Computer Network Final Project: CNline

第 28 組

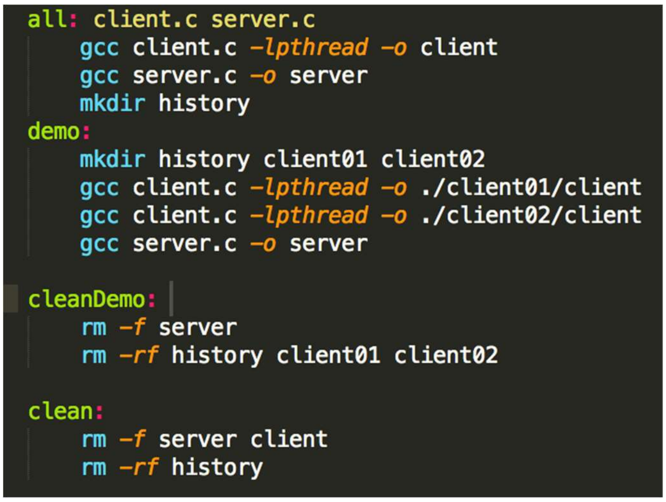
b04701232　陳柔安　r06944026　方珮雯

Protocol Specification

- Use TCP to send and recv messages % files

Run server and clients:

Check Makefile:



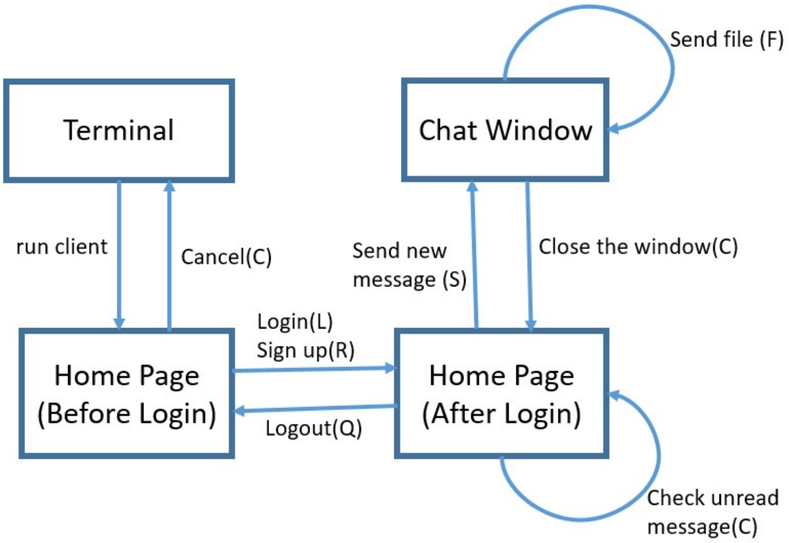
```
all: client.c server.c
    gcc client.c -lpthread -o client
    gcc server.c -o server
    mkdir history
demo:
    mkdir history client01 client02
    gcc client.c -lpthread -o ./client01/client
    gcc client.c -lpthread -o ./client02/client
    gcc server.c -o server

cleanDemo:
    rm -f server
    rm -rf history client01 client02

clean:
    rm -f server client
    rm -rf history
```

```
$ make
// build client.c, server.c and then make “history” folder to store future data
$ make demo
// folders “client01”, “client02” are created
// build files are created respectively to simulate behaviors of individual users
```

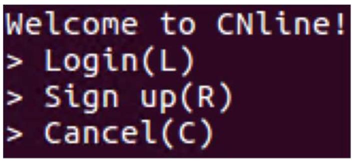
```
/*   in the same terminal     */
$ ./ server 3000              // run server with port number 3000
/*   open a new terminal      */
$ cd client01         // switch to the folder of the first client
$ ./client                    // run the first client
/*   open another new terminal   */
$ cd client02         // switch to the folder of the second client
$ ./client                    // run the second client
```

User Guide

UI Flow



1. Home Page (Before Login)



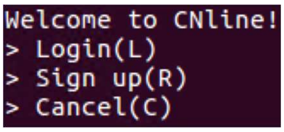
```
Welcome to CNline!
> Login(L)
> Sign up(R)
> Cancel(C)
```

