Introduction to Computer System

Lab Report

Wang, Chen

16307110064

December 28th, 2018

Contents

[I. General functionality of the application 3](#_Toc533791904)

[1. Overview 3](#_Toc533791905)

[2. Brief Usage Guide 3](#_Toc533791906)

[3. Runtime screenshot 4](#_Toc533791907)

[II. Implementation details 5](#_Toc533791908)

[1. Wrapper methods 5](#_Toc533791909)

[2. Server-side methods 6](#_Toc533791910)

[3. Client-side methods 8](#_Toc533791911)

# General functionality of the application

## Overview

This lab is based on the network chapter of the CS:APP 3e text book and we need to implement a chat room that can compile and run on Linux system using C language.

This version of implementation adopts C/S architecture and there are two executable objects can be compiled and executed, namely client and server. The server uses the localhost network loop to send and receive packages with the client. The port of the server is defined as 8000. When executing this application, you ought to execute the server application first, seeing the Prepared to get connected... words indicates that the server is ready for the clients to connect to.

Then you can start at most LISTENQ number of clients. Here this constant is defined as 1024, adhering to the convention of the CS:APP textbook. For the first user that want to log in, the password is 2019. Every time a new user comes to the chat room, he is given a randomly generated pass code with which this user can invite others into the chat room. A new user can only log in to the chat room with correct password provided. However, there is no restriction from whom the pass code is retrieved from. That is, all the generated pass codes have the same effect.

There are two modes in the chatting room. The first is private chat. Only the user with specified name can receive the message. If the user with the specified name is not online, sending private message will fail and an error message will appear. The second method is public chat. There is no need to input a receiver’s name and all the online users will be able to receive the message.

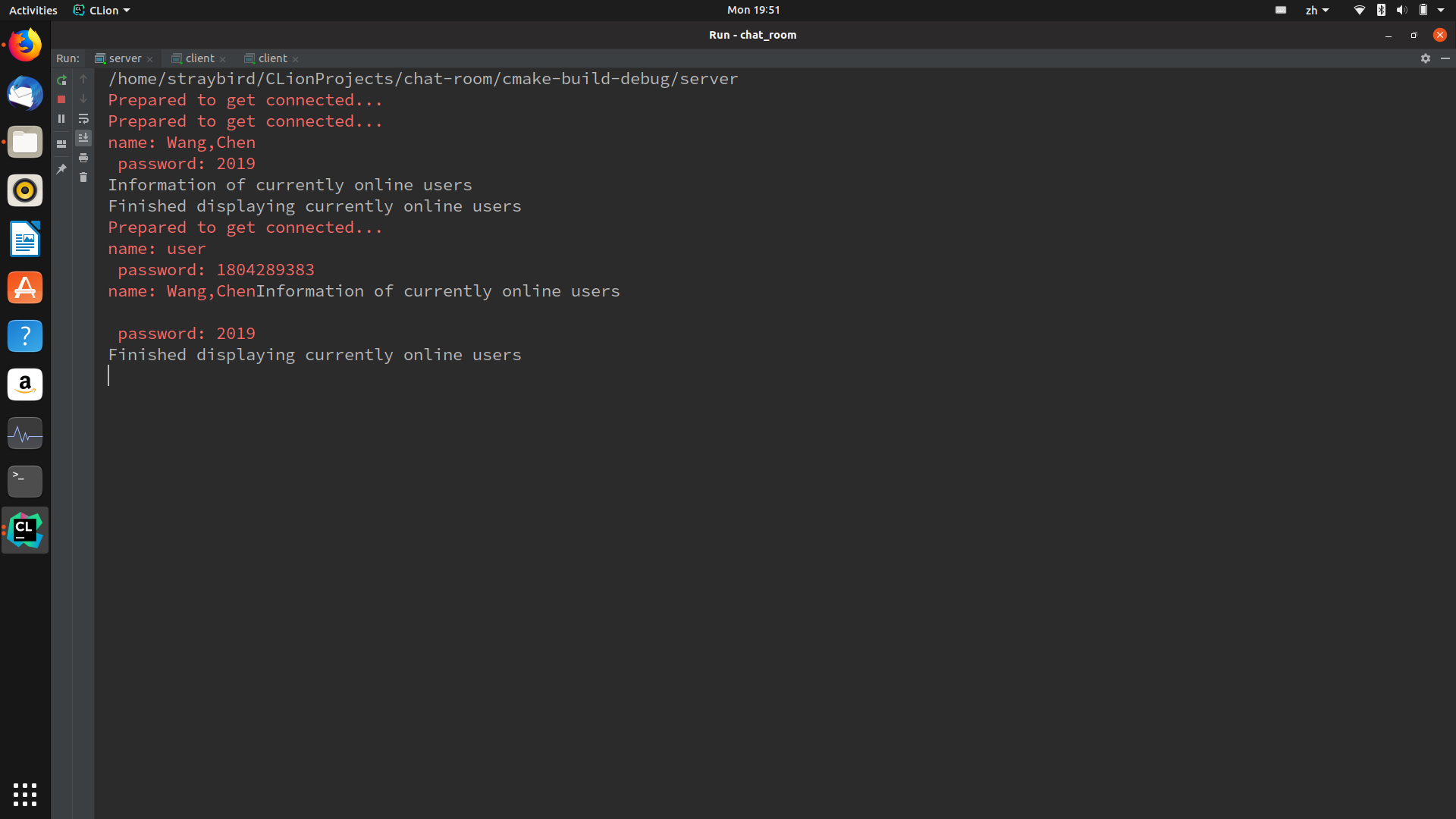
A client process disconnects with the server when it logs out. And the process will terminate if the user then chooses the exit option.

