Computer Networks - Assignment 3

Part 1: Implement SSH using socket programming

Objective

The objective is to implement SSH (Secure Shell) using socket programming. The SSH protocol facilitates encrypted communication and data sharing between two computers, ensuring secure access to a remote server. The client-side interface simulates a terminal, allowing users to connect to the server using a specified username and password. The server is responsible for authenticating the user, maintaining user credentials, and providing system information upon successful login.

Components

1. **Server (`server.py`):**

   The server script establishes a socket connection and listens for incoming connections. Upon connection, it authenticates users, retrieves and displays system information, and allows users to execute commands in a remote shell. The server supports concurrent connections and utilizes subprocess for command execution.

2. **Client (`client.py`):**

   The client script connects to the server using a specified username, server address, and port. It sends authentication credentials, receives a welcome message, and enters a loop where users can input commands. The client displays command outputs and supports an interactive shell-like experience.

3. **User Authentication (`loginUser.py`):**

   This module manages user authentication, loading user data from a CSV file, updating last login times, and saving changes. It uses hashed passwords for security and provides functions for authenticating users and managing user data.

4. **Command Execution (`command.py`):**

   The command module handles command execution, including a special case for the ‘cd’ command to change directories. It uses subprocess to run commands and returns the current directory and command output/error.

Conclusion:

The code provides a simple remote shell experience with user authentication and basic security considerations.

Part 2: DNS Resolver Application Overview

Objective

The DNS resolver application facilitates the translation of domain names to IP addresses through a client-server model. The client sends DNS requests with a unique Transaction ID to the server, which maintains domain-to-IP mappings. The server responds with the IP address corresponding to the requested domain.

Components:

1. **Client (`client.py`):**
    - Takes user input for a domain name.
    - Sends DNS requests with a Transaction ID to the server.
    - Displays the corresponding IP address or an error message.
2. **Server (`server.py`):**
    - Listens for DNS requests and processes them.
    - Looks up domain names in a mappings file.
    - Sends DNS responses with Transaction IDs and IP addresses or error codes.
3. **Mappings File (`mappings.txt`):**
    - Contains at least 5 entries mapping domain names to IP addresses.

Conclusion:

This DNS resolver application provides a basic yet effective solution for translating domain names to IP addresses. It employs a simple client-server architecture with Transaction IDs and demonstrates the DNS resolution process.

Part-3: Echo Client/Server Protocol Independence

Overview

The provided Python scripts, `client.py` and `server.py`, implement an echo client-server system with support for both IPv4 and IPv6. The code utilizes the `socket` library and `getaddrinfo` to enable communication over different network protocols.

Components

1. **Start Client Function.`client.py`**

- Accepts command-line arguments for the target port (`-p` or `--port`) and target host (`-a` or `--addr`).
- Uses `getaddrinfo` to obtain address information for the target host and port.
- Attempts to create a socket and connect to the server, handling errors gracefully.
- Establishes a connection and sends user input to the server.
- Displays the server's response.

2. **Start Server Function.** `server.py`
    - Accepts a port number as a command-line argument (-p or –port).
    - Creates a socket with support for both IPv4 and IPv6.
    - Binds the socket to the specified port and listens for incoming connections.
    - Accepts client connections and echoes received data back to the client.

Conclusion

The implemented echo client-server system demonstrates protocol independence, allowing seamless communication over both IPv4 and IPv6. The code provides a foundation for building networked applications that can adapt to different network protocols dynamically.

Part4: Collaborative Code Editing Client-Server Application

Overview :

This client-server application allows multiple users to collaboratively edit a shared code file. The users can make code changes in real-time, and the server maintains the shared code file, incorporating contributions from all connected clients.

Components:

1. `client.py`:

    The client-side script allows users to connect to the server, authenticate themselves, and then contribute to the shared code file.

- Users provide their author name.
- Users can input code changes or initiate a save operation.
- Code changes are sent to the server for processing.
- The user receives feedback on the success of their actions.

2. `server.py`:

    The server-side script manages client connections and handles code changes and save requests.

- The server listens for incoming connections.
- Each connected client operates in its thread for concurrent collaboration.
- Code changes are collected and applied to the shared code file.
- Save requests trigger the server to persist the current shared code to a file.

Features:

1. **Real-time Collaboration:**

   Users can make code changes simultaneously, and the server updates the shared code file in real-time. Each change is tagged with the author's name, providing transparency on who contributed to the code.

2. **Save Operation:**

   Users can save the current state of the shared code file, allowing them to persist the collaborative work. The server handles the save request and appends the current code changes to a file.

Conclustion

The addition of Multi-Server Support enhances the collaborative code editing application, making it adaptable to distributed development scenarios. This feature provides flexibility and scalability, allowing teams to collaborate effectively across multiple servers.