Capstone project

Introduction

The Real-Time Stock Market System is a client-server application designed to provide a platform for real-time stock trading and market updates. The system allows clients to execute buy and sell orders, retrieve stock prices, and receive updates on stock information. With the growing need for timely and accurate stock market information, it is crucial to develop a secure, efficient, and scalable platform that can handle multiple transactions and provide up-to-date stock data. The Real-Time Stock Market System aims to meet this need by offering a robust and responsive solution for real-time stock trading.

Objective:

The primary objective of the Real-Time Stock Market System is to deliver a secure, efficient, and real-time trading platform for users. The system is designed to handle multiple concurrent clients, process buy and sell orders accurately, and provide up-to-date stock market information. The specific objectives of the Real-Time Stock Market System are:

- To develop a secure and reliable real-time trading system capable of handling numerous transactions and clients.
- To provide a user-friendly interface for clients to execute buy/sell orders and retrieve stock data.
- To ensure the accuracy and integrity of stock price information and transaction processing.
- To provide real-time updates on stock prices and trading activity.
- To manage multiple client connections efficiently using a multi-threaded server.

Methodology:

1. Requirements Gathering

- Identify the requirements of the stock market application:
- Handle GET_STOCK_PRICES requests to retrieve current stock prices.

- Handle BUY and SELL requests to process stock transactions.
- Implement a server-client architecture to facilitate communication between clients and the stock market server.

2. Design

- Design the server-side logic:
- Create a handleRequest function to process incoming requests from clients.
- Implement a std::map to store stock prices and symbols.
- Define the logic for handling GET_STOCK_PRICES, BUY, and SELL requests.
- Design the client-side logic:
- Create a client program that can send requests to the server.
- Implement a user interface to input commands (GET_STOCK_PRICES, BUY, SELL).

3. Implementation

- Implement the server-side logic:
- Create a socket and bind it to a specific address and port.
- Listen for incoming connections and accept them.
- Receive requests from clients, process them using the handleRequest function, and send responses back to clients.
- Implement the client-side logic:
- Create a socket and connect to the server.
- Send requests to the server and receive responses.
- Display the received responses to the user.

4. Testing

- Test the server-side logic:
- Verify that the server can handle multiple clients simultaneously.

- Test the handleRequest function with different input requests.
- Verify that the server sends correct responses to clients.
- Test the client-side logic:
- Verify that the client can connect to the server successfully.
- Test the client's user interface to input commands.
- Verify that the client receives correct responses from the server.

5. Deployment

- Deploy the server program on a suitable platform (e.g., Linux, Windows).
- Deploy the client program on a suitable platform (e.g., Linux, Windows).
- Configure the server and client programs to communicate with each other.

6. Maintenance

- Monitor the server and client programs for errors and performance issues.
- Update the stock prices and symbols in the server's std::map as needed.
- Refactor the code to improve performance, security, and maintainability.
- By following this methodology, you can ensure that your stock market application is designed, implemented, and tested thoroughly to meet the requirements of your users.

System Requirements

Hardware Requirements

- Operating System: Linux/Unix-based
- Processor: Multi-core processor
- Memory: 4 GB RAM or more
- Storage: 10 GB or more

Software Requirements

- Programming Language: C++

- Libraries: sys/socket.h, netinet/in.h, arpa/inet.h, unistd.h
- Compiler: GCC

Source code :

Server code :

```
   #include <iostream>
#include <string>
#include <map>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
std::map<std::string, float> stockPrices = {
        {"AAPL", 150.0},
        {"GOOG", 2500.0},
        {"MSFT", 200.0}
};
std::string handleRequest(const std::string& request) {
        if (request == "GET_STOCK_PRICES") {
                std::string response;
                for (const auto& [symbol, price] : stockPrices) {
                        response += symbol + ": $" + std::to_string(price) + "\n";
                }
                return response;
        } else if (request.substr(0, 4) == "BUY ") {
```

```
            std::string symbol = request.substr(4);
            if (stockPrices.find(symbol) != stockPrices.end()) {
                // In a real application, you would handle inventory and purchasing logic
                return "Order placed to buy " + symbol;
            } else {
                return "Stock symbol not found.";
            }
        } else if (request.substr(0, 5) == "SELL ") {
            std::string symbol = request.substr(5);
            if (stockPrices.find(symbol) != stockPrices.end()) {
                // In a real application, you would handle inventory and selling logic
                return "Order placed to sell " + symbol;
            } else {
                return "Stock symbol not found.";
            }
        } else {
            return "Invalid command.";
        }
}
int main() {
        // Create socket
        int sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd < 0) {
            perror("socket creation failed");
```

