QuizzGame

Rebegea Irina[1]

[1] Faculty of Computer Science, “Al. I Cuza” University of Iasi, Romania

Abstract. The project focuses on the development of a quiz application.
Keywords: quiz, multithreading, TCP.

1 Introduction

The theme is to build a trivia game by implementing a multi-threaded server that can accept an unlimited number of clients. The server coordinates clients that answer a set of questions, one at a time, in the order in which they register. Each client has a pre-defined number of seconds to answer a question. At the end of the game, the results obtained by all participants are centralised and the winner is shown to each participant.

The program contains two components, the server, and the client. All the logic is implemented in the server component, which maintains continuous communication with the client via socket. At the end of the game, the server breaks the connection with all clients.

2 Implemented technologies

2.1 TCP server

TCP (Transmission Control Protocol) is a connection-oriented transport protocol, which means that a connection must be established and maintained before data can be transferred. TCP has the following roles: it decides how packets will be split so that they can be delivered over the network; it sends and accepts packets from the network; it controls the flow of data; it retransmits packets that have been mixed up, ensuring that data is delivered without duplication or loss; it accepts incoming packets.

I chose this protocol because the server needs to be in permanent contact with the client. Also, sent questions and received answers should not be lost or sent in a different order, because corruption of the information would alter the outcome of the quiz.

2.2 Multithreading

In order to coordinate multiple clients simultaneously, I chose to use one thread for each client since this method is more efficient than creating *n* processes for *n* clients and takes up less memory space.

2.3 Sockets

For communication between client and server I used stream sockets, specific to the TCP protocol, as it provides support for UNIX.

2.4 SQLite database

Questions and answers are saved in a database with two tables. The link between the tables is made through the question_number field. Correct answers have in addition the attribute correct_answer with the value 1. When the database is searched, a question and the 4 possible answers are displayed. Here is a sequence containing some instructions for populating the tables:

```
-- populating the questions table
INSERT INTO questions VALUES(1, 'I) In which continent is the country Japan located?');
-- populating the answers table
INSERT INTO answers VALUES(1, '1. Asia', 1, 1);
INSERT INTO answers VALUES(1, '2. Europe', 0, 2);
INSERT INTO answers VALUES(1, '3. Africa', 0, 3);
INSERT INTO answers VALUES(1, '4. America', 0, 4);
```

The fields of the questions table are question_number and question. The fields of the answers table are question_number (foreign key), answer, correct_answer and answer_number.

3 Application design

3.1 Description of functions in each thread

The application has two components, the server and the client. Player information is stored in a struct and retrieved by the server in the register_user() function.

The start_game() function marks the quiz as "started" or disconnects the user, depending on the choice the user makes at the command line.

If the user enters the command "CTRL + C", they are removed from the competition.

When the questions are finished, the maximum of all scores is calculated and the winner's name is sent to all participants.

play_game() sends the questions and answer choices one at a time to the players, receives the players' answers within the set time limit and checks them, then calculates the current score for each player.

