BLG 433E Computer Communications Assignment

Asst.Prof Gökhan Seçinti
Res. Asst. Sultan Çoğay

In the given assignment, you are required to write Python (v3) code for a client-server program and establish successful communication. Client program should contain two distinct phases in order to execute the communication successfully. First, it should authenticate itself to the server and then server will start the "Guess the Word" game and client program should provide the sufficient interface to a user to play the game. Details of the implementation are given below.

Authentication Mechanism (30 Points)

Server authenticates client by using unique **privateString**(which is shared between client and server beforehand).

Preliminary Information about Authentication
The main objective of an authentication is to prevent anyone from imitating 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 in- formation to use them later in order to 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 given motivation described above, you need to avoid sending your unique **privateString** plainly over the network. To encrypt your unique key, sha1 hashing mechanism is going to be used in this project. **You may use hashlib library for sha1 algorithm.**

In the defined protocol, first, you need to initialize a TCP connection. Then, client program should send the string "Start_Connection" to initiate the protocol. After receiving the start command, the server sends a random string (**randomString**). Client program concatenates its unique **privateString** and the received **randomString** to create a string of hex which contains 64 chars (both **privateString** and **randomString** is 32 characters long). This string will be used as the input of sha1 hashing function to create a hash of the data.

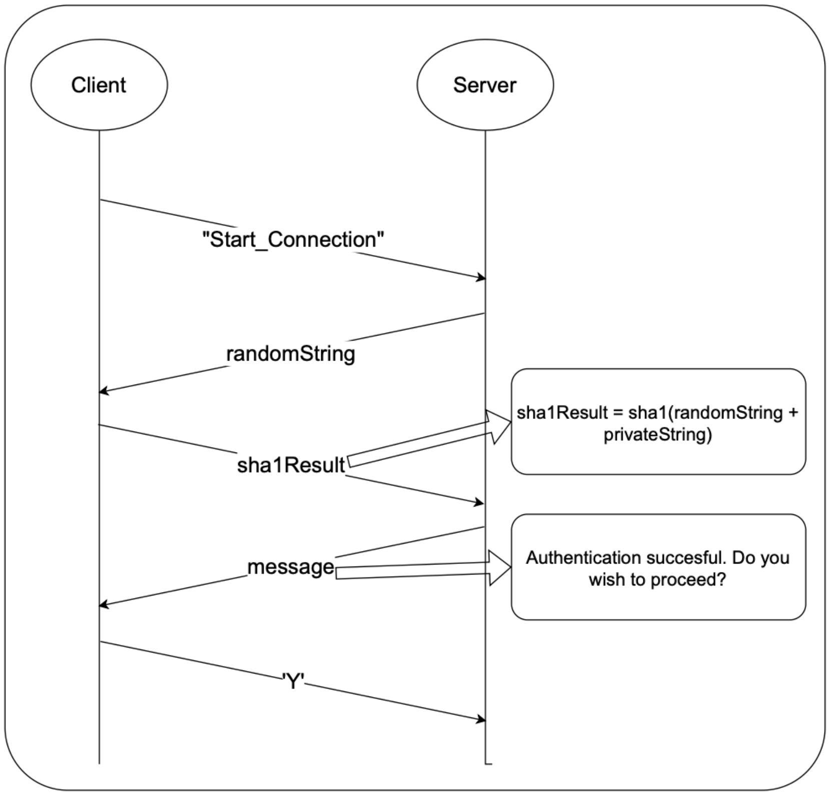
Client
Server
"Start_Connection"
randomString
sha1Result
sha1Result = sha1(randomString + privateString)
message
Authentication succesful. Do you wish to proceed?
'Y'

Figure 1: Authentication Process

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 **hashlib**. Conversion to hex expands the size of the stream to 40 chars. You are going to use this char array to authenticate yourself to the server. Thus, you will send a char array to the server with the size of 40 chars. If the calculation and the transmission are successful, server sends 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.

Number Guessing Game over TCP (70 Points)

In this part of the communication, server starts new game according to client. When a new game starts, client has to guess for the correct number which server will pick in 30 seconds. The server picks numbers between [0-36], and client can either guess the number directly or client can guess whether the number is even or odd. If client guesses number correctly, server awards client with 35 points. If client guesses whether the number is even or odd, it gains 1 point. If client loses, then server sends -1 points to the client.

Server periodically sends remaining time to the client and concludes the game when the time is up. Thus, as oppose to the first part of this assignment, the communication between peers (client and server) is **asynchronous**. The applications should follow a pre-defined format in order to successfully communicate with the server.

> ***About Asynchronous Communication***
> *Some of these messages could be sent asynchronously by the server without any incoming data event. For example, after game is started, server may send remaining time periodically without receiving any "Get Remaining Time" instruction from a client. So, any client application should be able to receive and send data simultaneously.*

Client may send 4 different types of instructions as shown in figures 2 and 3. All of these instructions, except GUESS instruction, contain a single byte. The format of these instructions is given bellow:

```
*0------------------7*
|     Packet_type    |
*--------------------*
# UInt8
00: Start Game
01: Terminate the game
02: Get rem. time
```

Figure 2 : Client Instructions

```
*0----------------7-8--------------15*
|  Packet_type:03  |   Payload Size  |
|        Payload: Guessed Number     |
|      ...                ...        |
*------------------------------------*
# Packet_type Field: UInt8
# Payload field: char array
```

Figure 3: Guess Instruction

The server application has 3 types of messages related with the game. Client should be able to parse these messages and generate outputs for the players, if necessary, in asynchronous manner.

```
*0----------------7-8--------------15*
|    Packet_type   |  Payload Size   |
|              Payload               |
|      ...                ...        |
*------------------------------------*
# Packet Type (Uint8)
# 00: Question (char array)
# 01: Remaining Time (uint16)
# 02: End of the Game (int16)
```

Figure 4: Server Messages

When the game starts, server sends question packet to client which asks" What is your guess? Number, even, odd?". Client displays the payload of the question packet as is. Additionally, server sends remaining time every 3 seconds to the client. Client must display the remaining time always. After client makes a guess or client runs out of time, server sends end of the game packet which has resulting points in its payload.

Client must react to user inputs **immediately** and, send corresponding packets to the server. This means that, client application should not be unresponsive when waiting the server packets. For this reason, client application must use multithreading. You can use **threading** python library.

Submission Rules:

-You must zip your solution (client.py and server.py) and upload to Ninova.

-Any form of plagiarism will not be tolerated. You must do this assignment on your own.

-You can use tcp sockets.

-Client and server applications will communicate using localhost.

- No late applications will be accepted.

- Both client and server codes will be tested against different client and server implementations. So make sure you follow the packet types correctly.