указанного набора (заданного *af*), который будет реализован в данном соединении. Протоколы обозначаются символьными константами с префиксом IPPROTO_ (например, IPPROTO_IP или IPPROTO_UDP). Допускается значение protocol=0 (протокол не указан), в этом случае используется значение по умолчанию для данного вида соединений.

Возвращаемый параметр представляет собой дескриптор соединителя.

Если операция *socket* завершилась успешно, выходной параметр равен дескриптору соединителя, в противном случае - INVALID_SOCKET (-1). С помощью оператора *WSAGetLastError* можно получить код ошибки, проясняющий причину отрицательного результата.

Уничтожает сокет функция:

closesocket(**SOCKET** *s*),

где *s* - переменная типа TSocket, полученная в результате вызова функции *Socket.*

Пример создания сокета:

```
Var
    MySokcet : Tsocket;
begin
    MySocket := socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
            If (MySocket = INVALID_SOCKET) then
        ShowMessage('Сокет не создан')
    Else
    // Ваши дальнейшие действия
End.
```

3.7.3 Привязка адреса к сокету

Функция связывания сокета с физическим адресом имеет вид:

Int bind(**SOCKET** *s*, **const struct socketaddr FAR*** *name*, **int** *namelen*),

где *s* - целочисленный код дескриптора;

name - содержит три величины: IP-адрес, код протокольного набора, номер порта, который определяет характер приложения;

namlen - определяет длину второго параметра.

Структура адресной информации имеет вид:

```
struct sockaddr_in{
short sin_family; {Указывается протокол}
```

unsigned short *sin_port;* {Указывается порт}
struct in_addr *sin_addr;* {Указывается IP-адрес}
char *sin_zero[8]*};

В серверной части приложения IP-адрес можно сделать равным INADDR_ANY (или =0), т.к. серверу не обязательно знать свой IP-адрес. При корректном выполнении функция *bind* возвращает код 0, в противном случае SOCKET_ERROR=-1.

Пример:

```
Var
    MySockAddr : TsockAddr_In;
    ErrBind : integer;
begin
    MySockAddr.sin_family := AF_INET;
    MySockAddr.sin_port := htons(1024);
    MySockAddr.sin_addr.s_addr := INADDR_ANY;
    ErrBind := bind(MySock, MySockAddr, SizeOf(MySockAddr));
                        If (ErrBind <> 0) then
          ShowMessage('Ошибка связывания адреса с сокетом')
    Else   // Ваши дальнейшие действия
end.
```

3.7.4 Ожидание и обработка запросов на подключение клиента

Для того, чтобы сокет ожидал подключения, его необходимо перевести в ожидающее состояние при помощи функции *listen*:

int listen(**SOCKET** *s*, **int** *backlog*),

где *backlog* - задает максимальный размер очереди для приходящих запросов соединения, т.е. сколько запросов может быть принято на обслуживание без потерь. Ожидающий сокет посылает каждому отправителю сообщение-отклик, подтверждающее получение запроса на соединение.

Запросы из очереди, сформированной функцией *listen*, обрабатываются функцией *accept*, устанавливающей связь с сокетом клиента:

int accept(**SOCKET** *s*, **struct sockaddr FAR*** *addr*, **int FAR*** *addrlen*),

где *s* - дескриптор соединителя, который прослушивает соединение (тот же, что и в *listen*);
addr - указатель на структуру, которая содержит адрес;
addrlen – указатель на длину адреса *addr*.
При возникновении ошибки возвращается код INVALID_SOCKET.

Приведем пример использования функций *linsten* и *accept*:

```
Var
    AcceptSocket : Tsocket;
    AcceptAddr : ^SockAddr_In;
    SizeAcceptAddr : ^Integer;
    ErrListen : integer;
Begin
    ErrListen := listen(MySock, 5);
    If (ErrListen <> 0) then ShowMessage('Очередь для зпаросов не сформированна')
    Else Begin
AcceptSocket := accept(MySock, @AcceptAddr, @SizeAcceptAddr);
If (AcceptSocket = INVALID_SOCKET) then
    ShowMessage('Клиент не присоединился')
else
// Ваши дальнейшие действия
End;
End.
```

3.7.5 Подключение к серверу

Клиент для соединения с сервером должен использовать функцию *connect*:

int connect(**SOCKET** *s*, **const struct socketaddr FAR*** *name,* **int** *namelen*),

где *s* - дескриптор сокета;

name - идентификатор адреса места назначения (указатель на структуру данных);

namelen - длина этого адреса.

