1/20/2021

PF Final Project

Report

Members:

Saad Bin Khalid: 20k-0161 (C)

M. Mudabir: 20k-0273

Shazaib Qureshi: 19k-1511

Contents

C-Chat

Introduction:

“C-Chat” is an application using C-language which essentially is a communication platform of messages (strings). It consists of following *major* features:

1. A secure and efficient login and sign-up system.
2. A platform to communicate with all of the users of your application at once (like a public group chat).
3. A platform to communicate only with the client you want to chat with (a private connection with that client).

Although we have illustrated and set the codes such that the users will be able to communicate only through the local-host IP (127.0.0.1), the program also runs successfully on LAN or even the internet. The sockets used here follow the TCP protocol and Winsock programming functions. The program files are not shown in this report, but only their features and methodologies are discussed. A detailed explanation is in the comments of source codes.

Parts of The Program:

The application is divided into at least six major program files, out of which three are servers (all of them must be running), two are clients (you can include more if you want), and one is an important header file which is used by both servers and clients.

Functions and APIs:

Following are the major and notable functions and APIs used in this project:

FUNCTION	ACTION
socket(…)	Opens a required socket
bind(…)	Binds a socket to an address.
listen(…)	Make a socket listen for connections.
connect(…)	Connect to a server socket.
accept(…)	Accept the listening socket.
recv(…)	Receives a string from a socket.
send(…)	Sends a string to a socket.
SOCKET	A datatype for socket identifiers.
CLOSESOCKET(…)	Closes a socket from connectivity.
ISVALIDSOCKET(…)	Checks if a solid valid or not.
WSADATA	Data type for necessary Winsock buffers.
WSAStartup(…)	Starts Winsock APIs.
WSACleanup()	Closes Winsock
_kbhit()	Detects keyboard input.
Select(…)	Filters out available sockets.
fd_set	A data type for a set of socket identifiers.
FD_ZERO(…)	Zeros an fd_set.
FD_SET(…)	Includes another socket in the set
FD_ISSET(…)	Checks for connectivity of a socket in a set.
FD_CLR(…)	Removes a socket from the set.

User defined Functions:

Following functions were defined by the developer for efficiency:

FUNCTION	ACTION
listening(…);	Returns a listening socket at specified IP & port.
connection(…);	Returns a socket connected at specified IP & port.
login(…); (server)	Verifies the client data and lets the client pass.
login(…); (client)	Inputs data from user and sends to server.
createAcc(…); (server)	Receives and stores the data in a FILE.
createAcc(…); (client)	Inputs data from user and send it to server.

Algorithms:

1. The Header File:

Our important header file is named as *"header"* and includes some important Winsock APIs and pre-defined functions' libraries. It is necessary for our servers and clients to communicate with each other.

It also includes aliases of some functions. They are renamed as according to the preferences.

2. The Main Server:

Our welcoming and multiplexing server is named as 'MAIN_SERVER' in the program. It provides multiple options to the users at the starting.

No user can proceed ahead without being verified by MAIN_SERVER. Its basic working algorithm is:

1. Creates a socket “server” and starts listening on it.
2. Waits for a specified timeout for any connection or requests and then refreshes.
3. If a connection is detected by the server, it accepts the remote client and starts listening again for more.
4. When the request from the client is received, it splits its flow into two parts and deals with clients separately, allowing the socket to accept more clients.
5. The request is actually a choice or action which is then used by the server to call specific functions accordingly with separately deal with clients.
6. The login function accepts information from the client and only allow it to proceed further if it is valid by verifying it from a maintained file.
7. The sign-up function accepts information from the client and simple stores it in the file, latter to be used by the login function.

3. The Private Chats Server:

The “PRIVATE_SERVER” is responsible for establishing targeted private connections between its clients separately. It can accept any number of clients and connect them with their specified private clients.

Its basic working algorithm is:

1. Creates a socket “server” and starts listening on it.
2. Waits for a specified timeout for any connection or requests and then refreshes.
3. If a connection is detected by the server, it accepts the remote client and starts listening again for more.

4. When the request from the client is received, it splits its flow into two parts and deals with clients separately, allowing the socket to accept more clients.
5. The connection starts by two requests from the client which are to be received, one is the targeted client ID while the other is a string to be delivered to that ID.
6. It then requests all of its other clients for their IDs and receives them.
7. It checks the targeted ID one by one with the received IDs. When the received ID matches the targeted ID, it stops checking and sends the message to that specific client only.
8. Meanwhile, if another request is received, it repeats the above working again.

