My Sister's Good Old Quizz Game

Rebeca Costache

December 2023

Contents

1 Introduction

When I was a little girl, my sister would always get me and my neighbour to compete against each other in general-knowledge contests: she used to reward whoever correctly answered the question with one candy. This is one of the reasons I chose the QuizzGame project proposed by Continental. My overall vision of this project is to achieve full functionality of the specifications of this project in order to make both my sister and myself proud and my objective is to also implement something innovative to this project, so that I will get to sew my creativity into my code, just the way my sister sewed the passion for knowledge in us.

2 Applied Technologies

Communication between server and client(s) will be made through Transmission Control Protocol, because of the following reasons:

- **Reliability**: TCP ensures that data is delivered in the correct order and without errors, thanks to the fact that it provides connection-oriented communication and thanks to the acknowledgement of the successfully sent/received data.
- **Flow Control**: TCP includes flow control mechanisms to prevent fast senders from overwhelming slow receivers. It adjusts the rate of data transmission based on the receiver's ability to process the incoming data, preventing congestion and packet loss.
- **Connection Establishment and Termination**: Thanks to the three-way handshake (to establish a connection between the sender and receiver),TCP ensures that both parties are ready for data exchange. Similarly, it ensures a graceful connection termination through a four-way handshake.
- **Suitable for Long-Running Connections**: Last, but not least, TCP is well-suited for applications that require long-running connections, as it can maintain a stable connection over an extended period of time.

3 Application Structure

The overview structure of the project is presented in Figure 1. The server creates a thread for each client, which will implement the whole game.

Firstly, the client will receive a message with the first steps from the server: login/signup. The client will enter the username and the thread responsible for this client will search in an XML file if the username exists (in case of login) or it will add a new username (in case of signup).

Secondly, the client can choose whether they want to enter an existing game or if they want to create a new game.

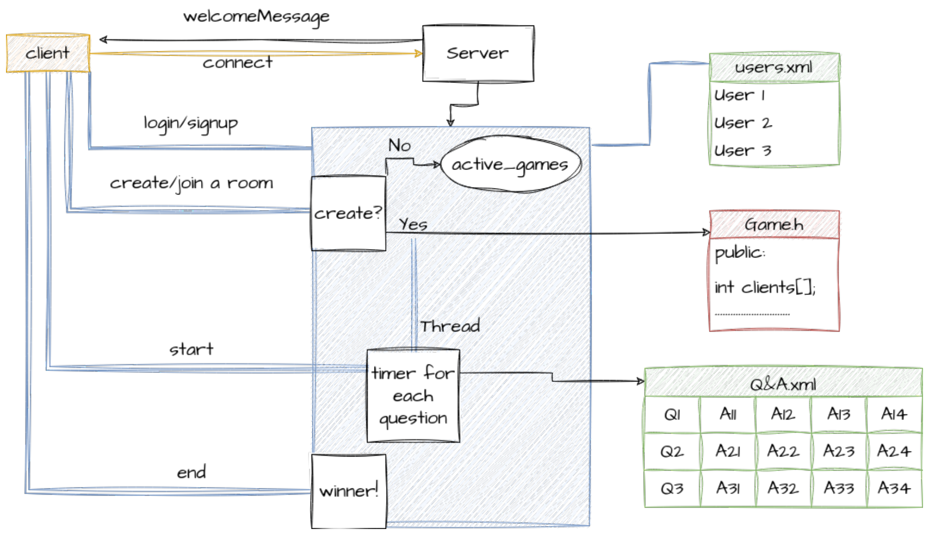

Figure 1: Quizz Diagram

1. **Create a new game**: the thread calls a function which will create a new instance of the class Game (defined in "Game.h", a separate header which is included in the main.cpp). This new instance will have associated with it an array with all the clients participating in the current game and an unique id which will be sent to the client who created the game (the master) in order to be shared with other clients who would like to join this room (P.S.: the Game class has some more data members, other than those described here). The client then waits for all of its friends to join, after which it will enter "start", the command which will actually start the game (only the master can start the game).

