

Linux下C/S架构群聊系统实现

项目题目及功能

本项目是报备获批后的自主选题。功能与第一个题目：类似QQ的聊天系统有所不同。是一个Linux下C/S架构群聊系统，可以实现的功能有

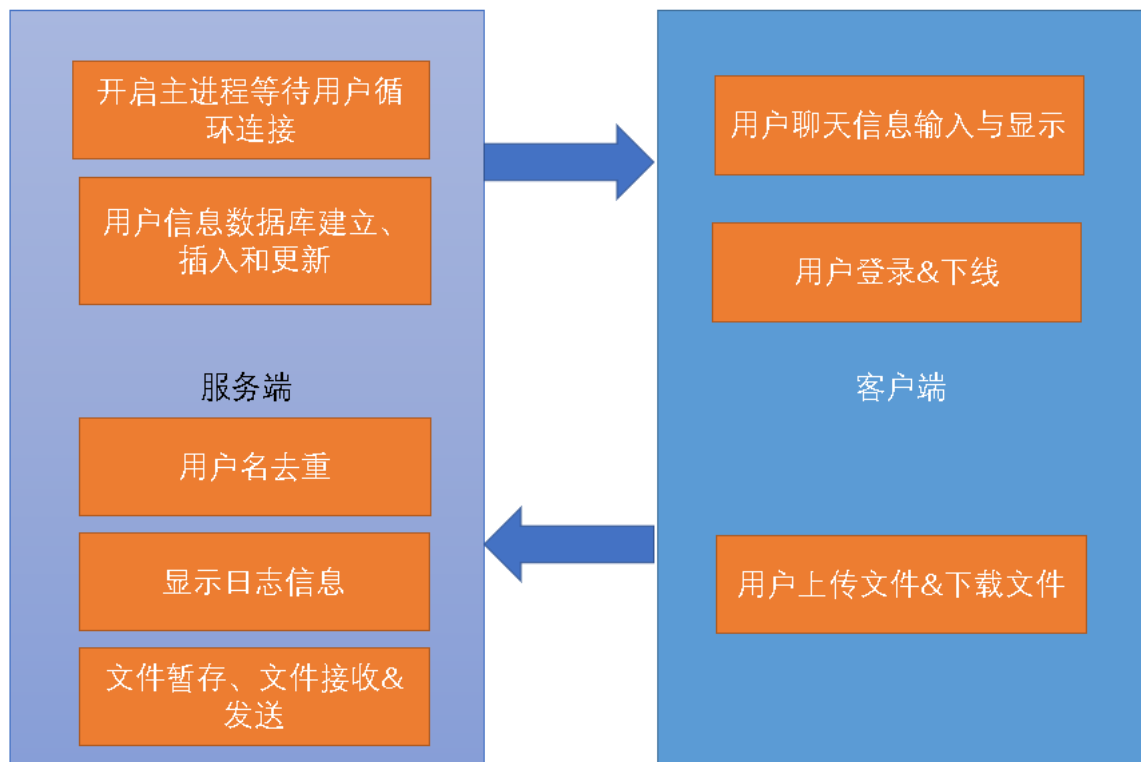
- 用户登录，登出
- 群聊信息发送
- 群聊中文件传输
- 在线人数查看
- 其他小功能
 - 避免用户名重复
 - 用户信息数据化存储

项目组人员及任务分工

个人独立完成

项目整体设计框架图及说明

这个C/S架构是通过一种类似TCP并发服务器编程模型实现的，服务器侦听客户机，为每个客户机创建一个线程。整体设计框架如下：

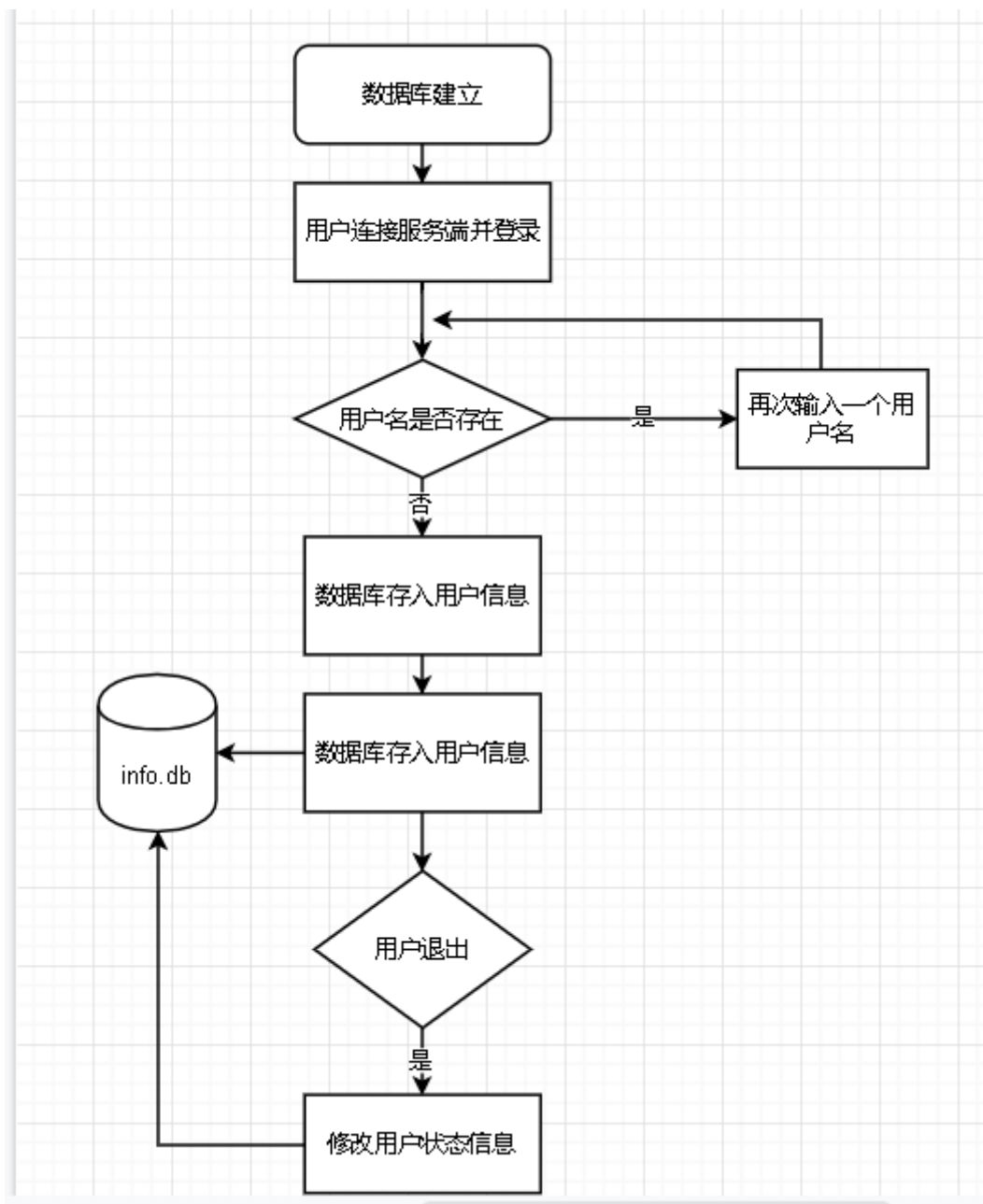


打开一个服务端，首先创建一个数据库，执行listen函数；当有新的客户端打开后，服务端通过accept函数获取新的socket,开启一个与process函数子线程；用户登录时，可以判断用户名是否已经被注册过了，做到用户名去重；在用户发送聊天信息和上传&下载文件时，服务器端会打印相关日志信息；用户上传的文件会在服务端暂存，其他用户可以下载暂存在服务器端的文件。

用户端子进程建立，用户首先需要使用用户名和密码登录，触发服务端在数据库插入相关信息；用户端可以消息和在终端界面看到消息（消息是英文）；可以上传文件到服务端，文件在服务端暂存；可以下载服务端暂存的文件；用户最后可以退出，退出会触发服务端修改数据库中对应的在线状态。

项目分模块设计说明、流程图

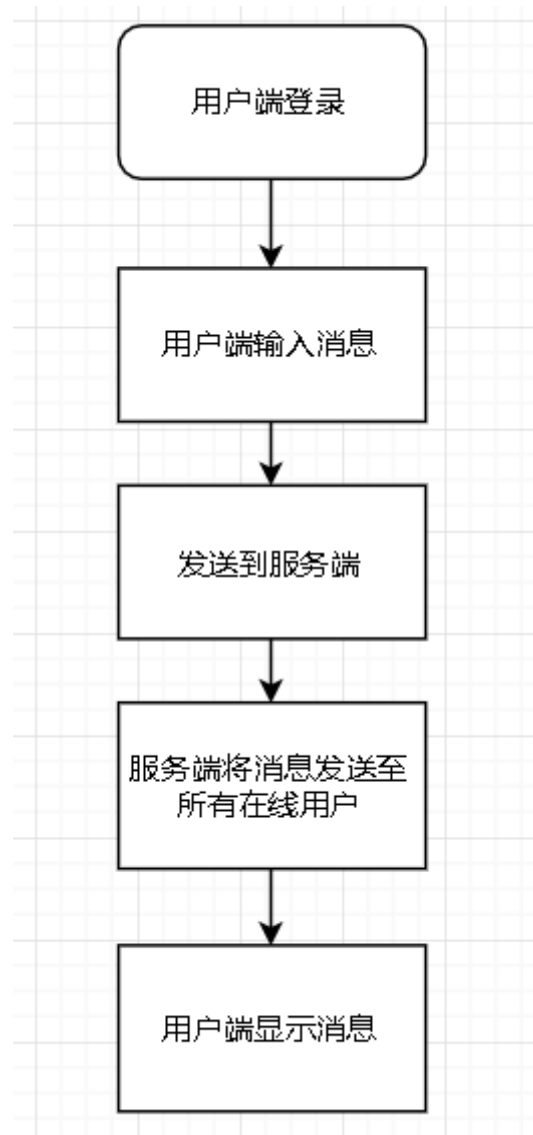
数据库建立与维护&用户登录与退出



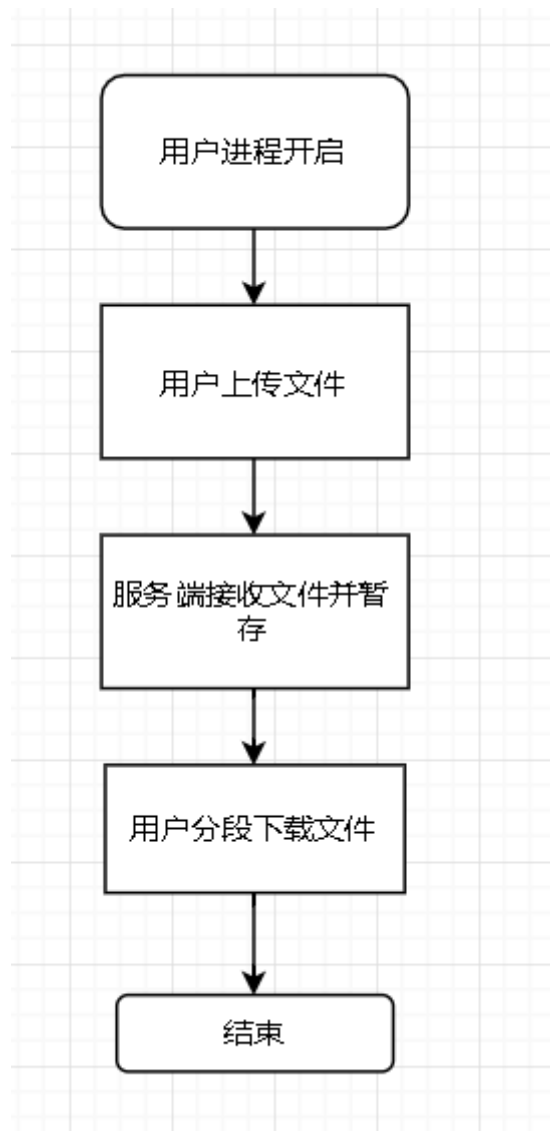
首先，在主进程刚开启时，就新建/打开了一个数据库.用户端打开，子进程建立后，用户输入用户名和密码；如果用户名已存在，就会提示重新输入；否则将用户名，密码，用户状态在线，还有socketaddr作为一条记录插入数据库的相关表中；用户端接收到退出指令后，服务端会触发数据库后台修改相关用户在线状态信息，更新数据库。

