İÇİNDEKİLER

1 Introduction

This project is a multi-threaded chat app written in C. It uses the socket programming model to establish connections between multiple clients and the server. The server can handle multiple clients concurrently, with each client running on a separate thread.

The server maintains a list of all connected clients, storing relevant information such as the client's socket ID, user ID, name, surname, and telephone number. This information is stored in a struct called ‘client‘, and an array of these structs is maintained to keep track of all connected clients.

2
SERVER.C

The server listens for incoming connections on a specified port. When a client connects, the server accepts the connection, stores the client's details in the 'Client' array, and creates a new thread to handle the client's requests. This allows the server to handle multiple clients concurrently, as each client is serviced by its own thread.

2.0.1 SUPPORTED COMMANDS

The server supports several commands from the client:

2.0.1.1 CONFIG

This command is used to configure the client's details. The server receives the client's user ID, name, surname, and telephone number, and stores them in the 'Client' array. It also creates a directory for the client to store their chat history.

2.0.1.2 ADD

This command is used to add a contact to the client's contact list. The server receives the user ID of the contact to be added, checks if the contact exists, and if so, adds the contact to the client's contact list.

2.0.1.3 SEND

This command is used to send a message to another client. The server receives the user ID of the recipient and the message, checks if the recipient exists, and if so, sends the message to the recipient and stores the message in the chat history of both the sender and the recipient.

2.0.1.4 HISTORY

This command is used to view the chat history between the client and another user. The server receives the user ID of the other user, checks if the user exists, and if so, sends the chat history to the client.

2.0.1.5 LIST

This command is used to list all online clients. The server sends a list of all connected clients, excluding the client who sent the command.

The server continues to listen for and handle client requests until it is terminated. It uses mutexes and condition variables to ensure thread safety when accessing shared resources.

2.0.2 FUNCTIONS

2.0.2.1 main()

The ‘main‘ function in this code is the entry point of the server program.

1. It starts by creating a socket using the ‘socket‘ function. The parameters to this function specify that the socket should use the IPv4 protocol (PF_INET), the TCP protocol (SOCK_STREAM), and the IP protocol (0).

2. It then sets up a ‘sockaddr_in‘ structure for the server address. It specifies that the address should use the IPv4 protocol (AF_INET), the port number should be 8080 (converted to network byte order using ‘htons‘), and the IP address should be any valid address on the server (INADDR_ANY, also converted to network byte order).

3. The ‘bind‘ function is then called to bind the socket to the server address. If this fails (i.e., the function returns -1), the program returns 0 and exits.

4. The ‘listen‘ function is then called to make the socket listen for incoming connections. The second parameter to this function is the maximum number of pending connections. If this fails, the program also returns 0 and exits.

5. A message is printed to the console indicating that the server has started listening on port 8080.

6. The program then enters an infinite loop where it accepts incoming connections using the ‘accept‘ function. For each connection, it stores the client’s socket ID and address in the ‘Client‘ array, sets the client’s index, creates a new thread to handle the

client using the ‘doNetworking‘ function, and increments the client count.

7. After the loop (which in this case will never actually happen because the loop is infinite), the program waits for all client threads to finish using the ‘pthread_join‘ function.

The ‘doNetworking‘ function, which is not fully shown in the provided code, is expected to handle communication with each client. It takes a pointer to a ‘client‘ structure as a parameter, which it uses to get the client’s index and socket ID.

2.0.2.2 void *doNetworking(void *ClientDetail)

The ‘doNetworking‘ function is a thread function that handles communication with a client. It’s designed to be run in a separate thread for each client that connects to the server. Here’s a general explanation of what it does based on typical server-client communication patterns:

1. The function takes a single parameter, which is a pointer to a ‘client‘ structure. This structure contains information about the client, such as their socket ID and user ID.

2. At the start of the function, it casts the parameter to a ‘client‘ pointer and extracts the client’s index and socket ID.

3. It then enters a loop where it receives data from the client using the ‘recv‘ function. The received data is expected to be a command from the client.

