ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)

Кафедра вычислительных машин, комплексов, систем и сетей

Н.И. Черкасова

ПРОГРАММИРОВАНИЕ СЕТЕВЫХ ПРИЛОЖЕНИЙ

Учебно-методическое пособие
по выполнению лабораторных работ № 6, 7

для студентов IV курса
направления 09.03.01
очной формы обучения

Москва
ИД Академии Жуковского
2020

УДК 004.7.057:004.42
ББК 6Ф7.3
Ч-48

Рецензент:
Романчева Н.И. – канд. техн. наук, доцент

Черкасова Н.И.

Ч-48 Программирование сетевых приложений [Текст] : учебно-методическое пособие по выполнению лабораторных работ № 6, 7 / Н.И. Черкасова. – М.: ИД Академии Жуковского, 2020. – 32 с.

Данное учебно-методическое пособие издается в соответствии с рабочей программой учебной дисциплины «Программирование сетевых приложений» по учебному плану для студентов IV курса направления 09.03.01 очной формы обучения.

Рассмотрено и одобрено на заседаниях кафедры 29.08.2020 г. и методического совета 29.08.2020 г.

УДК 004.7.057:004.42
ББК 6Ф7.3

В авторской редакции

Подписано в печать 14.12.2020 г.
Формат 60x84/16 Печ. л. 2 Усл. печ. л. 1,86
Заказ № 706/1008-УМП16 Тираж 90 экз.

Московский государственный технический университет ГА
125993, Москва, Кронштадтский бульвар, д. 20

Издательский дом Академии имени Н. Е. Жуковского
125167, Москва, 8-го Марта 4-я ул., д. 6А
Тел.: (495) 973-45-68
E-mail: zakaz@itsbook.ru

Содержание

Лабораторная работа №6

1.Цель лабораторной работы

Целью лабораторной работы является освоение:

1) Приемов работы с использованием интерфейса программирования Windows Sockets для различных сетевых протоколов.
2) . Структуры и состава интерфейса программирования Windows Sockets.
3) Особенности работы с сокетами в C#.

2.Содержание отчёта

Отчет по лабораторной работе должен включать:
1) цель лабораторной работы;
2) конкретный вариант задания на выполнение;
3) тексты программ;
4) схемы алгоритмов;
5) результаты выполнения программ.

3.Задание на выполнение:

1) Создать приложение, дейтаграммный клиент на основе интерфейса программирования Windows Sockets.
2) Создать приложение сервер на основе интерфейса программирования Windows Sockets.
3) Создать чат на основе сокетов.

4. Краткие теоретические сведения.

4.1.Интерфейс прикладного программирования Windows Sockets (Winsock)

Windows Sockets (Winsock) – это сетевой интерфейс прикладного программирования, а не протокол, основной интерфейс доступа к различным сетевым базовым протоколам, реализованный на всех платформах. Winsock многое унаследовал от реализации BSD Sockets, но также включает Microsoft-specific расширения, постоянно продолжая развиваться и расширяться. Winsock поддерживает надежное, ориентированное на соединение, а также

ненадежное, не ориентированное на соединение взаимодействия. Начиная с ОС Windows 2000 включается Winsock 2.2, который доступен для потребительских версий в виде надстройки.

Winsock обеспечивает:

1. поддержку по механизму scatter-gather и ассинхронный ввод-вывод I/O.
2. поддержку Quality of service (QoS) если нижележащая сеть поддерживает QoS, приложения могут согласовывать между собой максимальные задержки и полосы пропускания.
3. расширяемость - Winsock можно использовать не только с протоколами, поддерживаемыми Windows 2000 но и с другими.
4. поддержку интегрированных пространств имен, отличных от определенных протоколом, который использует приложение вместе с Winsock. Например, сервер может опубликовать свое имя в Active Directory, а клиент, используя расширения пространства имен найти адрес сервера в Active Directory[1-3].
5. поддержку многоадресных сообщений передаваемых сразу нескольким адресатам.

Подробная информация о функционировании и использовании WinApi Sockets.см[4-6]. В приложении 3 в Таблица 1 представлены функции для работы с сокетами Windows

Особенности программирования сокетов в C#

Как показано в [5-6-] в основе межсетевых взаимодействий по протоколам TCP и UDP лежат сокеты. В .NET сокеты представлены классом System.NET.Sockets.Socket, который предоставляет низкоуровневый интерфейс для приема и отправки сообщений по сети.

Для создания объекта сокета можно использовать один из его конструкторов. Например, сокет, использующий протокол Tcp:

```
Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
```

Или сокет, использующий протокол Udp:

```
Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
```

Таким образом, при создании сокета мы можем указывать разные комбинации протоколов, типов сокета, значений из перечисления AddressFamily. Однако в то же время не все комбинации являются корректными. Так, для работы через протокол Tcp, нам надо обязательно указать параметры: AddressFamily.InterNetwork, SocketType.Stream и ProtocolType.Tcp. Для Udp набор параметров будет другим: AddressFamily.InterNetwork, SocketType.Dgram и ProtocolType.Udp. Для

других протоколов набор значений будет отличаться. Поэтому использование сокетов может потребовать некоторого знания принципов работы отдельных протоколов. Хотя в отношении Tcp и Udp все относительно просто.