3.2 Application diagram

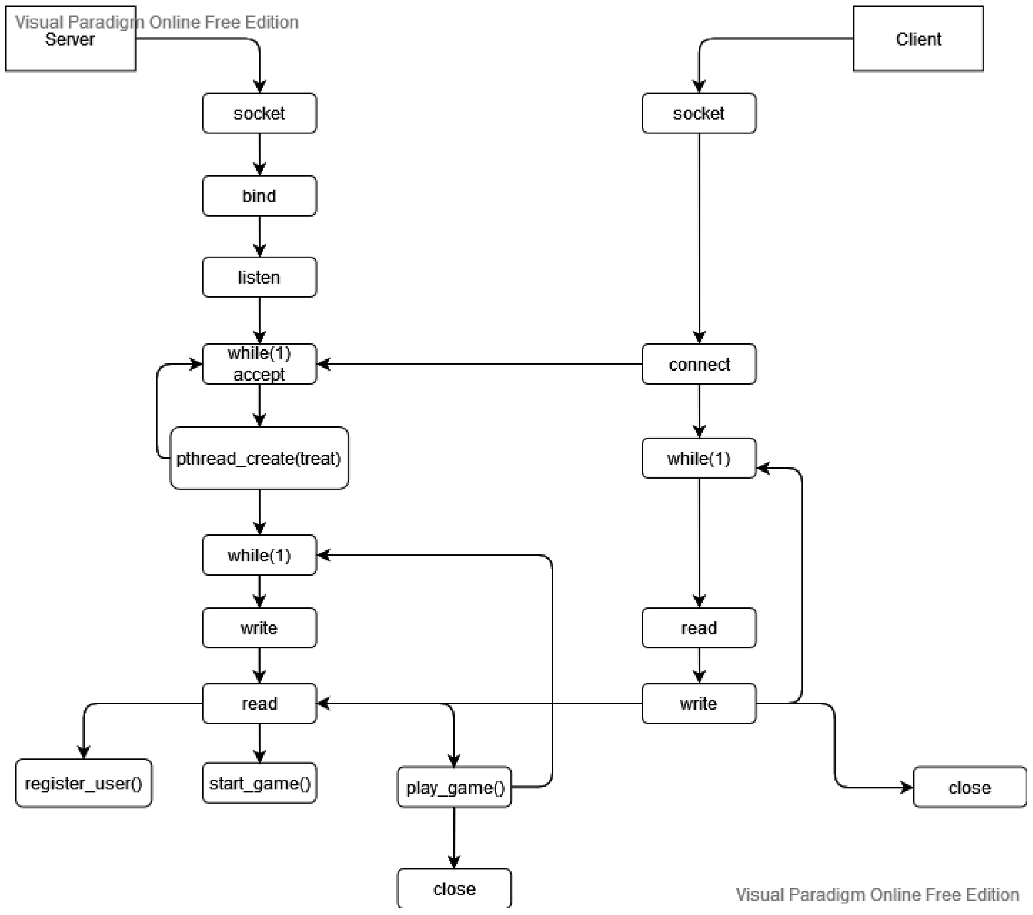

Figure 3.1: Application diagram

4 Implementation details

4.1 Communication protocol between client and server

Both client and server create a new connection endpoint using the socket() primitive. The server then attaches a local address to the socket via bind() and calls listen() to wait for connections. In an infinite loop, the server blockingly waits for a connection request with accept(). The client calls connect() and tries to establish the connection. After connecting, the server creates a new thread for the client and sends the message "Welcome! Please enter a username:". The client reads the message and passes it on to the user's screen, then takes the input from the user and writes it to the server. The server reads the message from the client and processes it using the functions described in the previous chapter. At the end of the quiz, the connection between server and client is closed with close().

A player can enter a username, then command "y" if they want to start the game, or "n" to be disconnected. After selecting "y" they will be shown the first question, with possible answer choices, and will have to answer within 5 seconds. After the time expires or after he answers, a message will be displayed on the screen with their current score and feedback (right/wrong answer). If the user wants to leave the game, they will be disqualified and disconnected from the server. At the end of the rounds, the player with the highest score will be shown to everyone.

4.2 Relevant pieces of code

The following code sequence implements the **treat()** function within a thread:

```
static void* treat(void* arg) {
/*
* Function implemented by every thread, in which all other func-
tions are called.
*/
char i = 49;
int value;
int ret = 1;
char buff[256] = { 0 };
char buff2[256] = { 0 };
struct thData tdL;
tdL = *((struct thData*)arg);
pthread_detach(pthread_self());
while (1) {
ok = 1;
highscore = 0;

if (registered_user[tdL.idThread] == 0) {
printf("[thread %d] Taking the username from the user...\n",
tdL.idThread);
register_user((struct thData*)arg);
printf("[thread %d] The player with the username %s has been reg-
istered.\n", tdL.idThread, player[tdL.idThread].username);
player_number++;
}

else if (registered_user[tdL.idThread] == 1) {
if (started_game[tdL.idThread] == 0 && game_over[tdL.idThread] ==
0) {
printf("[thread %d] Checking the start of the game...\n", tdL.id-
Thread);
```

```
start_game((struct thData*)arg);
}

else if (started_game[tdL.idThread] == 0 && game_over[tdL.id-
Thread] == 1) {
player_number--;
pthread_exit(NULL);
}


else if (started_game[tdL.idThread] == 1 && game_over[tdL.id-
Thread] == 0) {
printf("[thread %d] Game status: %d\n", tdL.idThread,
game_over[tdL.idThread]);
play_game((struct thData*)arg);
printf("[thread %d] Game status: %d\n", tdL.idThread,
game_over[tdL.idThread]);
}

else {
for (i = 0; i < player_number; i++) {
if (game_over[i] == 0)
ok = 0;
}
if (ok == 1) {
printf("[server] The score can be printed.\n");
if (write(tdL.cl, "Wait for the winner to be announced...\n", 40)
<= 0) {
printf("[Thread %d]\n", tdL.idThread);
perror("[Thread] write() error to client.\n");
game_over[tdL.idThread] = 1;
left[tdL.idThread] = 1;
}

for (tdL.idThread = 0; tdL.idThread < player_number; tdL.id-
Thread++) {
if (left[tdL.idThread] == 0) {
if (player[tdL.idThread].points >= highscore) {
       highscore = player[tdL.idThread].points;
       memset(winner, 0, sizeof(winner));
       strcpy(winner, player[tdL.idThread].username);

}
}
}


if (write(tdL.cl, winner, sizeof(winner)) <= 0) {
printf("[Thread %d]\n", tdL.idThread);
perror("[Thread] write() error to client.\n");
game_over[tdL.idThread] = 1;
left[tdL.idThread] = 1;
}
left[tdL.idThread] = 1;
player[tdL.idThread].status = 1;
player[tdL.idThread].points = 0;

pthread_exit(NULL);
}

else {
printf("[server] Wait a bit longer...\n");
if (write(tdL.cl, "Wait for the winner to be announced...\n", 40)
<= 0) {
printf("[Thread %d]\n", tdL.idThread);
perror("[Thread] write() error to client.\n");
game_over[tdL.idThread] = 1;
left[tdL.idThread] = 1;

}
sleep(3);
}
}
}
} /* while */

pthread_exit(NULL);
return(NULL);

  };
```

