metin, daire, logo, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Operating Systems Course Project Report**

**Name-Surname:** Berkay Gümüşay

**Student ID:** 21011084

**E-mail:** [berkay.gumusay@std.yildiz.edu.tr](mailto:berkay.gumusay@std.yildiz.edu.tr)

**Project Contents**

1. Project Description………………………………………………………..3
2. Solution Of The Project………………………………………………….3
3. server.c …………………………………………………………………………4
   1. Short Code Description …………………………………………4
   2. Function Prototypes ……………………………………………..4
4. client.c ………….………………………………………………………………7
   1. Short Code Description..………………………………………..7
   2. Function Prototypes………………………………………………7
5. Explanation Of The Operations………………………………………9
6. Screen Outputs……………………………………………………………12

**1- Project Description**

We have been asked to program a messaging application that performs information exchange between programs through sockets.

**2- Solution of the Project**

This project can be implemented through socket programming, ensuring communication between the server and clients, and enabling multiple clients to connect to the server using multithreading.

**3- server.c**

**a) Short code description**

This code creates the socket on the server side and provides a multithreaded server structure that performs the processed tasks in the project.

**b) Functions Prototypes**

**1) <int> main( )**

This function enables the establishment of the socket structure and the creation of a multithreading-friendly framework. With the “while(1)” and the code within it, it continuously listens to the client.

**2) <void> handleClient(void\*)**

This function performs the necessary operation based on the information sent from the client. In the case of multiple client connections, it ensures the multithreading process.

**3) <char\*\*> parseString(char\*, int\*)**

This function parses the character array sent to it into segments using a comma and stores the string number.

**4) <int> findFromSubfolders(char\*)**

This function verifies the existence of the userID value sent to it in the system.

**5) <int> createRequiredFiles(user)**

This function creates the necessary files for the user after the user registration process.

**6) <void> createFolder(char\*)**

This function creates a folder with the same name as the userID.

**7) <int> addUsers(char\*, SOCKET)**

This function adds the users sent by the client to the friend list of the logged-in user in the system.

**8) <int> deleteUsers(char\*, SOCKET)**

This function deletes the users sent by the client from the friend list of the logged-in user in the system.

**9) <int> isExistInFile(user\*, int, char\*)**

This function verifies if the input user is present in the friend list of the currently logged-in user in the system.

**10) <void> listContacts(char\*, SOCKET)**

This function checks whether the given userID is present in the user's connections.

**11) <int> isInFriends(char\*, char\*)**

This function performs the task of sending messages to the userID received from the client.

**12) <int> sendMessage(char\*, SOCKET)**

This function performs the task of sending messages to the userIDs received from the client.

**13) <void> addMessageHistory(char\*, char\*, char\*, char\*)**

This function adds the messages sent by the user to the “messageHistory.txt” where their message history is stored.

**14) <int> checkMessages(char\*, SOCKET)**

This function sends the unread messages of the logged-in user to the client.

**15) <void> messageReOrganize(char\*)**

This function changes unseen message to seen message.

**16) <void> showMessageBetweenUsers(char\*, SOCKET)**

This function sends the messaging history between two users to the client.

**17) <void> deleteMessage(char\*, SOCKET)**

This function deletes the selected message sent by the user.

**18) <void> showAllMessages (char\*, SOCKET)**

This function sends the entire messaging history of the user to the client.

**4- client.c**

**a) Short code description**

This code serves as the client-side implementation of a communication protocol with a server. It handles user authentication, interaction, and message checking through sockets, and it may involve multi-threading for asynchronous operations.

**b) Function Prototypes**

**1) <DWORD WINAPI> messageCheckingThread(LPVOID)**

This function represents a thread that continuously checks for new messages in a file and prints a message when a new one is detected.

**2) <void> sendMessage(SOCKET)**

This function takes the message to be sent and the ID of the user to whom the message will be sent as input and sends it to the server.

**3) <void> addUser(SOCKET)**

This function takes the IDs of the users to be added to the contact list as input and sends them to the server.

**4) <void> listContacts(SOCKET)**

This function prints the contact list of the user received from the server to the screen.

