**Computer Network Project Submission**

**A Client-Server Application**

**Submitted By:**

**Name: Harsh Trada, Roll No: U22CS116**

**Name: Hardik Hemnani, Roll No: U22CS081**

**Name: Aasutosh Baraiya, Roll No: U22CS109**

**Name: Anil Prajapati, Roll No: U22CS136**

**Submitted To: Dr. Keyur Parmar**

**Department of Computer Science and Engineering**

SARDAR VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY SURAT

2024

**Application Details:**

The client-server program establishes a communication channel between a client application and a server application. In this specific scenario, the client-server program facilitates the retrieval of descriptions for predefined keywords. The server hosts a dictionary containing keyword-description pairs, and the client can request a description by sending a keyword to the server. The server responds with the corresponding description, if available.

**Client-Side Details:**

**Description:**

This Python program utilizes the Tkinter library to create a simple graphical user interface (GUI) for a client-side application. The application is designed to interact with a server running on the localhost (127.0.0.1) at port 3000. The user interface prompts the user to enter a keyword, which is then sent to the server. Upon receiving a response from the server, the description associated with the keyword is displayed in a text box within the GUI.

**Pseudocode:**

1. Import necessary libraries: tkinter for GUI and socket for communication in python.

2. Define a function to get the keyword name get\_keyword\_desc ():

a. Retrieve the keyword entered by the user from the entry widget.

b. Create a TCP socket for communication with the server.

c. Define the server address as localhost and port 3000.

d. Connect to the server by printing the initial message.

e. Send the keyword message to the server.

f. Receive the response data from the server.

g. Close the socket connection.

h. Enable the text description widget for editing.

i. Clear any existing text in the description widget.

j. Insert the received data into the description widget.

k. Disable the text description widget to prevent further editing.

3. Create a Tkinter window for GUI:

a. Set the window title.

b. Configure padding for the window.

c. Set frame dimensions and styling.

d. Pack the frame into the window.

4. Create an entry widget for the user to input the keyword.

5. Create a button widget to submit the keyword.

a. Set the button's appearance and behavior.

b. Link the button to the function named get\_keyword\_desc ().

6. Create a text widget to display the description.

a. Set dimensions and disable editing.

7. Pack the label, entry, button, and text widgets into the frame.

8. Run the Tkinter event loop to display the GUI and handle user interactions.

**Server-Side Details:**

**Description:**

The server-side code sets up a TCP socket and listens for incoming connections on localhost at port 3000. When a client connects, it receives the keyword sent by the client. If the keyword exists in the dictionary, the server retrieves the corresponding description and sends it back to the client. If the keyword is not found, the server sends a message indicating that the description was not found. The server continues listening for connections and processing requests until terminated.

**Pseudocode:**

1. Import the socket module.

2. Define a dictionary containing keyword-description pairs.

3. Create a TCP socket for the server.

4. Define the server address as localhost and port 3000.

5. Bind the socket to the server address.

6. Listen for incoming connections with a maximum backlog of 10 connections.

7. Enter a loop to handle client connections:

a. Accept a client connection.

b. Retrieve data sent by the client.

c. Decode the received data to obtain the keyword.

d. Check if the keyword exists in the dictionary:

* If found, retrieve the corresponding description.
* Send the description back to the client.

e. If the keyword is not found, send a message indicating that the description was not found.

f. Close the connection with the client.

8. Terminate the server when done.

**Applications:**

1. **Online Dictionary Service**:
   1. Provide a comprehensive collection of terms and definitions.
   2. Allow clients to request definitions for various terms to enhance vocabulary and understanding.
2. **Educational Quiz Game**:
   1. Store questions and answers within the server's dictionary.
   2. Clients can request questions and receive corresponding answers, turning the application into an interactive quiz game.
3. **Collaborative Learning Platform**:
   1. Create a platform for users to share educational resources and explanations.
   2. Users submit keywords, and the server retrieves relevant explanations contributed by other users or experts.
4. **Technical Support System**:
   1. Establish a system for providing technical support on software or hardware-related issues.
   2. Clients submit keywords related to their problems, and the server provides troubleshooting steps or solutions.
5. **Remote Assistance Tool**:
   1. Implement a system where clients can request assistance on specific topics or issues.
   2. The server provides detailed instructions or solutions based on the keywords provided by clients.

**Conclusion:**

In summary, the client-server application project has been instrumental in deepening our understanding of network communication, socket programming, and GUI development. By implementing a system for retrieving keyword descriptions, we have honed our skills in designing and deploying networked applications. This project lays the foundation for exploring more complex applications and real-world solutions in the future, showcasing the versatility and practicality of client-server architectures in various domains.