Upload questions and possible answers from the database:

```
void get_object(int n, sqlite3* db, void* arg)
{
/*
* Helper function that loads a question from the database and its
possible answers.
*/
struct thData tdL;
tdL = *((struct thData*)arg);
int rc;
int rc1;
int x;
char buff[256];
sqlite3_stmt* res, * res1;
char* sql_question = "SELECT question FROM questions WHERE ques-
tion_number = ?";
char* sql_answer = "SELECT answer FROM answers WHERE question_num-
ber = ?";

rc = sqlite3_prepare_v2(db, sql_question, -1, &res, 0);
rc1 = sqlite3_prepare_v2(db, sql_answer, -1, &res1, 0);

sqlite3_bind_int(res, 1, n);
sqlite3_bind_int(res1, 1, n);

int step = sqlite3_step(res);

memset(buff, 0, sizeof(buff));
if (step == SQLITE_ROW) {
strcpy(buff, sqlite3_column_text(res, 0));
strcat(buff, "\n");
if (write(tdL.cl, buff, sizeof(buff)) <= 0) {
printf("[Thread %d]\n", tdL.idThread);
perror("[Thread] write() error to client.\n");
game_over[tdL.idThread] = 1;
left[tdL.idThread] = 1;
}
}
int i = 4;
while (i > 0) {
int step1 = sqlite3_step(res1);
if (step1 == SQLITE_ROW) {
memset(buff, 0, sizeof(buff));
strcpy(buff, sqlite3_column_text(res1, 0));
strcat(buff, "\n");
if (write(tdL.cl, buff, sizeof(buff)) <= 0) {
printf("[Thread %d]\n", tdL.idThread);
perror("[Thread] write() error to client.\n");
game_over[tdL.idThread] = 1;
left[tdL.idThread] = 1;
}
i--;
}
}
}
```

Checking the answer received from the player:

```
void check_answer(int n, sqlite3* db, void* arg, char* buff1)
{
/*
* Helper function that checks the answer received from the client
with the correct answer in the database.
```

```
*/
struct thData tdL;
tdL = *((struct thData*)arg);

char buff[256]; /* right answer */
char punctaj[256];
int rc;
int j;

memset(buff, 0, sizeof(buff));
memset(punctaj, 0, sizeof(punctaj));

// --------------------------------
sqlite3_stmt* res;

char* sql_answer = "SELECT answer_number FROM answers WHERE ques-
tion_number = ? AND correct_answer = 1;";

rc = sqlite3_prepare_v2(db, sql_answer, -1, &res, 0);

sqlite3_bind_int(res, 1, n);

int step = sqlite3_step(res);

if (step == SQLITE_ROW) {
strcpy(buff, sqlite3_column_text(res, 0));
strcat(buff, "\n");

}

if (atoi(buff1) == atoi(buff)) {
player[tdL.idThread].points = player[tdL.idThread].points + 10;
if (write(tdL.cl, "The answer is correct! Current score: ", 39) <=
0) {
printf("[Thread %d]\n", tdL.idThread);
perror("[Thread] write() error to client.\n");
game_over[tdL.idThread] = 1;
left[tdL.idThread] = 1;

}
int j = snprintf(punctaj, sizeof(punctaj), "%d\n", player[tdL.id-
Thread].points);

if (write(tdL.cl, punctaj, sizeof(punctaj)) <= 0) {
printf("[Thread %d]\n", tdL.idThread);
perror("[Thread] write() error to client.\n");
game_over[tdL.idThread] = 1;
left[tdL.idThread] = 1;
}
}

else {
if (write(tdL.cl, "The answer is wrong. Better luck next time!
Current score: ", 60) <= 0) {
printf("[Thread %d]\n", tdL.idThread);
perror("[Thread] write() error to client.\n");
game_over[tdL.idThread] = 1;
left[tdL.idThread] = 1;

}
int j = snprintf(punctaj, sizeof(punctaj), "%d\n", player[tdL.id-
Thread].points);
if (write(tdL.cl, punctaj, sizeof(punctaj)) <= 0) {
printf("[Thread %d]\n", tdL.idThread);
perror("[Thread] write() error to client.\n");
game_over[tdL.idThread] = 1;
left[tdL.idThread] = 1;
}
}
}
```

5 Conclusions

The application can be improved by designing an attractive graphical interface that makes it easier to communicate with users.

Another idea is to display the question that other clients have reached when a new client registers during a started game, instead of having each client start with the first question.

6 References

https://ro.wikipedia.org/wiki/Transmission_Control_Protocol
https://www.techtarget.com/searchnetworking/definition/TCP
https://profs.info.uaic.ro/~computernetworks/files/5rc_ProgramareaInReteaI_ro.pdf
https://profs.info.uaic.ro/~computernetworks/files/6rc_ProgramareaInReteaII_Ro.pdf
https://profs.info.uaic.ro/~computernetworks/files/7rc_ProgramareaInReteaIII_Ro.pdf