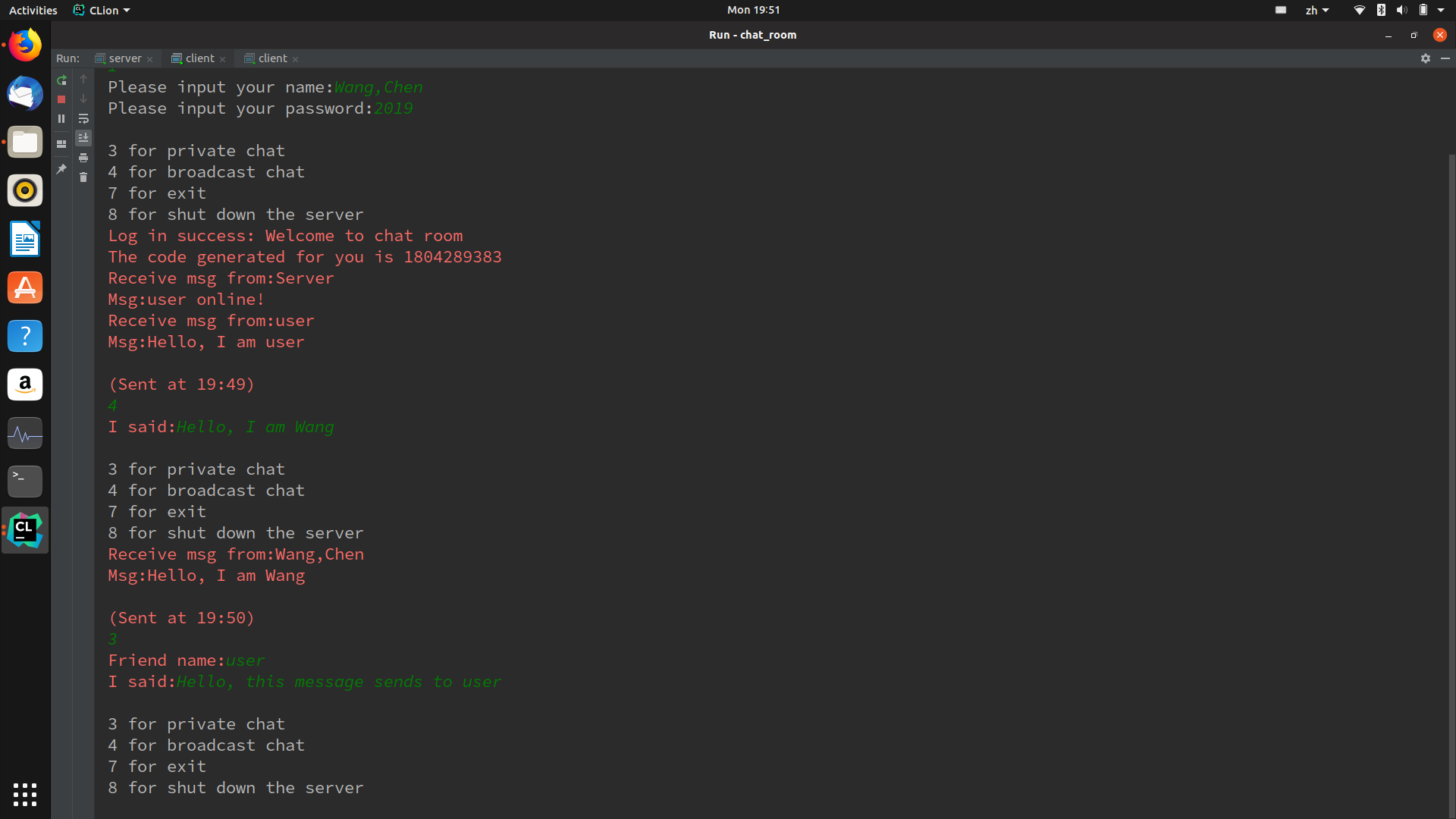
The server can be turned off from an administrator client. The username of the administrator is admin, whose password is either 2019 or any inviting code. All the client will be forced terminated when the server shuts.

