YILDIZ TECHNICAL UNIVERSITY

BLM3011

Operating System Project

Abdulkadir Türe 20011042

abdulkadir.ture@std.yildiz.edu.tr

Contents

About the Project

The project at hand is a messaging application built on the foundation of socket programming, employing a server-client architecture. Leveraging the principles of socket programming, this application is designed to facilitate multi-user communication through the utilization of a server and multiple client instances, all implemented in the C programming language.

The primary objective of the project is to realize a robust and versatile multi-messaging application, fostering seamless communication between a central server and numerous clients. The intricacies of socket programming serve as the backbone, enabling efficient data exchange and real-time communication among connected clients. By harnessing the power of C, the project aims to create a reliable and efficient platform for users to engage in concurrent conversations through the server-client model.

The overarching goal is to provide a practical and functional solution for multi-user communication, with an emphasis on scalability and responsiveness. Through the implementation of server-client architecture and socket programming principles, the project endeavors to showcase the capabilities of C in building a sophisticated messaging application capable of handling concurrent interactions among users.

Project Requirements

This project has been tested on the Ubuntu 22.04 operating system. Due to the variation in operating system libraries required for socket programming, it is important to note that this project may not be compatible with different operating systems such as Windows.

The implementation relies on specific operating system libraries for socket programming, and these libraries may differ between operating systems. As a result, the project has been tailored and tested specifically for the Ubuntu 22.04 environment. It is recommended to ensure compatibility with the specified operating system to guarantee the seamless execution of the server-client messaging application.

Project Functions

User Registration

The User Registration function facilitates the process of user registration within the messaging application. When a user expresses the intention to register, the system prompts them to provide essential information, including their phone number (formatted as +905555555555), first name, last name, and password. These details are then concatenated into a string, which is transmitted to the server accompanied by the /register tag.

Upon receiving the string, the server dissects the information based on the specified tag and invokes the corresponding function to handle user registration. The server allocates a unique User ID as part of the registration process. This ID is automatically generated by the system and serves as a reference for server-side operations. Notably, the User ID may be utilized when necessary for various server-related tasks.

To enhance user-friendliness, the system allows users to log in using their phone numbers and passwords. This approach eliminates the need for users to memorize User IDs, contributing to a more seamless and user-centric experience.

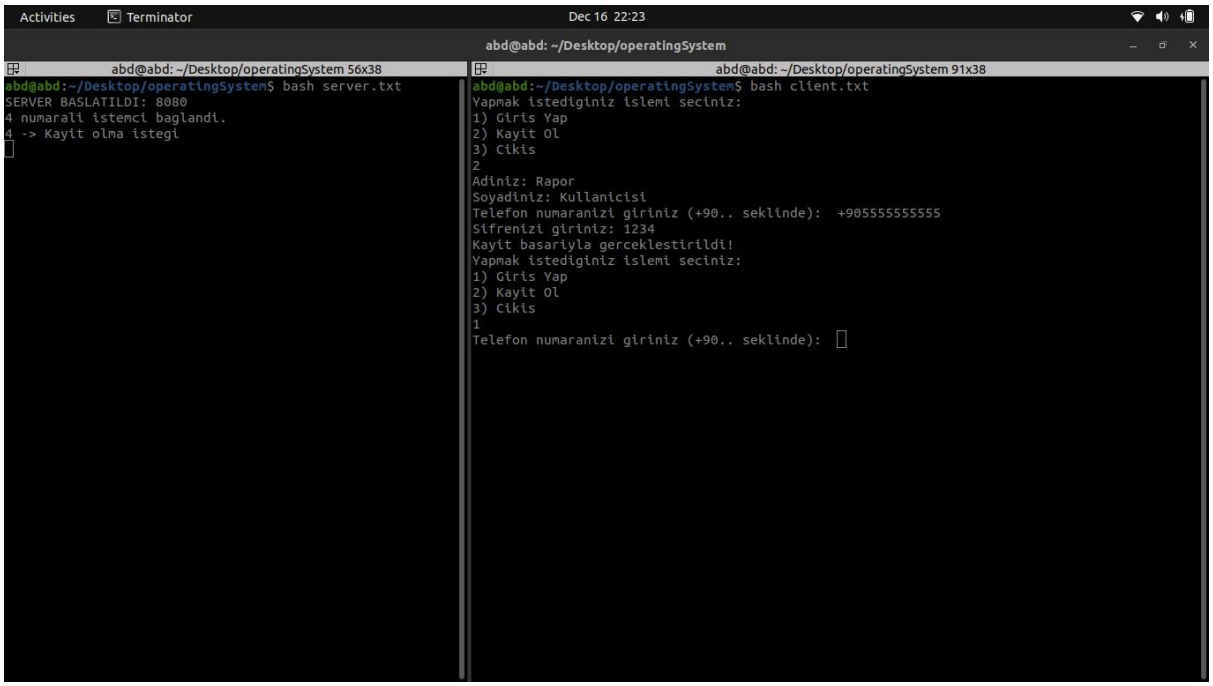

Image 1 Right: Server terminal , Left: Client Terminal

User Login

The User Login function enables users to access the system by providing their phone number and password. To initiate the login process, the user transmits the relevant information to the server, accompanied by the /login tag. The server then queries the user database, specifically the users.csv file, to determine whether the provided credentials match an existing user entry.

The server conducts a thorough examination of the users.csv file, where each line represents an individual user along with their associated information. Based on the outcome of the query, the server responds to the client with either a positive or negative confirmation. A positive response signifies a successful login, while a negative response indicates that the provided credentials do not correspond to any registered user.

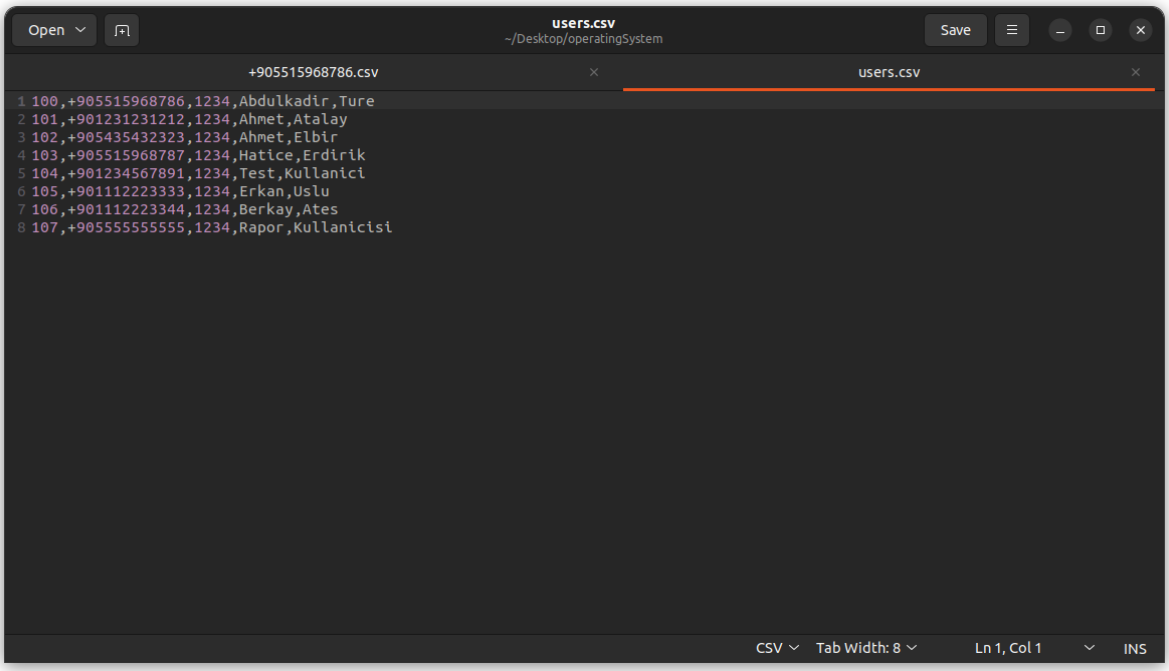

Image 2 users.csv

Seamless Communication and Multithreading

Upon successful login, the user establishes a communication link with the server through two separate socket connections. The first socket connection is dedicated to transmitting user requests to the server and subsequently receiving and processing the server's responses. This bidirectional channel serves as the primary means for users to interact with the application, conveying their requests and interpreting the server's feedback.

