Lab

2

Programming with Sockets in Java

Prepared for UCC Networking Course, June - August 2012.

Purpose of this lab:

This labs provides practical experience with writing a client-server application for the Internet.

Software Tools:

- The programming for this lab is done in Java and requires the use of *Java (stream) sockets*.

What to turn in:

- Turn in a report with your answers to the questions in this lab, including the plots, hard copies of all your Java code, and the anonymous feedback form.

What to turn in:

- A report with your answers to the questions in this lab, including the plots, copies of your MATLAB code, and the anonymous feedback form.
- The symbol indicates questions for the lab report.
- The lab exercises and the lab reported are to be completed individually.
- The estimated time to complete the lab is 3 hours.

Preparing for Lab 2

Tutorials on the Java programming language can be found on the Internet.

- **General Introduction to Java:**
  - Java Tutorial – A practical guide for programmers: http://java.sun.com/docs/books/tutorial/
  - R. Sedgewick, K. Wayne: Introduciton to Programming in Java, http://introcs.cs.princeton.edu/java/home/

- **Socket programming:**

There are two principal types of sockets: Datagram Sockets and (Stream) Sockets. Java datagram sockets use the UDP transport protocol to transmit traffic. Stream sockets use the TCP transport protocol to transmit traffic. In Java Stream sockets are implemented by the "Sockets" class.

In the Java Workshop, we used Datagram sockets. This lab uses (Stream) sockets. Information on network Programming in Java with examples can be found at:

  - "All about sockets": http://docs.oracle.com/javase/tutorial/networking/sockets/
  - Java Socket Tutorial: http://www.javaworld.com/jw-12-1996/jw-12-sockets.html

Examples of a client and server using (Stream) sockets can be found at:
Client: http://www.cs.uic.edu/~troy/spring05/cs450/sockets/EchoClient.java
Server: http://www.cs.uic.edu/~troy/spring05/cs450/sockets/EchoServer.java

Comments

The grade for this project is weighted as follows:

- 50% Implementation
- 20% Functional Tests
- 30% Write-up

Implement an Addressbook service

- The *Addressbook* service is a client-server application. A client sends a query message to the server containing an email address. The server responds to the client with a message that contains the full name which corresponds to the email address.

- A client process *Addressbook_client* submits query messages to the server. The query message contains an email address.

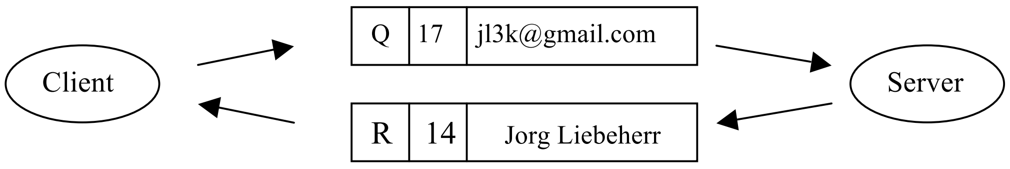
- The server process *Addressbook_Server* runs on a workstation with a known domain name (e.g., `www.cnn.com') or a known IP address (e.g., `64.3.4.2') and listens on a well-known port number for client requests. When a request arrives, the server performs a lookup database to find the full name which corresponds to the email address, and sends the full name to the requesting client.

- The following messages format is used for query and response:

Message Type (1 byte)	AString Length (1 byte)	AString (≤ 255 bytes)

  - *Message Type* is set to “Q” (ASCII 81) for queries, and “R” (ASCII 82) for responses.
  - *AString Length* is an unsigned 8-bit integer.
  - *AString* is a character string with up to 255 characters.

Example (simplified):



Requirements:

- *Addressbook_Server* and *Addressbook_Client* must be able to run on different machines on the Internet.
- The Address_Client must be able to access a test server (The address and port number of the test server will be announced in class).
- The database of the server is initialized from a text file which has entries of the form:
  “Name” “email address”
  An example text file is given in names.txt (see Dropbox).
- The database of the server must be able to handle an arbitrary number of entries.
- The server must be able to process multiple queries in a row.
- You get 10% extra credit if your server can handle multiple clients simultaneously.

Your Task:

1. Implement the *Addressbook* service systems. Both server and clients are to be implemented exclusively with sockets (stream sockets). The implementation must be done in Java.

2. Test your client implementation with the server specified in the requirements.

3. Deliver a max 1 page write-up, which describes your implementation approach, a description of your test strategy, and known bug list.

Hints:

1. A client process contacts the server by connecting to the well-known port number on the server machine.
2. The server process *accept*s the connection request and then handles the *Addressbook* request.
   (*Note:* Alternatively, the server process may *create* a new thread process that handles the request using a new socket (with a number that is different from the well-known port number.)

Submission Guidelines:

- Your submission consists of:
  - Source files (*.java) and class files (*.class) of the client and server.
  - A README file containing brief instructions for compiling and running your program. (If instructions are missing or the files cannot be compiled/started with the given instructions, the functional tests will fail).
  - Your 1-page write-up.
- Place all materials of your submissions into a single directory. The name of the directory should be Lab2-XXX-YYY, where XXX is your last name and YYY is your given name. Copy the directory to the Dropbox folder for Lab 2.