2016

International Journal of Humanities & Information Technology (IJHIT)

Volume – 1, Issue - 1

ISSN : Applied

INTERNATIONAL JOURNAL OF HUMANITIES & INFORMATION TECHNOLOGY (IJHIT)

WWW.IJHIT.COM

Implementation of Connection Oriented Socket Programming In C#

Sanjay Kumar Shukla[1], Meenakshi Shukla[2]

[1](Research Scholar/Department of Computer Science & Engineering/ Sri Krishnadevraya University/ India)
[2](Associate Prof./Department of Computer Science & Engineering/ Sri Krishnadevraya University / India)

1. Definition

A **socket** is one endpoint of a two-way communication link between two programs running on the **network**. A **socket** is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. An endpoint is a combination of an IP address and a port number.

Keywords: *I/O stream, I/O buffer, local, remote, handshaking, client, server*

2. Introduction

An Internet socket is characterized by at least the following :

2.1. Local Socket Address: Local IP address and port number

2.2. Protocol: A transport protocol (e.g., TCP, UDP, raw IP, or others).

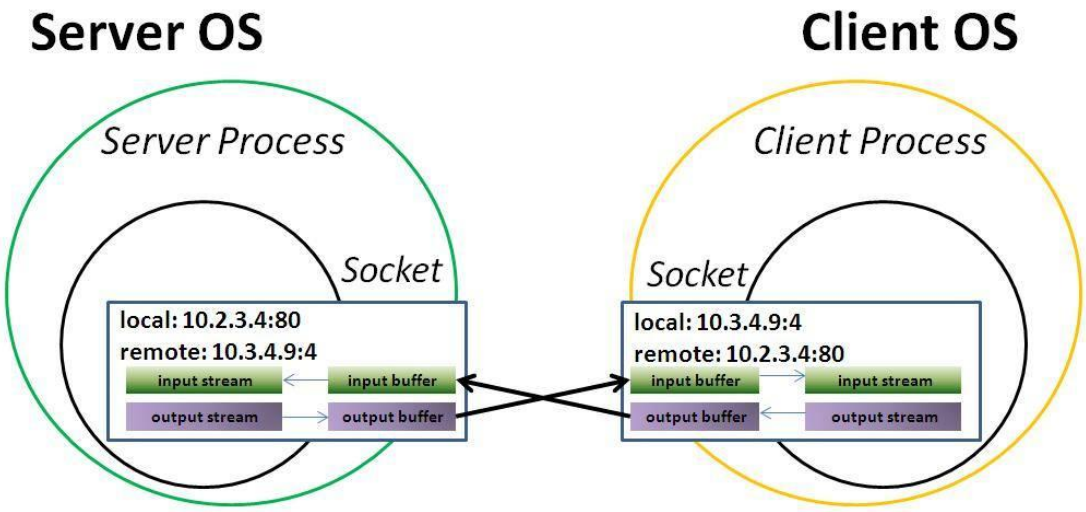
Consequently, TCP port 53 and UDP port 53 are different, distinct sockets.

A socket that has been connected to another socket, e.g. during the establishment of a TCP connection, will also be characterized by the following:

2.3. Remote Socket Address.

As discussed in the client-server section below, a TCP server may serve several clients concurrently. The server creates one socket for each client, and these sockets share the same local socket address from the point of view of the TCP server, and have different remote address for each client.

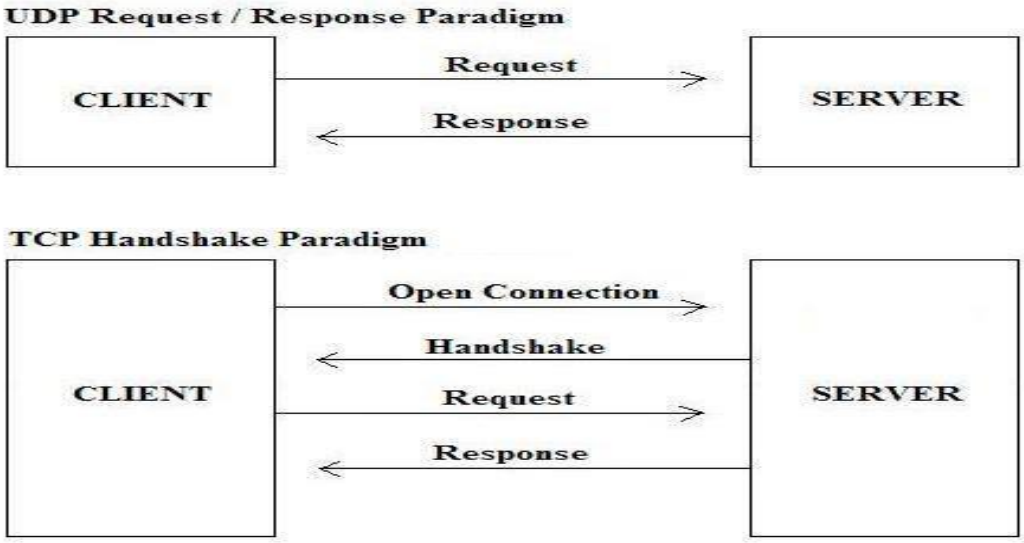
Within the operating system and the application that created a socket, a socket is referred to by a unique integer value called a *socket descriptor*. The operating system forwards the payload of incoming IP packets to the corresponding application by extracting the socket address information from the IP and transport protocol headers and stripping the headers from the application data.