Simultaneously, the second socket connection is reserved for handling notifications destined for the user. This connection ensures that incoming notifications are promptly displayed on the user's screen. By utilizing two distinct socket connections and implementing multithreading, the messaging application achieves a clear separation between user-related functions and notification functionalities.

The implementation of two separate threads enhances the program's efficiency and organization. The first thread manages user requests and responses, ensuring a smooth and responsive user experience. Meanwhile, the second thread is responsible for handling notifications, delivering real-time updates to the user without causing interruptions or delays in the main user interface.

The utilization of multithreading, along with the segregation of functionalities into distinct socket connections, contributes to the overall coherence and flawless operation of the program. This design choice minimizes potential complexities, allowing for a seamlessly functioning messaging application where user interactions and notifications coexist harmoniously. The threads operate independently, preventing any interference or conflicts, thereby ensuring the program's robustness and reliability.

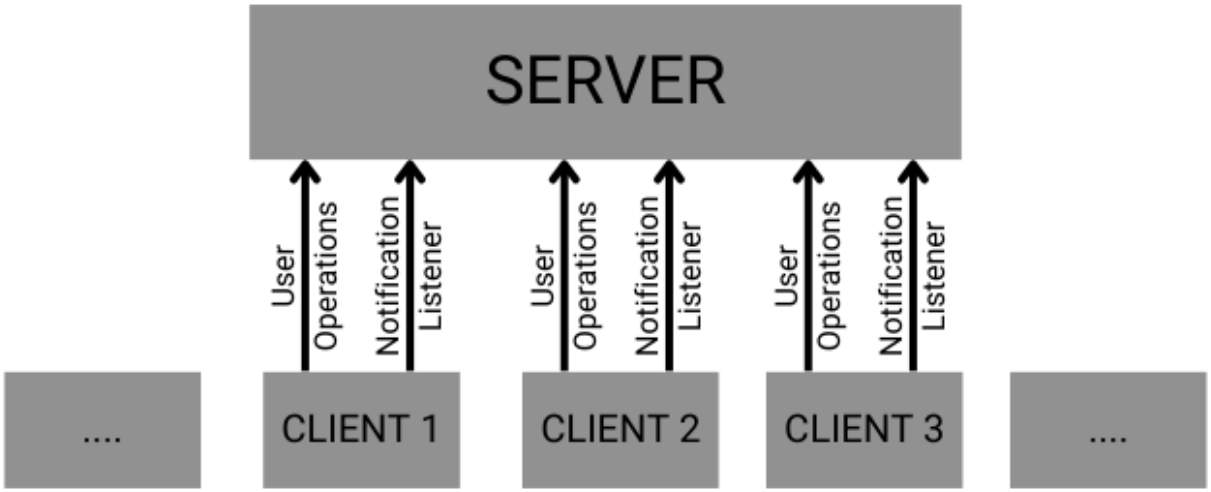

Image 3 Communication Structure

Message Sending

The messaging application empowers users to send messages to a designated phone number through the user-friendly message sending option. On the client side, the user is prompted to input the recipient's phone number and the desired message. Subsequently, this information is transmitted to the server using the /sendMessage tag.

Upon receiving the message details, the server dynamically generates a conversation file (a .csv file) based on the sender and recipient phone numbers. All conversation files are systematically organized within the "messages" folder. Each conversation file contains essential details such as the sender's phone number, recipient's phone number, message send date, message ID, message read status (indicated by + or -), and the content of the message.

Simultaneously, if the recipient is online at the time of message submission, a notification is promptly dispatched to their terminal screen. This immediate notification feature ensures that users are promptly informed of incoming messages, fostering real-time communication.

