metin, yazı tipi, logo, grafik içeren bir resim

Açıklama otomatik olarak oluşturuldu

BAHCESEHIR UNIVERSITY

FACULTY OF ENGINEERING AND NATURAL SCIENCE

COMPUTER ENGINEERING

INTRODUCTION TO COMPUTER NETWORKS

TERM PROJECT

Halil Furkan Ozan – 2102105

Muhammed Can Harmandar – 2101895

Emre Mutlu - 2104085

**Objective:**

The design and implement a networked peer-to-peer chat application. The shared design document (Functional Specification) specifies the necessary protocols need to implement. In our application we implement the protocols exactly as specified.

**Requirements:**

1. Have 4 processes: Peer Discovery, Service Announcer, Chat Responder, Chat Initiator. These processes should work as outlined in their respective specifications.
2. Successfully detect all available users in the Local Area Network.
3. Interactive UI, display when a user is detected, correctly display users’ online/away presence, allow the end user to choose to display users/exchange numbers to generate secret keys/send encrypted message/send unencrypted message.
4. Successfully chat with any available user in the Local Area Network.
5. Successfully encrypt/decrypt each peer-to-peer conversation.
6. Output a chat log, containing timestamps and content of all messages exchanged in a chat session.

**Libraries:**

**ekran görüntüsü, metin, siyah içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Main Menu**  
**Welcome Screen:**

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

The ‘welcome’ function is designed to display an ASCII art message stored in the ascii\_welcome string. It indents each line differently based on the line number to create a visually appealing display. The lines are displayed with a slight delay between each one to enhance the visual effect.

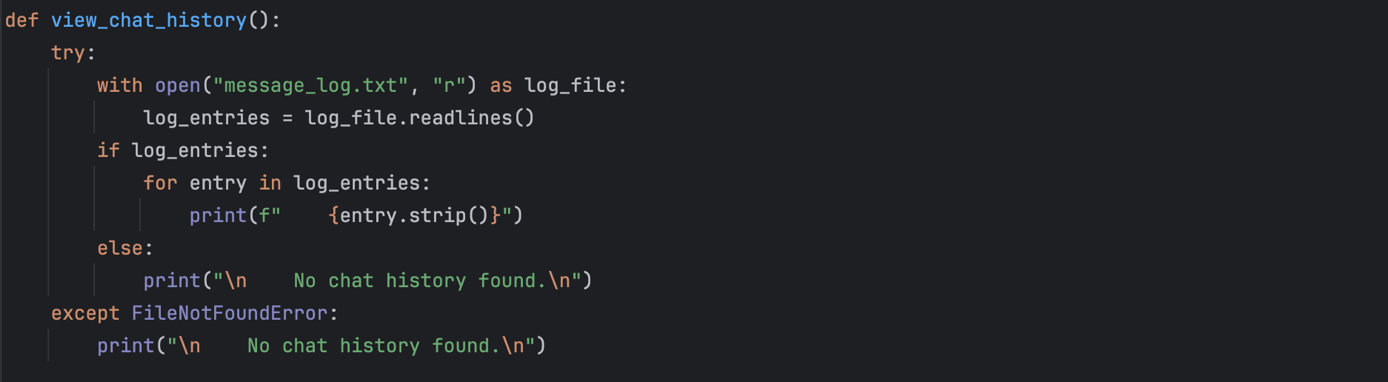
metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**display\_menu:** This function is responsible for showing a menu of options to the user in a formatted manner.

**clear\_cli:** This function clears the command-line screen based on the user's operating system.

**get\_username:** This function prompts the user to input their username and returns the entered value.



The view\_chat\_history function is designed to read and display the chat history from a file named message\_log.txt. Here's a breakdown of what each part of the function does:

* Open the file **message\_log.txt** in read mode ("r"). Using the with statement ensures that the file is properly closed after its contents are read, even if an error occurs.
* This line reads all the lines from the file and stores them in a list called log\_entries.
* **If log\_entries**: line checks if the **log\_entries** list is not empty. If it contains entries, the function iterates over each entry and prints it to the console. The entry.strip() method is used to remove any leading and trailing whitespace, including newline characters.
* If the **log\_entries** list is empty (meaning the file is empty), the function prints a message indicating that no chat history was found.

