Part 1: Investigative Research
To investigate Socket Programming and its application

Abstract:

Using the concept of sockets, communication between hosts and processes within the same computer are allowed. Depending on the operating systems implementation, support can be provided that can enable different I/O devices and drivers to work together. All modern operating systems have some sort of implementation for a socket interface as socket interface has become one of the fundamental technologies underlying the internet. If hardware manufacturers start supplying built-in codes for socket programming, communication between hosts will occur a lot faster then it occurs today as we all know hardware implementation is a lot efficient then software implementation.

This report is presented after I conducted a research in order to investigate into socket programming and its applications.

Introduction:

Many papers and reviews have been published that have investigated into as to what socket programming is. As to start off with our investigation this report contains the history and background of socket programming, clearly mentioning what sockets are. This report will also mention the application of different types and ways of socket programming encompassing its functions, how it works and why is it so useful.

History and Overview of Socket Programming:

Sockets were developed in 1981 in the University of California, Berkeley. The project was funded by ARPA (Advanced Research Projects Agency) in 1980. At the time when they were originated they were called the Berkeley Sockets and worked with the 4.2BSD UNIX Operating System released in 1983. These are also known as the BSD (Berkeley Software Distribution) socket API (Application Programming Interface). The main objective for its development was the transport of TCPIP software to UNIX. At first they only worked as an API up until 1989 when UC Berkeley released new versions of its operating system and networking library that was free from the licensing constraints of UNIX protected by the AT&T's copyright. However In 1986, AT&T had introduced the Transport Layer Interface (TLI) with socket-like functionality although it was more networks independent. UNIX includes both TLI and Sockets after SVR4. With a programmers perspective TLI in theory is better than sockets in terms of stack independence **[1]**.

The Berkeley Socket API has formed a de facto standard for abstraction of network sockets. With a level of abstraction available it has become extremely useful and from the origins of

Berkeley UNIX it has moved to all breeds of UNIX, Windows and Macintosh **[1]**. Applications in C programming that perform inter-process communication across a network are based on library comprised in the Berkeley Socket API. It is the combination of sockets with internet that makes talking to arbitrary machines around the world easy and virtually at the same place **[2]**.

What is a Socket, why is it necessary and what does it do?

A socket is a software mechanism that has the ability to hide the programmer from the details of how data is actually transmitted **[3]**. By hiding these details the programmer is shielded from the low level details of the network, these can include; media types, packet sizes, packet retransmission, network addresses and many more. To further explain as to what sockets do and what they keep the user or programmer away from doing we can take help of a common transmission. In it data is transmitted across the network in the form of packets called datagrams that contain the header (address and port number of where the packets are going to) and payload (the actual data). As the datagrams are of finite length it is necessary to split the data into packets and then reassemble them at the destination. It is often that one or more packets are lost or corrupted during transmission, requiring the need of re-transmission. If this is not done the packets will arrive out of order. The sockets were therefore developed to keep track of this; splitting the data into packets, generating headers, parsing the headers of the incoming packets and keeping track of what packets that are received and yet to be received **[4]**.

As Socket is a communication mechanism it is used to implement in a client-server application **[5]**. A Server uses a passive socket for incoming connections while a client uses an active socket to initiate a connection **[3]**. During this the server process runs unattended and continuously acting as a host waiting for incoming connections, whereas the client process is directly or indirectly user driven. Socket mechanism first being introduced in the 4.2 BSD UNIX systems in conjunction with the TCP/IP protocol, therefore in UNIX systems Server processes are known as Daemon processes **[5]**. However in a Windows environment they are known as services **[6]**. A Server process can be initiated as part of the system boot up sequence. Even after a user logs out; a server process can be initiated by a user that it carries on running **[5]**.

Sockets and Operating Systems [1]:

- *BSD UNIX:* Sockets provide standalone and networked IPC services and are part of the kernel.
- *Windows:* Winsock, the Windows socket API is a multivendor specification to standardize the use of TCPIP under Windows, based on the Berkley sockets interface.
- MS-DOS, Windows, Mac OS and OS/2 provide sockets in the form of libraries.

Operations performed by every Socket [3]:

- Connect to a remote machine
- Send Data
- Receive Data
- Close a Connection

Additional Operations performed by a Server Socket [3; 5]:

- Bind to a port (i.e. associating a port number or a network address to the socket)
- Listening to incoming data and waiting for connection attempts
- Accept connections from a remote machine on the bound port

Fashion in which Client Socket operates [4]:

- New Socket is created
- Socket attempts to connect to the remote machine
- Transmission of data is full duplex between the Client and Server
- After the completion of data transmission each or both sides can close the connection. Different protocols differ in the way connections are closed for example HTTP requires connection to be closed after each request is serviced where as with FTP multiple request can be serviced using the same connection.

Programming Calls in Sockets [7]:

- Neither hosts need to be scanned nor there a need for DNS to be communicated by programmers.
- It is the C Library routines that each return a pointer to an object
- It is good programming practice to try to connect to each address listed in the IP numbers address list.
- The client tries to connect to each server until it gets an answer resulting in connection or it runs out of servers that are available.

Socket Addressing [7]:

- Communication takes place using host-port pairs that is a port on the server and the other on the client.
- The Socket address structure used for communication in arbitrary domains has two fields; one that defines the interpretation of the data – the address family field and the other known as the address field containing the data.

A socket address on the TCP/IP internet consists of two parts [3]:

- An internet (IP) address; a 32 bit number represented by 4 (or 6 for new) decimal numbers - a unique identifier for a network interface card within an administered domain. A TCP/IP Host may have as many addresses as it has network interfaces.
- A 16 bit port number; that acts an entry point to an application residing on a host. Ports define entry points for services provided by server applications. Important commercial applications such as Oracle have their own well known ports.

File Descriptors [6; 8]:

- File descriptors are numbers used for I/O usually used during opening and creating calls **[7]**.
- Four groups of numbers required for Socket Mechanism to communicate **[5]**:
  1. Identification Number or address of the remote host
  2. Port number of the remote host
  3. Identification Number or address of the local host
  4. Port number of the local host

Blocking and Non Blocking Sockets:

Sockets that do not return data until the sending or receiving of all the data is done are known as blocking sockets the other is the non blocking which allows the return even if data is not completely send or received from that operation. The major problem caused by this is that when a socket continues to listen, the program may hang due to socket waiting for the data that has never arrived. The fastest sockets code uses the non blocking sockets. Sockets can be set to blocking or non blocking modes using different functions depending on the syntax of the language, it's relatively easier to change modes in other languages compared to C where it is quite complex **[2]**.

Socket Functions:

These functions are as described and used by Sockets [5]:

Socket() [8]:

This function is used to create sockets; the way it works is that it takes in three arguments to create an endpoint for communication and to return a descriptor. The three arguments are:

1. Domain: specifying the protocol family of the created socket
2. Type: whether its stream-oriented, datagram or raw socket.
3. Protocol: the kind of protocol is used TCP, UDP or IP relative to the type.

Select():

To find out which sockets are active, in an application using several sockets. These sockets can be the ones either ready for incoming data or any outstanding exceptional conditions. The function not only returns the active sockets but also the number of the largest descriptor to be checked and how long is the wait for the selection to be completed.

Connect():

This is used in connection oriented client sockets in order to establish a connection with a remote host. For connectionless sockets connect() means a default target for the sending receiving of data, allowing the functions of send() and receive(). A connect function returns error code in terms of numbers; 0 for success and -1 for error **[9]**.

Bind():

It is used for incoming connection requests. After the socket is created, it is given an address family, however to assign an address the bind function is required. It is necessary for a socket to be bound before it may accept incoming connections. 0 is returned on success and -1 if an error occurs **[10]**.

Listen():

This type of function is only applicable to stream and sequential socket types. The purpose of this function is to prepare a bound socket for accepting incoming connections **[11]**.

Accept() and Close():

The accept function accepts a connection request from a remote host, with acceptance success and failure status denoted by 0 and -1 **[12]**.

In order to release the resources allocated by the socket, the system waits for a close() call to occur. This is highly important when a connect() call fails or is retried. Each close() call must be associated with its matching socket() call **[13]**.

Types of Socket Programming:

Stream Sockets

Before mentioning what stream socket programming is it is better to mention what a stream is; it is any flowing sequence containing characters into or out of a process. A stream can either be an *input-stream* - attached to some input source for the process such as a socket into which characters flow from the Internet, or an *output-stream* - attached to some output source for the process, such as a socket out of which characters flow into the Internet **[14]**.

Stream Sockets are also known as *Connection Oriented Sockets* are reliable two-way connected communication streams. Say if two items are output into the socket in the order "A, B", they will arrive in the order "A, B" at the receiving end. The transmission is error free. Any errors that are encountered will be due to the way they were output **[15]**.

Connection-Oriented Stream Socket Paradigm [16]:

Server	**Client**
Create a socket with `socket` function	Create a socket with `socket` function
Bind the socket to a port with `bind` function	
Set up a connection queue with `listen` function	
Establish a connection with `accept` function	Request a connection with `connect` function
Read and write data with `read` and `write` function	Read and write data with `read` and `write` functions

Protocol Used: to achieve this high level of data transmission quality stream sockets use the Transmission Control Protocol, known as "TCP" – a session based protocol **[3]**. It makes sure that the data arrives sequentially and error-free **[15]**.

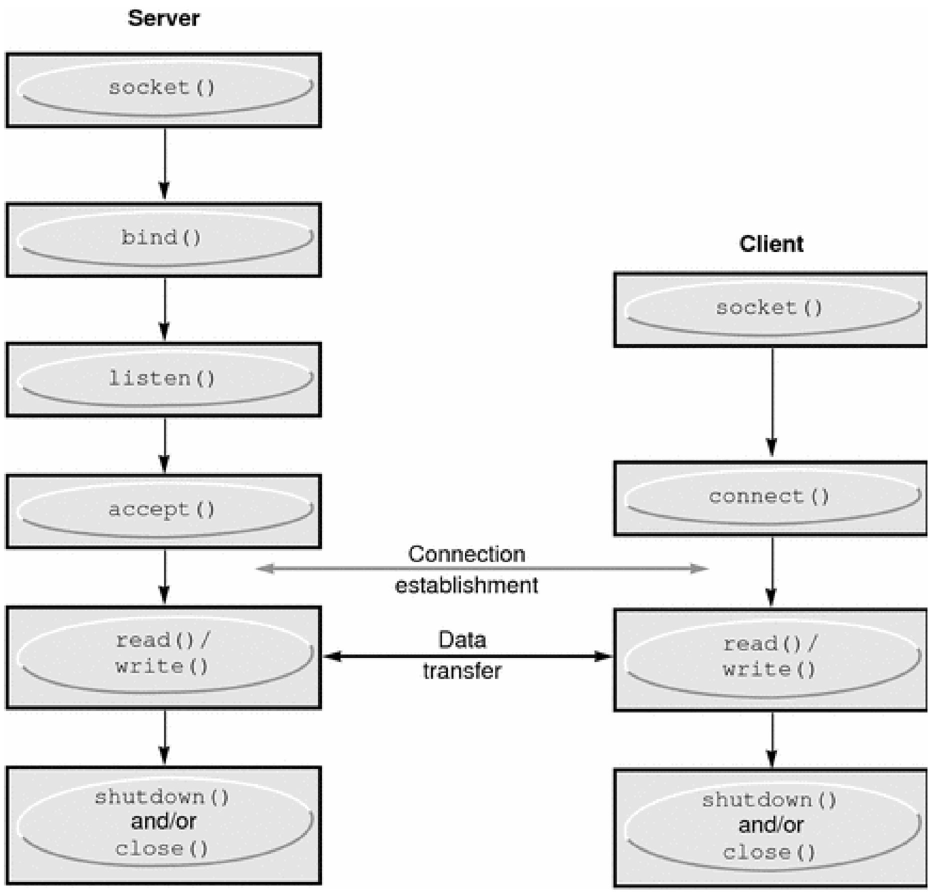


Figure1.1 above showing Stream Socket operation [17]:

There are two types of client-server applications [14].

