Лабораторная работа

Тема: Создание сетевого приложения на основе TCP сокетов

Теоретическая часть

Сокет — конечная точка связи двустороннего канала между 2 компьютерами.

Если мы соединим 2 сокета, то получим канал, через который можно передавать данные в обе стороны. Одна сторона канала называется сервером, другая — клиентом.

Для передачи/приема данных нужно открыть канал. В конце всех операций — закрыть.

Типы сокетов

Существует 2 вида сокетов: потоковые, дейтаграммные.

Потоковый сокет — это сокет, который состоит из потока байтов, который может быть двунапрямленным (в обе стороны). Он берет на себя всю ответственность о доставке данных и исправлении ошибок. Особенностью есть возможность передачи больших объемов данных. Использует протокол TCP (Transmission Control Protocol), именно который обеспечивает поступление данных на другую сторону в нужной последовательности и без ошибок.

Дейтаграммный сокет — в отличие от потокового, имеет ограничения по размеру. Реализован через протокол UDP (User Datagram Protocol), который не отвечает за приход в конечную точку всех данных. Одним из плюсов — не нужно создавать соединения между 2 сторонами. Это очень важно, когда затраты времени недопустимы.

Более подробная информация

http://msdn.microsoft.com/ru-ru/library/system.net.sockets.socket(v=vs.110).aspx

Практическая часть

Для примера создадим приложение-чат. Приложение будет состоять из двух частей: серверное приложение и клиентское приложение.

Реализация серверной части

Создадим новый проект Приложение Windows Forms на языке C#.

