**ECEN 602 – Machine Problem 1**

**Submission:**

**Team 11:**

1. Dhanraj Murali – UIN:734003894
2. Swetha Reddy Sangem – UIN: 433003591

**Implementation Roles:**

1. Dhanraj – Server Implementation
2. Swetha – Client Implementation

Both were involved in the ChatGPT part of the submission. Same has been executed and tested with Hera servers.

**Submission:**

1. Submission 1 – Source Code/Make file written by the team
2. Submission 2 – Source Code written by team and optimized by ChatGPT/Make file
3. Submission 3 – Source Code/Make file generated by ChatGPT
4. README.pdf
5. Test case screenshots as PDF

**Execution:**

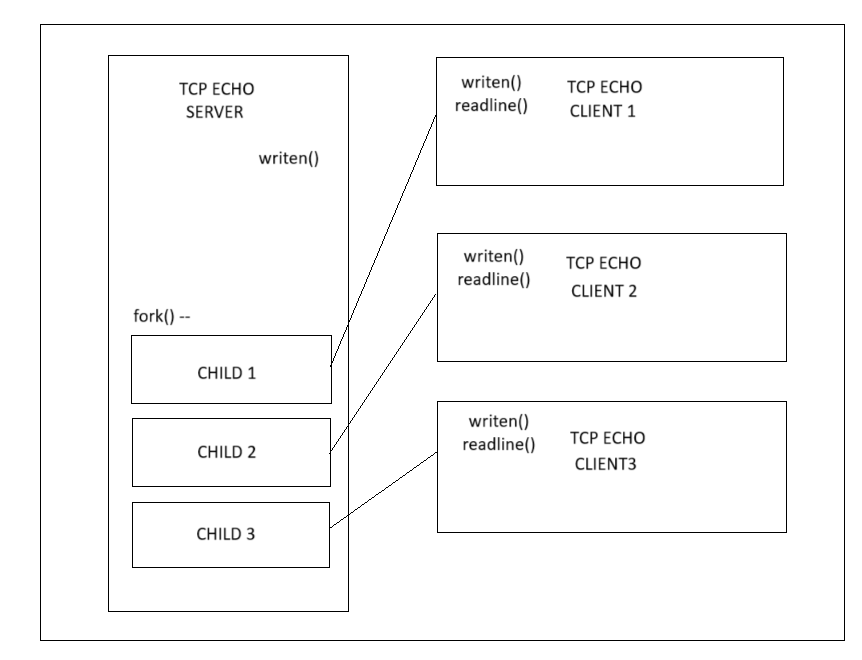
Run the following commands in order,

Open 2 terminals,

1. Terminal 1: make clean; make
2. Terminal 1: ./server <port\_no>
3. Terminal 2: ./client <127.0.0.1> <port\_no>

**Description:**

**Architecture:**

****

**Code Description:**

1. writen();

The *writen()* function uses the *write()* function to write the data stored in character array to a socket descriptor (client/server) that is provided as an input. The write function returns the number of bytes that is written to the socket. If all the bytes are written and the value becomes 0, we will decrement the value of the number of bytes to be written and check if number of bytes to be written is 0. If so, then we conclude that the write is complete. The write function will return -1 if there is any error, in the case when errno is EINTR then we will reset the value of the number of bytes that is written to the socket to 0 and try to write again, else we will return -1.

1. readLine();

The *readLine* reads a line of text (terminated by a newline character) from a socket. It uses the *readChar()* function to read characters one by one until it encounters a newline or reaches a maximum line size. It returns the number of characters read, -1 on error, or 0 if it reaches the end of the data.

1. readChar();

This is implemented since the built-in *read*() function is slow. *readChar()* reads a single character from a socket. It maintains a buffer to store received data and returns one character at a time from that buffer. If the buffer is empty, it reads more data from the socket. It returns 1 if it successfully reads a character, 0 if it reaches the end of the data, and -1 on error.

1. sigchild();

The *sigchild*() function uses *waitpid* to check the termination status of the process which is located using process id provided as input. It returns the value of the child process that is terminated and prints it on to the console. It is invoked using the signal function in the main. This signal function must be initiated only once before the server accept its first client connection i.e, the server’s first forking.

**Concurrent client handling:**

The function *fork()* is used to handle multiple client instances. The server process is forked into parent and child process. In the listen function call in the server, the maximum client that can be handled by the server is mentioned in the queue argument as 16. This means the server can handle only 10 client processes simultaneously.

Once the *fork()* is invoked by the process, it return 2 values. The first value returns the process id of the child to the parent process. The second return returns 0 to the child process. We can identify the child process by checking return value of the *fork()* function call.

The child is only meant to handle the request of the client whose connection is already established, so the client is not required to continue listening for any new client connection requests on the configured port. So, we close the listening function.

The parent is not responsible for handling the read and write to the client process. So this read/write block is executed only in child by applying if condition to check if return value of *fork()* is 0. Then the parent continues to listen for any new client connection establishment request.

**Error handling:**

All built-in functions that are used are as follows,

1. socket()
2. bind()
3. listen()
4. connect()
5. read()
6. accept()

All above functions return -1 in case an error is encountered. We check the return value for ‘-1’ or < 0 and if it is matching then we print the message in console with message.

* We check the number of input arguments in the *main* functions of the client and the server, and if the number of inputs is varying than what is expected then we throw an error and return.
* We check the return value of the *write* function in the *writen* function, if it is less than 0 then we check if errno is EINTR. If it is EINTR we try to write again. If it is a different error then we return.
* We check the return value of read function in the *readChar* function and if it is less than 0, we return -1. The function also checks for EOF condition and returns 0.
* We check return value of *readChar* function in *readLine* function and return -1 in case of error.

**References:**

[1] Unix Network Programming, Volume 1, The Sockets Networking API, 3rd Edition

[2] Beej’s Guide to Network programming