Клиент-серверное приложение на сокетах TCP

Общий принцип работы сокетов в C# такой же, как и в WinApi.
На рис1 представлена схема работы серверного сокета TCP и на рис.2 - схема работы клиентского сокета TCP.

В приложении 1 представлены Листинг1 –программа сервер сокета TCP и Листинг2 –программа клиент сокета TCP.

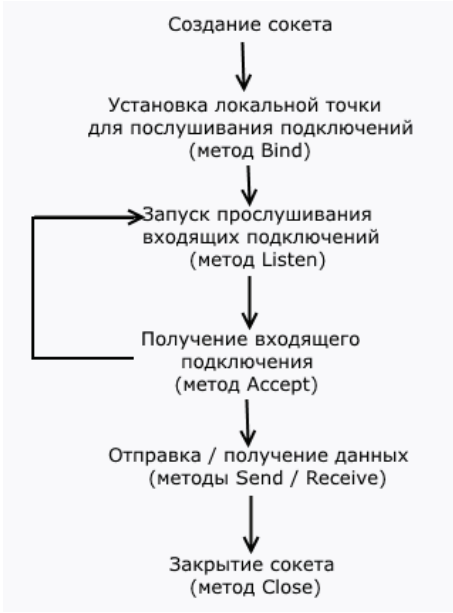

Рис1.- Схема работы серверного сокета TCP

Рис2.- Схема работы клиентского сокета TCP

Выполнение программ Листинг 1 и Листинг 2:

Консоль клиента

```
Введите сообщение: привет мир
ответ сервера: ваше сообщение доставлено
```

Консоль сервера

```
Сервер запущен. Ожидание подключений...
22:34: привет мир
```

Клиент-серверное приложение на сокетах UDP.

Протокол UDP не требует установки постоянного подключения.
В приложении 1 представлены Листинг3 –программа сокета UDP.

Вначале пользователь вводит порты для приема данных и для отправки. Предполагается, что два приложения клиента, которые будут между собой взаимодействовать, запущены на одной локальной машине. Если адреса клиентов различаются, то можно предусмотреть и ввода адреса для отправки данных.

После ввода портов запускается прослушивание входящих сообщений. В отличие от tcp-сервера не надо вызывать методы Listen и Accept. В бесконечном цикле напрямую можно получить данные с помощью метода ReceiveFrom(), который блокирует вызывающий поток, пока не придет очередная порция данных. В качестве параметра в метод передается массив байтов, в который надо считать данные, и удаленная точка, с которой приходят эти данные. Метод возвращает количество считанных байтов.

Этот метод возвращает через ref-параметр удаленную точку, с которой получены данные:
IPEndPoint remoteFullIp = remoteIp as IPEndPoint

То есть, несмотря на то, что в данном случае прием и отправка сообщений разграничены и текущий клиент отправляет данные только на введенный вначале порт, но вполне можем добавить возможность ответа на сообщения, используя данные полученной удаленной точки (адрес и порт).
В главном потоке происходит отправка сообщений с помощью метода SendTo(). В метод передается массив отправляемых данных, а также адрес, по которому эти данные надо отправить. Таким образом, приложение сразу осуществляет функции и сервера, и клиента.
Результат запуска двух копий приложения Листинга 3 и ввода разных данных для портов представлены ниже.

Первый клиент:

```
Введите порт для приема сообщений: 4004
Введите порт для отправки сообщений: 4005
Для отправки сообщений введите сообщение и нажмите Enter

127.0.0.1:4005 - привет порт 4004
добрый день, порт 4005
чудная погодка
```

Второй клиент

```
Введите порт для приема сообщений: 4005
Введите порт для отправки сообщений: 4004
Для отправки сообщений введите сообщение и нажмите Enter

привет порт 4004
127.0.0.1:4004 - добрый день, порт 4005
127.0.0.1:4004 - чудная погодка
```

В [7] приводятся рекомендации по наиболее эффективному использованию классов, содержащихся в System.Net:

- По возможности рекомендуется всегда использовать WebRequest и WebResponse вместо приведения типов к классам потомков. Приложения, использующие **WebRequest** и **WebResponse**, могут

использовать преимущества новых протоколов Интернета, не требуя при этом серьезного изменения кода.

- При написании приложений ASP.NET, которые выполняются на сервере с использованием классов **System.Net**, с точки зрения производительности зачастую лучше использовать асинхронные методы для GetResponse и GetResponseStream.
- Число открытых подключений к интернет-ресурсам может заметно влиять на производительность и пропускную способность сети. В **System.Net** по умолчанию используется два подключения для каждого приложения на один ведущий узел. С помощью свойства ConnectionLimit в ServicePoint приложения можно увеличить число его подключений для конкретного узла. Свойство ServicePointManager.DefaultPersistentConnectionLimit позволяет увеличить это значение по умолчанию для всех узлов.
- При написании протоколов уровня сокета по возможности используйте TcpClient или UdpClient вместо прямого использования Socket. Эти два клиентских класса инкапсулируют создание сокетов TCP и UDP, не требуя от вас обработки сведений о подключении.
- При доступе к узлам, требующим ввода учетных данных, используйте класс CredentialCache для создания кэша учетных данных вместо того, чтобы предоставлять их с каждым запросом. Класс **CredentialCache** выполняет поиск соответствующих запросу учетных данных в кэше, освобождая вас от необходимости создавать и предоставлять учетные данные на основе URL.