4. Depending on the received command, the function calls the appropriate handler function. For example, if the command is "CONFIG", it calls a function to handle client configuration; if the command is "ADD", it calls a function to handle adding a contact; and so on.

5. The handler functions are expected to take the client’s index, the client’s socket ID, and the received data as parameters. They handle the command from the client, possibly receiving additional data from the client as needed, and send a response to the client.

6. The loop continues until the client disconnects, at which point the function returns and the thread terminates.

2.0.2.3 int getUserByIdIfExist(int userID)

The ‘getUserByIdIfExist‘ function is a utility function that checks if a user with a given ID exists among the connected clients.

Here’s a general explanation of what it does:

1. The function takes a single parameter, which is the user ID to check.

2. It then loops through the array of connected clients. For each client, it checks if the client’s user ID matches the given user ID.

3. If it finds a match, it returns the index of the client in the array. This index can then be used to access the client’s information in the array.

4. If it doesn’t find a match after checking all clients, it returns -1 to indicate that no client with the given user ID exists.

This function is useful for commands that require interacting with a specific client, such as sending a message or viewing chat history. By calling this function and checking its return value, the server can determine whether the requested operation is valid or not.

2.0.2.4 void writeToHistoryFile(int client_uID, int dest_uID, char *message)

1. The function takes three parameters: the user ID of the client sending the message, the user ID of the recipient client, and the message itself.

2. Purpose is to write the message to a history file. This is typically done to keep a record of all messages exchanged between clients.

3. The function start by constructing the filename of the history file.

4. Then opens the history file in append mode. If the file doesn’t exist, it’s created.

5. Writes the message to the file with its senders’ user ID.

This function is called whenever a "SEND" command is received from a client. After sending the message to the recipient client, the server calls this function to store the message in the history file.

2.0.2.5 void handleConfig(int index, int clientSocket, char *data);

The handleConfig function is used to handle the configuration of a client's details. The function takes three parameters: the index of the client in the Client array, the client's socket ID, and a data buffer.

The function first prints a message indicating that the client has requested configuration.

It then receives data from the client's socket. This data is expected to be the client's user ID, which is converted from a string to an integer using atoi and stored in the Client array.

The function then receives more data from the client's socket. This data is expected to be the client's name, surname, and telephone number, which are stored in the Client array.

Prints a message indicating that the client has been configured with the received details.

Then creates a directory for the client. The directory's name is the client's user ID. If the directory already exists, a message is printed to indicate this.

Finally, the function creates a subdirectory named "history" inside the client's directory. This subdirectory is likely used to store the client's chat history.

2.0.2.6 void handleAdd(int index, int clientSocket, char *data)

The handleAdd function is used to handle the addition of a new contact for a client.

The function takes three parameters: the index of the client in the Client array, the client's socket ID, and a data buffer.

First prints a message indicating that the client has requested to add a contact.

It then receives data from the client's socket. This data is expected to be the user ID of the contact the client wants to add, which is converted from a string to an integer using atoi.

Then opens a file named "contacts.txt" in the client's directory. This file is used to store the client's contacts.

The function checks if the contact already exists in the file. If the contact does not exist, the function adds the contact's user ID, name, surname, and telephone number to the file.

Finally, prints a message indicating that the contact has been added.

2.0.2.7 void handleSend(int index, int clientSocket, char* data)

This function is called when a client wants to send a message to another client. It takes three parameters: the index of the client in the Client array, the client's socket ID, and a data buffer. The function first receives data from the client's socket. This data is expected to be the user ID of the recipient client, which is converted from a string to an integer using atoi. The function checks if the recipient client exists. If the recipient client does not exist, the function returns without doing anything. The function then receives the message from the client's socket. The sender's user ID is added to the message. The message is written to the history files of both the sender and the recipient. Finally, the message is sent to the recipient client.

