Практическое задание № 3. Сокеты (Sockets)

Теория

Материал составлен на основе ресурсов: Джок А. Сокеты. [Электронный ресурс]. URL: http://www.osp.ru/cw/2002/03/48355 (дата обращения: 15.01.2014); Передача данных с использованием сокетов. [Электронный ресурс]. URL: http://staff.science.uva.nl/~rshulako/manuals/java/vol12/ch5.html (дата обращения: 15.01.2014).

Сокеты – средстве межпроцессной коммуникации между клиентской и серверной программами через локальные или глобальные сети, а также между различными процессами на одном компьютере.

Сокеты обеспечивают двухстороннюю связь типа «точка-точка» между двумя процессами. Они являются основными компонентами межсистемной и межпроцессной связи. Каждый сокет представляет собой конечную точку связи. Он имеет определенный тип, и один или нескольких связанных с ним процессов.

Сокеты, созданные в 80-х годах в Калифорнийском университете в Беркли, впервые явились миру Unix в обличье интерфейса Berkeley Sockets Interface. Это был программный механизм, созданный с целью помочь мощным, объединенным в сеть компьютерам организовать эффективный обмен информацией.

К середине 90-х годов корпорация Microsoft разработала собственный вариант сокетов – Windows Sockets (сегодня чаще называемые WinSock), с помощью которых Windows-приложения могли взаимодействовать по сетевым соединениям.

Программисты используют понятия «клиент» и «сервер», чтобы подчеркнуть отличия между компьютером, выполняющим «звонок», и компьютером, к которому этот звонок обращен. Компьютеры с серверными сокетами поддерживают открытым коммуникационный порт, всегда оставаясь готовыми к неожиданному входящему вызову.

Клиентский сокет на одном компьютере использует предопределенный сетевой адрес, для того чтобы связаться с серверным сокетом на другом компьютере. После того как соответствующие сокеты установили диалог между собой, два компьютера смогут начать обмениваться данными и услугами.

В момент соединения на сервере происходит небольшая суматоха. Вместо того чтобы продолжать сеанс связи, используя исходный порт, сервер переключает диалог на «вспомогательный» порт, чтобы освободить основную линию для других клиентов, которые, возможно, попытаются обратиться к серверу.

Прежде чем приложение сможет выполнять передачу или прием данных, оно должно создать сокет, указав при этом адрес узла IP, номер порта, через который будут передаваться данные, и тип сокета.

IP-адрес используется для идентификации компьютера, на котором запущено приложение, номер порта – для идентификации приложения в рамках данного компьютера.

Что касается типов сокетов, то их два – потоковые и датаграммные.

С помощью потоковых сокетов можно создавать каналы передачи данных между двумя приложениями. Потоковые сокеты позволяют передавать данные только между двумя приложениями, так как они предполагают создание канала между этими приложениями. Однако иногда нужно обеспечить взаимодействие нескольких клиентских приложений с одним серверным или нескольких клиентских приложений с несколькими серверными приложениями. В этом случае вы можете либо создавать в серверном приложении отдельные задачи и отдельные каналы для каждого клиентского приложения, либо воспользоваться датаграммными сокетами. Последние позволяют передавать данные сразу всем узлам сети, хотя такая возможность редко используется и часто блокируется администраторами сети.

Для передачи данных через датаграммные сокеты не нужно создавать канал – данные посылаются непосредственно тому приложению, для которого они предназначены с использованием адреса этого приложения в виде сокета и номера порта. При этом одно клиентское приложение может обмениваться данными с несколькими серверными приложениями или наоборот, одно серверное приложение - с несколькими клиентскими.

К сожалению, датаграммные сокеты не гарантируют доставку передаваемых пакетов данных. Даже если пакеты данных, передаваемые через такие сокеты, дошли до адресата, не гарантируется, что они будут получены в той же самой последовательности, в которой были переданы. Потоковые сокеты, напротив, гарантируют доставку пакетов данных, причем в правильной последовательности.

Причина отстутствия гарантии доставки данных при использовании датаграммных сокетов заключается в использовании такими сокетами протокола UDP, который, в свою очередь, основан на протоколе с негарантированной доставкой IP. Потоковые сокеты работают через протокол гарантированной доставки TCP.