2. **Join an already existing game**: The client will only need to enter a valid id and then wait for the master of the game to start the game.

This way, the server is concurrent: each client is served concurrently thanks to the threads. Moreover, the server can have multiple games playing at the same time, each game with an unlimited number of clients.

The game in itself will be handled in the threads which will be synchronized with a function Timer() in order to make sure each client gets the question at the same time and has the same number of seconds to answer. Also, the score of each player is stored in an array which is a data member of the Game class.

4 Implementation Aspects

4.1 Real Usage Scenarios

This is an application that is intended to be used by all people interested in quizz games. It is a great way of spending time with friends, as the application can admit an unlimited number of clients and games. If any player disconnects, the game continues to run with all the remaining clients, making it adaptable to different situations as the one described here. The server will take care of everything: sending the questions, receiving the answers, computing the scores and sending the top-three winners. The client can login/signup, can create a room/enter a room.

4.2 Non-blocking socket

Because I have to implement a server that accepts an unlimited number of clients, the best way to do so is to set the flag non-blocking of the socket; this also allows performing socket operations without blocking the execution of the server, making it easier to detect disconnections.

```
        int flags = fcntl(sockfd, F_GETFL, 0);
        fcntl(sockfd, F_SETFL, flags | O_NONBLOCK);
```

Source Code 1: Socket Code.

4.3 Login and Signup

The thread reads the command from the client and acts accordingly: if it is a login command, it will search to find a matching in the DOM tree of the users.xml file, if it is a signup command, it will search to ensure the username given by the client is not already taken. The thread will send a message to the client whether the command was executed successfully or not.

```
        if (strstr(command, "login") != NULL)
    {
        for (xml_node<> *user = root_node->first_node("user"); user && !user_found; user = user->next_sibling())
        {
            if (strcmp(user->value(), username) == 0)
            {
                printf("[Thread] User found!\n");
                strcat(command_response, "Login successful!\n");
                user_found = true;
            }
        }
        if (user_found == false)
        {
            printf("[Thread] User not found!\n");
            strcat(command_response, "Login unsuccessful! User does not exist. \n");
            user_found = false; }}
```

Source Code 2: Login Code.

Unlike the login, signup is done using Pugi library due to some problems encountered using Rapidxml.

```
        pugi::xml_document doc;
        if (!doc.load_file("users.xml")){
            cout << "[Server Thread] Could not load XML file." <<
 endl;
        }
        pugi::xml_node usersList = doc.child("UsersList");
        pugi::xml_node newUser = usersList.append_child("user");
        newUser.append_child(pugi::node_pcdata).set_value(
username);
        if (!doc.save_file("users.xml")){
            cout << "[Server Thread]Could not save XML file." <<
endl;
        }
        user_found = false;
        login_ok = true;
        logged_users.push_back(username);
        number_logged++;
```

Source Code 3: Signup Code.

4.4 Creating a room

After login, the client can choose to enter a game by creating its own room where they can wait for other clients to enter.

```
    int createRoom(int fd){
    Game g;
    g.Add_client(fd);
    active_games.push_back(g);
    number_of_games++;
    id_queue.push_back(g.id_uniq);
    return g.id_uniq;
}
```

Source Code 4: Create Room Code.

Here, before pushing the id in the *id_queue* vector, I will implement a way of ensuring that no two games have the same id.

4.5 Joining a room

All the clients that want to enter a specific room can do so by entering the id of that room that they want to join.

```
    int joinRoom(int fd, int id){
    for (int i = 0; i <= number_of_games; i++)
    {
        if (active_games[i].id_uniq == id)
        {
            active_games[i].Add_client(fd);
            printf("[Thread] Room found!\n");
            return 1;
        }
    }
    printf("[Thread] Room not found!\n");
    return 0;
}
```

Source Code 5: Join Room Code.