/*　Sign Up　*/

Step 1:　type ‘R’ or ‘r’ to sign up

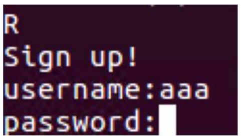
Step 2:　type new username and password

Step 3:　If sign up success, you can choose what to do next



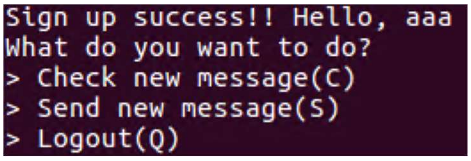
```
Welcome to CNline!
> Login(L)
> Sign up(R)
> Cancel(C)
```

➜



```
R
Sign up!
username:aaa
password:
```

➜



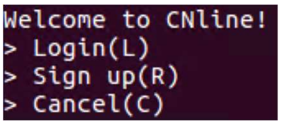
```
Sign up success!! Hello, aaa
What do you want to do?
> Check new message(C)
> Send new message(S)
> Logout(Q)
```

/*　Log in　*/

Step 1:　type ‘L’ or ‘l’ to log in

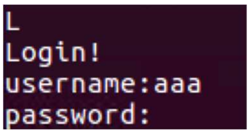
Step 2:　type your username and password

Step 3:　If sign up success, you can choose what to do next as well



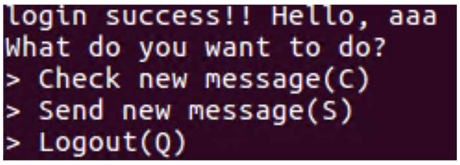
```
Welcome to CNline!
> Login(L)
> Sign up(R)
> Cancel(C)
```

➜



```
L
Login!
username:aaa
password:
```

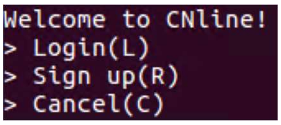
➜



```
login success!! Hello, aaa
What do you want to do?
> Check new message(C)
> Send new message(S)
> Logout(Q)
```

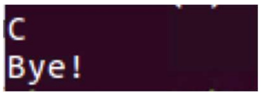
/* Cancel */

type ‘C’ or ‘c’, your CNline client session will close



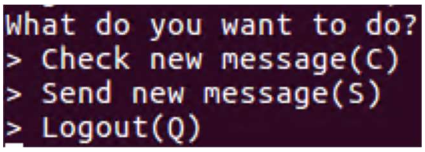
```
Welcome to CNline!
> Login(L)
> Sign up(R)
> Cancel(C)
```

→



```
C
Bye!
```

2. Home page (After Login)

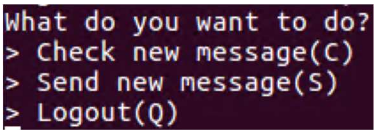


```
What do you want to do?
> Check new message(C)
> Send new message(S)
> Logout(Q)
```

/* Send new messages */

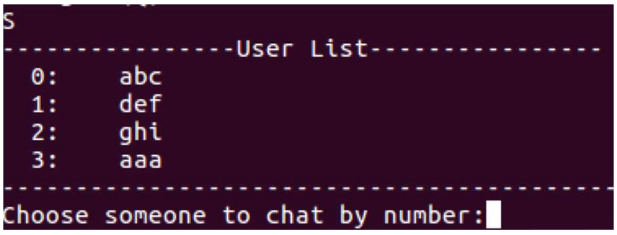
Step 1: type ‘S’ or ‘s’ to check the user list

Step 2: type target user number to open the chat window



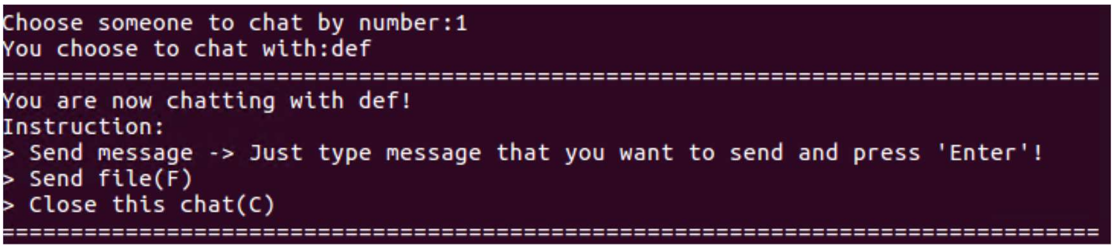
```
What do you want to do?
> Check new message(C)
> Send new message(S)
> Logout(Q)
```

