

Advanced network programming

Team leader: Zhuo Dou

Team member: Cui Haipeng, Chen Han, Chen Shuyu, Ji Yue

1、Multi-thread

"SocketObject" is a structure which store socket elements, "socket" represent current socket, "name[]" is a list to store current user names and "address" is used to store current thread's address, "ring[]" stored the user-name who is notified to connect .

```
struct SocketObject{
    char name[30];
    unsigned address;
    SOCKET socket;
    char ring[30];
};
```

"sockets[THREAD_NUM]" is an important global variable, is used to store child thread's information.

```
//最重要的全局变量sockets，相当于线程池，管理子线程的信息
struct SocketObject sockets[THREAD_NUM];
```

"_beginthreadex" is a function to create child thread, after create a new thread, we need to store the thread's information in "SocketObject", finally add the "SocketObject" into "socket[]".

```
//存储线程Id的无符号int
unsigned threadAddress;
//创建线程
_beginthreadex(NULL, 0, (unsigned int (__stdcall *) (void *)) ThreadFun, NULL, 0, &threadAddress);
//将该线程的信息保存到全局变量sockets中
struct SocketObject socketObject;
socketObject.address=threadAddress;
socketObject.socket=new_socket;
sockets[++g_count]=socketObject;
```

In Linux, the way to launch multi-thread is different.

```
//启动监听线程
//HANDLE handle=_beginthreadex(NULL, 0, &Client_thread,(void *)&scilent, 0,NULL);
void *ret=NULL;
pthread_t thread1;
int ret_thrd1;
ret_thrd1 = pthread_create(&thread1, NULL, (void *)&Client_thread, (void *) &scilent);
... ..
```

2、User Login

Create database file by using sqlite 3, then insert some username and password into the file.

```
int insertData(){
    //创建数据库文件, 文件名为dzData
    sqlite3 *db;
    char *err_msg=0;
    int rc = sqlite3_open("dzData", &db);
    //printf("\nRC open Database = %d\n",rc);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return 0;
    }
    char *sql = "DROP TABLE IF EXISTS User;"
               "CREATE TABLE User(Id INT, Name TEXT, Password TEXT);"
               "INSERT INTO User VALUES(1, 'dz', '123');"
               "INSERT INTO User VALUES(2, 'han', '123');"
               "INSERT INTO User VALUES(3, 'haipen', '123');"
               "INSERT INTO User VALUES(4, 'aaa', '123');"
               "INSERT INTO User VALUES(5, 'bbb', '123');"
               "INSERT INTO User VALUES(6, 'ccc', '123');"
               "INSERT INTO User VALUES(7, 'ddd', '123');"
               "INSERT INTO User VALUES(8, 'eee', '123');";
    rc = sqlite3_exec(db, sql, 0, 0, &err_msg);
    //printf("\nRC sqlite_exec = %d\n",rc);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "SQL error: %s\n", err_msg);
        sqlite3_free(err_msg);
        sqlite3_close(db);
        return 0;
    } else {
        fprintf(stdout, "Table created successfully\n");
    }
    return 1;
}
```

When a user log in, it will search the database to verify if the username and password are both correct, if not then it will failed to connect.

```
//search database
//查找数据库, 验证帐号密码
sqlite3_stmt * res;
char *sql = "SELECT Name,Password FROM User WHERE Name = @id";
int rc = sqlite3_prepare_v2(db, sql, -1, &res, 0);
if (rc == SQLITE_OK) {
    int idx = sqlite3_bind_parameter_index(res, "@id");
    rc=sqlite3_bind_text(res, idx, nameArray, -1, SQLITE_STATIC);    // the string is static
    //printf("\nRC sqlite_exec = %d -> %s\n",rc,sql);
} else {
    fprintf(stderr, "Failed to execute statement: %s\n", sqlite3_errmsg(db));
}

int step =sqlite3_step(res);
```

3、 send and receive messages (private/public).

There are exits a listener to judge the input in a thread, if users send a message, func broadcast[] will be used to regenerate the sentence then other client will receive it.

```
//监听的线程
unsigned int _stdcall Client_thread(void* s){
    SOCKET socket=*(SOCKET*)s;
    while(1){
        char recData[BROADCAST_MAX];
        int ret = recv(socket, recData, BROADCAST_MAX, 0);
        if (ret > 0)
        {
            recData[ret] = 0x00;
            //printf("%s\r\n",charArray);
        }else{
            printf("receive error.");
            closesocket(socket);
            WSACleanup();
            return -1;
        }
    }
}
```

In addition, server will distinguish the private/public message, if the message is defined as a private one and the target user is being connected, then func send() will only send the message to him, if the message is public then it will be send to all users.

```
int i;
char broadcast[BROADCAST_MAX];
//结构化广播的消息
broadcast[0]=0x00;
strcat(broadcast,"From ");
strcat(broadcast,nameArray);
strcat(broadcast,":");
strcat(broadcast,revData);
if(private_index>=0){
    //如果是私聊，只发给一个用户
    send(sockets[private_index].socket, broadcast, strlen(broadcast), 0);
}else{
    //不是私聊，广播给在线的用户
    for(i=0;i<=g_count;i++){
        //printf("%s\n",sockets[i].name);
        //printf("%p\n",sockets[i].socket);
        send(sockets[i].socket, broadcast, strlen(broadcast), 0);
    }
}

broadcast[0]=0x00;
```

Private: If sever received a command “#Private user-name”, func strtok() is used to extract the user-name, then use for loop to find whether the user is connect on the server, if user is connect then store the private_index(the target user’s address) into socket, then send the message to him.

```

} else if (is_begin_with(revData, "#Private") == 1) {
    // 私聊
    printf("I got private.\n");
    char name[50];
    name[0] = 0x00;
    char *p;
    strtok(revData, " ");
    // 提取#Private <username>空格之后的username, 允许名字中含有空格 (但未测试)
    while ((p = strtok(NULL, " ")) != NULL) {
        // printf(p);
        // printf("\n");
        strcat(name, p);
        strcat(name, " ");
    }
    if (strlen(name) == 0) {
        char* errorSend = "Sorry, Please input name after #Private (there should be a space after Private).";
        send(new_socket, errorSend, strlen(errorSend), 0);
    } else {
        name[strlen(name) - 1] = 0x00;
        int j;
        // 查找该用户是否在线
        for (j = 0; j <= g_count; j++) {
            if (strcmp(sockets[j].name, name) == 0) break;
        }
        if (j > g_count) {
            // for循环跑完, 说明用户不在线
            char* errorSend = "Sorry, I can't find name in user list.";
            send(new_socket, errorSend, strlen(errorSend), 0);
        } else {
            // 如果小于等于g_count, 说明找到了, 保存index到这个socket中
            private_index = sockets[j].address;
            char* message = "Success be private.";
            send(new_socket, message, strlen(message), 0);
        }
    }
    continue;
}

if (private_index > 0) {
    // 如果是私聊, 只发给一个用户
    // printf("%d, private address:\n", private_index);
    for (int i = 0; i <= g_count; i++) {
        // printf("%d\n", sockets[i].address);
        if (sockets[i].address == private_index) {
            send(sockets[i].socket, broadcast, strlen(broadcast), 0);
            break;
        }
    }
}
}

