

**Санкт-Петербургское государственное бюджетное профессиональное  
образовательное учреждение  
«Колледж информационных технологий»**

**ОТЧЕТ**  
**по самостоятельной работе**  
**по МДК «Прикладное программирование»**  
**для специальности 230115**  
**«Программирование в компьютерных**  
**системах»**

**Выполнил:**

**студент группы №12**  
**Снятков В.С.**

**Проверила**

**преподаватель СПб ГБПОУ**  
**Матысик Ирина Алексеевна**

**Санкт-Петербург, 2014 год**

# **СОДЕРЖАНИЕ**

## Введение

## 1. Теоретическая ЧАСТЬ

1.1. Практика сетевого программирования

1.2. Сетевые компоненты ServerSocket и ClientSocket

1.3. Понятие и назначение объекта Socket

1.4. Последовательность действий при разработке сетевого приложения

1.5. Автоопределение IP-Адреса локальной машины

1.6. Передача данных между клиентом и сервером

1.7. Передача списка пользователей

## 2. Практическая часть

2. 1. Постановка задачи

2.2. Алгоритм решения задачи

2.3. Инструкция пользователя

## ЗАКЛЮЧЕНИЕ

## Список используемой литературы

## ПРИЛОЖЕНИЕ А

## ПРИЛОЖЕНИЕ Б

## **ВВЕДЕНИЕ**

В последнее время всё чаще при разработке прикладных программ повышенное внимание уделяется на наличие в конечной программе сетевых инструментов. И это неудивительно, учитывая популярность сетевых приложений: ведь многие из вас наверняка сидели в чатах (программа для общения по сети), играли в сетевые игры (такие, как Counter-Strike, Quake и прочие). Именно поэтому данная тема самостоятельной работы будет интересна и актуальна.

Практика сетевого программирования также успешно развивается и в нашем колледже. В прошлом учебном году студенты нашего колледжа защищали несколько дипломных работ, которые были связаны с сетевым программированием.

В данной самостоятельной работе будет рассмотрено сетевое программирование в среде C++ Builder 6, хотя некоторые представленные примеры можно использовать в более поздних версиях C++ Builder, включая C++ Builder 2010.

Требуется самостоятельно освоить теоретический материал по тематике выданной работы, а так же разработать проект, основанный на сетевой технологии разработки приложений. Возможно, данный материал самостоятельной работы будет востребован при выполнении дипломной работы, как часть выданного задания.

# 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## 1.1. Практика сетевого программирования

Программирование сетей в среде C++ Builder может осуществляться несколькими способами:

### *Способ 1. Программирование с использованием библиотеки winsock.h*

Сетевое программирование на основе Windows Sockets основано на WinAPI, что несколько затрудняет понимание программного кода, но обеспечивает надёжность результата. Данная библиотека не зависит от версии C++ Builder. Но для того, чтобы написать приложение на winsock.h, нужно ещё иметь представление о создании потоков при помощи WinAPI, поэтому данный метод не рассматривается в самостоятельной работе.

### *Способ 2. Программирование с использованием компонентов Indy*

Данный метод, несмотря на частоту его использования при сетевом программировании, обладает одним очень серьёзным минусом, который может стать преградой при переходе от одной версии C++ Builder к другой. С каждой новой версией C++ Builder выпускается новая версия компонентов Indy. Таким образом, происходит прямая зависимость от версии данной библиотеки, что весьма неудобно, поэтому данный метод не рассматривается в самостоятельной работе.

### *Способ 3. Программирование с использованием компонентов ServerSocket и ClientSocket*

Данный способ очень схож с первым способом в плане архитектуры программирования – в обоих случаях используется Windows Sockets. Отличием данного способа от указанных выше является простота использования при разработке, т.к. данные компоненты являются адаптацией winsock.h. Именно данный способ будет рассмотрен в самостоятельной работе.

## 1.2. Сетевые компоненты ServerSocket и ClientSocket

### Компонент ClientSocket



Данный компонент находится на вкладке Internet и выполняет функции клиентского сокета.

### Рисунок 1 – Компонент ClientSocket



Рисунок 2 – Расположение компонента в Палитре компонентов

Свойства:

**String Host** - имя удалённого сервера.

**String** Address - IP-адрес удалённого сервера.

Для указания сервера можно задать значение только одного из этих свойств. Если же заданы оба значения, то предпочтение отдаётся имени.

**unsigned int Port** - порт, используемый сервером (может принимать значение от 1 до 65534).

**TClientType** ClientType - свойство, определяющее режим чтения/записи у клиента. Принимает только одно из двух значений:

- `ctNonBlocking` - указывает, что используется асинхронный режим чтения/записи;
- `ctBlocking` - указывает, что используется синхронный режим чтения/записи

В первом случае приложение не будет ждать окончания операции и продолжит свою работу. Во втором же случае выполнение приложения приостановится до завершения соответствующей операции.

**bool Active** - определяет, является ли соединение активным.

Методы:

**void** Open() - устанавливает соединение с сервером. При удачном выполнении метода свойство **Active** переходит в **true**.

**void Close()** - разрывает соединение с сервером. При удачном выполнении метода свойство **Active** переходит в **false**.

### События:

**OnConnecting(TObject \*Sender, TCustomWinSocket \*Socket)** - событие наступает в тот момент, когда сокет сервера найден, но соединение ещё не установлено.

**OnConnect(TObject \*Sender, TCustomWinSocket \*Socket)** - событие наступает после установления соединения с сервером.

**OnDisconnect(TObject \*Sender, TCustomWinSocket \*Socket)** - событие наступает в момент разрыва соединения с сервером.

**OnRead(TObject \*Sender, TCustomWinSocket \*Socket)** - событие наступает в момент получения клиентом сообщения от сервера.

### Компонент ServerSocket



Данный компонент также находится на вкладке Internet и реализует одновременно функции слушающего и серверного сокетов.

Рисунок 3 – Компонент ServerSocket



Рисунок 4 – Расположение компонента в Палитре компонентов

### Свойства:

**unsigned int Port** - порт, используемый для создания сервера.

**TServerType ServerType** - режим чтения/записи на сервере. Может принимать одно из двух значений:

- **stNonBlocking** - асинхронный режим чтения/записи;
- **stThreadBlocking** - блокирующий (синхронный) режим чтения/записи.

В первом случае сервер будет работать в обычном режиме «очереди», т.е. обрабатывать запросы по мере их поступления. Во втором же случае для каждого клиента должен создаваться отдельный поток, в котором и будет обслуживаться клиент. Это имеет свои минусы.

**int ThreadCacheSize** - число кэшируемых потоков. Данное свойство определяет максимальное число обслуживаемых потоков в случае, когда сервер работает в блокирующем режиме. Это означает, что в случае превышения заданного в этом свойстве числа новые клиенты не будут обслуживаться до тех пор, пока не освободится хотя бы один поток, а это весьма неудобно.