聊天信息的发送与显示

当用户在终端输入时，先判断是否是要发送文件，如果不是文件，就将消息包装发送给服务端；服务端接收到消息后，向群聊中所有用户发送该消息，用户端收到将该消息并将消息显示出来

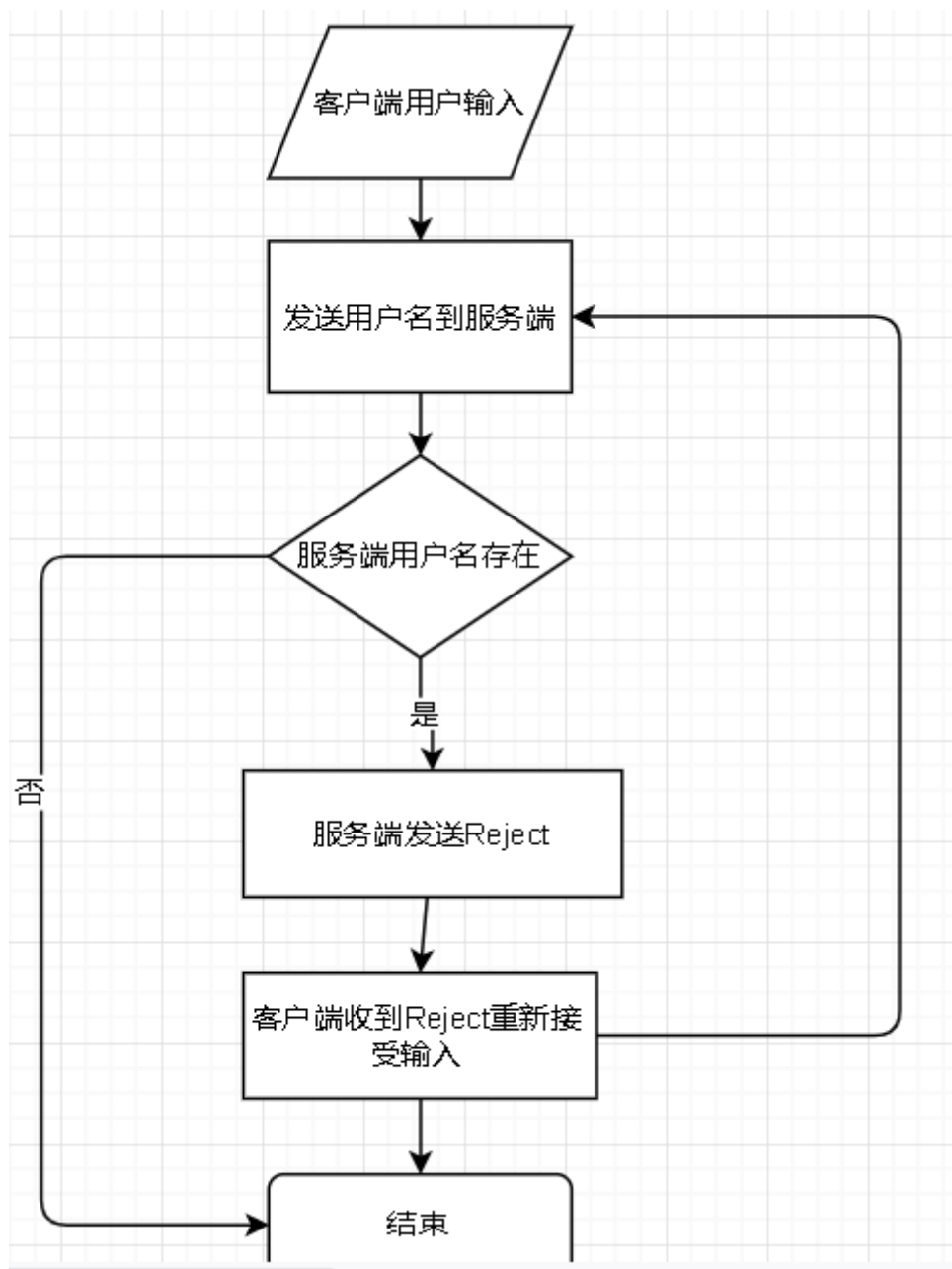


文件发送与接收



当用户输入上传文件指令时，需要输入文件的路径作为参数，然后用户进程读取文件大小后，按照大小固定的分段（1024bytes）上传到服务端缓冲区；其他在线用户输入下载指令，可以从服务端缓冲区分段下载文件。上传操作和下载操作在服务端都有日志信息输出。

用户名避免重复：



客户端接受输入，并将输入内容发到服务端；服务端使用函数usernameExisted()判断用户名是否重复，如果重复就发送Reject到客户端重新接受输入，否则结束。

项目关键数据结构及说明

用户信息**UserTable**表：

```
username TEXT PRIMARY KEY,\n        password TEXT,\n        state INTEGER,\n        socketaddr INTERGER
```

用户名，主键，字符类型；

密码，字符类型；

状态，整数，0或1；

sockaddr:用户连接的套接字描述符

用户信息使用结构体来存储

```
//用户信息的结构体
typedef struct
{
    pthread_t threadNumber;
    int sock;
    char UserName[50];
    char password[50];
    struct sockaddr address;
    int addr_len;
} connection_t;
//默认人数上限为100人
static connection_t conn[100];
```

connection_t结构体中

第一个成员变量threadNumber是用户线程的线程id

第二个成员变量sock是套接字描述符，用来区分连接；

第三个第四个成员变量使用char字符数组存储用户名和密码信息；

第五个成员变量是struct sockaddr类型的变量address，是Linux套接字地址结构，包含用户端的地址信息：名称，端口，IP地址

第六个变量addr_len是地址结构的字节数

定义了一个容量为100的结构体数组conn,规定聊天系统的默认人数上限为100

SendInfo函数发送消息的格式

上传文件：

!! Username send you a file: Filename

其中：

Username指用户名

Filename指文件名

用户退出：

Username quit the chat room, There are Usernumber people in the Chatroom now!!

其中：

Username指用户名

Usernumber指在线人数

用户登录：

Username has entered the Chatroom! There are Usernumber people in the Chatroom now!!**

其中：

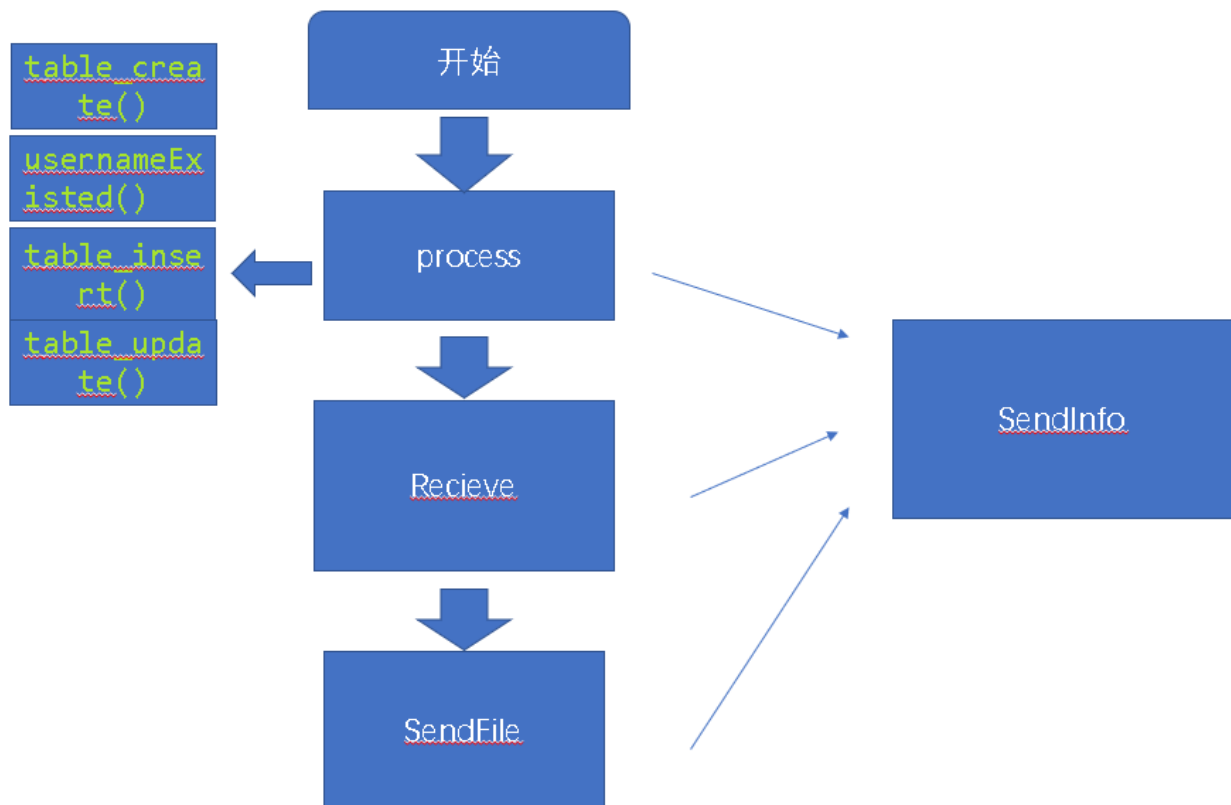
Username指用户名

Usernumber指在线人数

项目关键函数说明及流程图

服务器端代码部分：server.c

服务端代码中函数的调用关系流程图：



process():处理客户端连接

```
//当有一个客户机要连接时，就connect
void * process(void * ptr)
{
    char* buffer;

    int len;
    clientNumber = 0;    //初始化客户端人数
    long addr = 0;
    while(1){
        //等待连接
        if(clientNumber < 100)
        {
            conn[clientNumber].sock = accept(ServerSock,
            &conn[clientNumber].address, &conn[clientNumber].addr_len);
        }
    }
}
```

```

else
    break;
//读取消息长度
read(conn[clientNumber].sock, &len, sizeof(int));
if (len > 0)
{
    //necessary information of a client
    addr = (long)((struct sockaddr_in
*)&conn[clientNumber].address)->sin_addr.s_addr;
    buffer = (char *)malloc((len+1)*sizeof(char));
    buffer[len] = '\0';
    read(conn[clientNumber].sock, buffer, len);
    //提取用户名和密码
    int i=0;
    for(i=0;i<len;i++){
        if(buffer[i] == ' '){
            break;
        }
    }
    strcpy(conn[clientNumber].password, buffer+i+1);
    buffer[i] = '\0';
    strcpy(conn[clientNumber].UserName, buffer);

    //判断用户名是否存在
    if(usernameExisted(conn[clientNumber].UserName,
clientNumber))
        //reject connection and send reject information to the
clients
        {
            send(conn[clientNumber].sock, "Reject", 6, 0);
            --clientNumber;
        }
    else
    {
        //数据库插入值
        //完成用户注册，将用户加入数据库
        table_insert(conn[clientNumber].UserName,
conn[clientNumber].password, 1,conn[clientNumber].sock ,db);

        //向连接的客户端发送成功消息
        send (conn[clientNumber].sock, "You have entered the
chatroom, Start CHATTING Now!\n", 51, 0);
    }
}

