Lab#1 – Client / Server Communications

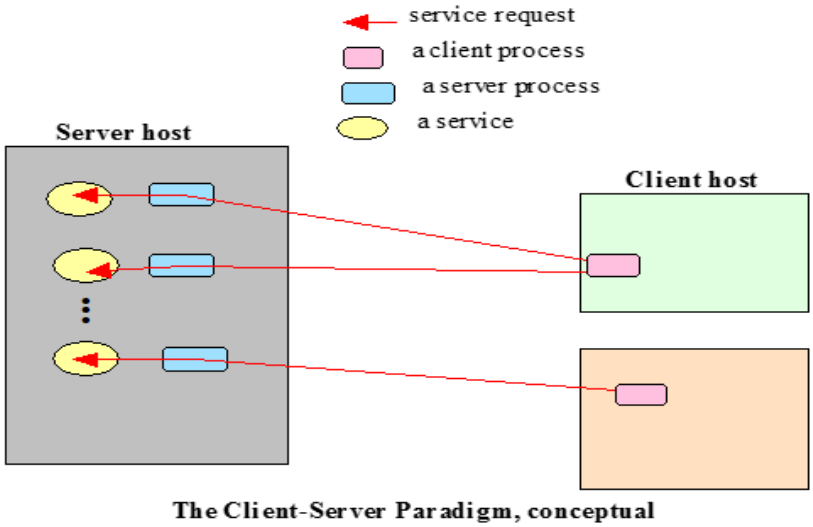
Objective

In this lab you will modify and extend the sample programs of client server communications given out in Lab#0. You will demonstrate your running programs to the TA. You will also write a very short lab report containing the answer to a question in Task#2. Submit the lab report on Blackboard, the deadline is the date of Lab#1.

Background information

A. The Client / Server Communication Model

The client / server communication model is the most prevalent distributed communication model. It is service-oriented, and employs a request-response protocol. A server process, running on a server host, provides access to a service. A client process, running on a client host, accesses the service via the server process. The interaction of the process proceeds according to a protocol.



The Client-Server Paradigm, conceptual

A protocol/service session

In the context of the client-server model, we will use the term **session** to refer to the interaction between the server and one client. The service managed by a server may be accessed by multiple clients who desire the service, sometimes concurrently. Each client, when serviced by the server, engages in a **separate session** with the server, during which it conducts a dialog with the server until the client has obtained the service it required.

A protocol is needed to specify the rules that must be observed by the client and the server while conducting a service. Such rules include specifications on matters such as

(i) how the service is to be located,

(ii) the sequence of inter-process communication, and,
(iii) the representation and interpretation of data exchanged with each IPC. On the Internet, such protocols are specified in the RFCs.

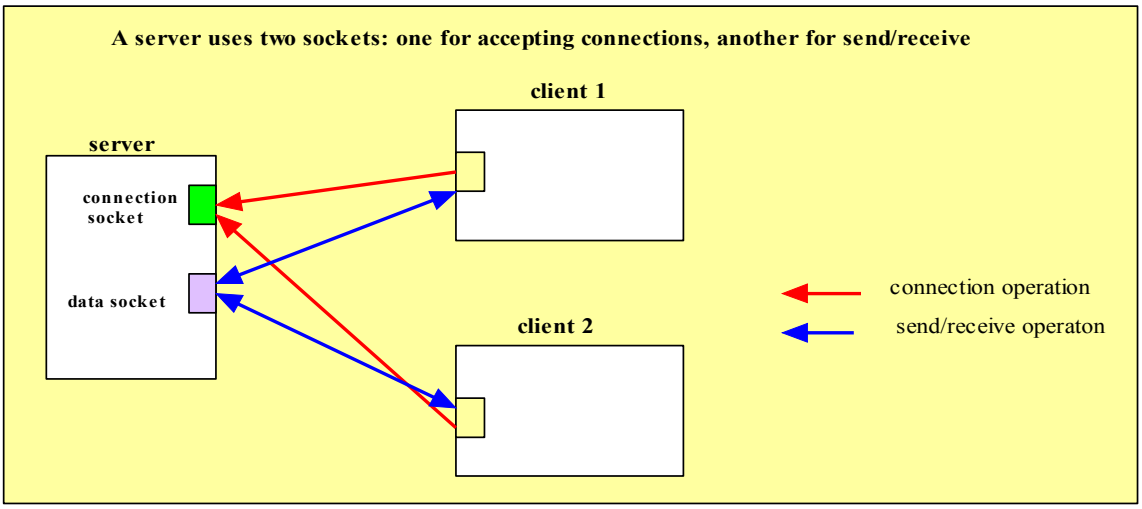
B. The Java stream socket API (Single Thread Example)

Java supports both connection and connectionless socket communications. Connectionless socket communications is done using the *DatagramPacket* over a *DatagramSocket*. Connection oriented communications is done utilizing the *ServerSocket* stream along with *DatagramSocket*. In this lab we will focus on stream oriented connection.

