

Online Quiz System Using Client-Server Architecture

Project Overview

The project focuses on the development of a client-server quiz application using C++ and socket programming. This system is designed to facilitate real-time, interactive quizzes where multiple clients can connect to a central server, answer questions, and receive instant feedback on their performance. The server manages the quiz by sending questions to all connected clients, collecting their answers, and calculating their scores. The primary objective of this project is to demonstrate the practical application of socket programming in creating a scalable and interactive assessment tool that can be used in educational, training, or entertainment contexts.

Introduction

Interactive learning tools have gained significant importance in education, training, and even entertainment. Quizzes, in particular, are a popular method for assessing knowledge, offering immediate feedback, and engaging participants. However, the implementation of such tools, especially in a networked environment where multiple users interact simultaneously, requires a robust and scalable architecture. This project addresses this need by developing a client-server-based quiz application using socket programming in C++.

The client-server model is an effective way to handle multiple users concurrently, allowing a central server to manage the quiz while clients participate from different locations. This approach not only ensures real-time interaction but also lays the groundwork for further enhancements, such as user authentication, question randomization, and database integration.

This project will provide valuable insights into the implementation of networked applications, particularly in the context of educational technology, and will serve as a stepping stone for more advanced applications.

Problem Statement

Development of a Client-Server Quiz Application Using Socket Programming

In today's educational and training environments, the need for interactive, scalable, and real-time assessment tools is paramount. Traditional methods of administering quizzes can be limiting, particularly when attempting to scale to multiple participants who are geographically dispersed. The challenge lies in creating a system that allows multiple users to simultaneously connect to a central server, participate in a quiz, and receive instant feedback, all while ensuring efficient and reliable communication.

Objective

The objective of this project is to design and implement a client-server quiz application using socket programming in C++. The system will enable multiple clients to interact with a central

server that manages the quiz, processes responses, and calculates scores in real-time. This project will address key technical challenges such as handling multiple client connections, ensuring secure data transmission, and providing a user-friendly interface.

This project not only provides a practical solution to the problem of scalable quiz administration but also demonstrates the principles of socket programming and client-server architecture, contributing to the broader field of educational technology.

The specific objectives are:

- To establish a connection between the client and the server using socket programming.
- To send quiz questions from the server to the client.
- To receive answers from the client and track the client's score on the server.
- To display the quiz questions and receive answers on the client-side.
- To display the final score to the client at the end of the quiz.

System Requirements

- **Hardware Requirements:**

Operating System: Linux/Unix-based

Processor: Multi-core processor

Memory: 4 GB of RAM or more

Storage: 10 GB or more

- **Software Requirements**

Programming Language C++

Libraries: POSIX Sockets API (sys/socket.h, arpa/inet.h, unistd.h)

Compiler: GCC

The client-server quiz application encompasses the following core functionalities:

Functionality

1. Server-Side Functionality:

- **Client Management:** Allows for the simultaneous acceptance and management of connections from several clients.
- **Quiz Administration:** Distributes quiz questions to clients, receives and processes their responses.
- **Score Calculation:** Involves comparing the client's responses with the right answers and computing a score.
- **Result Communication:** Sends the final score to each client after quiz completion.

2. Client-Side Functionality:

- **Connection Setup:** Establishes a connection to the server.
- **User Interaction:** Prompts the user for their name and quiz answers.
- **Answer Submission:** Sends user answers to the server for evaluation.
- **Score Display:** Receives and displays the final score from the server.

Modules

Server Components:

Socket Server: The server is responsible for creating and managing network connections with clients.

Quiz Manager: Handles the quiz's administration, sends questions to clients, and processes their responses.

Score Calculator: Evaluates client answers, calculates scores, and sends results back to clients.

Data Communication Handler: Manages the sending and receiving of data between the server and clients.

Client Component:

Socket Client: Establishes a connection to the server and maintains communication.

User Interface: Provides prompts for user input, including quiz answers, and displays questions and scores.

Answer Processor: Sends user answers to the server and receives responses.

Score Display: Receives and displays the final score from the server.

Source code:

Server Code:

```
#include <iostream>
#include <string>
#include <vector>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <unistd.h>
```

```
#define PORT 8080
```

```
#define BUFFER_SIZE 1024
```

```
struct Question {  
    std::string question;  
    std::vector<std::string> options;  
    int correctOption; // Index of the correct option (0-based)  
};
```

```
std::vector<Question> questions = {  
    {"What is the capital of France?", {"1) Paris", "2) London", "3) Rome", "4) Madrid"}, 0},  
    {"What is 2 + 2?", {"1) 3", "2) 4", "3) 5", "4) 6"}, 1},  
    {"What is the capital of Spain?", {"1) Lisbon", "2) Madrid", "3) Berlin", "4) Rome"}, 1},  
    {"What is the largest planet in our Solar System?", {"1) Earth", "2) Mars", "3) Jupiter", "4) Saturn"}, 2},  
    {"What is the chemical symbol for water?", {"1) H2O", "2) O2", "3) CO2", "4) NaCl"}, 0}  
};
```

```
void handleClient(int client_socket) {  
    char buffer[BUFFER_SIZE] = {0};  
    int score = 0;  
  
    // Ask for the client's name  
    std::string namePrompt = "Enter your name: ";  
    send(client_socket, namePrompt.c_str(), namePrompt.length(), 0);  
    recv(client_socket, buffer, BUFFER_SIZE, 0);  
    std::string client_name(buffer);  
    memset(buffer, 0, BUFFER_SIZE);  
  
    // Send quiz questions
```

```

for (const auto& q : questions) {
    std::string questionWithOptions = q.question + "\n";
    for (const auto& option : q.options) {
        questionWithOptions += option + "\n";
    }

    send(client_socket, questionWithOptions.c_str(), questionWithOptions.length(), 0);
    recv(client_socket, buffer, BUFFER_SIZE, 0);

    int client_answer = atoi(buffer) - 1; // Convert to 0-based index
    if (client_answer == q.correctOption) {
        score++;
    }
    memset(buffer, 0, BUFFER_SIZE);
}

// Send the final score
std::string scoreMessage = "Your score: " + std::to_string(score) + "/" +
std::to_string(questions.size());
send(client_socket, scoreMessage.c_str(), scoreMessage.length(), 0);

std::cout << "Client " << client_name << " scored " << score << " out of " <<
questions.size() << std::endl;

close(client_socket);
}

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    socklen_t addrlen = sizeof(address);

```

```

if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
    perror("Socket creation failed");
    return 1;
}

address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
    perror("Bind failed");
    return 1;
}

if (listen(server_fd, 3) < 0) {
    perror("Listen failed");
    return 1;
}

std::cout << "Server connected. Waiting for clients..." << std::endl;

while (true) {
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address, &addrlen)) < 0) {
        perror("Accept failed");
        continue;
    }

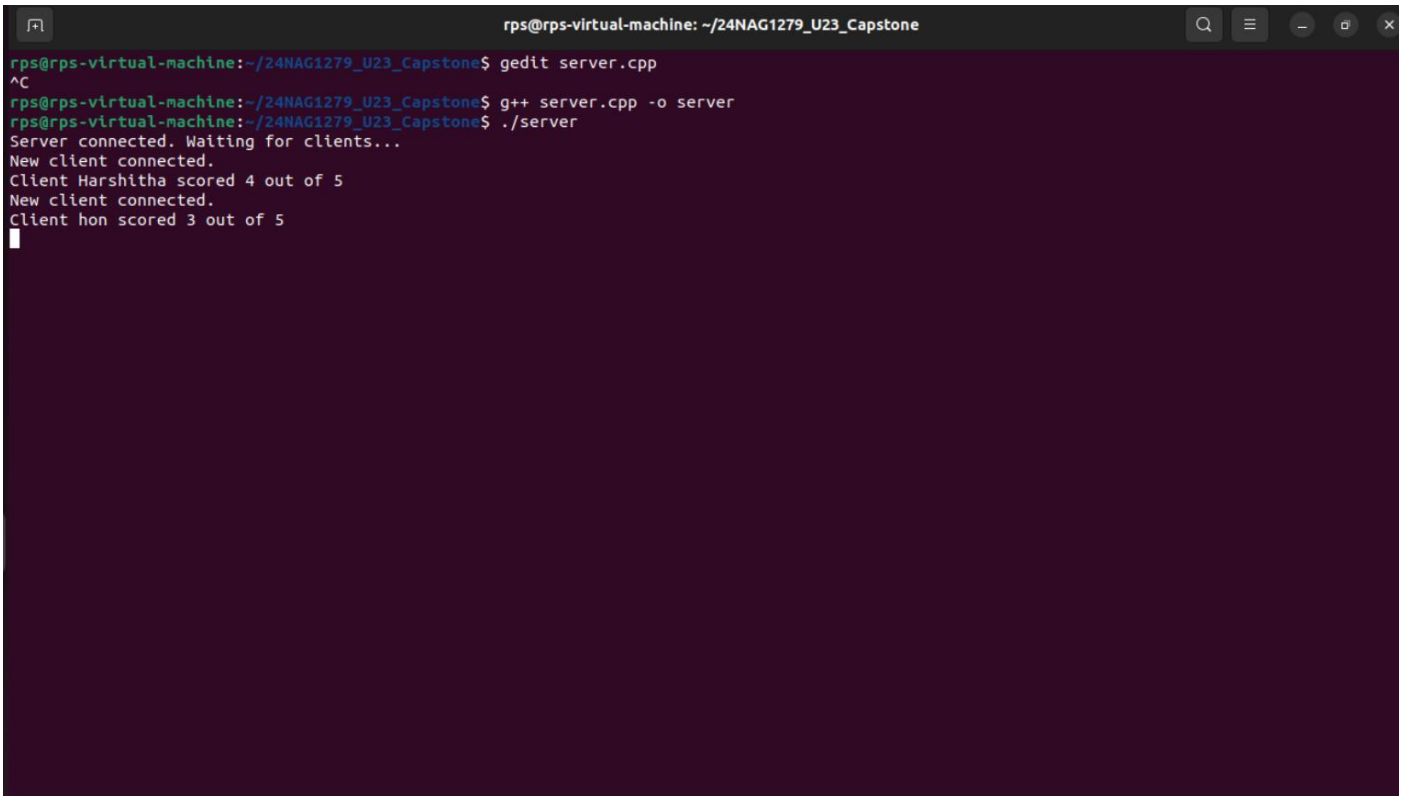
    std::cout << "New client connected." << std::endl;
    handleClient(new_socket);
}

return 0;