2.0.2.8 void handleHistory(int index, int clientSocket, char* data)

This function is called when a client wants to view the message history with another client. It takes three parameters: the index of the client in the Client array, the client's socket ID, and a data buffer. The function first receives data from the client's socket. This data is expected to be the user ID of the other client, which is converted from a string to an integer using atoi. The function checks if the other client exists. If the other client does not exist, the function sends a message to the client indicating that the other client does not exist and then returns. The function then opens the history file between the client and the other client. The function reads each line from the history file and sends it to the client. Finally, the function closes the history file.

3
CLIENT.C

Client side of the system. Uses sockets to connect to a chat server and communicate with it. Here's a breakdown of what the code does:

The ‘main‘ function starts by parsing the user ID from the command-line arguments. It then creates a socket and connects it to the server. If the connection fails, the function returns immediately.

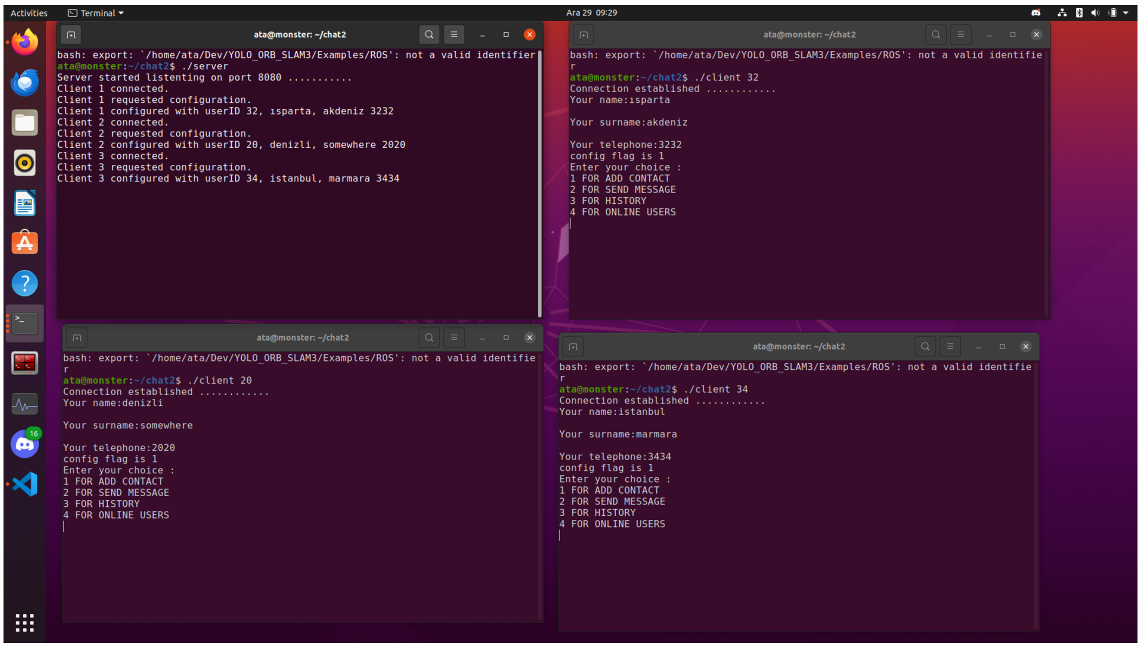
Once the connection is established, the function creates a new thread that runs the ‘doRecieving‘ function. This function continuously receives data from the server and prints it to the console. This allows the client to receive messages from the server at any time.

Back in the ‘main‘ function, if the user ID is not zero and the client has not been configured yet, the function sends a "CONFIG" command to the server, followed by the user ID and the user's details. The user's details are read from the console using ‘scanf‘.

The function then enters a loop where it continuously reads a command from the console and sends it to the server. The commands can be "ADD" to add a contact, "SEND" to send a message, "HISTORY" to view the message history with a contact, or "LIST" to view the online users. The details of the command, such as the contact's user ID or the message, are read from the console and sent to the server.

Finally, when the loop ends, the function cancels the receiving thread and returns. Note that the loop currently has no exit condition, so the only way to exit the program is to manually terminate it.