→



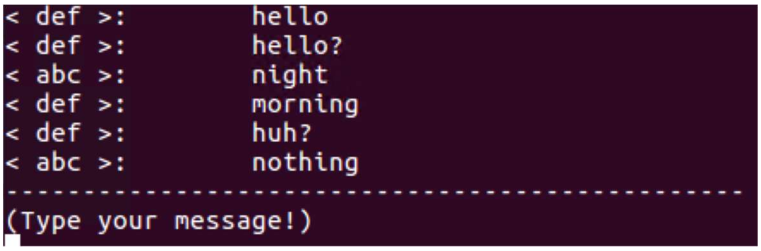
```
S
----------------User List----------------
  0:    abc
  1:    def
  2:    ghi
  3:    aaa
-----------------------------------------
Choose someone to chat by number:
```

then the chat window will open.



```
Choose someone to chat by number:1
You choose to chat with:def
==================================================================================
You are now chatting with def!
Instruction:
> Send message -> Just type message that you want to send and press 'Enter'!
> Send file(F)
> Close this chat(C)
==================================================================================
```

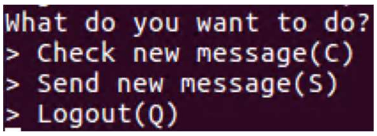
also, historical messages will be loaded.



```
< def >:        hello
< def >:        hello?
< abc >:        night
< def >:        morning
< def >:        huh?
< abc >:        nothing
----------------------------------------------
(Type your message!)
```

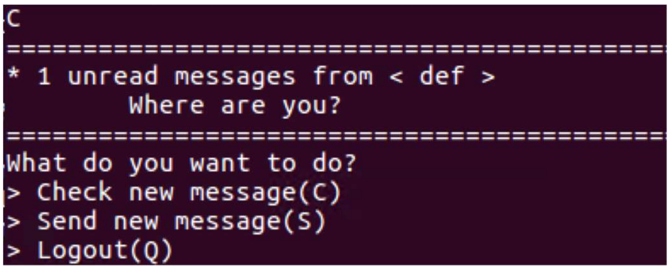
/* Check new messages */

type ‘C’ or ‘c’ to check unread messages,



```
What do you want to do?
> Check new message(C)
> Send new message(S)
> Logout(Q)
```

→

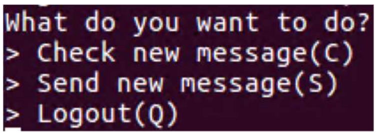


```
C
===========================================
* 1 unread messages from < def >
        Where are you?
===========================================
What do you want to do?
> Check new message(C)
> Send new message(S)
> Logout(Q)
```

then you can see messages sent to you when you were offline.

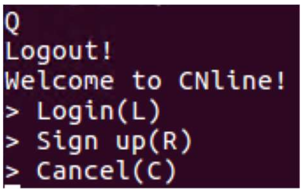
/*   Log out   */

type ‘Q’ to log out, and then you’ll back to the Home page.



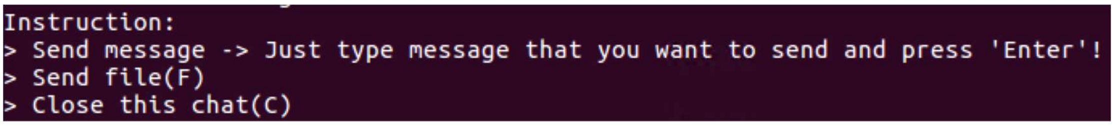
```
What do you want to do?
> Check new message(C)
> Send new message(S)
> Logout(Q)
```

→



```
Q
Logout!
Welcome to CNline!
> Login(L)
> Sign up(R)
> Cancel(C)
```

3. Chat Window

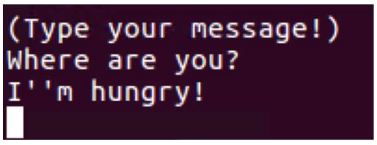


```
Instruction:
> Send message -> Just type message that you want to send and press 'Enter'!
> Send file(F)
> Close this chat(C)
```

If you choose to send new messages at Home Page (After Login), you’ll enter the chat window with the selected target user.