Таким образом, функция *connect* сообщает IP-адрес и номер порта удаленной машине. Если адресное поле структуры *name* содержит нули, функция *connect* вернет ошибку WSAEADDRNOTAVAIL (или SOCKET_ERROR=-1).

Попытка подключения к серверу описана в следующем примере:

```
Var
    ClientSocket : TSocket;
    ServerAddr : SockAddr_In;
    ErrConnect : integer;
begin
    ServerAddr.sin_family := AF_INET;
    ServerAddr.sin_port := htons(1024);
    ServerAddr.sin_addr.s_addr := inet_addr(<IP-адрес сервера>);
    ErrConnect := connect(ClientSocket, ServerAddr, SizeOf(ServerAddr));
    If (ErrConnect <> 0) then
```

```
            ShowMessage('Не могу подключиться к серверу')
        Else
            // Ваши дальнейшие действия
    end.
```

3.7.6 Отправка и прием сообщений

Для отправки сообщений используется функция *send* или *sendto*:

int send (**SOCKET** *s*, **const char FAR*** *buf*, **int** *len*, **int** *flags*),

где *s* – дескриптор сокета на удаленной машине;
buf – указатель на массив символов, подлежащих пересылке;
len – размер второго параметра;
flgas - служит для целей диагностики и управления передачей данных. Рекомендуется приравнивать его нулю.

Пример рассылки сообщений оператором *send*:

```
                                    Var
        Buf : array [0..255] of char;
        ErrSend : integer;
    Begin
        ErrSend := send(AcceptSocket, Buf, SizeOf(Buf), 0);
                        If (ErrSend = SOCKET_ERROR) then
            ShowMessage('Ошибка передачи данных')
                                        Else
            // Ваши дальнейшие действия
    End.
```

Функция *sendto* служит для пересылки данных без установки соединения, то есть отправка данных идет без подтверждения получения данных:

int sendto (**SOCKET** *s*, **const char FAR*** *buf*, **int** *len*, **int** *flags*, **const struct sockaddr FAR*** *to*, **int** *tolen*);

где *s* – дескриптор отправителя;
buf – указатель на массив данных, предназначенных для пересылке;
len – размер второго параметра;
flags - служит для целей диагностики и управления передачей данных;
to – адресная структура сокета удаленной машины;
tolen – размер адресной структуры сокета удаленной машины.

Пример посылки сообщений функцией *sendto*:

```
Var
    Buf : array [0..255] of char;
```

```
        MySocket : TSocket;
        SendToAddr : SockAddr_In;
        ErrSendTo : ineteger;
    Begin
        ErrSenTo := sendto(MySocket, Buf, SizeOf(Buf), 0, SendToAddr,
SizeOf(SendToAddr));
        If (ErrSendTo = SOCKET_ERROR) then
            ShowMessage('Ошибка передачи данных')
                                    Else
            // Ваши дальнейшие действия
    End.
```

Для приема сообщения применяется функция *recv* или *recvfrom*. Функцию *recvfrom* так же, как и *sendto*, можно применять, не соединяясь с отправителем.

int recv (**SOCKET** *s*, **char FAR*** *buf*, **int** *len*, **int** *flags*);

где s – дескриптор сокета получателя;
buf – указатель на массив полученных данных;
len – размер второго параметра;
flags - служит для целей диагностики и управления передачей данных.

Пример приема сообщений функцией *recv*:

```
Var
    Buf : array [0..255] of char;
    MySock : TSocket;
    ErrRecv : integer;
Begin
    ErrRecv := recv(MySock, buf, SizeOf(Buf), 0);
    If (ErrRecv = SOCKET_ERROR) then
        ShowMessage('Не могу принять сообщение')
                                Else
        // Ваши дальнейшие действия
End.
```

int recvfrom (**SOCKET** *s*, **const char FAR*** *buf*, **int** *len*, **int** *flags*, **const struct sockaddr FAR*** *to*, **int** *tolen*);

где *s* – дескриптор получателя;
buf – указатель на массив полученных данных;
len – размер второго параметра;
flags - служит для целей диагностики и управления передачей данных;
to – адресная структура сокета удаленной машины;
tolen – размер адресной структуры сокета удаленной машины.