4 Outputs



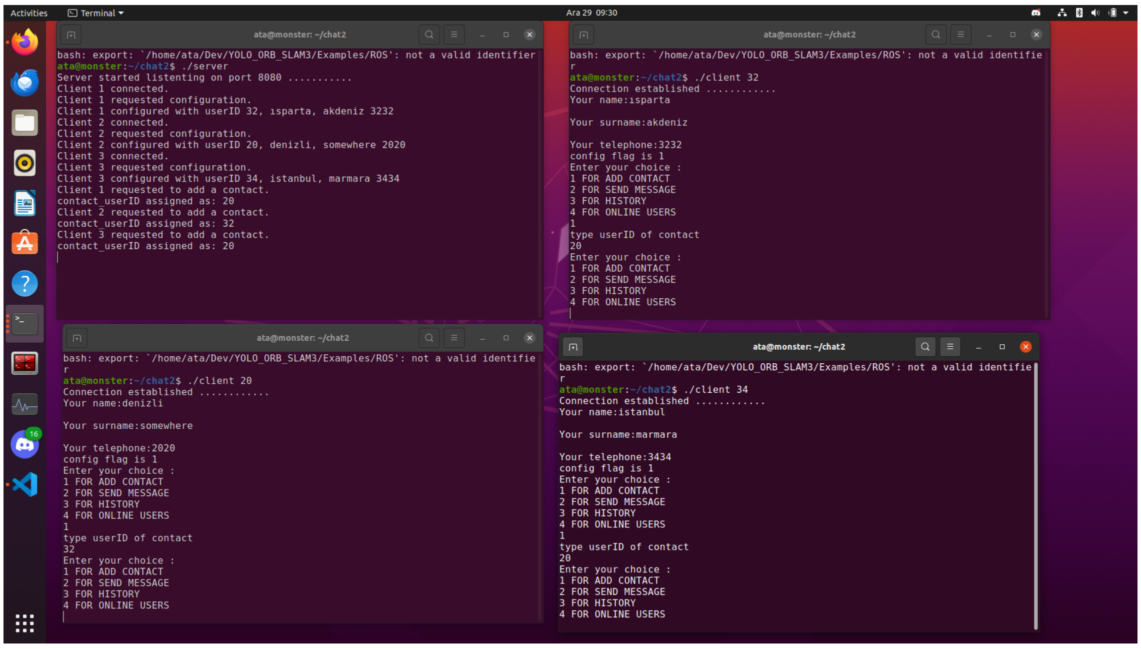
```
bash: export: `/home/ata/Dev/YOLO_ORB_SLAM3/Examples/ROS': not a valid identifier
ata@monster:~/chat2$ ./server
Server started listening on port 8080 ...........
Client 1 connected.
Client 1 requested configuration.
Client 1 configured with userID 32, ısparta, akdeniz 3232
Client 2 connected.
Client 2 requested configuration.
Client 2 configured with userID 20, denizli, somewhere 2020
Client 3 connected.
Client 3 requested configuration.
Client 3 configured with userID 34, istanbul, marmara 3434
```

```
bash: export: `/home/ata/Dev/YOLO_ORB_SLAM3/Examples/ROS': not a valid identifie
r
ata@monster:~/chat2$ ./client 32
Connection established ...........
Your name:ısparta

Your surname:akdeniz

Your telephone:3232
config flag is 1
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
```

```
bash: export: `/home/ata/Dev/YOLO_ORB_SLAM3/Examples/ROS': not a valid identifie
r
ata@monster:~/chat2$ ./client 20
Connection established ...........
Your name:denizli

Your surname:somewhere

Your telephone:2020
config flag is 1
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
```

```
bash: export: `/home/ata/Dev/YOLO_ORB_SLAM3/Examples/ROS': not a valid identifie
r
ata@monster:~/chat2$ ./client 34
Connection established ...........
Your name:istanbul

Your surname:marmara

Your telephone:3434
config flag is 1
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
```

