CS341 Project 1 REPORT

20150109 Donggyun Kim

There are 3 program files, ‘client.c’, ‘server.c’ and ‘functions.h’.

1) functions.h

‘functions.h’ is a header file that is shared by both ‘client.c’ and ‘server.c’. It defines the buffer size, message structure, and checksum function.

BUF_SIZE: The maximum number of bytes which can be loaded in sockets. This is used to make buffers for data and socket communication. The value 10000000 is motivated by restriction about message size in the protocol, which is 10MB.

struct message: A structure for message, both for sending and receiving. Following the protocol, it has 2-bytes op filed at the first of the message which is followed by 2-bytes checksum field and 4-bytes keyword field. Then 64-bits length field follows and the remaining one is data field. Here I restricted the maximum data size by BUF_SIZE-16, so that whole size of the message cannot exceed 10MB.

checksum: This function is copied from http://locklessinc.com/articles/tcp_checksum/ and used by both ‘client.c’ and ‘server.c’. It computes 1’s complement sum of the whole data splitted by each 16 bits, then return 1’s complement of the sum. This is called when client or server fills checksum field of a message to pack the message to send. This is called also when they check validity of received message by computing checksum over whole the message and check whether it’s 0 or not.

2) client.c

‘client.c’ is a program for clients. It defines a function ‘parse’ and uses predefined functions from various standard libraries and ‘checksum’ function from ‘functions.h’.
Roughly it can be partitioned into Argument Parsing, Message Packing, Socket Opening and Connection Establishment, Communication, and Message Unpacking and Socket Closing.

Argument Parsing: Here client checks whether 8 arguments are given by the command line and parses arguments by using ‘parse’ function. ‘parse’ function detects four formats - ‘-h’, ‘-p’, ‘-o’, and ‘-k’ - and packs corresponding arguments into given char *args[], ordered by host → port → op → keyword.

Message Packing: Then, Client packs a message to send. It fills each field with given arguments. It fills op field and length field by the network order. After filling all but checksum field, it computes checksum of the whole message and fill the value into checksum field.

Socket Opening and Connection Establishment: Now client creates a socket for communication and connects it to the server with sockaddr_in struct filled by given arguments from the command line.

Communication: After connection is successfully established, client sends a message to the server by writing the packed message struct to connected socket. If it fails to send data it intended, prints error message and exits. Otherwise, it receives a message from the server by reading from the socket to a buffer. Here it also checks whether length of received message equals length of one it sent and exits if not.

Message Unpacking and Socket Closing: Finally, client writes content of data field of the received message into the stdout and finishes its lifetime by closing the connected socket.

3) server.c
‘server.c’ is a program for servers. It defines functions ‘encode’ and ‘decode’ which encrypts and decrypts a text message by shift encoding/decoding with a given keyword, respectively. As ‘client.c’, it includes standard libraries and checksum function from ‘functions.h’
Roughly it can be partitioned into Argument Parsing, Socket Opening, Communication and Message Unpacking, Protocol Specification Checking, Text Encryption or Decryption, Message Packing and Communication, and Socket Closing.

Argument Parsing: Server checks validity of argument format and extracts a port number from it.

Socket Opening: Server creates a listening socket and bind it with server address. Then it set the socket to listen. It allows any IP addresses to request connection.

The remaining parts are for servicing multiple clients. Server forks a child whenever it accepts a connection from a client, so that the child serves the client and parent continue the loop for waiting other connection requests. Child and Parent closes useless sockets(listening socket and connected socket, respectively) to avoid memory leak.

Communication and Message Unpacking: Child process of the server receives a message from its client and unpacks the message. It recieves a message at most BUF_SIZE length, which is 10MB. If it failes, it closes the socket and exits. It converts op field and length field into the host order.

Protocol Specification Checking: It checks whether the message violates the protocol by checking the length field and compute the checksum. If either of them is violated, it closes the socket and exits.

Text Encryption or Decryption: After checking validity of the message, it encodes or decodes data field of the message using ‘encode’ function or ‘decode’ function, depends on op field. If op field is neither 0 or 1, it closes the socket and exits.

Message Packing and Communication: After converting the data, the child process packs the data into a new message, with same op, keyword and length field and zeroed checksum field. Then it recomputes the checksum and fill the value to checksum field. After packing the message, child sends it to its client.

Socket Closing: Child process closes the socket either from ignoring requests by error or protocol violance, or completion of the communication. Parent process closes the connected socket right after it forked and continue to the next iteration.

Instructions to Compile

```
>> make all
```

Self-Test Results

```
>> ./server -p 10000
>> ./client -h 127.0.1.1 -p 10000 -o 0 -k cake < input.txt > output.txt
>> ./client -h 127.0.1.1 -p 10000 -o 1 -k cake < output.txt > output2.txt


<input.txt>
networks
I love CS341! It's great!
This is a short message.
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
 1234567890
	~!@#$%^&*()-_+=/?,.<>[]{}\|

<output.txt>
pedaqruw
k lyzg cc341! mv's qvgad!
xjic mu a clqrd qgsceie.
kfedojihsnmlwrqpavutezyxidcbmhgfqlkjuponytsrcxwvgbaz
 1234567890
	~!@#$%^&*()-_+=/?,.<>[]{}\|

<output2.txt>
networks
i love cs341! it's great!
this is a short message.
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz
 1234567890
	~!@#$%^&*()-_+=/?,.<>[]{}\|
```