Name- ABHINAY KUMAR

Roll No- 2201CS04

Lab5 - Report: Telnet Client Implementation in C

Introduction

This project aimed to develop a simple Telnet client using C and the Winsock library for Windows. Telnet is a protocol that enables bidirectional, text-based communication between a client and server over a network, typically for remote system access. The primary goal was to establish a connection to a Telnet server, send commands, and display the server's real-time responses.

Implementation Approach

1. **Winsock Initialization**: The implementation begins by initializing the Winsock library with `WSAStartup()`, which is required for using network functions on Windows.

2. **Address Resolution**: The server address is resolved using the `getaddrinfo()` function, which supports both IPv4 and IPv6 addresses. The server name is provided as a command-line argument.

3. **Socket Creation and Connection**: A socket is created using `socket()`, followed by attempts to connect to the server using `connect()`. The program tries all resolved IP addresses until a successful connection is established.

4. **Handling the Telnet Protocol**: Once connected, a Telnet-specific control sequence (IAC DO TERMINAL-TYPE) is sent to negotiate the terminal type, allowing the server to understand the client's terminal capabilities and preventing display issues.

5. **Main Interaction Loop**: The core of the client is a loop that accepts user input as commands to send to the server. The server's responses are then displayed to the user.

6. **Telnet Control Sequence Handling**: Telnet servers often send control sequences (typically starting with the IAC byte `\xFF`), which need to be filtered out from the server's response before being displayed. This is handled by filtering these sequences from the data received from the server.

7. **Cleanup**: When the connection is closed, all resources such as the socket are cleaned up using `closesocket()` and `WSACleanup()` to prevent resource leaks.

Challenges Faced

1. **Telnet Protocol Specificity**: Handling Telnet-specific control sequences, sent by servers for terminal negotiation, was one of the major challenges. If not managed properly, these sequences could lead to garbled or unexpected output. To resolve this, a function was implemented to filter out control sequences, ensuring clear server responses.

2. **Socket Programming on Windows**: Working with the Winsock library posed unique challenges compared to socket programming on Linux, primarily due to the need for `WSAStartup()` and `WSACleanup()`. Ensuring proper socket management and cleanup after disconnections required careful resource handling.

3. **Buffer Management**: Managing data buffers was challenging, especially when handling Telnet control sequences. Ensuring proper null-termination and processing of the buffer without data loss or overflow was critical.

4. **Server Compatibility**: Not all Telnet servers responded consistently to client requests. Some servers produced non-standard outputs or required specific commands. Adapting the client to communicate with various servers involved testing across multiple public Telnet servers.

Conclusion

The Telnet client implementation successfully connects to a Telnet server, sends commands, and displays the server's responses while handling Telnet control sequences. This project provided valuable experience in network programming with sockets and exposed the complexities of the Telnet protocol. While challenges were

encountered with control sequences and server compatibility, they were overcome through careful design and testing.