Figure 4.1 Start the System



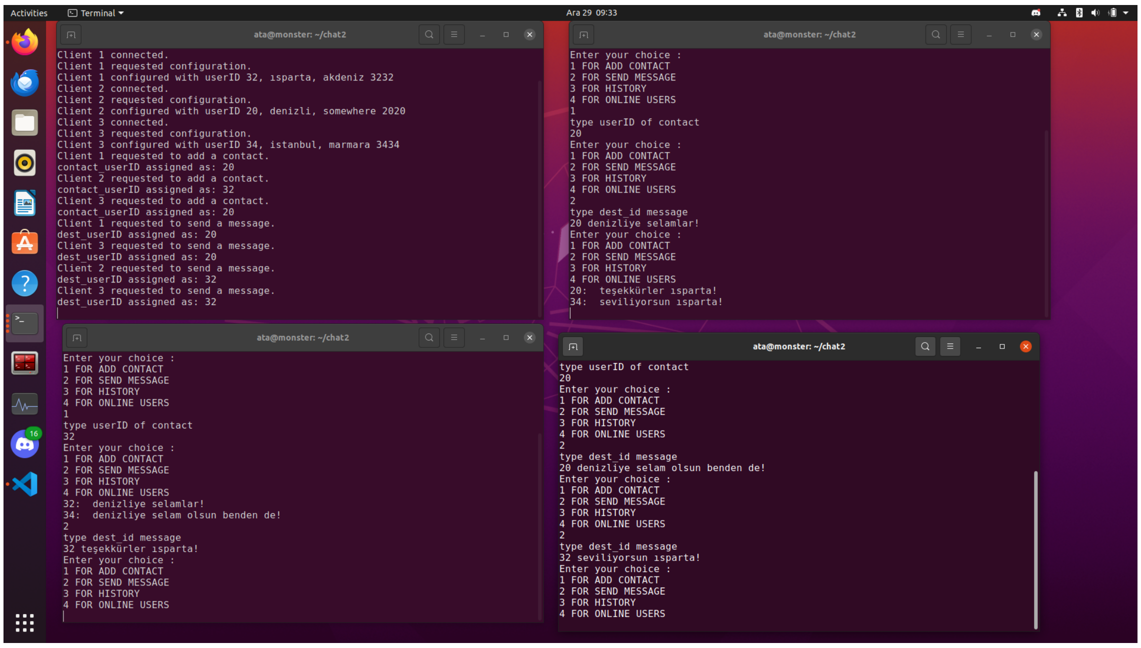
```
bash: export: `/home/ata/Dev/YOLO_ORB_SLAM3/Examples/ROS': not a valid identifier
ata@monster:~/chat2$ ./server
Server started listening on port 8080 ...........
Client 1 connected.
Client 1 requested configuration.
Client 1 configured with userID 32, ısparta, akdeniz 3232
Client 2 connected.
Client 2 requested configuration.
Client 2 configured with userID 20, denizli, somewhere 2020
Client 3 connected.
Client 3 requested configuration.
Client 3 configured with userID 34, istanbul, marmara 3434
Client 1 requested to add a contact.
contact_userID assigned as: 20
Client 2 requested to add a contact.
contact_userID assigned as: 32
Client 3 requested to add a contact.
contact_userID assigned as: 20
```

```
bash: export: `/home/ata/Dev/YOLO_ORB_SLAM3/Examples/ROS': not a valid identifie
r
ata@monster:~/chat2$ ./client 32
Connection established ...........
Your name:ısparta

Your surname:akdeniz

Your telephone:3232
config flag is 1
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
1
type userID of contact
20
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
```

```
bash: export: `/home/ata/Dev/YOLO_ORB_SLAM3/Examples/ROS': not a valid identifie
r
ata@monster:~/chat2$ ./client 20
Connection established ...........
Your name:denizli

Your surname:somewhere

Your telephone:2020
config flag is 1
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
1
type userID of contact
32
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
```

```
bash: export: `/home/ata/Dev/YOLO_ORB_SLAM3/Examples/ROS': not a valid identifie
r
ata@monster:~/chat2$ ./client 34
Connection established ...........
Your name:istanbul

Your surname:marmara

Your telephone:3434
config flag is 1
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
1
type userID of contact
20
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
```