4. The Public Group Chat Server:

The “PUBLIC_SERVER” is responsible for establishing public connections between all of its clients. It can accept any number of clients and make them communicate with each other all at once.

Its basic working algorithm is:

1. Creates a socket “server” and starts listening on it.
2. Waits for a specified timeout for any connection or requests and then refreshes.
3. If a connection is detected by the server, it accepts the remote client and starts listening again for more.
4. When the request from the client is received, it splits its flow into two parts and deals with clients separately, allowing the socket to accept more clients.
5. The connection starts by a request which is actually a string to be delivered.

6. It then iterates through all of its online clients and sends each of them the received text.
7. Meanwhile, if another request is received, it repeats the above working again.

5. The Clients:

The "CLIENT_" is responsible for requesting and receiving connections with our three servers. It sends and receives the specific actions, errors, and strings of our program. It also stores the chats in text files which is again used to continue chatting from the left point.

Its basic working algorithm is divided into three connections:

1. Connection with MAIN_SERVER.
2. Connection with PRIVATE_SERVER.
3. Connection with PUBLIC_SERVE.

Out of these three, only the first one is mandatory; the other twos are optional and can only be selected one at a time.

1. With Main server:

1. Prompts the user to either login or sign-up and receives the choice.
2. Creates a socket "server" and connects at the MAIN_SERVER's listening port.
3. Sends the answer by the user to the server.
4. Calls specified functions which further deal with the server. Meanwhile, it is able to accept more connections.
5. The login function asks information from the user and send it to the server for verification.

6. Only when the server responds back with a success message, it allows the user to proceed ahead. Otherwise, it asks again for valid information.
7. Meanwhile, the sign-up function also asks information from the user and send it to the server for storing it in the file.
8. After this, the connection is aborted and the client is free to move ahead.

At this point, the user is asked to either opt for the PRIVATE_SERVER or for the PUBLIC_SERVER.

2. With Private Chat server:

1. Prompts the user to either create a new private contact or to open an existing one.
2. When a new contact is to made, it asks for a valid remote targeted ID and creates a new file for storing the private chats.
3. When an existing contact is to open, it asks for a valid remote targeted ID and opens the specified file to print the previous conversation.
4. Creates a socket “server” and connects at the PRIVATE_SERVER’s listening port.
5. Waits for a specified timeout for any action, either from the user or from the server, and then refreshes.
6. If input from the keyboard is detected, it stores the text.
7. It then sends the targeted ID to the server along with the string to be delivered.
8. It again waits for a specified timeout for any action, either from the user or from the server, and then refreshes.
9. If a request from the server is detected, it sends its own ID for receiving the remote string.

10. Meanwhile it also stores the conversation in the file and prints it on the screen.
11. If the user types a message starting from “xxx”, it aborts the connection.

3. With PUBIC_SERVER server:

1. Opens a file identified by the user’s ID. This file was automatically created during the sign-up session.
2. Prints previous conversations.
3. Creates a socket “server” and connects at the PUBLIC_SERVER’s listening port.
4. Waits for a specified timeout for any action, either from the user or from the server, and then refreshes.
5. If input from the keyboard is detected, it stores the text.
6. It then sends the string to be delivered to the server.
7. It again waits for a specified timeout for any action, either from the user or from the server, and then refreshes.
8. If a request from the server is detected, it immediately receives the text.
9. Meanwhile it also stores the conversation in the file and prints it on the screen.
10. If the user types a message starting from “xxx”, it aborts the connection.

When the connection from either of the PRIVATE_SERVER or the PUBLIC_SERVER is aborted, the user is guided to an option for exiting the application. If the user chooses to continue, it again asks him to either opt for the PRIVATE_SERVER or for the PUBLIC_SERVER.

Remember:

1. Always remember to start all of the servers before starting any client.
2. Do not delete or rename the files.
3. When the user is asked for a private connection, a valid ID should be entered. If the ID is not on the accounts file, there will be no communication.
4. Due to slow internet, the server sometimes gets hanged. Such problem is detectable and can be solved by opening that server and pressing enter.
5. A secret trip message is also useable. If you want to corrupt and shutdown the servers, send a sentence starting with “yyy”. This will close the server with all of its clients.

Conclusion:

In conclusion, the project is a small and efficient app for delivering strings which can be used for personal purposes and establishing web servers and clients. The TCP protocol is used which means that the messages are safe and will be delivered completely.