The organization of conversation files within the "messages" folder facilitates easy retrieval and management of messages. Each file encapsulates a comprehensive record of the conversation, allowing users to review message history and related details effortlessly.

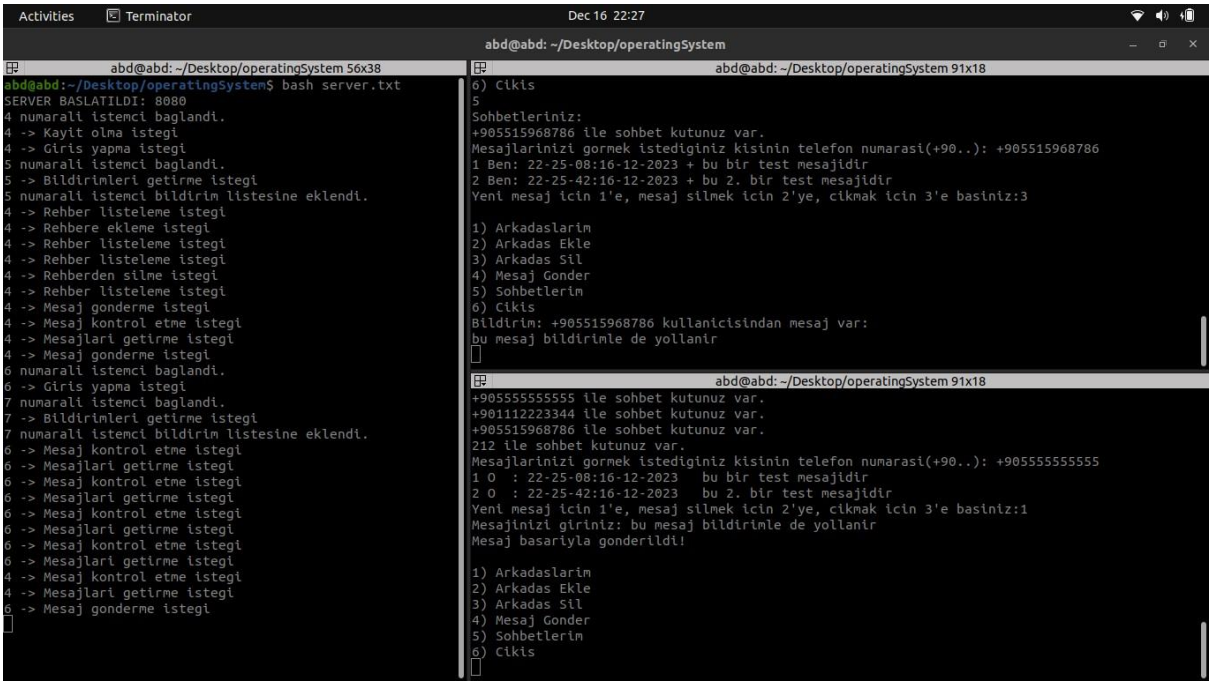

Image 4 Server, Client 1 , Client 2 , Example 2

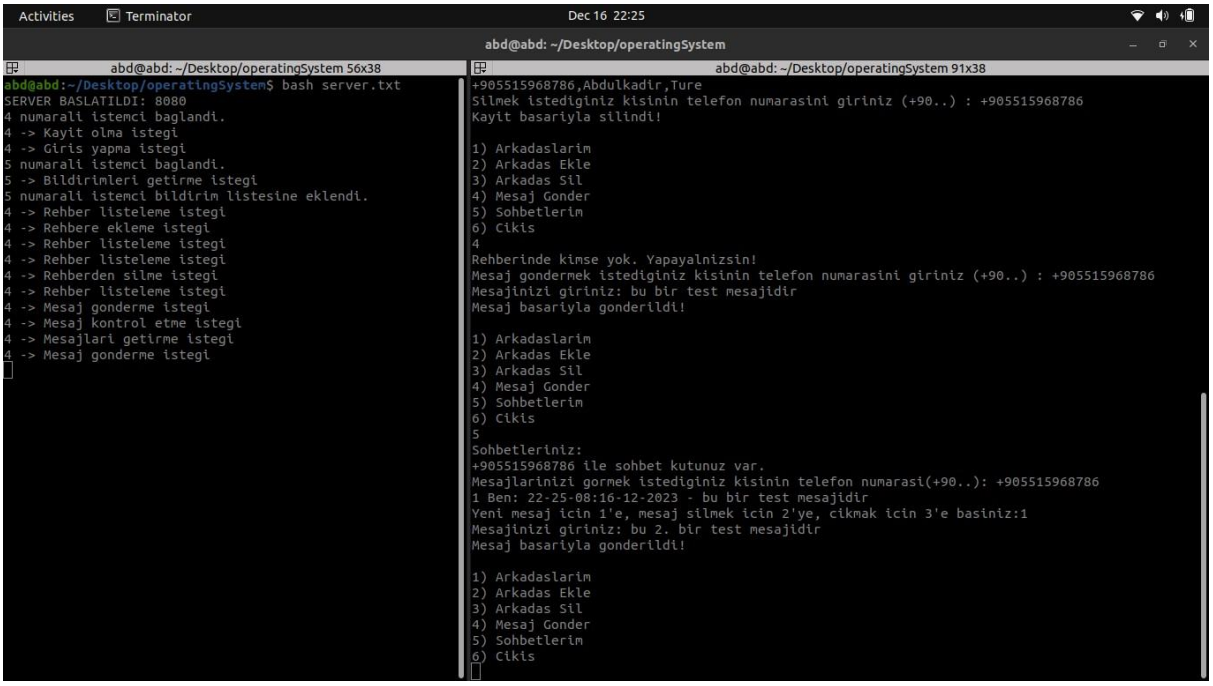

Image 5 Server, Client 1 Example 3

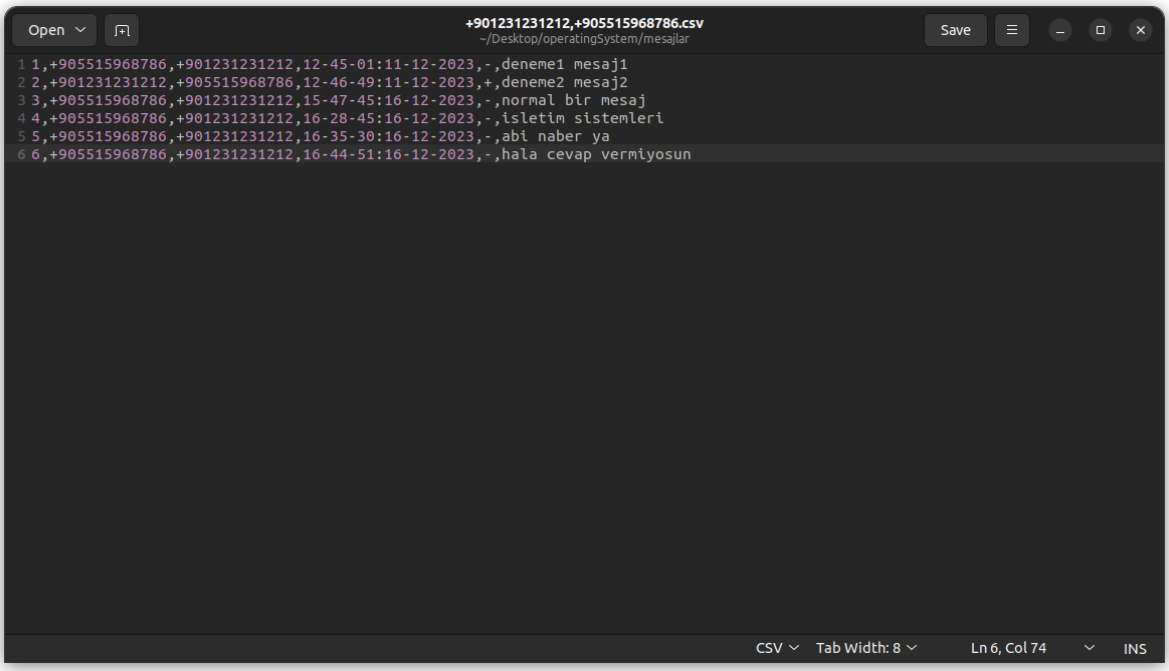

Image 6 Database Message Recording Example

Message Deletion and Read Status

Within an ongoing conversation, users have the flexibility to delete specific messages based on their unique message