Figure 4.2 Add Contact



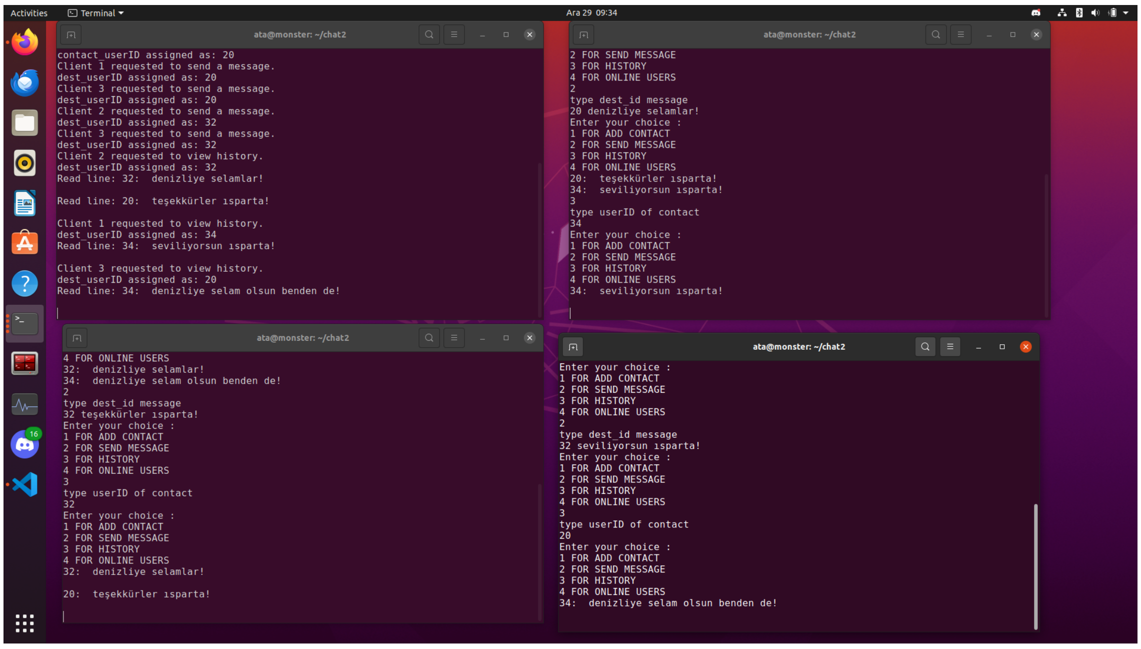
```
Client 1 connected.
Client 1 requested configuration.
Client 1 configured with userID 32, ısparta, akdeniz 3232
Client 2 connected.
Client 2 requested configuration.
Client 2 configured with userID 20, denizli, somewhere 2020
Client 3 connected.
Client 3 requested configuration.
Client 3 configured with userID 34, istanbul, marmara 3434
Client 1 requested to add a contact.
contact_userID assigned as: 20
Client 2 requested to add a contact.
contact_userID assigned as: 32
Client 3 requested to add a contact.
contact_userID assigned as: 20
Client 1 requested to send a message.
dest_userID assigned as: 20
Client 3 requested to send a message.
dest_userID assigned as: 20
Client 2 requested to send a message.
dest_userID assigned as: 32
Client 3 requested to send a message.
dest_userID assigned as: 32
```

```
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
1
type userID of contact
20
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
2
type dest_id message
20 denizliye selamlar!
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
20:  teşekkürler ısparta!
34:  seviliyorsun ısparta!
```

```
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
1
type userID of contact
32
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
32:  denizliye selamlar!
34:  denizliye selam olsun benden de!
2
type dest_id message
32 teşekkürler ısparta!
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
```

```
type userID of contact
20
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
2
type dest_id message
20 denizliye selam olsun benden de!
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
2
type dest_id message
32 seviliyorsun ısparta!
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
```

