/\*

\*

\* 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 <errno.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <pthread.h>

#include "chatroom\_utils.h"

#define MAX\_CLIENTS 4

void initialize\_server(connection\_info \*server\_info, int port){

if((server\_info->socket = socket(AF\_INET, SOCK\_STREAM, 0)) < 0){

perror("Failed to create socket");

exit(1);

}

server\_info->address.sin\_family = AF\_INET;

server\_info->address.sin\_addr.s\_addr = INADDR\_ANY;

server\_info->address.sin\_port = htons(port);

if(bind(server\_info->socket, (struct sockaddr \*)&server\_info->address, sizeof(server\_info->address)) < 0){

perror("Binding failed");

exit(1);

}

const int optVal = 1;

const socklen\_t optLen = sizeof(optVal);

if(setsockopt(server\_info->socket, SOL\_SOCKET, SO\_REUSEADDR, (void\*) &optVal, optLen) < 0){

perror("Set socket option failed");//SOL\_SOCKET: 基本套接口 //SO\_REUSEADDR 共享port

exit(1);

}

if(listen(server\_info->socket, 3) < 0) {

perror("Listen failed");

exit(1);

}

//Accept and incoming connection

//printf("Waiting for incoming connections...\n");

}

void send\_public\_message(connection\_info clients[], int sender, char \*message\_text, char \*roomname){

message msg;

msg.type = PUBLIC\_MESSAGE;

strncpy(msg.username, clients[sender].username, 20);

strncpy(msg.roomname, clients[sender].roomname, 20);

strncpy(msg.data, message\_text, 256);

int i = 0;

for(i = 0; i < MAX\_CLIENTS; i++){

// i != sender /\*do not Send to yourself\*/

if(/\*i != sender &&\*/strcmp(clients[i].roomname, roomname) == 0 && clients[i].socket != 0 ){

if(send(clients[i].socket, &msg, sizeof(msg), 0) < 0){

perror("Send failed");

exit(1);

}

}

}

}

void send\_private\_message(connection\_info clients[], int sender,char \*user, char \*message\_text){

message msg;

msg.type = PRIVATE\_MESSAGE;

strncpy(msg.username, clients[sender].username, 20);

strncpy(msg.roomname, clients[sender].roomname, 20);

strncpy(msg.data, message\_text, 256);

int i;

for(i = 0; i < MAX\_CLIENTS; i++){

if(clients[i].socket != 0 && (strcmp(clients[i].username, user) == 0 || strcmp(clients[i].roomname, user) == 0)){

if(send(clients[i].socket, &msg, sizeof(msg), 0) < 0){

perror("Send failed");

exit(1);

}

return;

}

}

msg.type = USER\_ERROR;

sprintf(msg.data, "Username \"%s\" does not exist or is not logged in.", user);

if(send(clients[sender].socket, &msg, sizeof(msg), 0) < 0){

perror("Send failed");

exit(1);

}

}

void send\_connect\_message(connection\_info \*clients, int sender){

message msg;

msg.type = CONNECT;

strncpy(msg.username, clients[sender].username, 21);

strncpy(msg.roomname, clients[sender].roomname, 21);

int i = 0;

for(i = 0; i < MAX\_CLIENTS; i++){

if(clients[i].socket != 0){

if(i == sender){

msg.type = SUCCESS;

if(send(clients[i].socket, &msg, sizeof(msg), 0) < 0){

perror("Send failed");

exit(1);

}

}else{

if(send(clients[i].socket, &msg, sizeof(msg), 0) < 0){

perror("Send failed");

exit(1);

}

}

}

}

}

void send\_disconnect\_message(connection\_info \*clients, char \*username){

message msg;

msg.type = DISCONNECT;

strncpy(msg.username, username, 21);

int i = 0;

for(i = 0; i < MAX\_CLIENTS; i++){

if(clients[i].socket != 0){

if(send(clients[i].socket, &msg, sizeof(msg), 0) < 0){

perror("Send failed");

exit(1);

}

}

}

}

void send\_user\_list(connection\_info \*clients, int receiver) {

message msg;

msg.type = GET\_USERS;

char \*list = msg.data;

int i;

for(i = 0; i < MAX\_CLIENTS; i++){

if(clients[i].socket != 0){

list = stpcpy(list, clients[i].username);

list = stpcpy(list, "\t");

list = stpcpy(list, clients[i].roomname);

list = stpcpy(list, "\n");