- *RFC Implementation:* In the first a client-server application that is an implementation of a protocol standard defined in an RFC. For such implementations, the client and server programs must conform to the rules set as standard by the RFC. Say if a client program is an implementation of the FTP client and the server program an implementation of the FTP server. It is necessary for both programs to be compliant with the RFC standards, even though they are programmed by different developers. If they fail to be compliant interpretability issues will arise. A client or server program that implements a protocol defined in an RFC, it should use the same port number associated with the protocol. There is an advantage in this as independent developers create their own client and server applications that should be able to communicate with each other for e.g. an Apache Web Server providing information to a Netscape Browser or even a FTP Client on a PC uploading files to a UNIX FTP Server.
- *Proprietary Application:* The other sort of client-server application is a *proprietary* client-server application. In this implementation the client and server programs do not necessarily conform to any existing RFC standard. A single developer can create both

the client and server programs, having complete control over whatever is in the code, however the code does not implement as a public-domain protocol, thus other independent developers will not be able to develop code that can interoperate with that application. During development of a proprietary application, the developer should not to use one of the ‘well-known’ port numbers defined in the RFCs as that might result not only in legal compellability but also technical difficulties during running in a live environment.

Applications that use Stream Sockets [1]: Applications with large data capacity and that require services for a persistent connection. Such applications can be web browsers that use the HTTP protocol which uses stream sockets on port 80 to get pages. Other common examples may include FTP that uses port 21, POP3 using port number 110 and Telnet that uses the port number 23.

Datagram Sockets

They are also known as Connectionless Sockets or transmit and pray protocols **[1]**. They are simple however unreliable. In a transmission sending a datagram, it may arrive, however it may arrive out of order. If it arrives, the data within the packet will be error-free. It is known as connection less as the need to maintain an open connection like that in stream sockets is not required. All that is required is the IP header to be attached along with the destination information before sending it to the other end, requiring no prior connection **[15]**.

Protocols Used: Datagram sockets do not use TCP and instead User Datagram Protocol or "UDP" along with IP that is used for routing. It may sound that UDP is worthless compared to the three way handshake connection **[14]** and the error control provided in the TCP’s Stream Socket, however UDP is used as it is much faster due to lack of overheads and for some applications its far better **[16]**. For example the tftp protocol that sits on the UDP protocol asks for an acknowledgement every time a packet is sent. If the acknowledgment is not received in the allocated time the packets are resent. This acknowledgement procedure is very important while implementing socket applications **[15]**.

Connectionless Datagram Socket Paradigm [16]:

Server	**Client**
Create a socket with `socket` function	Create a socket with `socket` function
Bind the socket to a port with `bind` function	Bind the socket to a port with `bind` function
Send and receive data with `recvfrom` and `sendto` functions	Send and receive data with `recvfrom` and `sendto` functions

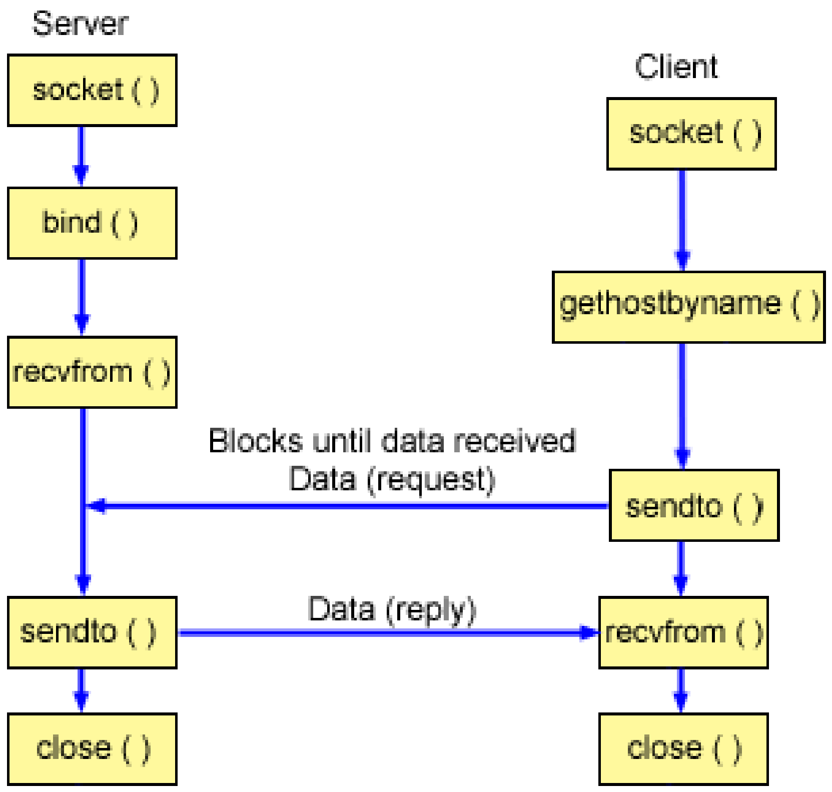


Figure1.2 above showing Datagram socket operation [18]:

Applications that use Datagram Sockets: an everyday example can be applications using real-time multimedia streaming **[16]**; even if 1 or 2 frames are missed it won't be noticed as at an average the streaming is at 25 frames per second. With TCP's error prevention it will slow down the transmission producing pathetic frame rate **[16]**. UDP sockets are usually required for quick DNS lookups on port 53; Windows shared printer name lookup on port 137 or even for single use query reply actions. They are also good for Broadcast applications, where the name of recipient is unknown **[1]**.

Raw Sockets:

Sockets that on incoming and outgoing packets allow access to packet headers using IPv4 protocols are called Raw Sockets. These are usually used on the Transport and network layer. Unlike non-raw sockets that discard the header and only receive the payload, raw sockets always receive the packets with headers included. Raw sockets are part of the underlying operating system's networking API and not a programming language level construct. These types of sockets are needed for either new protocols or those with no user interface like the internet control message protocol or the ICMP. It is supported by most socket interfaces like the BSD, Linux 2.2 and Windows socket interface **[19]**. In order to use raw sockets in UNIX it is mandatory to have the root authority. Packets that the kernel does not explicitly support use Raw Sockets to generate and receive. In order to send or receive over raw sockets the following command needs to be typed **[1]**.

```
s=socket(AF_INET,SOCK_RAW,[protocol])
```

Raw Socket Example: Ping is a fine example of raw sockets. The way it works is by sending out the 'ICMP'. To respond to the ping packets the kernel has a built in code, although it does not have a code to generate these packets. Over the raw socket the ping packet generator formats an ICMP echo packet sends it waiting for a response **[1]**.