Figure 4.3 Send Messages Each Other



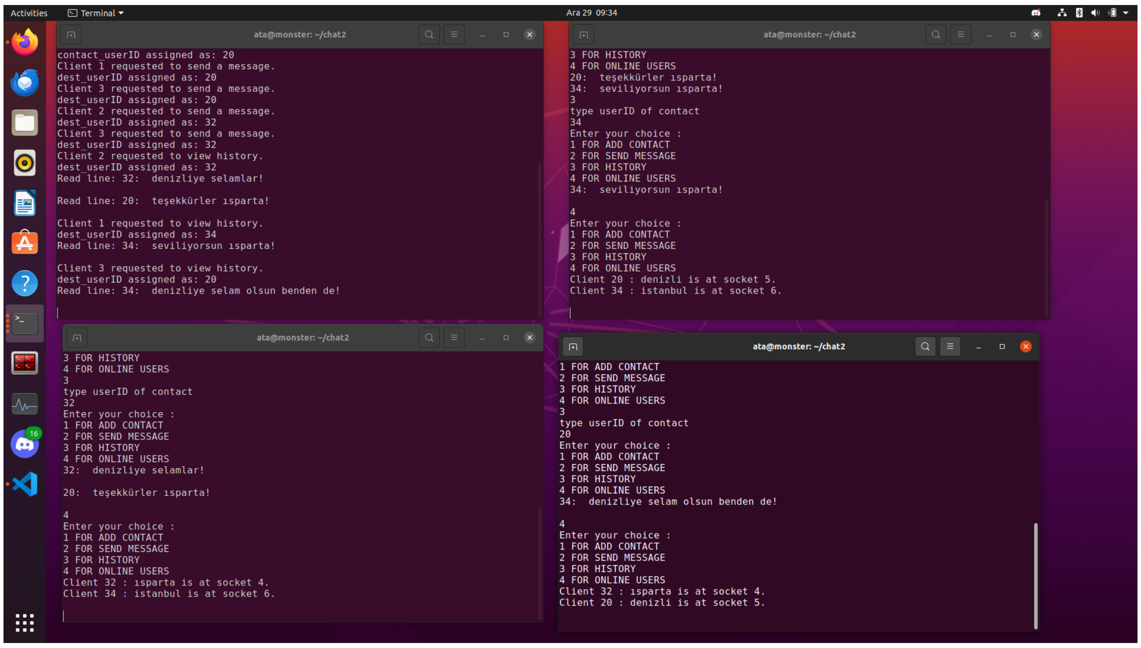
```
contact_userID assigned as: 20
Client 1 requested to send a message.
dest_userID assigned as: 20
Client 3 requested to send a message.
dest_userID assigned as: 20
Client 2 requested to send a message.
dest_userID assigned as: 32
Client 3 requested to send a message.
dest_userID assigned as: 32
Client 2 requested to view history.
dest_userID assigned as: 32
Read line: 32:  denizliye selamlar!

Read line: 20:  teşekkürler ısparta!

Client 1 requested to view history.
dest_userID assigned as: 34
Read line: 34:  seviliyorsun ısparta!

Client 3 requested to view history.
dest_userID assigned as: 20
Read line: 34:  denizliye selam olsun benden de!
```

```
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
2
type dest_id message
20 denizliye selamlar!
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
20:  teşekkürler ısparta!
34:  seviliyorsun ısparta!
3
type userID of contact
34
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
34:  seviliyorsun ısparta!
```

```
4 FOR ONLINE USERS
32:  denizliye selamlar!
34:  denizliye selam olsun benden de!
2
type dest_id message
32 teşekkürler ısparta!
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
3
type userID of contact
32
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
32:  denizliye selamlar!

20:  teşekkürler ısparta!
```

```
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
2
type dest_id message
32 seviliyorsun ısparta!
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
3
type userID of contact
20
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
34:  denizliye selam olsun benden de!
```