**Service Announcer**

**metin, yazılım, multimedya yazılımı, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu**

* **client\_id:** The username of the client that will be broadcasted.
* A dictionary data is created with a single key-value pair where the key is 'username' and the value is the client\_id.
* The data dictionary is converted to a JSON string using json.dumps.
* The JSON string is encoded to a byte array (client\_data) using UTF-8 encoding. This is necessary for sending data over a network.
* **broadcast\_address:** The broadcast IP address for the local network.
* **destination\_port:** The port number (6000) to which the data will be sent.
* UDP socket is created using IPv4 addressing (AF\_INET) and the UDP protocol (SOCK\_DGRAM).
* The socket option **SO\_BROADCAST** is set to 1, enabling the socket to send broadcast messages.
* An infinite loop (while True) is used to repeatedly send the broadcast message.
* **sendto(client\_data, (broadcast\_address, destination\_port)):** The byte-encoded client data is sent to the specified broadcast address and port.
* **time.sleep(8):** The loop pauses for 8 seconds before sending the next broadcast message, ensuring periodic broadcasting.

**Peer Discovery**

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

The peer\_discovery function listens for broadcast messages on the local network to discover other clients. It updates and maintains a record of discovered hosts along with their discovery timestamps.

* **discovered\_hosts:** A dictionary to store discovered hosts with their IP addresses as keys and usernames as values.
* **timestamp:** A dictionary to store the time (in a human-readable format) when each host was last discovered.
* **timestamp\_seconds:** A dictionary to store the time (in seconds since the epoch) when each host was last discovered.
* **local\_ip\_address:** Retrieves the local IP address of the machine running this function.
* **peer\_discovery\_socket:** Creates a UDP socket using IPv4 addressing (AF\_INET) and the UDP protocol (SOCK\_DGRAM).
* Binds the socket to the address '0.0.0.0' and port 6000. The address '0.0.0.0' allows the socket to listen on all available network interfaces.
* Enters an infinite loop to continuously listen for incoming UDP packets.
* Receives data from the socket with a buffer size of 1024 bytes. The source address and port of the sender are also captured.
* Decodes the received data from bytes to a UTF-8 string.
* Parses the JSON string into a dictionary.
* Extracts the username (server\_id) from the JSON data.
* If the username in the received message matches the current user's username (username), the message is ignored.
  + If the source address is not already in the discovered\_hosts dictionary:
    - Updates **timestamp\_seconds** with the current time (in seconds) for the discovered host.
    - Updates **timestamp** with the current time (in human-readable format) for the discovered host.
    - Adds the source address and username to the **discovered\_hosts** dictionary.
  + If the source address is already in discovered\_hosts:
    - Updates the timestamps (timestamp\_seconds and timestamp) with the current time for the discovered host.

**View Online Users**

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

The view\_online\_users function displays a list of users categorized by their online status. It determines the status based on the time elapsed since each user was last seen:

* Users not seen for more than 15 minutes are marked as "Offline".
* Users seen between 10 seconds and 15 minutes ago are marked as "Away".
* Users seen within the last 10 seconds are marked as "Online".

This function helps keep track of the presence and activity status of users on the network.

**Encryption and Decryption:**

**metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Chat Initiator:**

**metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu**

* **des\_key** and **des\_iv** are the 8-byte key and initialization vector required for DES encryption.
* k is the DES cipher object created using the pyDes library, configured for CBC (Cipher Block Chaining) mode with PKCS5 padding.
* **encrypt\_message** encrypts the plaintext message using the DES cipher.
* **decrypt\_message** decrypts the encrypted\_message back to plaintext.
* The **chat\_initiator** function starts by prompting the user to input the username (peer\_id) they want to chat with.
* It checks if the **peer\_id** is in **last\_seen** (a dictionary presumably containing last seen times of users).
* Establishes a secure connection using sockets.
* Encrypts the message before sending it over the network.
* Logs each sent message with a timestamp in a file message\_log.txt.

**metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu**

* Establishes an unsecure connection using sockets.
* Sends the message as a JSON-encoded string without encryption.
* Logs each sent message with a timestamp in a file message\_log.txt.

**Chat Responder:**

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Creates a TCP/IP socket.
* Binds the socket to all available interfaces (0.0.0.0) on port 6001.
* Listens for incoming connection requests.
* Enters an infinite loop to continuously accept incoming connections.
* When a connection is accepted, it captures the peer's socket, IP address, and port.
* Looks up the peer's ID using the hosts dictionary.
  + Attempts to receive up to 1024 bytes of data from the peer.
  + Tries to decrypt the received data using decrypt\_message.
    - If decryption is successful, it prints and logs the decrypted message.
  + If decryption fails (throws an exception), it assumes the data is a plain text message in JSON format.
    - Attempts to parse the JSON data.
    - If the data contains an unencrypted\_message key, it prints and logs this message.
  + If JSON decoding fails, it prints a message indicating non-JSON data was received.
    - Closes the peer's socket after processing the message.

**WireShark**

**Unsecured Connection**

**yazılım, multimedya yazılımı, grafik yazılımı, metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Secured Connection**

**yazılım, multimedya yazılımı, metin, grafik yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu**