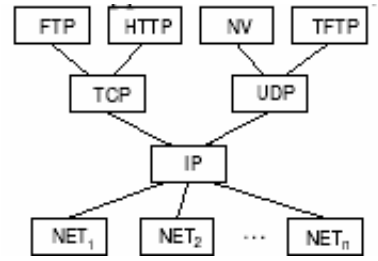
LAB 12: JAVA Sockets Programming

Goals

Become familiar with basics of networking: internet architecture, IP addresses, TCP connections, ports. Learn the basics of Java sockets programming: creation and usage of sockets; communication between server/client applications running on different machines on the network.

Basics of Networking

Internet Architecture:



Suppose you want to transfer a file from an FTP server on one machine to your machine using an FTP client. Let us follow the steps required to initiate such transfer.
When you start your FTP client and try to connect to an FTP server on some machine (say, denali.ccs.neu.edu) your FTP client first finds out IP address of the server (using DNS for example; denali's IP is 129.10.116.200). Then it sends a request to open a connection with denali (this request is sent to port 21 – the default port for FTP protocol). If denali is running an FTP server it will respond to the requests that come on port 21. Denali will figure out where the request came from and will reply with an ACK. We will not go into the specifics of establishing a TCP session; we will only note that server and client eventually find out each other's IP addresses and ports to use for this connection. Once the connection is established data can be transferred.

Basics of Java Sockets Programming

What is a socket?
A socket is an end-point of a two-way connection between two applications that communicate over the network. In Java a socket can be represented with an instance of Socket class. The java.net package provides two classes that we are going to use for this lab: Socket (client side of the connection) and ServerSocket (server side).

So what is socket programming?
Socket programming allows one to write applications that communicate over the network. Each computer has a number of ports, and an application can reside on any of these ports. We say that an application is listening on a particular port if it constantly checks for messages coming from network and destined to this particular port number. Usually server side application will be listening on some well-known port number and reply to client applications attempts to establish communication (web servers – port 80, ssh – port 22, FTP servers – port 21, etc).

Relevant Java API reference:
http://java.sun.com/j2se/1.4.2/docs/api/java/net/ServerSocket.html
http://java.sun.com/j2se/1.4.2/docs/api/java/net/Socket.html
http://java.sun.com/j2se/1.4.2/docs/api/java/net/InetAddress.html

http://java.sun.com/j2se/1.4.2/docs/api/java/net/DataInputStream.html
http://java.sun.com/j2se/1.4.2/docs/api/java/net/PrintStream.html

Client and Server Examples

1. Take a look at the code for MyServer and MyClient classes (pay attention to comments in the code for explanation). Describe the steps the server will take if it gets a packet from the network destined to port 6070. For now the machine that the clients try to connect to is "localhost". What happens if you set the variable *remote* to be something else (say, denali.ccs.neu.edu)?

2. Let's take a look at what the steps for client side program would be:

- Open a socket
- Open an input stream and output stream to the socket
- Read from and write to the stream according to the server's protocol
- Close the streams
- Close the socket

What are the steps that MyClient application takes to connect to the server? Look at the while loop and try to understand what exactly happens in the body of this loop and what has to happen for the program to exit this while loop.

3. Can you figure out how to get the IP address or host name of the remote side of the socket? (Hint: look for System.out lines in the code…). Modify the code for the client program: make it try to connect to an alternative server if the connection for the first one fails. Note that you need to only retry to connect once. If this fails just exit the program. The information about what happens when a connection attempt is not successful can be found in the API (hint: Socket class, connect() calls).

Network Collaborative Guessing Game

Now let's see who is fast and skillful in doing network programming in Java!
You have an opportunity to take part in a collaborative guessing game over the network. The game proceeds as follows:

Each player can log in and get assigned a "lucky number". The goal of the players is to guess the "lucky numbers" of as many players as possible. These "lucky numbers" are string representations of doubles chosen at random by the server. This is why is not likely one will make a correct random guess. However players can query the server for the "lucky numbers" of other users and submit obtained information.

So what will you need to do to play this game? You will have to modify the example code for MyClient.java. Make the program connect to the server and either query the server for "lucky numbers" manually (console input/output) or write a program that will fetch information automatically for you.

Here are the steps you will have to go through to play:

0. First you need to find out server's name and port number. (Ask your TA about server name/IP address; the port will most probably be 6070)
1. To log in send "login:*first_last*" to the server, where *first_last* is your first and last name
2. The server will send you back "replylogin:*id*" and *id* will be a string representing an integer indicating your unique *id* in server's database ("replylogin:failed" will be sent if error occurs).
3. Now you can send a string "list" to the server to find out who else is logged in. The server will send you back the name of one of the logged in users at random, with some probability the server will send you a string "exit". If the server sends "exit" your client will have to exit – the game is over for you and you will have to start from scratch (Note: this is not enforced, but please be nice and follow the rules!).
4. If you receive "reply:*user_name*", then you can send server "request:*user_name*" and get the lucky number of user *user_name.*
5. The server will reply to your request with "replyrequest:*user_name*:*number*".
6. Now you can submit "guess:*id*:*user_name*:*number*" and get your points for the correct guess (*id* is that unique number that the server sent you when you logged in). Note: all the messages can be converted to strings to be sent over the network.

Good luck!

Appendix – Game Protocol

Message format - Client:
login:*first_last* – login request
list – get name of one of the players
request:*first_last* – get lucky number of player *first_last*
guess:*id*:*first_last*:*number* – submit your guess of user *first_last*'s lucky number

Message Format - Server:
reply:error – sent upon reception of incorrectly formatted message
replylogin:*id*:*number* | replylogin:failed – reply to login
reply:*user_name* | exit – response to "list" message
replyrequest:*first_last*:*number* | replyrequest:empty – response to request message
replyguess:correct | replyguess:incorrect – a reply to submission of your guess

Score points: how many users' numbers you guessed + how many times your number was guessed by others.

Disclosure: "points" mentioned above unfortunately have nothing to do with extra credit points. But they are still a great way to impress you peers and professors ☺

Sources

http://java.sun.com/docs/books/tutorial/networking/sockets/
http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets_p.html
http://java.sun.com/j2se/1.4.2/docs/api/java/net/ServerSocket.html
http://java.sun.com/j2se/1.4.2/docs/api/java/net/Socket.html