Функции для работы с сокетами

1. Для создания сервером сокета используется конструктор класса TcpListener:

```
TcpListener Listener = new TcpListener(IPAddress IP, int Port), где
     Listener – ссылка на серверный сокет (Listener для приема заявок от клиентских
     сокетов);
     IP – IP-адрес, который используется для идентификации машины серверного
     приложения;
     Port – номер порта, который используется для идентификации серверного приложения
     в рамках конкретной машины.
```

2. Выбор клиентского сокета из очереди ожидающих обслуживания клиентов выполняется с помощью метода AcceptSocket класса TcpListener:

```
Socket ClientSock = Listener.AcceptSocket(), где ClientSock – ссылка на клиентский
сокет.
```

3. Для чтения данных из серверного сокета используется метод Receive класса Socket:

```
ClientSock.Receive(byte[] buff), где buff – буфер, в который производится запись.
```

4. Отключение серверного процесса от клиентского сокета производится с помощью метода Close класса Socket:

```
ClientSock.Close().
```

5. Завершение «прослушивания» серверного сокета выполняется с помощью метода Stop класса TcpListener:

```
Listener.Stop().
```

6. Для создания сокета клиентским процессом используется метод Connect класса TcpClient:

```
Client.Connect(IPAddress IP, int Port), где
     IP – IP-адрес машины серверного приложения;
     Port – номер порта, через который производится обмен сообщениями.
```

7. Для получения файлового потока клиентского сокета используется метод GetStream класса TcpClient:

```
Stream stm = Client.GetStream(), где stm – файловый поток.
```

8. Для записи последовательности байт в сокет используется метод Write класса Stream:

```
stm.Write(byte[] buff, int start, int end), где
     buff – буфер, из которого производится запись;
     start – номер байта, с которого выполняется запись;
     end – номер байта, до которого выполняется запись.
```

9. Для закрытия сокета клиентским процессом используется метод Close класса TcpClient:

```
Client.Close().
```

Алгоритм обмена сообщениями с помощью сокетов

Обмен сообщениями с использованием сокетов выполняется по следующему алгоритму:

1. Сервер создает сокет с помощью конструктора класса TcpListener и приступает к его прослушиванию.
2. Клиент подключается к сокету сервера с помощью метода Connect класса TcpClient.
3. Сервер выбирает из очереди первый клиентский сокет, ожидающий обслуживания, с помощью метода AcceptSocket класса TcpListener.
4. Клиент получает файловый поток сокета с помощью метода GetStream класса TcpClient.
5. Клиент записывает с помощью файлового потока сообщения в канал сервера с помощью метода Write класса Stream.
6. Сервер считывает сообщения из сокета с помощью метода Receive класса Socket.
7. Пока у клиента есть сообщения, выполняется последовательность шагов 5-6. Иначе выполняется переход к шагу 8.
8. Клиент отключается от сокета сервера с помощью метода Close класса TcpClient.
9. Если еще остались клиенты на обслуживание, то выполняется переход к шагу 3, иначе – к шагу 10.
10. Сервер отключается от сокета клиента с помощью метода Close класса Socket.
11. Сервер приостанавливает работу сокета с помощью метода Stop класса TcpListener.

Разработка приложения «Чат v 1.0»

Разработаем приложение, которое позволяет с помощью сокетов отправлять серверу сообщения сразу от нескольких клиентов, необязательно расположенных на одном узле сети. Сервер представляет собой форму, на которой отображаются все полученные им сообщения (см. рис. 1).

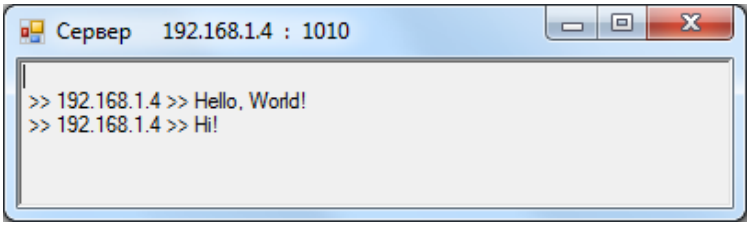

Рис. 1. Окно сервера при коммуникации с помощью сокетов

Клиент представляет собой форму, содержащую два поля ввода и две кнопки (см. рис. 2).

