CPTS 455 Project 1
TCP/IP Socket Programming
Assigned Sept. 16, 2013
Due: Oct 7, 2013, 11:59PM (turnin using Angel Dropbox)

In this project you will develop a client and server that interact over a TCP connection. The specified protocol involves use of several different kinds of serialization of protocol data so that you become familiar with the programming techniques used to deal with each of them.

Specifically, the client and server are **C programs** that need to meet the following requirements:

1. The client program needs to take two command-line arguments that specify the name (or IP address) of the server and the port on which the server is listening. The server program needs to take a command-line argument that specifies the port on which it is to listen.
2. You will start the server first. Once started, it should not quit until exited using a "Ctrl-C" signal (implementing this takes no code on your part). When the client starts it will connect to the server.
3. Once the client has connected, the server will send a welcome message "Welcome to The Server" to the client as a new-line-terminated (\n) string (the 1st communication of our little application protocol). **Note that a null terminating character should not be sent when a newline-terminated string is specified.** Remember that the client code must be prepared for the server to say something other than this; if so it should close the connection, print an error message, and exit.
4. After receiving the welcome message from the server, the client will prompt the user to enter 8-digit ID number and a name (maximum 20 characters) using the keyboard – i.e. from standard input. This ID and the name will then be sent in that order to the server as 2 newline-terminated strings (2nd communication of the application protocol).
5. The server, after receiving the ID number and name from the client, will look up the pair of them in a table (which may be created and stored as part of your program code – don't do anything fancy, just make an array of a few ID numbers and names in your code) and send a validation message "Success" if the pair is found in the table, or send "Failure" otherwise, to the client (3rd communication of the protocol) – whichever one is sent is sent as a newline-terminated string.
6. If failure occurs the server will allow steps 4 and 5 to be repeated, up to a total of 3 tries. **After 3 denials the server closes the socket. The client should take advantage of this server behavior by detecting the closed socket and giving up.**
7. After receiving a Success message from the server, the client will prompt the user with a message asking for a password (maximum 512 characters). This password will be sent to the server by first sending the length as a 2-byte binary number in network byte order (remember to use htons() on the sending side and ntohs() on the receiving side) followed by the string (not null terminated). **Note that this is a different way of sending a string**

than used previously in steps. And again, no null character is sent at the end of the string.

8. The server, after receiving the password from the client, will verify that the password matches the previously received name and ID number in the table in your program code. The server will either send the message “Congratulations <name>; you've just revealed the password for <ID number> to the world!” or the message “Password incorrect” as in step 7 depending on whether the password was correct or not. As before, the server will allow steps 7 and 8 to be tried at most three times and the client should take advantage of this behavior and then close the connection.
9. The client, after receiving the result message, will print out the result and close the socket (if it is success message) and exit. If it is a failure message, the client will allow 2 more chances to send the password to the server by prompting the user to enter password using keyword. After 3 trials, in case of failure message, the client will close the socket and exit.
10. When the server closes its communication socket it resumes listening for client connections.

Additional requirements, hints, etc.

11. Check the return values of all system calls, print an appropriate message and exit if an error occurs. Also, remember that send and recv on TCP sockets do not necessarily send and receive as much data as was requested to be sent/received.
12. Make sure that buffers are big enough for the values that may be stored in them.
13. To go *from* a hostname or an IP address expressed in dotted decimal notation as a string *to* a value suitable for storing in a struct sockaddr_in for use in a call to connect you use the library function getaddrinfo(). Its use is described in Donahoo and Calvert second edition. If you have the first edition of the book it describes the gethostbyname() function which it is also possible to use – or just read the man page for getaddrinfo().