ID numbers. It is crucial to note a key constraint: users can only delete messages that they have personally sent. Deleting messages from the counterpart user's side is restricted for security and privacy reasons.

To initiate message deletion, the user specifies the ID number of the message they wish to remove. The system verifies the ownership of the message and, if authorized, proceeds to remove it from the conversation. This feature empowers users to manage their message history according to their preferences, contributing to a personalized and organized messaging experience.

Moreover, the messaging application prioritizes user-friendly interaction by incorporating read status indicators. Each message is accompanied by a + or - sign, providing a visual cue to users regarding the read status of their messages. This feature allows users to discern whether their sent messages have been read by the recipient.

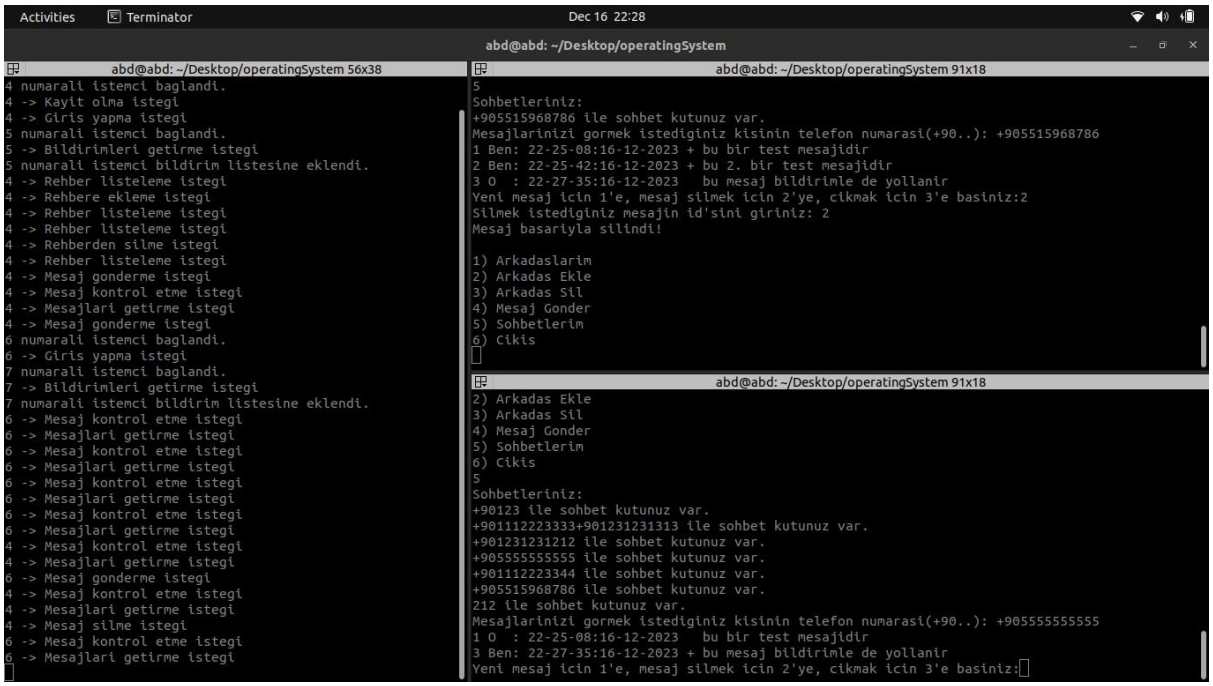

Image 7 Read Status Example

Address Book Operations

Every user possesses a personalized address book, providing a dedicated space to manage contacts efficiently. Within this address book, users can add, view, and delete contacts, each identified by their phone number, name, and surname. The address book for each user is stored in a .csv file under the "rehber" directory.

