CSE 473 – Introduction to Computer Networks

Lab 1

John DeHart *Due 9/9/2015*

General instructions for labs. Labs will include a moderate amount of programming. Although the programs are typically not very long, they do involve multiple parts that must communicate over the network. Getting the interaction right is usually the hardest part. You are expected to write your programs with care and professionalism. The requirements are the following:

- All code should be formatted to be easy to read and should include appropriate comments.
- The main file of the program should start with a comment block that includes your name, the date and an explanation of what the program does.
  - o Write this so that it can be understood by a reader ***WHO HAS NEVER SEEN THIS LAB ASSIGNMENT*** – that is, your documentation should be self-contained enough that any knowledgeable Java programmer could understand.
- Use variable names that indicate the intended use and are not too long.
- Lines should be limited to fewer than 80 characters in length.

TAs can deduct up to 20% for badly formatted, hard-to-read, or inadequately commented code. This may seem harsh, but it's very easy to avoid such penalties by just exercising a modicum of care and taking pride in your work.

Specific instructions for this lab. You are required to complete this assignment on your own. While you can discuss the lab in a general way with other students, you must not share code or your Wireshark output.

In this lab, you are to write and test a client/server application that implements a simple storage service with four operations: *get*(), *put*(), *swap*() and *remove*(). 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 UDP packets with a payload consisting of ASCII characters starting with either *get*, *put, swap* or *remove*. The colon character is used as a delimiter, so a *get* command should be formatted as

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

and a *put* command as

```
put:another key string:and the corresponding value
```

The server should respond to a get command with a UDP packet containing a payload similar to

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

or

```
no match
```

This table summarizes the commands and return payloads:

Command	Found?	Payload
`get`	yes	`success:value`
	no	`no match`
`put`	yes	`updated:key`
	no	`success`
`swap`	yes	`success`
	no	`no match`
`remove`	yes	`success`
	no	`no match`

If the input packet is not a well-formed *get*, *put, swap* or *remove*, the server should reply with

`error:unrecognizable input:`*put a copy of the input packet's payload here*

The server should take an optional command line argument. If present, it is interpreted as the port number that the server is to use. If no port number is specified, the server should use port 31357.

The client program should take from 4-5 arguments. The first two arguments are the name of the host that the server is running on and the port number on which it's listening. The third argument is either *get*, *put, swap* or *remove* and the remaining arguments are the argument strings for its respective operation. The client should send an appropriately formatted *get*, *put, swap* or *remove* packet to the server (based on the command line arguments) and print the payload of the packet returned by the server. Do not do any error-checking in the client, but do check for errors in the server.

Write both programs in Java and name the files `MapServer.java` and `MapClient.java`. Your server can use Java's *HashMap* data structure to implement the core storage service and you may use the *String.split* method to divide the packet into parts delineated by colon characters. To test your code, open two shell windows (or command windows if you are using Windows) and run the server in one window and the client in the other. You may also use the provided *testScript* to run through a whole series of tests (use *testScript.cmd* if you are running on Windows). Note that when running the script, you will need to supply the host name (or IP address) of the host on which the server is running as a command line argument.

Once you are satisfied that your programs are working correctly on a single computer, you can test them using two different computers (call them *A* and *B*), both of which have the *Wireshark* packet capture tool installed. After starting your server on *B*, start a *Wireshark* packet capture on *B* using the filter "udp.port==31357". Also run *Wireshark* on computer *A* (with the same filter) and then run *testScript* on *A*. You should see a number of entries in both *Wireshark* windows. Examine the data, looking in detail at the UDP fields and the payload portion of the UDP packet. Compare what you observe in the *Wireshark* windows to what you expect to see based on the contents of *testScript*.

The computers in the Urbauer labs already have *Wireshark* configured, but *Wireshark* can also be downloaded and installed on your own computer. The companion web site for the textbook includes a series of *Wireshark* labs. The first of these describes how to install *Wireshark* and try it out.

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 four files: *MapServer.java*, *MapClient.java, ServerPackets.pcap,* and *ClientPackets.pcap*. When you are done, do an *svn commit to* upload your files to the repository. 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.