МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Невинномысский технологический институт (филиал)

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к выполнению лабораторных работ по дисциплине «Инфокоммуникационные системы» для студентов направления 15.03.04 Автоматизация технологических процессов и производств

Невинномысск 2021

Методические указания предназначены для студентов направления подготовки 15.03.04 Автоматизация технологических процессов и производств. Они содержат основы теории, порядок проведения лабораторных работ и обработки экспериментальных данных, перечень контрольных вопросов для самоподготовки и список рекомендуемой литературы. Работы подобраны и расположены в соответствии с методикой изучения дисциплины «Инфокоммуникационные системы». Объем и последовательность выполнения работ определяются преподавателем в зависимости от количества часов, предусмотренных учебным планом дисциплины.

Методические указания разработаны в соответствии с требованиями и рекомендациями ФГОС ВО в части содержания и уровня подготовки выпускников направления 15.03.04 Автоматизация технологических процессов и производств

Составители: канд. техн. наук, А.А. Евдокимов

Рецензент: канд. техн. наук , доцент Д.В. Болдырев

Содержание

Лабораторная работа 1

Обмен информацией по протоколу TCP

Цель работы: получить навык разработки приложения-сервера и приложения-клиента на базе транспортного протокола TCP.

Краткие сведения из теория

Протокол TCP (Transmission Control Protocol) предназначен для организации обмена данными между компьютерами сети, обеспечивая соединение между компьютерами перед началом обмена данными. Данные могут передаваться в обоих направлениях. В C# классы, поддерживающие взаимодействие приложений по протоколу TCP, являются одними из самых простых классов, которые обеспечивают сетевое взаимодействие по потоковым принципам. Существует более низкоуровневая технология – технология сокетов – концов двустороннего канала связи между двумя программами. Классы работы с протоколом TCP также используют сокеты, но ими управляют методы самих классов, а не программисты. Архитектура таких клиент-серверных комплексов приложений содержит два приложения – приложение-сервер и приложение-клиент. Приложение-сервер не инициирует соединений – оно ждет, когда к нему присоединяться клиенты. Ожидание заключается в постоянном прослушивании определенного порта компьютера. Приложение-клиент при выполнении соединения указывает IP-адрес компьютера, на котором установлено приложение-клиент, и порт, который этот сервер прослушивает. После соединения можно осуществлять обмен данными на основании созданных соединением потоков ввода и вывода (аналогичные потоки будут созданы и в приложении-клиенте, и в приложении-сервере). Разберем последовательность действий при создании комплекса сетевых приложений, который называют «Эхо-сервер». Особенность этого комплекса заключается в том, что приложение-сервер откликается клиенту «эхом», повторяя сообщение, полученное от клиента. Клиент отключается, когда отсылает сообщение «QUIT».

Код приложения-клиента:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
// пространства имен для работы в сети
```

```
using System.Net;
using System.Net.Sockets;
// пример приложения эхо-клиента
namespace Client
{

    class Program
    {
    // указание номера порта приложения-сервера
        const int ECHO_PORT = 8080;
        public static void Main(string[] args)
        {
            Console.Write("Имя:");
            string UserName = Console.ReadLine();

        Console.WriteLine("Протокол работы:");
            try
            {
                // подсоединение к приложению-серверу
            TcpClient eClient = new
TcpClient("127.0.0.1",ECHO_PORT);
                // получение из соединения потока ввода
            StreamReader rs = new
StreamReader(eClient.GetStream());
                // создание потока вывода для
соединения
            NetworkStream ws = eClient.GetStream();
        // формирование текста, включающего имя клиента
                string dataToSend=UserName+"\r\n";
            // отсылка его на сервер
                byte[] data =
Encoding.UTF8.GetBytes(dataToSend);
                ws.Write(data,0,data.Length);
            // цикл диалога с сервером
                while(true)
                {
          // формируем сообщение для отсылки на сервер
              Console.Write(UserName+":");
                  dataToSend=Console.ReadLine();
                  dataToSend+="\r\n";
              // отсылаем сообщение

data=Encoding.UTF8.GetBytes(dataToSend);
```

```
                    ws.Write(data,0,data.Length);
// если хотим закончить диалог, вводим QUIT
// и выходим из цикла, так как сервер на него не
//ответит
                    if(dataToSend.IndexOf("QUIT")>-1)
                        break;
                    // ждем ответ от сервера
                  string returnData=rs.ReadLine();
                  // печатаем полученное сообщение
                 Console.WriteLine("Сервер: "+returnData);
                    }
                // сеанс диалога с сервером закончен,
                // поэтому закрываем соединение
                    eClient.Close();
                }
                catch(Exception exp)
                {
    // вывод сообщения в случае возникновения оцибки
                    Console.WriteLine("Исключение:" + exp);
                }
            }
        }
}
```

Таким образом, приложение-клиент может состоять из одной функции Main, в которой:

1. создается соединение с приложением-сервером:

```
    TcpClient eClient = new
TcpClient("127.0.0.1",ECHO_PORT);
```

2. формируются потоки ввода и вывода для этого соединения:

```
    // получение из соединения потока ввода
    StreamReader rs = new
StreamReader(eClient.GetStream());
      // создание потока вывода для соединения
    NetworkStream ws = eClient.GetStream();
```

3. организуется диалог с сервером:

```
while(true)
{
     // формируем сообщение для отсылки на сервер
dataToSend
     .  .   .
     // отсылка сообщения на сервер
     data=Encoding.UTF8.GetBytes(dataToSend);
     ws.Write(data,0,data.Length);
     // ждем ответ от сервера
     .  .   .
     string returnData=rs.ReadLine();
     // обрабатываем полученное сообщение

}
```

4. закрывается соединение:

```
eClient.Close();
```

Теперь разберем приложение-сервер. Оно имеет более сложную структуру, поскольку ему следует отслеживать диалог с каждым из клиентов в отдельности. Таким образом, приложение имеет два класса: класс для работы с отдельным клиентом и класс-сервер. Класс работы с отдельным клиентом должен иметь поле, хранящее сокет связи именно с этим клиентом. Через этот сокет класс получает доступ к потокам ввода и вывода для связи с клиентом. Также в классе должен быть метод, который осуществляет диалог с клиентом – получает потоки для ввода и вывода и формирует цикл получения и отсылки сообщений клиенту. По окончании диалога сокет закрывается, одновременно закрывая потоки ввода и вывода.

```
// класс управления диалога с клиентом
public class ClientHandler
{
     // сокет клиента
     public TcpClient clSocket;
     // метод осуществления диалога с клиентом
     public void RunClient()
     {
          // создание потоков ввода и вывода с
          //приложением-клиентом
```

```
            StreamReader rs = new
        StreamReader(clSocket.GetStream());
            NetworkStream ws = clSocket.GetStream();
        // получение имени клиента, который подключился
        //к серверу
            string returnData = rs.ReadLine();
            string userName = returnData;
            Console.WriteLine(userName + " на сервере");
          // цикл диалога с клиентом
            while (true)
            {
                // получение сообщения от клиента
                    returnData = rs.ReadLine();
                // если клиент перешлет собщение QUIT,
        //диалог заканчивается
                    if (returnData.IndexOf("QUIT") > -1)
                    {
                        Console.WriteLine("До свидания,
        "+userName);
                        break;
                    }
                    Console.WriteLine(userName + ": " +
        returnData);
                    returnData += "\r\n";
          // сервер формирует и отсылает полученное
          // сообщение – отзывается «эхом»
                byte[] dataWrite =
        Encoding.UTF8.GetBytes(returnData);
                    ws.Write(dataWrite, 0,
        dataWrite.Length);
             }
          // закрывает сокет после окончания диалога
            clSocket.Close();
        }
    }
```

Класс-сервер должен, во-первых, начать прослушивать заданный порт, во-вторых, организовать цикл, в котором будет обрабатываться подключение одного клиента. Поскольку неизвестно, сколько приложений-клиентов будет подключаться, цикл должен быть бесконечным. Внутри его итерации происходи получение информации о приложении-клиенте (сокет), создается объект для организации диалога с этим клиентом и для него вызывается метод

RunClient. Так как два и более клиент могут одновременно отсылать сообщения серверу, методы RunClient для нескольких объектов должны работать одновременно. Это достигается тем, что эти методы вызывается как отдельные потоки приложения, т.е. приложение-сервер должно поддерживать многопотоковость.

```
    // приложение-сервер
    class Program
    {
        // порт для прослушивания
      const int ECHO_PORT = 8080;
      // переменная для хранения количества
      //подключенных клиентов
        public static int nClients = 0;
        static void Main(string[] args)
        {
            try
            {
// создается объект для ожидания подключения клиентов
//по протоколу TCP к конкретному порту
            TcpListener clListener = new
TcpListener(ECHO_PORT);
                // начало прослушивания
            clListener.Start();
                Console.WriteLine("Ждемс...");
            // цикл ожидания подключения клиентов
                while(true)
                {
        // фиксация подключения очередного клиента
                    TcpClient client =
clListener.AcceptTcpClient();
// создание объекта для диалога с подключившимся
//клиентом
                    ClientHandler cHandler = new
ClientHandler();
        // указываем сокет для подключения к клиенту
                    cHandler.clSocket=client;
// формирование отдельного потока для общения с этим
//клиентом вызов метода диалога с клиентом
                    Thread clientTread = new Thread
            (new ThreadStart(cHandler.RunClient));
                    // запуск этого потока
```

```
            clientTread.Start();
             }
        // завершение прослушивания
            clListener.Stop();
        }
        catch(Exception exp)
        {
        // вывод информации о возникшей ошибке
            Console.WriteLine("Исключение:" + exp);
        }
    }
}
```

Задание к работе

1. Разработать два оконных приложения операционной системы MS Windows: приложение-сервер и приложение-клиент. Приложение-клиент отправляет запрос на перевод чисел из одной системы счисления в другую и число. Интерфейс приложения-клиента должен поддерживать изменение IP-адреса и порта для соединения, типа запроса, числа, вывод результата. Приложение-сервер производит преобразование числа в соответствии с принятым запросом и отправляет результат клиенту. Интерфейс приложения-сервера должен поддерживать изменение IP-адреса и порта для соединения

2. Произвести тестирование приложения, составить отчет и защитить преподавателю.

Таблица 1 – Варианты задания к лабораторной работе № 1.

Вариант	Запрос 1	Запрос 2
1	10 → 16	16 → 10
2	2 → 10	10 → 2
3	10 → 8	8 → 10

4	3 → 10	10 → 3
5	3 → 16	16 → 3
6	10 → Фибоначчи	Фибоначчи → 10
7	2 → 3	3 → 2
8	Фибоначчи → 2	2 → Фибоначчи
9	10 → СОК	СОК → 10
10	16 → СОК	СОК → 16
11	Фибоначчи → СОК	СОК → Фибоначчи
12	16 → Фибоначчи	Фибоначчи → 16
13	3 → СОК	СОК → 3
14	8 → Фибоначчи	Фибоначчи → 8
15	10 → Смешанная	Смешанная → 10

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером варианта, фамилией студента и группы.

2. Цель работы, задание.

3. Код разработанных приложений.

4. Снимки экрана, поясняющие работу приложений

5. Выводы о проделанной работе.

Контрольные вопросы

1. Заголовок TCP.

2. Начала сеанса TCP.

3. Завершение сеанса TCP.

4. Функции протокола TCP.

5. Операции TCP.

6. Класс TcpClient.

5. Домены приложений.

6. Как работает .NET.Remoting.

7. Высокоуровневые протоколы, использующие TCP.

8. Достоинства и недостатки TCP.

Лабораторная работа 2

Обмен информацией по протоколу UDP

Цель работы: получить навык разработки приложений для обмена информацией на базе протокола UDP.

Краткие сведения из теории

Протокол UDP (User Datagramm Protocol) – это простой, ориентированный на дейтаграммы протокол без организации соединения, предоставляющий быстрое, но необязательно надежное транспортное обслуживание. Дейтаграмма – это отдельный, независимый пакет данных, несущий информацию, достаточную для передачи от источника до пункта назначения, поэтому никакого дополнительного обмена между источником, адресатом и транспортной сетью не требуется. Этот протокол используют в случаях, когда приложению нужно осуществлять групповую рассылку, когда размеры дейтаграмм невелики, если приложение не требует большого объема данными, когда повторная передача не требуется, когда требуются низкие накладные расходы. Разберем простой вид приложения, который можно считать простым чатом между клиентами. Два пользователя, которые хотят общаться с помощью такого чата, должны установить одно и то же приложение. Каждый экземпляр находятся на своем компьютере и прослушивает некоторый порт. Для осуществления общении пользователь указывает свой порт, IP-адрес удаленного пользователя и порт, который он прослушивает. После запускается два потока приложения – поток принятия данных с удаленной точки и поток формирования и отсылки данных на удаленную точку собеседнику. Для получения каждого сообщения (дейтаграммы) соединение между удаленными точками устанавливается заново. Для тестирования работы данного приложения нужно запустить два экземпляра этого приложения на одной машине, указав локальный адрес 127.0.0.1 и разные порты для этих экземпляров. Для реализации такого приложения-чата требуется только один

класс, в котором основной поток используется для отсылки сообщений удаленному экземпляру этого приложения и создается другой поток, который осуществляет получение сообщений от удаленного пользователя. Таким образом, ввод нового сообщения и получение нового сообщения от удаленного пользователя могут произойти одновременно. Реализуется это приложение с помощью трех функций – функция Main, которая запускает потоки чтения и записи, функция Send, которая формирует и отсылает сообщение, и функция Receive, которая принимает сообщения с удаленной точки.

Функция Main разбивается на этапы:

- ввод информации о соединении – IP-адрес и порт удаленного пользователя, порт локального компьютера;
- создание отдельного потока для вызова функции получения сообщений Receive;
- запуск цикла формирования новых сообщений для отсылки (вызов функции Send).

Функция Receive получает данные от удаленной точки и печатает это сообщение в окне приложения. Работает по следующей схеме:

- устанавливается соединение, инициатором которого является удаленная точка (объект класса UdpClient). Для этого достаточно указать только локальный порт, который будет прослушиваться нашей копией приложения;
- далее запускается цикл получения сообщений и вывода их на экран. Получение сообщения осуществляет метод Receive класса UdpClient. Этот метод через ссылку-параметр получает данные удаленной точки и возвращает набор байтов сообщения. Далее удаленную точку можно использовать для получения информации об IP-адресе и порте удаленного клиента.

Функция Send формирует данные для отправки собеседнику по схеме:

- устанавливается соединение (объект класса UdpClient), инициатором которого является наш экземпляр приложения;
- указывается удаленная точка посредством задания IP-адреса и номера удаленного порта;
- формируется сообщение и отсылается с помощью функции Send класса UdpClient. Метод Send получает в качестве параметров массив байтов для передачи, длину массива и удаленную точку, который требуется эти данные передать.

Код приложения-чата:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication4
{
    class Program
    {
        // IP-адрес удаленной точки
      private static IPAddress remoteIPAddress;
//порт, который прослушивается удаленным пользователем
        private static int remotePort;
//порт, который прослушивается нашим эквемпляром
//приложения
        private static int localPort;
       static void Main(string[] args)
        {
            try
            {
// ввод данных для организации соединения – адрес и
//номера портов
Console.WriteLine("Введите локальный порт");
localPort = Convert.ToInt16(Console.ReadLine());
Console.WriteLine("Введите удаленный порт");
remotePort = Convert.ToInt16(Console.ReadLine());
Console.WriteLine("Введите удаленный IP-адресс");
remoteIPAddress = IPAddress.Parse(Console.ReadLine());
// запуск потока принятия сообщения – запуск
// функции Receive как отдельного потока
Thread tRec = new Thread(new ThreadStart(Receive));
tRec.Start();
// бесконечный цикл формирования сообщений для отсылки
               while (true)
               {
// вызов функции отсылки нового сообщения, которое
// вводится с клавиатуры и передается в качестве
//параметра
               Send(Console.ReadLine());
               }
```

```
            }
            catch(Exception exp)
            {
                  // вывод сообщения о возникшей ошибке
             Console.WriteLine("Исключение:" +
         exp.ToString());
            }
          }
// функция отсылки сообщения datagram на удаленную
//точку
       private static void Send(string datagram)
         {
// создание объекта-соединения с удаленной точкой
         UdpClient sender = new UdpClient();
         // создание объекта с данными удаленной точки
            IPEndPoint endPoint = new
IPEndPoint(remoteIPAddress, remotePort);
            try
            {
// формирование массива байтов сообщения
             byte[] bytes =
         Encoding.UTF8.GetBytes(datagram);
// отсылка сообщения: первый параметр – массив
//отсылаемых байтов
// второй параметр – длина отсылаемых байтов
// третий параметр – данные удаленной точки,
// к которой производится отсылка
sender.Send(bytes, bytes.Length, endPoint);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
            finally
            {
// в любом случае соединение должно быть закрыто
             sender.Close();
            }
         }
// функция получения сообщений с удаленных точек (может
//быть несколько)
       public static void Receive()
         {
```

```
//получение соединения путем прослушивания своего
//локального порта
        UdpClient receivingUdpClient = new
    UdpClient(localPort);
        // обнуляем данные об удаленной точке -
        // значения заполнятся при соединении
            IPEndPoint RemoteIpEndPoint = null;
            try
            {
Console.WriteLine("Добро пожаловать в чат!");
                // цикл получения сообщений
             while (true)
                {
// получаем присылаемые байты, параметр - ссылка,
// которая заполнется данными удаленной точки,
// приславшей сообщение. То, что это ссылка,
//указывается с помощью ключевого слова ref.
              byte[] receiveBytes =
                    receivingUdpClient.Receive(ref
                RemoteIpEndPoint);
// формируется строковое представление полученных
//байтов и выводится сообщение на экран
                   string returnData =
Encoding.UTF8.GetString(receiveBytes);
Console.WriteLine("-" + returnData);
                }
            }
            catch (Exception e)
            {
// вывод сообщения о возникшей ошибке
Console.WriteLine("Исключение:" + e.ToString());
            }
        }
    }
}
```

Разберем поэтапно принципы построения приложения по обмену сообщениями двух пользователей.

1. Создадим проект типа WindowsApplication. За работу главного окна приложения отвечает класс Form1. Кнопки «Старт» и «Стоп» будут связаны с инициацией события начала диалога с другим пользователем и остановку этого диалога. Кнопка «Отправить» производит отправку очередного

сообщения собеседнику. Текстовое поле под кнопками предназначено для ввода имени собеседника. После старта диалога с пользователем оно должно стать недоступным. Ниже расположено текстовое поле, в которое пользователь вводит сообщение для отправки, и далее многострочное текстовое поле (его свойство Multyline=true), в котором пользователь видит принятые сообщения.

2. Для настройки IP-адреса и портов (локального и удаленного) создадим диалоговое окно (Form2 – окно проекта – контекстное меню – Добавить – Новую форму).

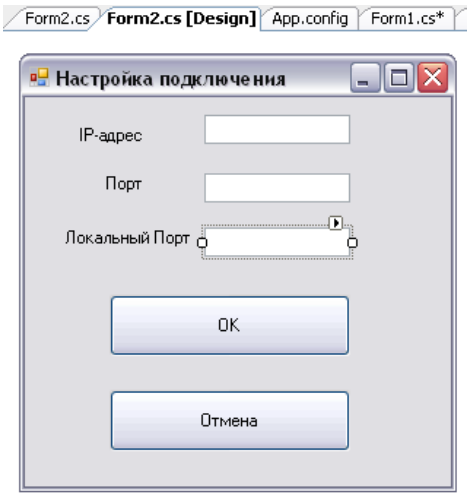

Рисунок 1 – Диалог настройки параметров соединения

Вызов этой форму будет происходить при выборе пункта контекстного меню «Настройка» главного окна:

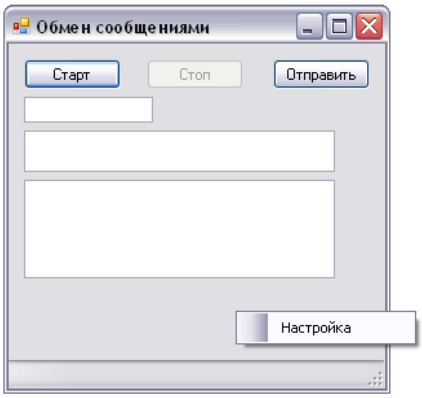

Рисунок 2 – Контекстное меню (contextMenuStrip1)

Для прикрепления контекстного меню требуется поместить в форму главного окна элемент управления ContextMenuStrip и выбрать его в свойство Form1 ContextMenuStrip. Для корректного закрытия диалогового окна

нужно установить свойства кнопок «OK» и «Отмена» DialogResult (OK или Cancel). На этом заканчивается процесс построения пользовательского интерфейса приложения, который выполняется дизайнером форм. Далее требуется написать код, обслуживающий данный интерфейс.

3. В класс Form2 следует добавить инструменты получения данных, которые вводятся в поля для задания IP-адреса и портов. Для этого в класс Form2 добавляются свойства:

```
// свойство получения IP-адреса.
// IPText – имя текстового поля ввода IP-адреса
public string IPAdress
{
     get { return IPText.Text; }
}

// свойство получения номера удаленного порта.
// PortBox – имя текстового поля ввода этого номера
public int Port
{
    get { return Int32.Parse(PortBox.Text); }
}

// свойство получения номера локального порта.
// LPortText – имя текстового поля ввода этого
номера
public int LocalPort
{
   get { return Int32.Parse(LPortText.Text); }
}
```

Далее к этим свойствам можно будет обращаться как к переменным класса Form2.