5. Список вопросов к защите лабораторной работы

1) Методы передачи информации, поддерживаемые Winsock.
2) Типы соединений, поддерживаемые Winsock.
3) Транспортные протоколы, поддерживаемые Winsock.
4) Инициализация Windows Socket. Получение информации о протоколах.
5) Создание Socket. Основные параметры.
6) Структура адреса Socket для протокола IP.
7) Структура приложения – простой клиент на основе Windows Socket.
8) Структура приложения – простой сервер на основе Windows Socket.

Лабораторная работа№7

1.Цель лабораторной работы

Целью лабораторной работы является освоение:
1. Особенностей широковещания.
2. Приемов работы с использованием интерфейса программирования Windows Sockets для односторонних сетевых протоколов.
4.Составления приложений, поддерживающих широковещание с использованием интерфейса программирования Windows Sockets.

2.Содержание отчёта

Отчет по лабораторной работе должен включать:
1) цель лабораторной работы;
2) конкретный вариант задания на выполнение;
3) тексты программ;
4) схемы алгоритмов;
5) результаты выполнения программ.

3.Задание на выполнение:

3.1.Создать клиентское и серверное приложение, ведущее рассылку сообщений с использованием механизма широковещанияСоздать приложение сервер сообщений с использованием механизма многоадресной рассылки в рамках группы..

4.Теретические сведения
4.1.Механизм широковещания.

Под широковещанием предполагается передача данных от одной рабочей станции всем рабочим станциям ЛВС. Все компьютеры сети могут получать и обрабатывать подобные сообщения, если они поддерживают неориентированные на соединения протоколы.

Высокая рабочая нагрузка сети при широковещании, является следствием требования обработки широковещательного сообщения каждого компьютера

ЛВС, хотя большинству компьютеров сети эти данные не нужны, и они отбрасываются. Тем не менее время на обработку пакетов данных тратится.

Каждый раз, когда UDP получает дейтаграмму от IP, он осуществляет фильтрацию, основанную на номере порта назначения, а иногда и на номере порта источника. Если указанный порт не обслуживается в текущий момент каким-либо процессом, дейтаграмма отбрасывается, и генерируется ICMP сообщение о недоступности порта. (TCP осуществляет подобную фильтрацию, основанную на своих номерах портов.) Если в UDP дейтаграмме обнаружена ошибка контрольной суммы, UDP ее отбрасывает.

Проблема широковещательных запросов заключается в том, что хосты, которые совсем не заинтересованы в получении этих запросов, должны их обрабатывать. Если в сети находится 40 хостов, но только 10 из них участвуют в работе этого приложения, в каждый момент времени один из 10 посылает широковещательный запрос UDP, при этом остальные 30 хостов должны обработать этот запрос, который проходит весь путь по стеку протоколов до UDP уровня, прежде чем дейтаграмма будет отброшена. UDP дейтаграмма отбрасывается этими 30 хостами.

Существуют три типа IP адресов: персональный (unicast), широковещательный (broadcast) и групповой (multicast).

Широковещательные запросы.

1) Ограниченный широковещательный адрес (limited broadcast address) - это адрес 255.255.255.255. Он может быть использован в качестве адреса назначения для IP дейтаграмм в процессе конфигурации хоста, когда хост может не знать собственной маски подсети и даже своего IP адреса. Дейтаграмма, направляющаяся на ограниченный широковещательный адрес, никогда (ни при каких условиях) не будет перенаправлена маршрутизатором. Она может существовать только на локальном кабеле. При наличии нескольких интерфейсов хоста, большинство систем воспринимают 255.255.255.255 как адрес широковещательного адреса первого интерфейса, который был сконфигурирован, и не позволяют послать дейтаграмму на все подключенные интерфейсы, поддерживающие широковещательную адресацию.

2) В широковещательном адресе сети (net-directed broadcast) идентификатор хоста устанавливается во все единичные биты. Широковещательный адрес для сети класса A, имеет вид netid.255.255.255, где netid это идентификатор сети класса A. Маршрутизаторы должны перенаправлять широковещательные запросы, направляемые в сеть, при этом,

однако, должна присутствовать опция, позволяющая отключить это перенаправление.

3) Широковещательный адрес подсети (subnet-directed broadcast address) имеет идентификатор хоста, установленный в единицы, однако определенный идентификатор подсети. Для того, чтобы классифицировать адрес как широковещательный адрес подсети, необходимо знать маску подсети.

4) Широковещательный адрес всех подсетей (all-subnets-directed broadcast address), также требует знания маски подсети в сети назначения. Только в этом случае можно определить различие между широковещательным адресом всех подсетей и широковещательным адресом сети. И идентификатор подсети, и идентификатор хоста, в данном случае, устанавливаются в единицы. Например, если маска подсети назначения 255.255.255.0, то IP адрес 128.1.255.255 это широковещательный запрос, направленный во все подсети. Однако, если сеть не разбита на подсети, это широковещательный запрос, направляемый в сеть.

