ECEN 602 – MP1 Readme File

**Echo Protocol**

The Echo Protocol is a service in the Internet Protocol Suite that was defined in 1983 in RFC 862 by Jon Postel. It was originally proposed to test and measure an IP network. A host may connect to a server that supports the Echo Protocol using either TCP or UDP on the well-known port number 7. The server responds with an exact copy of the data it received. Figure 1 describes the Echo Protocol

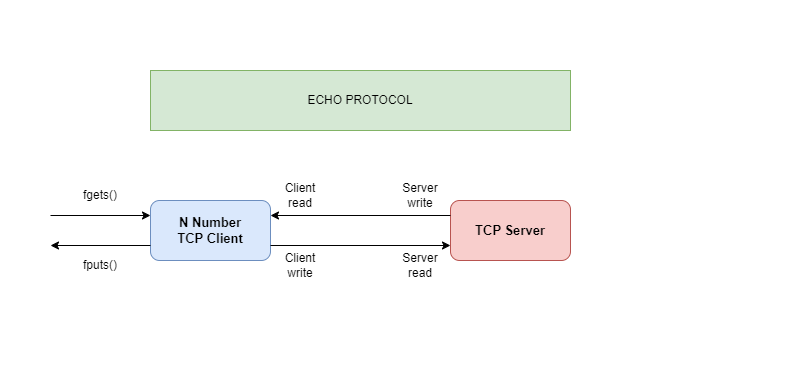


Figure 1 Echo Protocol

**TCP Client:**

The client code establishes a TCP connection to the server and facilitates communication between the user and the server. Below are the process followed in the client code organization,

**Socket Creation:** A TCP socket is created using the socket() system call.

**Server Connection:** The client attempts to connect to the server using the connect() system call, where it specifies the server's IP address and port number.

**User Input Handling:** By calling the function func(sktsever) ,the client waits for the user to enter a line of text from the terminal, which is then sent to the server via the socket using the write() system call.

**Echo Response:** After sending the data, the client waits for the server's response, which is expected to be the same message (echoed). The response is read using the read() system call and then displayed to the user.

**Exiting:** The client continues to accept input and echo messages until the user types "exit". When this occurs, the client closes the connection and exits the network.

**TCP Server:**

The server code implements a concurrent TCP echo server that listens for incoming client connections on a specified port. It performs the following steps:

**Socket Creation:** The server starts by creating a TCP socket using the socket() system call.

**Binding:** The server binds the sockets to a specific IP address and port number provided as a command-line argument. This allows the server to listen for incoming connections on the designated port.

**Listening:** The server enters a listening state using the listen() system call, where it waits for clients to connect. It allows maximum of 10 clients to connect simultaneously, by handling each new client in a separate child process.

**Client Connection Handling:** When a client connects, the server accepts the connection using the accept() system call and forks a new process to handle communication with the client. The child process receives data from the client, processes it, and echoes it back using the read() and write() system calls.

**Echo Service:** The child process continues to read input from the client, echo the message back, and log the interaction to the terminal until the client closes the connection or sends the "exit" message.

**Shutdown:** The server handles client disconnections and child process termination properly. It includes a signal handler for SIGCHLD to avoid zombie processes caused by the forked child processes.

This structure allows the server to handle multiple clients concurrently while providing an efficient and reliable echo service. Each client interaction occurs in its own child process, ensuring that multiple clients can connect and communicate independently.