4. Работа главного окна приложения после начала диалога с удаленным собеседником разделяется на два самостоятельных потока – во-первых, поток формирования сообщений для отсылки по сети, во-вторых, поток приема сообщений от удаленного клиента. Таким образом, окно Form1 должно содержать специальную переменную для хранения информации о втором потоке приложения (receiver). Кроме этого, класс должен содержать переменные для хранения параметров настройки (IP-адрес, номера удаленно-

го и локального портов, удаленная точка, имя пользователя приложения, сообщение, флаг отсутствия диалога):

```
// флаг отсутствия диалога
private bool done = true;
// IP-адрес удаленного компьютера
private IPAddress remoteAddress;
// номер локального порта
private int localPort;
// номер удаленного порта
private int remotePort;
// информация об удаленной точке
private IPEndPoint remoteEP;
// имя пользователя
private string name;
// сообщение
private string message;
// поток получения сообщений
private Thread receiver;
```

5. Разберем сначала, какие методы требуется разработать для работы потока получения сообщений. Проблема заключается в том, что этот поток, будучи вторым по приоритету (основной поток формирует и отсылает сообщения), не имеет права доступа к элементам пользовательского интерфейса, т.е. к элементам окна. Значит, нет возможности показать полученное сообщение пользователю непосредственно сразу после получения сообщения. Этим должна заниматься отдельный метод основного потока приложения, который вызывается из потока receiver с помощью специального метода Invoke. Таким образом, для обеспечения получения сообщений требуется написать два метода класса Form1 – для запуска цикла получения сообщений и для отображения полученного сообщения в окне:

```
// метод, реализующий поток получения сообщений
private void Listener()
{
      done = false;
      try
      {
        // пока приложение не закрыто, получаем
сообщения
          while (!done)
          {
```

```
                // осуществляется прослушивание
подключения
            // удаленных клиентов по протоколу UDP
            IPEndPoint ep = null;
                UdpClient client = new
UdpClient(localPort);
            // формирование массива байтов
полученного сообщения
                byte[] buffer = client.Receive(ref
ep);
                client.Close();
            // формирование строкового представления
            // полученного сообщения
                message =
Encoding.UTF8.GetString(buffer);
                // вызов метода главного потока
DisplayReceivedMessage
            // для показа полученного сообщения
            this.Invoke(new
    MethodInvoker(DisplayReceivedMessage));
                }
        }
        catch (Exception ex)
        {
             // вывод сообщения о возникшей ошибке
           MessageBox.Show(this, ex.Message, "Ошибка
    Listener",
                        MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
        }
}

// метод показа полученного сообщения пользователю
private void DisplayReceivedMessage()
{
    // получение строкового представления времени
получения сообщения
      string time = DateTime.Now.ToString("t");
    // textMessages - многострочное текстовое поле
показа
    // полученных сообщений. Добавление в него
нового сообщения,
```

```
        // показанного с новой строки, с указанием
    времени его получения
          textMessages.Text = time + " " + message +
    "\r\n" +
                                        textMessages.Text
                                        ;
          // показ в статусной строке окна времени
          // получения последнего сообщения
          statusBar.Text = "Последнее сообщение
    получено в " + time;
     }
```

6. Теперь обсудим последовательно все события и соответствующие методы-обработчики для основного потока.

Сначала пользователь настраивает **параметры подключения** с помощью контекстного меню. Таким образом, обрабатывается событие выбора пункта меню. В этом методе-обработчике должен вызываться диалог Form2 и после его закрытия должны сохраняться данные, которые введены в поля этого диалога в переменные класса Form1:

```
   // обработчик команды меню «Настройка»
   private void SetupToolStripMenuItem_Click(object
sender, EventArgs e)
   {
       // создание объекта диалогового окна Form2
       Form2 dlg = new Form2();
       // вызов диалога в модальном режиме
       if (dlg.ShowDialog() == DialogResult.OK)
       {
          // при выходе по нажатию кнопки OK сохранение
параметров
          // из свойств объекта dlg
            // IP-адрес удаленного компьютера
          remoteAddress =
    IPAddress.Parse(dlg.IPAdress);
            // номер локального порта
          localPort = dlg.LocalPort;
          // номер удаленного порта
            remotePort = dlg.Port;
       }
    }
```

Далее пользователь вводит свое имя и нажимает кнопку «Старт». Реакция на это событие - создание отдельного потока получения сообщений, для реализации которого был разработан метод Listener, который мы уже разобрали:

```
    // метод – обработчик начатия кнопки «Старт»
    private void buttonStart_Click(object sender,
EventArgs e)
    {
          // сохранение имени пользователя из
текстового поля textName
         name = textName.Text;
         // далее это текстовое поле становится
    недоступным для редактирования
          textName.ReadOnly = true;
          try
          {
            // создание отдельного потока работы метода
Listener
              receiver = new Thread(new
    ThreadStart(this.Listener));
                // поток должен быть «фоновым»
            receiver.IsBackground = true;
            // запуск потока приема сообщений
                receiver.Start();
                // становятся доступными кнопки «Стоп»,
«Отправить»,
            // а кнопка «Старт» становится недоступной
            buttonStart.Enabled = false;
                buttonStop.Enabled = true;
                buttonSend.Enabled = true;
           }
           catch (Exception ex)
           {
            // вывод сообщения о возникшей ошибке
                MessageBox.Show(this, ex.Message,
"Ошибка Start",
                             MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
           }
```

```
    }
```

Теперь можно формировать новые сообщения и отсылать их собеседнику. Для этого пользователь должен вводить сообщение в специальное текстовое поле и нажимать кнопку «Отправить». Метод-обработчик этого события должен формировать соединение с удаленным клиентом и отправлять сообщение:

```
  // метод-обработчик события нажатия кнопки
«Отправить»
  private void buttonSend_Click(object sender,
EventArgs e)
  {
        try
      {
         // формирование массива байтов сообщения для
       отправки
         // сообщение берется из текстового поля
       textMassage
         byte[] data = Encoding.UTF8.GetBytes
                           (name + ": " +
                      textMassage.Text);
         // формирование подключения с удаленной
    точкой
         UdpClient client = new UdpClient();
         remoteEP = new IPEndPoint(remoteAddress,
    remotePort);
         // отправка сообщения на удаленную точку
         client.Send(data, data.Length, remoteEP);
         // очистка поля сообщения и установка в него
    курсора
         // для формирования нового сообщения –
    перенос фокуса
         textMassage.Clear();
         textMassage.Focus();
      }
      catch (Exception ex)
      {
         // вывод сообщения о возникшей ошибке
         MessageBox.Show(this, ex.Message, "Ошибка
       Send",
```

```
                    MessageBoxButtons.OK,
                MessageBoxIcon.Error)
        }
    }
```

Окончание диалога с удаленным пользователем может возникнуть по двум причинам – пользователь нажал кнопку «Стоп» и пользователь закрыл приложение. В обоих случаях действия должны быть одинаковыми – формируется последнее сообщение для удаленного пользователя, говорящее о том, что пользователь отключился. Для отправки этого сообщения создадим собственный метод:

```
    // метод формирования и отсылки последнего сообщения
    private void StopListener()
    {
        // формирование массива байтов сообщения
        byte[] data = Encoding.UTF8.GetBytes(name + "
покинул чат");
        // подключение к удаленной точке
        UdpClient client = new UdpClient();
        remoteEP = new IPEndPoint(remoteAddress,
remotePort);
        // отсылка сообщения на удаленную точку
        client.Send(data, data.Length, remoteEP);
        done = true;
        // меняем доступность кнопок «Старт», «Стоп»,
«Отправить»
        buttonStart.Enabled = true;
        buttonStop.Enabled = false;
        buttonSend.Enabled = false;
    }
```

Вызываться этот метод будет в двух случаях – в методе-обработчике события нажатия кнопки «Стоп» и в методе-обработчике события закрытия окна:

```
// метод-обработчик события нажатия кнопки «Стоп»
private void buttonStop_Click(object sender, EventArgs
e)
{
    StopListener();
```

```
}

// метод-обработчик события закрытия окна формы (Close)
private void Form1_FormClosing(object sender,
FormClosingEventArgs e)
{
    // если кнопку «Стоп» не нажимали, сообщить
собеседнику об окончании сеанса
    if (!done)
          StopListener();
}
```

Задание к работе

1. Разработать два оконных приложения операционной системы MS Windows, которые реализовывали бы функции приложений из лабораторной работы № 1, но использующие в качестве базового протокол UDP.

2. Произвести тестирование приложения, составить отчет и защитить преподавателю.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером варианта, фамилией студента и группы.

2. Цель работы, задание.

3. Код разработанных приложений.

4. Снимки экрана, поясняющие работу приложений

5. Выводы о проделанной работе.

Контрольные вопросы

1. Высокоуровневые протоколы, базирующиеся на UDP.

2. Заголовок UDP.

3. Класс UdpClient.

4. UDP для передачи сообщений.

5. Широковещательная передача.

6. Достоинства и недостатки UDP.

Лабораторная работа 3

Передача файла по сети по протоколу UDP

Цель работы: получить навык разработки приложений на базе протокола UDP для передачи файлов.

Краткие сведения из теории

Передача файла характеризуется тем, что данные файла сначала нужно считать с диска в оперативную память. Только после этого можно передавать данные по сети. На удаленной точке принимаемый файл нужно будет сохранять. Для этого потребуется знать его размеры (чтобы создать буфер необходимого размера) и его тип, чтобы можно было корректно далее с ним работать. Поэтому целесообразно предварять отсылку файла передачей служебной информации, необходимой для его корректного сохранения. Соответственно, на удаленной точке сначала потребуется прочитать служебные данные о файле, а затем принять и сохранить передаваемый файл. Продемонстрируем этот подход на примере двух приложений передачи файла – первое передает файл, второе – принимает и сохраняет. Для передачи/приема служебной информации в обоих приложениях создадим одинаковый класс, объект которого можно сериализовать, и который хранит информацию о имени файла и его размере:

```
    // класс данных о файле
    [Serializable]
    public class FileDetails
    {
public string FILETYPE = "";  // расширение файла
public long FILESIZE = 0;     // размер файла
    }
```

Приложение по отсылке файла состоит из главной функции, в которой запрашивается имя файла, создается соединение для отсылки, вызывается функция отсылки сначала информации о файле, делается пауза, чтобы потоки не пересеклись, а далее вызывается функция отсылки самого файла:

```
//класс отсылки файла
    class Program
```

```
    {
//объект со служебной информацией о передаваемом файле
      private static FileDetails fileDet = new
    FileDetails();
      // IP-адрес удаленной точки
        private static IPAddress remoteIPAddress;
        // удаленный порт
        private const int remotePort = 5002;
        // объект-соединения
private static UdpClient sender = new UdpClient();
        // удаленная точка соединения
        private static IPEndPoint endPoint;
        // файловый поток передаваемого файла
        private static FileStream fs;
        static void Main(string[] args)
        {
            try
            {
             // ввод информации о соединении
Console.WriteLine("Введите удаленный IP-адрес");
remoteIPAddress = IPAddress.Parse(Console.ReadLine());
            // создание удаленного соединения
                endPoint = new
IPEndPoint(remoteIPAddress, remotePort);
Console.WriteLine("Введите полное имя файла");
// открытие файла для чтения - для последующей передачи
                fs = new
FileStream(@Console.ReadLine(),
                                FileMode.Open,
                            FileAccess.Read);
            // отсылка данных о файле
            SendFileInfo();

            // создание паузы
                Thread.Sleep(2000);
                // отсылка файла
                SendFile();
            }
            catch (Exception e)
            {
                Console.WriteLine("Исключение:" +
e.ToString());
            }
```

```
        }
        // функция отсылки данных о файле
      public static void SendFileInfo()
        {
// тип файла - это его расширение. Оно берется
// из последних трех симоволов имени файла.
//Выделение этих символов из строки осуществляется с
//помощью функции Substring с параметрами: первый
//параметр - индекс первого символа выделяемой
// подстроки, второй параметр - длина подстроки.
        fileDet.FILETYPE =
    fs.Name.Substring((int)fs.Name.Length - 3, 3);
// длина файла - длина открытого файлового потока
        fileDet.FILESIZE = fs.Length;
// сериализация данных класса FileDetails в поток
//MemoryStream, связанный с оперативной памятью
        XmlSerializer fileSer = new
    XmlSerializer(typeof(FileDetails));
            MemoryStream stream = new MemoryStream();
            fileSer.Serialize(stream, fileDet);
// установка позиции в потоке в его начало
            stream.Position = 0;
// получение массива байтов из сформированного потока в
//памяти выделение массива нужной длины
        byte[] bytes = new byte[stream.Length];
// чтение данных из этого потока
            stream.Read(bytes, 0,
Convert.ToInt32(stream.Length));
// отправка данных на удаленную точку
    Console.WriteLine("Отправка информации о файле");
            sender.Send(bytes, bytes.Length, endPoint);
        // закрытие потока, связанного с памятью
            stream.Close();
        }
// функция отсылки данных файла
        private static void SendFile()
        {
// выделение памяти под массив байтов, достаточный
// для хранения всего файла
            byte[] bytes = new byte[fs.Length];
// чтение данных в этот массив из файла
        fs.Read(bytes, 0, bytes.Length);
// отправка данных файла на удаленную точку
```

```
            Console.WriteLine("Отправка файла");
            try
            {
                sender.Send(bytes, bytes.Length,
endPoint);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
            finally
            {
// закрытие файлового потока и соединения
                fs.Close();
                sender.Close();
            }
            Console.WriteLine("Файл успешно
отправлен");
        }
    }
```

Приложение, принимающее файл, может иметь ту же структуру (три функции), но с обратными целями – принять информацию от удаленного приложения:

```
// класс приложения, принимающего файл по сети
class Program
{
    // объект для сохранения данных о файле
    private static FileDetails fileDet;
    // локальный порт, на который будет прислан файл
    private static int localPort = 5002;
    // соединение, инициированное удаленным клиентом
    private static UdpClient receivingUdpClient = new
    UdpClient(localPort);
    // удаленная точка
    private static IPEndPoint RemoveIpEndPoint = null;
    // файловый поток для сохранения файла
    private static FileStream fs;
    // массив байтов, которые будут получены по сети
    private static byte[] receiveBytes = new byte[0];
```

```
    static void Main(string[] args)
    {
        // получение информации о получаемом файле
        GetFileDetails();
        // получение самого файла
        ReceiveFile();
      }

    // функция получения данных о файле
    private static void GetFileDetails()
    {
        try
        {
            Console.WriteLine("Ожидание получения
        информации о файле");
// получение информации о файле в виде массива байтов
receiveBytes = receivingUdpClient.Receive (ref
RemoveIpEndPoint);
    Console.WriteLine("Получили информацию о файле");
            // запись полученных данных в поток,
            // связанный с оперативной памятью
              MemoryStream stream = new MemoryStream();
              stream.Write(receiveBytes, 0,
    receiveBytes.Length);
              // установка позиции в начало потока для
    последующих операций
            stream.Position = 0;
            // десериализация данных в объект из потока
            XmlSerializer fileSer = new
        XmlSerializer(typeof(FileDetails));
fileDet = (FileDetails)fileSer.Deserialize(stream);
        Console.WriteLine("Получен файл типа " +
    fileDet.FILETYPE + " размера " + fileDet.FILESIZE +
    " байт");
         }
         catch (Exception e)
         {
              Console.WriteLine(e.ToString());
         }
    }
    // функция получения файла
    public static void ReceiveFile()
    {
```

```
        try
        {
              Console.WriteLine("Ожидание получения
    файла");
              // получение массива байтов данных файла
receiveBytes = receivingUdpClient.Receive (ref
RemoveIpEndPoint);
        Console.WriteLine("Файл получен. Сохранение");
            // создание нового файла с именем temp и
            // запись в него полученных данных
              fs = new FileStream("temp." +
    fileDet.FILETYPE, FileMode.Create,
    FileAccess.ReadWrite,  FileShare.ReadWrite);
        fs.Write(receiveBytes, 0, receiveBytes.Length);
        fs.Close();

              Console.WriteLine("Файл сохранен");
              Console.WriteLine("Открытие
    соответствующей программой");
              // запуск приложения, которое
    ассоциировано с этим типом файла
            Process.Start(fs.Name);
        }
        catch (Exception e)
        {
              Console.WriteLine(e.ToString());
        }
        finally
        {
            // закрытие соединения с удаленной точкой
            receivingUdpClient.Close();
        }
    }
}
```

Задание к работе

1. Разработать два приложения ОС MS Windows: файл-сервер и клиент. Интерфейс приложений должен поддерживать изменение IP-адреса и порта для соединения, имени файла.

2. Произвести тестирование приложения, составить отчет и защитить преподавателю.

Таблица 2 – Варианты задания для лабораторной работы № 3

Вариант	Задание
1	Файл-сервер передает клиенту запрошенный файл. Клиент производит следующие вычисления и преобразования, выводит результирующую информацию и содержимое файла на экран. Удалить в тексте лишние пробелы. Лишними считаются те, которые идут непосредственно за пробелом. Подсчитать количество исправлений.
2	Файл-сервер передает клиенту запрошенный файл. Клиент производит следующие вычисления и преобразования, выводит результирующую информацию и содержимое файла на экран. Подсчитать количество встреч каждой из следующих букв: "а", "в", "и", "п" в базовом тексте.
3	Файл-сервер передает клиенту запрошенный файл. Клиент производит следующие вычисления и преобразования, выводит результирующую информацию и содержимое файла на экран. Подсчитать доли процентов встречи следующих букв: "е", "о", если суммарный процент встречаемости всех этих букв равен 100% или процент встречаемости е% + о% равен 100%.
4	Файл-сервер передает клиенту запрошенный файл. Клиент производит следующие вычисления и преобразования, выводит результирующую информацию и содержимое файла на экран. По правилам оформления машинописных текстов перед знаками .,!?:; пробелы не ставятся, но обязательно ставятся после этих знаков. Удалите лишние пробелы. Подсчитать количество исправлений.
5	Файл-сервер передает клиенту запрошенный файл. Клиент производит следующие вычисления и преобразования, выводит результирующую информацию и содержимое файла на экран. По правилам оформления машинописных текстов перед знаками .,!?:; пробелы не ставятся, но обязательно ставятся после этих знаков. Расставьте недостающие пробелы. Подсчитать количество исправлений.
6	Файл-сервер передает клиенту запрошенный файл. Клиент производит следующие вычисления и преобразования, выводит резуль-

	тирующую информацию и содержимое файла на экран. Найти из исходного текста второе предложение и вернуть его в переменную Perem, а также вывести на экран весь исходный текст и найденное предложение.
7	Файл-сервер передает клиенту запрошенный файл. Клиент производит следующие вычисления и преобразования, выводит результирующую информацию и содержимое файла на экран. Удалить из базового текста 2, 4, 6, 8 слова.
8	Файл-сервер передает клиенту запрошенный файл. Клиент производит следующие вычисления и преобразования, выводит результирующую информацию и содержимое файла на экран. Удалить из базового текста 3, 5, 7, 10 слова.
9	Файл-сервер передает клиенту запрошенный файл. Клиент производит следующие вычисления и преобразования, выводит результирующую информацию и содержимое файла на экран.
10	Файл-сервер передает клиенту запрошенный файл. Клиент производит следующие вычисления и преобразования, выводит результирующую информацию и содержимое файла на экран. Вставить в базовый текст вместо букв «а» - «АА».
11	Файл-сервер передает клиенту запрошенный файл. Клиент производит следующие вычисления и преобразования, выводит результирующую информацию и содержимое файла на экран. Вставить в базовый текст вместо букв «е» и «о» - «ББ».
12	Файл-сервер передает клиенту запрошенный файл. Клиент производит следующие вычисления и преобразования, выводит результирующую информацию и содержимое файла на экран. Поменять местами первое и последнее слова в базовом тексте.
13	Файл-сервер передает клиенту запрошенный файл. Клиент производит следующие вычисления и преобразования, выводит результирующую информацию и содержимое файла на экран. Заменить в пятой строке файла все строчные буквы прописными, а прописные заменить строчными.
14	Файл-сервер передает клиенту запрошенный файл. Клиент производит следующие вычисления и преобразования, выводит результирующую информацию и содержимое файла на экран. Заменить цифры во второй строке файла словами.

15	Файл-сервер передает клиенту запрошенный файл. Клиент производит следующие вычисления и преобразования, выводит результирующую информацию и содержимое файла на экран. Найти слово в файле с наибольшим количеством букв и вывести его на экран.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером варианта, фамилией студента и группы.
2. Цель работы, задание.
3. Код разработанных приложений.
4. Снимки экрана, поясняющие работу приложений
5. Выводы о проделанной работе.

Контрольные вопросы

1. Сериализация.
2. Функции файл-сервера.
3. Методы для отправки и приема файла.
4. Преимущества и недостатки протокола UDP для файлового обмена.

Лабораторная работа 4

Отправка сообщений электронной почты

Цель работы: получить навык разработки приложения для обмена сообщениями с почтовым сервером

Краткие теоретические сведения

В языке C# имеются средства для приема и отправки сообщений электронной почты. Они находятся среди классов пространства имен System.Web.Mail. Аналогичные средства можно найти в другом пространстве имен System.Net.Mail. Многие приложения удаленного взаимодействия пользователей предполагают отсылку сообщений электронной почты клиентам. Разберем программный код, который реализует эту задачу. Приложение

должно сформировать сообщение (объект класса MailMessage), которое помимо текста может включать список файлов-вложений. Требуется указать email-ы источника и адресата, тему сообщения. Далее через свойства класса SmtpMail сообщение отправляют адресату. Многие сервера отправки почты (smtp-сервера) требуют авторизации на сервере, чтобы можно было производить отправку сообщений. Задаются эти параметры с помощью полей сообщения (Fields) - имени пользователя, пароля и порта отправки сообщений. Таким образом, для отправки сообщения электронной почты требуется выполнить следующую последовательность действий:

```
// создание объекта-сообщения электронной почты
MailMessage email = new MailMessage();
// указание адреса отправителя
email.From = "myaddress@mail.ru";
// указание адреса получателя письма
email.To = "myclient@mail.ru";
// указание темы письма
email.Subject = "Тестовое послание";
// указание формата письма – может быть в виде текста
или html-разметки
email.BodyFormat = MailFormat.Text;
// задание текста сообщения
email.Body = "Это просто текстовое сообщение";
// добавление файла к письму
email.Attachments.Add(new MailAttachment("C:\\1.doc",
MailEncoding.Base64));

// задание свойств письма для подключения к smtp-
серверу
// указание требования аутентификации пользователя
email.Fields.Add
    ("http://schemas.microsoft.com/cdo/configuration/sm
tpauthenticate", "1");
// задание имени пользователя
email.Fields.Add
        ("http://schemas.microsoft.com/cdo/configuratio
    n/sendusername", "myaddress@mail.ru ");
// задание пароля пользователя
email.Fields.Add
    ("http://schemas.microsoft.com/cdo/configuration/se
ndpassword", "myparol");
```

```
// задание порта для подключения к smtp-серверу
email.Fields.Add
    ("http://schemas.microsoft.com/cdo/configuration/sm
tpserverport", "2525");

// задание адреса smtp-сервера
SmtpMail.SmtpServer = "smtp.mail.ru";

// отправка электронного письма
SmtpMail.Send(email);
```

Задание к работе

1. Разработать приложение ОС MS Windows, которое отправляет сообщение-письмо почтовому серверу и демонстрирует выполнение одной из команд SMTP.

