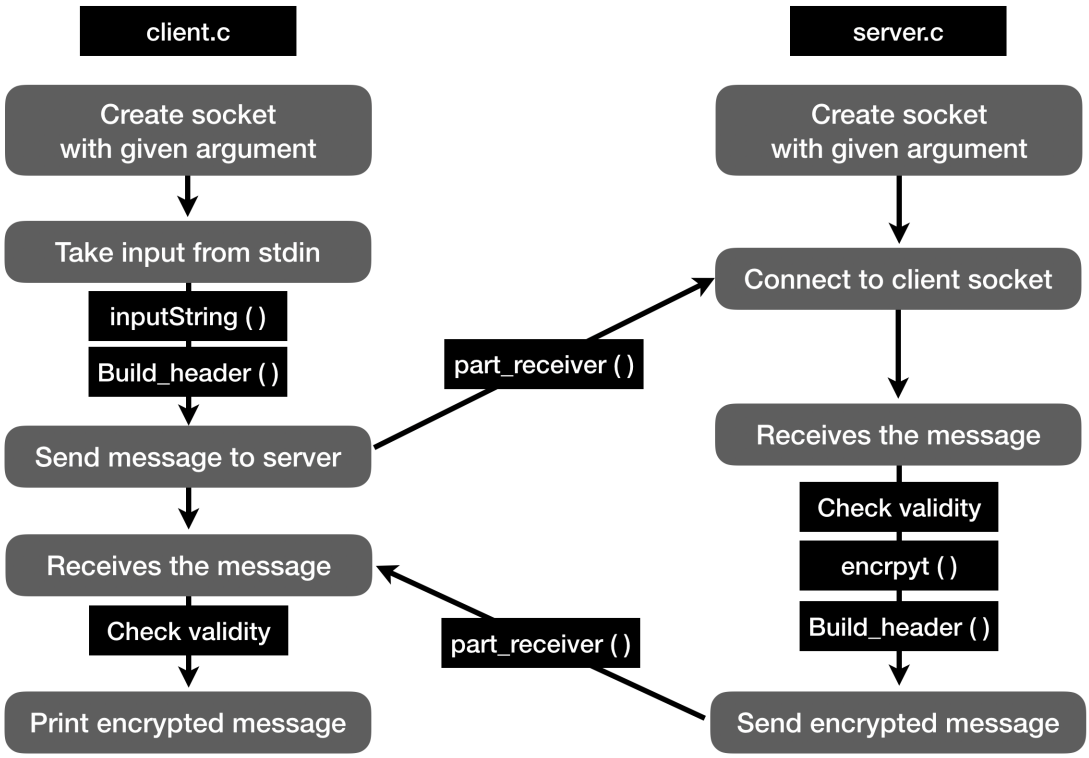
20140496 Jaehwi Jang
CS341 Introduction to Network

Project #0: Socket programming

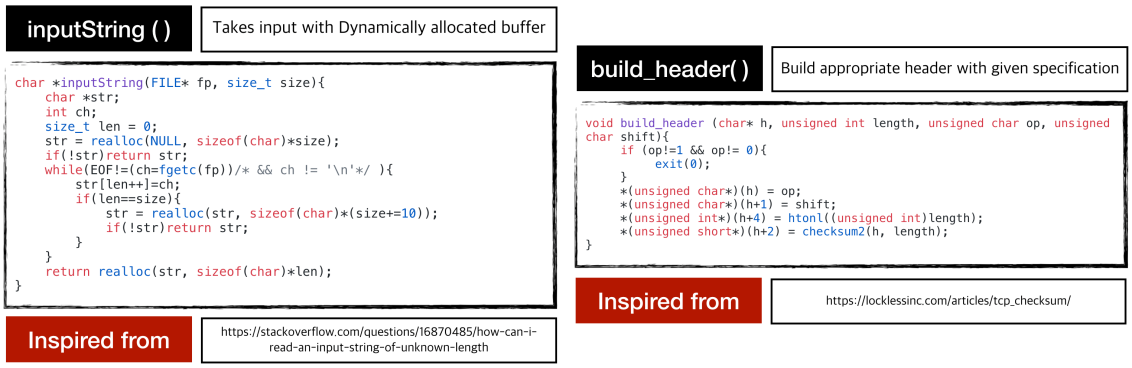
I used previous works from CS230 as skeleton codes. The last assignment of CS230 is to implement simple socket program which can perform basic shell operation remotely.

The most critical difference between simple socket programming and this assignment is that it requires to follow application layer protocol given.
The following is diagram showing how my server-client works in general scheme. Although basic frame work has same format of simple socket communication, several functions are added to meet the specification. Blacked box indicates newly implemented on top of regular socket communication. I would like to briefly explain what each function does and its structure.



[Figure 1. General scheme for my code]

Before the start, I want to point out that inputString() and checksum() functions are mostly inspired from other's code.
inputString() inspired from:
https://stackoverflow.com/questions/16870485/how-can-i-read-an-input-string-of-unknown-length
checksum() inspired from:
https://locklessinc.com/articles/tcp_checksum/



[Figure 2 Basic codes are inspired from the Internet]

inputString()

This function is used only for client side. This function creates string buffer which allocated dynamically for stdin input. It initializes the buffer with certain length and if the input goes beyond it, it reallocate with 10 more bytes in size. I modified the code to terminate elongation only for EOF input.

Check validity – checksum2()

Every sending part of this scenario requires to check the validity of the messages. We can check the validity by looking up checksum value . II cited the code that calculates checksum from locklessinc.com.

build_header()

One task this function does is to check whether op field is valid (have appropriate value of 0 or 1). Another task is to put appropriate header field infront of original message. It takes account difference in endianness betweeen the host and the networks when it deals with 4 byte data.

part_receiver()

As send() function can not send the whole message at once for a big message, we need to call recv() function several times to get the full message. part_receiver() function call recv() function until the full message arrives.

part_receiver() function is used in both client.c and server.c but slightly different. I implemented dynamic allocated buffer in server side since client can anticipate the size of the incoming message while server can not.

First part of part_receiver() function checks first 8 bytes of message. If it gets a header portion, part_receiver() can call recv() function several times until it gets the message of the length specified in the header.

encrypt()

This function is only for the server side. it encrypts or decrypts the message according to given header information. First, it checks the header bits in the message. After so, it passes the encryption information to cipher() function. cipher() function goes through every string bytes in the message, and perform per char operation.