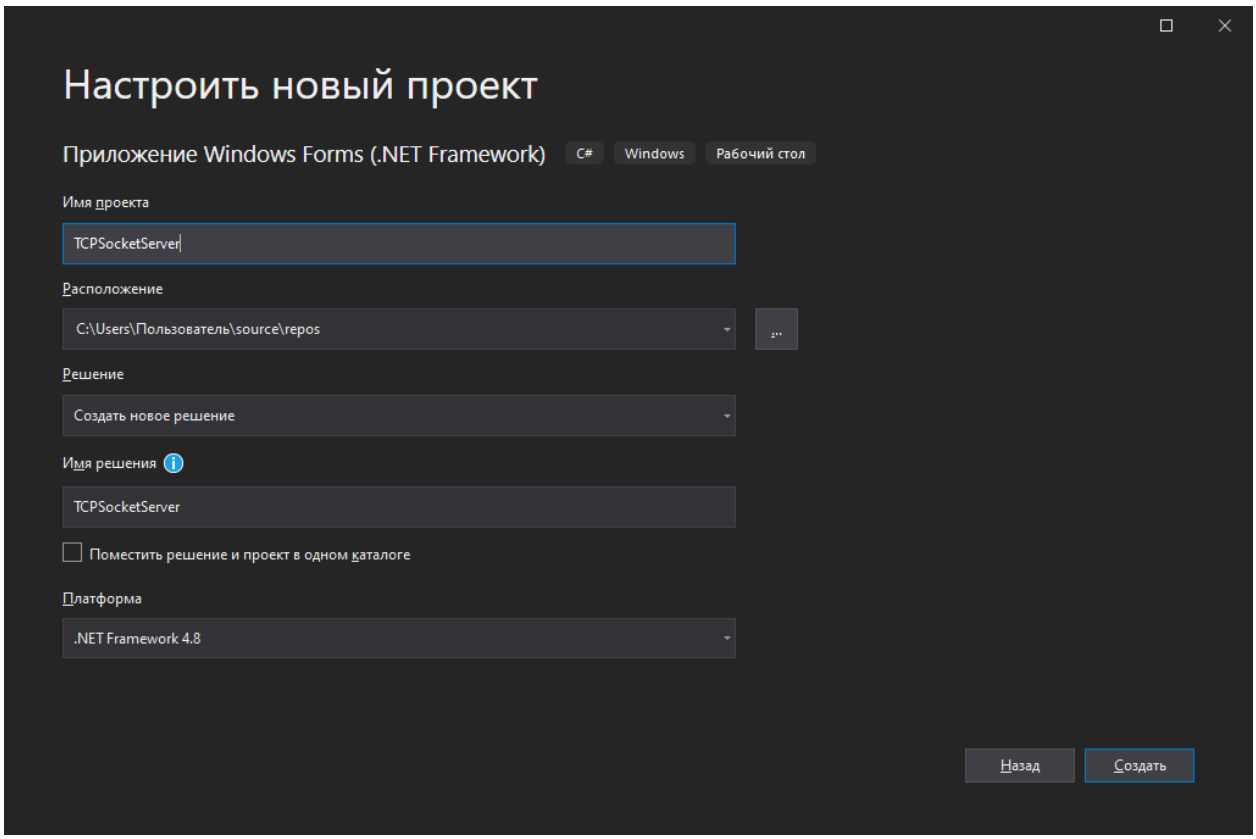

Рис.1. Создание проекта приложения

Добавим в проект пространства имен, методы которых будут использованы в проекте:

```
// Добавляем пространства имен для работы сокетов
using System.Net.Sockets;
using System.Net;
using System.Threading;
```

Объявим необходимые глобальные переменные:

```
// Раздел глобальных переменных
private static Socket Server; // Создаем объект сокета-сервера
private static Socket Handler; // Создаем объект вспомогательного сокета

private static IPHostEntry ipHost; // Класс для сведений об адресе веб-узла
private static IPAddress ipAddr; // Предоставляет IP-адрес
private static IPEndPoint ipEndPoint; // Локальная конечная точка

private static Thread socketThread; // Создаем поток для поддержки потока
private static Thread WaitingForMessage; // Создаем поток для приёма сообщений
```

Реализуем функцию startSocket для запуска сокета с целью прослушивания и ожидания подключения клиента:

```
// Функция запуска сокета
 private void startSocket()
 {
     // IP-адрес сервера, для подключения
     string HostName = "0.0.0.0";
     // Порт подключения
     string Port = tbPort.Text;
```

```
            // Разрешает DNS-имя узла или IP-адрес в экземпляр IPHostEntry.
            ipHost = Dns.Resolve(HostName);
            // Получаем из списка адресов первый (адресов может быть много)
            ipAddr = ipHost.AddressList[0];
            // Создаем конечную локальную точку подключения на каком-то порту
            ipEndPoint = new IPEndPoint(ipAddr, int.Parse(Port));

            try
            {
                // Создаем сокет сервера на текущей машине
                Server = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
            }
            catch (SocketException error)
            {
                // 10061 – порт подключения занят/закрыт
                if (error.ErrorCode == 10061)
                {
                    MessageBox.Show("Порт подключения закрыт!");
                    Application.Exit();
                }
            }

            // Ждем подключений
            try
            {
                // Связываем удаленную точку с сокетом
                Server.Bind(ipEndPoint);
                // Не более 10 подключения на сокет
                Server.Listen(10);

                // Начинаем "прослушивать" подключения
                while (true)
                {
                    Handler = Server.Accept();
                    if (Handler.Connected)
                    {
                        // Позеленим кнопочку для красоты, чтобы пользователь знал, что
                        соединение установлено
                        bConnect.Invoke(new Action(() => bConnect.Text = "Клиент
                        подключен"));
                        bConnect.Invoke(new Action(() => bConnect.BackColor =
                        Color.Green));
                        // Создаем новый поток, указываем на ф-цию получения сообщений в
                        классе Worker
                        WaitingForMessage = new System.Threading.Thread(new
                        System.Threading.ParameterizedThreadStart(GetMessages));
                        // Запускаем, в параметрах передаем листбокс (история чата)
                        WaitingForMessage.Start(new Object[] { lbHistory });
                    }
                    break;
                }
            }
            catch (Exception e)
            {
                throw new Exception("Проблемы с подключением");
            }

    }
```