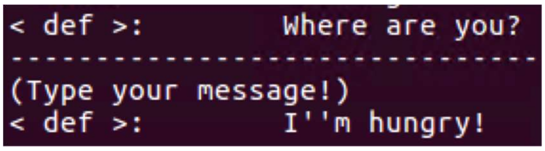
/*   Send message */

Simply type your message and press enter.



```
(Type your message!)
Where are you?
I''m hungry!
```

If the receiver is online, he or she will receive the message immediately.

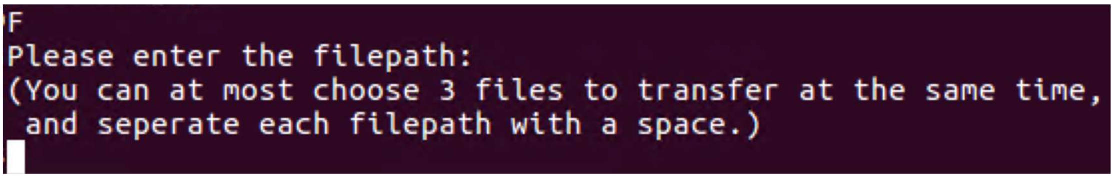


```
< def >:        Where are you?
-------------------------------
(Type your message!)
< def >:        I''m hungry!
```

(P.S. “Where are you” was sent when the receiver is offline. Therefore, this message was seen via “check new messages” when the receiver got online afterwards.)

/*   Send File */

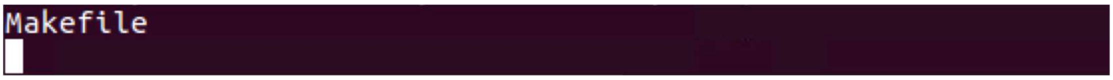
Step 1:   type ‘F’ to send file



```
F
Please enter the filepath:
(You can at most choose 3 files to transfer at the same time,
 and seperate each filepath with a space.)
```

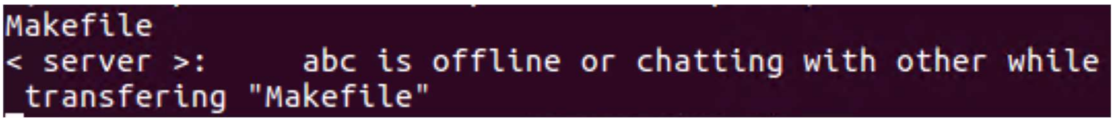
Step 2: Type filepath.

(You may transfer at most 3 files. Separate their filepath with space.)



```
Makefile
```

For example, if you want to send the file “Makefile” in your local client folder, simply type the filename, and this file will be sent to the receiver’s client folder.

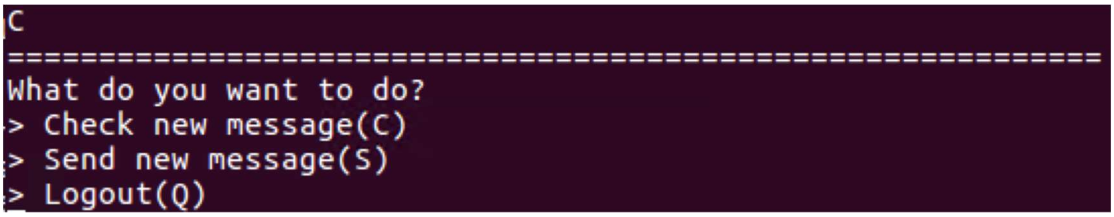


```
Makefile
< server >:      abc is offline or chatting with other while
 transfering "Makefile"
```

File transfer will fail if the receiver is offline!