**bool Active** - определяет, является ли сервер активным.

**TCustomWinSocket\* Socket->Connections** - индексированный массив активных подключений к серверу. Каждый элемент этого массива содержит в себе информацию о клиентском сокете.

**int Socket->ActiveConnections** - переменная, хранящая в себе количество активных подключений к серверу.

Два последних свойства обычно используются в тех случаях, когда серверу необходимо разослать всем клиентам сети.

```
for (int i = 0; i < ServerSocket1->Socket->ActiveConnections; i++)  
{  
    ServerSocket->Socket->Connections[i]->SendText("Hello, world");  
}
```

Методы:

**void Open()** - создаёт сервер. При удачном выполнении метода свойство **Active** переходит в **true**.

**void Close()** - разрушает сервер. При удачном выполнении метода свойство **Active** переходит в **false**.

События:

**OnClientRead(TObject \*Sender, TCustomWinSocket \*Socket)** - событие наступает в момент получения сервером от клиента какого-либо сообщения.

**OnClientConnect(TObject \*Sender, TCustomWinSocket \*Socket)** - событие наступает в момент подключения клиента к серверу.

**OnClientDisconnect(TObject \*Sender, TCustomWinSocket \*Socket)** - событие наступает в момент отключения клиента от сервера.

### 1.3. Понятие и назначение объекта Socket

Итак, сокеты – это некие коммутаторы, обслуживающие соединения в сети. Проще говоря, сокеты хранят информацию о подключениях в сети, и с помощью данной информации осуществляют передачу данных по сети от одного узла к другому.

По своему назначению сокеты разделяются на три типа:

#### 1. Клиентские сокеты

*Клиентские сокеты иницируют соединение, указывая DNS/IP удалённого сервера и порт, используемый сервером.*

#### 2. Серверные сокеты

*Серверные сокеты берут запрос из очереди запросов и непосредственно устанавливают соединение с клиентским сокетом.*

Таким образом, клиентский сокет в итоге получает описание серверного сокета, с которым он установил связь.

Если соединение установлено, то в работу вступает объект **Socket**. Он является объектом класса **TCustomWinSocket** и доступно только для чтения. Применяется обычно при обработке каких-либо событий, происходящих в сети (например, подключение клиента).

Рассмотрим свойства и методы объекта Socket.

#### Свойства:

**String** RemoteAddress - IP-адрес удалённой машины.

**String** RemoteHost - имя удалённой машины (DNS).

**unsigned int** RemotePort - порт сокета удалённого подключения.

**String** LocalAddress - IP-адрес локальной машины.

**String** LocalHost - имя локальной машины (DNS).

**unsigned int** LocalPort - порт сокета локального подключения.

#### Основные методы:

**int \_\_fastcall** SendText(**AnsiString** S) - данный метод отправляет сокету получателя строку. В случае удачного выполнения функции возвращает 0.

**int \_\_fastcall SendBuf(void\* Buf, int Count)** - данный метод посылает сокету получателя то, что находится в буфере **Buf**. Вторым параметром является количество байт посылаемой информации.

**AnsiString \_\_fastcall ReceiveText()** - данный метод позволяет считать полученную от сокета отправителя строку. Возвращает строку типа **AnsiString**.

**int \_\_fastcall ReceiveBuf(void\* Buf, int Count)** - данный метод позволяет считать N байт из посланного отправителем буфера данных. Возвращает **-1**, если ни один байт не прочитан. Если методу удалось считать хотя бы один байт, то функция вернёт количество считанных байт информации.

Компонент **ServerSocket** может использовать вышеуказанные свойства и методы, например:

```
ServerSocket->Socket->Connections[0]->SendText("Hello, world");
```

Именно таким образом сервер отправит первому подключившемуся к нему клиенту сообщение «Hello, world!».

#### 1.4. Последовательность действий при разработке сетевого приложения

При разработке сетевого приложения важно чётко понимать последовательность действий, совершаемых при каком-либо действии, обрабатываемом обеими сторонами (и сервером, и клиентом). Рассмотрим последовательность действий в некоторых ситуациях.

Таблица 1 – Основные группы событий компонентов

| Событие клиента                      | Событие сервера        |
|--------------------------------------|------------------------|
| <i>Подключение клиента к серверу</i> |                        |
| Open()                               |                        |
| OnConnecting                         | OnClientWrite          |
| OnConnect                            | OnClientConnect        |
| <i>Отключение клиента от сервера</i> |                        |
| Close()                              |                        |
| OnDisconnect                         | OnClientDisconnect     |
| <i>Передача данных серверу</i>       |                        |
| SendText, SendBuf, ...               | OnClientRead           |
| <i>Приём данных от сервера</i>       |                        |
| OnRead                               | SendText, SendBuf, ... |

Конечно, можно рассмотреть и большее количество случаев, но здесь были рассмотрены только те основные ситуации, которые пригодятся нам при разработке сетевого приложения.

### 1.5. Автоопределение IP-Адреса локальной машины

Подключив библиотеку `winsock.h`, мы можем воспользоваться одним из приёмов программирования – автоопределением IP локального компьютера. Это удобно в ситуации, когда при запуске серверного приложения необходимо прописывать IP-адрес создаваемого сервера.

Но прежде чем рассмотреть этот приём, нужно пояснить некоторые моменты Windows Sockets.

`WORD wVersionRequested;`

Версия библиотеки `winsock.h`. Обычно её задают таким образом:

```
wVersionRequested = MAKEWORD(1, 0);
```

Это означает, что используется первая версия библиотеки.

`WSADATA wsaData;`

Структурная переменная, хранящая в себе данные об устройстве, с помощью которого будет создаваться подключение.

```
int err = WSAStartup(wVersionRequested, &wsaData);
```

Функция запуска устройства. Входными параметрами является версия библиотеки и переменная, в которую будут записываться все параметры устройства. При удачном запуске возвращает 0.

`hostent *h;`

Переменная структурного типа `HOSTENT`.

Сама структура выглядит так:

```
typedef struct hostent {  
    char FAR    *h_name;  
    char FAR    FAR **h_aliases;  
    short       h_addrtype;  
    short       h_length;  
    char FAR    FAR **h_addr_list;  
} HOSTENT;
```

**h\_name** - Имя компьютера (DNS).

**h\_aliases** – Массив альтернативных имён компьютера.

**h\_addrtype** – Тип возвращаемого IP-адреса (IPv4, IPv6).

**h\_length** – Длина адреса в байтах.

**h\_addr\_list** – Массив адресов локального компьютера. Если указать **h\_addr** в качестве параметра, то он возвращает первый адрес из списка (**h\_addr\_list[0]**).

