实验八 进程间通信——Socket（2）

**一、实验目的**

1、了解采用Socket通信的原理。

2、掌握Socket的创建及使用方法。

**二、实验原理**

1、通过Socket进行进程间通信的流程：服务器端先初始化Socket，然后与端口绑定(bind)，对端口进行监听(listen)，调用accept阻塞，等待客户端连接。在这时如果有客户端初始化一个Socket，然后连接服务器(connect)，若连接成功，则客户端与服务器端的连接就建立了。客户端发送数据请求，服务器端接收请求并处理请求，然后把回应数据发送给客户端，客户端读取数据，最后关闭连接，一次交互结束。

2、TCP服务器端依次调用socket()、bind()、listen()之后，就会监听指定的socket地址。TCP客户端依次调用socket()、connect()后将向TCP服务器发送一个连接请求。TCP服务器监听到这个请求后，调用accept()函数接收请求，连接建立成功。之后可以开始网络I/O操作，类同于管道的阻塞读写I/O操作。

三、**实验内容**

1、创建一个服务器端和若干个客户端。

服务器端可实现包括：接收并区分来自客户端的数据，将用户输入的内容在服务器上输出，并将原内容群发至所有在线客户端（类似qq群聊形式）；服务器可主动向在线客户端发送数据；并可以统计在线人数等。

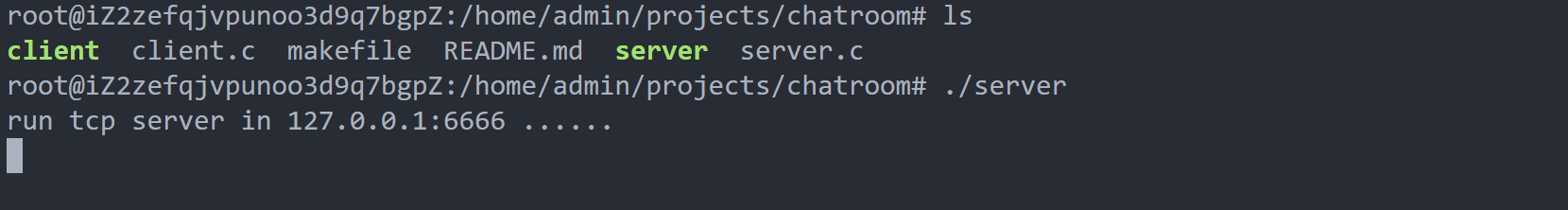
客户端可实现包括：输入文字并且向服务器发送消息，接收来自服务器端的数据；用户控制客户端退出。

2、在服务器端和客户端基于socket实现其通信过程。

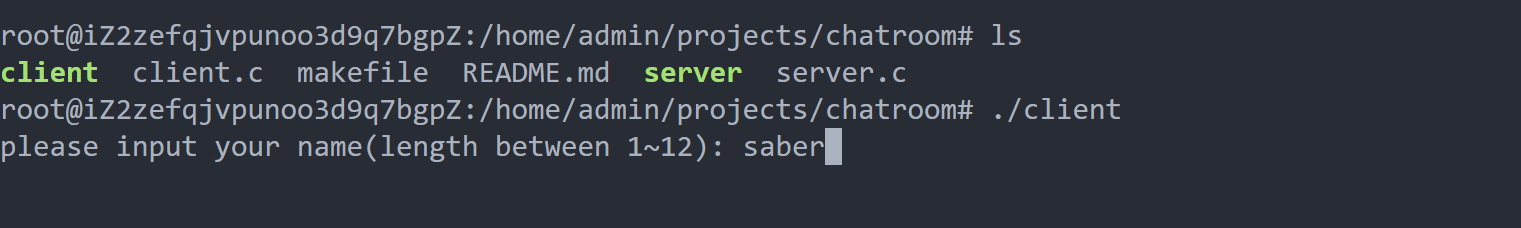
3、 服务器与客户端的具体内容可根据实际工作情况自由发挥，充分体现原创精神即可。需要：1）简述程序功能；2）具体的实现源代码及注释；3）实现的过程和结果截图。