In Java, the stream-mode socket API is provided with two classes:

- **ServerSocket**: for accepting connections and
- **Socket**: for data exchange.

Pictorially this is shown in the figure below.



Key methods in the ServerSocket class

Method/constructor	Description
ServerSocket(int port)	Creates a server socket on a specified port.
Socket accept() throws IOException	Listens for a connection to be made to this socket and accepts it. The method blocks until a connection is made.
public void close() throws IOException	Closes this socket.
void setSoTimeout(int timeout) throws SocketException	Set a timeout period (in milliseconds) so that a call to accept( ) for this socket will block for only this amount of time. If the timeout expires, a java.io.InterruptedIOException is raised

Key methods in the Socket class

Method/constructor	Description
Socket(InetAddress address, int port)	Creates a stream socket and connects it to the specified port number at the specified IP address
void close() throws IOException	Closes this socket.
InputStream getInputStream( ) throws IOException	Returns an input stream so that data may be read from this socket.
OutputStream getOutputStream( )throws IOException	Returns an output stream so that data may be written to this socket.
void **setSoTimeout**(int timeout) throws SocketException	Set a timeout period for blocking so that a read( ) call on the InputStream associated with this Socket will block for only this amount of time. If the timeout expires, a **java.io.InterruptedIOException** is raised

Server Side Code

The following are the basic Java lines of code to support connection oriented communications from the server's perspective

```
ServerSocket connectionSocket = new ServerSocket(portNo);
// wait to accept a connecion request, at which
//  time a data socket is created
Socket dataSocket = connectionSocket.accept();
```

It is then possible to create read or write streams for the data socket in the following manner. Note that the input stream uses a *BufferedReader* and the output a *PrintWriter* for character input and output.

```
InputStream inStream = dataSocket.getInputStream();
input = new BufferedReader(new InputStreamReader(inStream));
OutputStream outStream = dataSocket.getOutputStream();
output = new PrintWriter(new OutputStreamWriter(outStream));
```

Note: Any output to the socket needs to be flushed.

```
output.flush();
```

At the end of communications the socket and connection needs to be closed.

```
dataSocket.close( );
connectionSocket.close( );
```

Client Side Code

Client side coding is much simpler in that is only necessary to request for a connection by simply utilizing the Socket constructor.

```
Socket mySocket = new Socket(acceptorHost, acceptorPort);
```

Similar to the server all input and output is performed using the data stream objects.

Tasks

Task #1: Echo TCP Server

Use the provided TCPServer.java and TCPClient.java program, from Lab#0. Modify them so that the client repeated takes a line of user's input and writes it to the stream and the server reads repeatedly from the stream, printing out the result of each read.

Presentation to the TA:

Show the TA the running program.

Task#2: Contrasting the reliability of UDP versus TCP

Use the provided UDPServer.java and UDPClient.java program. Modify them to make a test kit to determine the conditions in which datagrams are sometimes dropped. The client program should be able to vary the number messages and their size, from the command-line arguments. The server should detect when a message from a particular client is missed, by making use of sequence numbers for the datagram packets.

Modify the TCPServer.java and TCPClient.java in the same way. Then use the same conditions under which the UDP datagrams are dropped, to run the TCP session. Compare the results and note any differences.

Presentation to the TA:

Show the TA the running program under conditions where the UDP datagrams are sometimes dropped. Show the running TCP program under the same conditions.

Lab report:

In your lab report, present the results from comparing the reliability of UDP and that of TCP. Compare the results, comment on and explain the difference.

Task#3: Set a timer

Modify the programs you wrote for Task#1 to have the server send back a reply message to the client every time the client sends a message to the server. The client sets a timeout on its socket so that it can inform the user when the server does not reply. Test your program by making the server not reply occasionally.

Presentation to the TA:

Show the TA the running program where the server sometimes does not reply and the client informs the user of this event.

Submitting lab report :

Submit your (very short) lab report on Blackboard.