20140040 Keonil Kim

CS341 Project0

1. The instruction to compile the program

make all: create server and client program
make server: create server program
make client: create client program
make test-client: test client program with test server port 5000, 5003
make clean: remove compiled client and server program

2. Self-test result of the server & client

I tested with the given txt files for test. The client program properly works with the test server with port 5000, 5002, and 5003. For the server program, I ran it on the AWS EC2 ubuntu linux server, and test it with my client program. With my test cases, the server behaves exactly the same way the provided test server does. I also test both client and server with wrong checksum, multiple connections, zero size text, 100MB size text.

3. The structure of the server and client

A) client.c

There are five parts in client.c and it is also clearly commented in the source file.

1. “Functions”

Define additional functions used in the client program. I implement ‘calc_checksum’ which calculates the checksum of the given message with given length. It does not exactly return the real checksum, rather, it returns the value right before the checksum. I apply the last ‘~’ operation with the result of this function to use it generally in my program

2. “Variables”

I define most of variables used in the client program.

3. “Part A”

In Part A, it parses the command line with getopt function and save it to variables. Then it creates a socket and connects to a server. Although I commented all the perror lines, it catches error properly when socket api functions are not working well.

4. “Part B”

In Part B, it gets message through stdin, adds header, calculates the checksum, and creates the message for the server. I used get char and feof, ferror functions and for handling stdin and calc_checksum defined at Functions part is used here.

5. “Part C”

In Part C, it sends and receives the message to/from server. While loops are applied for both send and recv functions until the bytes sent/received reaches the expected amounts. Perror lines are also commented, working well when uncommented. After the message received, calc_checksum is used again to check if the checksum of the received message is valid. If not, connection is closed immediately. Then the received message, without header, will be printed out to monitor with printf function. The whole while loop containing Part B and Part C breaks when it reaches EOF from the received message.

B) server.c

There are four parts in client.c and it is also clearly commented in the source file.

1. “Functions”

Define additional functions used in the server program. I implement ‘calc_checksum’ which calculates the checksum of the given message with given length. It does not exactly return the real checksum, rather, it returns the value right before the checksum. I apply the last ‘~’ operation with the result of this function to use it generally in my program. This is identical to the function defined in client.c
Also, for server program, I implement enc and dec function to cipher the message. Instead of using ASCII, I declare abc array and shift with the number (mod 26). dec function is simply implemented with enc with negative input.

2. “Variables”

I define most of variables used in the server program.

3. “Part A”

In Part A, it parses the command line with getopt function and save it to variables. Then it creates a socket and does bind and listen through api. Again, perror lines for each function is commented, while working well when uncommented.
I set backlog 100 for listen function.

4. “Part B”

In Part B, it uses accept function and receives/sends functions. Multiple connection is handled with fork(). receiving and sending works as it does in client program. Additional part is to handle the header even if the header is not received at once. Before it sends message back to server, it encrypts or decrypts the message as given with op in the header of the message received.
Also, it calculates checksum and add it to the header, and then sends it to the client.