CS 408 - Computer Networks - Fall 2018

Course Project: Simple Quiz Game

This project is made of two steps; each has different deadlines as specified below.

Project Step 1 <u>Deadline</u>: November 21, 2018, 22:00
Project Step 1 <u>Demo</u>: to be announced

Project Step 2 <u>Deadline</u>: December 17, 2018, 22:00
Project Step 2 <u>Demo</u>: to be announced

Introduction

You are going to implement a client-server application which will operate as a simple quiz game. The application will consist of two separate modules. To conduct the game, you need to build a **server** module. Furthermore, for your players to connect and play, you also will implement a **client** module. In this document, the client module will also be referred simply as **player,** either term can be used to express the same concept.

On the **client** side, our **players** will be able to enroll to an open quiz game and will be able to play with **other** players. On the other hand, the **server** will act as a facilitator so that it handles incoming connections from players and helps players communicate so that they can participate in a quiz game and play it properly. Also server should keep scores during the game and broadcast the results of the game when the quiz game is finished.

The server listens on a predefined port and accepts incoming client connections. There may be one or more clients connected to the server at the same time. Each client knows the **IP address** and the listening **port** of the server (to be entered through the Graphical User Interface (GUI)). Clients connect to server on corresponding port and identify themselves with their names. Server needs to keep the names of currently connected clients in order to avoid the same name to be connected more than once at a given time to the server.

The abovementioned introduction is for the entire project, which is to be completed in two steps. Each step is built on the previous step and each has specific deadlines and demos.

Project Step 1 (Deadline: November 21, 2018, 22:00):

In this step, you are going to develop a server module that can accept *two* client connections. The connected clients should have unique names at a given time. Thus, if a client wants to connect with a name that is currently connected to the server, then it should not be connected.

When there are precisely two successfully connected players, the game should be initiated by the server automatically. The first player is the one with alphabetically smallest name. After that one, the second player's turn starts. The game is to be played in several rounds as explained below.

1. The server should send a "ask question" command to the player in turn.
2. The player receives this command and sends his/her question and the corresponding correct answer to the server.
3. The server receives both the question and the correct answer.
4. The server sends the question to the other player.
5. Other player receives the question and sends his/her answer to that question to the server.
6. The server receives the other player's answer and checks if it matches with the correct answer provided by the asking player in step 2.
7. The result of the above check (correct answer or wrong answer) is sent to both players by the server.
8. The roles of asking and answering players exchange and steps 1-7 are repeated.

These steps and the game continue until one of the clients sends a *termination* command to server or one of the clients disconnect.

For the first step of the project, you DO NOT have to keep scores.

For programming rules and submission specifications, please read the corresponding sections at the end of this document.

Server Specifications:

- There is only one server running on this system.
- The port number on which the server listens is not to be hardcoded; it should be taken from the Server GUI.
- The server will start listening on the specified port. It must handle multiple clients simultaneously (in the first step two clients only, but in the second step more than two). To do so, whenever a client is connected to the listening sockets, it should immediately be switched to another socket.

- All activities of the server should be reported using a rich text box on the Server GUI including names of connected clients as well as all the received questions, answers and the status of each turn (the player got the answer right/wrong), etc. We cannot grade your homework if we cannot follow what is going on; so, the details contained in this text box is very important.
- When the server application is closed (even abruptly), nothing should crash!

Client specifications:

- The server IP address and the port number must not be hardcoded and must be entered via the *Client GUI*.
- In the first step of the project, there will be two clients in the system. However, in the second phase, there could be any number of clients.
- If the server or the client application closes, the other party should understand disconnection and act accordingly. Your program must not crash!
- All activities of the client should be reported using a rich text box on the *Client GUI* including connection information, send/received question and answers as well as status of each turn, etc. We cannot grade your homework if we cannot follow what is going on, so the details contained in this text box is very important.
- Each client must have a name. This name must be entered using the *Client GUI*. This name is also sent to the server. The server identifies the clients using their names. Each client must have a unique username.
- Each client can disconnect from the system at any time. Disconnection can be done by pressing a button on *Client GUI* or just closing the client window.
- Both connection and message transfer operations will be performed using TCP sockets.