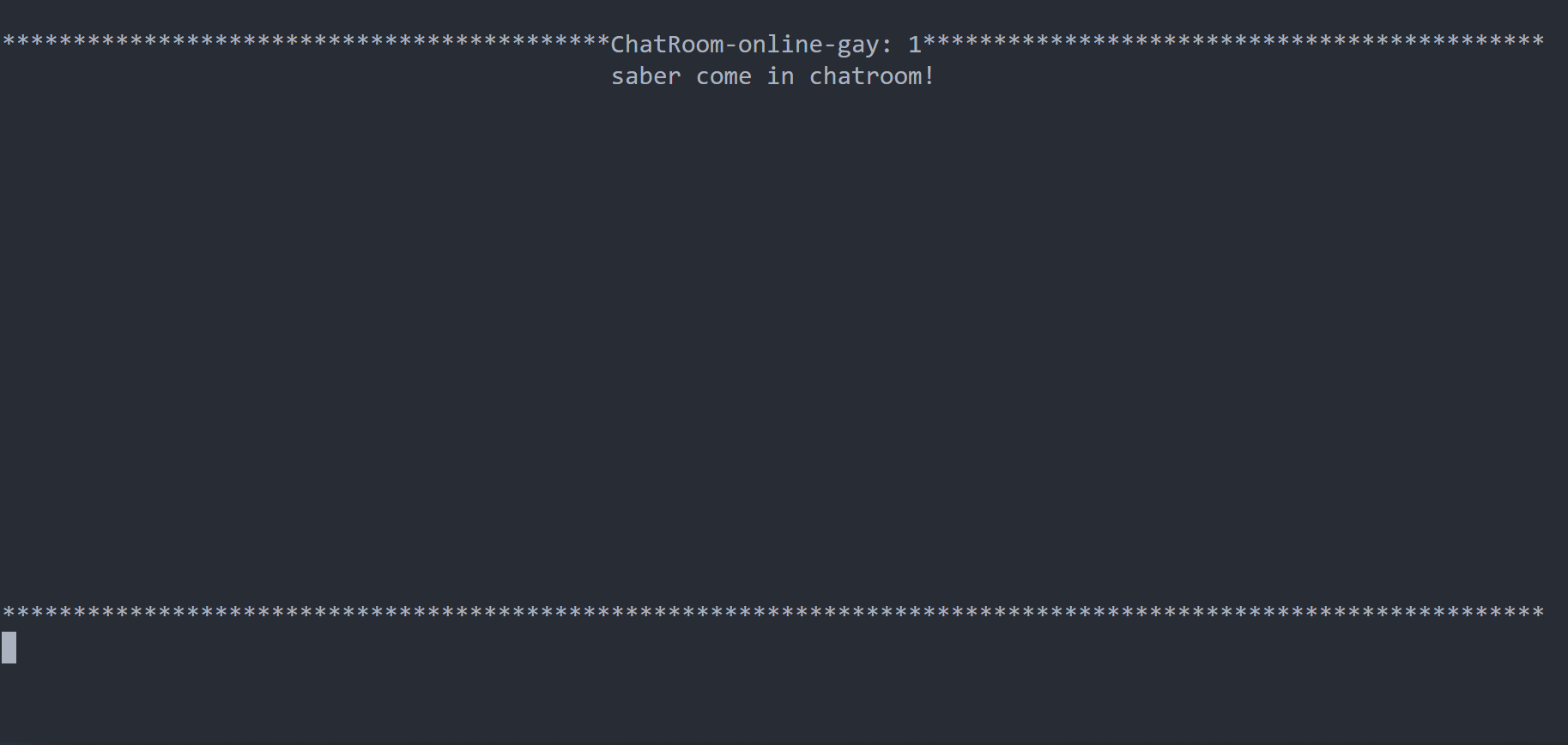
截图：

1. 运行 server。server监听服务器的6666端口, 等待客户端连接, 每一个客户端发送的消息都会被server转发到所有客户端