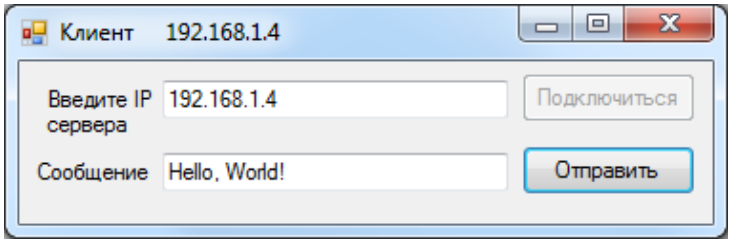

Рис. 2. Окно клиента при коммуникации с помощью сокетов

Первое поле предназначено для ввода IP-адреса сервера. При нажатии на кнопку «Подключиться» клиент производит подключение к сокету сервера. Второе поле предназначено для ввода сообщения для отправки. При нажатии на кнопку «Отправить» сообщение, в котором указан IP-адрес компьютера клиента (для его идентификации) и текст сообщения, введенный пользователем, будет отправлено серверу.

Код реализации сервера с комментариями приведен ниже:

```
    public partial class frmMain : Form
    {
        private Socket ClientSock;                      // клиентский сокет
        private TcpListener Listener;                   // сокет сервера
        private List<Thread> Threads = new List<Thread>();      // список потоков приложения
(кроме родительского)
        private bool _continue = true;                          // флаг, указывающий
продолжается ли работа с сокетами

        // конструктор формы
        public frmMain()
        {
            InitializeComponent();

            IPHostEntry hostEntry = Dns.GetHostEntry(Dns.GetHostName());    // информация об
IP-адресах и имени машины, на которой запущено приложение
            IPAddress IP = hostEntry.AddressList[0];                        // IP-адрес,
который будет указан при создании сокета
            int Port = 1010;                                                // порт, который
будет указан при создании сокета

            // определяем IP-адрес машины в формате IPv4
            foreach (IPAddress address in hostEntry.AddressList)
                if (address.AddressFamily == AddressFamily.InterNetwork)
                {
                    IP = address;
                    break;
                }

            // вывод IP-адреса машины и номера порта в заголовок формы, чтобы можно было его
использовать для ввода имени в форме клиента, запущенного на другом вычислительном узле
            this.Text += "     " + IP.ToString() + "  :  " + Port.ToString();

            // создаем серверный сокет (Listener для приема заявок от клиентских сокетов)
            Listener = new TcpListener(IP, Port);
            Listener.Start();

            // создаем и запускаем поток, выполняющий обслуживание серверного сокета
            Threads.Clear();
            Threads.Add(new Thread(ReceiveMessage));
            Threads[Threads.Count-1].Start();
        }

        // работа с клиентскими сокетами
        private void ReceiveMessage()
        {
            // входим в бесконечный цикл для работы с клиентскими сокетом
            while (_continue)
            {
                ClientSock = Listener.AcceptSocket();           // получаем ссылку на
очередной клиентский сокет
                Threads.Add(new Thread(ReadMessages));          // создаем и запускаем поток,
обслуживающий конкретный клиентский сокет
                Threads[Threads.Count - 1].Start(ClientSock);
            }
        }

        // получение сообщений от конкретного клиента
        private void ReadMessages(object ClientSock)
        {
```

```
            string msg = "";        // полученное сообщение

            // входим в бесконечный цикл для работы с клиентским сокетом
            while (_continue)
            {
                byte[] buff = new byte[1024];                           // буфер прочитанных
из сокета байтов
                ((Socket)ClientSock).Receive(buff);                     // получаем
последовательность байтов из сокета в буфер buff
                msg = System.Text.Encoding.Unicode.GetString(buff);     // выполняем
преобразование байтов в последовательность символов

                rtbMessages.Invoke((MethodInvoker)delegate
                {
                    if (msg.Replace("\0","") != "")
                        rtbMessages.Text += "\n >> " + msg;             // выводим полученное
сообщение на форму
                });
                Thread.Sleep(500);
            }
        }

        private void frmMain_FormClosing(object sender, FormClosingEventArgs e)
        {
            _continue = false;      // сообщаем, что работа с сокетами завершена

            // завершаем все потоки
            foreach (Thread t in Threads)
            {
                t.Abort();
                t.Join(500);
            }

            // закрываем клиентский сокет
            if (ClientSock != null)
                ClientSock.Close();

            // приостанавливаем "прослушивание" серверного сокета
            if (Listener != null)
                Listener.Stop();
        }
    }
```