Malicious Activities Using Raw Sockets: denial of service and IP address spoofing are some of the attacks performed by raw sockets as users are allowed to craft packet headers themselves. Windows Xp faced a Distributed Denial of service of attack due to the presence of the implemented Winsock interface with raw sock support, however to overcome Microsoft released a non-removable hot-fix, disabling Winsock's raw socket support. It was considered not wise by computing fellow for Microsoft to include full raw socket support in Home Edition of the Windows Xp OS platform. With the removal of this support it posed great problems for Microsoft as it broke compatibility with applications that were using them before the Blaster worm that led to Distributed Denial of service attack **[20]**.

Sequenced Packet Sockets:

It is a connection oriented socket quite similar to the stream sockets. The main difference between the two is that Sequenced Packet Sockets maintain record boundaries by using the SOCK_SEQPACKET type. This means that it can send a record using one or more output operations and receive one or more input operations, however a single operation never transfers parts of more than one record. This makes sequenced packet sockets more programmer friendly with strong reliability features of Stream sockets and by record demarcation features at the protocol level like that of UDP in Datagram Sockets **[21]**.

Part 2: Practical Investigative Work
Multiplication Server - A Client-Server Java Application Using Socket Programming to Communicate between the two.

Background of the Program:

The program documented in this section is Client-Server Application called the **Multiplication Server**. The program is developed using Java language enabled with Sockets. This program is developed to show the functionality of Sockets Programming, that is how clients can communicate with the server; therefore the program developed is quite simple and there might not be an implementation in a real life scenario. All programs that use Sockets Programming use the similar concept of communication and if someone can get familiarized with this, then the development of complex or even mission critical programs is no big deal depending on the knowledge of programming syntax skills in a language. The purpose of the program is to show how a remote machine can communicate with a host.

The functionality of the program is to receive 2 integers from a remote machine or a client and send it to the host or the server to perform multiplication on it. The product of the two integers is returned and displayed on the clients interface. The two integers received and their product can be seen on the server's interface aswell.

As this program demonstrates the principles required for a client server environment to operate effectively, with a little modification at both the server and client end, the sockets code within this program can be used for more complex operations like accessing 'views' in databases at remote locations or even printing using a shared printer at a remote location.

Programming Language Used in the Program:

The programming Language used in order to perform our required operation is Java as mentioned earlier. There are a number of reasons for using Java over others;

- Simple: Unsafe features like pointers present in C and C++ are removed from java that required the need to free memory. This caused 50% of the common bugs.
- Object Oriented: Everything in java is either a; class, method, or an object. There is an extensive class library to interface with the host operating system and the network
- Networkable: With the TCP/IP capabilities built in allow easy access over the network.
- Platform Independence: Java can run on any Operating System as long as the java virtual machine software is installed on the machine. The code is complied faster than C or C++.

- Security: java has features like encapsulation and setting methods as private that ensure the methods are isolated and integrity is maintained. Absence of pointers prevents illegal access to the memory.

Type of Socket Programming Used:

As mentioned in Part 1- The Investigative Research, that there are multiple ways of sockets programming. I will be using the Connection Oriented type or Stream Sockets in our investigative practical implementation. The main purpose of this that I would like to show how a connection is made before the transmission takes place as its importance is already mentioned in Part-1. Connectionless Sockets or Datagrams do not require a connection to be established before transmission takes place therefore will not be used. Raw sockets use the Internet Protocol and as our application will not be using this therefore it will not be used aswell.

Creating the Server:

The distinctive features the Server application has as compared to the Client are that the server:

1. can associate a port number to it for transmission to take place with clients
2. waits and listens for any incoming data coming from the client
3. accepts connections coming from a client or remote machine on the associated port

The Server application used in our program is shown below.

The Import packages required in order to run our server:

```
import java.net.*;
import java.io.*;
import javax.swing.*;
```

The Server application inherits the Java GUI features to implement GUI in the Server application:

```
public class MultiplicationServer extends JFrame
{
```

Two attributes are declared one for the text to be displayed on the server side and the other to hold the port number that will be associated or bound to the server.

```
  private JTextArea textWindow = new JTextArea();
  private int VirtualConnectionPort;
```

The constructor below configures the GUI frame loads the Server by calling a private method called LoadServer().

```
 public MultiplicationServer(int VirtualConnectionPortIn)
{
   VirtualConnectionPort = VirtualConnectionPortIn;
   setTitle("Multiplication Server");
   add("Center",textWindow);
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   setSize(400, 300);
   setVisible(true);
   LoadServer();
}
```

The LoadServer():
This private method contains variables required to execute the functionality of the server application.

```
private void LoadServer()
{
```

The first variable we declare is to create a general socket

```
Socket connection;
```

The second variable is to create a socket with the distinctive features of a server mentioned above:

```
ServerSocket listenport;
```

As we are using stream sockets we need to handle low level communication that takes place in the form of bytes aswell as high level communication that allows types like integers, strings and characters.

To handle low level input object we declare

```
InputStream inStream;
```

To handle high level input objects we declare

```
DataInputStream inDataStream;
```

Likewise low and high level output objects are declared

```
OutputStream outStream;
DataOutputStream outDataStream;
```

Creating sockets and declaring the above objects and variables were required for a server socket to perform its function, however as our application is a multiplication server we further need to declare variables that will perform the function of executing the request from the client, we therefore declare:

```
    String client;
    int integer1, integer2, product;
    boolean connected;
```

For the server to accept connections from a client we start an infinite loop which lasts until the server application is terminated. The way connections are accepted is in a FIFO fashion one at a time. In order for a client to be connected to the server, the server has to be free at that time waiting and listening on the port *'VirtualConnetionPort'* to accept connection from a client. If the server is already engaged with a client, a second client cannot connect.

```
    while(true)
    {
      try
            {
```

To start accepting connection requests we first need to create a server socket that is bound to a port.

```
                listenport = new ServerSocket (VirtualConnectionPort);
            textWindow.append("Listening for client to connect on the Virtual Connection
Port " + VirtualConnectionPort + "\n");
```

Next we need to assign a connection variable to the object of the socket class returned from the accept() method
The Boolean variable *connected* is then set to true.

```
                connection = listenport.accept ();
                connected = true;
```