```

```

        //向所有在线用户广播发送日志信息
        char mesStart[50] = "User ";
        char mesMiddle[30] = " has entered the Chatroom!\n";
        char mesNumber[50];
        sprintf(mesNumber, "There are %d people in the Chatroom
now!!\n", clientNumber+1);
        strcat(mesStart, conn[clientNumber].UserName);
        strcat(mesStart, mesMiddle);
        strcat(mesStart, mesNumber);
        printf("%s", mesStart);
        SendInfo(mesStart, -1);
        //创建一个线程，执行Recieve函数，传递的实参是用户信息结构体
        pthread_create(&threadClient[clientNumber], 0, Receive,
&conn[clientNumber]);
        conn[clientNumber].threadNumber =
threadClient[clientNumber];
    }
    //释放缓冲区
    free(buffer);
}
clientNumber += 1;//已经用掉的连接数量+1
}
//服务端线程停止
pthread_exit(0);
}

```

说明：在服务端刚开启时，就新建/打开了一个数据库。之后用户端打开，用户输入用户名和密码，这时process函数调用usernameExisted()判断用户名是否存在；如果用户名已存在，就会提示重新输入；否则将用户名，密码，用户状态在线，还有用户端的套接字描述符作为一条记录插入数据库的相关表中。随后拼接生成打印登录成功和聊天室人数的日志信息。最后为每一个连接的用户创建一个线程，这个线程执行Recieve()函数，传递的实参是用户信息结构体。

这个函数主体部分是在一个循环中实现的，循环最大次数为100，这也就是聊天室/群聊中的最大人数。

数据库操作相关

创建数据库和表：`create_table()`

```
sqlite3* create_table(){
    //打开sql文件
    int len;
    len = sqlite3_open(DB_NAME,&db);
    if(len)//如果出错
    {
        /* fprintf函数格式化输出错误信息到指定的stderr文件流中 */
        fprintf(stderr, "Can't open database: %s\n",
sqlite3_errmsg(db));//sqlite3_errmsg(db)用以获得数据库打开错误码的英文描述。
        sqlite3_close(db);
        exit(1);
    }
    else
        printf("你创建了一个叫做 'User' 的sqlite3 数据库!\n");

    char *sql = " CREATE TABLE UserTable(\n
                username TEXT PRIMARY KEY,\n
                password TEXT,\n
                state INTEGER,\n
                socketaddr INTERGER);" ;

    //第五个参数储存error msg
    //建表
    sqlite3_exec(db,sql,NULL,NULL,&zErrMsg);
    return db;
}
```

说明：在这个函数中，创建一个名为info.db的数据库，并在返回值中返回它的指针类型变量。需要用到sqlite3的头文件，sqlite3的API函数**sqlite3_open()**。其中这个API的第一个参数是要创建的数据库的名字，在这里我们使用的是宏定义中的info.db。

sqlite3_exec()函数的第二个参数是可以执行的字符串类型sql语句。在这里就是执行的CREATE TABLE(建表)操作。

table_insert():将用户信息插入表

将注册用户的用户名，密码，用户是否在线，通信标识符等作为一条信息插入数据库

```
void table_insert(char *username,char *password,int state,int
socket,sqlite3 * db){
    char *sqlstr;
    sqlstr = sqlite3_mprintf("insert into UserTable values
('%s','%s',%d,%d)", username, password,state,socket);
    if (sqlite3_exec(db, sqlstr, NULL, NULL, &zErrMsg) != 0)
    {
        printf("error : %s\n", sqlite3_errmsg(db));
    }
}
```

这里用到了sqlite3中的插入操作。其中sqlite3_exec()要执行的sql语句是通过一个叫做sqlite3_mprintf() API来完成的拼接，将函数形参中的变量传入sql字符串语句中。如果操作出现错误，可以打印错误。

table_update(): 用户信息的更新

```
void table_update(char *username,sqlite3 * db){
    char *sqlstr;
    sqlstr = sqlite3_mprintf("UPDATE UserTable SET state = 0 WHERE
username = '%s'",username);
    if (sqlite3_exec(db, sqlstr, NULL, NULL, &zErrMsg) != 0)
    {
        printf("error : %s\n", sqlite3_errmsg(db));
    }
}
```

当用户退出时，通过该函数，可以将对应用户名的那条记录中state(在线状态)字段设为0，表示用户下线

提示信息的发送

SendInfo()

```
int SendInfo(void* Info, int exception)
{
    char *info = Info;
    for(int i = 0; i < 100; ++i){
        //send to the client that exists and doesn't quit room
        if(conn[i].addr_len != -1 && conn[i].addr_len != 0 &&
conn[i].sock != exception){
            if(send (conn[i].sock, info , strlen(info) + 1, 0) == -1)
                printf("error occured, send to %s fail",
conn[i].UserName);
            if(fileDistributing == 0)
                printf("--- [%s] to [%s] 发送成功!\n", info,
conn[i].UserName);
        }
    }
    return 0;
}
```

说明：这个函数第一个参数Info就是要分发的信息，它是一个字符类型的指针。这个函数的功能是可以将一个用户的消息、状态分发到其他所有在线用户那里。这个操作通过一个循环遍历来实现，通过按序遍历有100个结构体的数组，如果某个结构体中addr_len这个描述用户端的地址结构字节数的变量值有效，而且用户端的套接字sock也有有效值，说明索引i对应的用户在线，继而服务端用send向其发送消息。

同时还可以在服务中断打印消息发送成功的日志信息。

接收和处理消息

Recieve(): 接收客户端输入，进行相应的处理

```
//用来处理与一个用户的通信
void* Receive(void* clientStruct)
{
    connection_t* clientInfo = (connection_t *)clientStruct;
    while(1)
    {
        //如果服务器在发送文件，就不再读取缓冲区数据
```

```

if(fileDistributing) continue;
//从客户端读取数据到缓冲区
char *Buffer;
int messageLen = 0;
read(clientInfo->sock, &messageLen, sizeof(int));
//消息长度有效
if(messageLen > 0)
{
    Buffer = (char *)malloc((messageLen+1)*sizeof(char));
    //从缓冲区读取消息
    read(clientInfo->sock, Buffer, messageLen);
    //输入的第一个字符必须是冒号
    if(Buffer[0] != ':') continue;
    Buffer[messageLen] = '\0';
    //当客户端收到退出信息
    if( Buffer[1] == 'q' && Buffer[2] == '!' )
    {
        //拼接退出的消息并把用户从聊天室移除
        char quit[] = " quit the chat room\n";
        char quitMessage[50];
        char quitNumber[50];
        quitMessage[0] = '\0';
        //更新数据库, 有用户退出
        table_update(clientInfo->UserName,db);
        //打印聊天室人数
        sprintf(quitNumber, "There are %d people in the Chatroom
now!!\n", --clientNumber);
        strcat(quitMessage, clientInfo->UserName);
        strcat(quitMessage, quit);
        strcat(quitMessage, quitNumber);
        //把这个用户退出的消息发送给所有用户端
        SendInfo(quitMessage, -1);
        //设置使得用户信息结构体无效
        clientInfo->addr_len = -1;
        //把用户线程关掉
        pthread_exit(&clientInfo->threadNumber);
    }
    //从客户端收到上传文件的信息
    else if ( Buffer[1] == 'f' && Buffer[2] == 'w')
    {
        char sign[] = "!!";
        char file[] = " send you a file: ";
        char fileMessage[50];

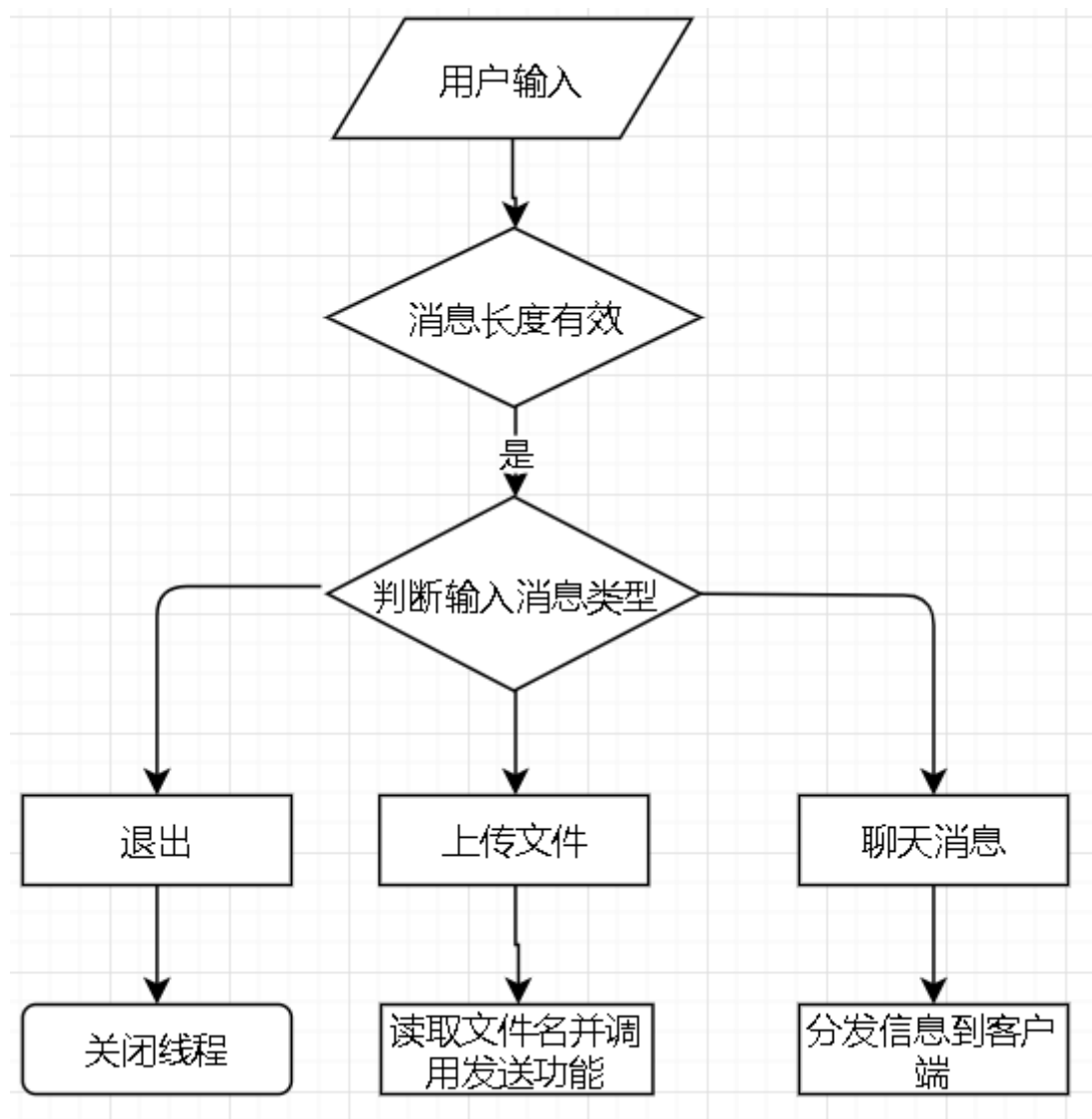
```

```

char Filename[FILE_NAME_MAX_SIZE];
fileMessage[0] = '\0';
strcat(fileMessage, clientInfo->UserName);
strcat(fileMessage, file);
//从缓存区读取文件名
for(int t = 4; t < messageLen-1; t++)
    Filename[t-4] = Buffer[t];
Filename[messageLen-5]='\0';
//拼接字符串，发送 文件传送 相关的日志信息
strcat(fileMessage, Filename);
strcat(sign, fileMessage);
SendInfo(sign, -1);
//函数参数为要上传文件的那个用户
SendFile(clientInfo);
}
else{
    //聊天信息发送
    //拼接信息
    char begin[] = " says";
    char messageDistribute[200];
    messageDistribute[0] = '\0';
    strcat(messageDistribute, clientInfo->UserName);
    strcat(messageDistribute, begin);
    strcat(messageDistribute, Buffer);
    //向所有用户发送聊天信息
    SendInfo(messageDistribute, -1);
}
free(Buffer);
}
else
    continue;
}
}

```

流程图



文件发送

SendFile():服务端向用户端发送文件

```
//该函数实现一个用户向其他用户发送文件
int SendFile(connection_t* clientStruct)
{
    int size;
    int filesize;
    char buffer[1024];
    int len;
    fileDistributing = 1;
    //读取文件大小
    read(clientStruct->sock, &size, sizeof(int));
    read(clientStruct->sock, &filesize, sizeof(int));
    //向除了上传文件的用户之外的其他用户分发文件大小信息
    char filesizeString[20];
    char filesizeStrings[2];
```

```

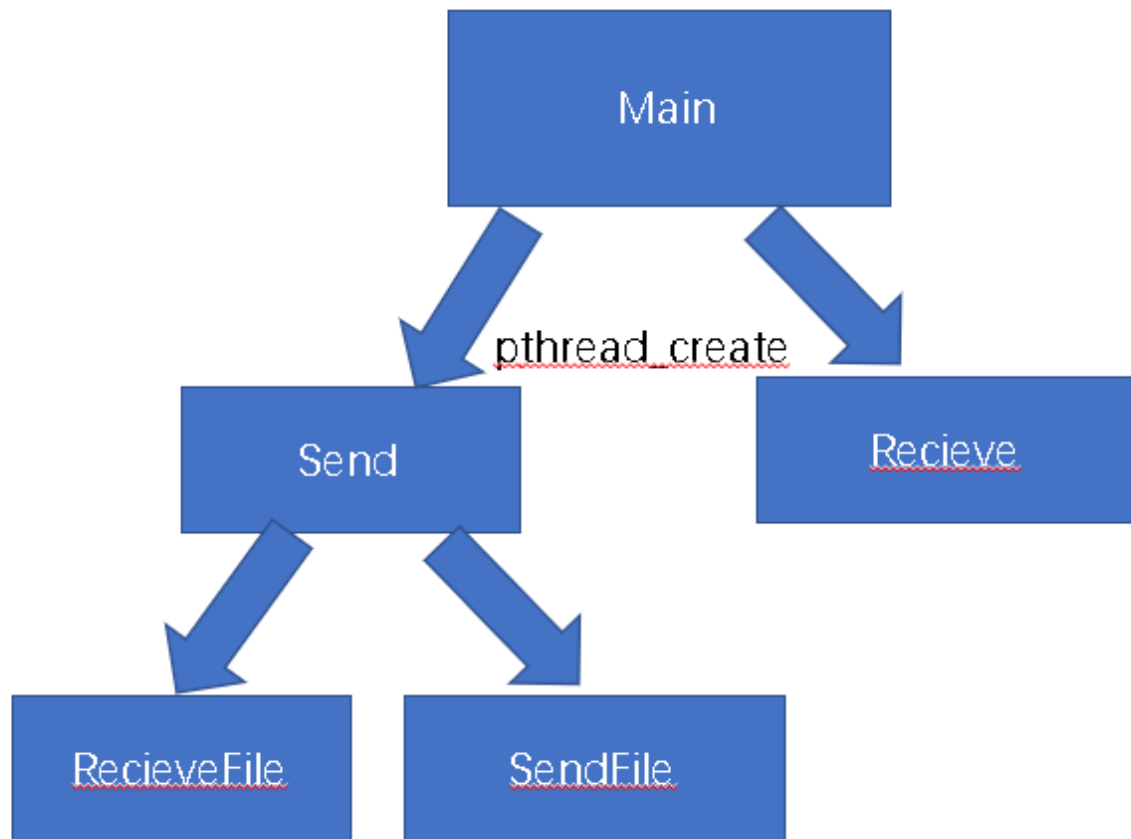
sprintf(filesizeString, "%d", filesize);
sprintf(filesizeStringsSize, "%ld", strlen(filesizeString));
SendInfo(filesizeStringsSize, clientStruct->sock);
SendInfo(filesizeString, clientStruct->sock);
//分部分发送文件，直到末尾
for(int i=0; i < filesize/1024+1; ++i)
{
    read(clientStruct->sock, &len, sizeof(int));
    read(clientStruct->sock, buffer, len);
    printf("收到了 %ld 字节\n", strlen(buffer));
    //发送成功信息
    SendInfo(buffer, clientStruct->sock);
    printf("发送第 %d 部分成功!\n", i + 1);
    //刷新缓冲区
    bzero(buffer, BUFFER_SIZE);
}
printf("成功发送所有部分!\n");
//标志位设为零
fileDistributing = 0;
return 0;
}

```

读取文件大小filesize之后，分了filesize/BUFFER_SIZE+1块发送，每块大小1024 bytes。每发送一个部分，在服务端打印成功的日志信息。

用户端代码：client.c

client.c函数调用关系：



主函数main():

```
int main ()
{
    int sockfd, n, MessageSize, PasswordSize;
    pthread_t threadSend;
    pthread_t threadReceive;
    struct sockaddr_in serv, cli;
    char rec[1000];
    char send[80];
    char serAddress[80];

    //输入客户端IP
    //如果直接回车，代表默认
    printf("输入客户端IP(type <Enter> to use default): ");
    fgets(serAddress, sizeof(serAddress), stdin);
    if(serAddress[0] == '\n')
    {
        //默认address
    }
}
```

```

        strcpy(serAddress, "127.0.0.1\n");
    }
    serAddress[strlen(serAddress) - 1] = '\0';

    //input UserName
    Start: printf("输入用户名和密码，用空格隔开： " );
    //用户名
    //fgets函数读取到回车才会停止
    fgets(username_and_password, sizeof(username_and_password), stdin);
    username_and_password[strlen(username_and_password) ] = '\0'; //cut
the '\n' ending
    int input_len= strlen(username_and_password);
    //建立套接字
    sockfd = socket (PF_INET, SOCK_STREAM, 0);
    bzero (&serv, sizeof (serv));
    serv.sin_family = PF_INET;
    //8888端口
    serv.sin_port = htons (8888);
    serv.sin_addr.s_addr = inet_addr (serAddress /*server address*/);

    //连接服务端
    if(connect (sockfd, (struct sockaddr *) &serv, sizeof (struct
sockaddr)) == -1)
    {
        printf("connect %s failed\n", serAddress);
        exit(1);
    }

    //把用户名和密码发到server
    write(sockfd, &input_len, sizeof(int));
    write(sockfd, username_and_password, input_len);

    //收到成功连接的信息
    n = read (sockfd, rec, 1000);
    rec[n] = '\0';
    //判断是否收到Reject
    if(rec[0] == 'R')
    {
        //重新接收输入
        rec[0] = '\0';
        printf("用户名已存在，请换一个.\n");
        goto Start;
    }

```

```

}
else
{
    //doesn't been rejected, open threads that are needed

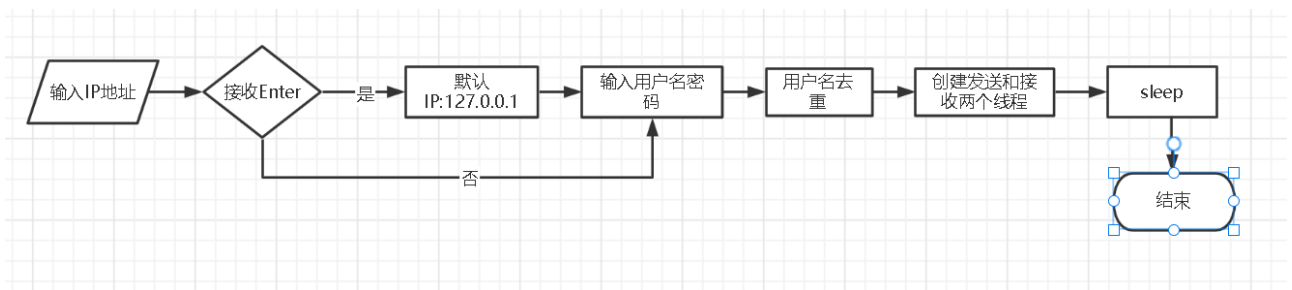
    // //用户名没有重复，就设置密码
    fputs(rec, stdout);
    //开始发送
    pthread_create(&threadSend, 0, Send, &sockfd);
    //然后接收
    pthread_create(&threadReceive, 0, Receive, &sockfd);
}
//让程序维持较长时间
for(int i = 0; i < 99; ++i)
    sleep(9999);

//close and free everything then quit
pthread_exit(&threadSend);
pthread_exit(&threadReceive);
close(sockfd);

return 0;
}

```

流程图：



输入的IP地址可以是实际的IP地址，如果直接按空格，就是默认的IP:127.0.0.1。然后客户端使用套接字建立TCP连接。

之后对输入的用户名发到服务端，如果收到Reject信息，需要重新输入不重复的用户名；否则可以pthread_create创建发送、接收两个线程，分别与Send和Recieve函数绑定；设置很长的睡眠时间，确保主进程不关闭。最后pthread_create关闭两线程后退出。

Send()函数：接收用户端输入

```
//终端接收收入，发送给服务器
void* Send(void* Socket)
{
    char sender[80];
    char Filename[FILE_NAME_MAX_SIZE];
    int *SocketCopy = Socket;
    while(1)
    {
        //如果正在读文件，则跳过
        if(fileReading) continue;
        //接受输入
        fgets(sender, sizeof(sender), stdin);

        //如果要接收文件
        if(sender[1] == 'f' && sender[2] == 's')
        {
            fileReading = 1;
            char destination[50];
            //读取目标文件地址并保存到destination
            for(int i = 4; i < strlen(sender) - 1; ++i)
                destination[i - 4] = sender[i];
            destination[strlen(sender) - 5] = '\0';
            //接收文件，SocketCopy端的将文本内容存入destination文件中
            ReceiveFile(destination, *SocketCopy);
            continue;
        }
        //如果这是一条普通消息，就发送它到服务端
        int messageSize = strlen(sender) + 1;
        write(*SocketCopy, &messageSize, sizeof(int));
        write(*SocketCopy, sender, messageSize);
        //检查这是否是一条退出的消息
        if(strcmp(sender, ":q!\n") == 0)
            exit(1);

        //检查这是否是发送文件
        else if(sender[1] == 'f' && sender[2] == 'w')
        {
            printf("请再次输入文件名(包括地址):\n");
            scanf("%s", Filename);

            //打开文件同时将文件大小发到服务端
```

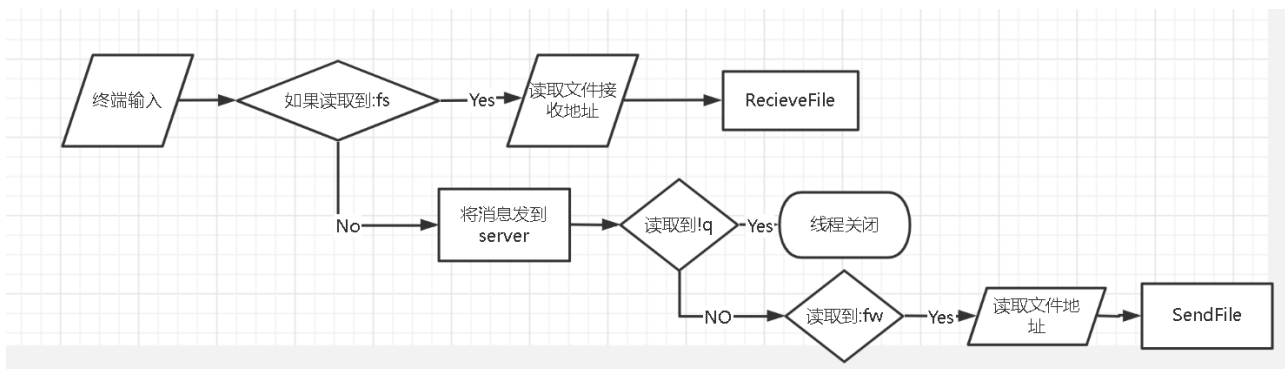
```

        FILE *fp=fopen(Filename, "r");
        fseek(fp, 0L, SEEK_END);
        int Filesize=ftell(fp);
        int intSize = sizeof(int);
        write(*SocketCopy, &intSize, sizeof(int));
        write(*SocketCopy, &Filesize, sizeof(int));

        //发送文件，同时将flag置0
        sendfile( Filename, SocketCopy );
        fileReading = 0;
    }
}
}