В приложении 2 представлены листинги широковещательнах сокетов на C# и WinSockets

4.2. Многоадресное вещание.

Способность одного процесса передавать данные одному или более получателям называется многоадресная рассылка. Однако, методика присоединения процесса к многоадресному сеансу зависит от применяемого для передачи данных протокола.

Многоадресная передача по протоколу IP является видоизмененной формой широковещания. Для нее необходимо, чтобы все заинтересованные в приеме и передаче узлы были членами особой группы.

При присоединении группы к многоадресному вещанию, на сетевом адаптере добавляется фильтр. Он требует от сетевого оборудования обрабатывать и транслировать по сетевому стеку до соответствующего процесса только данные, предназначенные групповому адресу, к которому присоединился процесс.

Многоадресная рассылка

Технология многоадресной рассылки (multicasting) позволяет отправлять данные от одной рабочей станции в сети, а затем тиражировать их для остальных, не создавая большой нагрузки на сеть. Она является альтернативой широковещанию (broadcasting). При многоадресной рассылке данные передаются в сеть, только если их запрашивает получатель.

На платформе Win32 протоколов, поддерживающих многоадресную рассылку и доступных из Winsock, только два: IP и ATM. Отметим, что некоторые устаревшие сетевые адаптеры не могут принимать и отправлять информацию по адресам многоадресной IP-рассылки.

Многоадресная рассылка обладает двумя важными характеристиками: плоскостью управления (control plane) и плоскостью данных (data plane). Первая определяет способ организации членства в группах, вторая - способ распространения данных среди членов группы. Любая из этих плоскостей может быть корневой или равноправной. Пример корневой плоскости управления – протокол ATM.

Равноправная плоскость управления позволяет соединиться с группой любому участнику. При этом, протокол реализованный для членов группы зависит от программиста. Равноправной плоскостью управления, например, является многоадресная IP-рассылка.

Также может быть маршрутизируемой и немаршрутизируемой и плоскость данных. В маршрутизируемой плоскости данных есть основной участник. Передача данных происходит только между ним и остальными. Трафик при этом может быть и одно-, и двунаправленным. Пример маршрутизируемой плоскости данных – ATM.

В немаршрутизируемой плоскости данных любой член группы вправе передавать данные любому другому члену. Многоадресная рассылка в сетях ATM маршрутизируется и в плоскости управления, и в плоскости данных, в то время как в сетях IP она не маршрутизируется ни в одной из плоскостей.

При перечислении записей протоколов информацию о поддержке многоадресной рассылки можно извлечь из поля dwServiceFlags структуры WSAPROTOCOL_INFO (установлен бит XP1_SUPPORT_MULTIPOINT). Установка бита XP1_MULTIPOINT_CONTROL_PLANE означает поддержку маршрутизируемой плоскости управления, а установка бита XP1_MULTIPOINT_DATA_PLANE – маршрутизируемой плоскости данных.

Многоадресная рассылка в сетях IP

Для многоадресной рассылки в сетях IP используются групповые адреса (multicast address). Они должны лежать в диапазоне 224.0.0.0 – 239.255.255.255. При этом часть этих адресов зарезервирована для специальных целей. Так, например, адрес 224.0.0.0 –базовый, 224.0.0.1 используется как групповой адрес всех систем данной подсети, 224.0.0.2 – как групповой адрес всех маршрутизаторов данной подсети, 224.0.1.24 – как групповой адрес WINS-сервера. IP требует, чтобы специальные адреса использовались только для многоадресного трафика.

Для управления клиентами многоадресной рассылки и их членством в группах был разработан специальный протокол IGMP (Internet Group Management Protocol). Он позволяет осуществлять действия, для которых

нельзя воспользоваться средствами IP, например, присоединить рабочие станции из разных подсетей к одной многоадресной группе.

Список вопросов к защите лабораторной работы

1) Свойства многоадресной рассылки.
2) Особенности широковещания
3) Протоколы, поддерживающие широковещание
4) Протоколы, поддерживающие многоадресную рассылку
5) Преобразование сетевых адресов, представленных в сетевом порядке в системный.
6) Серверные API – функции для Socket.
7) Клиентские API – функции для Socket.
8) Проверка установок и соединений с помощью команд ОС Windows.