Viewing Address Book:

Users have the option to view their entire address book, gaining a comprehensive overview of their stored contacts. This feature allows for quick reference and easy access to contact information.

Adding Contacts:

Users can enrich their address book by adding new contacts. Through a user-friendly interface, individuals can input the phone number, name, and surname

of the contact they wish to add. The system then appends this information to the user's address book file, facilitating efficient contact management.

Deleting Contacts:

For enhanced customization, users have the ability to remove contacts from their address book. By specifying the phone number of the contact to be deleted, users can streamline their address book according to their preferences.

The segregated storage of each user's address book in individual .csv files ensures data integrity and accessibility. The "rehber" directory serves as a centralized location for managing contact information.

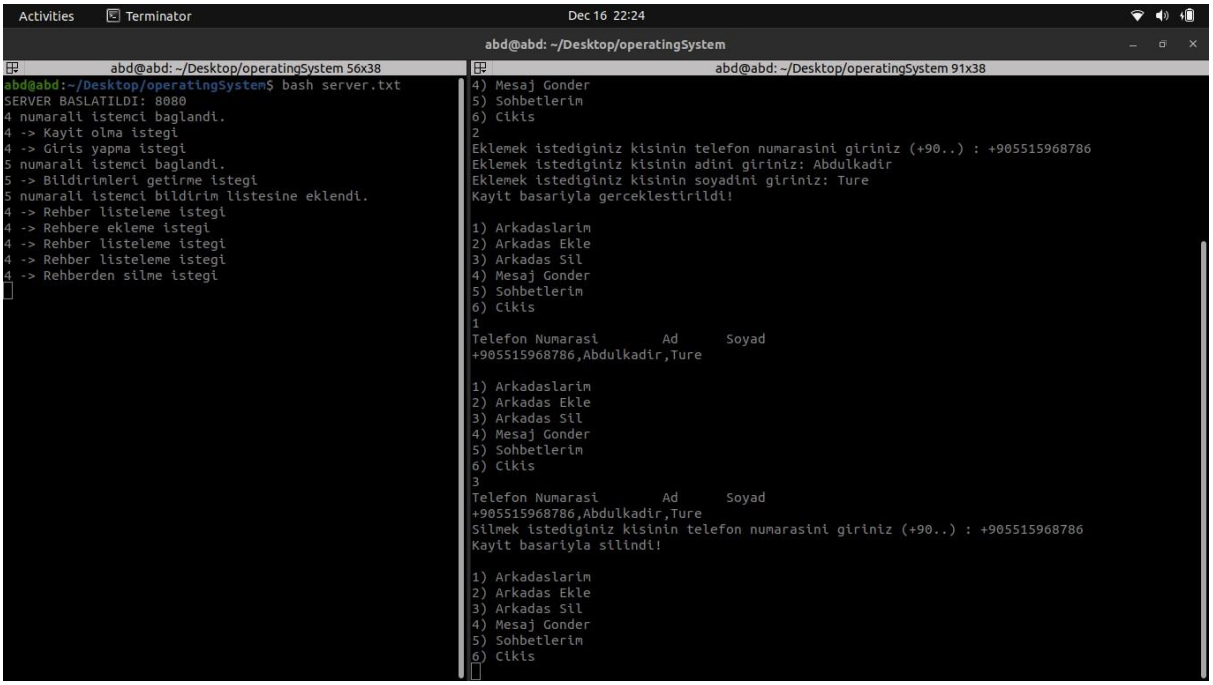

Image 8 Address Book Operations Example

Image 9 User Guide Record Example

My Chats

In the "My Chats" section, users can access a comprehensive overview of all their past conversations. This area serves as a centralized hub where all previously engaged contacts are listed. To delve into the details of a particular conversation, users are prompted to provide the phone number associated with the contact. Once entered, the system displays a chronological log of all messages exchanged within that specific conversation.

