CS032 2008 Lab 04
Sockets

1 Introduction

In this lab you are going to learn about socket programming in Java and write your own server/client application, using your knowledge from the Threads lab. You are going to use the server/client model in your next assignment, NewsWhere.

The lab includes a short introduction to sockets and the java.net package, which is going to help you write your own chat room application. You are going to implement a client that connects to a server, which maintains a list of all connected clients in the chat room. Your client is going to be able to communicate to all other connected clients by reading from and writing to a socket.

2 What is a Socket?

By definition a socket is one end-point of a two-way communication link between two programs running on the network. In other words, it is through sockets that clients access the network and transmit data.

On the server-side: Normally, the server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket, for a client to make a connection request on that port.

On the client-side: The client knows the hostname (this can be an IP address or a network ID such as cslab1a) of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client attempts to contact the server on the given hostname and port. The client also needs to give the server a way to contact it and so it binds to a local port number that it will use during this connection. This is usually assigned by the system.

If everything goes well, the server accepts the connection. Upon acceptance, the server binds a new socket with its remote endpoint set to the address and port of the client and the same local endpoint. It also creates a new socket so that it can continue to listen to the original port for connection requests while tending to the needs of the newly-connected client.

Unfortunately, in the real world, sockets rarely go according to plan. The client, once it has initiated its connection, blocks all threads until a response has been received from the server. However, sometimes the server is busy or the request is lost over the network, and your program will be waiting forever. Just in case, it is good practice to use a timeout, which is the amount of time a client waits before giving up or trying to connect again. For more information on the uglier side of clients, take CS168 or wait until later in the semester when we cover sockets in C++.

3 Sockets in Java

In Java, socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes – `java.net.Socket` and `java.net.ServerSocket` – that implement the client side and the server side of the connection, respectively.

The `java.net.Socket` class is used by the client to establish a connection to the server. Connecting to the server can be accomplished by creating a `Socket` object. The socket at client side just needs to know the host name (the name of the machine where server is running) and the port where the server is listening. Here is one way to create a socket:

`Socket mySocket = new Socket("localhost", PortNumber);`[1]

[1] `localhost` is the hostname of the local machine.

Establishing a server that monitors a particular port is done by creating an instance of the `java.net.ServerSocket` class. Here is one way to do that:

```
ServerSocket myServerSocket = new ServerSocket(PortNumber);
```

The next step is to tell the newly created server to listen indefinitely and accept incoming requests. This is done by using the `accept()` method of the ServerSocket class. When a request comes, `accept()` returns a Socket object representing the connection.

4 Reading from and Writing to a Socket

Communication between client and server is achieved by reading from and writing to a socket.

On the client side, communicating with the server is accomplished in two steps. First, the Input and Output stream corresponding to the Socket object are obtained. That can be done by using the `getInputStream()` and the `getOutputStream()` methods of the Socket class. In code it would be:

```
BufferedReader in = new BufferedReader(new InputStreamReader(mySocket.getInputStream()));
PrintWriter out = new PrintWriter(mySocket.getOutputStream(), true);
```

Once these are set up, you can read and write using the corresponding streams. For example:

```
String line = in.readLine();
out.println("Echo: " + line);
```

On the server side, communicating with the client is no different from how the client communicates with the server. Since the communication has to continue until the client breaks the connection, the reading and writing is done within a loop. Once the client breaks the connection or stops sending the request, the Socket object representing the client has to be closed (similar to closing a file descriptor after you've opened it). This can be done by calling the `close()` method on the Socket object. Don't forget to close the input and output stream before you close the socket.

5 Write Your Own Server/Client Application

We provide you with the stencil code of a simple chatting application based on the server/client model. You can copy the stencil code from /course/cs032/pub/lab04. Your task is to read carefully the comments before each method and implement all of the methods. You are going to do this in two steps. First, you are going to implement the client, and once you get it working, you are going to fill in the server. Your knowledge about threads is going to be useful in implementing the server since, in this model, a different thread is used to handle each client connection.

6 Part I: Implementing the Client Side

For this part we provide you with a running server (your TAs will tell you the hostname and port on which it is running) and a configuration file, `ChatConfig.java`, that contains the two parameters which the client needs to connect to the server: the hostname of the server machine and the port on which the server is listening. The classes you have to complete are `chatClient.java` and `Client.java`.

The `chatClient.java` class contains the main method, which creates a new Socket associated with a new instance of the `Client.java` class.

The `Client.java` class contains a thread for writing to and a thread for reading from the socket. Remember that in the constructor, you not only have to create the threads, but start them as well.

Once the client is connected to the server it has to be able to send messages to all other connected clients and receive messages from them. Sending a message is acomplished by first sending the message to the server, which maintains a list of all connected clients. The server then sends the message to the rest of the clients.

When you complete the classes you can test your client by running the `runChat` script, passing it a user name. The script opens a new client connection to the server, effectively signing you into the chat room. A new terminal will pop up, which displays the messages received from the server. Now you should be able to chat with the rest of your cs032 fellows. Enjoy!

You have now reached CHECKPOINT 1.

7 Part II: Implementing the Server Side

Once you make sure that you can successfully sign on to the chat and send and receive messages, you are going to complete the server-side classes: `Server.java` and `ClientHandler.java`. The server represents the chat room: it keeps track of all signed-on users, receives messages from them and sends the messages to all other users in the chat room.

The `Server.java` class has a ServerSocket which listens for connection requests from clients and accepts the connections. Once the connection is established, a new socket is associated with the client and a new ClientHandler is created. The ClientHandler deals with receiving messages from the user and sending them to all users (including the sender).

When you finish implementing the server don't forget to change server hostname in your configuration file to "localhost" before running the script.

Congratulations! You have your own chat room! CHECKPOINT 2.

8 Checkpoints

1. 50 points TA Initials: __________
   - `part I` runs successfully.

2. 50 points TA Initials: __________
   - `part II` runs successfully.