Several Internet socket types are available:

- Datagram sockets, also known as connectionless sockets, which use User Datagram Protocol (UDP).
- Stream sockets, also known as connection-oriented sockets, which use Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP).
- Raw sockets (or *Raw IP sockets*), typically available in routers and other network equipment. Here the transport layer is bypassed, and the packet headers are made accessible to the application

3. Difference Between Connection Less And Connection Oriented Socket

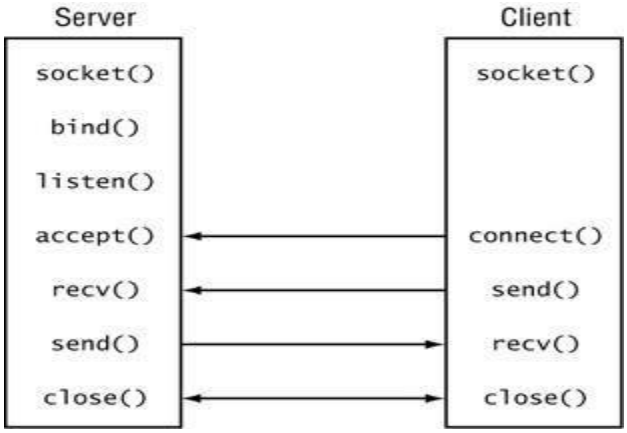


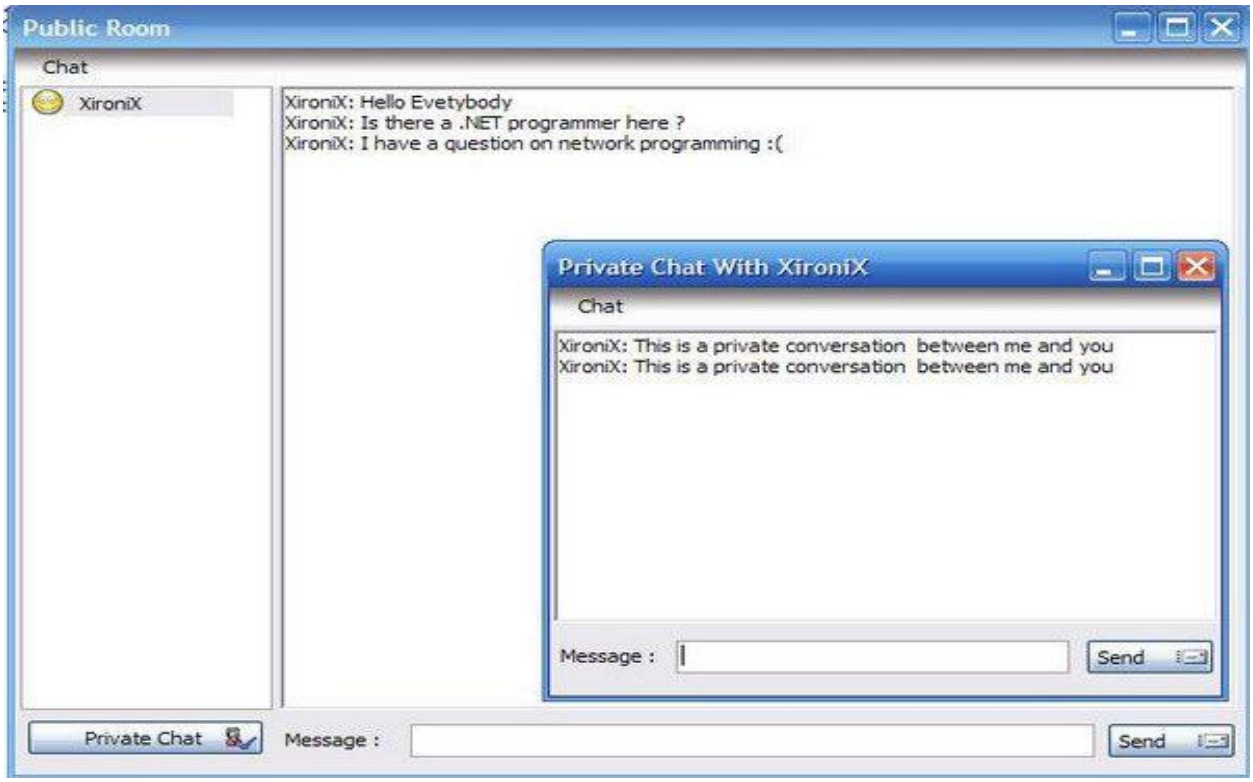
In Terms	TCP	UDP
Reliability	TCP is connection-oriented protocol. When a file or message sent it will be delivered unless the connection fails. If the connection is lost, the server will request the lost part. There is no corruption while transferring a message.	UDP is connectionless protocol. When you a send a data or message, you don't know if it'll get there, it could get lost on the way. There may be corruption while transferring a message.
Ordered	When the low level parts of the TCP "stream" arrive in the wrong order, resend requests have to be sent, and all the out of sequence parts have to be put back together, so requires a bit of work to piece together.	If you send two messages out, you don't know what order they'll arrive in i.e. **not ordered**
Weight	**Heavyweight:** when the low level parts of the TCP "stream" arrive in the wrong order, resend requests have to be sent, and all the out of sequence parts have to be put back together, so requires a bit of work to piece together.	**Lightweight:** No ordering of messages, no tracking connections, etc. It's just fire and forget! This means it's a lot quicker, and the network card / OS has to do very little work to translate the data back from the packets.
Streaming	It uses the stream with nothing distinguishing where one packet ends and another packet starts for reading data.	UDP does not use streaming and it uses datagrams instead of streams