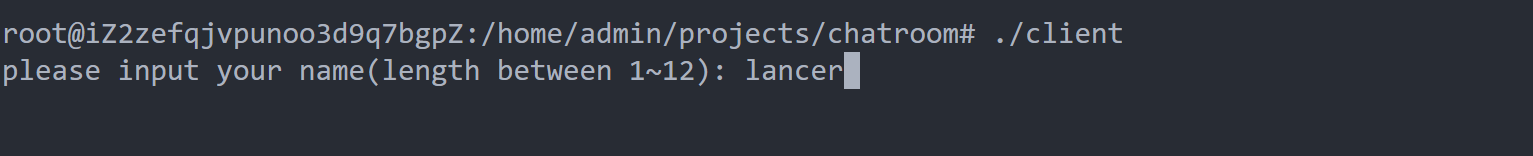


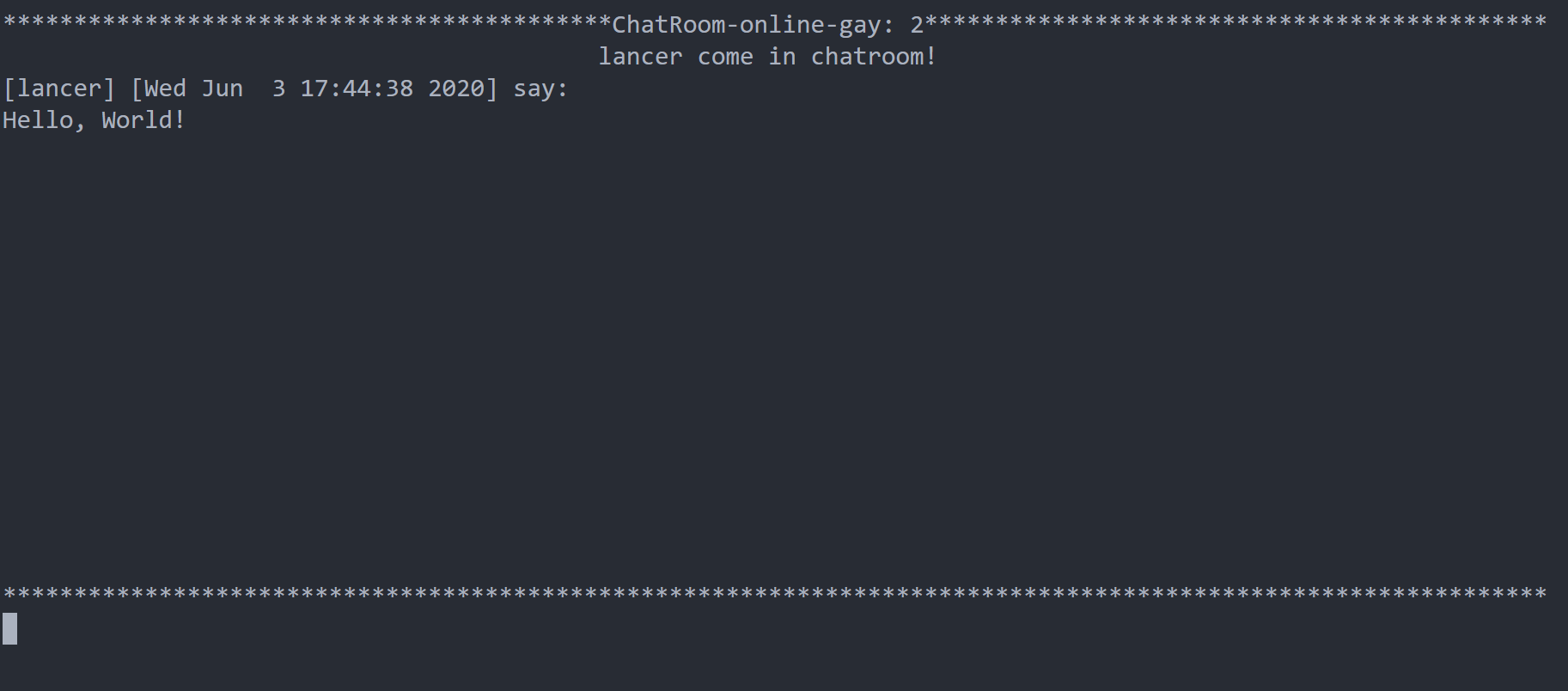
1. 运行client-1. 输入在chatroom中使用的昵称 **saber**