```

流程图



SendFile():用户端向服务端上传文件:

```

void sendfile(char* Filename, void* Socket)
{
    int *SocketCopy = Socket;
    char buffer[1025];
    FILE *fp;
    fp = fopen(Filename, "r");

    if(NULL == fp)
    {
        printf("目标文件:%s 不存在\n", Filename);
        return;
    }
    else

```

```

{
    //读取文件，循环发送
    int length = 0;
    while((length = fread(buffer, sizeof(char), BUFFER_SIZE, fp)) >
0)
    {
        write(*SocketCopy, &length, sizeof(int));
        if(write(*SocketCopy, buffer, length) < 0)
        {
            printf("上传文件:%s 失败.\n", Filename);
            break;
        }
        //缓存区置零
        bzero(buffer, BUFFER_SIZE);
    }
}
fclose(fp);
printf("文件:%s 上传成功!\n", Filename);
}

```

客户端先根据文件地址打开要上传的文件，然后读取文件，向服务端循环分块发送，直到末尾，然后打印上传成功消息。

RecieveFile():从服务端下载接收文件存到本地

```

//从服务器端接收文件
void ReceiveFile(char* dest, int Socket)
{
    //be prepared to receive file
    char buffer[BUFFER_SIZE];
    printf("你想把文件存储在 %s\n", dest);
    FILE *fp = fopen(dest, "w");
    if(NULL == fp)
    {
        printf("文件:\t%s 打不开\n", dest);
        exit(1)
    }
    bzero(buffer, BUFFER_SIZE);

    //读取文件大小，把字符串转Int类型

```



```

char filesize[20];
char filesizeStringSize[2];
int L1 = read(Socket, filesizeStringSize, 2);
int L2 = read(Socket, filesize, atoi(filesizeStringSize) + 1);
int filesizeInt = atoi(filesize);

//prepare to receive the file
int length = 0;
int i = 0;
fileReading = 1;

//receiving the file in parts according to file size
while(i < filesizeInt/1024 + 1)
{
    length = read(Socket, buffer, BUFFER_SIZE);
    if(fwrite(buffer, sizeof(char), length - 2, fp) < length - 2)
    {
        printf("文件:\t%s 写入失败\n", dest);
        return;
    }
    printf("成功接收文件第 %d 部分!\n", ++i);
    bzero(buffer, BUFFER_SIZE);
}

//print success message and free neccessary things
printf("从服务器成功收到文件并存入 %s!\n", dest);
fileReading = 0;
fclose(fp);
}

```

先读取要存入的目标文件地址，然后读取文件大小；接着用BUFFER循环分块按顺序读取到末尾，将读取的文本文件内容写入目标文件中。

Receive(): 打印从服务端收到的消息

```

void* Receive(void* Socked)
{
    int *SockedCopy = Socked;
    char Receiver[80];

    while(1){

```

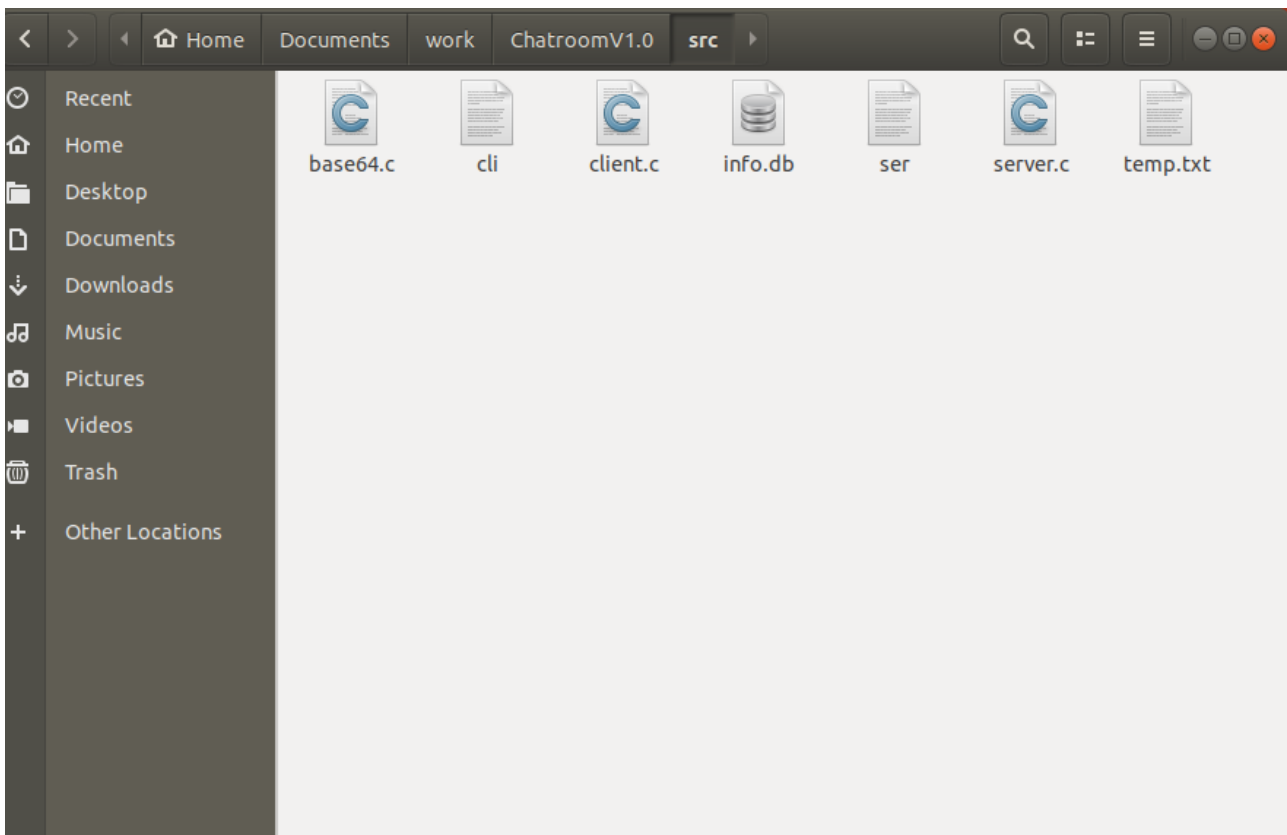
```

        if(fileReading == 1)
            continue;
        //不断地读取消息
        int reveiverEnd = 0;
        //读取的内容保存在Receiver
        reveiverEnd = read (*SocketCopy, Receiver, 1000);
        if(Receiver[0] == '!' && Receiver[1] == '!')
            fileReading = 1;
        Receiver[reveiverEnd] = '\0';
        //输出接收到的内容
        fputs(Receiver, stdout);
        Receiver[0] = '\0';
    }
}

```

这个函数用来接收server传来的文字聊天信息，并输出，来实现群聊功能。

项目文件列表、文件功能说明及项目编译步骤



必须保留

两个源程序文件

server.c是服务器端对应的文件

clinet.c是客户端对应的文件

可删

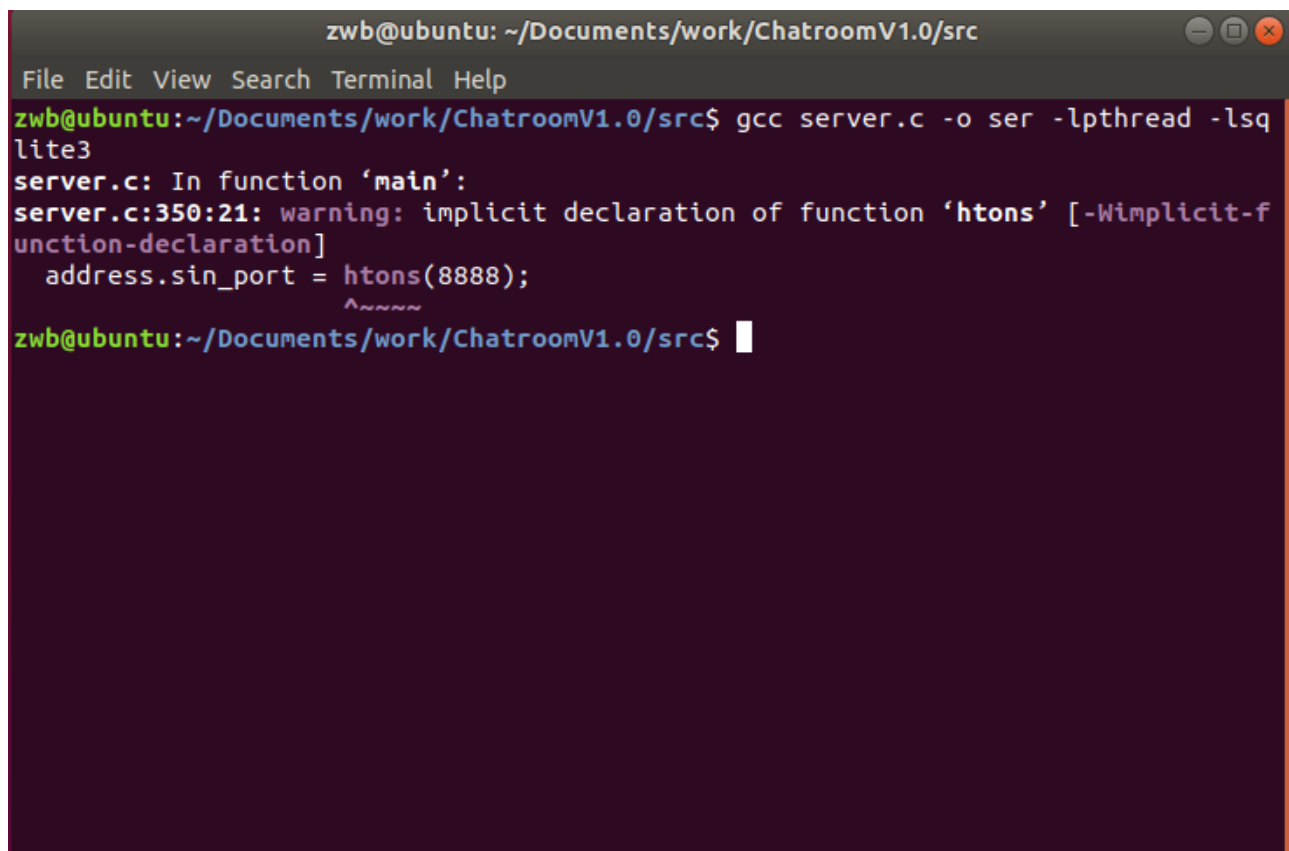
cli 和ser分别是client.c和server.c编译产生的object文件（中间文件）

info.db是sqlite3数据库文件，在ser第一次执行时会默认创建一个.db文件，和一个空表用来存放用户信息

base64.c文件非空，是用来测试文件传输功能的文本类型的文件；temp.txt默认为空，主要为了测试文件接收功能用的。

编译步骤

首先在项目文件下打开终端，先编译server.c文件。因为本项目用到了sqlite3和pthread.h头文件,所以在编译时要加入-lpthread和-lsqlite3选项



```
zwb@ubuntu: ~/Documents/work/ChatroomV1.0/src
File Edit View Search Terminal Help
zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$ gcc server.c -o ser -lpthread -lsqlite3
server.c: In function 'main':
server.c:350:21: warning: implicit declaration of function 'htons' [-Wimplicit-function-declaration]
    address.sin_port = htons(8888);
                      ^~~~~~
zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$
```