Examples	World Wide Web (Apache TCP port 80), e-mail (SMTP TCP port 25 Postfix MTA), File Transfer Protocol (FTP port 21) and Secure Shell (OpenSSH port 22) etc.	Domain Name System (DNS UDP port 53), streaming media applications such as IPTV or movies, Voice over IP (VoIP), Trivial File Transfer Protocol (TFTP) and online multiplayer games etc

4. Introduction of Connection-Oriented Socket Technology

The world of IP connectivity revolves around two types of communication: connection-oriented and connectionless.

In a connection-oriented socket, the TCP protocol is used to establish a session (connection) between two IP address endpoints. There is a fair amount of overhead involved with establishing the connection, but once it is established, the data can be reliably transferred between the devices.





5. Algorithm to Implement Connectionless Socket Programming

5.1. Algo to Creating a Socket: socket()

- Operation to create a socket
  - int socket(int domain, int type, int protocol)
  - Returns a descriptor (or handle) for the socket
  - Originally designed to support any protocol suite
- Domain: protocol family

- Use PF_INET for the Internet
- Type: semantics of the communication
  - SOCK_STREAM: reliable byte stream
  - SOCK_DGRAM: message-oriented service
- Protocol: specific protocol
  - UNSPEC: unspecified. No need for us to specify, since PF_INET plus SOCK_STREAM already implies TCP, or SOCK_DGRAM implies UDP

5.2. Algo to Sending and Receiving Data

- Sending data
  - ssize_t write(int sockfd, void *buf, size_t len)
  - Arguments: socket descriptor, pointer to buffer ofdata to send, and length of the buffer
  - Returns the number of characters written, and -1 on error
  - send(): same as write() with extra flagsparameter
- Receiving data
  - ssize_t read (int sockfd, void *buf, size_t len)
  - Arguments: socket descriptor, pointer to buffer to place the data, size of the buffer
  - Returns the number of characters read (where 0 implies “end of file”), and -1 one error
  - recv(): same as read() with extra flagsparameter
- Closing the socket
  - int close(int sockfd)