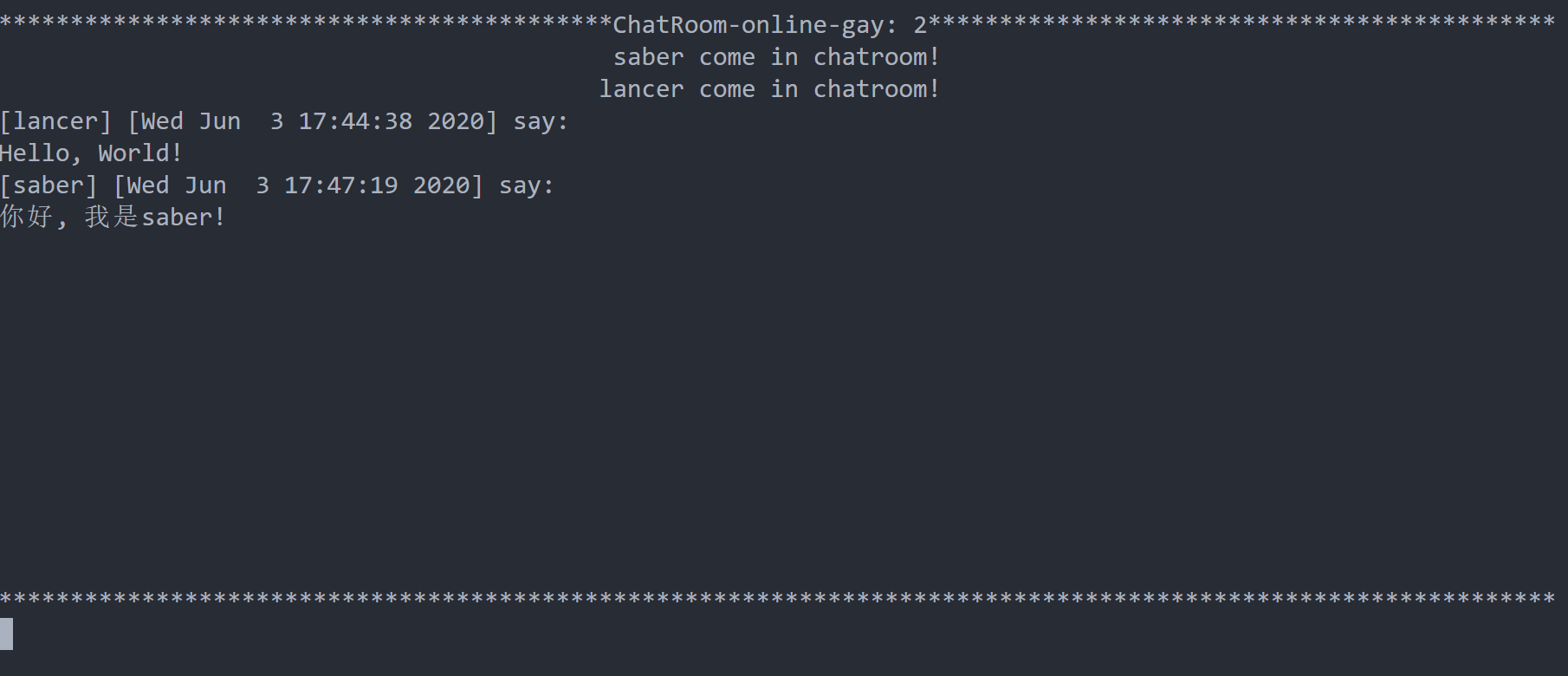


1. 运行client-2, 使用昵称 **lancer** ，并发送一条消息

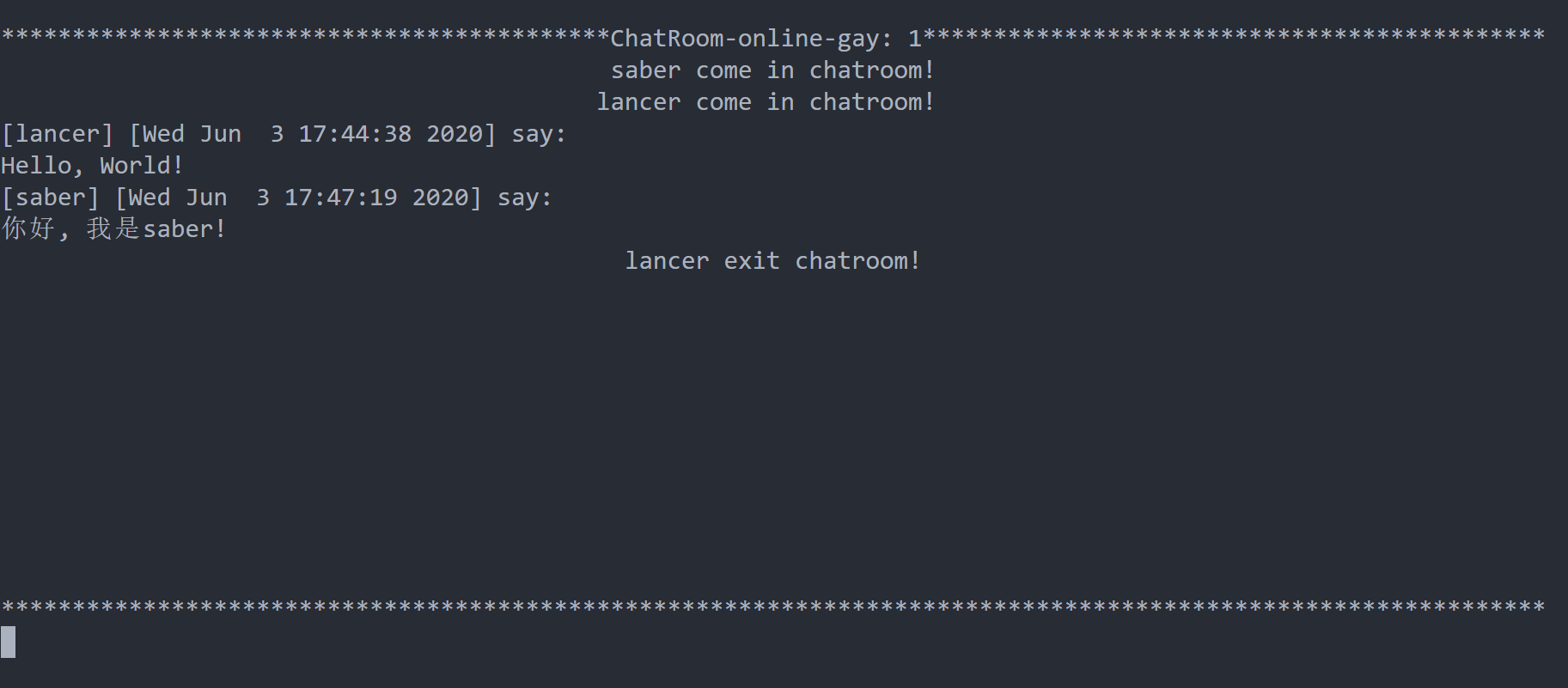




1. 查看 client-1，即saber中接收的内容，并发送一条消息



1. 在 client-2 中按ctrl+c退出客户端, 查看client-1中online人数的变化



## 四，代码

**Makefile**:

require:

sudo apt-get install libncurses5-dev libncursesw5-dev

server: server.c

gcc server.c -o server -pthread

client: client.c

gcc client.c -o client -pthread -lncursesw

.PHONY: build

build:

make require && make server && make client

**server.c**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <signal.h>

#include <string.h>

#include <time.h>

#include <pthread.h>

#include <stdarg.h>

#include <assert.h>

/\*

server->clients message formate:

[SERVER\_INS or sender's name] & [SERVER\_INS\_xxx] & [online gay] & [update time] & [message]

\*/

#define SERVER\_PORT 6666

#define MAX\_CLIENT\_NUM 32

#define SERVER\_INS "SERVER-INS"

#define SERVER\_INS\_FMT "%s&%d&%d&%s&%s"

#define SERVER\_INS\_MESSAGE 0

#define SERVER\_INS\_USER\_COME 1

#define SERVER\_INS\_USER\_EXIT 2

struct client\_t {

pthread\_t thread;

int sockfd;

char username[20];

int name\_length;

};

struct client\_t clients[MAX\_CLIENT\_NUM];

int sockfd;

pthread\_mutex\_t mutex;

void \*request\_handler(void \*sockfd);

void sigint\_handler(int num);

char \*get\_time();

struct client\_t \*get\_client();

void send\_all(char buffer[], int length);

void init\_client();

void clear\_client(struct client\_t \*client);

void delete\_clients();

struct client\_t \*get\_client\_from\_sockfd(int sockfd);

void send\_server\_ins(char \*fmt, ...);

int online\_gay();

void forward\_server\_ins(int ins\_num, char \*message);

void forward\_message(char \*username, char \*message);

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

{

// set sigint handler

signal(SIGINT, sigint\_handler);

// ignore sigpipe

signal(SIGPIPE, SIG\_IGN);

// init all clients

init\_client();

// init mutex

int init\_res = pthread\_mutex\_init(&mutex, NULL);

assert(init\_res == 0);

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

assert(sockfd != -1);

// bind

struct sockaddr\_in serv\_addr;

memset(&serv\_addr, 0, sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = inet\_addr("0.0.0.0");

serv\_addr.sin\_port = htons(SERVER\_PORT);

assert(bind(sockfd, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) != -1);

// listen

assert(listen(sockfd, 20) != -1);

printf("run tcp server in %s:%d ......\n", "127.0.0.1", SERVER\_PORT);

struct sockaddr\_in clnt\_addr;

socklen\_t clnt\_addr\_size = sizeof(clnt\_addr);

while (1){

int cfd = accept(sockfd, (struct sockaddr \*)&clnt\_addr, &clnt\_addr\_size);

printf("[%s] connect from %s:%d socket: %d\n", get\_time(), inet\_ntoa(clnt\_addr.sin\_addr), clnt\_addr.sin\_port, cfd);

struct client\_t \*c = get\_client();

if(c == NULL){

close(cfd);

continue;

}

c->sockfd = cfd;

// create a new thread to connect to client

pthread\_t thread;

int res = pthread\_create(&thread, NULL, request\_handler, (void\*)(&cfd));

if(res == 0){

c->thread = thread;

}else{

close(cfd);

}

}

close(sockfd);

return 0;

}

void \*request\_handler(void\* sockfd){

int fd = \*((int\*)sockfd);

printf("current thread: %ld socket: %d\n", pthread\_self(), fd);

struct client\_t \*client = get\_client\_from\_sockfd(fd);

char buffer[1024];

char data[1024];

int n = -1;

// read username, user come in chat-room

if ((n = recv(fd, buffer, 1024, 0)) > 0){

buffer[n] = '\0';

strcpy(client->username, buffer);

client->name\_length = n;

strncpy(data, buffer, n);

data[n] = '&';

}

// send 'user come in chatroom msg'

forward\_server\_ins(SERVER\_INS\_USER\_COME, client->username);

while ((n = recv(fd, buffer, 1024, 0)) > 0){

buffer[n] = '\0';

printf("[%s]: %s\n", client->username, buffer);

forward\_message(client->username, buffer);

}

// user get out of the chat room

close(fd);

clear\_client(client);

// send 'user get out of the chatroom' msg

forward\_server\_ins(SERVER\_INS\_USER\_EXIT, client->username);

pthread\_exit(NULL);

}

void sigint\_handler(int num){

close(sockfd);

delete\_clients();

pthread\_mutex\_destroy(&mutex);

exit(0);

}

char \*get\_time(){

time\_t t = time(NULL);

char \*time = ctime(&t);

char \*res = time;

while (\*time != '\n')

{

time++;

}

\*time = '\0';

return res;

}

struct client\_t\* get\_client(){

struct client\_t \*client = NULL;

pthread\_mutex\_lock(&mutex);

for(int i=0; i<MAX\_CLIENT\_NUM; i++){

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

client = &clients[i];

}

}

pthread\_mutex\_unlock(&mutex);

return client;

}

void send\_all(char buffer[], int length){

pthread\_mutex\_lock(&mutex);

for (int i = 0; i < MAX\_CLIENT\_NUM; i++){

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

send(clients[i].sockfd, buffer, length, 0);

}

}

pthread\_mutex\_unlock(&mutex);

}

void init\_client(){

for (int i = 0; i < MAX\_CLIENT\_NUM; i++){

clients[i].sockfd = 0;

clients[i].thread = 0;

}

}

struct client\_t \*get\_client\_from\_sockfd(int sockfd){

for (int i = 0; i < MAX\_CLIENT\_NUM; i++){

if(clients[i].sockfd == sockfd){

return &clients[i];

}

}

return NULL;

}

void clear\_client(struct client\_t \*client){

pthread\_mutex\_lock(&mutex);

client->sockfd = 0;

client->thread = 0;

pthread\_mutex\_unlock(&mutex);

}

void delete\_clients(){

for (size\_t i = 0; i < MAX\_CLIENT\_NUM; i++){

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

close(clients[i].sockfd);

}

}

}

int online\_gay(){

int gay = 0;

pthread\_mutex\_lock(&mutex);

for (int i = 0; i < MAX\_CLIENT\_NUM; i++){

if (clients[i].sockfd != 0 && clients[i].thread != 0){

gay++;

}

}

pthread\_mutex\_unlock(&mutex);

return gay;

}

// forward one client's message to all clients

void forward\_message(char \*username, char \*message){

send\_server\_ins(SERVER\_INS\_FMT, username, SERVER\_INS\_MESSAGE, online\_gay(), get\_time(), message);

}

void forward\_server\_ins(int ins\_num, char \*message){

send\_server\_ins(SERVER\_INS\_FMT, SERVER\_INS, ins\_num, online\_gay(), get\_time(), message);

}

void send\_server\_ins(char \*fmt, ...){

char buffer[1024];

va\_list args;

va\_start(args, fmt);

vsprintf(buffer, fmt, args);

va\_end(args);

int length = strlen(buffer);

buffer[length+1] = '\n';

printf("[%s]: %s\n", "SEND\_ALL", buffer);

send\_all(buffer, length);

}

**Client.c**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <signal.h>

#include <string.h>

#include <pthread.h>

#include <ncurses.h>

#include <locale.h>

#include <assert.h>

#define SERVER\_ADDR "39.107.83.159"

#define SERVER\_PORT 6666

#define SCREEN\_BEGIN\_LINE 1

#define SERVER\_INS "SERVER-INS"

#define SERVER\_INS\_MESSAGE 0

#define SERVER\_INS\_USER\_COME 1

#define SERVER\_INS\_USER\_EXIT 2

#define USER\_COME\_TIPS " come in chatroom!"

#define USER\_EXIT\_TIPS " exit chatroom!"

#define ONLINE\_GAY\_TIPS "ChatRoom-online-gay: "

#define ONLINE\_GAY\_TIPS\_LENGTH (strlen(ONLINE\_GAY\_TIPS))

#define USER\_COME\_TIPS\_LENGTH (strlen(USER\_COME\_TIPS))

#define USER\_EXIT\_TIPS\_LENGTH (strlen(USER\_EXIT\_TIPS))

int sockfd;

int next\_writed\_row = SCREEN\_BEGIN\_LINE;

WINDOW \*win;

char username[12];

int name\_length;

void \*read\_user\_input(void \*sockfd);

void \*recv\_from\_server(void \*sockfd);

void sigint\_handler(int num);

void refresh\_input\_box();

void init\_client(int sockfd);

void write\_display\_win(char buffer[]);

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

printf("please input your name(length between 1~12): ");

scanf("%s", username);

name\_length = strlen(username);

if(name\_length > 12){

printf("username length error!\n");

exit(0);

}

// set sigint handler

signal(SIGINT, sigint\_handler);

setlocale(LC\_ALL, "");

win = initscr(); /\* 初始化屏幕 \*/

curs\_set(TRUE);

clear();

refresh\_input\_box();

// create a socket

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

assert(sockfd != -1);

struct sockaddr\_in addr;

socklen\_t addrlen = sizeof(addr);

addr.sin\_family = AF\_INET;

addr.sin\_addr.s\_addr = inet\_addr(SERVER\_ADDR);

addr.sin\_port = htons(SERVER\_PORT);

int res = connect(sockfd, (struct sockaddr\*)&addr, addrlen);

assert(res != -1);

init\_client(sockfd);

pthread\_t p1, p2;

assert(pthread\_create(&p1, NULL, read\_user\_input, &sockfd) == 0);

assert(pthread\_create(&p2, NULL, recv\_from\_server, &sockfd) == 0);

pthread\_join(p1, NULL);

pthread\_join(p2, NULL);

return 0;

}

void init\_client(int sockfd){

if(send(sockfd, username, name\_length, 0) == -1){

printf("init your name error!\n");

exit(0);

}

}

void \*read\_user\_input(void \*sockfd){

int fd = \*((int\*)sockfd);

char buffer[1024];

while(1){

getstr(buffer);

send(fd, buffer, strlen(buffer), 0);

refresh\_input\_box();

}

}

void \*recv\_from\_server(void \*sockfd){

int fd = \*((int \*)sockfd);

char buffer[1024];

int n = -1;

while((n = recv(fd, buffer, 1024, 0)) > 0){

buffer[n] = '\0';

write\_display\_win(buffer);

}

}

void sigint\_handler(int num){

close(sockfd);

endwin();

exit(0);

}

void refresh\_input\_box(){

int row, clo;

getmaxyx(stdscr, row, clo);

int line\_row = row \* 4 / 5;

for (int i = line\_row; i < row; i++){

move(i, 0);

clrtoeol();

}

move(line\_row, 0);

for (int i = 0; i < clo; i++){

printw("\*");

}

move(line\_row+1, 0);

refresh();

}

void write\_display\_win(char buffer[]){

// save current position

int old\_row, old\_clo;

getyx(win, old\_row, old\_clo);

int cur\_row, cur\_clo;

int row, clo;

getmaxyx(stdscr, row, clo);

int line\_row = row \* 4 / 5;

int length = strlen(buffer);

int last\_row = next\_writed\_row + length / clo + (length % clo == 0 ? 0 : 1);

if (last\_row >= line\_row){

for (int i = SCREEN\_BEGIN\_LINE; i < line\_row; i++){

move(i, 0);

clrtoeol();

}

next\_writed\_row = SCREEN\_BEGIN\_LINE;

}

char \*begin = buffer;

char \*ins\_header = strsep(&begin, "&");

char \*ins\_num = strsep(&begin, "&");

char \*online\_gay = strsep(&begin, "&");

char \*time = strsep(&begin, "&");

char \*message = begin;

// update online gay

move(0, 0);

int online\_gay\_begin\_clo = (clo - strlen(online\_gay) - ONLINE\_GAY\_TIPS\_LENGTH) / 2;

for (int i = 0; i < online\_gay\_begin\_clo; i++){

printw("\*");

}

move(0, online\_gay\_begin\_clo);

printw("%s%s", ONLINE\_GAY\_TIPS, online\_gay);

getyx(win, cur\_row, cur\_clo);

for (int i = cur\_clo; i < clo; i++){

printw("\*");

}

refresh();

// write data

move(next\_writed\_row, 0);

switch (atoi(ins\_num))

{

case SERVER\_INS\_MESSAGE:

printw("[%s] [%s] say:\n%s", ins\_header, time, message);

break;

case SERVER\_INS\_USER\_COME:

move(next\_writed\_row, (clo-strlen(message)-USER\_COME\_TIPS\_LENGTH)/2);

printw("%s%s", message, USER\_COME\_TIPS);

break;

case SERVER\_INS\_USER\_EXIT:

move(next\_writed\_row, (clo-strlen(message)-USER\_EXIT\_TIPS\_LENGTH)/2);

printw("%s%s", message, USER\_EXIT\_TIPS);

break;

default:

printw("[%s] [%s] say:\n%s", "no-name gay", "just now", "fuck! it means occured bugs if you see this line!");

break;

}

// mvprintw(next\_writed\_row, 0, buffer);

refresh();

// update next\_writed\_row

getyx(win, row, clo);

next\_writed\_row = (clo == 0 ? row : (row + 1));

// back to old position

move(old\_row, old\_clo);

refresh();

}