The user experience is designed for simplicity and efficiency, allowing users to effortlessly navigate through their chat history. The chronological arrangement of messages ensures that conversations are presented in a structured manner, facilitating easy reference and retrieval of information.

Here's a step-by-step breakdown of the process:

Accessing My Chats:

Users navigate to the "My Chats" section to view a consolidated list of all contacts they have interacted with.

Entering Phone Number:

To access the details of a specific conversation, users input the phone number associated with the contact they wish to review.

Displaying Conversation:

Once the phone number is provided, the system retrieves and displays the entire conversation, presenting messages in chronological order. This chronological organization allows users to trace the progression of their interactions over time.

By providing a centralized and user-friendly interface for reviewing past conversations, the "My Chats" section enhances the overall messaging experience, offering users a convenient way to revisit and reflect on their communication history.

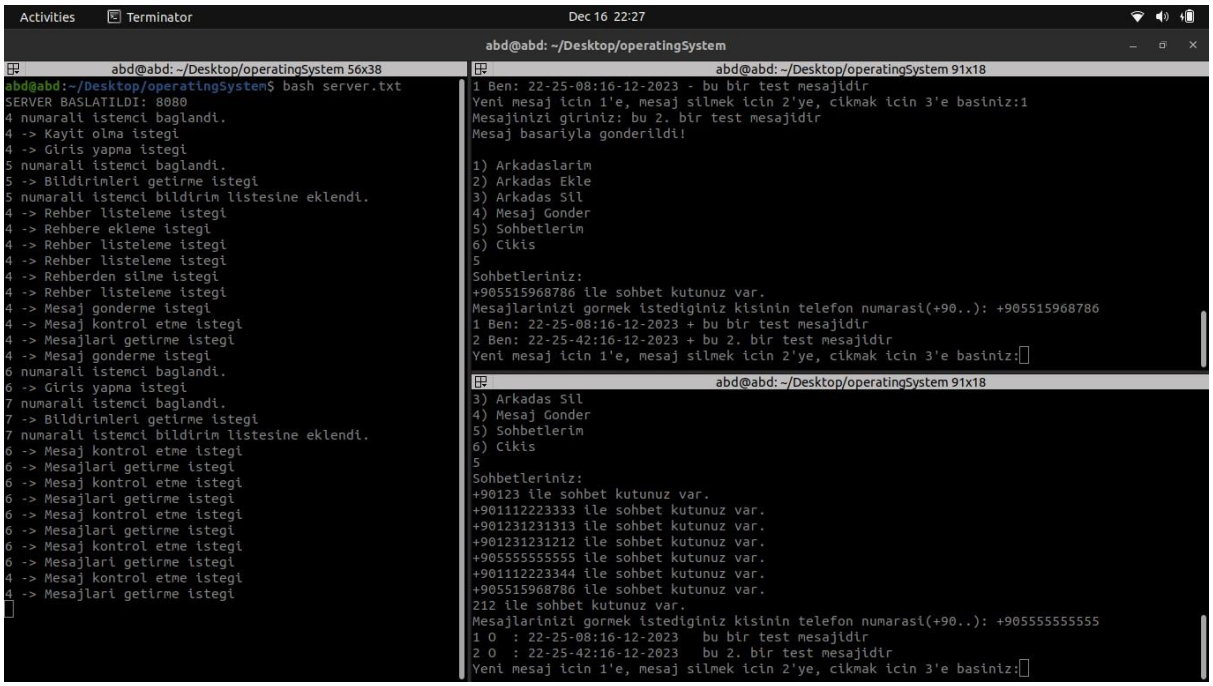

Image 10 My Chats Example

References

https://www.geeksforgeeks.org/socket-programming-cc/

https://www.scaler.com/topics/socket-programming-in-c/

https://www.linuxhowtos.org/C_C++/socket.htm

https://www.binarytides.com/socket-programming-c-linux-tutorial/