5.3. Byte Ordering

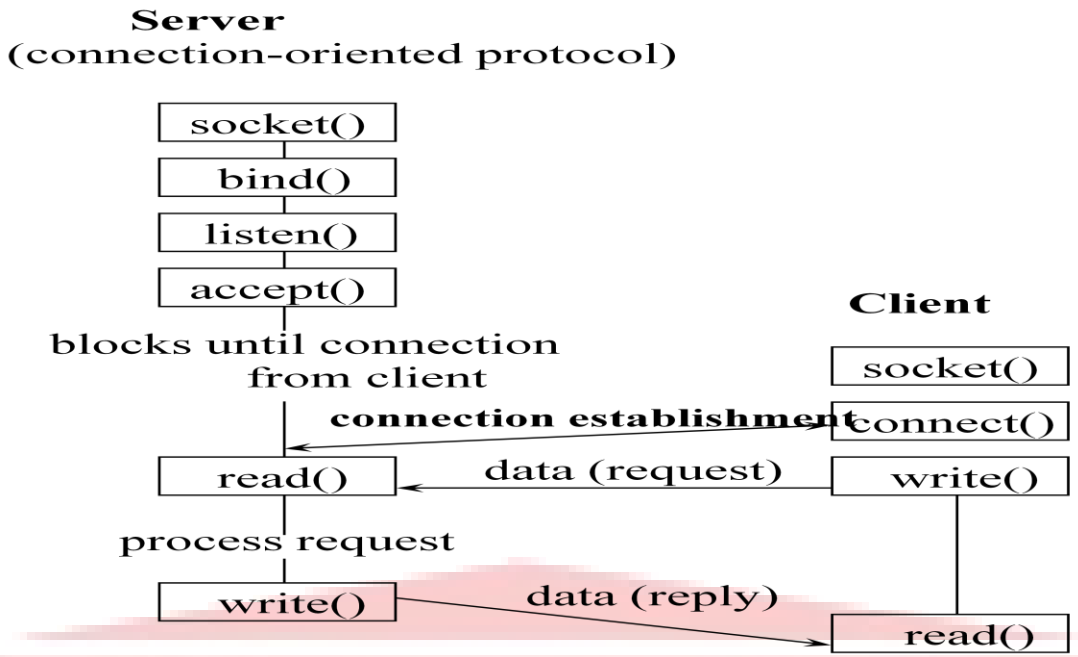
Port numbers and IP Addresses (both discussed next) are represented by multi-byte data types which are placed in packets for the purpose of routing and multiplexing. Port numbers are two bytes (16 bits) and IP4 addresses are 4 bytes (32 bits), and a problem arises when transferring multi-byte data types between different architectures. Say Host A uses a “big-endian” architecture and sends a packet across the network to Host B which uses a “little-endian” architecture. If Host B looks at the address to see if the packet is for him/her (choose a gender!), it will interpret the bytes in the opposite order and will wrongly conclude that it is not his/her packet. The Internet uses big-endian and we call it the network-byte-order and it is really not important to know which method it uses since we have the following functions to convert host-byte-ordered values into network-byte-ordered values and vice versa:

5.3.1. To convert port numbers (16 bits):

- Host -> Network unit16_t htons(uint16_t hostportnumber)
- Network -> Host unit16_t ntohs(uint16_t netportnumber)

5.3.2. To convert IP4 Addresses (32 bits):

- Host -> Network unit32_t htonl(uint32_t hostportnumber)
- Network -> HostUnit32_t ntohl(uint32_t netportnumbe



6. Example of socket programming in C#

6.1. Creating socket
6.2. Binding socket
6.3. Connecting socket
6.4. Sending data
6.5. Receiving data
6.6. Closing socket

6.1. Creating Socket

```
public socket(

AddressFamily addressFamily,

   SocketType socketType,
   ProtocolType protocolType

   );
```

6.2. Binding Socket

```
IPAddress  bindAddress = IPAddress.Parse( userInputString );
IPEndPoint bindEndPoint = new IPEndPoint( bindAddress, 5150 );
Socket     mySocket = null;

try
```

```
{
   mySocket = new Socket(
      bindAddress.AddressFamily,
      SocketType.Stream,
      ProtocolType.Tcp);
   mySocket.Bind( bindEndPoint );
}
catch ( SocketException err )
{
   if ( mySocket != null )
      mySocket.Close();
}
```

6.3. Connecting socket

```
Socket      tcpSocket = null;
IPHostEntry  resolvedServer;
IPEndPoint   serverEndPoint;

try
{
   resolvedServer = Dns.Resolve( "server-name" );
   foreach( IPAddress addr in resolvedServer.AddressList )
   {
      serverEndPoint = new IPEndPoint( addr, 5150 );
      tcpSocket = new Socket(
         addr.AddressFamily,
         SocketType.Stream,
         ProtocolType.Tcp
         );
      try
      {
         tcpSocket.Connect( serverEndPoint );
      }
      catch
      {
         // Connect failed so try the next one make sure to close the socket we opened
         if ( tcpSocket != null )
            tcpSocket.Close();
         continue;
      }
      break;
   }
}
catch ( SocketException err )
{
   Console.WriteLine("Client connection failed: {0}", err.Message);
}
// Now use tcpSocket to communicate to the server
```