Приложение 1

Листинг1 –программа сервер сокета TCP

```
.
using System;
using System.Text;
using System.Net;
using System.Net.Sockets;
namespace SocketTcpServer
{
class Program
{
static int port = 8005; // порт для приема входящих запросов
static void Main(string[] args)
{
// получаем адреса для запуска сокета
IPEndPoint ipPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"), port);
// создаем сокет
Socket listenSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
try
{
// связываем сокет с локальной точкой, по которой будем принимать данные
listenSocket.Bind(ipPoint);
// начинаем прослушивание
listenSocket.Listen(10);
Console.WriteLine("Сервер запущен. Ожидание подключений...");
while (true)
{
Socket handler = listenSocket.Accept();
// получаем сообщение
StringBuilder builder = new StringBuilder();
nt bytes = 0; // количество полученных байтов
byte[] data = new byte[256]; // буфер для получаемых данных
do
{
bytes = handler.Receive(data);
builder.Append(Encoding.Unicode.GetString(data, 0, bytes));
}
while (handler.Available>0);
```

```
Console.WriteLine(DateTime.Now.ToShortTimeString() + ": " + builder.ToString());
// отправляем ответ
string message = "ваше сообщение доставлено";
data = Encoding.Unicode.GetBytes(message);
handler.Send(data);
// закрываем сокет
handler.Shutdown(SocketShutdown.Both);
handler.Close();
}
}
catch(Exception ex)
{
Console.WriteLine(ex.Message);
}
}
}
}
```

Листингт2 Простой клиент TCP

```
using System;
using System.Text;
using System.Net;
using System.Net.Sockets;
namespace SocketTcpClient
{
class Program
{
/ адрес и порт сервера, к которому будем подключаться
static int port = 8005; // порт сервера
static string address = "127.0.0.1"; // адрес сервера
static void Main(string[] args)
{
try
{
IPEndPoint ipPoint = new IPEndPoint(IPAddress.Parse(address), port);
Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
// подключаемся к удаленному хосту
socket.Connect(ipPoint);
Console.Write("Введите сообщение:");
```

```
string message = Console.ReadLine();
byte[] data = Encoding.Unicode.GetBytes(message);
socket.Send(data);
// получаем ответ
data = new byte[256]; // буфер для ответа
StringBuilder builder = new StringBuilder();
int bytes = 0; // количество полученных байт
do
{
bytes = socket.Receive(data, data.Length, 0);
builder.Append(Encoding.Unicode.GetString(data, 0, bytes));
}
while (socket.Available > 0);
Console.WriteLine("ответ сервера: " + builder.ToString());
// закрываем сокет
socket.Shutdown(SocketShutdown.Both);
socket.Close();
}
catch(Exception ex)
{
Console.WriteLine(ex.Message);
}
Console.Read();
}
}
}
```

Листинг3 программы UDP-клиента

```
using System;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using System.Net.Sockets;
namespace SocketUdpClient
{
class Program
{
static int localPort; // порт приема сообщений
static int remotePort; // порт для отправки сообщений
static Socket listeningSocket;
 static void Main(string[] args)
```

```
{
Console.Write("Введите порт для приема сообщений: ");
localPort = Int32.Parse(Console.ReadLine());
Console.Write("Введите порт для отправки сообщений: ");
remotePort = Int32.Parse(Console.ReadLine());
Console.WriteLine("Для отправки сообщений введите сообщение и нажмите Enter");
Console.WriteLine();
try
{
listeningSocket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
ask listeningTask = new Task(Listen);
listeningTask.Start();
// отправка сообщений на разные порты
while (true)
{
string message = Console.ReadLine();
byte[] data = Encoding.Unicode.GetBytes(message);
EndPoint remotePoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"), remotePort);
listeningSocket.SendTo(data, remotePoint);
}
}
catch (Exception ex)
{
Console.WriteLine(ex.Message);
}
finally
{
Close();
}
}
// поток для приема подключений
private static void Listen()
{
try
{
//Прослушиваем по адресу
IPEndPoint localIP = new IPEndPoint(IPAddress.Parse("127.0.0.1"), localPort);
listeningSocket.Bind(localIP);
```

```
while (true)
{
// получаем сообщение
  StringBuilder builder = new StringBuilder();
int bytes = 0; // количество полученных байтов
byte[] data = new byte[256]; // буфер для получаемых данных
//адрес, с которого пришли данные
EndPoint remoteIp = new IPEndPoint(IPAddress.Any, 0);
do
{
bytes = listeningSocket.ReceiveFrom(data, ref remoteIp);
builder.Append(Encoding.Unicode.GetString(data, 0, bytes));
}
while (listeningSocket.Available > 0);
// получаем данные о подключении
IPEndPoint remoteFullIp = remoteIp as IPEndPoint;
// выводим сообщение
Console.WriteLine("{0}:{1} - {2}", remoteFullIp.Address.ToString(),
  remoteFullIp.Port, builder.ToString());
}
}
catch (Exception ex)
{
Console.WriteLine(ex.Message);
}
finally
{
Close();
}
}
// закрытие сокета
private static void Close()
{
if (listeningSocket != null)
{
listeningSocket.Shutdown(SocketShutdown.Both);
listeningSocket.Close();
listeningSocket = null;
}
}
}
```

Приложение2
Широковещательная рассылка
Листинг 1- Клиент(C#)

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;

namespace MulticastApp
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            SendMessage(); // отправляем сообщение
         }
         catch (Exception ex)
         {
            Console.WriteLine(ex.Message);
         }
      }
      private static void SendMessage()
      {
         int groupPort = 11000;
         IPEndPoint groupEP = new
IPEndPoint(IPAddress.Parse("255.255.255.255"), groupPort);
         try
         {
            while (true)
            {
               string message = Console.ReadLine();
               Socket socket = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);
               socket.SetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.Broadcast, 1);
```