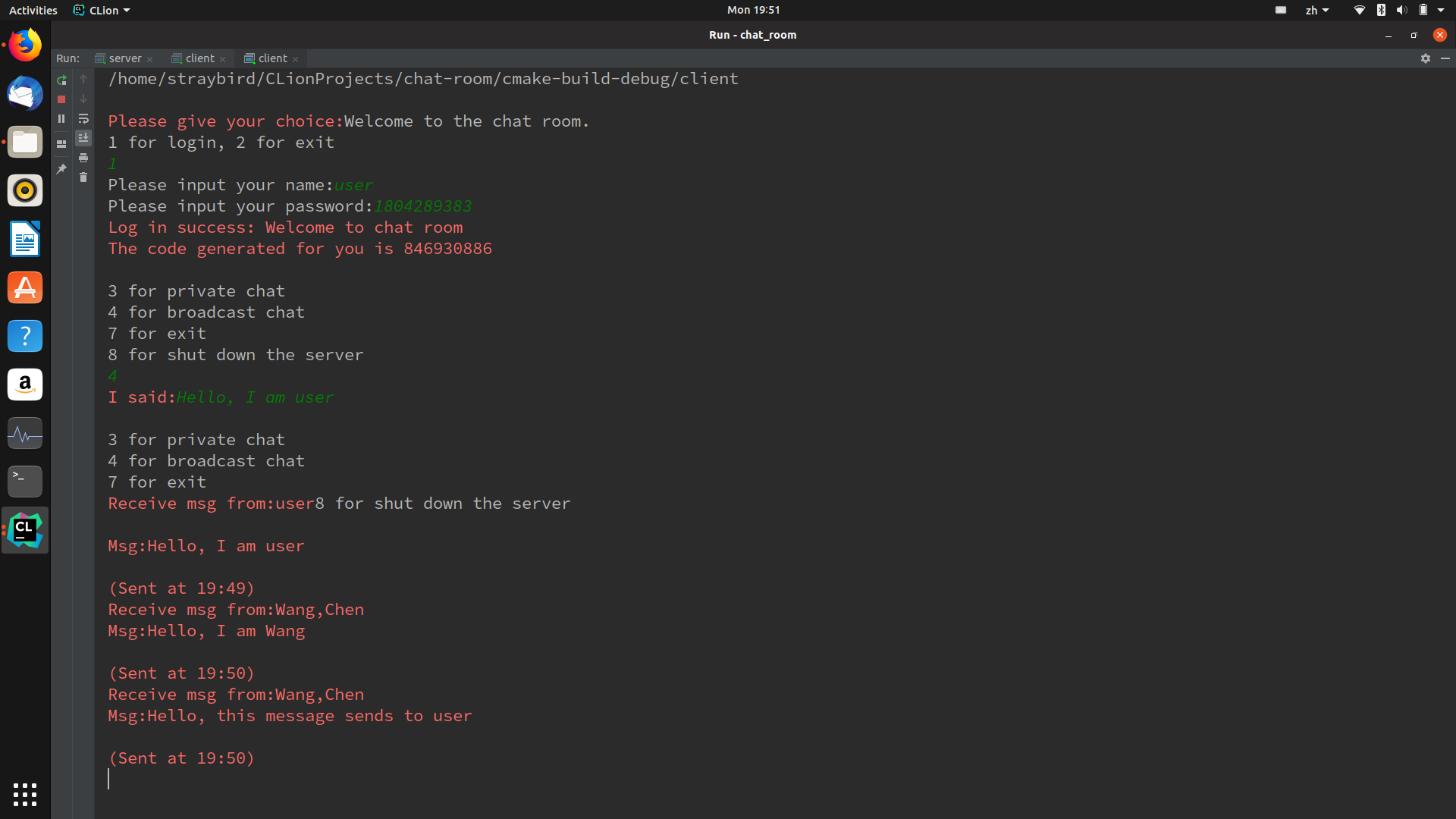
## Brief Usage Guide

* Either use the make file to execute “make server” and “make client” operation or import this project into JetBrains Clion and build with the pre-worked CMakeLists.txt;
* Make sure the port 8000 on your localhost is vacant;
* Run server;
* Run client;
* Log in the first account with any username and password 2019;
* (Selective) Log in other accounts with any username and dynamically generated inviting code from pervious logged users;
* Chat in either private or broadcast mode;
* Log in with super user account with name admin and password 2019;
* Select the shut down server option to terminate server and all the client processes.

## Runtime screenshot







# Implementation details

## Wrapper methods

The wrapper.c and wrapper.h contain wrapper methods for error handling, process control, I/O, socket methods, thread interfaces and the implementation of Robust I/O. This version is adapted from CS:APP 3e textbook and contains only the essential methods that are required in this chat room application.

|  |  |
| --- | --- |
|  |  |
|  | /\* Misc constants \*/ |
|  | #define MAXLINE 8192 /\* Max text line length \*/ |
|  | #define LISTENQ 1024 /\* Second argument to listen() \*/ |
|  |  |
|  | /\* My own error-handling functions \*/ |
|  | void unix\_error(char \*msg); |
|  | void posix\_error(int code, char \*msg); |
|  | void gai\_error(int code, char \*msg); |
|  | void app\_error(char \*msg); |
|  |  |
|  | /\* Process control wrappers \*/ |
|  | unsigned int Sleep(unsigned int secs); |
|  |  |
|  |  |
|  | /\* Unix I/O wrappers \*/ |
|  | void Close(int fd); |
