Part A Design and Test Documentation

Jiaye Wang

February 1, 2021

Design

This application is developed on `tux8`. Port 30001, 30002, 30003 for proxy server, TCP server, and UDP server, respectively. The implementation is mainly referenced from chapter 6 of *Beej's Guide to Network Programming* and the UDP server/client example from `man getaddrinfo(3)`.

The development process of this application is roughly divided into two parts: network and information processing. The network has a bigger weight. Since this application has Beej's code as a reference, the logic and implementation are not hard to understand. After the server and client built the connection. The client and the server are essentially listening to each other. The client sends a message, the server receives the message, and sends feedback back to the client. This process repeats ideally until the client is terminated. However, when the client sends the termination command, clint is not simply exited and server receives that command and exit too. The logic here is after the server received the termination command from the client, the server needs to send back to the client a termination command too, and then exit. The termination command that the client received from the server tells it can exit now.

The proxy server on the other hand is acting as an intermediary, that is, it has both client and server functionality. Hence, it is not a real server or a real client. It is a server when a client connect to it, and it is a client when it connects to a real server. For question 2, the proxy uses TCP when communicating with the server and when communicating with the client. For question 3, the proxy uses TCP when communicating with the client and

UDP when communicating with the server.

The proxy server will listen the command from the client and redirect to the real server, then gets the feedback from the real server and sends back to the client. In this application, the proxy server support an addition command, the `all` command. The real server will treat it as invalid. The `all` command sends seven requests to the actual server, concatenation seven responses with a header, and send back to the client.

Note in the proxy server, it uses `read()/write()` when communicating with the real server, instead of `recvfrom()/ sendto()` or `recv()/send()`. The main purpose is those methods will able to handle both UDP and TCP server (this *shortcut* may cause potential issues that the test cases not covered).

Finally, it is the implementation, that is, put everything together.Notice that there are plenty reusable code. For instance, both TCP server/client and the proxy server need to generate two stream socket and both TCP and UDP servers need to process the command in the same way, in this case, get corresponding weather of the day. Therefore, to make the application more organized, two additional files are created: `network.c` and `cmd_procssor.c`.

`network.c` has four functions to generate a socket for TCP client, TCP server, UDP server, and UDP client, respectively. `cmd_processor.c` include possible commands for real servers, hand-coded weather for each day of the week, and a simply implementation to verify the command and get the weather of the week with the given day. The TCP client also has a pre-check of the user command. It will not allow more than 4 + 1 characters inputs, since our longest valid command is `quit` with a new line character.

Testing

The testing is mainly based on the black box testing. For the network part, check the connections with unexpected hostname or port and whether or not correct number of command line arguments are used for the client and proxy server. The TCP client is not supposed to connect to the UDP server. Only one TCP client, TCP server, UDP server or proxy server is allow to run at same time, that is, the user does not allow to run two TCP servers at the same time.

Check all the commands and `all` command with proxy and without proxy that are work as expected. Then check all possible command that are not expected, these include: command is too long, a proper command concatenates with some random letters.

Please check the limitation section for more test limits

Limitation

The TCP and UDP servers only allow one connection, that is, only one TCP client or proxy server can connect to the server, another connection request will not work. Also, the TCP clint is not able to connect to the TCP server if the proxy server is already connected to the TCP server. However, if the proxy server is connect to a UDP server, TCP server and TCP client can still build a connection with corresponding port.

Terminate all the running servers and client and then restart. For example, if TCP client, proxy server, and TCP server are running and want to run the UDP server, then need to terminate all three and run TCP client, proxy server, and UDP server.

The command must be single one word, without space, i.e. `<Mon>`, `<quit>`. `<Mon Tue>` works, it will send back both Monday and Tuesday' weathers, but it considers as unexpected. Empty input should not be allowed, but it does not affect the program. However, not encourage with too many empty inputs.

Limits on the user environment. User must have internet connection. User need to know how to find the hostname of his/her machine. Some possible ways are listed in README.