Данный массив является двумерным, поэтому для получения самого IP-адреса нужно будет обращаться к массиву по двум индексам ([порядковый номер адреса][порядковый номер цифры IP-адреса]), например:

```
sprintf(LocalIp, "%d.%d.%d.%d", (unsigned char)h->h_addr_list[0][0],  
(unsigned char)h->h_addr_list[0][1], (unsigned char)h->h_addr_list[0][2],  
(unsigned char)h->h_addr_list[0][3]);
```

Листинг автоматического получения IP-адреса с комментариями:

```
// Начинаем процесс получения IP-адреса машины через сокеты  
WORD wVersionRequested;  
WSADATA wsaData;  
wVersionRequested = MAKEWORD(1, 0);  
int err = WSStartup(wVersionRequested, &wsaData);  
if (err == 0) {  
    // 1. Создаём буфер (в данном случае - 128 символов)  
    char Buf[128];  
    // 2. Для получения имени машины и записи её в буфер пользуемся  
    // функцией gethostname(char* <строка>, int <длина строки>)  
    gethostname(&Buf[0], 128);  
    // 3. Далее создаём переменную структурного типа hostent  
    hostent *h;  
    // 4. Теперь, зная имя машины, пытаемся получить её IP-адрес  
    // Используем функцию gethostbyname(const char * FAR <имя>)  
    h = gethostbyname(&Buf[0]);  
    // 5. Проверяем, вернула ли нам функция структуру.  
    if (h != NULL) {  
        char *LocalIp = new char[15];  
        sprintf(LocalIp, "%d.%d.%d.%d", (unsigned char)h->h_addr_list[0][0],  
            (unsigned char)h->h_addr_list[0][1], (unsigned char)h->h_addr_list[0][2],  
            (unsigned char)h->h_addr_list[0][3]);  
        // Далее можно указать действие по отображению IP в Edit,  
        например:
```

```
edAddress->Text = LocalIp;  
}  
}
```

Вместо этого программного кода можно было узнать IP с помощью командной строки. Выполнить cmd – запустить командную строку. Ввести команду ipconfig.

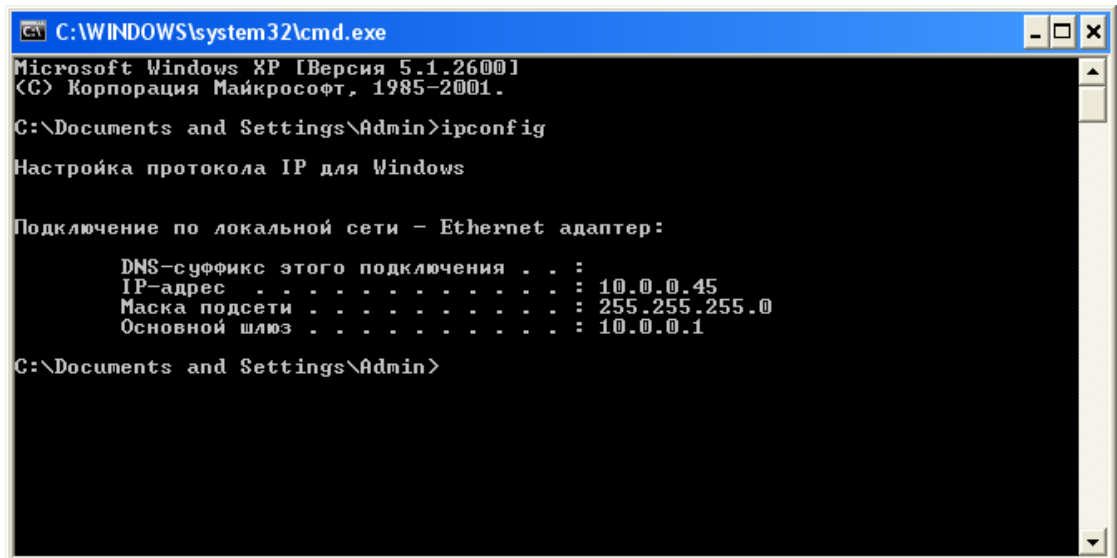


Рисунок 5 – Окно с результатами выполнения командной строки

## 1.6. Передача данных между клиентом и сервером

**Client:**

```
ClientSocket->Socket->SendText(Edit1->Text);
```

**Server:**

*OnClientRead:*

```
for (int i=0; i<ServerSocket1->Socket-> ActiveConnections ;i++)
```

```
ServerSocket->Socket->Connections[i]->SendText(Socket->ReceiveText());
```

**Client:**

*OnRead:*

```
Memo1->Text=Socket->ReceiveText();
```

## 1.7. Передача списка пользователей

//Объявление глобальных переменных

```
AnsiString Nickname; //Имя пользователя (только вошедшего)
```

```
TStringList *Name_List = new TStringList(); //Здесь будут храниться  
имена подключившихся пользователей
```

```

//Считывание сервером сообщение от клиента (Socket)

void __fastcall TForm1::ServerSocket1ClientRead(TObject *Sender,
        TCustomWinSocket *Socket)
{
    AnsiString S; //В эту переменную будет записано сообщение

    S = Socket->ReceiveText(); //в переменную записывается пришедшее
сообщение

    if (S[1] == '#')        //проверяется если первый символ равен "#" то
выполнить

    {    // Регистрация на сервере

        Nickname = S.SubString(2, S.Length()); //в переменную "Nickname"
записываем имя вошедшего пользователя

        Name_List->Add(Nickname); //добавляем в список игроков имя этого
пользователя

        for (int i=0; i<ServerSocket1->Socket->ActiveConnections; i++)

            //посылаем всем подключенным пользователям список пользователей

            ServerSocket1->Socket->Connections[i]->SendText("#" +
Name_List->Text);

        }

    }

    //Убрать из списка отключившего клиента

    void __fastcall TForm1::ServerSocket1ClientDisconnect(TObject *Sender,
        TCustomWinSocket *Socket)
    {
        int i;

        //проходим по всем подключенным пользователям

        for (i=0; i < ServerSocket1->Socket->ActiveConnections; i++)
        {

```

```

        if (ServerSocket1->Socket->Connections[i] == Socket) break;
    }
    Name_List->Delete(i); //удаляется ушедший пользователь из списка
пользователей
    for (i=0; i < ServerSocket1->Socket->ActiveConnections; i++)
    { if (ServerSocket1->Socket->Connections[i] != Socket)
        //посылаем всем подключенным пользователям список пользователей
        ServerSocket1->Socket->Connections[i]->SendText("#" +
Name_List->Text);
    }
}

```

## 2. ПРАКТИЧЕСКАЯ ЧАСТЬ

### 2. 1. Постановка задачи

Требуется разработать приложение «Клиент-Сервер» для обмена сообщениями в чате, обладающее следующими функциональными характеристиками: первая часть – серверная – должна обеспечить подключение пользователей к сети. Вторая часть – клиентская – должна отображать список подключенных клиентов в сети, а так же обеспечивать передачу текстовых сообщений между клиентами сети.

Входные данные сервера: IP-адрес локальной машины, на которой запущено приложение, номер порта. Выходные данные сервера: количество пользователей, подключенных к чату. Входные данные клиента: IP-адрес сервера, номер порта, имя пользователя. Выходные данные клиента: список пользователей чата, сообщения, передаваемые в чате.

#### Серверное приложение

Внешний вид приложения может выглядеть примерно так:



Рисунок 6 – Интерфейс серверного приложения

## Клиентское приложение

Клиентское приложение будет отличаться от сервера набором функций, ведь клиенту нужно не только связываться с клиентом, но и передавать сообщения и/или файлы другим пользователям.

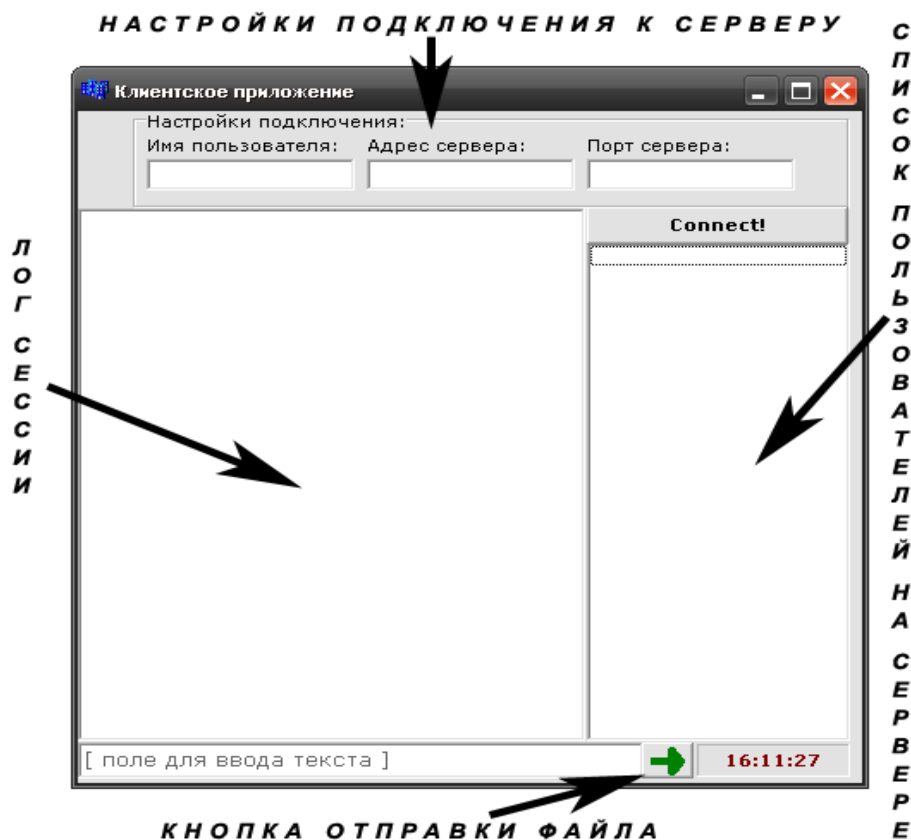


Рисунок 7 – Интерфейс клиентского приложения

## 2.2. Алгоритм решения задачи

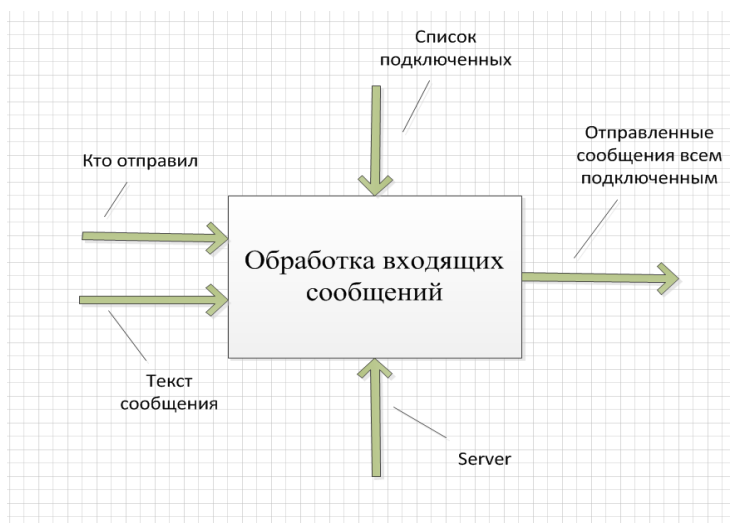


Рисунок 8 -Схема работы приложения Сервер

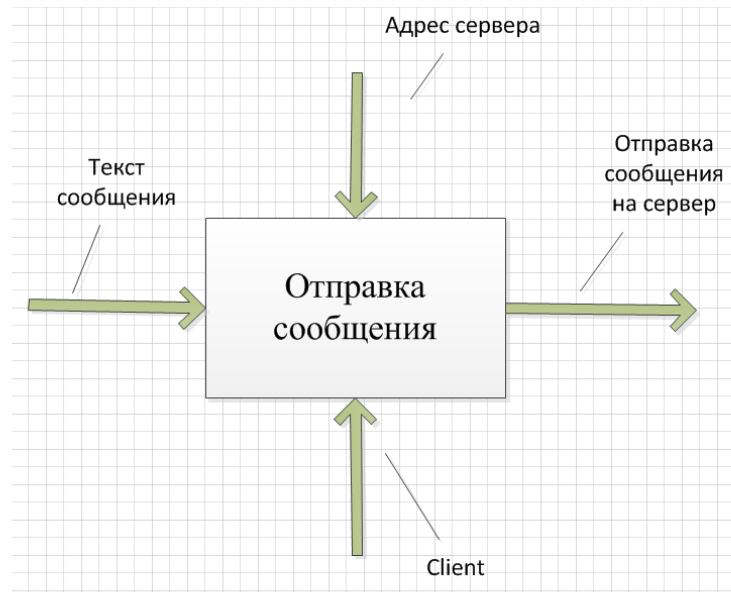


Рисунок 9 - Схема работы приложения Клиент.

### 2.3. Инструкция пользователя

1. Запустит приложение Server.exe.
2. В появившемся окне нажмите на кнопку «Старт!» (Рисунок 10).