```
socket.SendTo(System.Text.ASCIIEncoding.ASCII.GetBytes(message), groupEP);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
  }
}
```

Листинг 2 –Сервер(C#)

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;

namespace MulticastApp
{
  class Program
  {
    static void Main(string[] args)
    {
      try
      {
        Thread receiveThread = new Thread(new ThreadStart(ReceiveMessage));
        receiveThread.Start();
      }
      catch (Exception ex)
      {
        Console.WriteLine(ex.Message);
      }
    }
```

```
        private static void ReceiveMessage()
        {
            int groupPort = 11000;
            IPEndPoint groupEP = new
IPEndPoint(IPAddress.Parse("255.255.255.255"), groupPort);
            Socket socket = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);
            socket.SetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.Broadcast, 1);
            try
            {
                while (true)
                {
                    Socket listener = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);
                    IPEndPoint localEndPoint = new IPEndPoint(IPAddress.Any, 11000);
                    listener.Bind(localEndPoint);
                    EndPoint ep = (EndPoint)localEndPoint;
                    Console.WriteLine("Ready to receive…");
                    byte[] data = new byte[1024];
                    int recv = listener.ReceiveFrom(data, ref ep);
                    string stringData = Encoding.ASCII.GetString(data, 0, recv);
                    Console.WriteLine("received: {0} from: {1}", stringData,
ep.ToString());
                    listener.Close();
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
}
```

Листинг 3 – Сервер(WinSock)

```
#pragma comment(lib, "WS2_32.lib")
#include"winsock2.h"
#include<iostream>
#include<conio.h>
using namespace std;
```

```
#define MYPORT 9002
int main()
{
	WSADATA wsaData;
	WSAStartup(MAKEWORD(2, 2), &wsaData);
	SOCKET sock;
	sock = socket(AF_INET, SOCK_DGRAM, 0);
	char broadcast = '1';
	if (setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &broadcast,
sizeof(broadcast)) < 0)
	{
		cout << "Error in setting Broadcast option";
		closesocket(sock);
		return 0;
	}
	struct sockaddr_in Recv_addr;
	struct sockaddr_in Sender_addr;
	int len = sizeof(struct sockaddr_in);
	char sendMSG[50] = "";
	cin >> sendMSG;
	char recvbuff[50] = "";
	int recvbufflen = 50;
	Recv_addr.sin_family = AF_INET;
	Recv_addr.sin_port = htons(MYPORT);
	Recv_addr.sin_addr.s_addr = INADDR_BROADCAST;
	sendto(sock, sendMSG, strlen(sendMSG) + 1, 0, (sockaddr*)&Recv_addr,
sizeof(Recv_addr));
	recvfrom(sock, recvbuff, recvbufflen, 0, (sockaddr *)&Recv_addr, &len);
	cout << "\n\n\tReceived message from Reader is => " << recvbuff;
	cout << "\n\n\tpress any key to CONT...";
	_getch();
	closesocket(sock);
	WSACleanup();
}
```

Листинг 4 -Клиент(WinSock)

```
#pragma comment(lib, "WS2_32.lib")
#include <winsock2.h>
```

```
#include <iostream>
#include <conio.h>
using namespace std;
#define MYPORT 9001
int main()
{
	WSADATA wsaData;
	WSAStartup(MAKEWORD(2, 2), &wsaData);
	SOCKET sock; sock = socket(AF_INET, SOCK_DGRAM, 0);
	char broadcast = '1';
	char sendMSG[] = "hoho haha";
	if (setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &broadcast,
sizeof(broadcast)) < 0)
	{
		cout << endl << "ERROR in setting broad option" << endl;
		closesocket(sock);
		return 0;
	}
	struct sockaddr_in Recv_addr;
	struct sockaddr_in Sender_addr;
	int len = sizeof(struct sockaddr_in);
	char recvbuff[50];
	int recvbufflen = 50;
	char sendMSDG[] = "Broadcast message from READER";
	Recv_addr.sin_family = AF_INET;
	Recv_addr.sin_port = htons(MYPORT);
	Recv_addr.sin_addr.s_addr = INADDR_ANY;
	if (bind(sock, (sockaddr*)&Recv_addr, sizeof(Recv_addr)) < 0)
	{
		cout << "err binding" << endl<<WSAGetLastError();
		_getch();
		closesocket(sock);
		return 0;
	}
	while (1)
	{
		recvfrom(sock, recvbuff, recvbufflen, 0, (sockaddr*)&Sender_addr,
&len);
		cout << "MSG: " << recvbuff << endl;
	}
```

```
	//cout << endl << "getch" << endl;
	system("pause");
	if (sendto(sock, sendMSG, strlen(sendMSG) + 1, 0, (sockaddr*)&Sender_addr, sizeof(Sender_addr)) < 0)
	{
		cout << "errrr" << WSAGetLastError();
		system("pause");
		closesocket(sock);
		return 0;
	}
	else
		cout << "No errrorrrr" << endl;
	closesocket(sock);
	WSACleanup();
	return 0;
}
//int main()
//{
//
//}
```