/* Close this chat */

type ‘C’, this chat will close, and then you’ll back to the Home page (After Login).



```
C
============================================================
What do you want to do?
> Check new message(C)
> Send new message(S)
> Logout(Q)
```

System and program design

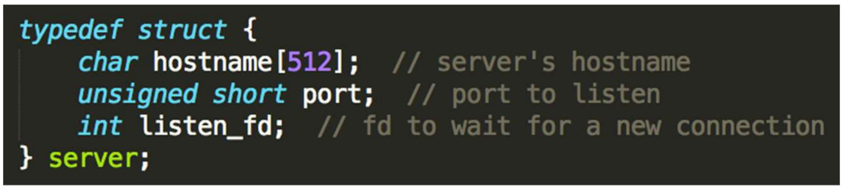
- Both server.c and client.c defines PKT_BUFSIZE 65536

1. server.c

- predefined structure:

typedef struct server

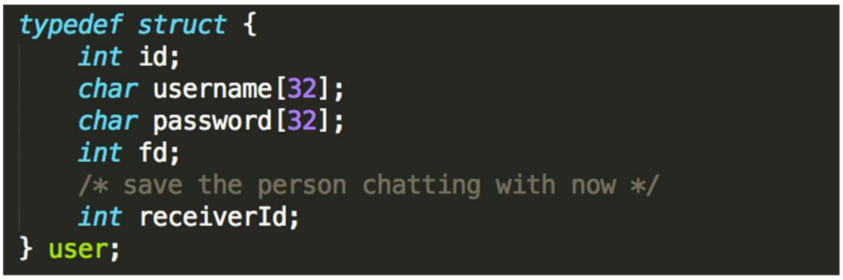
server structure records server’s hostname. port and socket file descriptor.



```
typedef struct {
    char hostname[512];  // server's hostname
    unsigned short port;  // port to listen
    int listen_fd;  // fd to wait for a new connection
} server;
```

typedef struct user

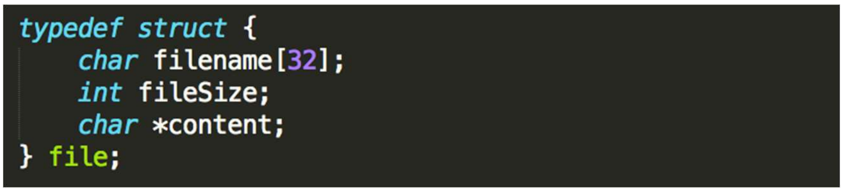
user structure records information of a specific user, including his or her id on userList, username, password, socket fd and receiver’s id on user list.



```
typedef struct {
    int id;
    char username[32];
    char password[32];
    int fd;
    /* save the person chatting with now */
    int receiverId;
} user;
```

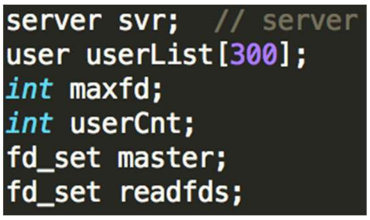
typedef struct file

file structure will be created during file transfer. After the server receives files to send from a client, it will create a *user* structure to formulate messages transferred to another client.



```
typedef struct {
    char filename[32];
    int fileSize;
    char *content;
} file;
```

- global variables:



```
server svr;   // server
user userList[300];
int maxfd;
int userCnt;
fd_set master;
fd_set readfds;
```

- *svr* will be set in ***init_server()*** in main()
- *userList[]* will be set afterwards in ***init_user()***. During a server session, a new user will be added to userList after registration success.
- *maxfd* is retrieved via ***getdtablesize()***. It is an argument in select() in main().
- *userCnt* is set in ***init_server()***. This variable helps us find sender or receiver information with for loops.
- *master* and *readfds* act as arguments in select()

- **int main()**

init_server()

↓

set fds

↓

main loop:

select()

↓

accept()

↓

recv()