Пример прием сообщений, используя функцию *recvfrom*:

```
Var
Buf : array [0..255] of char;
MySocket : TSocket;
RecvFromAddr : SockAddr_In;
ErrRecvFrom : ineteger;
Begin
ErrRecvFrom := recvfrom(MySocket, Buf, SizeOf(Buf), 0, RecvFro-
mAddr, SizeOf(RecvFromAddr));
If (ErrRecvFrom = SOCKET_ERROR) then
ShowMessage('Не могу принять сообщение')
Else
// Ваши дальнейшие действия
End.
```

3.8 Рекомендации по выполнению работы

Приведем несколько советов для упрощения создания приложения. Функции *accept*, *connect*, *recv* являются блокирующими, т.е. выполнение программы задерживается на некоторое время, в течение которого нельзя выполнять никаких действий. Вы будете пользоваться таким чатом, когда во время ожидания сообщения программа зависает? Рекомендуем все блокирующие функции вывести в отдельные потоки (TThread), работающие параллельно с программой и не приводящие к зависанию программы. В клиентской части можно вынести в отдельный поток прослушивание порта на предмет получения данных от сервера и обработку получаемых данных. В серверной части, создать несколько потоков, содержащих в себе функцию *accept*; при получении запроса на подключение, создается еще один поток, выполняющий постоянное прослушивание порта на предмет передачи данных вновь подключенного клиента. Таким образом, в серверной части имеем n+1 поток, где n-количество клиентов.

3.9 Контрольные вопросы

3.9.1 Поддерживала ли WinSock 1.1 протокол UDP?
3.9.2 Верно ли утверждение, что функция accept является неблокирующей?
3.9.3 Что выполняет функция bind?
3.9.4 Исправьте ошибку: Socket(MySock, IPPROTO_IP, SOCK_STREAM); .
3.9.5 Опишите общие действия сервера и клиента.
3.9.6 Что означает следующая запись: Connect(Socket1, Addr1, SizeOf(Addr1));?
3.9.7 Что необходимо сделать, чтобы создать серверный сокет?
3.9.8 Для чего используются потоки?

3.10 Заключение

До разработки библиотеки WinSock сокетная технология не была доступна для широкого круга пользователей. Она использовалась только в институтах, правительственных и военных учреждениях. Поэтому при разработке библиотеки учитывались интересы и желания конечного пользователя. В результате получился мощный инструмент для сетевого программирования. С его помощью можно писать приложения, независящие от используемого протокола. Благодаря функциям библиотеки программирование ведется не на машинном языке, а на прикладном уровне. Помимо всего прочего, с помощью сокетов можно писать собственные сетевые протоколы. Несмотря на кажущуюся легкость и простоту программирования на базе сокетов, иногда возникают ситуации, когда происходит потеря пакетов. Выявить и исправить причину позволяют сетевые утилиты ОС Windows. Также с помощью сетевых утилит возможна пересылка текстовых сообщений, управление общими ресурсами, диагностика сетевых подключений.

На базе приведенных сведений возможно решение таких задач, как разработка многофункционального чата, игровых программ для сети, организация связи между программами, работающими на разных станциях в сети без обращения к файл-серверу и т.д. Например:

- написать чат, используя API-функции;
- написать сетевую игру;
- реализовать программный продукт, имитирующий работу сотовой станции;
- написать программу, выполняющую мониторинг сети;
- сканирование сети и определение топологии;
- написать компонент для обмена данными по одному из протоколов TCP/IP, UDP, IPX/SPX и т.д.;
- реализовать систему удаленного администрирования рабочих станций;
- интернет–пейджер.

Список использованных источников

1 Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы. – СПб.: Питер, 2002. – 672 с.: ил.

2 Сетевые средства Microsoft Windows NT Server 4.0; Перевод с англ. СПб.: - BHV – Санкт-Петербург, 1997. – 527с.: ил.

3 Козлов А.В. Программирование для Интернет в Delphi 5. – М.: ЗАО «Издательство БИНОМ», 2001. – 368 с.: - ил.

4 Фролов А.В, Фролов Г.В. Глобальные сети компьютеров. Практическое введение в Internet, E-Mail, FTP, WWW и HTML, программирование для Windows Sockets, Том 23. - М.: Диалог-МИФИ, 1996. - 283 с.: - ил.

5 Джонс Э., Оланд Дж. Программирование в сетях Microsoft Windows. Мастер-класс: Пер. с англ. – СПб: Питер. Издательско-торговый дом «Русская редакция», 2002. – 608 с.: – ил.

6 Снейдер И. Эффективное программирование TCP/IP. Библиотека программиста – СПб: Питер, 2002 – 320 с.:ил.

7 www.citforum.ru/nets/tpns/glava_4.shtml

8 www.citforum.ru /nets/protocols2/2_09_03.shtml

9 www.citmgu.ru/nets/index.html

10 www.lan.ru/tcpip/main.html