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. It is a server when it connected with a client, and it is a client when it connected 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.

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

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.

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.

Testing

The testing is mainly based on the black box testing.

First by checking the connections with unexpected hostname or port.