Таблица 3 – Варианты задания для лабораторной работы № 4

Вариант	Задание
1	HELO. Идентифицирует модуль-передатчик для модуля-приемника (hello).
2	MAIL. Начинает почтовую транзакцию, которая завершается передачей данных в один или несколько почтовых ящиков (mail).
3	RCPT. Идентифицирует получателя почтового сообщения (recipient).
4	DATA. Строки, следующие за этой командой, рассматриваются получателем как данные почтового сообщения. В случае SMTP, почтовое сообщение заканчивается комбинацией символов: CRLF-точка-CRLF.
5	RSET. Прерывает текущую почтовую транзакцию (reset).
6	NOOP. Требует от получателя не предпринимать никаких действий, а только выдать ответ ОК. Используется главным образом для тестирования.(No operation).
7	QUIT. Требует выдать ответ ОК и закрыть текущее соединение.
8	VRFY. Требует от приемника подтвердить, что ее аргумент является действительным именем пользователя. (См. примечание.).

9	SEND. Начинает почтовую транзакцию, доставляющую данные на один или несколько терминалов (а не в почтовый ящик).
10	SOML. Начинает транзакцию MAIL или SEND, доставляющую данные на один или несколько терминалов или в почтовые ящики.
11	SAML. Начинает транзакцию MAIL и SEND, доставляющие данные на один или несколько терминалов и в почтовые ящики.
12	EXPN. Команда SMTP-приемнику подтвердить, действительно ли аргумент является адресом почтовой рассылки и если да, вернуть адрес получателя сообщения (expand).
13	HELP. Команда SMTP-приемнику вернуть сообщение-справку о его командах.
14	TURN. Команда SMTP-приемнику либо сказать ОК и поменяться ролями, то есть стать STMP- передатчиком, либо послать сообщение-отказ и остаться в роли SMTP-приемника.

2. Произвести тестирование приложения, составить отчет и защитить преподавателю.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером варианта, фамилией студента и группы.

2. Цель работы, задание.

3. Код разработанных приложений.

4. Снимки экрана, поясняющие работу приложений

5. Выводы о проделанной работе.

Контрольные вопросы

1. Как работает электронная почта?

2. Протокол SMTP.

3. SMTP-команды.

4. Коды ответов на SMTP-команды.

5. Обязательная и дополнительная информация сообщения электронной почты.

6. Заголовок сообщения электронной почты.

7. Протокол POP3.

8. Протокол IMAP.

9. Протокол NNTP.

Лабораторная работа 5

Использование криптографии для обеспечения безопасности передачи данных по сети

Цель работы: получить навык разработки приложений для обмена сообщениями, защищенного средствами .NET.

Краткие теоретические сведения

Любое сетевое взаимодействие, особенно для приложений электронной коммерции, обязано обеспечить безопасность передаваемых данных. Самым распространенным средством для решения этой задачи является использование криптографии. В языке C# для этих целей предусмотрено пространство имен System.Security.Criptography.

Среди средств криптографии особенно популярны:

- хэширование;
- симметричное шифрование (DES, RS2, Triple-DES, Rijndaen);
- асимметричное шифрование (RSA, DSA).

Хэширование предполагает шифрование «в одну сторону» без расшифровки. Его часто применяют тогда, когда на удаленной части требуется сохранять данные пользователя. Например, в базе данных на сервере сохраняются данные о паролях клиентов. Хранение явных представлений паролей в базе данных небезопасно. Поэтому на стороне клиента перед отправкой пароль хэшируется и на сервер посылается и сохраняется в базе данных хэш-представление пароля пользователя. Про алгоритм хэширования «знает» только приложение-клиент, поэтому знание хэш-представления ничего не дает злоумышленнику, который получил доступ к базе данных.

Разберем два приема хэширования данных.

1. Хэширование без ключа. Существует несколько алгоритмов хэширования без ключа, например, алгоритм M5. Для хэширования требуется

создать специальный провайдер (объект типа MD5CryptoServiceProvider), затем хэшировать набор байтов сообщения, после придется преобразовать в строку и добавить символ конца строки и далее отправить другому приложению, записав данные в поток соответствующего сокета.

```
// читаем сообщение с клавиатуры
string dataToSend=Console.ReadLine();
  // получаем массив байтов сообщения
byte [] data=Encoding.UTF8.GetBytes(dataToSend);
// создание провайдера для хэширования
MD5CryptoServiceProvider md5Provider = new
MD5CryptoServiceProvider();
  // получение хэшированного массива байтов
byte[] hashResult = md5Provider.ComputeHash(data);
  // получение строки из хэшированных байтов
dataToSend = Encoding.UTF8.GetString(hashResult);
  // добавление символов конца строки перед
оправкой на другому приложению
dataToSend+="\r\n";
// получение байтов хэшированного сообщения для
отправки
  data = Encoding.UTF8.GetBytes(dataToSend);
// отправка данных
ws.Write(data,0,data.Length);
```

2. Хэширование с ключом. Для хэширования используется ключ, который определяется пользователем. Это становится дополнительной мерой безопасности, поскольку одним и тем же алгоритмом будет по-разному зашифровано одно и то же сообщение в зависимости от используемого ключа. Пользователь должен только постоянно помнить этот ключ. Хэширование с ключом происходит с помощью специального криптопотока, запись в который и означает проведение хэширования.

```
// читаем ключ пользователя с клавиатуры
string keyStr=Console.ReadLine();
  // получаем массив байтов ключа
byte [] key=Encoding.UTF8.GetBytes(keyStr);

// читаем сообщение с клавиатуры
string dataToSend=Console.ReadLine();
  // получаем массив байтов сообщения
```

```
    byte [] data=Encoding.UTF8.GetBytes(dataToSend);

    // создание объекта хэширования с указанием ключа
    HMACSHA1 hmac = new HMACSHA1(key);
    // создание криптопотока. Второй параметр указывает
объект хэширования
    CryptoStream cs = new CryptoStream(Stream.Null,
hmac, CryptoStreamMode.Write);
    // запись сообщения в криптопоток – хэширование
сообщения
    cs.Write(data, 0, data.Length);
    cs.Close();

    // получение хэшированного сообщения для отсылки –
хэшированный набор байтов
    // получаем через свойство Hash объекта хэширования
    dataToSend = Encoding.UTF8.GetString(hmac.Hash);
    // добавляем символы конца строки
    dataToSend += "\r\n";
    // получение байтов хэшированного сообщения для
отправки
      data = Encoding.UTF8.GetBytes(dataToSend);
    // отправка данных
    ws.Write(data,0,data.Length);
```

В качестве примера использования шифрования переведем приложение «эхо-сервера» с использованием асимметричного шифрования (симметричное шифрование применяется аналогично, даже проще за счет использования только одного ключа). Будем применять для шифрования алгоритм RSA. Схема работы приложений будет следующей. Клиент, подключаясь к серверу, получает от последнего открытый ключ, с помощью которого можно будет зашифровывать информацию. Провайдер RSA имеет специальные средства для сохранения и загрузки ключевых параметров в xml-формат (ToXmlString, FromXmlString). На стороне клиента сообщения шифруются далее с помощью полученного ключа (метод Encrypt) и отправляются на сервер. Сервер же полученные сообщения дешифрует (метод Decrypt) и использует далее в своей работе.

