CS 332 Computer Networks **Spring 2018**

Project 1

Assigned: Monday, January 22 *Due: Monday, February 5, 11:59:59pm.*

1 Introduction

In this assignment you will write a client program which will communicate using sockets with a server program provided by me. You will be given a skeleton of the client program that provides command line argument parsing (following the specifications given below), but the communication part of the client is all yours. The tar file that accompanies this project contains a binary of the server, the client skeleton, a file called "NetworkHeader.h" that should be #included in your client, and a "standard output" echo server that you can use for debugging. Instructions for running the server are included below.

2 The Protocol

This exercise has only three types of messages: `HELLO`, `ACK`, and `BYE`. `HELLO` messages are sent from a client to the server and `ACK` messages are sent from the server to a client. Each message is a simple character string which consists of several fields, and which must be terminated by a new-line character, `\n`. All fields of a message must be separated by exactly one blank space ␣, and the characters sent must not include the string ending null character (\0) contained at the end of all C strings. The number of fields varies with the message type.

Each `HELLO` message has four fields:

- a version field (must be set to the string "CS332" for this project),
- a type field, in this case, the string "HELLO".
- a firstname field (this is your first name), and
- a lastname field (this is your last name).

A sample `HELLO` message, in C string format, is as follows:

`CS332␣HELLO␣JENNIFER␣SMITH\n`

Upon connecting to the server, a client sends a `HELLO` message and waits for a reply. When the server receives a `HELLO` message, it constructs an `ACK` message and sends it back to the client. The `ACK` message has three fields, all separated by a single space (and again no \0 characters), with the final field followed by a newline character:

- a version field (once again "CS332"),
- a type field, in this case, "ACK", and
- a cookie — this is a string that must be set by the person who starts the server (more on this below).

Once your client receives the `ACK` message, it must print (to standard output) the contents of the received ACK message. It is then ready to end its conversation with the server. This is done by sending the server a `BYE` message. The `BYE` message contains (in order) the version string, a type field (set to "BYE"), and the cookie sent by the server, all separated by a single space.
Note well: Do not include blank spaces within fields (even if your name contains spaces!).

3 The client program

The command line syntax for the client is given below. The client program takes several command-line arguments, corresponding to the firstname and lastname fields of the HELLO message (as described above). The hostname and port specifications are optional arguments. The person running the server includes the hostname (and possibly port) arguments, which must override any default definitions of the server host and port inluded in the NetworkHeader.h file (and actually since I've provided the argument parsing for your client, I've coded it to do just that).

```
Project1Client [-s <hostname>[:<port>]] -f <firstname> -l <lastname>
```

4 The Server

The command line syntax for the server is

```
Project1Server -s <cookie> -p <port>
```

The server responds to *correctly formatted* client messages by printing the contents of the received message. When the server receives the `BYE` message, it adds that it has "received final client message", prints out this final message, **and continues to run, waiting for additional clients that may wish to contact it**. You can run the server on any of the linux machines, but I am limiting the port numbers which each of you can use. (In the past, I just let people choose randomly, but invariably everyone chose mathcs01 and a low port number and there were port collisions.) Specifically, I have placed a port assignment file in the content area on the course Blackboard site. Each of you has been assigned a range of 500 port numbers to use as you like for all projects this semester. I expect you all to keep to your assigned port range.

5 The Echo Server

Most echo servers "echo" whatever they receive back to the machine that sent it. This particular server instead prints whatever it receives to standard output. Thus you can use it for debugging your client. The command line syntax for echo server is

```
StdoutEchoServer <port>
```

6 Some notes

- Although we haven't covered this in class yet, you should use (as the Pocket Socket Guide does) a stream based (i.e. TCP) protocol in your code. This adds a bit of complexity to things—unlike UDP, where messages have clear cut boundaries, there are no record boundaries in TCP. The ramifications for your project are that even though the server may send a complete string, your client may receive the string from the kernel in several different pieces, so you can't just use a single call to "recv()" to get the server message (you need to use an infinite loop and test for the receipt of the newline character).
- You should compile your source good using the gcc compiler (which is already installed on all of the linux machines we'll use). Your code must compile, using `gcc` (or, if appropriate, `g++`) without warnings with the `-Wall` flag set.
- You can test your code on any machines you like, though the final version must work correctly with the server running on mathcs01 (IP address 141.166.207.144) and the client on mathcs02 (which has IP address 141.166.207.143, though you won't need that to run the client). Since these are linux machines, I recommend that you write, compile, and test your code on linux machines. (Or at least be sure it compiles and runs on the specified linux machines. I know, for example, that using the terminal on a Mac doesn't always require the same compile flags, and that some system calls can be different, so if you're writing and compiling your stuff on a Mac, don't wait until the last minute to modify it for linux).

- You'll want to get a jump on this so that you have time to discuss difficulties you're having. You really don't want to wait until the last minute with these projects.

7 Project Submission

Project must be submitted using the guidelines on the class web page. The name of the executable that your makefile builds should be `Project1Client`. The name of the primary source file should be `Project1Client.c` (or `Project1Client.cpp` if you are coding in C++). Also, the name of your tar file should be XXXProject1.tar, where XXX represents, in uppercase, the first three letters of your last name. **IF YOU SUBMIT AN IMPROPERLY NAMED FILE, I WILL TREAT IT AS IF YOU HAVE NOT SUBMITTED ANYTHING!!!!** I have placed a text file, containing submission email addresses, in the content section of the course Blackboard page. You should submit your project by attaching the tar file to an email sent to the appropriate address. This has the effect of dropping the tar file into one of my Box folders.