Запуск прослушивающего сокета производится по нажатию кнопки в отдельном потоке. Это необходимо для сохранения функциональности формы приложения:

```
socketThread = new Thread(new ThreadStart(startSocket));
socketThread.IsBackground = true;
socketThread.Start();
bConnect.Enabled = false;
bConnect.Text = "Ожидание подключения";
bConnect.BackColor = Color.Yellow;
```

В рамках метода startSocket при подключении клиента происходит запуск потока WaitingForMessage для организации принятия сообщений от клиента. В рамках потока исполняется метод GetMessages:

```
// Ф-ция, работающая в новом потоке: получение новых сообщений ———
 public static void GetMessages(Object obj)
 {
     // Получаем объект истории чата (лист бокс)
     Object[] Temp = (Object[])obj;
     System.Windows.Forms.ListBox ChatListBox =
(System.Windows.Forms.ListBox)Temp[0];

     // В бесконечном цикле получаем сообщения
     while (true)
     {
         // Ставим паузу, чтобы на время освобождать порт для отправки сообщений
         Thread.Sleep(50);

             try
             {
                 // Получаем сообщение от клиента
                 string Message = GetDataFromClient();
             // Добавляем в историю + текущее время
             ChatListBox.Invoke(new Action(() =>
ChatListBox.Items.Add(DateTime.Now.ToShortTimeString() + " Client:  " + Message)));
         }
             catch { }

     }
 }
```

В свою очередь метод GetMessages использует метод GetDataFromClient для принятия и обработки сообщений от клиента:

```
// Получение информации от клиента
public static string GetDataFromClient()
{
    string GetInformation = "";

    byte[] GetBytes = new byte[1024];
    int BytesRec = Handler.Receive(GetBytes);

    GetInformation += Encoding.Unicode.GetString(GetBytes, 0, BytesRec);

    return GetInformation;
}
```

Отправка сообщений от сервера к клиенту производится с использованием кода:

```
            // Посылаем клиенту новое сообщение
            SendDataToClient(tbMessage.Text);
            // Добавляем в историю свое же сообщение + ник + время написания
            lbHistory.Items.Add(DateTime.Now.ToShortTimeString() + " Server: " +
tbMessage.Text.ToString());
            // Автопрокрутка истории чата
            lbHistory.TopIndex = lbHistory.Items.Count - 1;
            // Убираем текст из поля ввода
            tbMessage.Text = "";
```

При этом используется метод SendDataToClient:

```
// Отправка информации на клиент
 public static void SendDataToClient(string Data)
 {
     byte[] SendMsg = Encoding.Unicode.GetBytes(Data);
     Handler.Send(SendMsg);
 }
```

Внешний вид приложения приведен на рисунке 2.

