/\*

\*

\* Chatroom - a simple linux commandline client/server C program for group chat.

\* Author: Andrew Herriot

\* License: Public Domain

\*

\*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <stdbool.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include "chatroom\_utils.h"

// get a username from the user.

void get\_username(char \*username){

while(true){

/\*name\*/

printf("[Welcome!]\n");

printf("\tPlease enter your name:");

fflush(stdout);

memset(username, 0, 1000);

fgets(username, 22, stdin);

trim\_newline(username);

if(strlen(username) > 20){

puts("Username must be 20 characters or less.");

}else{

break;

}

}

}

// get a roomname and rules from the user.

void get\_roomname(char \*roomname){

while(true){

/\*Use rules and room\*/

printf("---------------------------------------\n");

printf("\tPlease enter group name (if doesn't exist,then create):");

fflush(stdout);

memset(roomname, 0, 1000);

fgets(roomname, 22, stdin);

trim\_newline(roomname);

printf("\n\n");

printf("Usage: <Message>\n");

printf("\t/w <Name or room> <Message>\n");

printf("\tBye\n");

printf("---------------------------------------\n");

if(strlen(roomname) > 20){

puts("Username must be 20 characters or less.");

}else{

break;

}

}

}

//send local username to the server.

void set\_information(connection\_info \*connection){

message msg;

msg.type = SET\_INFORMATION;

strncpy(msg.username, connection->username, 20);

strncpy(msg.roomname, connection->roomname, 20);

if(send(connection->socket, (void\*)&msg, sizeof(msg), 0) < 0){

perror("Send failed");

exit(1);

}

}

// get a memberinfo from the user.

void listmember(connection\_info \*connection){

/\*member\*/

printf("[Member List]\n");

message msg;

msg.type = GET\_USERS;

if(send(connection->socket, &msg, sizeof(message), 0) < 0){

perror("Send failed");

exit(1);

}

/\*

ssize\_t recv\_val = recv(connection->socket, &msg, sizeof(message), 0);

if(recv\_val < 0){

perror("recv failed");

exit(1);

}else if(msg.type ==GET\_USERS){

printf("\*\*\*%s", msg.data);

}

\*/

}

void stop\_client(connection\_info \*connection){

close(connection->socket);

exit(0);

}

//initialize connection to the server.

void connect\_to\_server(connection\_info \*connection, char \*address, char \*port){

while(true){

//Create socket

if ((connection->socket = socket(AF\_INET, SOCK\_STREAM , IPPROTO\_TCP)) < 0){

perror("Could not create socket");

}

connection->address.sin\_addr.s\_addr = inet\_addr(address);

connection->address.sin\_family = AF\_INET;

connection->address.sin\_port = htons(atoi(port));

//Connect to remote server

if (connect(connection->socket, (struct sockaddr \*)&connection->address , sizeof(connection->address)) < 0){

perror("Connect failed.");

exit(1);

}

get\_username(connection->username);

get\_roomname(connection->roomname);

set\_information(connection);

listmember(connection);

message msg;

ssize\_t recv\_val = recv(connection->socket, &msg, sizeof(message), 0);

if(recv\_val < 0){

perror("recv failed");

exit(1);

}else if(recv\_val == 0){ //recv回傳0表示網路中斷

close(connection->socket);

printf("The username \"%s\" is taken, please try another name.\n", connection->username);

continue;

}

break;

}

//puts("Connected to server.");

//puts("Type /help for usage.");

}

void handle\_user\_input(connection\_info \*connection){

char input[255];

fgets(input, 255, stdin);

trim\_newline(input);

if(strcmp(input, "/b") == 0 || strcmp(input, "/bye") == 0 || strcmp(input, "bye") == 0){

stop\_client(connection);

}else if(strcmp(input, "/l") == 0 || strcmp(input, "/list") == 0){

message msg;

msg.type = GET\_USERS;

if(send(connection->socket, &msg, sizeof(message), 0) < 0){

perror("Send failed");

exit(1);