To provide stream coming in and going out of the server an object needs to be returned from the **Stream* class.
To create an input stream from the client

```
                inStream = connection.getInputStream();
                inDataStream = new DataInputStream(inStream);
```

To create an output stream to the client

```
                outStream = connection.getOutputStream ();
                outDataStream = new DataOutputStream ( outStream );
```

To wait for a string from the client to establish connection between the client and the server

```
        client = inDataStream.readUTF();
                textWindow.append("Connection established with "
                                + client + "\n" );
```

While the connection is established between the client and server, a client can continue to send integers to the server for multiplication. The integers received are displayed on the server interface.

```
                while(connected)
                {
```

Receiving the first integer from the client

```
                        integer1 = inDataStream.readInt();
                textWindow.append( "First number received: "
                                + integer1 + "\n");
```

Reading the second integer from the client

```
                        integer2 = inDataStream.readInt();
                textWindow.append( "Second number received: "
                                + integer2 + "\n");
```

Multiplying the two integers received from the client:

```
                        product = integer1 * integer2;
                        textWindow.append( "product returned: "
                                + product + "\n");
```

After the multiplication has been performed at the server end the product is sent to the client

```
                outDataStream.writeInt (product);
                  }
                }
```

If whenever the connection is lost the server interface has to notify and to do this an IO exceptions is included.

```
        catch (IOException e)
                {
                        connected = false;
                }
              }
  }
}
```

To run the server application a main method needs to be implemented.

```
public class RunMultiplicationServer
 {
        public static void main (String[] args)
        {
```

To associate a port number to the Multiplication Server

```
                new MultiplicationServer(8800);
        }
}
```

Creating the Client:

Features performed by a socket in a client application are those performed by all sockets in both clients and servers.

1. Connect to a host
2. Send data
3. Receive data
4. Close connection

The Import packages required in order to run our server:

```
import java.net.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

The Client application inherits the Java GUI features to implement GUI in the Server application:

```
public class MultiplicationClient extends JFrame implements ActionListener
{
```

To display the visual components the following is declared:

```
  private JTextField value1 = new JTextField(5);
  private JLabel plus = new JLabel("*");
  private JTextField value2 = new JTextField(5);
  private JLabel equals = new JLabel("=");
  private JLabel product = new JLabel();
  private JTextField msg = new JTextField(10);
  private JButton addButton
          = new JButton("Get Result");
```

To declare low and high level input objects\

```
  private InputStream inStream;
  private DataInputStream inDataStream;
```

To declare low and high level output objects

```
  private OutputStream outStream;
  private DataOutputStream outDataStream;
```

To declare a socket

```
  private Socket connection;
```

To declare attributes that hold the address of a Server and the Virtual Connection Port

```
  private String Server;
  private int VirtualConnectionPort;
```

A constructor is implemented to accept strings and integers that is assigned to the remote machine and to VirtualConnectionPort

```
  public MultiplicationClient(String ServerIn, int VirtualConnectionPortIn)
  {
     Server = ServerIn;
     VirtualConnectionPort = VirtualConnectionPortIn;
```

To add the visual components to the client application

```
     add(value1);
```

```
    add(product);
    add(value2);
    add(equals);
    add(product);
    add(msg);
    add(addButton);
```

Configuring the frame

```
    setLayout(new FlowLayout());
    setTitle("Multiplication Client");
    msg.setHorizontalAlignment(JLabel.CENTER);
    addButton.addActionListener(this);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(250, 150);
    setLocation(300,300);
    setVisible(true);
```

Method that initiates the private method initiateClient

```
    initiateClient();
  }
```

The initiateClient method creates a new socket containing attributes of the host name and the port number requests response from the host in order to establish connection with the server.

```
 private void initiateClient()
 {
   try
   {
      connection = new Socket (Server, VirtualConnectionPort );
      msg.setText("Connected");
```

To create an input and output stream from and to the server the following code is added

```
        inStream = connection.getInputStream();
        inDataStream = new DataInputStream (inStream);

        outStream = connection.getOutputStream();
        outDataStream = new DataOutputStream(outStream);
```

In order to send the IP address to the server

```
         outDataStream.writeUTF(connection.getLocalAddress().getHostAddress());
      }
```

If the server we are trying to connect to is an unknown host we include an exception to notify us

```
      catch (UnknownHostException e)
      {
           msg.setText (" Host Unknown");
      }
```

To check if there is server running or not as a host we add an IO exception

```
      catch (IOException except)
      {
        msg.setText ("Check Server Availability");
      }
 }
```

The actionPerformed method determines the step taken when we send our request to the server.

```
 public void actionPerformed(ActionEvent e)
 {
    try
    {
```

To send the two integers to the server

```
        outDataStream.writeInt(Integer.parseInt(value1.getText()));
        outDataStream.writeInt(Integer.parseInt(value2.getText()));
```

Printing the received results from the server and looking for any error through an IO exception

```
        int result = inDataStream.readInt();
        product.setText("" + result);
    }
    catch(IOException ie)
    {
        ie.printStackTrace();
    }
 }
}
```

To run the Client application we need a main method