Код реализации клиента с комментариями приведен ниже:

```
    public partial class frmMain : Form
    {
        private TcpClient Client = new TcpClient();     // клиентский сокет
        private IPAddress IP;                           // IP-адрес клиента

        // конструктор формы
        public frmMain()
        {
            InitializeComponent();

            IPHostEntry hostEntry = Dns.GetHostEntry(Dns.GetHostName());    // информация об
IP-адресах и имени машины, на которой запущено приложение
            IP = hostEntry.AddressList[0];                                  // IP-адрес,
который будет указан в заголовке окна для идентификации клиента

            // определяем IP-адрес машины в формате IPv4
            foreach (IPAddress address in hostEntry.AddressList)
                if (address.AddressFamily == AddressFamily.InterNetwork)
                {
                    IP = address;
                    break;
                }
        }
```

```
        // подключение к серверному сокету
        private void btnConnect_Click(object sender, EventArgs e)
        {
            try
            {
                int Port = 1010;                                    // номер порта, через который
выполняется обмен сообщениями
                IPAddress IP = IPAddress.Parse(tbIP.Text);          // разбор IP-адреса сервера,
указанного в поле tbIP
                Client.Connect(IP, Port);                           // подключение к серверному
сокету
                btnConnect.Enabled = false;
                btnSend.Enabled = true;
            }
            catch
            {
                MessageBox.Show("Введен некорректный IP-адрес");
            }
        }

        // отправка сообщения
        private void btnSend_Click(object sender, EventArgs e)
        {
            byte[] buff = Encoding.Unicode.GetBytes(IP.ToString() + " >> " + tbMessage.Text);
// выполняем преобразование сообщения (вместе с идентификатором машины) в последовательность
байт
            Stream stm = Client.GetStream();
// получаем файловый поток клиентского сокета
            stm.Write(buff, 0, buff.Length);
// выполняем запись последовательности байт
        }

        private void frmMain_FormClosing(object sender, FormClosingEventArgs e)
        {
            Client.Close();         // закрытие клиентского сокета
        }
    }
```

Задание

1. Изучите разработанное приложение. Запустите исполняемый файл сервера и клиента на одной машине, затем перенесите один из компонентов распределенного приложения на другой вычислительный узел (лучше на ноутбук, т.к. администратор ЛВС вуза может запретить обмен сообщения между различными узлами ЛВС) и попробуйте его запустить. Запустите несколько клиентов и попробуйте отправить сообщения одному серверу.
2. Запустите несколько серверов на одной машине. Отправьте им сообщения от нескольких клиентов. Объясните, почему приложение перестало работать.
3. Модифицируйте приложение так, чтобы существовала возможность на сервере идентифицировать клиентов не по IP-адресу вычислительного узла, а по нику/логину пользователя.
4. Модифицируйте приложение так, чтобы получился полноценный чат. Клиент может отправлять сообщения всем клиентам, участвующим в беседе. Для этого каждый клиент должен иметь возможность просмотра всех сообщений от всех клиентов, а сервер должен содержать список клиентов, которые хотят участвовать в беседе, чтобы каждый раз выполнять им рассылку сообщений.
5. Изучите самостоятельно, как работать с UDP-сокетами. Модифицируйте приложение так, чтобы клиенты, желающие принять участие в беседе, находили сервер в сети с помощью широковещательного запроса (UDP-пакета), а не через ввод IP-адреса машины, на которой расположен сервер, в форме клиента.