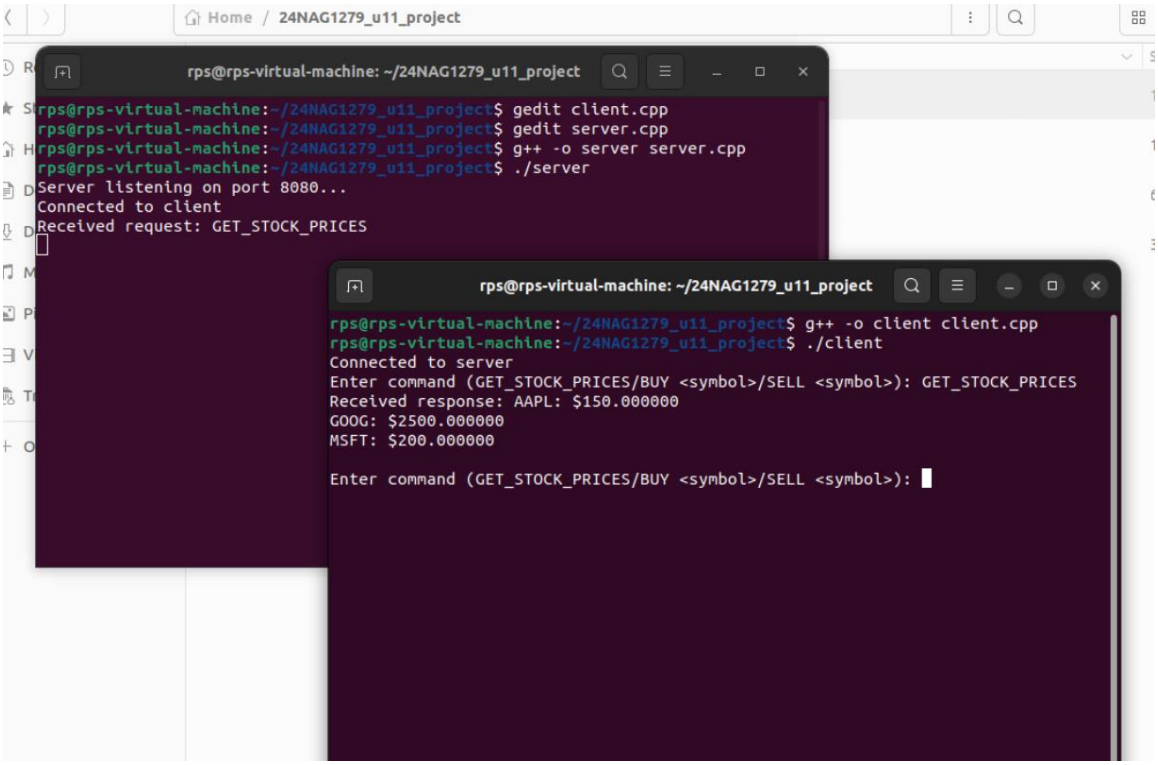
```
            return 1;
        }
         // Set address and port
        struct sockaddr_in addr;
        addr.sin_family = AF_INET;
        addr.sin_port = htons(8080);
        inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);
  // Bind socket to address and port
        if (bind(sockfd, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
            perror("bind failed");
            return 1;
        }
// Listen for incoming connections
        if (listen(sockfd, 3) < 0) {
            perror("listen failed");
            return 1;
        }
std::cout << "Server listening on port 8080..." << std::endl;
  while (true) {
            // Accept incoming connection
            struct sockaddr_in client_addr;
            socklen_t client_len = sizeof(client_addr);
            int client_sockfd = accept(sockfd, (struct sockaddr *)&client_addr, &client_len);
            if (client_sockfd < 0) {
```

```
            perror("accept failed");
            continue;
        }
    std::cout << "Connected to client" << std::endl;
  // Receive request from client
        char buffer[1024] = {0};
        int bytesReceived = recv(client_sockfd, buffer, 1024, 0);
        if (bytesReceived < 0) {
            perror("recv failed");
            close(client_sockfd);
            continue;
        }
          std::string request(buffer, bytesReceived);
        std::cout << "Received request: " << request << std::endl;
       // Handle request and send response to client
        std::string response = handleRequest(request);
        send(client_sockfd, response.c_str(), response.size(), 0);
       // Close socket
        close(client_sockfd);
    }
close(sockfd);
    return 0;
}
```

Client code

```
#include <iostream>
#include <string>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
int main() {
	// Create socket
	int sockfd = socket(AF_INET, SOCK_STREAM, 0);
	if (sockfd < 0) {
		perror("socket creation failed");
		return 1;
	}
  // Set address and port
	struct sockaddr_in addr;
	addr.sin_family = AF_INET;
	addr.sin_port = htons(8080);
	inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);
  // Connect to server
	if (connect(sockfd, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
		perror("connect failed");
		return 1;
	}
std::cout << "Connected to server" << std::endl;
```

```
while (true) {
        // Enter command
        std::string command;
        std::cout << "Enter command (GET_STOCK_PRICES/BUY <symbol>/SELL <symbol>): ";
        std::getline(std::cin, command);
// Send request to server
        send(sockfd, command.c_str(), command.size(), 0);
// Receive response from server
        char buffer[1024] = {0};
        int bytesReceived = recv(sockfd, buffer, 1024, 0);
        if (bytesReceived < 0) {
            perror("recv failed");
            return 1;
        }
    // Print received response
        std::string response(buffer, bytesReceived);
        std::cout << "Received response: " << response << std::endl;
    }
// Close socket
    close(sockfd);
    return 0;
}
```


Home / 24NAG1279_u11_project
rps@rps-virtual-machine: ~/24NAG1279_u11_project
rps@rps-virtual-machine:~/24NAG1279_u11_project$ gedit client.cpp
rps@rps-virtual-machine:~/24NAG1279_u11_project$ gedit server.cpp
rps@rps-virtual-machine:~/24NAG1279_u11_project$ g++ -o server server.cpp
rps@rps-virtual-machine:~/24NAG1279_u11_project$ ./server
Server listening on port 8080...
Connected to client
Received request: GET_STOCK_PRICES
rps@rps-virtual-machine: ~/24NAG1279_u11_project
rps@rps-virtual-machine:~/24NAG1279_u11_project$ g++ -o client client.cpp
rps@rps-virtual-machine:~/24NAG1279_u11_project$ ./client
Connected to server
Enter command (GET_STOCK_PRICES/BUY <symbol>/SELL <symbol>): GET_STOCK_PRICES
Received response: AAPL: $150.000000
GOOG: $2500.000000
MSFT: $200.000000
Enter command (GET_STOCK_PRICES/BUY <symbol>/SELL <symbol>):