4.6 Generating an unique id for each room

The following code is responsible with generating an unique id for each room such that when a client wants to join a room no confusion is made.

```
        bool id_gasit = false;
        bool id_is_good = true;
        while (!id_gasit){
            for (int i = 0; i < number_of_games; i++){
                if (g.id_uniq == id_queue[i]){
                    id_is_good = false;
                    break;
                }
            }
            if (id_is_good == true){
                id_queue.push_back(g.id_uniq);
                id_gasit = true;
            }
            else{
                int newId = g.idGenerator();
            }
        }
```

Source Code 6: Id Code.

4.7 Extracting the questions

All the questions are stored in a XML file which has a list of questions, each node being a question with 4 different possible answers: the client will write the correct answer, exactly the same way it was printed on screen.

```
        for (xml_node<> *question_node = root_node->first_node("
    question");
                  question_node;
                  question_node = question_node->next_sibling())
        {
            bzero(question, 500);
            strcat(question, question_node->value());
            strcat(question, "\n");
            for (xml_node<> *answer_node = question_node->
    first_node("answer");
                      answer_node;
                      answer_node = answer_node->next_sibling())
            {
                strcat(question, answer_node->value());
                strcat(question, "\n");
            }
            write(fd, question, strlen(question));
        }}
```

Source Code 7: Questions Code.

4.8 Timer

The timer will be handled both in the client: I used select in order to make sure that the client cannot read from stdin after 15 second by setting the timeval struct with 15 seconds.

```
int result = select(STDIN_FILENO + 1, &readfds, NULL, NULL, &
  timeout);
if (result == -1){
  perror("[Client] Error in select");
}
else if (result == 0){
  printf("Timeout! No answer given.\n");
  strcpy(answer, " ");
}
else{
  ssize_t bytesRead = read(STDIN_FILENO, buffer, sizeof(buffer)
   - 1);
  if (bytesRead > 0){
    buffer[bytesRead] = '\0';
    strcpy(answer, buffer);
  }
```

Source Code 8: Timer Code.

4.9 Scoring

The scoring will be the same for all questions: 10 points if you answered correctly or 0 otherwise.

```
      if(strcmp(correct_answer, answer) == 0){
              write(fd, "the answer is correct! +10points\n",
  strlen("the answer is correct! +10points\n"));
              for(int i=0; i< client_number; i++){
                  if(clients[i] == fd)
                      scores[i] += 10;
              }
          }else{
              write(fd, "the answer is incorrect! +0points\n",
  strlen("the answer is incorrect! 0points\n"));
          }
```

Source Code 9: Scoring Code.

4.10 End of Game

When all the questions were broadcast-ed to all players, the game ends: the thread created when starting the game will send a tuple containing the 3 highest scores. If the game only has one player, the function will set the second and third elements in the tuple to -1 so that the client will know that there is only one winner to be printed. The same thing if there are only 2 players. If there are more than 3 (or exactly 3) the function returns the winners in order.

Thus, the data member **bool end_game** is set to true and the destructor of the class will be called so that the game ends successfully.

```
        ~Game(){
        if (end_game){
            cout << "[Game.h] The game was destructed!" << endl;
            delete[] clients;
            delete[] scores;
        }
    }
```

Source Code 10: End Code.

5 Conclusion

This code can be improved by implementing a fun GUI and by finding a way to show the timer at the same time with the questions.

In the end, this project is an amazing opportunity for me to put into practice what I learnt this year at Computer Networks, but also what i learnt last year at Object-Oriented Programming and Operating Systems courses.

6 Bibliographic References

- https://profs.info.uaic.ro/ computernetworks/files/5rc_ProgramareaInReteaI_en.pdf
- https://profs.info.uaic.ro/ computernetworks/files/NetEx/S9/servTcpCSel.c
- https://rapidxml.sourceforge.net/manual.html
- https://gist.github.com/JSchaenzle/2726944
- https://www.scottklement.com/rpg/socktut/nonblocking.html
- https://profs.info.uaic.ro/ georgiana.calancea/Laboratorul_12.pdf