}

}

if(send(clients[receiver].socket, &msg, sizeof(msg), 0) < 0){

perror("Send failed");

exit(1);

}

}

void send\_too\_full\_message(int socket){

message too\_full\_message;

too\_full\_message.type = TOO\_FULL;

if(send(socket, &too\_full\_message, sizeof(too\_full\_message), 0) < 0){

perror("Send failed");

exit(1);

}

close(socket);

}

//close all the sockets before exiting

void stop\_server(connection\_info connection[]){

int i;

for(i = 0; i < MAX\_CLIENTS; i++){

close(connection[i].socket);

}

exit(0);

}

void handle\_client\_message(connection\_info clients[], int sender){

int read\_size;

message msg;

if((read\_size = recv(clients[sender].socket, &msg, sizeof(message), 0)) == 0){

printf("client [%s] leave.\n", clients[sender].username);

close(clients[sender].socket);

clients[sender].socket = 0;

send\_disconnect\_message(clients, clients[sender].username);

}else{

switch(msg.type){

case GET\_USERS:

send\_user\_list(clients, sender);

break;

case SET\_INFORMATION: ;

int i;

for(i = 0; i < MAX\_CLIENTS; i++){

if(clients[i].socket != 0 && strcmp(clients[i].username, msg.username) == 0){

close(clients[sender].socket);

clients[sender].socket = 0;

return;

}

}

strcpy(clients[sender].username, msg.username);

strcpy(clients[sender].roomname, msg.roomname);

printf("client [%s] log in room [%s]\n", clients[sender].username ,clients[sender].roomname);

send\_connect\_message(clients, sender);

break;

case PUBLIC\_MESSAGE:

send\_public\_message(clients, sender, msg.data, msg.roomname);

break;

case PRIVATE\_MESSAGE:

send\_private\_message(clients, sender, msg.username, msg.data);

break;

default:

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

break;

}

}

}

int construct\_fd\_set(fd\_set \*set, connection\_info \*server\_info,connection\_info clients[]){

FD\_ZERO(set);

FD\_SET(STDIN\_FILENO, set);

FD\_SET(server\_info->socket, set);

int max\_fd = server\_info->socket;

int i;

for(i = 0; i < MAX\_CLIENTS; i++){

if(clients[i].socket > 0){

FD\_SET(clients[i].socket, set);

if(clients[i].socket > max\_fd){

max\_fd = clients[i].socket;

}

}

}

return max\_fd;

}

void handle\_new\_connection(connection\_info \*server\_info, connection\_info clients[]){

int new\_socket;

int address\_len;

new\_socket = accept(server\_info->socket, (struct sockaddr\*)&server\_info->address, (socklen\_t\*)&address\_len);

if (new\_socket < 0){

perror("Accept Failed");

exit(1);

}

int i;

for(i = 0; i < MAX\_CLIENTS; i++){

if(clients[i].socket == 0) {

clients[i].socket = new\_socket;

break;

} else if (i == MAX\_CLIENTS -1){// if we can accept no more clients

send\_too\_full\_message(new\_socket);

}

}

}

void handle\_user\_input(connection\_info clients[]){

char input[255];

fgets(input, sizeof(input), stdin);

trim\_newline(input);

if(input[0] == 'q'){

stop\_server(clients);

}

}

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

//puts("Starting server.");

fd\_set file\_descriptors;

connection\_info server\_info;

connection\_info clients[MAX\_CLIENTS];

int i;

for(i = 0; i < MAX\_CLIENTS; i++){

clients[i].socket = 0;

}

if (argc != 2){

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

exit(1);

}

initialize\_server(&server\_info, atoi(argv[1]));

while(true){

int max\_fd = construct\_fd\_set(&file\_descriptors, &server\_info, clients);

if(select(max\_fd+1, &file\_descriptors, NULL, NULL, NULL) < 0){

perror("Select Failed");

stop\_server(clients);

}

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

handle\_user\_input(clients);

}

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

handle\_new\_connection(&server\_info, clients);

}

for(i = 0; i < MAX\_CLIENTS; i++){

if(clients[i].socket > 0 && FD\_ISSET(clients[i].socket, &file\_descriptors)){

handle\_client\_message(clients, i);

}

}

}

return 0;

}