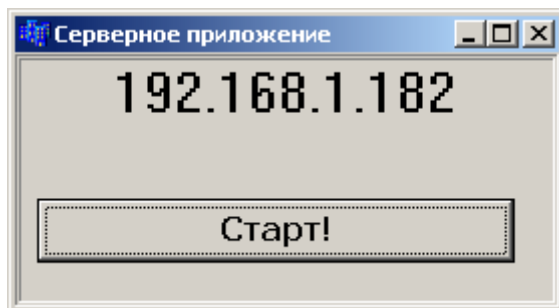


Рисунок 10 - Приложение – Сервер

Сверните окно и не закрывайте приложение, пока работа в сети не будет закончена.

3. Запустите приложение Client.exe.
4. В верхней части появившегося окна заполните поля «Имя клиента» (Ваш никнейм) и «Адрес сервера». IP-адрес сервера, можно узнать из приложения server.exe (Рисунок 10).

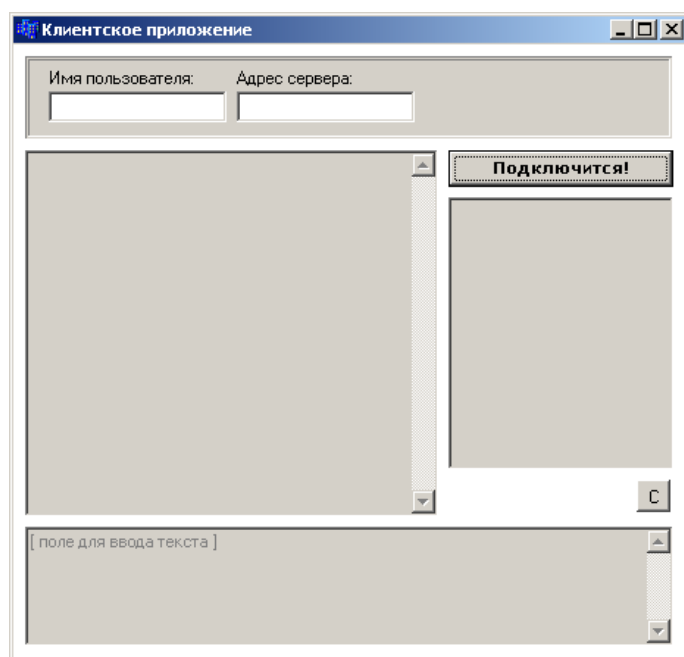


Рисунок 11 - Приложение – Клиент

5. Нажмите на кнопку «Подключиться!».

6. После этого в логе чата Вас известят о состоянии подключения и, в случае успеха, в списке пользователей отобразятся все, кто находятся в данной сети (Рисунок 12).

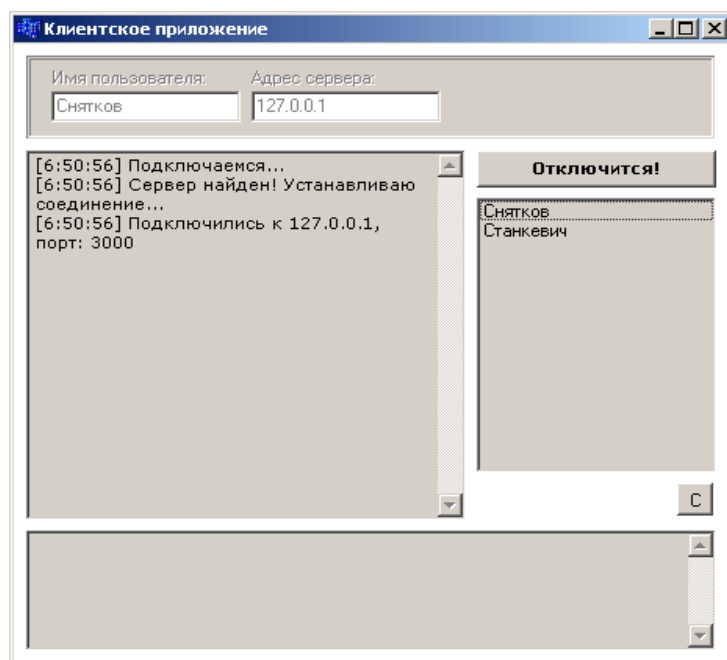


Рисунок 12 - Соединение

7. Для того чтобы отправить сообщение, введите его в текстовое поле внизу окна и нажмите на клавиатуре на клавишу «Enter» (Рисунок 13).

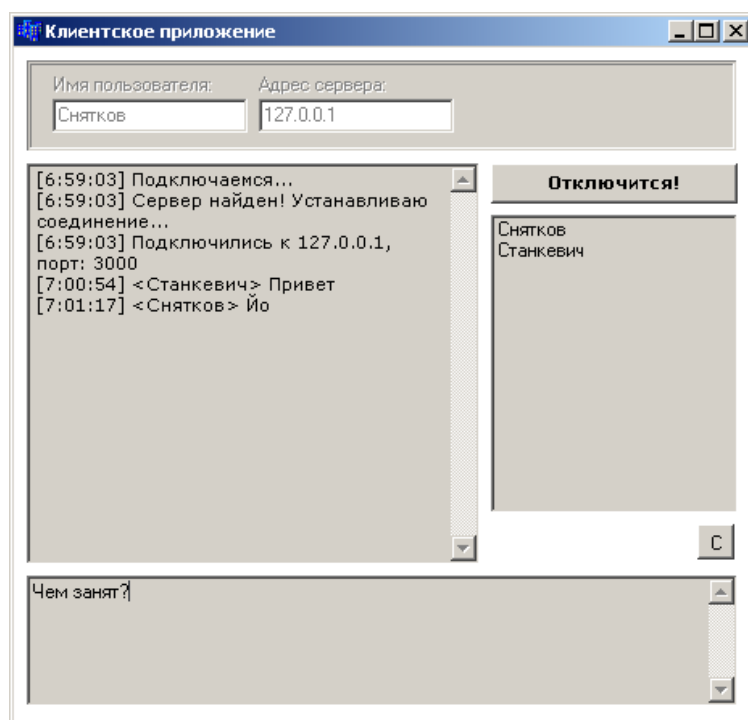


Рисунок 13 - Отправка сообщения

После этого в логе чата отобразится Ваше сообщение всем участникам сети (Рисунок 14).