6.4. Sending data

```
Socket   clientSocket = null;
byte [ ]  dataBuffer = new byte [ 1024 ];

// Create a TCP socket and connect to a server
try
{
   clientSocket.Send( dataBuffer );
}
catch ( SocketException err )
{
   Console.WriteLine("Send failed: {0}", err.Message);
}
```

6.5. Receiving data

```
IPAddress   bindAddress = IPAddress.Any;
IPEndPoint  bindEndPoint = new IPEndPoint( bindAddress, 5150 );
Socket      udpSocket;
byte [ ]    receiveBuffer = new byte [ 1024 ];
IPEndPoint  senderEndPoint = new IPEndPoint(bindAddress.AddressFamily, 0);
EndPoint  castSenderEndPoint = (EndPoint) senderEndPoint;
int       rc;

udpSocket = new Socket(
   bindAddress.AddressFamily,
   SocketType.Dgram,
   ProtocolType.Udp
   );
try
{
   udpSocket.Bind( bindEndPoint );
   rc = udpSocket.ReceiveFrom( receiveBuffer, ref castSenderEndPoint );
   senderEndPoint = (IPEndPoint) castSenderEndPoint;
   Console.WriteLine("Received {0} bytes from {1}", rc, senderEndPoint.ToString());
}
catch ( SocketException err )
{
   Console.WriteLine("Error occurred: {0}", err.Message);
}
finally
{
   udpSocket.Close();
}
```

6.6. Closing socket

```
Socket   tcpSocket;
Byte [ ]  receiveBuffer = new byte [ 1024 ], requestBuffer = new byte [ 1024 ];
int      rc;
```

```
// Establish a TCP connection, such that tcpSocket is valid
try
{
   // Initialize the requestBuffer and send request to server
   tcpSocket.Send( requestBuffer );
   // Since this socket will not be sending anything shut it down
   tcpSocket.Shutdown( SocketShutdown.Send );
   while (1)
   {
      rc = tcpSocket.Receive( receiveBuffer );
      if ( rc > 0 )
      {
         // Process data
      }
      else if ( rc == 0 )
      {
         tcpSocket.Close();
         break;
      }
   }
}
catch ( SocketException err )
{
   Console.WriteLine("An error occurred: {0}", err.Message);
}
```

7. Advantage of Using a Connection Oriented Protocal Such As TCP-

a) Reliability of packet arrival
b) No loss of data
c) Acknowledgement for packet sending and receiving

8. Disadvantage of Using a Connection Oriented Protocol Such As TCP-

a) Packet acknowledgement may add overhead
b) packets are not tagged with sequence numbers
c) Loss or duplication of data packets is more likely to occur
d) The application layer must assume responsibility for correct searchers of the data packets

9. Application of Connection Oriented Programming

- World Wide Web (Apache TCP port 80),
- e-mail (SMTP TCP port 25 Postfix MTA),
- File Transfer Protocol (FTP port 21) and Secure Shell (OpenSSH port 22) ,
- Online Chatting (global, local) etc.

10. Conclusion

The code provided for both the EchoServer and EchoClient are written for the Windows platform, using the Winsock Library. You can use any compiler that provides the windows libraries, like the Microsoft VC++ compiler,

or the CodeWarrior IDE. The usage of the IDE itself is clearly explained in the tutorials that come with CodeWarrior or MSVC++. In either case, you need to open a new project, and add the given source files to the project, and use the build option for getting the executables for both the Client and Server

References

[1]. Cisco Networking Academy Program, CCNA 1 and 2 Companion Guide Revised Third Edition, P.480, ISBN 1-58713-150-1
[2]. www-306.ibm.com - AnyNet Guide to Sockets over SNA
[3]. books.google.com - UNIX Network Programming: The sockets networking API
[4]. books.google.com - Designing BSD Rootkits: An Introduction to Kernel Hacking
[5]. Wikipedia: Berkeley sockets 2011-02-18
[6]. Goodheart 1994, p. 11
[7]. Goodheart 1994, p. 17
[8]. Wikipedia: Transport Layer Interface 2011-02-18