```

Public: If sever received a command “#Public”, then it will send the message to all the clients.

```

    continue;
} else if (strcmp(revData, "#Public") == 0) {
    // 将private标记为-1, 即可发出公开消息
    private_index = -1;
    char* message = "Success be public.";
    send(new_socket, message, strlen(message), 0);
    continue;
} else if (is_begin_with(revData, "#Ping") == 1) {

```

4、Transfer files (Upload)

In “server.c”, we create a while loop to judge every received command from the client, including

“#Exit”、“#ListU”、“#Private”、“#Public”、“#TrfU”、“#TrfD”.

When the sever received “#TrfU”, it will start to upload a file. To begin with, it will receive the file title, func access() is used to judge whether the file is exist, if the file has already exist, then break the process, otherwise the main thread will send a command “UPLOAD_ACK” to the client.

```
continue;
}else if(is_begin_with(revData,"#TrfU")==1){
    //传输文件指令
    printf("I got file.\n");
    char fileName[50];
    fileName[0]='\0';
    char *tempName=NULL;
    strtok(revData," ");
    tempName=strtok(NULL," ");
    //加入文件头file/
    strcat(fileName,fileTitle);
    strcat(fileName,tempName);
    //判断文件是否存在, access是C语言的函数
    if(!access(fileName,0)){
        char * wrongMessage="Sorry, file exists.Please change your fileName.";
        send(new_socket, wrongMessage, strlen(wrongMessage), 0);
    }else{
        //如果文件存在
        char sendAck[100];
        sendAck[0]='\0';
        strcat(sendAck,"UPLOAD_ACK:");
        strcat(sendAck,tempName);
        //返回给client 一个以UPLOAD_ACK开头的消息, 客户端接收之后可以开始发送文件数据
        send(new_socket, sendAck, strlen(sendAck), 0);
        printf("try to save file %s\n",tempName);
        //保存文件, 理论上应该写个函数包装一下, 然而我懒了
        FILE* fp;
        char data[FILE_DATA_MAX];
        data[0]='\0';
        fp=fopen(fileName,"wb");
        if(fp==NULL){
            printf("fail to save file named %s.\n",fileName);
            continue;
        }
        printf("start to save file...\n");
```

```

while(1){
    memset(data,0,sizeof(data));
    //接收文件数据
    int length=recv(new_socket,data,sizeof(data),0);
    if(length==SOCKET_ERROR){
        char * message="Fail to save file.It may be caused by incomplete file.";
        printf("%s\n",message);
        send(new_socket,message,strlen(message),0);
        break;
    }else if(strcmp(data,file_end_ack)==0){
        //如果client发送file_end_ack表明文件已经传输完毕了
        //没想到更好的方法来让服务器知道文件传输完毕
        char * message="Save file success.";
        printf("%s\n",message);
        send(new_socket,message,strlen(message),0);
        break;
    }else{
        //如果收到的不是结束ack, 则向文件里写入数据
        if(fwrite(data,1,length,fp)<length){
            printf("Write Failed.\n");
            break;
        }
    }
}
}
}

```

Once the client have received "UPLOAD_ACK" then it will begin to send the file data, when the whole file has been sendd successfully, the client will send a command "file_end_ack" back.

```

//如果接受到以UPLOAD_ACK的消息。表示主线程发送过上传文件的指令
//服务器已经确认可以传输了，子线程将数据发送
if(is_begin_with(recData,"UPLOAD_ACK")){
    char *fileName=NULL;
    strtok(recData,":");
    fileName=strtok(NULL," ");
    if(fileName==NULL){
        printf("receieve file name error.\n");
        continue;
    }else{
        //文件指针用于传输文件
        FILE *fp;
        char data[FILE_DATA_MAX];

        fp=fopen(fileName,"rb");
        if(fp==NULL){
            printf("file dose not exist.\n");
            continue;
        }

        while(1){
            memset((void *)data,0,sizeof(data));
            //读取文件数据
            int length=fread(data,1,sizeof(data),fp);
            if(length==0){
                Sleep(500);
                //如果上传完成。发送给服务器确认码。服务器即可退出监听的循环
                send(socket,file_end_ack,strlen(file_end_ack),0);
                printf("File send Success\n");

                break;
            }
            //发送给服务器数据
            int ret=send(socket,data,length,0);
            //putchar('.');
            if(ret==SOCKET_ERROR){
                printf("Failed to send file.It may be caused by incomplete file.\n");
                break;
            }
        }
        fclose(fp);
        continue;
    }
}
}
}

```

5、Transfer files (Download)

In a similar way, when the sever received “#TrfD”, it will start to download a file. Firstly, the sever will try to find the file according to the file path, if the file is not exist, then the process will break, otherwise the main thread will send a command “FILE_SEND” to the client.

```
}else if(is_begin_with(revData,"#TrfD")==1){
    //下载文件指令。与上传很类似
    char fileName[50];
    fileName[0]=0x00;
    char * p=NULL;
    //fileName[0]=0x00;
    //char *tempName;
    strcat(fileName,"file/");
    strtok(revData," ");
    p=strtok(NULL," ");
    strcat(fileName,p);
    //printf("%s\n",fileName);
    FILE *fp;
    char data[FILE_DATA_MAX];
    data[0]=0x00;

    fp=fopen(fileName,"rb");
    if(fp==NULL){
        //如果文件指针打开失败。说明文件不存在。返回给client
        char* message="file dose not exist.";
        send(new_socket,message,strlen(message),0);
        printf("%s\n",message);
        continue;
    }
    //通知client可以开始上传文件了
    send(new_socket,FILE_SEND,strlen(FILE_SEND),0);
    //此处如果不延迟发送。socket可能会将两个信息合并在一起发送
    Sleep(500);
    //返回给client文件名,client需要用这个名字创建文件
    send(new_socket,p,strlen(p),0);

    while(1){
        memset((void *)data,0,sizeof(data));

        int length=fread(data,1,sizeof(data),fp);
        //fread返回参数为读到该文件数组的长度。如果为零说明文件读完了
        if(length==0){
            //与之前sleep同理。不用会把两个socket同时传
            Sleep(500);
            send(new_socket,file_end_ack,strlen(file_end_ack),0);
            printf("File send Success\n");
            break;
        }

        //将文件发送给client
        int ret=send(new_socket,data,length,0);
        //putchar('.');
        if(ret==SOCKET_ERROR){
            char * message="Failed to send file.It may be caused by incomplete file.";
            printf("%s\n",message);
            send(new_socket,message,strlen(message),0);
            break;
        }
    }
    fclose(fp);
    continue;
}
```


Once the client have received "FILE_SEND" then it will begin to receive the file data, when the whole file has been received successfully, the client will send a command "file_end_ack" back.

```
//通知client可以开始接受文件了
send(new_socket, FILE_SEND, strlen(FILE_SEND), 0);
//此处如果不延迟发送, socket可能会将两个信息合并在一起发送
Sleep(500);
//返回给client文件名, client需要用这个名字创建文件
send(new_socket, p, strlen(p), 0);

while(1){
    memset((void *)data, 0, sizeof(data));

    int length=fread(data, 1, sizeof(data), fp);
    //fread返回参数为读到该文件数组的长度, 如果为零说明文件读完了
    if(length==0){
        //与之前sleep同理, 不用会把两个socket同时传
        Sleep(500);
        send(new_socket, file_end_ack, strlen(file_end_ack), 0);
        printf("File send Success\n");
        break;
    }
    //将文件发送给client
    int ret=send(new_socket, data, length, 0);
    //putchar('.');
    if(ret==SOCKET_ERROR){
        char * message="Failed to send file.It may be caused by incomplete file.";
        printf("%s\n", message);
        send(new_socket, message, strlen(message), 0);
        break;
    }
}
fclose(fp);
continue;
```


6、List file

When the sever received “#ListF”, according to the file path, we can find the file list, then return the result to client.

```
// 返回 #ListF
}else if(strcmp(revData,"#ListF")==0){
    //显示文件列表
    char path[200];
    path[0]=0x00;
    char result[1024];
    result[0]=0x00;
    strcat(path,"file");
    viewFiles(path,result);
    send(new_socket,result,strlen(result),0);
    continue;
}
```

```
void viewFiles(char * path,char * result){
    //根据path路径查找文件列表，并将结构化信息保存到result指针
    struct _finddata_t files;
    //char cFileAddr[300];
    long File_Handle;
    strcat(path,"/*.*");
    //printf("%s\n",path);
    //int i=0;
    File_Handle = _findfirst(path,&files);
    if(File_Handle==-1)
    {
        strcat(result,"Not found.");
    }else{
        do{
            if(files.name[0]!='.' && files.attrib!=_A_SUBDIR){
                strcat(result,files.name);
                strcat(result,"\n");
                //printf("find a file:%s\n",files.name);
            }
        }while( _findnext(File_Handle,&files)==0);
    }
}
```

7、List user

When the sever received “#ListU”, func getUserList() is used to traverse the sockets[] and get all online username.

```
}else if(strcmp(revData,"#ListU")==0){  
    //显示用户  
    char userList[1000];  
    getUserList(userList);  
    send(new_socket, userList, strlen(userList), 0);  
    userList[0]=0x00;  
    continue;  
}
```

```
void getUserList(char userList[]){  
    //结构化用户列表  
    strcat(userList,"users\n-----\n");  
    int i=0;  
    for(i=0;i<=g_count;i++){  
        //所有index小于等于g_count的均为在线对象  
        strcat(userList,sockets[i].name);  
        strcat(userList,"\n");  
    }  
    strcat(userList,"-----");  
}
```

8、Exit

If client send a command “#Exit” to sever, the thread and socket will be closed after the command has been sent. The sever will save the time and some related information into log file, then adjust the size of “sockets[]” that make sure there is no empty pointer.

```
//如果是关闭指令，将子线程关闭，再将socket关闭
if(strcmp(sendData, "#Exit")==0){
    TerminateThread(handle, 0);
    send(sclient, sendData, strlen(sendData), 0);
    closesocket(sclient);
    WSACleanup();
    break;
    //如果是上传指令，先在client判断指令是否正确
}
if(strcmp(recvData, "#Exit")==0){
    //如果是#Exit指令
    //记录时间
    time_t rawtime;
    struct tm * timeinfo;
    time (&rawtime);
    timeinfo = localtime (&rawtime);
    char saveLogMessage[100];
    sprintf(saveLogMessage, ("Time:%s%s log out.\n"), asctime(timeinfo), sockets[thisIndex].name);
    saveLog(saveLogMessage);
    if(thisIndex==g_count){
        //如果当前socket存在于sockets最后的位置，则直接将g_count-1即可
        //退出这个while循环这个线程就关闭了
        //socket关闭由client进行
        g_count--;
    }else{
        //如果当前socket存在于中间，则将最后一个socket放到当前的index下
        sockets[thisIndex]=sockets[g_count--];
    }
    printf("a socket is closed.\n");
    return 0;
}
```

In Linux, The different treatments for exceptions and exit.

```
//如果是关闭指令，将子线程关闭，再将socket关闭
if(strcmp(sendData, "#Exit")==0){
    //TerminateThread(handle, 0);
    //pthread_exit(0);
    pthread_cancel(thread1); //取消线程
    pthread_join(thread1, &ret);

    send(sclient, sendData, strlen(sendData), 0);
    //
    close(sclient);
    //shutdown(sclient, 2);
    break;
    //如果是上传指令，先在client判断指令是否正确
}
if(strcmp(recvData, "#Exit")==0){
    //如果是#Exit指令
    //记录时间
    time_t rawtime;
    struct tm * timeinfo;
    time (&rawtime);
    timeinfo = localtime (&rawtime);
    char saveLogMessage[100];
    sprintf(saveLogMessage, ("Time:%s%s log out.\n"), asctime(timeinfo), sockets[thisIndex].name);
    saveLog(saveLogMessage);
    if(thisIndex==g_count){
        //如果当前socket存在于sockets最后的位置，则直接将g_count-1即可
        //退出这个while循环这个线程就关闭了
        //socket关闭由client进行
        g_count--;
    }else{
        //如果当前socket存在于中间，则将最后一个socket放到当前的index下
        sockets[thisIndex]=sockets[g_count--];
    }
    printf("a socket is closed.\n");
    return 0;
}
```

9、Ring

Command “#Ring user-name” is used to remind users who are not online to connect. Firstly, get the target user’s name, searching in “socket[]”, if target user is online then send “he/she is online” back, if not then save the target user’s name into “sockets[].ring”.

```
continue;
}else if(is_begin_with(revData, "#Ring")==1){
    //ring指令
    printf("I got Ring.\n");
    char name[50];
    name[0]=0x00;
    char *p;
    strtok(revData, " ");
    while((p=strtok(NULL, " "))!=NULL){
        //printf(p);
        //printf("\n");
        strcat(name,p);
        strcat(name, " ");
    }
    if(strlen(name)==0){
        char* errorSend="Sorry, Please input name after #Ring (there should be a space after Ring).";
        send(new_socket, errorSend, strlen(errorSend), 0);
    }else{
        name[strlen(name)-1]=0x00;
        int i;
        //查找该用户是否已经在线
        for(i=0; i<=g_count; i++){
            if(strcmp(sockets[i].name, name)==0){
                char* message="He\\She is online.";
                send(new_socket, message, strlen(message), 0);
                break;
            }
        }
        //break只能跳出一层循环，所以这里还需要判断再跳出循环
        if(i<=g_count)continue;
        //如果i大于g_count，说明用户不在线，将ring的名字保存到该socket中
        strcpy(sockets[thisIndex].ring, name);

        char* message="Success save ring.";
        send(new_socket, message, strlen(message), 0);
    }
    continue;
```

10、Log file

Func saveLog() is used to save users' login information into log.txt. When users login, it will save the client ip address and the connection time after client connected with sever, and save the operation when user logout as well.

```
//将登录信息保存到日志中用到的时间
time_t rawtime;
struct tm * timeinfo;
time ( &rawtime );
timeinfo = localtime ( &rawtime );
if(step!=100 || strcmp((const char *)password,(const char *)sqlite3_column_text(res, 1))!=0){
    //如果登陆失败, 保存信息, 返回信息给客户端
    char * message="Wrong username or password.";
    send(new_socket, message, strlen(message), 0);
    char saveLogMessage[100];
    sprintf(saveLogMessage,("Time:%s\t%s login failed\n"),asctime(timeinfo),nameArray);
    saveLog(saveLogMessage);
}else{
    //如果登陆成功, 保存信息, 返回ACK给客户端
    char * returnAck="ACK";
    send(new_socket, returnAck, strlen(returnAck), 0);
    char saveLogMessage[100];
    sprintf(saveLogMessage,("Time:%s\t%s login success\n"),asctime(timeinfo),nameArray);
    saveLog(saveLogMessage);
    break;
}
}

void saveLog(char * message){
    //保存日志
    FILE* fp;

    fp=fopen("log.txt","a+");
    if(fp==NULL){
        printf("fail to save log\n");
        return;
    }
    fputs(message, fp);
    fclose(fp);
}
```

11. Connect server in Windows and client in Linux

The socket variable format is different from windows, it is “int” type but in windows it is “socket” type. Then we need to build an Oray in Windows Server in order to map the intranet to the public network, so that we can visit the server from another linux client.

```
int sclient=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
if (sclient == -1)
{
    printf("invalid socket !");
    exit(1);
}
struct sockaddr_in serAddr;
serAddr.sin_family = AF_INET;
serAddr.sin_port = htons(31448);
//serAddr.sin_addr.s_addr = inet_addr("139.162.250.136");
serAddr.sin_addr.s_addr = inet_addr("47.75.149.131");

//连接服务器，成功返回0，错误返回-1
if (connect(sclient, (struct sockaddr *)&serAddr, sizeof(serAddr)) < 0)
{
    perror("connect");
    exit(1);
}
```

 编辑  诊断

NetworkSocketServer

External network access
外网访问 u2402m4301.wicp.vip:31448

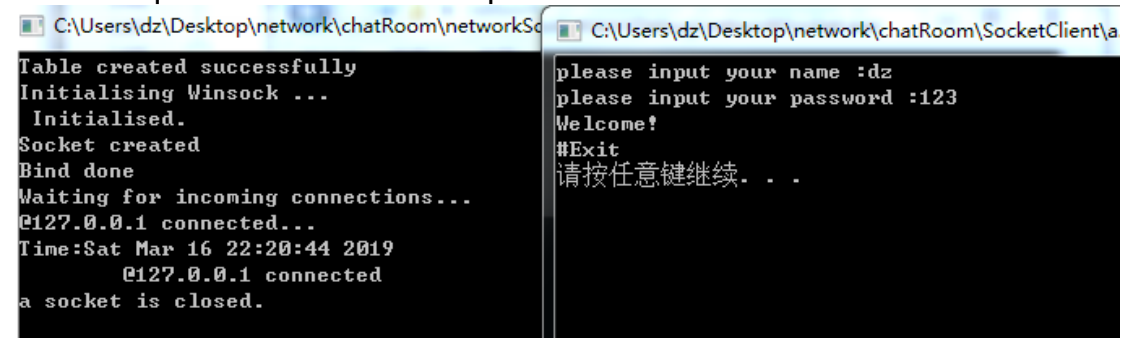
Intranet Host
内网主机 127.0.0.1:8888

Mapping broadband 1M 已用0B流量

3、Execution Screenshot

(1)、Connect Client/Server in TCP, and exit client. (#Exit)

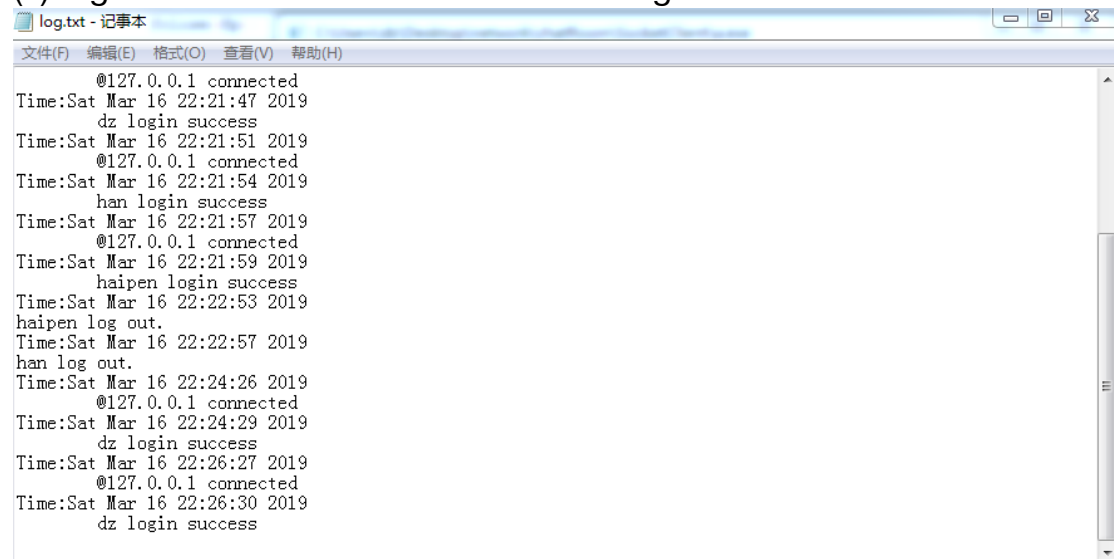
Sever require a username and password for connect with client.



```
C:\Users\dz\Desktop\network\chatRoom\networkServer>
Table created successfully
Initialising Winsock ...
Initialised.
Socket created
Bind done
Waiting for incoming connections...
@127.0.0.1 connected...
Time:Sat Mar 16 22:20:44 2019
@127.0.0.1 connected
a socket is closed.

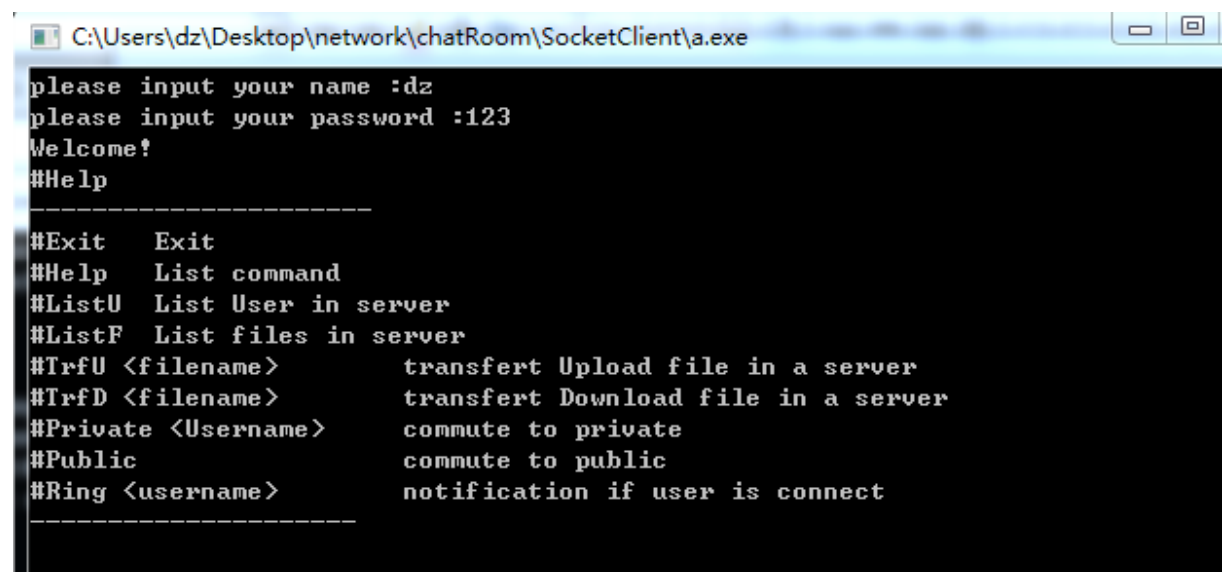
C:\Users\dz\Desktop\network\chatRoom\SocketClient\client.exe>
please input your name :dz
please input your password :123
Welcome!
#Exit
请按任意键继续. . .
```

(2)Log file which is used to store users' login information



```
log.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
@127.0.0.1 connected
Time:Sat Mar 16 22:21:47 2019
dz login success
Time:Sat Mar 16 22:21:51 2019
@127.0.0.1 connected
Time:Sat Mar 16 22:21:54 2019
han login success
Time:Sat Mar 16 22:21:57 2019
@127.0.0.1 connected
Time:Sat Mar 16 22:21:59 2019
haipen login success
Time:Sat Mar 16 22:22:53 2019
haipen log out.
Time:Sat Mar 16 22:22:57 2019
han log out.
Time:Sat Mar 16 22:24:26 2019
@127.0.0.1 connected
Time:Sat Mar 16 22:24:29 2019
dz login success
Time:Sat Mar 16 22:26:27 2019
@127.0.0.1 connected
Time:Sat Mar 16 22:26:30 2019
dz login success
```

(3)、List Command. (#Help)



```
C:\Users\dz\Desktop\network\chatRoom\SocketClient\client.exe>
please input your name :dz
please input your password :123
Welcome!
#Help
-----
#Exit      Exit
#Help      List command
#ListU     List User in server
#ListF     List files in server
#TrfU <filename>      transfert Upload file in a server
#TrfD <filename>      transfert Download file in a server
#Private <Username>   commute to private
#Public     commute to public
#Ring <username>      notification if user is connect
-----
```


(4)、Send and receive message by multi-threads

The screenshot displays a C++ chat application. On the left is the server console window titled 'client.c' with tabs for 'main.c', 'head.h', and 'client.c'. It shows the following output:

```
Table created successfully
Initialising Winsock ...
  Initialised.
Socket created
Bind done
Waiting for incoming connection
@127.0.0.1 connected...
Time:Sat Mar 16 22:20:44 2019
  @127.0.0.1 connected
a socket is closed.
@127.0.0.1 connected...
Time:Sat Mar 16 22:21:45 2019
  @127.0.0.1 connected
@127.0.0.1 connected...
Time:Sat Mar 16 22:21:51 2019
  @127.0.0.1 connected
@127.0.0.1 connected...
Time:Sat Mar 16 22:21:57 2019
  @127.0.0.1 connected
```

On the right are three client window instances, all titled 'C:\Users\dz\Desktop\network\chatRoom\SocketClient\a.exe'. Each window shows a login sequence:

- Client 1: 'please input your name :dz', 'please input your password :123', 'Welcome!', 'From haipen:Hello everyone!'
- Client 2: 'please input your name :han', 'please input your password :123', 'Welcome!', 'From haipen:Hello everyone!'
- Client 3: 'please input your name :haipen', 'please input your password :123', 'Welcome!', 'Hello everyone!', 'From haipen:Hello everyone!'

(5)、List all users in server. (#ListU)

This screenshot shows the same chat application but with the '#ListU' command being used. The server console window, titled 'C:\Users\dz\Desktop\network\chatRoom\networkSocket\bin\Debug\networkSocket.exe', shows the same connection logs as before, plus the command being received:

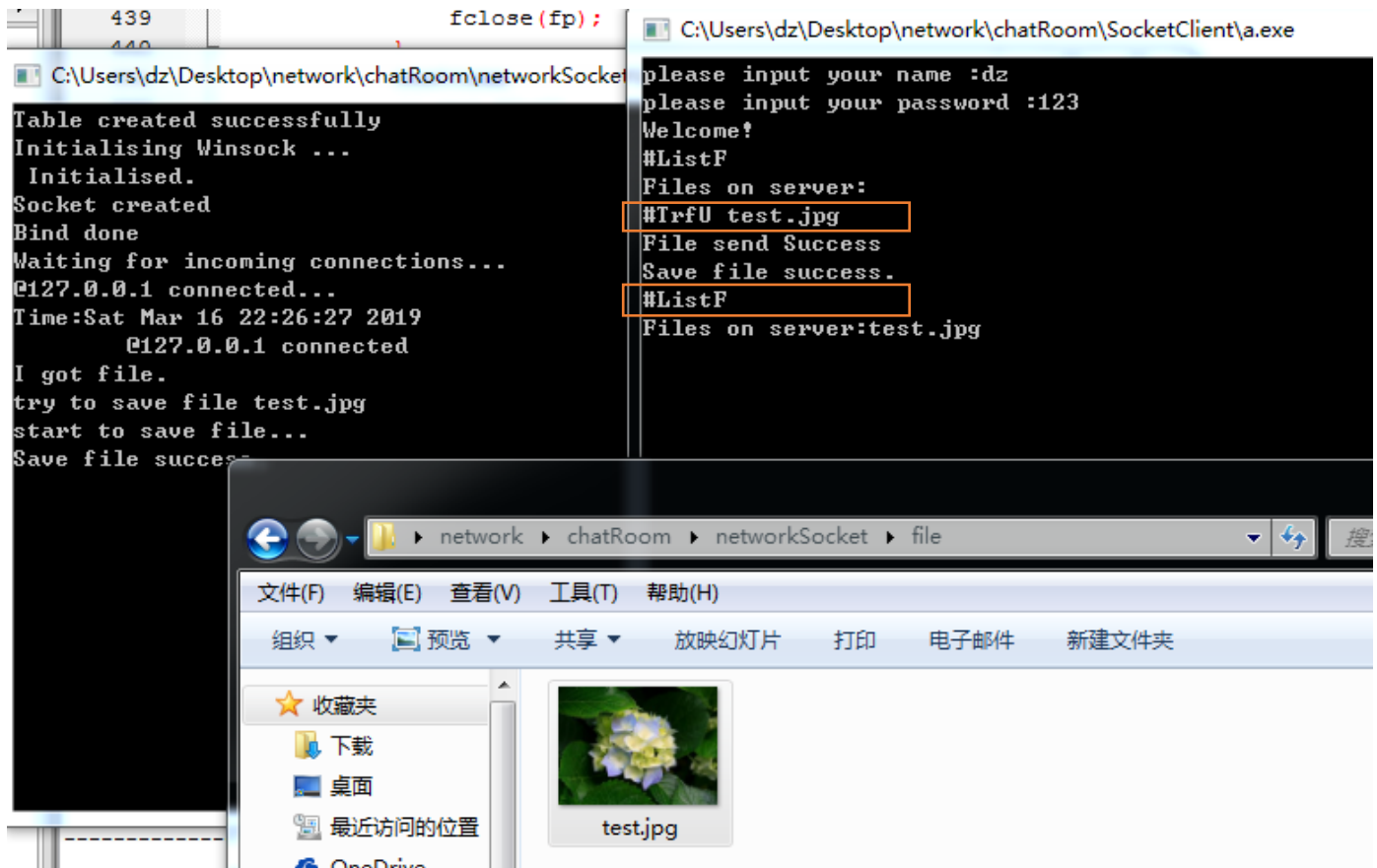
```
Table created successfully
Initialising Winsock ...
  Initialised.
Socket created
Bind done
Waiting for incoming connections...
@127.0.0.1 connected...
Time:Sat Mar 16 23:02:26 2019
  @127.0.0.1 connected
@127.0.0.1 connected...
Time:Sat Mar 16 23:02:40 2019
  @127.0.0.1 connected
```

The client windows show the following interactions:

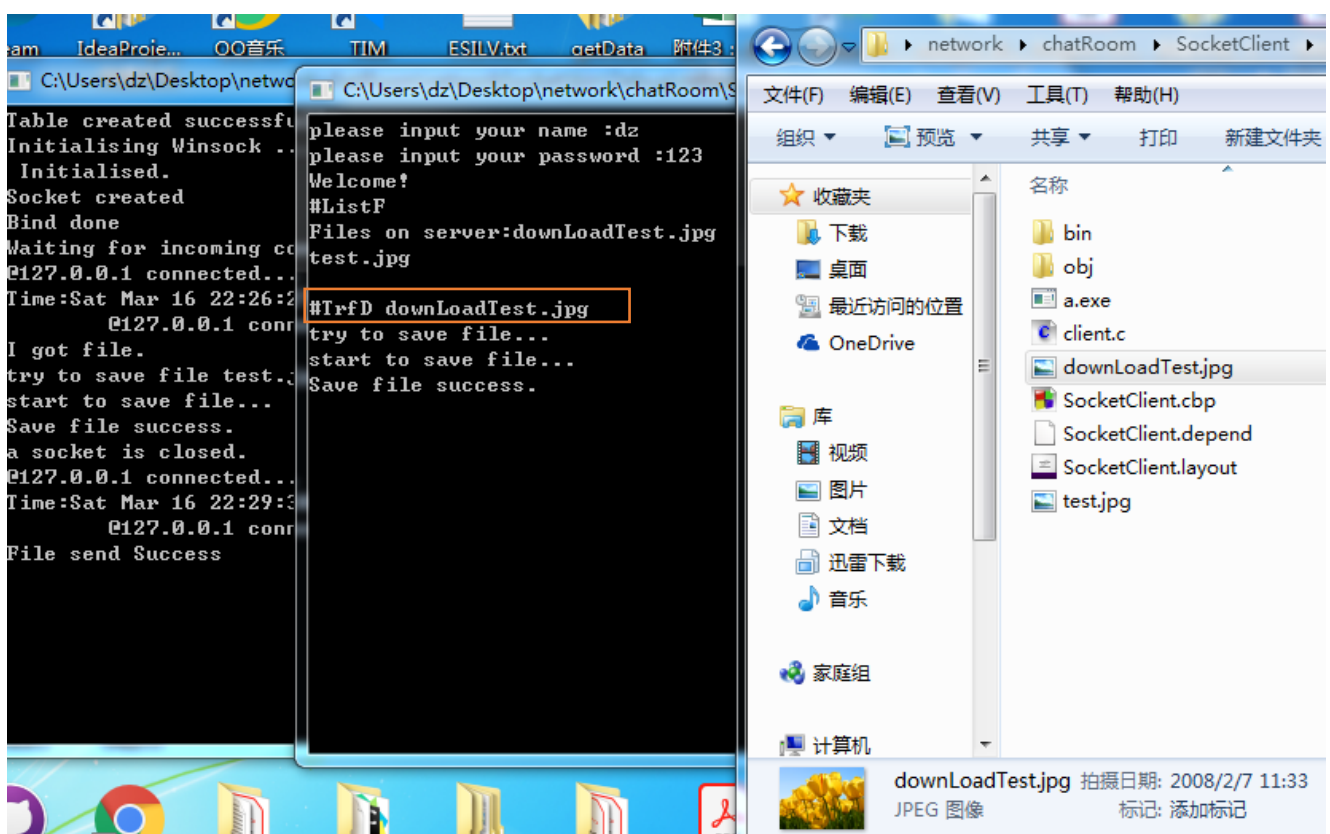
- Client 1 (dz): 'please input your name :dz', 'please input your password :123', 'Welcome!', '#ListU', 'users', followed by a list of users: 'dz'.
- Client 2 (han): 'please input your name :han', 'please input your password :123', 'Welcome!', '#ListU', 'users', followed by a list of users: 'dz', 'han'.
- Client 3 (haipen): 'please input your name :han', 'please input your password :123', 'Welcome!'.

(6)、Transfer files (jpeg).

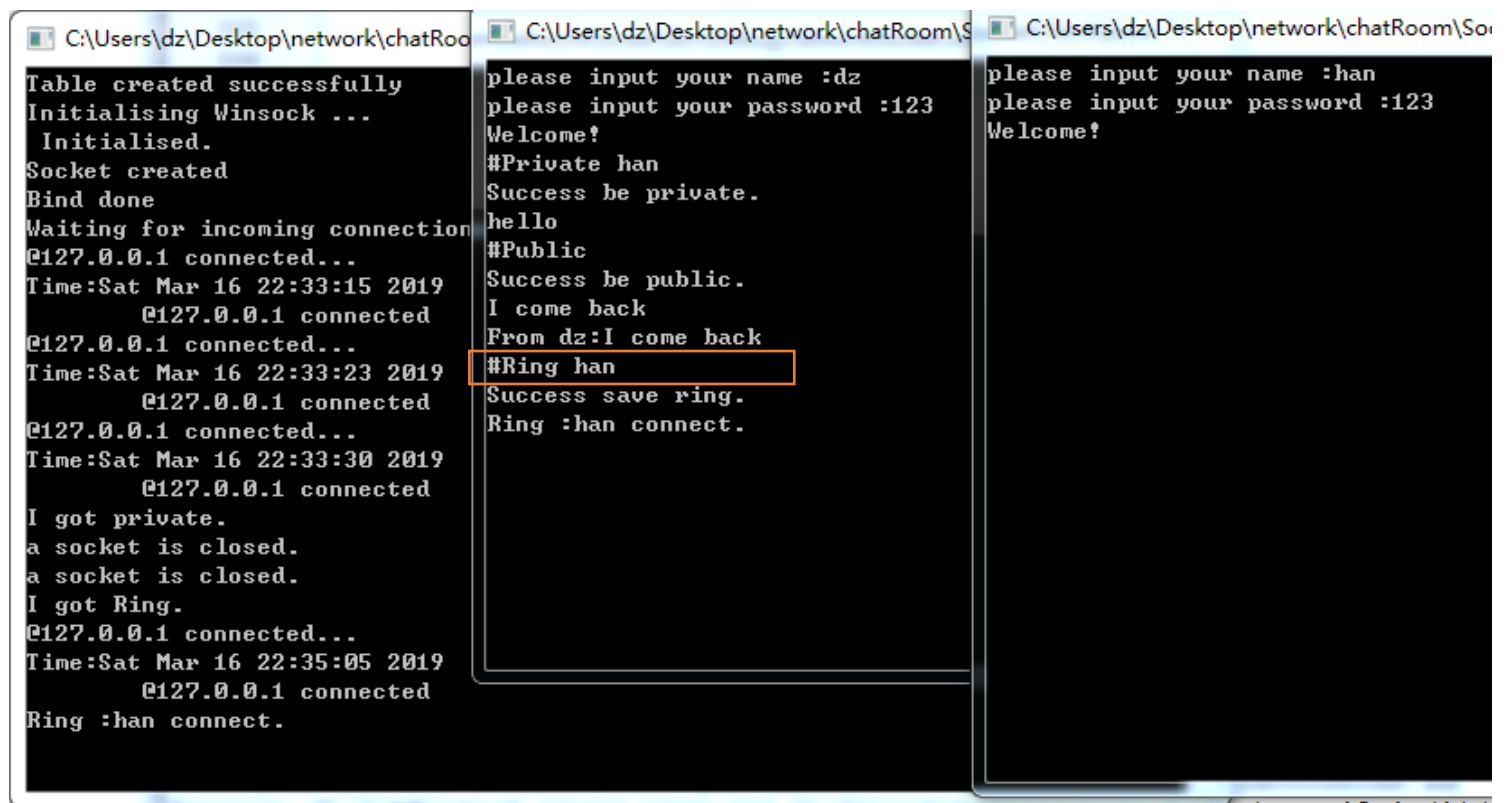
(List files in a server by command “#ListF” and transfer Upload file in a server by command “#TrfU”)



(transfer Upload file in a server by command “#TrfD”)



(7)、Notification if user is connect. (#Ring<user>)



The image shows three terminal windows illustrating the chat room functionality. The first window on the left is the server console, showing the process of creating a table, initializing Winsock, and waiting for connections. It logs several successful connections from 127.0.0.1. The middle window is a client console where a user named 'dz' logs in with password '123', switches to private mode, and sends a message to 'han'. The third window on the right shows the server's response to the client's message, indicating that the message was received and the user 'han' is now connected in private mode.

```

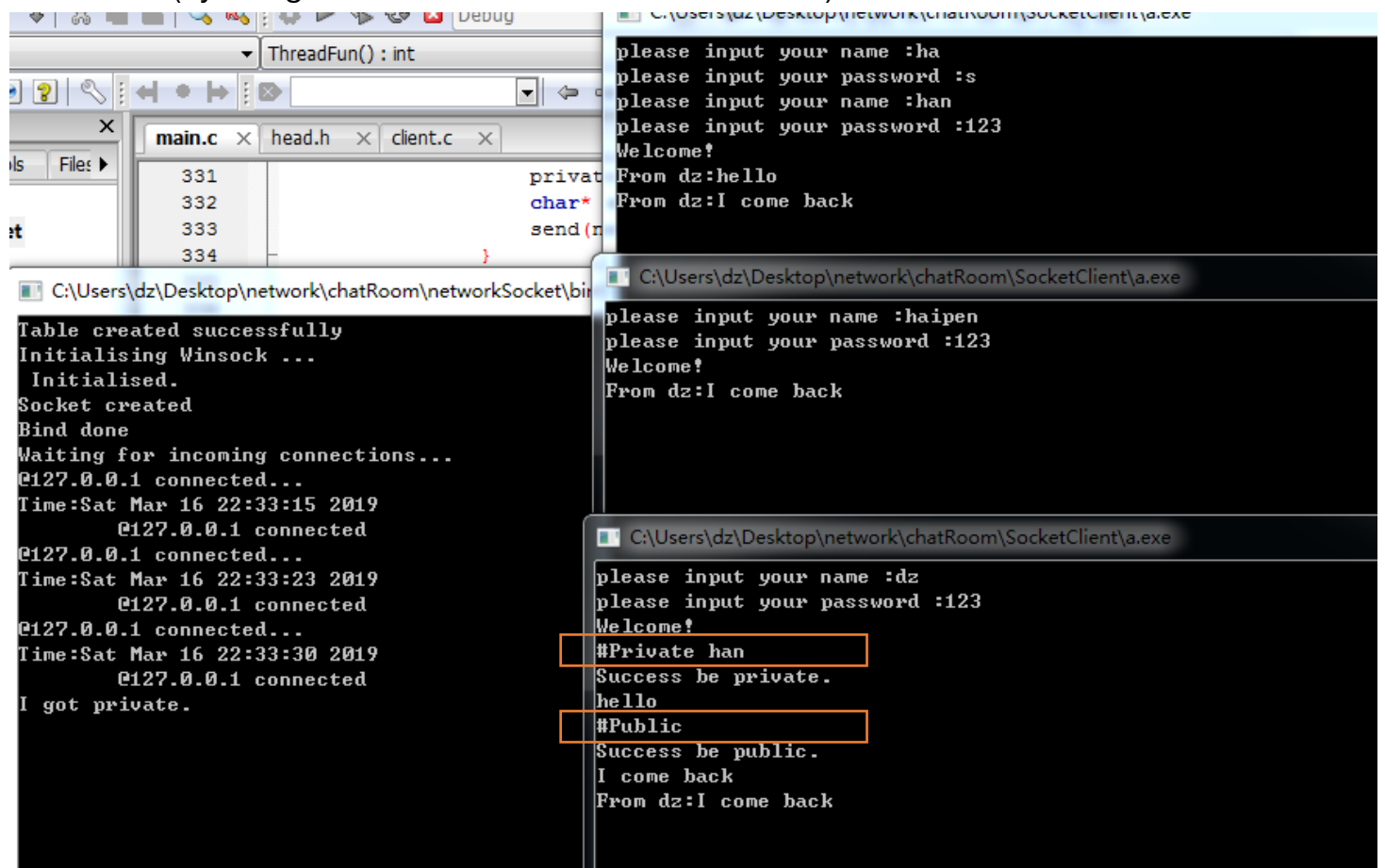
C:\Users\dz\Desktop\network\chatRoom>
Table created successfully
Initialising Winsock ...
Initialised.
Socket created
Bind done
Waiting for incoming connection...
@127.0.0.1 connected...
Time:Sat Mar 16 22:33:15 2019
    @127.0.0.1 connected
@127.0.0.1 connected...
Time:Sat Mar 16 22:33:23 2019
    @127.0.0.1 connected
@127.0.0.1 connected...
Time:Sat Mar 16 22:33:30 2019
    @127.0.0.1 connected
I got private.
a socket is closed.
a socket is closed.
I got Ring.
@127.0.0.1 connected...
Time:Sat Mar 16 22:35:05 2019
    @127.0.0.1 connected
Ring :han connect.

C:\Users\dz\Desktop\network\chatRoom\SocketClient>
please input your name :dz
please input your password :123
Welcome!
#Private han
Success be private.
hello
#Public
Success be public.
I come back
From dz:I come back
#Ring han
Success save ring.
Ring :han connect.

C:\Users\dz\Desktop\network\chatRoom\SocketClient>
please input your name :han
please input your password :123
Welcome!

```

(8)、Commute to private/public.
(by using command “#Private” and “#Public”)



This block contains a screenshot of a code editor and three terminal windows. The code editor shows the implementation of the private/public mode switching in the client code (client.c), with lines 331-334 defining the 'privat' variable and the 'send' function call. The three terminal windows show the client's interaction with the server. The first client window shows a user named 'ha' logging in and sending a message. The second client window shows a user named 'haipen' logging in and sending a message. The third client window shows a user named 'dz' logging in, switching to private mode, sending a message, switching back to public mode, and sending another message. The server console on the left shows the corresponding log messages for these interactions.

```

main.c | head.h | client.c
331     privat
332     char*
333     send(n
334 }

C:\Users\dz\Desktop\network\chatRoom\networkSocket\bin>
Table created successfully
Initialising Winsock ...
Initialised.
Socket created
Bind done
Waiting for incoming connections...
@127.0.0.1 connected...
Time:Sat Mar 16 22:33:15 2019
    @127.0.0.1 connected
@127.0.0.1 connected...
Time:Sat Mar 16 22:33:23 2019
    @127.0.0.1 connected
@127.0.0.1 connected...
Time:Sat Mar 16 22:33:30 2019
    @127.0.0.1 connected
I got private.

C:\Users\dz\Desktop\network\chatRoom\SocketClient\1.a.exe>
please input your name :ha
please input your password :s
please input your name :han
please input your password :123
Welcome!
From dz:hello
From dz:I come back

C:\Users\dz\Desktop\network\chatRoom\SocketClient\1.a.exe>
please input your name :haipen
please input your password :123
Welcome!
From dz:I come back

C:\Users\dz\Desktop\network\chatRoom\SocketClient\1.a.exe>
please input your name :dz
please input your password :123
Welcome!
#Private han
Success be private.
hello
#Public
Success be public.
I come back
From dz:I come back

```

(9)、Client in Linux.

The functions can be achieved

```
cuihaipeng — cb_console_runner DYLD_LIBRARY_PATH=: ~/Documents/codeblocks_projects/networkSoc...
please input your name :dz
please input your password :123
Welcome!
Open the reciving thread
hello
From dz:hello
#Help
-----
#Exit    Exit
#Help    List command
#ListU   List User in server
#ListF   List files in server
#TrfU <filename>      transfert Upload file in a server
#TrfD <filename>      transfert Download file in a server
#Private <Username>    commute to private
#Public      commute to public
#Ring <username>      notification if user is connect
-----
#ListU
users
-----
dz
-----
#ListF
a.jpg
b.txt

#Private dz
Success be private.
#Public
Success be public.
#Ring chp
Success save ring.
#TrfD b.txt
try to save file...
start to save file...
Save file success.
#TrfU /Users/cuihaipeng/Desktop/ahp.txt
#Exit
sh: pause: command not found

Process returned 0 (0x0)    execution time : 515.444 s
Press ENTER to continue.
```