```

}

Server Code Output

A terminal window with a dark purple background. The title bar reads 'rps@rps-virtual-machine: ~/24NAG1279_U23_Capstone'. The terminal shows the following commands and output:

```
rps@rps-virtual-machine:~/24NAG1279_U23_Capstone$ gedit server.cpp
^C
rps@rps-virtual-machine:~/24NAG1279_U23_Capstone$ g++ server.cpp -o server
rps@rps-virtual-machine:~/24NAG1279_U23_Capstone$ ./server
Server connected. Waiting for clients...
New client connected.
Client Harshitha scored 4 out of 5
New client connected.
Client hon scored 3 out of 5
```

Client Code:

```
#include <iostream>
```

```
#include <string>
```

```
#include <cstring>
```

```
#include <sys/socket.h>
```

```
#include <arpa/inet.h>
```

```
#include <unistd.h>
```

```
#define PORT 8080
```

```
#define BUFFER_SIZE 1024
```

```
int main() {
```

```
    int sock = 0;
```

```
    struct sockaddr_in serv_addr;
```

```
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
```

```
perror("Socket creation failed");  
return 1;  
}
```

```
serv_addr.sin_family = AF_INET;  
serv_addr.sin_port = htons(PORT);
```

```
if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {  
    perror("Invalid address/ Address not supported");  
    return 1;  
}
```

```
if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {  
    perror("Connection failed");  
    return 1;  
}
```

```
char buffer[BUFFER_SIZE] = {0};
```

```
// Send the user's name
```

```
std::string name;  
read(sock, buffer, BUFFER_SIZE);  
std::cout << buffer << std::endl;  
std::cin.ignore();  
std::getline(std::cin, name);  
send(sock, name.c_str(), name.length(), 0);
```

```
// Receive and answer quiz questions
```

```
for (int i = 0; i < 5; i++) {  
    read(sock, buffer, BUFFER_SIZE);  
    std::cout << "Question: " << buffer << std::endl;  
    memset(buffer, 0, BUFFER_SIZE);
```



```

std::string answer;

std::cout << "Enter the option number: ";

std::getline(std::cin, answer);

send(sock, answer.c_str(), answer.length(), 0);
}

// Receive and display the final score
read(sock, buffer, BUFFER_SIZE);

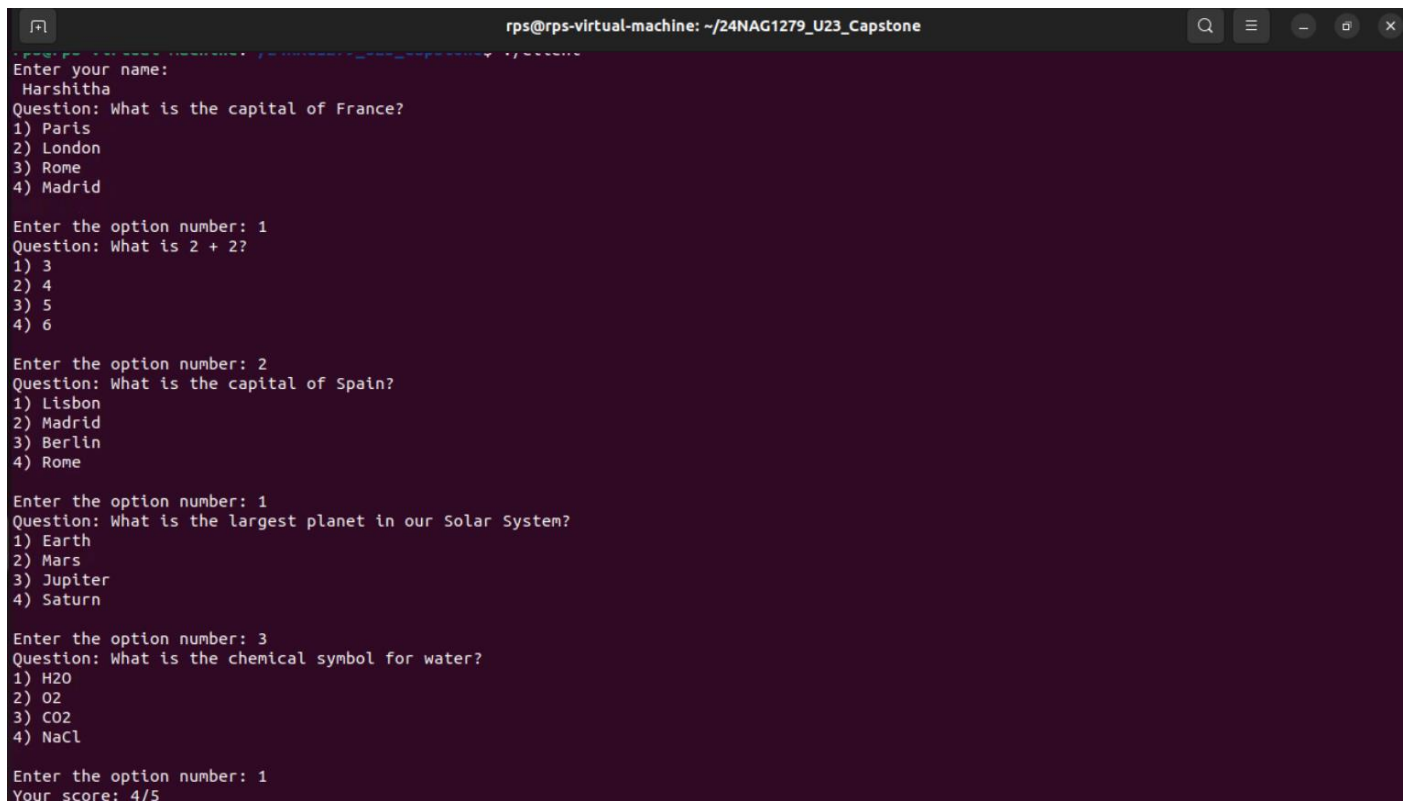
std::cout << buffer << std::endl;

close(sock);

return 0;
}