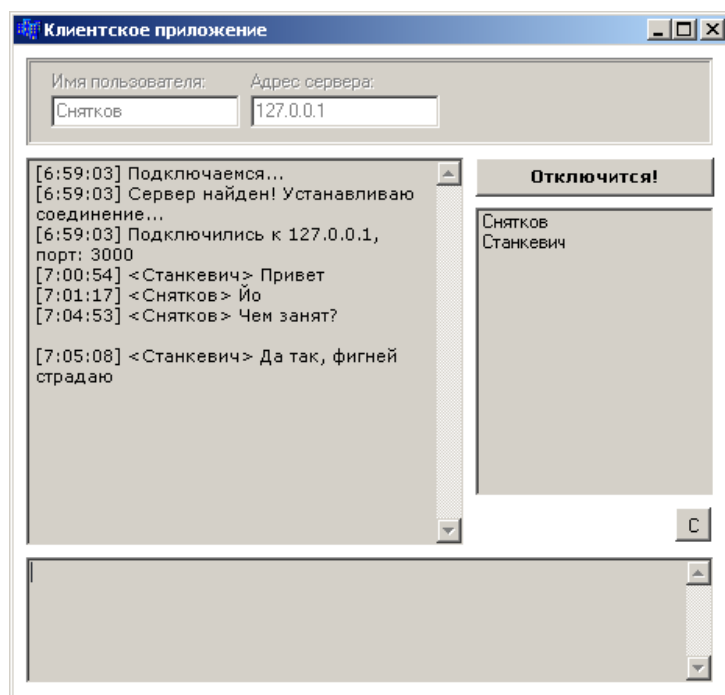


Рисунок 14 - Результат отправки сообщения

8. Чтобы отправить приватное сообщение, сначала в списке пользователей выберите того, кому хотите отправить это сообщение, затем повторите шаг 7.

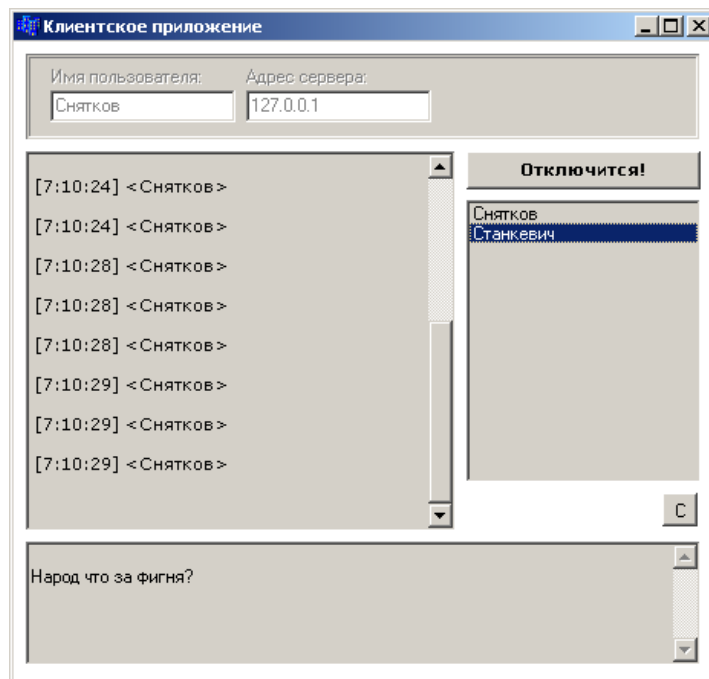


Рисунок 15 - Отправка приватного сообщения

В логе чата такие сообщения отображаются по-особому.

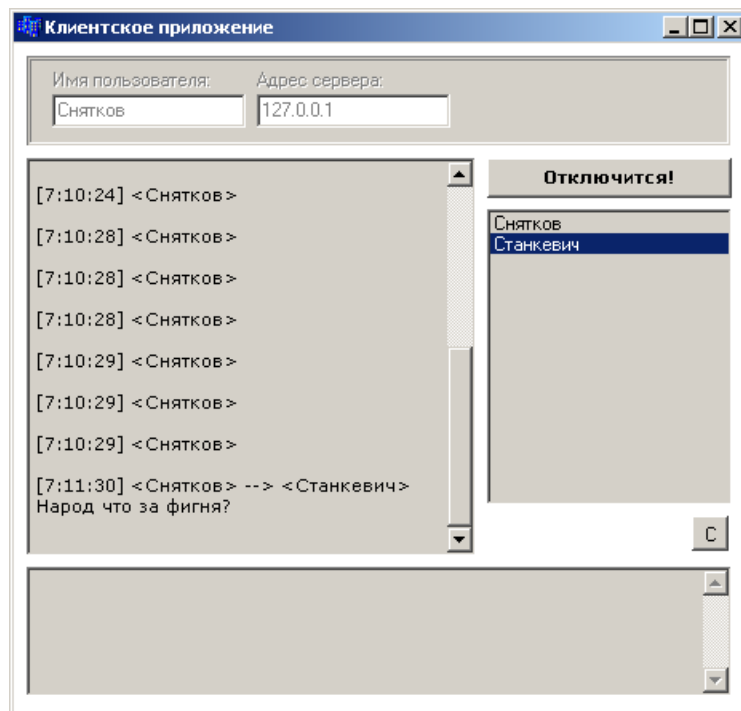


Рисунок 16 - Результат отправки приватного сообщения

## ЗАКЛЮЧЕНИЕ

В ходе выполнения самостоятельной работы была освоена тема «Разработка приложения с использованием сетевых технологий». В наше время очень развиты Компьютерные сети и поэтому сетевые приложения есть и будет актуальными в использовании. Разработанное приложение «Чат» можно использовать как средство общения нескольких пользователей между собой, соединенных с помощью локальной сети.

В процессе разработки приложения были самостоятельно изучены такие компоненты визуальной среды программирования C++ Builder как ServerSocket и ClientSocket, освоено взаимодействие компьютеров в локальной сети и принципы работы протокола TCP/IP.

Также в ходе выполнения самостоятельной работы были закреплены знания и умения использования языка C++ и среды C++ Builder.

В результате разработано функциональное сетевое приложение «Чат» для работы в локальной сети несколькими пользователями. Пользователи могут обмениваться общими и приватными сообщениями, наблюдать структуру сети и получать системную информацию о компьютерах в локальной сети.

## СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. *Аверкин В.П., Бобровский А.И.* Программирование на C++. Учебное пособие. Корона-Принт. 1999.
2. *Аверкин В.П.* Современное проектирование на C++. Обобщенное программирование и прикладные шаблоны проектирования. – М.: Вильямс, 2002.
3. *Аммерааль Л.* STL для программистов на C++. – М.: ДМК, 1999.
4. *Астахова И.Ф., Власов С.В.* Язык C++. Учебное пособие. 2003.
5. *Бондарев В.М.* Программирование на C++. 2005.
6. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++. 2001.
7. *Вайнер, Пинсон.* C++ Изнутри. 2000.
8. *Вандевурд, Джосаттис.* Шаблоны C++. Справочник разработчика. – М.: Вильямс. 2003.
9. *Глушаков, Коваль, Смирнов.* Язык программирования C++, учебный курс. 2001.
10. *Голуб А.И.* Правила программирования на С и C++. 2001.