|  |  |
|  |  |
|  | /\* Standard I/O wrappers \*/ |
|  | char \*Fgets(char \*ptr, int n, FILE \*stream); |
|  | void Fputs(const char \*ptr, FILE \*stream); |
|  |  |
|  |  |
|  | /\* Sockets interface wrappers \*/ |
|  | void Setsockopt(int s, int level, int optname, const void \*optval, int optlen); |
|  | int Accept(int s, struct sockaddr \*addr, socklen\_t \*addrlen); |
|  |  |
|  |  |
|  | /\* Protocol independent wrappers \*/ |
|  | void Getaddrinfo(const char \*node, const char \*service, |
|  | const struct addrinfo \*hints, struct addrinfo \*\*res); |
|  | void Freeaddrinfo(struct addrinfo \*res); |
|  |  |
|  |  |
|  | /\* Pthreads thread control wrappers \*/ |
|  | void Pthread\_create(pthread\_t \*tidp, pthread\_attr\_t \*attrp, |
|  | void \*(\*routine)(void \*), void \*argp); |
|  | void Pthread\_join(pthread\_t tid, void \*\*thread\_return); |
|  | void Pthread\_exit(void \*retval); |
|  |  |
|  |  |
|  | /\* Rio (Robust I/O) package \*/ |
|  | ssize\_t rio\_readn(int fd, void \*usrbuf, size\_t n); |
|  | ssize\_t rio\_writen(int fd, void \*usrbuf, size\_t n); |
|  |  |
|  |  |
|  | /\* Wrappers for Rio package \*/ |
|  | ssize\_t Rio\_readn(int fd, void \*usrbuf, size\_t n); |
|  | void Rio\_writen(int fd, void \*usrbuf, size\_t n); |
|  |  |
|  | /\* Reentrant protocol-independent client/server helpers \*/ |
|  | int open\_clientfd(char \*hostname, char \*port); |
|  | int open\_listenfd(char \*port); |
|  |  |
|  | /\* Wrappers for reentrant protocol-independent client/server helpers \*/ |
|  | int Open\_clientfd(char \*hostname, char \*port); |
|  | int Open\_listenfd(char \*port); |
|  |  |