```

Client Code Output:



```

rps@rps-virtual-machine: ~/24NAG1279_U23_Capstone
Enter your name:
Harshitha
Question: What is the capital of France?
1) Paris
2) London
3) Rome
4) Madrid
Enter the option number: 1
Question: What is 2 + 2?
1) 3
2) 4
3) 5
4) 6
Enter the option number: 2
Question: What is the capital of Spain?
1) Lisbon
2) Madrid
3) Berlin
4) Rome
Enter the option number: 1
Question: What is the largest planet in our Solar System?
1) Earth
2) Mars
3) Jupiter
4) Saturn
Enter the option number: 3
Question: What is the chemical symbol for water?
1) H2O
2) O2
3) CO2
4) NaCl
Enter the option number: 1
Your score: 4/5

```

Testing

1. Compilation and Setup:

Both the server (server.cpp) and client (client.cpp) applications were compiled successfully using g++. Server and client programs were executed in separate terminals or machines.

2. Functionality Testing:

Connection: Verified that the client successfully connects to the server.

Quiz Interaction: Ensured the client receives quiz questions, submits answers, and receives the correct score.

Scoring: Checked that correct answers are awarded points and incorrect answers are handled appropriately.

3. Usability Testing:

Prompts: Confirmed that user prompts and instructions are clear and understandable.

Input Handling: Tested that the client correctly processes user input and handles responses accurately.

4. Performance Testing:

Response Time: Measured the server's response time to client requests and ensured it was within acceptable limits.

Concurrency: Conducted tests with multiple clients to assess the server's ability to handle simultaneous connections.

5. Security Testing:

Data Integrity: Verified that data transmitted between the client and server remains intact and unaltered.

Input Validation: Ensured the server correctly validates inputs to prevent injection attacks or other security issues.

Conclusion

In conclusion, this project successfully implemented a client-server-based quiz game using C++. The server handles multiple client connections sequentially, presenting a series of quiz questions and calculating scores based on client responses. On the client side, users connect to the server, receive questions, submit answers, and view their final scores. The project effectively demonstrated basic socket programming principles and interactive application development. However, there are opportunities for future enhancements, including adding concurrency to handle multiple clients simultaneously, improving error handling and input validation, developing a graphical user interface for a better user experience, and incorporating security measures to safeguard data transmission. Overall, the project provides a solid foundation for understanding client-server communication and interactive software development, with potential for further growth and refinement.