Figure 4.4 Show History to Specific User



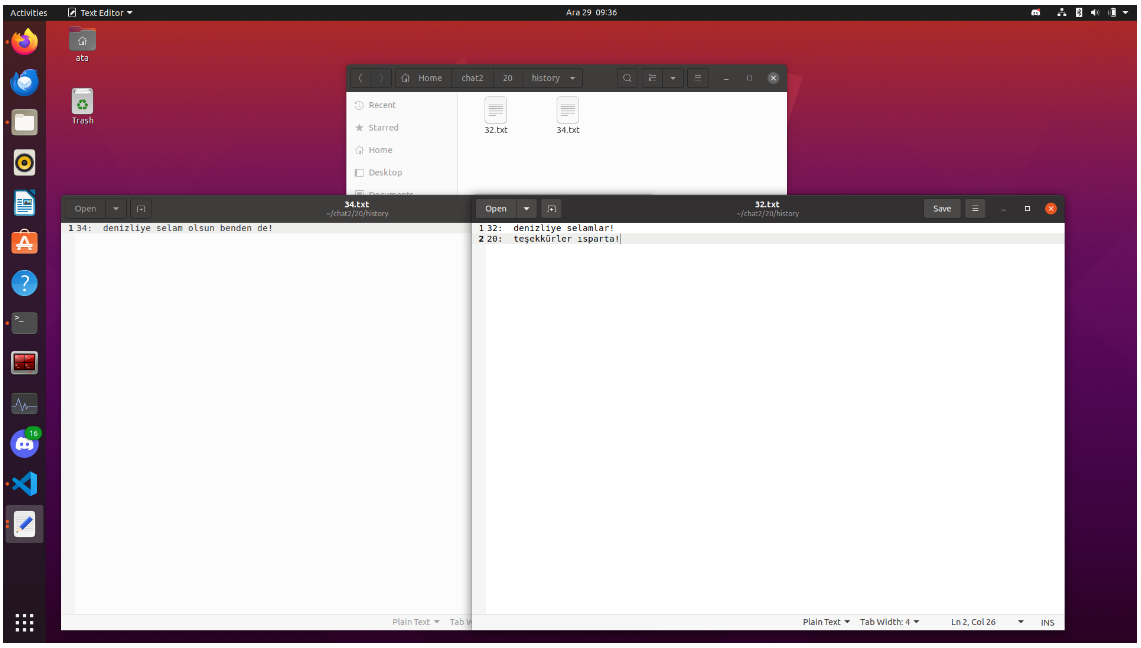
```
contact_userID assigned as: 20
Client 1 requested to send a message.
dest_userID assigned as: 20
Client 3 requested to send a message.
dest_userID assigned as: 20
Client 2 requested to send a message.
dest_userID assigned as: 32
Client 3 requested to send a message.
dest_userID assigned as: 32
Client 2 requested to view history.
dest_userID assigned as: 32
Read line: 32:  denizliye selamlar!

Read line: 20:  teşekkürler ısparta!

Client 1 requested to view history.
dest_userID assigned as: 34
Read line: 34:  seviliyorsun ısparta!

Client 3 requested to view history.
dest_userID assigned as: 20
Read line: 34:  denizliye selam olsun benden de!
```

```
3 FOR HISTORY
4 FOR ONLINE USERS
20:  teşekkürler ısparta!
34:  seviliyorsun ısparta!
3
type userID of contact
34
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
34:  seviliyorsun ısparta!

4
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
Client 20 : denizli is at socket 5.
Client 34 : istanbul is at socket 6.
```

```
3 FOR HISTORY
4 FOR ONLINE USERS
3
type userID of contact
32
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
32:  denizliye selamlar!

20:  teşekkürler ısparta!

4
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
Client 32 : ısparta is at socket 4.
Client 34 : istanbul is at socket 6.
```

```
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
3
type userID of contact
20
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
34:  denizliye selam olsun benden de!

4
Enter your choice :
1 FOR ADD CONTACT
2 FOR SEND MESSAGE
3 FOR HISTORY
4 FOR ONLINE USERS
Client 32 : ısparta is at socket 4.
Client 20 : denizli is at socket 5.
```

Figure 4.5 Show History



```
34:  denizliye selam olsun benden de!
```

```
32:  denizliye selamlar!
20:  teşekkürler ısparta!
```

Figure 4.6 Txt Files That Are Created