Приложение 3

Таблица 1 -Функции для работы с сокетами Windows

Функция	**Описание**
accept	Принимает входящую попытку соединения на сокете.
AcceptEx	Принимает новое соединение, возвращает локальный и удаленный адреса, а также принимает первый блок данных, посланный клиентским приложением.

bind	Связывает локальный адрес с сокетом.
closesocket	Закрывает существующий сокет.
connect	Устанавливает соединение с указанным сокетом.
ConnectEx	Устанавливает соединение с указанным сокетом, и (необязательно) посылает данные, как только соединение установлено.
DisconnectEx	Закрывает соединение на сокете, и позволяет снова использовать дескриптор этого сокета.
freeaddrinfo	Освобождает информацию об адресе, которая была получена динамически при помощи функции getaddrinfo.
gai_strerror	Помогает сформировать сообщения об ошибках из кодов ошибок типа EAI_*, возвращенных функцией getaddrinfo.
GetAcceptExSockaddrs	Анализирует данные, полученные функцией AcceptEx.
GetAddressByName	Запрашивает пространство имен или устанавливает пространства имен заданные по умолчанию для получения информации о сетевом адресе указанного сетевого сервиса. Этот процесс известен как Разрешение Имен Сервисов (Service Name Resolution).
getaddrinfo	Обеспечивает протоколо-независимую трансляцию имени хоста в его адрес.
gethostbyaddr	Получает информацию о хосте по сетевому адресу.
gethostbyname	Получает из базы данных хостов информацию, соответствующую имени хоста.
gethostname	Получает стандартное имя хоста для локального компьютера.
getnameinfo	Обеспечивает разрешение имени из адреса в имя хоста.
getpeername	Получает имя одноуровнего (peer) узла сети, с которым соединен сокет.

getprotobyname	Получает информацию о протоколе по его имени.
getprotobynumber	Получает информацию о протоколе по его номеру.
getservbyname	Получает информацию о сервисе по имени сервиса и протоколу.
getservbyport	Получает информацию о сервисе по номеру порта и протоколу.
getsockname	Получает локальное имя для сокета.
getsockopt	Получает опции сокета.
htonl	Преобразует порядок байтов **u_long** из формата хоста в сетевой формат TCP/IP (формат big-endian).
htons	Преобразует порядок байтов **u_short** из формата хоста в сетевой формат TCP/IP (формат big-endian).
inet_addr	Преобразует строку, содержащую десятично-точечный адрес протокола Internet (IPv4) в надлежащий адрес для структуры in_addr.
inet_ntoa	Преобразует адрес Internet (IPv4) из сетевого формата в строку, содержащую стандартный десятично-точечный адрес.
ioctlsocket	Управляет режимом ввода-вывода сокета.
listen	Переводит сокет в состояние ожидания входящих соединений.
ntohl	Преобразует порядок байтов **u_long** из сетевого формата TCP/IP в формат хоста (формат little-endian на процессорах Intel).
ntohs	Преобразует порядок байтов **u_short** из сетевого формата TCP/IP в формат хоста (формат little-endian на процессорах Intel).
recv	Получает данные от присоединенного или связанного сокета.
recvfrom	Получает датаграмму и сохраняет адрес источника.
select	Определяет статус одного или нескольких сокетов, ожидая в случае необходимости исполнения синхронного ввода-вывода.
send	Посылает данные на соединенный сокет.

sendto	Посылает данные определенному адресату.
setsockopt	Устанавливает опции сокета.
shutdown	Запрещает прием или передачу на сокете.
socket	Создает сокет и связывает его с определенным сервисом.
TransmitFile	Передает данные файла через соединенный сокет.
TransmitPackets	Передает данные из памяти или данные файла через соединенный сокет.
WSAAccept	Принимает соединение, основываясь на возвращаемом значении функции условия, обеспечивает качество обслуживания (QoS) спецификаций процесса и разрешает передачу данных соединения.
WSAAddressToString	Преобразует все компоненты структуры SOCKADDR в понятное для человека строковое представление адреса.
WSAAsyncGetHostByAddr	Асинхронно получает информацию о хосте по его адресу.
WSAAsyncGetHostByName	Асинхронно получает информацию о хосте по его имени.
WSAAsyncGetProtoByName	Асинхронно получает информацию о протоколе по его имени.
WSAAsyncGetProtoByNumber	Асинхронно получает информацию о протоколе по его номеру.
WSAAsyncGetServByName	Асинхронно получает информацию о сервисе по его имени и номеру порта.
WSAAsyncGetServByPort	Асинхронно получает информацию о сервисе по номеру порта и протоколу.
WSAAsyncSelect	Назначает сообщение, которое будет посылаться оконной процедуре в том случае, когда на сокете произойдет какое-либо событие.
WSACancelAsyncRequest	Отменяет незавершенную асинхронную операцию.
WSACleanup	Завершает использование Ws2_32.dll.
WSACloseEvent	Закрывает открытый дескриптор объекта события.

WSAConnect	Устанавливает соединение с сокетом другого приложения, обменивается данными соединения и определяет необходимое качество обслуживания (QoS), основываясь на указанной структуре FLOWSPEC.
WSACreateEvent	Создает новый событийный объект.
WSADuplicateSocket	Возвращает структуру, которая может использоваться для создания нового дескриптора общедоступного сокета (сокет может использоваться несколькими процессами).
WSAEnumNameSpaceProviders	Возвращает информацию о доступных пространствах имен.
WSAEnumNetworkEvents	Обнаруживает сетевые события для обозначенного сокета, очищает внутренние записи сетевых событий и сбрасывает событийные объекты (необязательно).
WSAEnumProtocols	Получает информацию о доступных транспортных протоколах.
WSAEventSelect	Определяет событийный объект, который будет связан с указанным набором FD_XXX сетевых событий.
WSAGetLastError	Возвращает статус ошибки для последней операции, которая завершилась неудачно.
WSAGetOverlappedResult	Получает результаты перекрытой (совместной) операции на указанном сокете.
WSAGetQOSByName	Инициализирует структуру QOS, основываясь на названном шаблоне, или предоставляет буфер для получения доступных имен шаблонов.
WSAGetServiceClassInfo	Получает информацию о классе (схему), имеющую отношение к указанному классу сервиса, от заданного поставщика пространства имен (name space provider).
WSAGetServiceClassNameByClassId	Получает название сервиса, связанного с указанным типом.

WSAHtonl	Преобразует порядок байтов **u_long** из формата хоста в сетевой формат.
WSAHtons	Преобразует порядок байтов **u_short** из формата хоста в сетевой формат.
WSAInstallServiceClass	Регистрирует схему класса сервиса в пространстве имен.
WSAIoctl	Управляет режимом сокета.
WSAJoinLeaf	Присоединяется к узлу многоточечной сессии, обменивается данными соединения и определяет необходимое качество обслуживания (QoS), основываясь на указанных структурах.
WSALookupServiceBegin	Инициализирует клиентский запрос, основываясь на критериях, содержащихся в структуре WSAQUERYSET.
WSALookupServiceEnd	Освобождает дескриптор, использовавшийся предыдущими вызовами WSALookupServiceBegin и WSALookupServiceNext.
WSALookupServiceNext	Получает информацию о требуемом сервисе.
WSANSPIoctl	Разработан для создания запросов к зарегистрированному пространству имен, управляющих вводом-выводом.
WSANtohl	Преобразует порядок байтов **u_long** из сетевого формата в формата хоста.
WSANtohs	Преобразует порядок байтов **u_short** из сетевого формата в формата хоста.
WSAProviderConfigChange	Уведомляет приложение о смене конфигурации поставщика.
WSARecv	Получает данные от соединенного сокета.
WSARecvDisconnect	Завершает прием на сокете и получает данные разъединения, если сокет является ориентированным на соединение.
WSARecvEx	Получает данные от соединенного сокета.
WSARecvFrom	Получает датаграмму и сохраняет адрес источника.

WSARecvMsg	Получает данные и дополнительную управляющую информацию от соединенных и несоединенных сокетов.
WSARemoveServiceClass	Удаляет схему класса сервиса из системного реестра.
WSAResetEvent	Сбрасывает состояние указанного событийного объекта в несигнальное.
WSASend	Посылает данные на соединенный сокет.
WSASendDisconnect	Инициализирует завершение соединения на сокете и посылает данные разъединения.
WSASendTo	Посылает данные определенному адресату, используя перекрываемый ввод-вывод где это возможно.
WSASetEvent	Устанавливает состояние указанного событийного объекта в сигнальное.
WSASetLastError	Устанавливает код ошибки.
WSASetService	Регистрирует или удаляет из системного реестра экземпляр сервиса в пределах одного или нескольких пространств имен.
WSASocket	Создает сокет, связанный с определенным поставщиком транспортных услуг (transport-service provider).
WSAStartup	Инициализирует использование Ws2_32.dll процессом.
WSAStringToAddress	Преобразует числовую строку в структуру SOCKADDR.
WSAWaitForMultipleEvents	Возвращает значение в том случае, когда один или все указанные событийные объекты находятся в сигнальном состоянии, или когда истекает интервал ожидания (тайм-аут).

Литература.

1.В.Г.Олифер, Н.А. Олифер. Сетевые операционные системы. - Спб: Питер, 2003.

2.Сетевые средства Microsoft Windows NT Server 4.0 / Пер. с англ. - Спб: Издательская группа BHV,1999.

3.Джонс Э., Оланд Дж. Программирование в сетях Microsoft Windows / Пер. с англ. -Спб:: Издательско-торговый дом «Русская редакция», 2002.

4.Рихтер Дж., Кларк Дж. Д. Программирование серверных приложений для Microsoft Windows 2000/ Пер. с англ. - Спб: : Издательско-торговый дом «Русская редакция», 2001.

5.Черкасова Н.И. Сетевые операционные системы. Сетевые файловые системы и служба каталогов Учебное пособие-МГТУ ГА. 2015г.

6 .Черкасова Н.И. Программирование сетевых приложений для ОС семейства Windows Учебное пособие-МГТУ ГА. 2007г.

7.https://docs.microsoft.com