Memo

Introduction:

This assignment is to design a network system, consists of a server and multiple client. This system supports client registration and direct communication between clients. Each client has to register with the server before it can communicate with other clients within the system.

Implementation:

Registration:

In our implementation, the system contains one centralized server and a fixed number of client. Each client registers to the server by sending a random message to the server. Once all clients have registered with the server, the server will broadcast all clients' ip address. Before then, clients are not able to communicate with each other.

Direct communication:

All direct communication between two clients will go through the server, and the server is responsible for routing messages to their destination. Each message is structure in the format "MsgType, dest_ip, content//END". If client A want to communicate with client B, it will send a message "SEND, client_B_IP, message//END". After the server receives that message, the server will include sender's ip in the outbound message and send the new message to client B.

Termination:

The centralized server will terminate in either one of the following condition

1) One of its registered client's socket has disconnected
2) The server received a termination signal from a client, a termination signal is a message in which the MsgType is "TERM".

Before the server terminate, it will broadcast a termination signal to all of the registered client. After receiving that signal, client will exit.

Design:

For demonstration purpose, out program support 2 clients and 1 server. APIs exposed to our end users include:

1) *Client-Side API:*
   void start() //start a client, bind to port 9000, and register to the server
   void send_message(char* msg, char* receiver_ip) //send a message to the given client
   void send_termination_signal() //send a termination signal to the server
2) *Server-Side API:*
   void start() //start the listening on port 8080 and block until all clients have registered
   void terminate //send termination signal to all clients and then terminate the server

An example of interactions might be

```
Server* s = new Server("127.0.0.1");
s->start();
Client* s = new Client("127.0.0.3", "127.0.0.1");
s->start();
Client* s2 = new Client("127.0.0.2", "127.0.0.1");
s2->start();
Client* s3 = new Client("127.0.0.4", "127.0.0.1");
s3->start();

s->send_message("hello", "127.0.0.3");
```

```
s->send_termination_signal();
```

Challenges & issues:

Challenge:

The major challenge we have is designing the message's format. Since the server might read in multiple messages with a single recv, a message format is necessary to break the received data into several messages. We designed a standard message format, and before sending a message, user's message content will be wrapped into that standard format, and then this wrapped message is sent to the server.

Issues:

1) Number of supported client

Currently, the number of client our server support is fixed. Our team need to modify our implementation so that clients can come and go, existing client's ip table must be updated accordingly.

2) Multi-threading / Child-Process

Our server is single-threaded, in the future, we want to introduce a thread pool or take advantage of child processes to handle client connection or messages. This way, one client will be less likely to block another.

3) Receive Buffer

Our receive buffer is hard coded. If the buffer is filled, but the last message has not been fully received, out program will have unexpected behavior. Our next step is to handle partial message on the server side.

4) Port

In both server and client, a server's and client's port is hardcoded. To make our program more flexible, and make it easier to test the whole system on one machine, we might consider modifying server's ip table's structure and change some methods' implementation.