**5) <void> deleteUsers(SOCKET)**

This function, by sending the user ID received as input to the server, performs the operation of removing the user from the contact list.

**6) <void> checkMessages(SOCKET)**

This function prints unread messages received from the server to the screen.

**7) <void> showMessageBetweenUsers (SOCKET)**

This function prints the messaging history between two users to the screen.

**8) <void> deleteMessage(SOCKET)**

This function sends the ID of the message to be deleted to the server.

**9) <void> showAllMessages (SOCKET)**

This function prints the entire messaging history of the relevant user to the screen.

**5- Explanation Of The Operations**

1. **List Contacts:**

In this operation, initially, on the client side, the "listContacts()" function is used by the server to access the "friends.txt" file inside the folder with the same name as the user's ID. All user IDs inside the file are added to a character array and sent to the client. Upon receiving the character array, the client sequentially prints it to the screen.

1. **Add User:**

In this operation, first, on the client side, the number of users to be added and the IDs of the users to be added are obtained from the user. These values are then sent to the server. After receiving the values on the server side, existing users in the system are added, and non-existing users are not added. After all the operations are completed, a variable indicating whether the operation was successful or not, called "isSuccess," is sent to the client. Based on this value, a message is printed to the screen on the client side.

1. **Delete User:**

In this process, first, on the client side, the number of users to be deleted and the IDs of the users to be deleted are obtained from the user. These values are then sent to the server. After receiving the values on the server side, users present in the user's friend list are deleted, and users not present in the friend list are not deleted. After all the operations are completed, a variable indicating whether the operation was successful or not, called "isSuccess," is sent to the client. Based on this value, a message is printed to the screen on the client side.

1. **Send Message:**

In this process, first, on the client side, the user's ID and the message to be sent to the user are obtained from the user. These values are then sent to the server. After receiving the values on the server side, it is checked whether the user to whom the message will be sent is present in both the system and the friend list. If the user exists in both places, the message is written to the "messages.txt" file inside the folder with the same name as the user's ID. After all the operations are completed, a variable indicating whether the operation was successful or not, called "isSuccess," is sent to the client. Based on this value, a message is printed to the screen on the client side.

1. **Check Messages:**

In this process, on the server side, initially, the "messages.txt" file inside the folder with the same name as the user's ID is opened. Messages with the last index "0" (i.e., unseen messages) are collected into a character array. This character array is then sent to the client. On the client side, the received character array is printed to the screen.

Afterward, the "messagesReOrganize()" function is called, which changes the last indices of the seen messages from "0" to "1."

1. **Show Message History:**

In this process, on the server side, access is made to the "messageHistory.txt" file inside the folder corresponding to the user. All messages sent by the user are appended to a character array. Subsequently, this character array is sent to the client. On the client side, the received character array is printed to the screen.

1. **Delete Messages:**

In this process, initially, all messages sent by the user are printed to the screen. Subsequently, the content of the message selected by the user is sent to the server. On the server side, the received message content is used to delete the selected message from both the user's message history and the message box of the recipient.

**6- Screen Outputs**

**Ex. 1) Registration**

İnput:

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Output:



**Ex. 2) Login**

İnput:



Output:



**Ex. 3) Main Menu**

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Ex. 4) Empty Contact List**

Output:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Ex. 5) Add User 2**

Input& Output:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Ex. 6) List Contacts**

Output:

metin, yazı tipi, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Ex. 7) Delete User**

Before: Input : After:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, yazı tipi, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, yazı tipi, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Ex. 8) Send Message (User in the Contact List)**

Input & Output:

metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Ex. 9) Show Message History**

Input & Output:

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Ex. 10) Show Message History (User with No Sent Messages)**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Ex. 11) Send Message (User not in the Contact List)**

Input & Output:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Ex. 12) Check Messages (Empty Message Box)**

Output:

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Ex. 13) Check Messages (When Message Is Arrived)**

Main Menu:

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Message Box:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Ex. 14) Deleting Message**

Deleting a message that sent user 3 from user 1

Message History of User 1 (Before):

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Message Box Of User 2 (Before):

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Delete The Selected Message:

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Message History of User 1 (After):

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Message Box Of User 2 (After):

metin, yazı tipi, çizgi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu