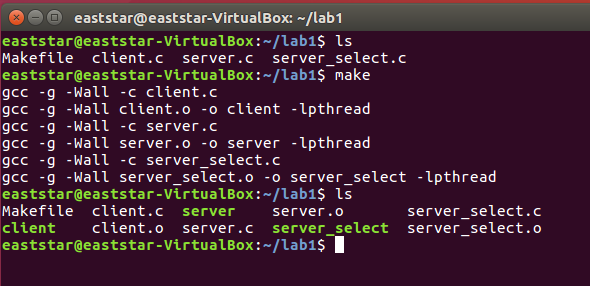
Project#1 Report - 20180065 Donggyu Kim

1. How to Compile the program

Type ‘make’ or ‘make all’ in the directory to generate executables.

It’ll generate three executables: client, server, server\_select.



2. Command line formats

1) Client => ./client –h <host> -p <port> -o <operation> -k <keyword>

My program doesn’t care about the order, but just follow the argument format.



2) Server => ./server –p <port>



3) Server\_select => ./server\_select –p <port>



3. Self Test

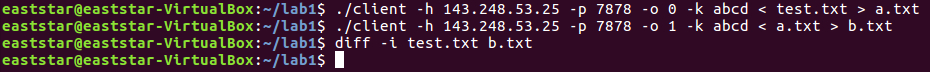
0) Generate a random test.txt file



Generate 50MB sized test.txt with alphabet and number.

1) Client Test

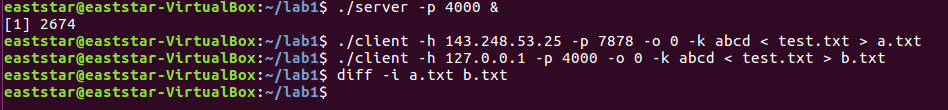
First, send test.txt data to test server(143.248.53.25:7878) for encryption. The encryption result of test.txt will be written in a.txt. Then, send this a.txt data to server again. This time for decryption, and its result will be written in b.txt. By comparing test.txt and b.txt, I can check whether the client works well or not.



No output from ‘diff’ verifies the client worked well.

2) Server Test

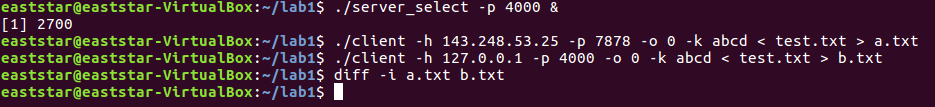
First, run my server with port 4000. Send the same test.txt data to both test server and my server. The result from test server will be in a.txt, and my server one will be in b.txt. Compare these two .txt files to check if my server works equally as test server.



My server worked well.

3) Server\_select

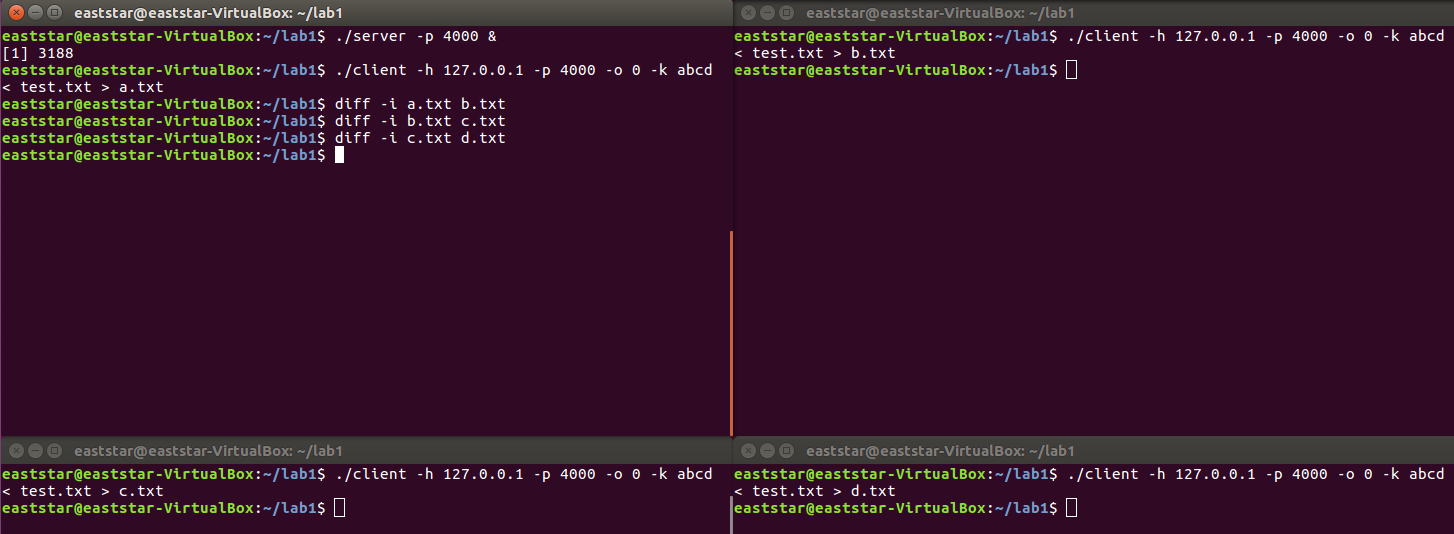
Almost same process with 2). Just use my server\_select instead of my server.