```
public class RunMultiplicationClient
{
           public static void main (String[] args )
           {
```

To specify the name of the host or server and the number of the virtual connection port. In this case the host name is “Inspiron” where as the port number is 8800

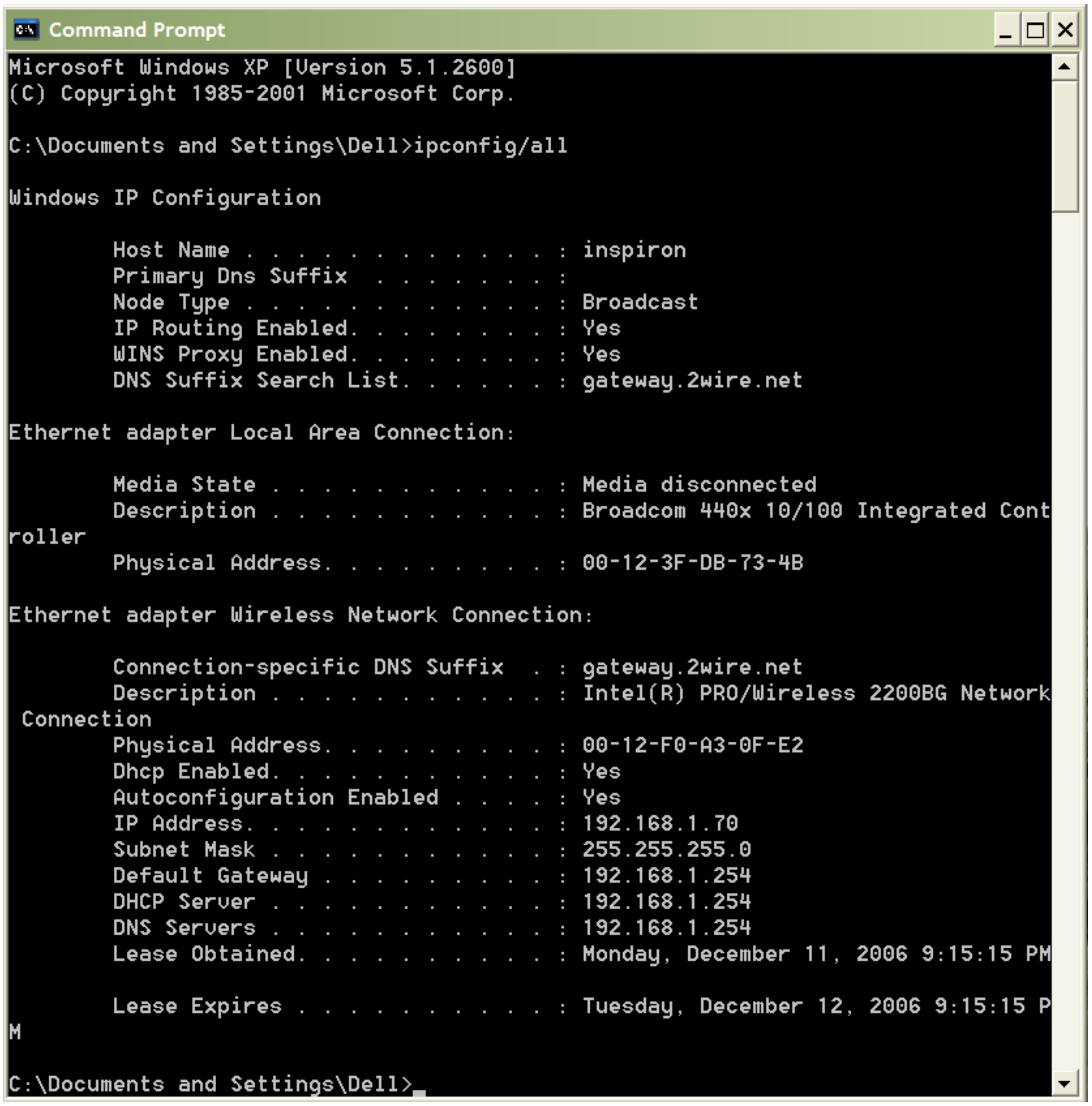
```
                      new MultiplicationClient("Inspiron", 8800);
           }
}
```

Connecting the Client to the Server

There are multiple ways of running the Client and Server application confirming the connection and successfully performing the intended operation. Both the client and server applications can run on the same machine, all we have to do is make sure that the port numbers in both applications are same and that the correct host name is mentioned in the main method responsible for running the client application.
We will however implement our program using two different machines one acting as a server whereas the other acting as a client containing the required classes respectively. Both machines will be connected to the same local area network using a wireless connection. As stream sockets use TCP as a protocol we have a choice of either using the wired or wireless media leaving us free from any complex connectivity implementations. This is because the wired media uses Ethernet- 802.3 as a protocol standard provided by the IEEE, where as if we use WIFI- 802.11 the wireless Ethernet in other words, also a standard provided by the IEEE, both use the TCP to communicate.
The two machines will be connected through a wireless router. The Operating System on both machines is Microsoft Windows XP.

Checking the two machines are connected to the same wireless network

We have to make sure that both machines are connected to the same wireless network, for this we can type the *ipconfig/all* command in the command prompt. See Figure 2.1.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Dell>ipconfig/all

Windows IP Configuration

        Host Name . . . . . . . . . . . . : inspiron
        Primary Dns Suffix  . . . . . . . :
        Node Type . . . . . . . . . . . . : Broadcast
        IP Routing Enabled. . . . . . . . : Yes
        WINS Proxy Enabled. . . . . . . . : Yes
        DNS Suffix Search List. . . . . . : gateway.2wire.net

Ethernet adapter Local Area Connection:

        Media State . . . . . . . . . . . : Media disconnected
        Description . . . . . . . . . . . : Broadcom 440x 10/100 Integrated Cont
roller
        Physical Address. . . . . . . . . : 00-12-3F-DB-73-4B

Ethernet adapter Wireless Network Connection:

        Connection-specific DNS Suffix  . : gateway.2wire.net
        Description . . . . . . . . . . . : Intel(R) PRO/Wireless 2200BG Network
 Connection
        Physical Address. . . . . . . . . : 00-12-F0-A3-0F-E2
        Dhcp Enabled. . . . . . . . . . . : Yes
        Autoconfiguration Enabled . . . . : Yes
        IP Address. . . . . . . . . . . . : 192.168.1.70
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.1.254
        DHCP Server . . . . . . . . . . . : 192.168.1.254
        DNS Servers . . . . . . . . . . . : 192.168.1.254
        Lease Obtained. . . . . . . . . . : Monday, December 11, 2006 9:15:15 PM

        Lease Expires . . . . . . . . . . : Tuesday, December 12, 2006 9:15:15 P
M

C:\Documents and Settings\Dell>_
```

Figure 2.1 displays the description of the Ethernet adapter, default gateway, the subnet mask, IP address of the machine, the Host Name and the DNS details.