## Server-side methods

|  |  |
| --- | --- |
|  |  |
|  | #include "server.h" |
|  | /\*\* The list of the structures of the currently online users \*/ |
|  | struct list\_node \*head = 0; |
|  |  |
|  | /\*\* A list of dynamically generated pass codes \*/ |
|  | long int codes[LISTENQ]; |
|  |  |
|  | /\*\* A lock for making log in operation thread safe \*/ |
|  | pthread\_mutex\_t log\_lock = PTHREAD\_MUTEX\_INITIALIZER; |
|  |  |
|  |  |
|  | /\*\* This function is an argument of the Pthread\_create method and |
|  | \* will be executed when a new thread is created. |
|  | \* When the client quits, it disconnects with the server and the |
|  | \* corresponding server thread exits.\*/ |
|  | void manage\_client(void \*arg) ; |
|  |  |
|  | /\*\* The main routine of the option-choosing interface, the loop will |
|  | \* not be broken until the client wants to quit.\*/ |
|  | int get\_choice(int fd) ; |
|  |  |
|  | /\*\* The event handler for the log in option that handles |
|  | \* 1) Comparing the name of the current logged users with |
|  | \* the user that wants to log in to make sure that there |
|  | \* won't be two users with the same name online at the same |
|  | \* time; |
|  | \* 2) Compare the password with the current password list to |
|  | \* make sure that the user enters a valid password. |
|  | \* 3) Dynamically generate a new password for the newly logged |
|  | \* user so that s/he can use this password to invite other |
|  | \* users into this chat room. |
|  | \* After handling the events above, it will transfer the control |
|  | \* to the manage chat function that handles massage sending and |
|  | \* receiving actions. |
|  | \* \*/ |
|  | int log\_in(int fd, struct account \*temp\_user) { |
|  |  |
|  | /\*\* Send a message to all the online users telling them |
|  | \* someone is online or offline.\*/ |
|  | int notice(char \*name, int choice) ; |
|  |  |
|  | /\*\* When a new user logs, the node of this user is created. \*/ |
|  | struct list\_node \*create\_node(struct account \*user\_msg, int conn\_fd) ; |
|  |  |
|  | /\*\* When a new user logs, the node of this user is inserted into the list. \*/ |
|  | void insert\_node(struct list\_node \*load\_user) ; |
|  |  |
|  | /\*\* Displays the info of currently online users on the server side. \*/ |
|  | int traverse(struct list\_node \*head) ; |
|  |  |
|  | /\*\* Handles the specified requests. \*/ |
|  | int manage\_chat(int connection\_fd) ; |
|  |  |
|  | /\*\* Checks if the user with the specified name is online. \*/ |
|  | int check\_online(char \*name) ; |
|  |  |
|  | /\*\* Find the user with the specified name if s/he is online. \*/ |
|  | struct list\_node \*find\_online(char \*user\_name) ; |
|  |  |
|  | /\*\* |
|  | \* Removes the node with the user information from the list |
|  | \* when the user is offline. \*/ |
|  | int remove\_node(struct list\_node \*user\_node) ; |
|  |  |
|  | /\*\* Handles the request of shutting down server. \*/ |
|  | int shut\_server() ; |
|  |  |
|  | /\*\* Handles the request of private chat. \*/ |
|  | int my\_private(int conn\_fd, struct chat \*temp) ; |
|  |  |
|  | /\*\* Handles the request of broadcast. \*/ |
|  | int broadcast(struct chat \*tran\_msg) ; |

## Client-side methods

|  |  |
| --- | --- |
|  |  |
|  | #include "client.h" |
|  | int sock\_fd = 0; |
|  | char my\_name[20]; |
|  |  |
|  | /\*\* |
|  | \* The main routine of the client side that starts listening a port, |
|  | \* displays the welcome and select option information and sends and |
|  | \* receives the package of option to select and the response from the |
|  | \* server. |
|  | \* This routine of this loop will not exit until the user selects the |
|  | \* quit option. |
|  | \* \*/ |
|  | int main() { |
|  |  |
|  | /\*\* The control is transferred to this method if the user successfully logs in.\*/ |
|  | int manage\_chat() { |
|  |  |
|  | /\*\* |
|  | \* When the user logs in, a new thread is created and the new thread will |
|  | \* enter this method that keeps receiving packages from the server side |
|  | \* displaying information of the message from others and the server. |
|  | \* \*/ |
|  | void \*read\_serv(void \*arg) { |
|  |  |
|  | /\*\* The handler method of private chat. \*/ |
|  | int my\_private(struct chat \*temp) { |
|  |  |
|  | /\*\* The handler method of broadcast. \*/ |
|  | int broadcast(struct chat \*tran\_msg) { |