My server\_select also worked well.

4) Handling multiple clients

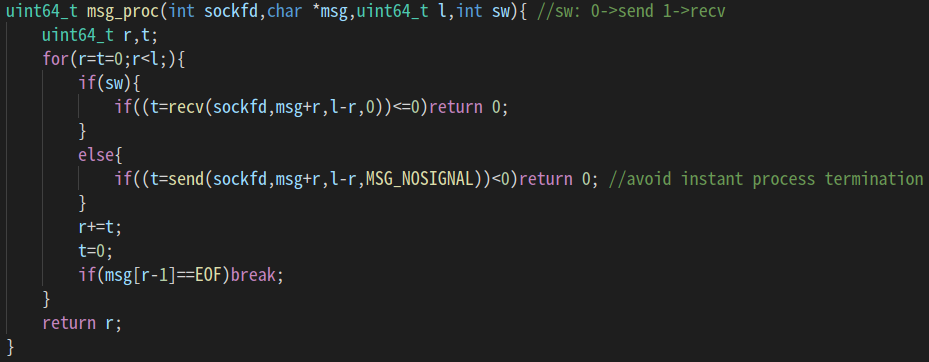
I don’t know how to run several clients at once, so I just executed my clients simultaneously on each terminal. The clients’ output is redirected to text files: a.txt, b.txt, c.txt, d.txt. Again, use diff to compare them.



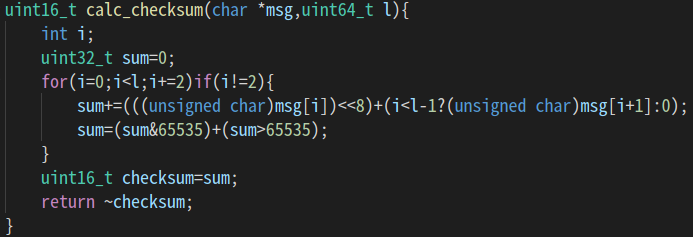
The clients terminated almost at the same time, and their results were same too.

4. Structure & Implementation

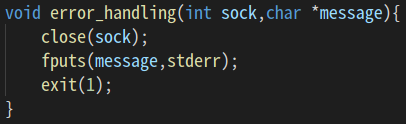
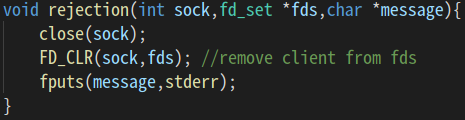
0) Helpful functions



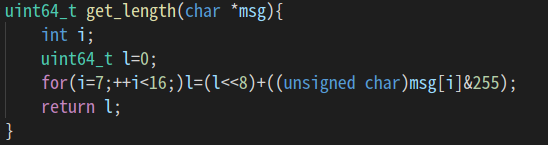
This is the simplest form of message processor. This function handles all of sending/receiving messages until the buffer becomes empty. This function is in both client and server with some modification by its purpose. It returns sent/received message length for success, and 0 for failure.