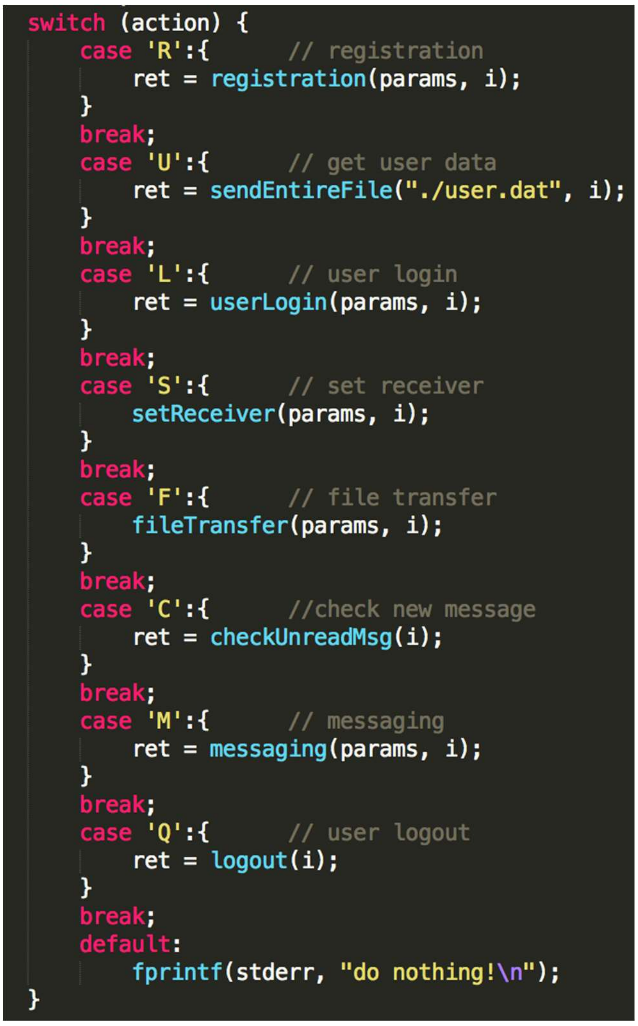
↓

switch actions

↓

pass remained message parameters to other functions.

switch actions:



```
switch (action) {
    case 'R':{      // registration
        ret = registration(params, i);
    }
    break;
    case 'U':{      // get user data
        ret = sendEntireFile("./user.dat", i);
    }
    break;
    case 'L':{      // user login
        ret = userLogin(params, i);
    }
    break;
    case 'S':{      // set receiver
        setReceiver(params, i);
    }
    break;
    case 'F':{      // file transfer
        fileTransfer(params, i);
    }
    break;
    case 'C':{      //check new message
        ret = checkUnreadMsg(i);
    }
    break;
    case 'M':{      // messaging
        ret = messaging(params, i);
    }
    break;
    case 'Q':{      // user logout
        ret = logout(i);
    }
    break;
    default:
        fprintf(stderr, "do nothing!\n");
}
```

- ***static void init_server(unsigned short port)***

  This function build socket based on assigned port at the beginning of main().

  It calls *gethostname(), socket(), setsockopt(), bind(), listen()* in order.

  All information in *svr* will be set afterwards.

- ***void init_user()***

  This function creates or opens a data file "user.dat", and then load information of all users to *userList*.

At the beginning of client messages. We add an uppercase letter that symbolizes expected action. Therefore, after the server accept a new socket and then receive a message, it can direct to corresponding function.

Registration

- ***int registration (char *params, int sockfd);***

  When a client send available username and password, add new user to *userList* and return messages to validate this user.

- ***int userLogin (char *params, int sockfd);***

  Judge login behaviors according to client's input username and password and then return a number. A client will login successfully when it get number '2'.

- ***int logout (int sockfd);***

  Close the client socket and set the corresponding user fd to -1.

Create a chat window

- ***void setReceiver(char *params, int sockfd);***

  When one client first select a receiver, two history chat log files will be created. One is for client A as sender and client B as receiver. The other is for client B as sender and client A as receiver.

  Server will set *receiverfd* of this client, then print history chat records on the client's screen if the history file already exists.

- ***int sendEntireFile(char *filepath, int sockfd);***

  This function is called in setReceiver. It retrieves history chat log files between two known clients. Only if sendEntireFIle return false will new history file been created in ***setReceiver()***.

Messaging

- ***int messaging (char *message, int sockfd);***

  Server will find the corresponding history file according to sender and receiver information at first. Then, after the server receives messages sent from the sender client, it will attach '0' if the receiver is offline or '1' if the receiver is online to the message. Finally, the server write the attached message to history records, and send the original message to the receiver if it's online.

- ***int checkUnreadMsg(int sockfd);***

  When a client log in, he or she can choose to check unread messages before to choose someone to chat. Since unread messages are attached a '0' in his history chat log files, if someone sent him a message when he is offline, he can see the unread message afterwards.