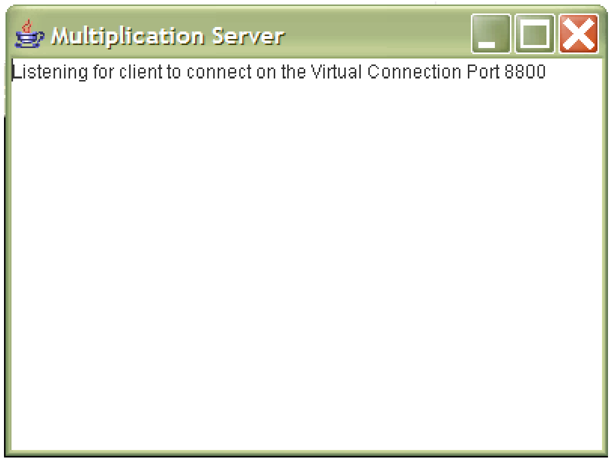
Things we cannot over look:

- We have to make sure that both machines display the same DNS, subnet mask and default gateway details, thus confirming they are using the same wireless router and connected to or are from the same network. If the details do not match please check the network connections. The TCP/IP details should be configured similarly, however if the setting is kept to automatic detection it works perfectly.
- Both machines should be WIFI enabled or at least one of them. Check the description of the Ethernet adapter. There will be no compatibility issues if one machine is connected using Ethernet and the other using WIFI.
- The corresponding application classes should be present on each machine
- Java complier should be installed on both machines
- The client application should contain the correct host name and both applications should have the same port number.
- The server should be running and listening on port when the client is initiated. The client will not operate unless the server is running.
- The server accepts requests from 1 client at a time, therefore if it is flooded with requests it will crash. It can honour requests from other clients after the first has closed the connection.

Running the Multiplication Server

Depending on the way we compile and execute our java classes, we execute our classes. JCreator program is used to handle our classes, compile them and execute them.

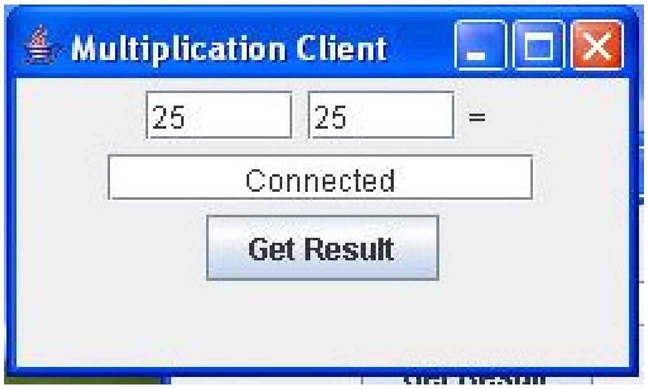
Running the Server Application:
On our server or host machine we execute the server application, displaying us the following GUI window. The screen shows us that the server is waiting and listing for an incoming connection from a client on port number 8800. See Figure 2.2.



```
Listening for client to connect on the Virtual Connection Port 8800
```

Figure 2.2

Running the Client Application:
Execute the client application on the client machine; the following screen appears on the Client machine. Connected appearing in the screen shows that the client is connected to the server. Here in the example we have entered 25 in both fields to send to the server for multiplication. See Figure 2.3.



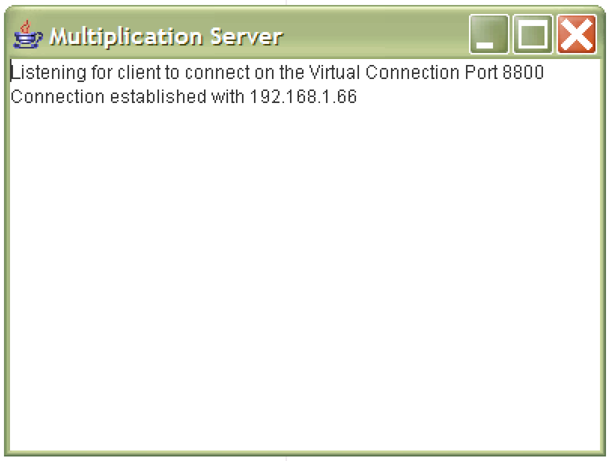
25 25

Connected

Figure 2.3

Connection established between the client and the server

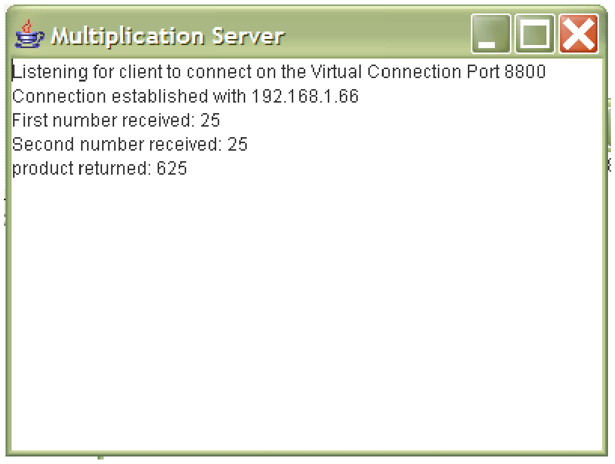
As the client class is executed it establishes a connection with the server. The IP address of the client is displayed on the server machine. See Figure 2.4.



```
Listening for client to connect on the Virtual Connection Port 8800
Connection established with 192.168.1.66
```

Figure 2.4

Input coming from the client to the server:
After the Get Result button is pressed on the client machine the server receives numbers from the client, they are displayed on the server interface along with the calculated product. See Figure 2.5.

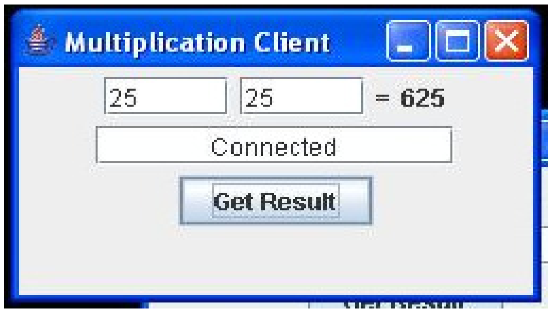


```
Listening for client to connect on the Virtual Connection Port 8800
Connection established with 192.168.1.66
First number received: 25
Second number received: 25
product returned: 625
```

Figure 2.5

Product received on the client side:

The product is received back by the client after the calculation is performed at the server end. See Figure 2.6.