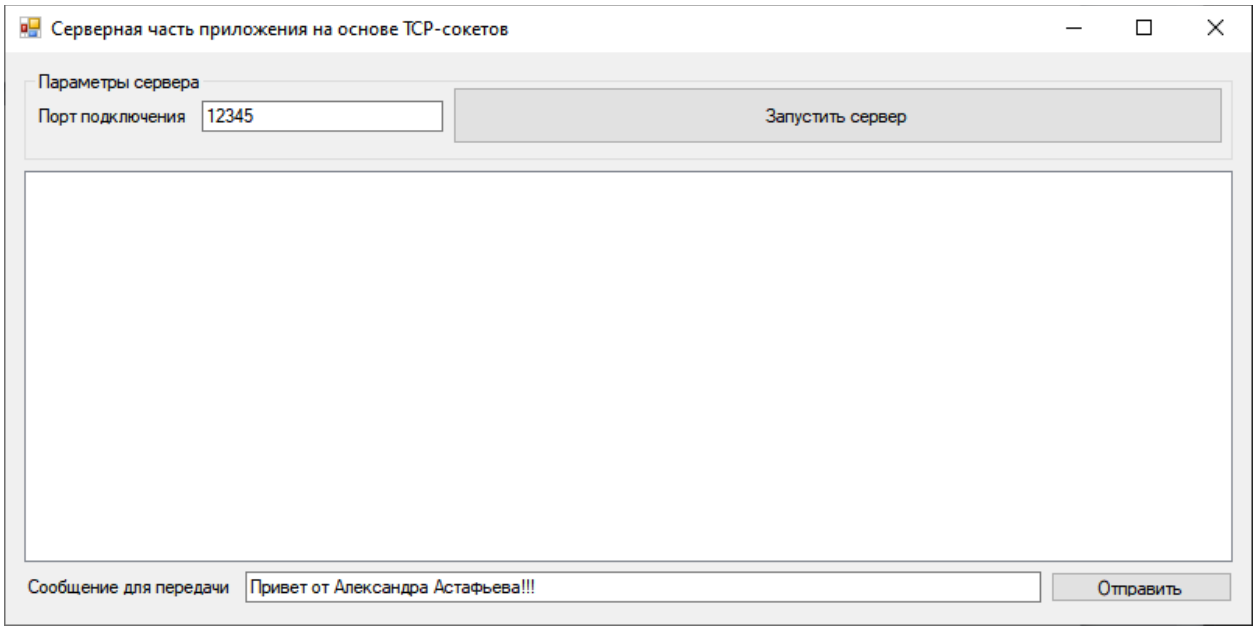

Рис. 2. Внешний вид приложения

Реализация клиентской части

Программная реализация клиентской части очень схожа с серверной. На первом этапе подключаем пространства имен:

```
// Добавляем пространства имен для работы сокетов
using System.Net.Sockets;
using System.Net;
using System.Threading;
```

Объявим необходимые глобальные переменные:

```
private static Socket Client; // Создаем объект сокета-сервера

private static IPHostEntry ipHost; // Класс для сведений об адресе веб-узла
private static IPAddress ipAddr; // Предоставляет IP-адрес
private static IPEndPoint ipEndPoint; // Локальная конечная точка

private static Thread socketThread; // Создаем поток для поддержки потока
private static Thread WaitingForMessage; // Создаем поток для приёма сообщений
```

Вид метода startSocket выглядит следующим образом:

```
private void startSocket()
 {
     // IP-адрес сервера, для подключения
     string HostName = tbAddress.Text;
     // Порт подключения
     string Port = tbPort.Text;

     // Разрешает DNS-имя узла или IP-адрес в экземпляр IPHostEntry.
     ipHost = Dns.Resolve(HostName);
     // Получаем из списка адресов первый (адресов может быть много)
     ipAddr = ipHost.AddressList[0];
     // Создаем конечную локальную точку подключения на каком-то порту
     ipEndPoint = new IPEndPoint(ipAddr, int.Parse(Port));

     try
     {
         // Создаем сокет на текущей машине
         Client = new Socket(AddressFamily.InterNetwork,
         SocketType.Stream, ProtocolType.Tcp);
         while (true)
         {
             // Пытаемся подключиться к удаленной точке
             Client.Connect(ipEndPoint);
             if (Client.Connected) // Если клиент подключился
             {
                 // Позеленим кнопочку для красоты, чтобы пользователь знал, что
                  соединение установлено
                 bConnect.Invoke(new Action(() => bConnect.Text = "Подключение
                  установлено"));
                 bConnect.Invoke(new Action(() => bConnect.BackColor =
                  Color.Green));
                 // Создаем новый поток, указываем на ф-цию получения сообщений в
                  классе Worker
                 WaitingForMessage = new System.Threading.Thread(new
                  System.Threading.ParameterizedThreadStart(GetMessages));
                 // Запускаем, в параметрах передаем листбокс (история чата)
                 WaitingForMessage.Start(new Object[] { lbHistory });
             }
             break;
         }
     }
     catch (SocketException error)
     {
         // 10061 – порт подключения занят/закрыт
         if (error.ErrorCode == 10061)
         {
             MessageBox.Show("Порт подключения закрыт!");
             Application.Exit();
         }
     }
 }
```