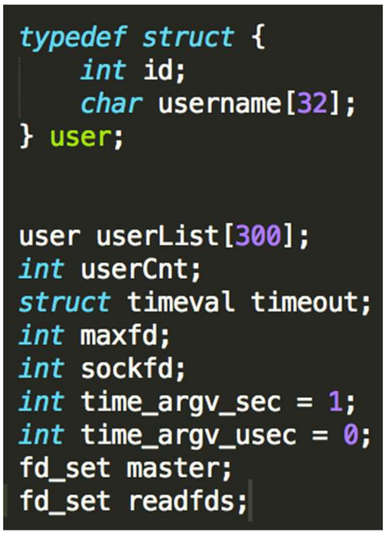
File Transfer

- ***void fileTransfer(char *params, int sockfd);***

  This function sets *file* information, such as filename, filesize and its content. The file will only transfer to target receiver successfully if he or she is online.

2. client.c

- predefined structure and global variables:



```
typedef struct {
    int id;
    char username[32];
} user;


user userList[300];
int userCnt;
struct timeval timeout;
int maxfd;
int sockfd;
int time_argv_sec = 1;
int time_argv_usec = 0;
fd_set master;
fd_set readfds;
```

- client.c also has a *userList*, because it will choose someone to chat by number.
- *sockfd* is the server's socket fd
- *master* and *readfds* act as arguments in select()

- ***main():***

getaddrinfo() ➔sockfd = socket() ➔Set the socket Non-blocking ➔connect() ➔

getsockopt() ➔select() ➔Turn the socket back to Blocking mode ➔main loop:

userLogin() ➔userReadOrSend()

Registration

- ***int userLogin (char *username, char *password, int sockfd)***

  This function acts as the main page. Users can choose to sign up (S), log in (L), or cancel this session. After a client type username and password, either username and password that becomes a message to send with either 'S' or 'L' attached to the beginning of message.

  As mentioned in server.c, if registration is validated by server, this client will recv specific return number and then allows main() to execute ***userReadOrSend()***.

Create a chat window

- ***void userReadOrSend(char *username, int sockfd);***

  This function acts as the userReadOrSend page. If this client chooses to check unread message, 'C' will be attached to the beginning of return message sent to server. Then server will call ***checkUnreadMsg()*** afterwards.

  If this client chooses to send message, ***userChooseTarget()*** executes to set receiver. 'S' will be attached to the beginning of return message sent to server. Then server will call ***setReceiver()*** that ensures connection between the two users and return history logs.

  Since a chat window will open, this function will call ***chooseToDo()***.

- ***void userChooseTarget(char *userListFile, char *receiver, int sockfd);***

  This function let a client choose a user to chat with. It sends server a message attached 'U', and then server will call *sendEntireFile()* and send back userList in "user.dat". Then user list will be loaded and show on the screen so that the client may choose the receiver number.

- ***int chooseToDo(char *sender, char *receiver, int sockfd, char *history);***

  This function acts as the chat window. At first, history chat logs will show on the client's screen. Also, via

  FD_SET(STDIN, &master);

  FD_SET(sockfd, &master);

  ,a client may listen to STDIN and sockfd to send and recv at the same time.

  Then a client enter the select() loop.

  Messaging

  If the fd is standard input, and a client send messages directly, 'M' will be attached to the beginning of message that causes server to call ***messaging()***

If the fd is server’s sockfd and recv messages has first char ‘M’, messages received will be printed on the screen.

File Transfer

If the fd is standard input, and a client press ‘F’ and then send files, he or she may start to transfer file. We use pthread to implement this function.

sendEntireFile()** is called in ***pthread_create()

If the fd is server’s sockfd and recv messages has first char ‘F’, ***recvEntireFile()*** will be called if the receiver is online.

- ***void *sendEntireFile(void *args);***

  This function is in thread function. Based on our default filesize and PKT_BUFSIZE, we copy the file into the buffer and sent it to server with ‘F’ attached to the beginning of messages.

- ***void recvEntireFile(char *params);***

  After the server calls ***fileTransfer()***, messages can be scanned to buffer. Then the client uses fwrite() that writes messages in buffer to a new file.