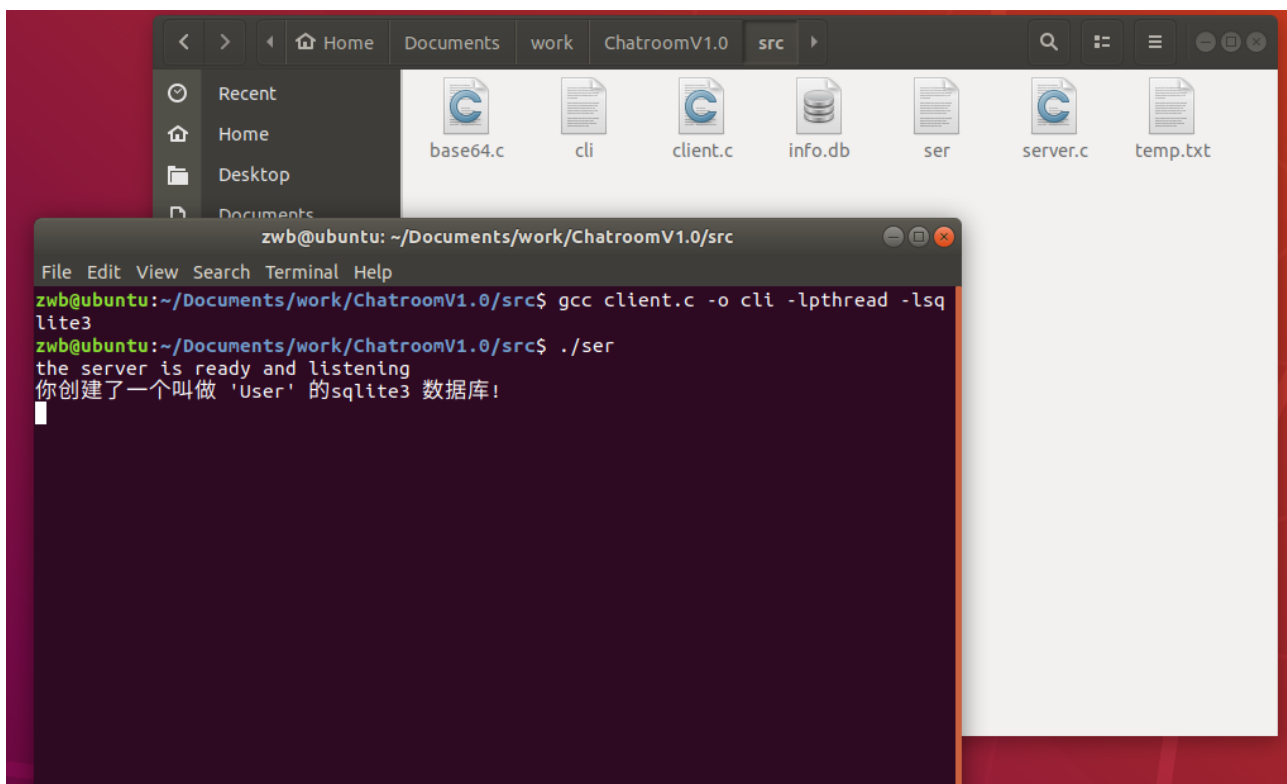
再编译client.c文件

```
zwb@ubuntu: ~/Documents/work/ChatroomV1.0/src
File Edit View Search Terminal Help
zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$ gcc client.c -o cli -lpthread -lsq
lite3
zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$
```

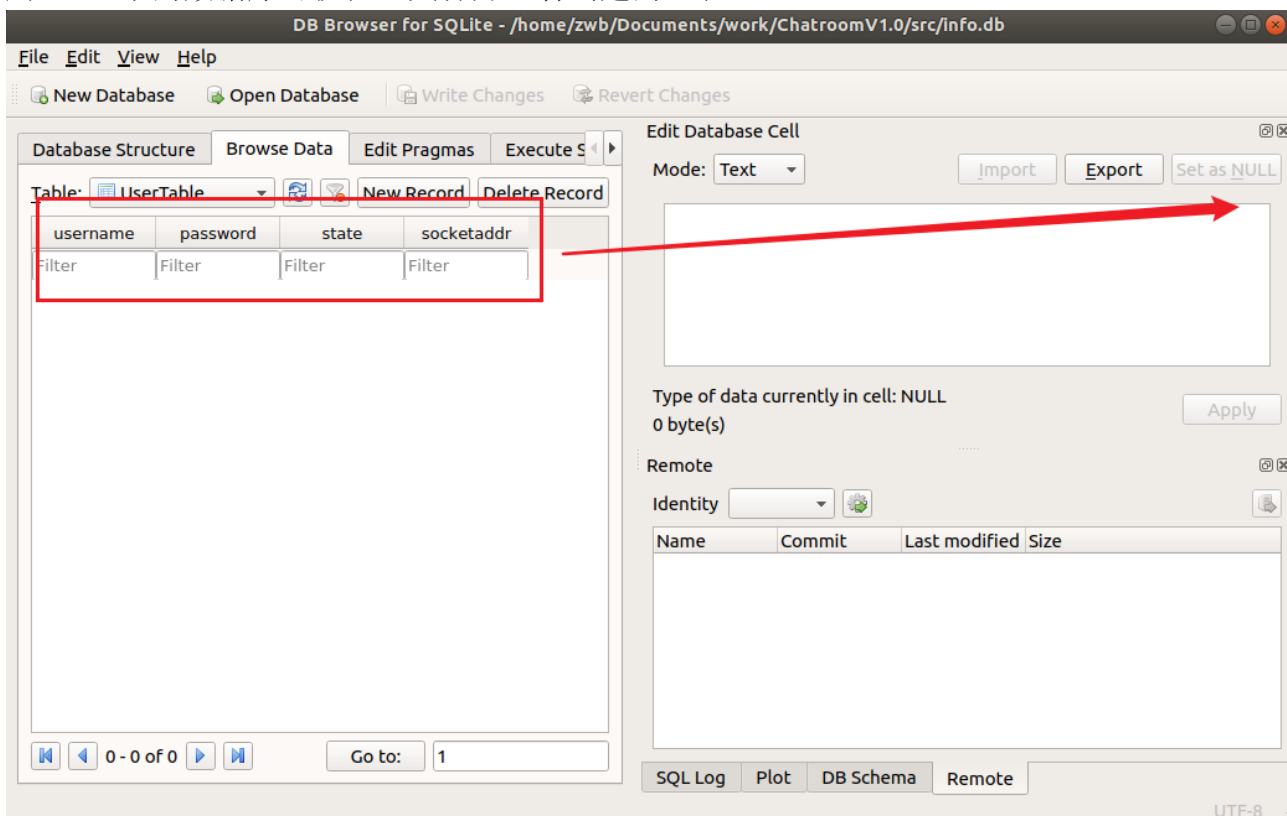
项目演示

运行

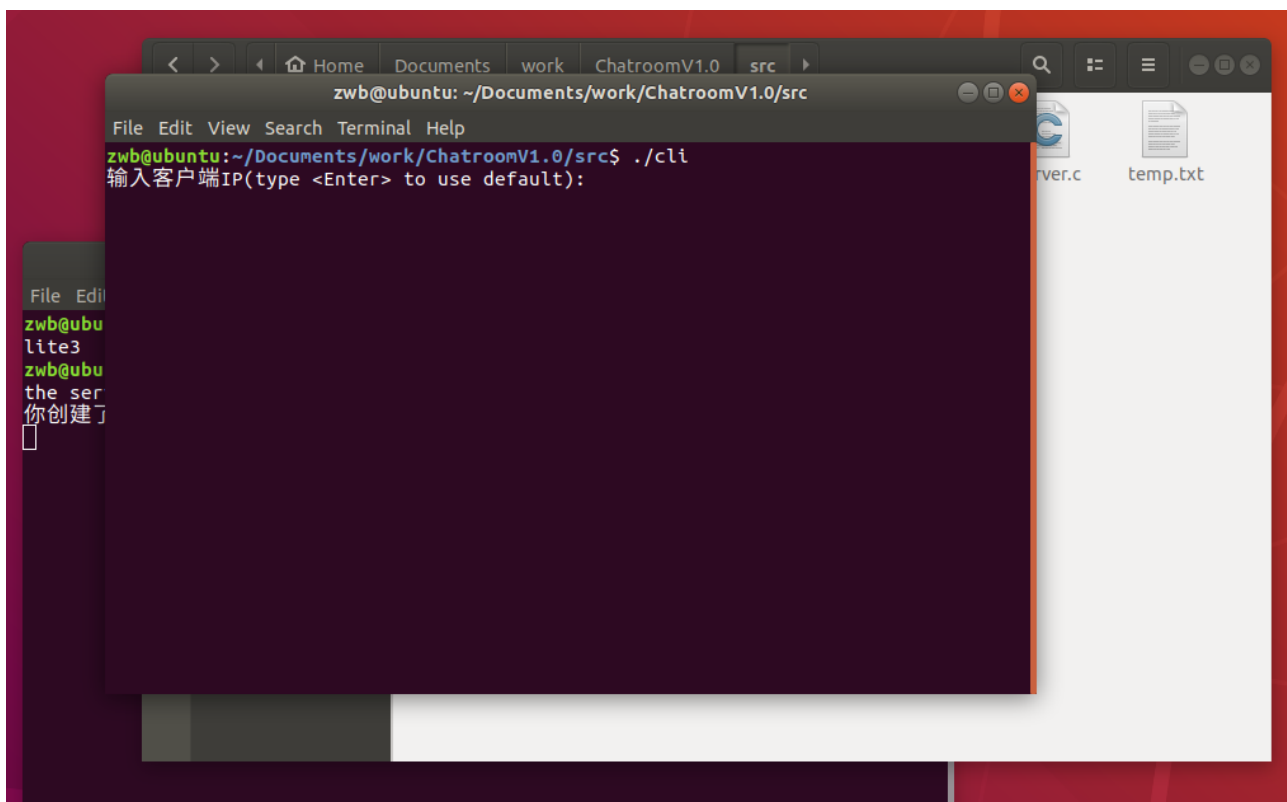
先运行server.c文件编译生成的object文件ser,在命令行输入./ser.发现此时项目文件夹下多了一个info.db的sqlite3数据库文件，这是设计让服务端默认创建的，用于保存用户名，密码，在线状态和socketaddr信息。