## ПРИЛОЖЕНИЕ А

### *Листинг программного кода серверного приложения*

```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include "stdio.h"
#include "winsock.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
TDateTime Time_Working;
AnsiString nickname;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
    // Начинаем процесс получения IP-адреса машины через сокет
    WORD wVersionRequested;
    WSADATA wsaData;
    wVersionRequested = MAKEWORD(1, 0);
    int err = WSAStartup(wVersionRequested, &wsaData);
    if (err == 0) {
        // 1. Создаём буфер (в данном случае - 128 символов)
        char Buf[128];
        // 2. Для получения имени машины и записи её в буфер пользуемся
        // функцией gethostname(char* <строка>, int <длина строки>)
        gethostname(&Buf[0], 128);
        // 3. Далее создаём переменную структурного типа hostent
        hostent *h;
        // 4. Теперь, зная имя машины, пытаемся получить её IP-адрес
        // Используем функцию gethostbyname(const char * FAR <имя>)
        h = gethostbyname(&Buf[0]);
        // 5. Проверяем, вернула ли нам функция структуру.
        if (h != NULL) {
            char *LocalIp = new char[15];
            sprintf(LocalIp, "%d.%d.%d.%d", (unsigned char)h->h_addr_list[0][0],
                (unsigned char)h->h_addr_list[0][1], (unsigned char)h->h_addr_list[0][2],
                (unsigned char)h->h_addr_list[0][3]);
            // Выводим IP адрес
            edAddress->Text = LocalIp;
        }
    }
}
```

```

    }
}
pTime->Caption = TimeToStr(Time());
// Обнуляем значение поля edPort
edPort->Text = "";
// Задаём Caption кнопке
if (btnSwitch->Tag == 0) btnSwitch->Caption = "Start!";
// Очищаем Мемо
mLog->Clear();
// Здесь мы задаём название приложения, отображаемое на панели
задач
Application->Title = "Сервер";
}
//-----
void __fastcall TForm1::tmrTimeTimer(TObject *Sender)
{
    pTime->Caption = TimeToStr(Time());
}
//-----
void __fastcall TForm1::btnSwitchClick(TObject *Sender)
{
    switch (btnSwitch->Tag)
    {
        case 0: {
            ServerSocket1->Port = StrToInt(edPort->Text);
            ServerSocket1->Open();
            if (ServerSocket1->Active)
            {
                mLog->Lines->Add "[" + TimeToStr(Time()) + "] Сервер создан!";
                edAddress->Enabled = false;
                edPort->Enabled = false;
                btnSwitch->Tag = 1;
                btnSwitch->Caption = "Shutdown!";
                Time_Working = 0;
                tmrWorking->Enabled = true;
            }
        }
        else
        {
            btnSwitch->Tag = 0;
            btnSwitch->Caption = "Start!";
            mLog->Lines->Add "[" + TimeToStr(Time()) + "] Сервер не удалось
создать!");
            edAddress->Enabled = true;
            edPort->Enabled = true;

```

```

    }
    }
    break;
    case 1:
    {
        ServerSocket1->Close();
        if (!ServerSocket1->Active)
        {
            mLog->Lines->Add "[" + TimeToStr(Time()) + "] Сервер разрушен!");
            edAddress->Enabled = true;
            edPort->Enabled = true;
            btnSwitch->Tag = 0;
            btnSwitch->Caption = "Start!";
            tmrWorking->Enabled = false;
            Form1->Caption = "Серверное приложение";
            Application->Title = "Сервер";
            lbUsers->Clear();
        }
        else
        {
            btnSwitch->Tag = 1;
            btnSwitch->Caption = "Shutdown!";
            mLog->Lines->Add "[" + TimeToStr(Time()) + "] Сервер не удалось
разрушить!");
            edAddress->Enabled = false;
            edPort->Enabled = false;
        }
    }
    break;
}
}
//-----
void __fastcall TForm1::tmrWorkingTimer(TObject *Sender)
{
    Time_Working += 1. / 86400;
    Form1->Caption = "Сервер работает: " + TimeToStr(Time_Working) + " ::
Пользователей: " + IntToStr(lbUsers->Items->Count);
    Application->Title = "В сети: " + TimeToStr(Time_Working);
}
//-----
void __fastcall TForm1::ServerSocket1ClientRead(TObject *Sender,
    TCustomWinSocket *Socket)
{
    AnsiString S;
    Sleep(500);

```

```

S = Socket->ReceiveText();
nickname = "";
if (S[1] == '#')
{
    // Регистрация на сервере
    nickname = S.SubString(2, S.Length());
    lbUsers->Items->Add(nickname);
    for (int i=0; i<ServerSocket1->Socket->ActiveConnections; i++)
        ServerSocket1->Socket->Connections[i]->SendText("#" +
lbUsers->Items->Text);
    if (nickname != "") Form1->mLog->Lines->Add "[" + TimeToStr(Time())
+ "]" Пришёл " + nickname + " (" + Socket->RemoteAddress + ")");
}
else
{
    // Получение сообщения на сервер и последующая отправка нужному
юзеру (или же всем)
    AnsiString SName, RName, Msg;
    SName = S.SubString(1, S.Pos("#")-1);
    int spos, fpos;
    spos = S.Pos("#")+1;
    RName = S.SubString(spos, S.Length());
    RName = RName.SubString(1, RName.Pos("~")-1);
    Msg = S.SubString(S.Pos("~")+1, S.Length());
    if (SName == "All")
    {
        if (Msg[1] == '%')
        {
            TStream* SF;
            int size;
            AnsiString tmp, tmp2;
            tmp = Msg.SubString(2, Msg.Pos("&")-2);
            tmp2 = Msg.SubString(Msg.Pos("&")+1, Msg.Length());
            size = StrToInt(tmp);
            AnsiString filename;
            filename = tmp2;
            SF = new TFileStream(filename, fmCreate, fmShareDenyNone);
            int received = Socket->ReceiveBuf(&SF, size);
            if (received > 0)
            {
                mLog->Lines->Add "[" + TimeToStr(Time()) + "]" <" + RName + ">
Передаётся файл test.txt");
                for (int i=0; i<ServerSocket1->Socket->ActiveConnections; i++)
                    ServerSocket1->Socket->Connections[i]->SendBuf(SF, size);
            }
        }
    }
}