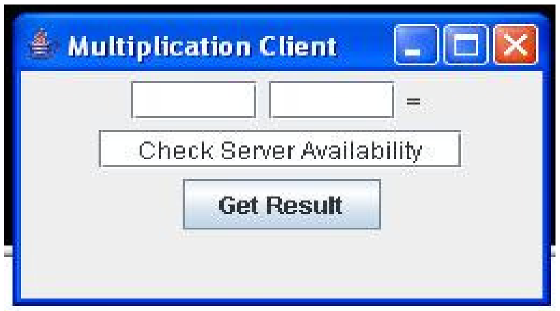
25 25 = 625

Connected

Figure 2.6

Handling Exceptions:

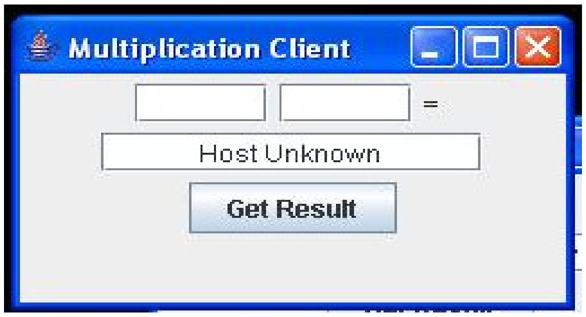
If the server is not available the client notifies the user. See Figure 2.7.



Check Server Availability

Figure 2.7

If the host is unknown on the network the client application notifies the user. See Figure 2.8.



Host Unknown

Figure 2.8

Conclusion:

In this report an attempt was made to investigate in to sockets programming and its application. Due to the constraint of length and scope of the report socket programming cannot easily be summarized. There is work done on socket programming ever since its advent. This is proved as Windows, Macintosh and almost all versions of UNIX provide interoperability with the sockets interface. Java is taking over sockets programming as it is taking away its market share in other aspects of computing, however socket programming started with C language and because C provides unmatched portability, unlike java with a lot of classes, C has yet not lost its popularity. Keeping all of these aspects on the table I believe socket programming has yet to evolve in its types and the ways it will perform in applications. All this will require the thought to have faster and reliable transactions between the server and clients rather than who leads in profiting from them.

References:

(1) *Introduction to Sockets*: web.njit.edu/~gblank/cis604/Lectures/**604Sockets**.ppt

(2) *Socket Programming HOWTO*; Author: Gordon McMillan, http://www.amk.ca/python/howto/sockets/

(3) *Java in two semesters*, 2nd edition, by Quentin Charatan and Aaron Kans, McGraw Hill Publication, 2006

(4) *Java Network Programming* by Elliotte Rusty Harold, O'Reilly and Associates Publication, 1997.

(5) *Sockets Programming*; Author Peter Burden, http://www.scit.wlv.ac.uk/~jphb/comms/sockets.html

(6) *Principles of Network and System Administration* by Mark Burgess, 2nd Edition, John Wiley and Sons Publication, 2004

(7) *An Introduction to Socket Programming*; Author: Reg Quinton (reggers@julian.uwo.ca); http://www.uwo.ca/its/doc/courses/notes/socket/

(8) *socket(2) - Linux man page*: http://www.die.net/doc/linux/man/man2/socket.2.html

(9) *connect(2) - Linux man page*: http://www.die.net/doc/linux/man/man2/connect.2.html

(10) *bind(2) - Linux man page*: http://www.die.net/doc/linux/man/man2/bind.2.html

(11) *listen(2) - Linux man page*: http://www.die.net/doc/linux/man/man2/listen.2.html

(12) *accept(2) - Linux man page*: http://www.die.net/doc/linux/man/man2/accept.2.html

(13) *close(2) - Linux man page*: http://www.die.net/doc/linux/man/man2/close.2.html

(14) *Computer Networking- A Top-Down Approach featuring the Internet*; 2nd Edition by James F. Kurose and Keith W Ross; Addison Wesley World Student Edition.

(15) *Beej's Guide to Network Programming Using Internet Sockets*; Author Brian "Beej" Hall; Dated: November 5th 2005; http://beej.us/guide/bgnet/output/html/theory.html

(16) *Berkeley Sockets Part I, Overview; Author: Mills Schmidt*; Dated: 2004; http://www.eecs.case.edu/courses/eecs338/spring2004/CParker/R9.html#overview

(17) *Stream Sockets Diagram from Sun's Website*: http://docs.sun.com/source/817-4415/images/7099.gif

(18) *Datagram Socket Diagram from IBM's Website*: http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=/rzab6/rzab6connectionless.htm

(19) *Raw(7)- Linux man page*: http://www.die.net/doc/linux/man/man7/raw.7.html

(20) *The Gibson Research Corporation- Denial of service, Investigation and Exploration pages*; Last Modified: 2005; http://www.grc.com/dos/intro.htm

(21) *Sequenced Packets Over Ordinary TCP*: http://urchin.earth.li/~twic/Sequenced_Packets_Over_Ordinary_TCP.html

Bibliography:

- Socket Concepts: http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=/
- Wikipedia: http://en.wikipedia.org/wiki/Main_Page
- Foldoc; Free Online Dictionary of Computing: http://foldoc.org/
- Concise Dictionary of Computing; Penguin Reference
- Java networking and communications by Todd Courtois . - Upper Saddle River, N.J. : Prentice Hall PTR, 1998 .
- Internetworking with TCP/IP . - Vol.1 : Principles, protocols, and architecture by Douglas E. Comer . - 3rd ed . - London : Prentice-Hall International, 1995 .
- Comer, Douglas Internetworking with TCP/IP . - Vol.2 : Design, implementation, and internals by Douglas E. Comer and David L . - Englewood Cliffs : Prentice-Hall; London; Prentice-Hall International, 1991 .
- Comer, Douglas Internetworking with TCP/IP by Douglas E. Comer and David L. Stevens . - BSD socket version . - Vol.3 : Client-server programming and applications . - Englewood Cliffs; London : Prentice Hall, 1993 .
- Socket Programming; the 10% you need- for the 90% of your work; Author: Steve Litt; Dated: 2001; http://www.troubleshooters.com/codecorn/sockets/
- Sockets Tutorial: http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html
- Sockets Programming in Java: A tutorial; By Quasy H. Mahmoud, 12/11/96 http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html
- Free BSD Developers' Handbook; Chapter 7 Sockets; Contributed by G.Adam Stanislav: http://www.freebsd.org/doc/en_US.ISO8859-1/books/developers-handbook/sockets.html