----------------- End of Step 1 ---------------------

Project Step 2 (Deadline: December 17, 2018, 22:00):

Second step of the project is built on the first step. In this step, you will modify previous client and server modules to add more functionalities.

The aim of this step is to make the game a multiplayer one. Thus, now multiple clients should be able to connect to the server. Moreover, the flow and the rules of the game also changes in this step in order to accommodate multiple players, as detailed below.

1) Server GUI should have two additional fields in this step; a text box field for taking *number of rounds* and a *start game* button.
2) In this step, game DOES NOT start automatically; game should start only when the *start game* button is clicked on Server GUI. There must be at least two clients connected in order to start the game.
3) The players of the game are the ones who successfully connected when *start game* button is clicked. After that point, new players SHOULD NOT be able to connect to the server and join the game. Though, the players in the game can disconnect at their will and this should not cause your program to crash.
4) This time since there can be any number of players, game should be played in Round Robin fashion. This means each player should ask exactly once in each round and answers all of the other players' questions. For stating the obvious, a player MUST NOT receive and answer his/her own question.
5) Server should keep scores of each player. Each correct answer means 1 point and false answer means 0 points. The score table must be sent to all players after each question.
6) When the game is finished, the server should broadcast these results to every connected player and announce the winner. The game will normally end when it is played *number of rounds* rounds. However, at any point during the game, if there is only one connected player remaining, the server should announce that player as winner and finish the game.

As in the step 1, all operations must be clearly shown on the client and server GUIs. For all operations, successful and unsuccessful connection messages, asked questions and answers, score board, each sent and received message during the game should be shown in detail at the GUIs. Error handling (e.g. disconnected players) should be performed properly so that the program does not crash.

----------------- end of Step 2 ---------------------

Group Work

- You can work in groups of **four** people. You may change groups between steps 1 and 2.
- Equal distribution of the work among the group members is essential. All members of the group should submit all the codes for both client and server. All members should be present during the demos.

Programming Rules

- Preferred languages are C# and Java, but C# is recommended.
- Your application should have a graphical user interface (GUI). **It is not a console application!**
- You must use pure TCP sockets as mentioned in the socket lab sessions. You are not allowed to use any other socket classes.
- In your application when a window is closed, **all threads related to this window should be terminated.**
- Client and server programs should be working when they are run on at least two separate computers. So please test your programs accordingly.
- Your code should be clearly commented. This affects up to 10% of your grade.
- Your program should be portable. It should not require any dependencies specific to your computer. We will download, compile and run it. If it does not run, it means that your program is not running. So, do test your program before submission.

Submission

- Submit your work to SUCourse. Each step will be submitted and graded separately.
- Provide a README file explaining your implementation, detailing any issues and/or other implementation details. If your program is lacking in some requested functionalities, clearly explain them. This file should also serve as a user's manual.
- Delete the content of debug folders in your project directory before submission.
- Create a folder named **Server** and put your server related codes and other files here.
- Create a folder named **Client** and put your client related codes and other files here.
- Create a folder named **XXXX_Surname_Name_StepY**, where XXXX is your SUNet ID (e.g. hskocadag_Kocadag_Sinem) and Y is the project step (1 or 2). Put your Server and Client folders into this folder.
    - Compress your XXXX_Surname_Name_StepY folder **using ZIP or RAR**.
- You will be invited for a demonstration of your work. Date and other details about the demo will be announced later.
- 24 hours late submission is possible with 10 points penalty (out of 100).

We encourage the use of SUCourse Discussion module for the general questions related to the project. For personal questions and support, you can send email to course TAs.

Good luck!

Albert Levi
Sinem Kocadağ – hskocadag@sabanciuniv.edu
Beste Seymen – besteseymen@sabanciuniv.edu