```

```

        SF->Free();
    }
}
else
{
    for (int i=0; i<ServerSocket1->Socket->ActiveConnections; i++)
        ServerSocket1->Socket->Connections[i]->SendText(RName + "~" +
Msg);
    mLog->Lines->Add "[" + TimeToStr(Time()) + "] <" + RName + "> " +
Msg);
}
}
else
{
    if (Msg[1] == '%')
    {
        TStream* SF;
        int size = StrToInt(Socket->ReceiveText());
        SF = new TFileStream("test2.txt", fmCreate, fmShareDenyNone);
        int received = Socket->ReceiveBuf(&SF, size);
        if (received > 0)
        {
            mLog->Lines->Add "[" + TimeToStr(Time()) + "] <" + RName + ">
Передаётся файл test.txt");
            for (int i=0; i<ServerSocket1->Socket->ActiveConnections; i++)
                ServerSocket1->Socket->Connections[i]->SendBuf(SF, size);
        }
    }
}
else

ServerSocket1->Socket->Connections[lbUsers->Items->IndexOf(SName)]->Send
Text(RName + "~" + Msg);
}
}
}
//-----
void __fastcall TForm1::ServerSocket1ClientDisconnect(TObject *Sender,
TCustomWinSocket *Socket)
{
    int i;
    for (i=0; i < ServerSocket1->Socket->ActiveConnections; i++)
    {
        if (ServerSocket1->Socket->Connections[i] == Socket) break;
    }
    lbUsers->Items->Delete(i);

```

```

        for (i=0; i < ServerSocket1->Socket->ActiveConnections; i++)
        {
            if (ServerSocket1->Socket->Connections[i] != Socket)
                ServerSocket1->Socket->Connections[i]->SendText("#" +
lbUsers->Items->Text);
        }
    }
}

```

## ПРИЛОЖЕНИЕ Б

### *Листинг программного кода клиентского приложения*

```

#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
    mLog->Clear();
    pTime->Caption = TimeToStr(Time());
    if (btnConnect->Tag == 0)
    {
        btnConnect->Caption = "Connect!";
        Edit1->Enabled = false;
    }
}
//-----
void __fastcall TForm1::tmrTimeTimer(TObject *Sender)
{
    pTime->Caption = TimeToStr(Time());
}
//-----
void __fastcall TForm1::btnConnectClick(TObject *Sender)
{
    switch (btnConnect->Tag)
    {
        case 0:
        {
            ClientSocket1->Host  = leAddress->Text;

```

```

ClientSocket1->Address = leAddress->Text;
ClientSocket1->Port  = StrToInt(lePort->Text);
Sleep(500);
ClientSocket1->Open();
if (!ClientSocket1->Active)
{
    mLog->Lines->Add "[" + TimeToStr(Time()) + "] Подключаемся...");
    leNickname->Enabled = false;
    leAddress->Enabled = false;
    lePort->Enabled = false;
    btnConnect->Tag = 1;
    btnConnect->Caption = "Disconnect!";
}
else
{
    mLog->Lines->Add "[" + TimeToStr(Time()) + "] Не удалось
ПОДКЛЮЧИТЬСЯ!");
    leNickname->Enabled = true;
    leAddress->Enabled = true;
    lePort->Enabled = true;
    btnConnect->Tag = 0;
    btnConnect->Caption = "Connect!";
}
}
break;
case 1:
{
    ClientSocket1->Close();
    if (!ClientSocket1->Active)
    {
        mLog->Lines->Add "[" + TimeToStr(Time()) + "] Отключились от
сервера!");
        leNickname->Enabled = true;
        leAddress->Enabled = true;
        lePort->Enabled = true;
        btnConnect->Tag = 0;
        btnConnect->Caption = "Connect!";
        lbUsers->Clear();
        Edit1->Enabled = false;
        Edit1->Text = "[ поле для ввода текста ]";
    }
    else
    {

```

```

        mLog->Lines->Add("[ " + TimeToStr(Time()) + " ] Не удалось
отключиться!");
        leNickname->Enabled = false;
        leAddress->Enabled = false;
        lePort->Enabled = false;
        btnConnect->Tag = 1;
        btnConnect->Caption = "Disconnect!";
    }
}
break;
}
}
//-----
void __fastcall TForm1::ClientSocket1Connecting(TObject *Sender,
        TCustomWinSocket *Socket)
{
    mLog->Lines->Add("[ " + TimeToStr(Time()) + " ] Сервер найден!
Устанавливаю соединение...");
}
//-----
void __fastcall TForm1::ClientSocket1Read(TObject *Sender,
        TCustomWinSocket *Socket)
{
    AnsiString S;
    S = Socket->ReceiveText();
    if ((S[1] == '#') && (S.Length() > 1)) lbUsers->Items->Text =
S.SubString(2, S.Length());
    if (S[1] != '#')
    {
        AnsiString RName, Msg;
        RName = S.SubString(1, S.Pos("~")-1);
        Msg = S.SubString(S.Pos("~")+1, S.Length());
        mLog->Lines->Add("[ " + TimeToStr(Time()) + " ] <" + RName + "> " +
Msg);
    }
}
//-----
void __fastcall TForm1::ClientSocket1Connect(TObject *Sender,
        TCustomWinSocket *Socket)
{
    ClientSocket1->Socket->SendText("#" + leNickname->Text);
    mLog->Lines->Add("[ " + TimeToStr(Time()) + " ] Подключились к " +
Socket->RemoteAddress + ", порт: " + IntToStr(Socket->RemotePort));
    Edit1->Enabled = true;
}

```

```

    Edit1->Text = "";
}

//-----
void __fastcall TForm1::ClientSocket1Error(TObject *Sender,
    TCustomWinSocket *Socket, TErrorEvent ErrorEvent, int &ErrorCode)
{
    switch(ErrorEvent)
    {
        case eeConnect:
        {
            mLog->Lines->Add("Не удалось подключиться к серверу!");
            leNickname->Enabled = true;
            leAddress->Enabled = true;
            lePort->Enabled = true;
            btnConnect->Tag = 0;
            btnConnect->Caption = "Connect!";
        }
        break;
    }
    ErrorCode = 0;
}

//-----
void __fastcall TForm1::Edit1KeyDown(TObject *Sender, WORD &Key,
    TShiftState Shift)
{
    if (Key == 13)
    {
        if (lbUsers->ItemIndex != -1)
            ClientSocket1->Socket->SendText(lbUsers->Items->operator
[[lbUsers->ItemIndex) + "#" + leNickname->Text + "~" + Edit1->Text);
        else
            ClientSocket1->Socket->SendText("All#" + leNickname->Text + "~" +
Edit1->Text);
        Edit1->Text = "";
    }
}

//-----
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
    TStream*   SF   =   new   TFileStream("test.txt",   fmOpenRead,
fmShareDenyNone);
    ClientSocket1->Socket->SendText("All#" + leNickname->Text + "~%" +
IntToStr(sizeof(SF)) + "&test.txt");
}

```

```
int sending = ClientSocket1->Socket->SendBuf(SF, sizeof(SF));
if (sending > 0) mLog->Lines->Add("* [" + TimeToStr(Time())+ "] Послан
файл test.txt...");
else mLog->Lines->Add("* [" + TimeToStr(Time())+ "] Не удалось
ПОСЛАТЬ файл test.txt...");
SF->Free();
}
//-----
```