April 21,2016

ISE 306- COMPUTER NETWORKS
Socket Programming Project

Assoc. Prof. Dr. Berk CANBERK
T.A. Gökhan SEÇİNTİ

Client – Server Communication with a Simple Authentication Protocol

In the given project, you are required to write **C/C++ code** for a client program and accomplish successful communication with the server which runs on the IP **160.75.26.117**. Client program should contain two distinct phases in order to complete communication successfully. First, it should authenticate itself to the server and then server is going to send a series of questions about popular culture and client program provides interface to a user to answer these questions. Details of the implementation are given below.

Part 1. Authentication Mechanism

An e-mail which contains a unique ***hexKey*** has been sent to each student. In the given mechanism, every student should use her/his own unique in order to authenticate herself/himself. Before giving further details about the mechanism, we should first investigate the main purpose of any authentication protocol.

Preliminary Information about Authentication
The main objective of an authentication is to prevent anyone from imitate you by using your unique information. Simply assigning unique IDs and password for users to be used in authentication is not an efficient solution for computer networks. Because, anyone who listens (or shares) the same communication medium could obtain your id and password information to use them later in order imitate you. Thus, in any secure communication protocol, users should avoid transmitting their critical information without an encryption mechanism to keep their identities safe.

With the same motivation described above, you need to avoid sending your unique ***hexKey*** plainly over the network. To encrypt your unique key, ***sha1*** hashing mechanism is going to be used in this project. The source code (*sha1.h, sha1.cpp*) of the necessary functions to use this hashing mechanism could be downloaded from Ninova. Furthermore, you could use the following website to test your implementation http://www.sha1.cz/.

In the defined protocol, first, you need to initialize a TCP connection with the **port** numbered **2016** of the server which runs on **160.75.26.117**. Then, client program should send the following string to initiate the protocol "**Start_Connection**". After receiving the start command, the server is going to send a random hex string ***(randomHex)***. Client program is

going to concatenate its unique ***hexKey*** and the received ***randomHex*** to create a string of hex which contains 64 chars. This string will be used as the input of sha1 hashing function to create a hash of the data. The sha1 function is going to return a byte stream (**unsigned char** in **C**) with size of 20 bytes. Then, you should convert this byte stream into a hex stream by using the functions in sha1.cpp. Conversion to hex is going to expand the size of the stream to 40 chars. You are going to use this char array to authenticate yourself to the server. However, lastly, you should add your student Id to the end of the stream by using '#' symbol between them. Thus, you will send a char array to the server with the size of 50 chars (40 sha1 result + 1 '#' symbol + 9 student ID). If the calculation and the transmission are successful, server is going to send to a message which informs you that you have successfully authenticated yourself and ask you whether you wish to proceed the second part or not. After receiving this message, the authentication part of the project is completed. Figure 1 shows the interactions between the client and the server during the authentication phase.

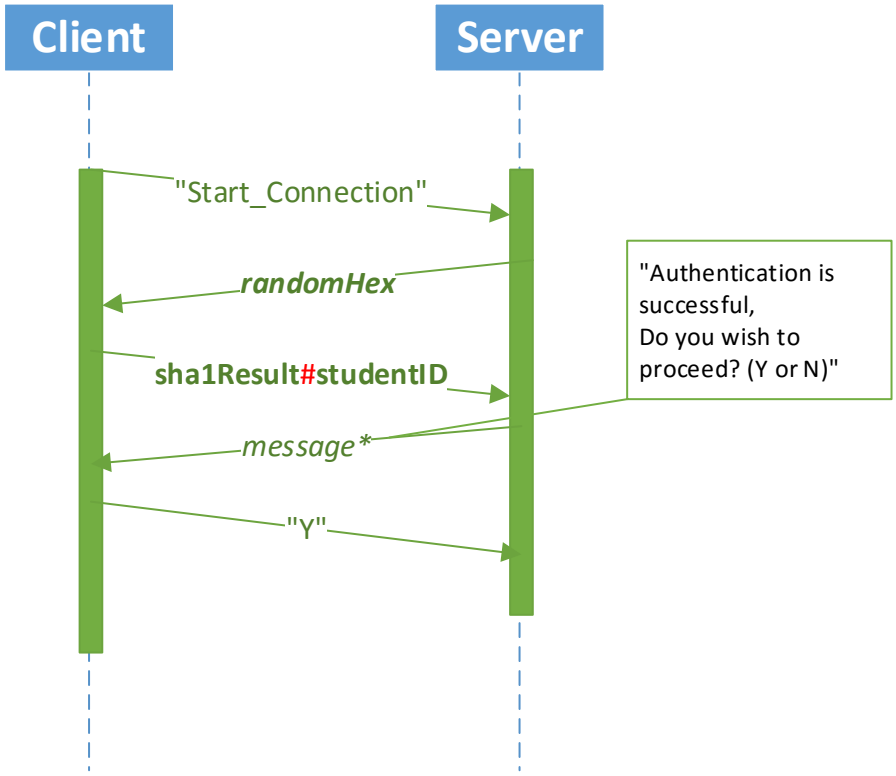

Figure 1: Interactions during the authentication

After, client sends "Y" to the server to notify the server to proceed, server is going to start the 2nd phase and start sending questions to the client.

Part 2. Quiz-up over TCP

In this part of the communication, server is going to send 6 random questions to the client and wait an answer (***a***,***b***,***c*** or ***d***) from the client after each question. Then the server is going to give feedback about the answers. Thus, you should implement a loop to handle incoming questions accordingly. The loop requires to realize the flow given in **Figure 2**.

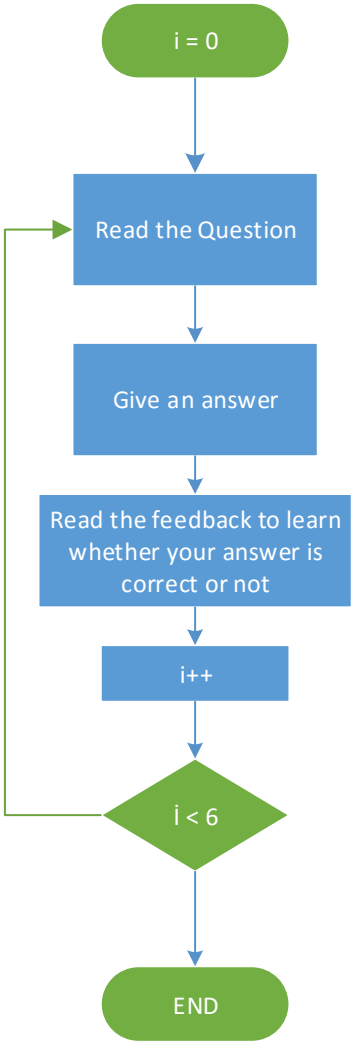

Figure 2: Client-side loop to handle incoming questions

After, completing the quiz, server is going to close the TCP connection and save your score and publish it on the website which can be visited by using any browser by writing the IP address of the server to the address bar.