LABORATORY MANUAL

SC2008/CZ3006/CE3005 Computer Network

No. 2: Programming Network Applications using Sockets

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

PROGRAMMING NETWORK APPLICATIONS USING SOCKETS

1. OBJECTIVE

To experiment with writing a real practical network program at application layer which accesses the services provided by the transport layer via sockets.

2. LABORATORY

For the lab location, please check
https://wish.wis.ntu.edu.sg/webexe/owa/aus_schedule.main

3. EQUIPMENT

PC with Windows OS & Java 2 Standard Edition.

4. DURATION

2 hours

Lab submissions are done via the NTULearn lab sites.
(Use the NTULearn lab site, not the main course site.)

5. ASSESSMENT

No assessment.

6. EXERCISE 2A: SOCKET PROGRAMMING

As discussed in Laboratory 1, the complex task of enabling different computers to communicate is fortunately being decomposed into more manageable 5-layer architecture of TCP/IP. So when we want to develop applications that communicate via a network, we only have to concentrate on the application layer, and making use of necessary services provided by the lower layers.

Or, do you still prefer to program it as a single large module?

The entry/end point for the application layer to the transport layer and vice versa is the pair of IP address/port number called a socket, which is specified in RFC 793. Later, this term also became used as the name of the APIs provided by the Berkeley-derived programming interfaces. Eventually programming of application that makes use of these APIs is known as socket programming.

An application layer protocol typically has two modules, a client module and a server module. The client module initiates a communication, while the server module waits for it. Programming an application based on this client/server relationship is also known as client/server programming.

In this lab, we will experiment with client/server socket programming, which is widely practised in the industries. Java programming language will be used to introduce the typical APIs available at the transport layer. This is not a course on programming. It is assumed that you have learned programming and experience with Java from other prior courses.

The *java.net* package provides the socket mechanism for applications to communicate via a network. In the TCP/IP protocol suite, there are two services provided at the transport layer, viz. the connection-oriented TCP service, and the connectionless UDP service. So, we would expect to find two types of sockets corresponding to these two services.

Visit *http://docs.oracle.com/javase/6/docs/api/* and browse through the classes and APIs in *java.net* package. You will be practising Java socket programming using these APIs in subsequent exercises.

7. EXERCISE 2B: PROTOCOLS

To ensure inter-operability between network programs written by different developers, standard protocols are fundamental to all communications. A protocol can be considered as an orderly sequence of messages and/or format used for communication between two computers.

With reference to the TCP/IP protocol suite, every layer at the source computer will indirectly communicate with its peer layer at the destination computer, so protocols are required at each layer.

To appreciate the concept of protocol, let us take a look at a simple protocol of TCP/IP suite defined at the application layer - RFC 865: Quote of the Day (QOTD) Protocol.

Obtain a copy of RFC 865 from *http://www.ietf.org/rfc.html* and read through it. You will need to implement this protocol in subsequent exercises.

8. EXERCISE 2C: RFC 865 UDP SERVER

The codes for the server-side program usually have the following pattern: (refer comments in codes)

```
public class Rfc865UdpServer {

   public static void main(String[] argv) {

      //
      // 1. Open UDP socket at well-known port
      //
      static DatagramSocket socket;
      try {
         socket = new DatagramSocket(...;
      } catch (SocketException e) {}

      while (true) {
         try {

            //
            // 2. Listen for UDP request from client
            //
            DatagramPacket request = new ...;
            socket.receive(request);
            ...

            //
            // 3. Send UDP reply to client
            //
            DatagramPacket reply = new ...;
            socket.send(reply);

         } catch (IOException e) {}
      }
   }
}
```

Implement a UDP server according to RFC 865 - Quote of the Day Protocol.

9. EXERCISE 2D: RFC 865 UDP CLIENT

On the client-side, the order is simply reversed: (refer comments in codes)

```
public class Rfc865UdpClient {

   public static void main(String[] argv) {

      //
      // 1. Open UDP socket
      //
      static DatagramSocket socket;
      try {
         socket = new ...;
      } catch (SocketException e) {}

      try {

         //
         // 2. Send UDP request to server
         //
         DatagramPacket request = new ...;
         socket.send(request);
         ...

         //
         // 3. Receive UDP reply from server
         //
         DatagramPacket reply = new ...;
         socket.receive(reply);
         ...

      } catch (IOException e) {}
   }
}
```

Implement a UDP client according to RFC 865.

10. EXERCISE 2E: INTER-OPERABILITY TESTINGS

First, test your client against your own server to make sure that they can communicate. What are the possible host names/IP addresses that you can use for your client to initiate a communication with your server on the same machine?

After successful testings, you may wish to test your client against other servers implemented by your friends in your class.

Once you are satisfied that your client is working, do a final **one-time** testing of your client against a given RFC 865 UDP server to be made known to you during your laboratory session. Send the following data to the server:

```
"YourName, YourLabGroup, YourClientIPAddress"
```

Upon receiving your request, the given RFC 865 UDP server will log it down, and a random quote of the day will be replied to your client. Hence you should receive the quote of the day after transmitting your data.

Finally, submit a print copy of your client program ONLY (into your network drive). Include the followings in the comments of your client source codes:

Name: YourName

Group: YourLabGroup

IP Address: YourClientIPAddress

11. EXERCISE 2F: RFC 865 TCP CLIENT (OPTIONAL)

If you have time, you may wish to try programming RFC 865 TCP client as follows:

```
public class Rfc865TcpClient {

   public static void main(String[] argv) {

      //
      // 1. Establish TCP connection with server
      //
      static Socket socket;
      try {
         socket = new Socket(...;
      } catch (Exception e) {}

      try {

         //
         // 2. Send TCP request to server
         //
         OutputStream os = socket.getOutputStream();
         os.write(...;
         ...

         //
         // 3. Receive TCP reply from server
         //
         InputStream is = socket.getInputStream();
         ...

      } catch (IOException e) {}
   }
}
```

12. EXERCISE 2G: RFC 865 TCP SERVER (OPTIONAL)

Similarly, you may wish to try programming RFC 865 TCP server as follows:

```
public class Rfc865TcpServer {

   public static void main(String[] argv) {

      //
      // 1. Open TCP socket at well-known port
      //
      static ServerSocket parentSocket;
      try {
         parentSocket = new ServerSocket(...;
      } catch (Exception e) {}

      while (true) {
         try {

            //
            // 2. Listen to establish TCP connection with clnt
            //
            Socket childSocket = parentSocket.accept();

            //
            // 3. Create new thread to handle client connection
            //
            ClientHandler client =
               new ClientHandler(childSocket);
            Thread thread = new Thread(client);
            thread.start();

         } catch (IOException e) {}
      }
   }
}

class ClientHandler implements Runnable {

   private Socket socket;

   ClientHandler(Socket socket) {
      this.socket = socket;
   }

   public void run() {

      //
      // 4. Receive TCP request from client
      //
      ...

      //
      // 5. Send TCP reply to client
      //
      ...
   }
}
```

With more practices, soon you'll be able to write any network applications to communicate through the Internet! :)