Метод GetMessages:

```
        // Ф-ция, работающая в новом потоке: получение новых сообщенй ———
         public static void GetMessages(Object obj)
         {
             // Получаем объект истории чата (лист бокс)
             Object[] Temp = (Object[])obj;
             System.Windows.Forms.ListBox ChatListBox =
(System.Windows.Forms.ListBox)Temp[0];

             // В бесконечном цикле получаем сообщения
             while (true)
             {
                 // Ставим паузу, чтобы на время освобождать порт для отправки сообщений
                 Thread.Sleep(50);
                     try
                     {
                         string Message = GetDataFromServer();
                         ChatListBox.Invoke(new Action(() =>
ChatListBox.Items.Add(DateTime.Now.ToShortTimeString() + " Server: " + Message)));
                     }
                     catch { }
             }
         }
```

Метод GetDataFromServer:

```
         // Получение данных от сервера
         public static string GetDataFromServer()
         {
             string GetInformation = "";

             // Создаем пустое “хранилище” байтов, куда будем получать информацию
             byte[] GetBytes = new byte[1024];
             int BytesRec = Client.Receive(GetBytes);

             // Переводим из массива битов в строку
             GetInformation += Encoding.Unicode.GetString(GetBytes, 0, BytesRec);

             return GetInformation;
         }
```

Метод SendDataToServer:

```
        // Отправка информации на сервер
         public static void SendDataToServer(string Data)
         {
             // Используем unicode, чтобы не было проблем с кодировкой, при приеме
информации
             byte[] SendMsg = Encoding.Unicode.GetBytes(Data);
             Client.Send(SendMsg);
         }
```

Код запуска сокета:

```
        socketThread = new Thread(new ThreadStart(startSocket));
        socketThread.IsBackground = true;
        socketThread.Start();
        bConnect.Enabled = false;
```

Код отправки сообщения на сервер:

```
        private void button2_Click(object sender, EventArgs e)
         {
             // Посылаем клиенту новое сообщение
             SendDataToServer(tbMessage.Text);
             // Добавляем в историю свое же сообщение + ник + время написания
             lbHistory.Items.Add(DateTime.Now.ToShortTimeString() + " Client:  " +
tbMessage.Text.ToString());
             // Автопрокрутка истории чата
             lbHistory.TopIndex = lbHistory.Items.Count - 1;
             // Убираем текст из поля ввода
             tbMessage.Text = "";
         }
```

Внешний вид клиентской части приведен на рисунке 3.

