CPTS 455 Project 1
TCP/IP Socket Programming
Assigned Sept. 9, 2011
Due: Sept. 23, 2011, 11:59PM (turnin using the class turnin page)

You may have used software distribution services that require you to obtain a product key before you can install or use a piece of software. In this project you will build a simple (and definitely "not ready for prime time") key distribution server and client. The server will conduct a two-step process to receive identification information from the client and issue a key if the client is authorized.

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 Product KDS" to the client as a null-terminated string (the 1st communication of our little application protocol).
4. After receiving the welcome message from the server, the client will prompt the user to enter 8-digit personal ID number and a product name (maximum 20 characters) using the keyboard. This ID and the product name will then be sent in that order to the server as null-terminated strings (2nd communication of the application protocol).
5. The server, after receiving the ID number and product 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) and send a validation message "Authentication approved" if the pair is found in the table, or send "Authentication Denied" otherwise, to the client (3rd communication of the protocol) – whichever one is sent is sent as a null-terminated string.
6. If authentication is denied 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 an approval message from the server, the client will prompt the user with a message asking for a password (maximum 20 characters). This password will be sent to the server by first sending the length as a 2-byte number in network byte order followed by the string (not null terminated). **Note that this is a different way of sending a string than used previously in steps.**
8. The server, after receiving the password from the client, will verify the received pair of ID and password against a list of legitimate pairs (again, this may be just a table in your program code). If the password is correct, the server will send the message "Your product key for <productname> is <10-digit-number>" to the client as a single, null-terminated string. If the password is incorrect, the server will send "Password incorrect" (null-terminated string) to the client. 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.
9. The client, after receiving the result message, will print out the result and close the socket (if it is success message). 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.

10. When the server closes its communication socket it resumes listening for client connections.

Additional requirements, hints, etc.

11. If you use lab machines make sure that you kill your servers before logging off.
12. 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 all of the data.
13. Make sure that buffers are big enough for the values that may be stored in them.
14. 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().