CSE 473 – Introduction to Computer Networks

Lab 2

John DeHart *Due 9/28/2015*

Review and follow the general instructions included in the Lab 1 writeup.

In this lab, you will be writing a map server and associated client that uses TCP. You will be using the Open Network Lab to test your client and server.

As with the UDP map server from lab1, the server stores a set of (*key*,*value*) pairs, where both the *key* and the *value* are strings, and where no two pairs can have the same *key*. The operation *get(k)* returns the *value* part of the pair whose key is *k*. The operation *put*(*k*,*v*) adds the pair (*k*,*v*) to the set, possibly replacing some other pair (*k*,*x*). The operation *swap*(*k1*,*k2*) swaps the values stored with the two keys. If either k1 or k2 is not found then the operation does nothing. The operation *remove*(*k*) deletes the pair (*k*,*v*) from the server.

More specifically, the server will accept TCP connections from clients, and will process multiple operations from the client, continuing until the client closes the connection. When the client closes the connection, the server waits for a connection from another client. In this lab, the server will only have one open connection at a time.

Each command from the client takes the form of a text string on its own line. There are five commands, "get", "get all", "put", "swap" and "remove". The colon character (':') is used as a delimiter. The payload requests and responses are of the same syntax as the previous lab. For example, a *get* packet's data would look like

```
get:this is the key string
```

and the server would respond with

```
success:this is the value string
```

or

```
no match
```

The only new command is the *get all* command, which takes no arguments. This command requests all of the key-value pairs stored in the map. All pairs are returned on a single line, with each key-value pair separated by a pair of colons. For example,

```
key:lock::major key:bee flat::public key:flabber jabber::heehaw
```

If an input packet is not well-formed, the server should reply with

`error:unrecognizable input:`*a copy of the offending input*

The server continues to process requests from the client, one command per line, until the client closes the connection. The server detects this when a *socket.read*() call returns –1.

The server should take two optional command line arguments. The first is the IP address to which the server's socket is to be bound. If this is omitted, the server should use the wildcard address. The second argument is a port number. If no port number is specified, the server should use port 31357. If the port number is specified, the IP address must also be specified. The IP address argument is useful when running on a server with multiple interfaces, since it allows you to force server connections to use a specified interface. We'll use this feature in ONL to ensure that our test traffic is not sent through ONL's control network.

The client program should take 1-2 arguments. The first argument is the name or IP address of the host that the server is running on and the second (if present) is the port number on which it's listening. The second argument defaults to 31357. The client should open a TCP connection to the server and then read lines of input from *System.in*. Each line of input is assumed to contain an appropriately formatted command. The client need not check the input; checking should be done at the server. Each line is sent to the server (with a terminating newline indication) and then the client should wait for a response, which is printed to *System.out*. When the user inputs a blank line, the client should close the connection and terminate. Write both programs in Java and name the files `TcpMapServer.java` and `TcpMapClient.java`, replacing the blank files that appear in the repository.

Here are a few tips that will be helpful when writing your programs.

To read from *System.in*, first declare a *BufferedReader* using

```
BufferedReader sysin = new BufferedReader(
                              new InputStreamReader(System.in));
```

Now, you can read lines of input using

```
String line = sysin.readLine();
```

When writing your socket code, use a *BufferedReader* and a *BufferedWriter*. When sending a line of text over the socket, use the *newLine()* method of the *BufferedWriter* to send a new line indication and the *readLine()* method to read complete lines from the socket.

To create a *ServerSocket* that is bound to a specific IP address and port, construct the socket using

```
new ServerSocket(portNumber, 0, ipAddress)
```

The first argument is the local port number that the socket should be bound to, and the last argument is the *InetAddress* of the interface you want to use. We'll ignore the middle argument (by specifying 0, we're effectively telling the system to use its default value for this argument). Note that this creates a listening socket. Any connection sockets created using this listening socket will be bound to the same port and address. To create a *ServerSocket* that uses the wildcard interface and a specific port, set the *ipAddress* argument to *null*.

The provided lab report template contains detailed instructions describing what you need to include in your report.

Logistics. Each student has been provided with a *subversion* repository with separate sub-directories for each lab and studio. For each lab, you will be provided with a Word document that you are to use as a template for your lab report. Read the lab report template carefully before you start working on your lab. It contains specific instructions regarding the things you need to turn in. You can save yourself time and aggravation if you read this before you start your work. Fill in *ALL* the required elements of the lab report where indicated and place the complete report in your svn repository (do not change the name). To complete this lab, you will need to create and turn in two source files: *TcpMapServer.java* and *TcpMapClient.java.* When you are done, do an *svn commit to* upload your files to the repository. You do not need to turn in Wireshark PCAP files this time. You should also turn in a copy of your lab report via svn. In addition, you are required to turn in a ***paper copy of your lab report***, fastened with a single staple or binder clip in the upper left corner.