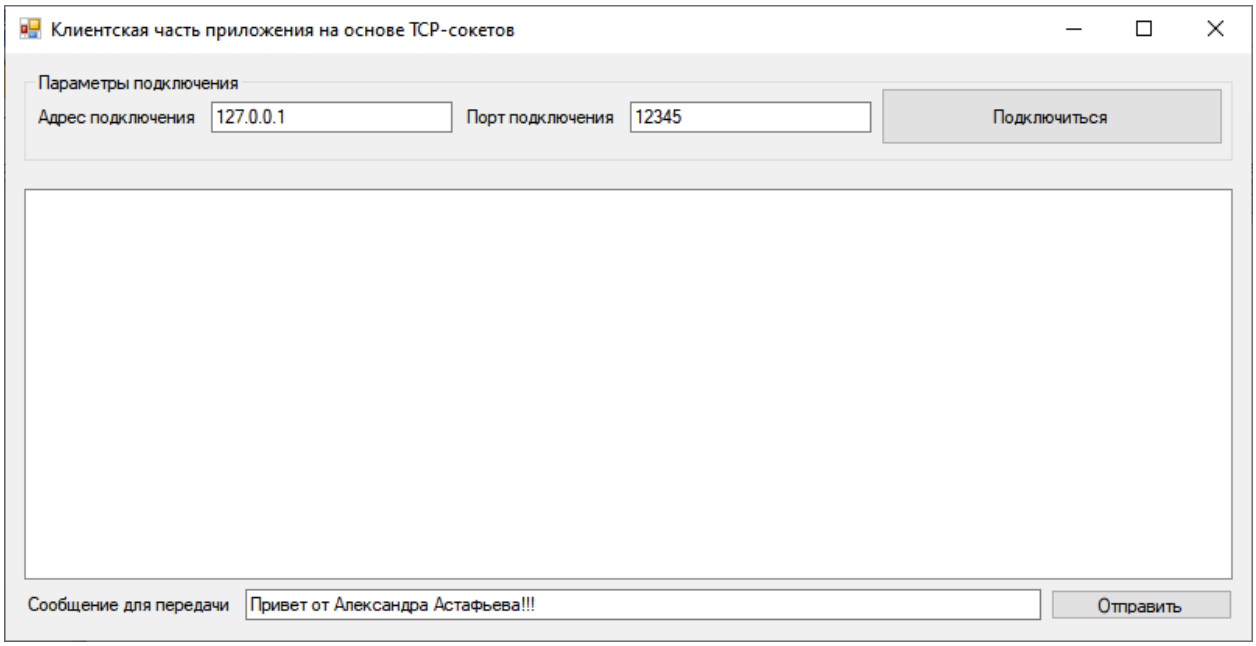

Рис. 3. Внешний вид клиентской формы

Процесс работы приложения:

1. Запускаем приложение-сервер.
2. Запускаем приложение-клиент.
3. На серверной части запускаем сокет с использованием кнопки «Запустить сервер». При этом необходимо указать порт подключения.
4. На клиентской части необходимо запустить сокет путем нажатия кнопки «Подключиться». При этом необходимо указать адрес подключения и порт подключения.
5. После успешной связи кнопки приложений окрасятся в зеленый цвет.
6. Для дальнейшей работы необходимо передавать сообщения с использованием кнопки «Отправить».

7. Выполнить индивидуальное задание согласно списка ниже.

Пример работы клиент-серверного приложения приведен на рисунке 4.