}

}else if(strcmp(input, "/h") == 0 || strcmp(input, "/help") == 0){

puts("bye or /b: Exit the program.");

puts("/help or /h: Displays help information.");

puts("/list or /l: Displays list of users in chatroom.");

puts("/w <username> <message> Send a private message to given username.");

}else if(strncmp(input, "/w", 2) == 0){

message msg;

msg.type = PRIVATE\_MESSAGE;

char \*toUser, \*chatMsg;

toUser = strtok(input+3, " ");

if(toUser == NULL){

puts(KRED "The format for private messages is: /m <username> <message>" RESET);

return;

}

if(strlen(toUser) == 0){

puts(KRED "You must enter a username for a private message." RESET);

return;

}

if(strlen(toUser) > 20){

puts(KRED "The username must be between 1 and 20 characters." RESET);

return;

}

chatMsg = strtok(NULL, "");

if(chatMsg == NULL){

puts(KRED "You must enter a message to send to the specified user." RESET);

return;

}

//printf("|%s|%s|\n", toUser, chatMsg);

strncpy(msg.username, toUser, 20);

strncpy(msg.data, chatMsg, 255);

if(send(connection->socket, &msg, sizeof(message), 0) < 0){

perror("Send failed");

exit(1);

}

}else{//regular public message

message msg;

msg.type = PUBLIC\_MESSAGE;

strncpy(msg.username, connection->username, 20);

strncpy(msg.roomname, connection->roomname, 20);

if(strlen(input) == 0) {

return;

}

strncpy(msg.data, input, 255);

//Send some data

if(send(connection->socket, &msg, sizeof(message), 0) < 0){

perror("Send failed");

exit(1);

}

}

}

void handle\_server\_message(connection\_info \*connection){

message msg;

//Receive a reply from the server

ssize\_t recv\_val = recv(connection->socket, &msg, sizeof(message), 0);

if(recv\_val < 0){

perror("recv failed");

exit(1);

}else if(recv\_val == 0){

close(connection->socket);

puts("Server disconnected.");

exit(0);

}

switch(msg.type){

case CONNECT:

printf(KYEL "[note: client [%s] log in room [%s].]" RESET "\n", msg.username,msg.roomname);

break;

case DISCONNECT:

printf(KYEL "[note: client [%s] leave.]" RESET "\n" , msg.username);

break;

case GET\_USERS:

printf("%s", msg.data);

break;

case PUBLIC\_MESSAGE:

printf(KGRN "%s" RESET ": %s\n", msg.username, msg.data);

break;

case PRIVATE\_MESSAGE:

printf(KWHT "private form %s:%s\n" RESET, msg.username, msg.data);

break;

case TOO\_FULL:

fprintf(stderr, KRED "Server chatroom is too full to accept new clients." RESET "\n");

exit(0);

break;

case USER\_ERROR:

fprintf(stderr, KRED "USER\_ERROR message type received." RESET "\n");

break;

default:

fprintf(stderr, KRED "Unknown message type received." RESET "\n");

break;

}

}

int main(int argc, char \*argv[]){

connection\_info connection;

fd\_set file\_descriptors;

if (argc != 3) {

fprintf(stderr,"Usage: %s <IP> <port>\n", argv[0]);

exit(1);

}

connect\_to\_server(&connection, argv[1], argv[2]);

//keep communicating with server

while(true){

FD\_ZERO(&file\_descriptors);

FD\_SET(STDIN\_FILENO, &file\_descriptors);

FD\_SET(connection.socket, &file\_descriptors);

fflush(stdin);

if(select(connection.socket+1, &file\_descriptors, NULL, NULL, NULL) < 0){

perror("Select failed.");

exit(1);

}

if(FD\_ISSET(STDIN\_FILENO, &file\_descriptors)){

handle\_user\_input(&connection);

}

if(FD\_ISSET(connection.socket, &file\_descriptors)){

//printf("\n%s > ",connection.username);

handle\_server\_message(&connection);

}

}

close(connection.socket);

return 0;

}