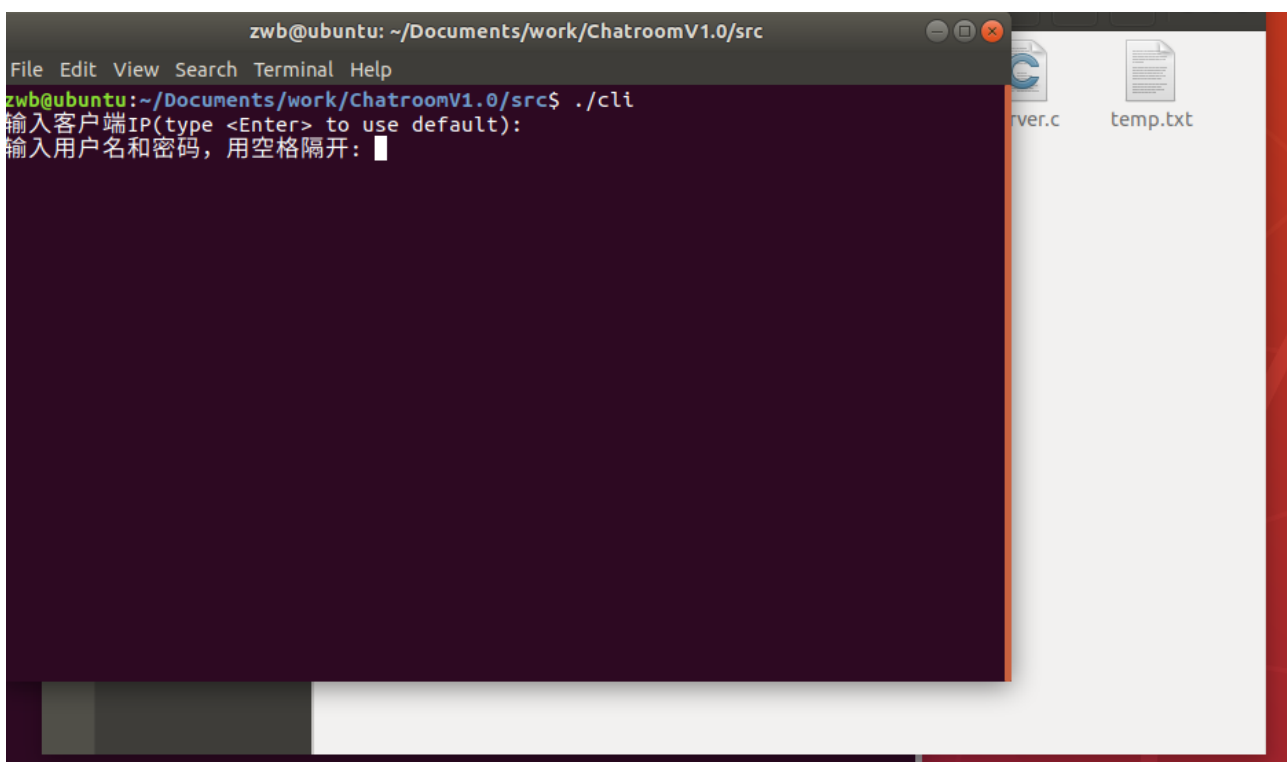
用Linux下的数据库可视化工具打开查看创建的空表User:



然后打开另一个终端，并输入./cli命令，运行客户端程序。每运行一次客户端程序，就有一个用户加入聊天室。



此时可以选择输入IP地址，也可以选择默认的IP地址。回车键代表选择默认（default）的客户端IP地址，即127.0.0.1

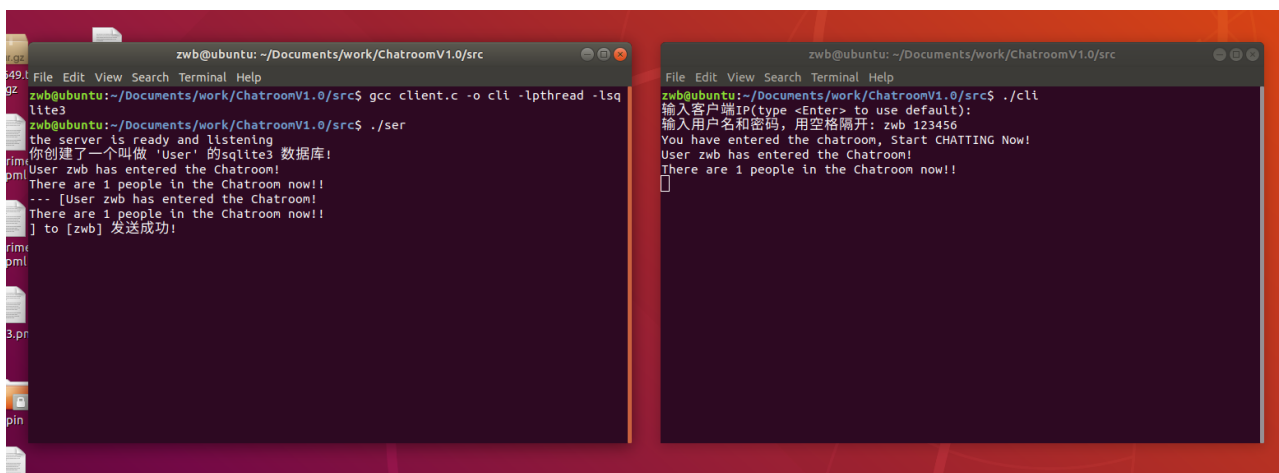


用户登录&退出

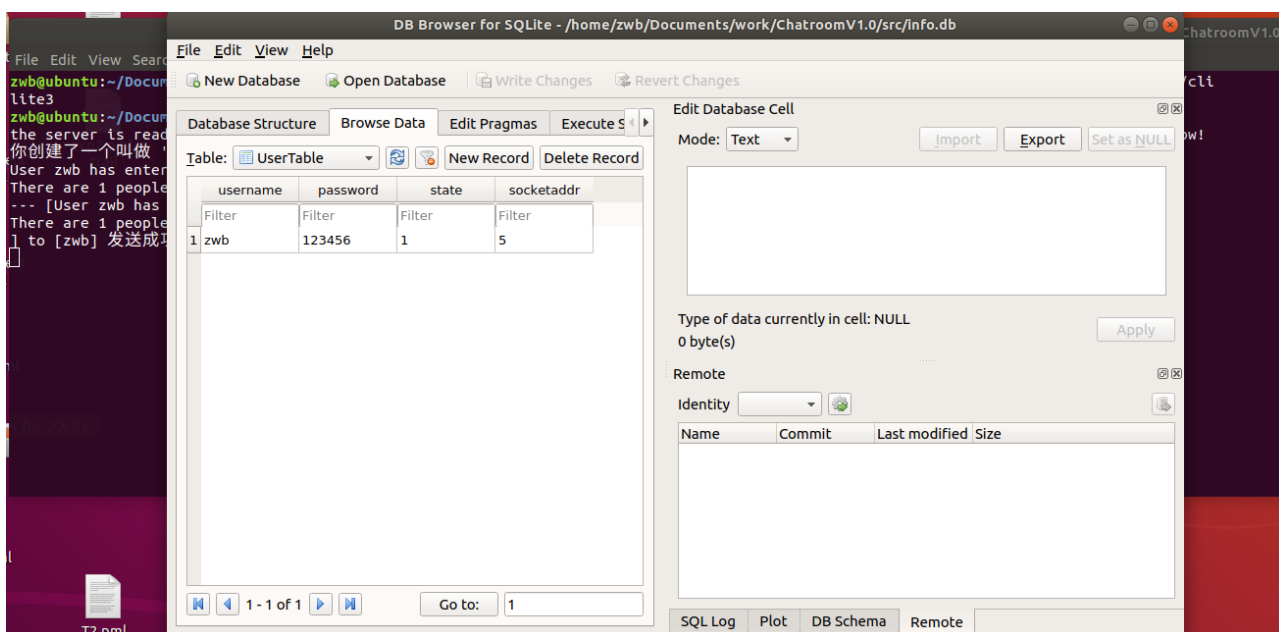
初始化后，以一个用户的登录和退出为例。下面进入用户登录环节：

输入用户名和密码，中间用空格隔开。我选择输入本人姓名缩写zwb,密码设为123456.

右侧客户端对应的终端显示用户zwb已登陆成功，进入聊天室，并显示当前聊天室人数为1人。左侧服务器对应的终端输出一些日志信息或者服务端和客户端之间的交互信息，e.g.消息发送成功



此时查看info.db的User表，可以看到用户名，密码，在线状态(status属性)，socketaddr已经存入了数据库。



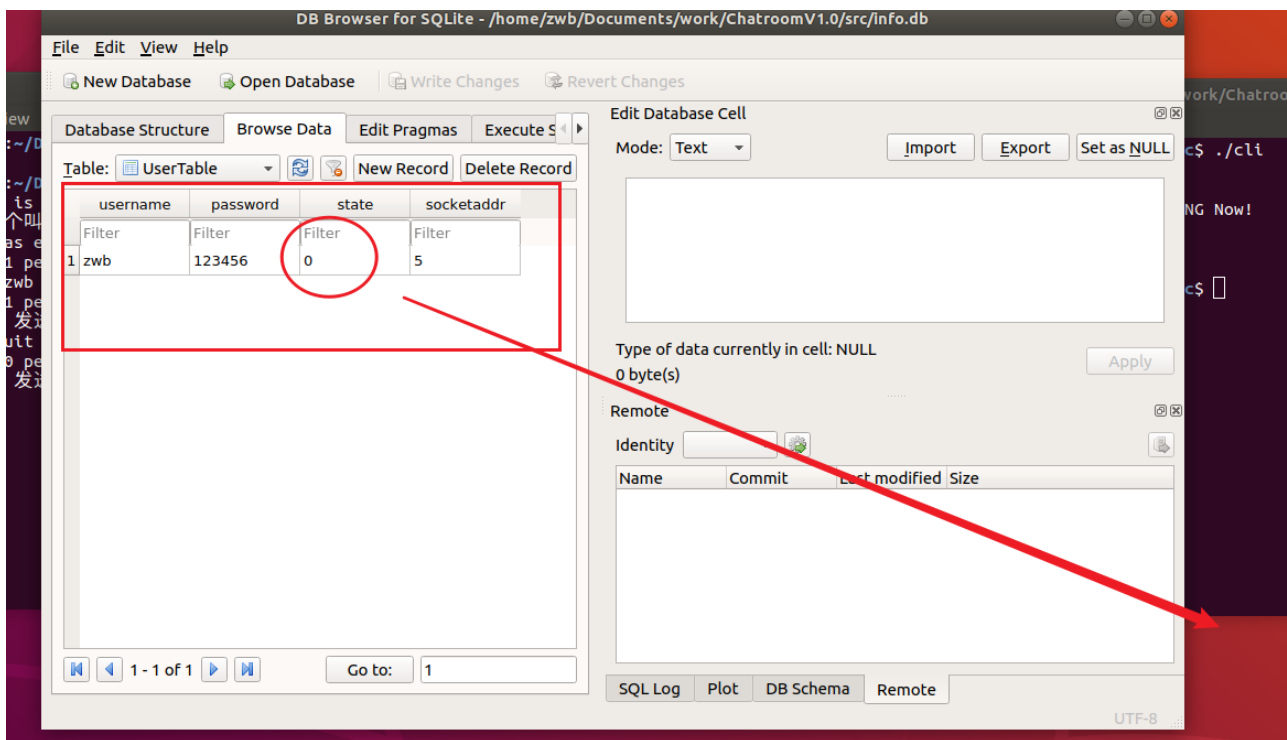
此时，在客户端终端输入:q!命令，用来向服务端请求退出聊天室。可以看大，此时客户端对应的线程终止。

```
zwb@ubuntu: ~/Documents/work/ChatroomV1.0/src
File Edit View Search Terminal Help
zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$ ./cli
输入客户端IP(type <Enter> to use default):
输入用户名和密码, 用空格隔开: zwb 123456
You have entered the chatroom, Start CHATTING Now!
User zwb has entered the Chatroom!
There are 1 people in the Chatroom now!!
:q!
zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$
```

同时, 客户端终端的日志信息更新, 用户zwb退出群聊, 并显示聊天室在线人数有0人。

```
zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$ gcc client.c -o cli -lpthread -lsq
lite3
zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$ ./ser
the server is ready and listening
你创建了一个叫做 'User' 的sqlite3 数据库!
User zwb has entered the Chatroom!
There are 1 people in the Chatroom now!!
--- [User zwb has entered the Chatroom!
There are 1 people in the Chatroom now!!
] to [zwb] 发送成功!
--- [zwb quit the chat room
There are 0 people in the Chatroom now!!
] to [zwb] 发送成功!
```

再次用可视化工具查看info.db数据库User表, 发现zwb的state属性变成了0, 代表已经离线。



群聊发消息

运行两个客户端，登录两个用户，第一个用户名是aaa，密码1234；第二个用户名bbb,密码5678。均采用默认IP地址。

```
zwb@ubuntu: ~/Documents/work/ChatroomV1.0/src
File Edit View Search Terminal Help
zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$ ./cli
输入客户端IP(type <Enter> to use default):
输入用户名和密码，用空格隔开: aaa 1234
You have entered the chatroom, Start CHATTING Now!
User aaa has entered the Chatroom!
There are 1 people in the Chatroom now!!
```

```
zwb@ubuntu: ~/Documents/work/ChatroomV1.0/src
File Edit View Search Terminal Help
zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$ ./cli
输入客户端IP(type <Enter> to use default):
输入用户名和密码，用空格隔开: bbb 5678
You have entered the chatroom, Start CHATTING Now!
User bbb has entered the Chatroom!
There are 2 people in the Chatroom now!!
```

以下这张图片是服务端对应终端的日志信息。显示群聊系统有两个人。

```

zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$ ./ser
the server is ready and listening
你创建了一个叫做 'User' 的sqlite3 数据库!
error : database is locked
User aaa has entered the Chatroom!
There are 1 people in the Chatroom now!!
--- [User aaa has entered the Chatroom!
There are 1 people in the Chatroom now!!
] to [aaa] 发送成功!
error : database is locked
User bbb has entered the Chatroom!
There are 2 people in the Chatroom now!!
--- [User bbb has entered the Chatroom!
There are 2 people in the Chatroom now!!
] to [aaa] 发送成功!
--- [User bbb has entered the Chatroom!
There are 2 people in the Chatroom now!!
] to [bbb] 发送成功!