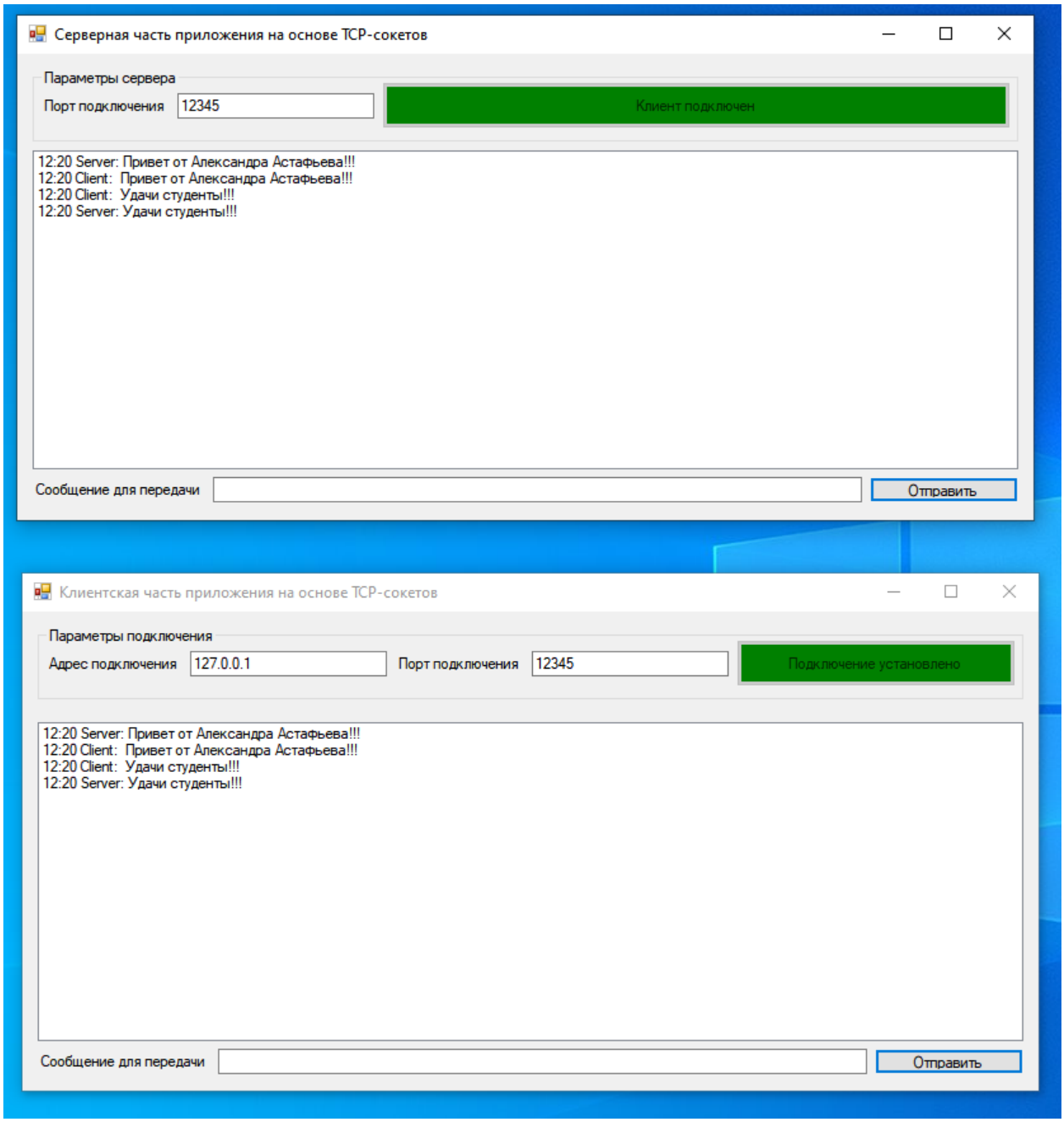

Рис. 4. Пример работы клиент-серверного приложения

Индивидуальное задание:

Используя сокеты в качестве механизма сетевого взаимодействия, создать приложение для решения следующей задачи:

1. Передать серверу список строк. Сервер должен вернуть клиенту длину самой длинной строки и саму строку.
2. Передать серверу список строки и букву. Сервер должен вернуть список строк исключая строки, начинающиеся на эту букву.
3. Передать на сервер два множества. Сервер должен вернуть клиенту пересечение этих множеств.
4. Передать на сервер список строк на русском и английском языках, а также числа. Сервер должен посчитать и вернуть количество русских строк, количество английских и количество чисел.
5. Передать на сервер список строк и букву. Сервер должен посчитать количество строк, которые содержат эту букву только один раз.
6. Передать на сервер множество чисел. Сервер должен вернуть сумму этих чисел и наименьший делитель этой суммы.
7. Передать на сервер множество чисел. Сервер должен вернуть среднее значение этого множества и множество с отклонениями от него каждого элемента.
8. Передать на сервер два множества. Сервер должен вернуть одно множество, состоящее из чисел обоих множеств, отсортированных по возрастанию.
9. Передать на сервер два множества. Сервер должен вернуть два множества исключив из них элементы, имеющиеся в обоих множествах.
10. Передать на сервер список строк. Сервер должен вернуть отсортированный по алфавиту в обратном порядке список этих строк.
11. Передать на сервер список строк на русском языке. Сервер должен вернуть количество использований каждой из букв во всех строках списка.
12. Передать на сервер список строк и два числа. Сервер должен вернуть список строк длинной от и до переданных чисел.