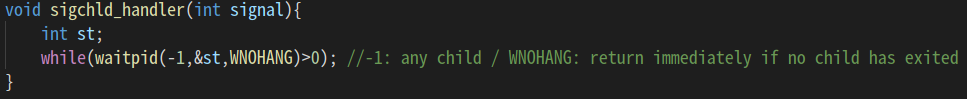
The function calc\_checksum calculates the message’s checksum, under the assumption of that message is protocol-following. It is implemented well – overflow during summation is avoidable.

Error handlers. Left one prints out the error message in stderr and terminates the process. Right one, only for ‘server\_selcect.c’ also handles some kind of error, but just removing clients from fd\_set. ‘server.c’ uses fork() and make connection between each child server and client, so the child server process should be terminated when some connection error occurs. On the other hand, ‘server\_select.c’ uses select(), which must not be terminated just because of a connection issue of a client so that server can still work for other clients.



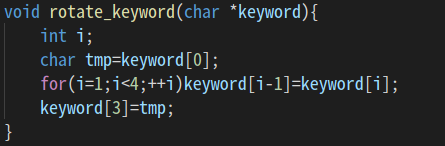
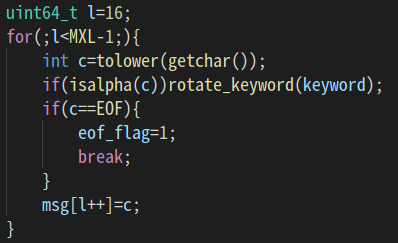
Extract length part form the message, while message is not violating the protocol. By comparing the return value of this and the real length of message, server can check if the message is protocol-violating and decide to reject or not.



As ‘server.c’ uses fork(), SIGCHLD handler is needed.

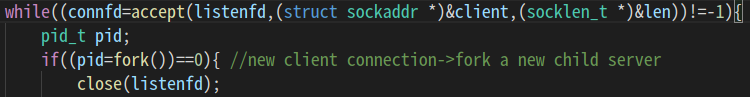
1) Client

The client structure is quite simple. It just follows the typical flow of connection-oriented one; creates a new socket-connect-send/recv. It reads the message from stdin and modifies it into the protocol form. And then just print out the server-operation result after some send/recv.

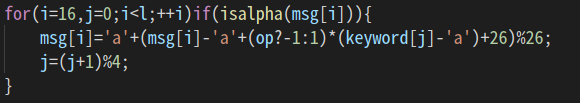
 

The client can send maximum 10MB for each time so the client should send it as some pieces. And rotating the keyword can make this problem easier.

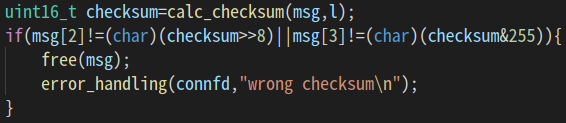
2) Server



For simultaneous multiple client handling, accept() and fork() is used as above. Each time a connection is established, parent server process forks a new child server for a new client.

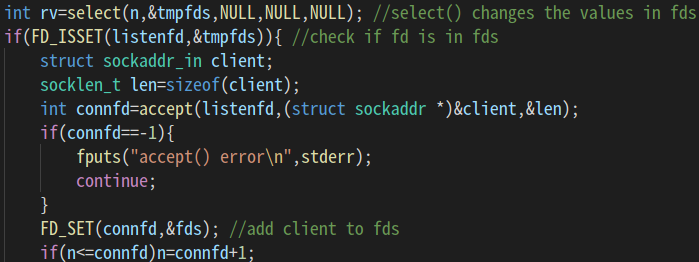


Encryption/decryption. The operation is applied only for alphabet characters.



If checksum of a message from client is wrong, the child server process closes connection socket and terminates.

3) Server\_select



The only difference between ‘server.c’ and ‘server\_select.c’ is handling a connection from a new client. Instead of using fork(), it uses select() and fd\_set to manage sockets.

+) I set the backlog value as 5 for linsten(), both of server and server\_select.