2201cs08

Aditya Yadav

Assign\_5: Telnet Client Implementation in C

Objective:

The objective of this assignment was to implement a basic Telnet client in C using sockets, allowing users to establish a connection with remote servers, send commands, and receive responses in real-time. The client also handles user input from the console and can terminate the connection using a "quit" command.

Implementation Overview:

The Telnet client is implemented using the following steps:

1. Socket Creation: A TCP socket is created using the socket() function, which is the foundation for connecting to remote servers.

2. Server Address Setup: The server's address is specified using a struct sockaddr\_in structure, which holds the IP address and port number.

3. Connection Establishment: The connect() function is used to initiate the connection with the remote server based on the provided address and port.

4. User Input and Communication: In a loop, the program reads user input from the console using fgets(), sends the input to the server using the send() function, and waits for a response using the recv() function.

5. Command Execution: The response from the server is displayed on the console. Users can send commands repeatedly until they enter the "quit" command to close the connection.

6. Error Handling: Comprehensive error handling is implemented using the perror() function to provide feedback if any system call (like socket(), connect(), or send()) fails.

7. Termination: The client detects the "quit" command and safely terminates the connection by closing the socket.

Code Structure:

- Header Inclusions : The code begins with including necessary header files such as <stdio.h>, <stdlib.h>, <string.h>,<winsock2.h>, and <ws2tcpip.h> for Windows-based socket programming.

- Winsock Initialization: Before creating the socket, the Winsock API is initialized using WSAStartup(), and WSACleanup() is called at the end of the program to clean up resources.

- Socket and Address Setup : The program uses the getaddrinfo() function to resolve the server’s domain name or IP address into a usable address structure.

- Main Loop: Inside the main loop, select() is used to monitor both server response and user input. This allows the client to receive data from the server asynchronously while also accepting commands from the user.

- Telnet Negotiation: The client handles basic Telnet negotiation commands such as IAC (Interpret As Command) when required.

Challenges Faced:

1. Telnet Protocol Handling:

- One of the main challenges was handling the Telnet protocol's negotiation sequences. Telnet uses special commands that need to be intercepted and processed separately. The negotiate() function was implemented to address these sequences.

- Telnet sends negotiation commands like IAC DO, IAC WILL, etc., and a basic implementation for negotiation was developed to respond to these commands appropriately. Handling advanced options such as encryption, window size, and echo was beyond the basic requirements of this assignment but is crucial for a complete Telnet implementation.

2. Cross-Platform Considerations:

- The original socket programming approach had to be adapted to work on Windows, where the Winsock API is required. In contrast, on Unix-based systems, no such additional initialization or linking is needed. This required the inclusion of winsock2.h and the linking of Ws2\_32.lib during compilation.

3. Input and Output Management:

- Managing both user input and server responses simultaneously required careful use of the select() function. This allowed non-blocking monitoring of both the socket (server communication) and the standard input (user commands).

4. Domain Name Resolution:

- Initially, the program only accepted IP addresses. However, after adding support for domain name resolution using getaddrinfo(), the client could resolve domain names and connect to servers by both IP address and hostname.

Example Commands Executed:

To demonstrate the functionality, the following example commands were executed after establishing a Telnet connection with a remote server:

1. login: Initiating a login session on the remote system.

2. ls: Listing directory contents on a Unix-based server.

3. pwd: Printing the working directory.

4. uptime: Displaying the system’s uptime.

5. quit: Terminating the Telnet connection and exiting the client.

Example Usage:

The following example demonstrates how the program can be run and used:

1. Compile the code using gcc on a Windows system, linking the required Winsock library:

bash

gcc -o telnet\_client telnet\_client.c -lws2\_32

2. Run the program and connect to a server:

bash

./telnet\_client

Enter server hostname (domain name or IP): towel.blinkenlights.nl

Enter server port: 23

Connected to server: towel.blinkenlights.nl:23

3. Send commands to the server and receive output:

bash

Enter command: ls

Server response:

<server output>

Enter command: quit

Exiting client...

Conclusion:

The Telnet client was successfully implemented using C sockets. It allows the user to connect to a remote server, send commands, and receive output in real-time. The client also supports basic Telnet negotiation, domain name resolution, and handles multiple commands until the connection is terminated by the "quit" command. By adhering to the assignment requirements, this implementation is expected to function correctly on various Telnet-enabled servers.

The primary challenges involved were handling Telnet protocol nuances and adapting the implementation for the Winsock API on Windows.

A screenshot of a computer screen

Description automatically generated