Приведем далее программный код приложений клиента и сервера, выделив строки, добавленные для применения шифрования.

Приложение-клиент:

```
using System;
using System.Collections.Generic;
using System.Net;
using System.IO;
using System.Net.Sockets;
using System.Text;
using System.Security.Cryptography;

namespace Client
{
    // пример приложения эхо-клиента
    class Program
    {
        const int ECHO_PORT = 8080;
        public static void Main(string[] args)
        {
            bool f = true;
          Console.Write("Имя:");
            string UserName = Console.ReadLine();
            Console.WriteLine("Протокол работы:");

            try
            {
                // соединение с сервером и отсылка
имени пользователя
             TcpClient eClient = new
         TcpClient("127.0.0.1",ECHO_PORT);
                StreamReader rs = new
StreamReader(eClient.GetStream());
                NetworkStream ws = eClient.GetStream();
                string dataToSend=UserName+"\r\n";
                byte[] data =
Encoding.UTF8.GetBytes(dataToSend);
                ws.Write(data,0,data.Length);

                // создание провайдера для шифрования
            RSACryptoServiceProvider rsa = new
         RSACryptoServiceProvider();

                // прием открытого ключа
                string str_key = rs.ReadLine();
```

```
                rsa.FromXmlString(str_key);

            // цикл обработки сообщений
                while(f)
                {
                    // ввод нового сообщения
                  Console.Write(UserName+":");
                    dataToSend=Console.ReadLine();
                  if(dataToSend.IndexOf("QUIT")>-1)
                        f=false;  // после отправки
сообщения - выхд из цикла

data=Encoding.UTF8.GetBytes(dataToSend);

                  // шифрование сообщения - набора
байтов сообщения
                    byte[]
result=rsa.Encrypt(data,false);
                // получение строкого представления
шифрованного сообщения
                // формат сообщения должен быть Base64
                    dataToSend =
Convert.ToBase64String(result);

                // добавление символов конца строки и
отсылка
                    dataToSend += "\r\n";
                    data =
Encoding.UTF8.GetBytes(dataToSend);
                    ws.Write(data,0,data.Length);

                // получение ответа от сервера
                    if (f)
                {
                   string returnData=rs.ReadLine();
                      Console.WriteLine("Сервер:
"+returnData);
                }
                 }
                 eClient.Close();
            }
            catch(Exception exp)
            {
```

```
                Console.WriteLine("Исключение:" + exp);
            }
        }
    }
}
```

В приложении-сервере изменения нужно внести только в класс, обслуживающий диалог с отдельным клиентом. Поэтому приведем здесь только метод RunClient этого класса, который этот диалог и осуществляет. Также выделим строки, добавленные для обеспечения шифрования.

```
// метод класса ClientHandler приложения-сервера
public void RunClient()
{
      // формирование потоков ввода и вывода и
получение имени клиента
    StreamReader rs = new
StreamReader(clSocket.GetStream());
      NetworkStream ws = clSocket.GetStream();
      string returnData = rs.ReadLine();
      string userName = returnData;
      Console.WriteLine(userName + " на сервере");

      // создание объекта-провайдера для шифрования
      RSACryptoServiceProvider rsa = new
RSACryptoServiceProvider();

    // получение ключа и его Xml-представления
    string key = rsa.ToXmlString(false);
      // отправка открытого ключа клиенту
    key += "\r\n";
      byte [] data = Encoding.UTF8.GetBytes(key);
      ws.Write(data, 0, data.Length);

    // цикл приема и отсылки сообщений
      while (true)
      {
           // получение сообщения от клиента
         returnData = rs.ReadLine();
           Console.WriteLine(returnData);

           // расшифровка полученного сообщения
           // получение байтов строки
```

```
            byte[] d =
Convert.FromBase64String(returnData);
          // расшифровка байтов
            byte[] res = rsa.Decrypt(d, false);
            // получение строки сообщения
          returnData = Encoding.UTF8.GetString(res);

            // выход, если клиет отключился
          if (returnData.IndexOf("QUIT") > -1)
            {
                 Console.WriteLine("До свидания,
"+userName);
                 break;
            }
            // вывод полученного сообщения и отправка
его «эхом» клиенту
          Console.WriteLine(userName + ": " +
    returnData);
            returnData += "\r\n";
            byte[] dataWrite =
Encoding.UTF8.GetBytes(returnData);
            ws.Write(dataWrite, 0, dataWrite.Length);
       }
       clSocket.Close();
   }
```

Задание к работе

1. Разработать два приложения ОС MS Windows (сервер и клиент), которые обмениваются информацией, используя для безопасности средства защиты информации, соответствующие варианту. Интерфейс приложений должен поддерживать изменение IP-адреса и порта для соединения, сообщения, запросы и информации, требуемой для протокола защиты данных.

2. Произвести тестирование приложения, составить отчет и защитить преподавателю.

Таблица 4 – Варианты задания для лабораторной работы № 5

Вариант	Протокол обмена сообщениями	Протокол шифрования	Хэш-функция для аутентификации

1	TCP	DES	HMACSHA-1
2	UDP	TripleDES	MACTripleDES
3	TCP	RC2	MD5
4	UDP	Rijndael	SHA-1
5	TCP	DSA/DSS	SHA-256
6	UDP	RSA	SHA-384
7	TCP	TripleDES	SHA-512
8	UDP	RC2	HMACSHA-1
9	TCP	Rijndael	MACTripleDES
10	UDP	DES	MD5
11	TCP	TripleDES	SHA-1
12	UDP	RC2	SHA-256
13	TCP	Rijndael	SHA-384
14	UDP	DSA/DSS	SHA-512
15	TCP	RSA	HMACSHA-1

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером варианта, фамилией студента и группы.

2. Цель работы, задание.

3. Код разработанных приложений.

4. Снимки экрана, поясняющие работу приложений

5. Выводы о проделанной работе.

Контрольные вопросы

1. Конфиденциальность, аутентификация, целостность, строгое выполнение обязательств.

2. Симметричные алгоритмы.

3. Асимметричные алгоритмы.

4. Алгоритмы хеширования или дайджеста сообщения.

5. Цифровая подпись.

6. Блочные и поточные шифры.

7. Иерархия криптографических классов.

8. Сертификат X509.

ЛИТЕРАТУРА

1. Пятибратов А.П. Вычислительные системы, сети и телекоммуникации. – М.: Финансы и статистика, 2005.

2. Головин Ю. А. Информационные сети:учебник.- М.: Академия, 2013.

3. Олифер, В. Г. Компьютерные сети. Принципы, технологии, протоколы : учебное.- М ; Спб. : Питер, 2009.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к выполнению лабораторных работ по дисциплине
«Инфокоммуникационные системы»
для студентов направления
15.03.04 Автоматизация технологических процессов и производств

Составители: А.А. Евдокимов