```

用户aaa在群聊系统中发送消息。具体方法是输入冒号开头的消息，然后回车

在终端输入 冒号开头 的字符 :how are you?下图中可以看到，当前群聊中的两个用户aaa和bbb对应的终端上都出现用户aaa发送的聊天信息：

<pre> zwb@ubuntu:~/Documents/work/ChatroomV1.0/src\$./cli 输入客户端IP(type <Enter> to use default): 输入用户名和密码, 用空格隔开: aaa 1234 You have entered the chatroom, Start CHATTING Now! User aaa has entered the Chatroom! There are 1 people in the Chatroom now!! User bbb has entered the Chatroom! There are 2 people in the Chatroom now!! :how are you? aaa says:how are you? </pre> <p style="text-align: center; color: red;">aaa</p>	<pre> zwb@ubuntu:~/Documents/work/ChatroomV1.0/src\$./cli 输入客户端IP(type <Enter> to use default): 输入用户名和密码, 用空格隔开: bbb 5678 You have entered the chatroom, Start CHATTING Now! User bbb has entered the Chatroom! There are 2 people in the Chatroom now!! aaa says:how are you? </pre> <p style="text-align: center; color: red;">bbb</p>
---	---

然后aaa和bbb分别多聊几句：

aaa和bbb对应的终端显示聊天信息如下：

```
zwb@ubuntu: ~/Documents/work/ChatroomV1.0/src
File Edit View Search Terminal Help
zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$ ./cli
输入客户端IP(type <Enter> to use default):
输入用户名和密码, 用空格隔开: aaa 1234
You have entered the chatroom, Start CHATTING Now!
User aaa has entered the Chatroom!
There are 1 people in the Chatroom now!!
User bbb has entered the Chatroom!
There are 2 people in the Chatroom now!!
:how are you?
aaa says:how are you?
:i am fine
aaa says:i am fine
bbb says:it is ok!!!
bbb says:have you eat lunch?
:oh,i am going now,bye
aaa says:oh,i am going now,bye
bbb says:see you again
□
```

aaa

```
zwb@ubuntu: ~/Documents/work/ChatroomV1.0/src
File Edit View Search Terminal Help
zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$ ./cli
输入客户端IP(type <Enter> to use default):
输入用户名和密码, 用空格隔开: bbb 5678
You have entered the chatroom, Start CHATTING Now!
User bbb has entered the Chatroom!
There are 2 people in the Chatroom now!!
aaa says:how are you?
aaa says:i am fine
:it is ok!!!
bbb says:it is ok!!!
:have you eat lunch?
bbb says:have you eat lunch?
aaa says:oh,i am going now,bye
:see you again
bbb says:see you again
```

bbb

服务端终端信息打印日志信息如下。日志的含义是，如果群聊中的文字消息发送成功，服务端就会向刚刚发送文字用户对应的客户端返回成功信息，并打印。

```
zwb@ubuntu: ~/Documents/work/ChatroomV1.0/src
File Edit View Search Terminal Help
] to [aaa] 发送成功!
--- [aaa says:how are you?
] to [bbb] 发送成功!
--- [aaa says:i am fine
] to [aaa] 发送成功!
--- [aaa says:i am fine
] to [bbb] 发送成功!
--- [bbb says:it is ok!!!
] to [aaa] 发送成功!
--- [bbb says:it is ok!!!
] to [bbb] 发送成功!
--- [bbb says:have you eat lunch?
] to [aaa] 发送成功!
--- [bbb says:have you eat lunch?
] to [bbb] 发送成功!
--- [aaa says:oh,i am going now,bye
] to [aaa] 发送成功!
--- [aaa says:oh,i am going now,bye
] to [bbb] 发送成功!
--- [bbb says:see you again
] to [aaa] 发送成功!
--- [bbb says:see you again
] to [bbb] 发送成功!
]
```

再多加入一个用户ccc,密码6666.此时服务端和客户端显示，群聊系统已经有3个人了。

```
zwb@ubuntu: ~/Documents/work/ChatroomV1.0/src
File Edit View Search Terminal Help
zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$ ./cli
输入客户端IP(type <Enter> to use default):
输入用户名和密码，用空格隔开: ccc 6666
You have entered the chatroom, Start CHATTING Now!
User ccc has entered the Chatroom!
There are 3 people in the Chatroom now!!
```

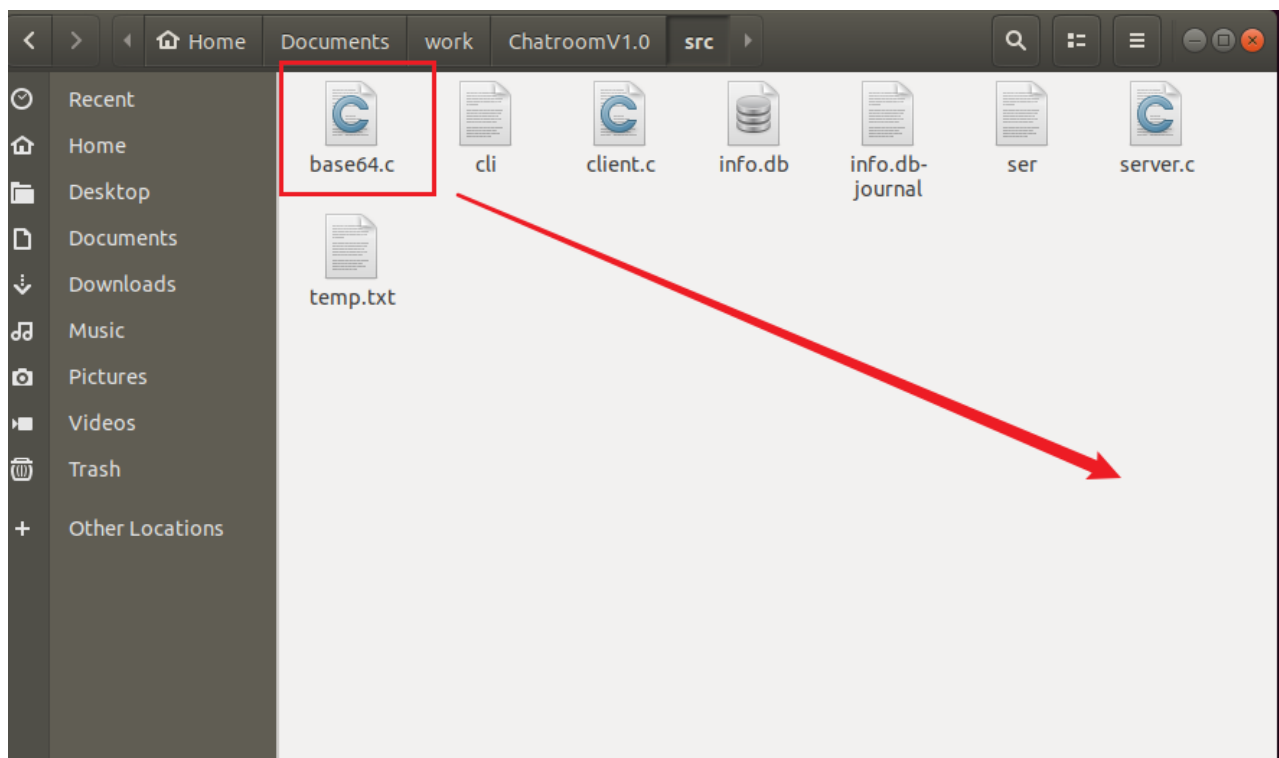
再查看User数据库表:

文件传输：

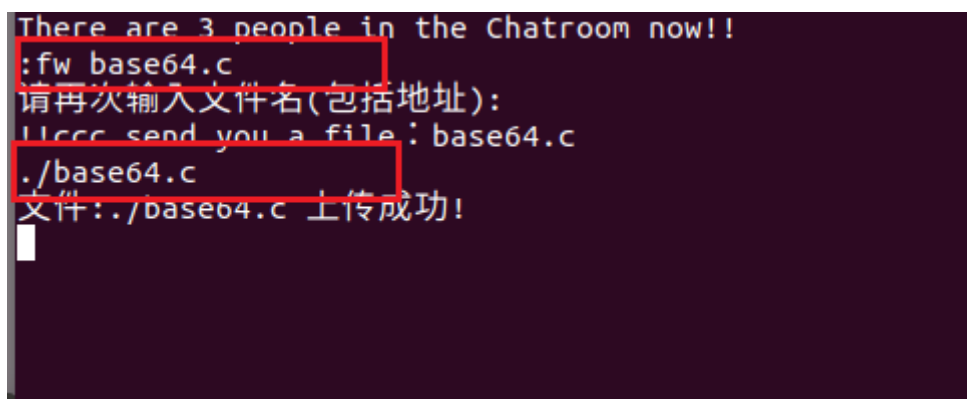
群聊系统某用户可以上传本地地文本文件，在服务端暂存；群聊系统中其他用户可以输入指令将文件从服务端下载到本地，存入指定的文本文件中

现在演示在用户ccc对应的客户端下，上传文件。为了演示需要，我临时创建了一个.c文件，名叫base64.c,是一个C语言源程序文件。

下面演示选择上传与项目代码同目录下的一个.c文件base64.c到服务端



输入:**fw base64.c**。此后所有在线用户的终端都出现了！！**ccc send you a file :base64.c**这个提示。同时，在ccc用户对应的终端提示用户ccc重新输入包括地址的文件名。接下来输入**./base64.c**。终端提示文件上传到服务端成功。



此时aaa和bbb终端打印信息如下

<pre> There are 1 people in the Chatroom now!! User bbb has entered the Chatroom! There are 2 people in the Chatroom now!! :how are you? aaa says:how are you? :i am fine aaa says:i am fine bbb says:it is ok!!! bbb says:have you eat lunch? :oh,i am going now,bye aaa says:oh,i am going now,bye bbb says:see you again User ccc has entered the Chatroom! There are 3 people in the Chatroom now!! !!ccc send you a file:base64.c </pre>	<pre> There are 2 people in the Chatroom now!! aaa says:how are you? aaa says:i am fine :it is ok!!! bbb says:it is ok!!! :have you eat lunch? bbb says:have you eat lunch? aaa says:oh,i am going now,bye :see you again bbb says:see you again User ccc has entered the Chatroom! There are 3 people in the Chatroom now!! !!ccc send you a file:base64.c </pre>
---	--

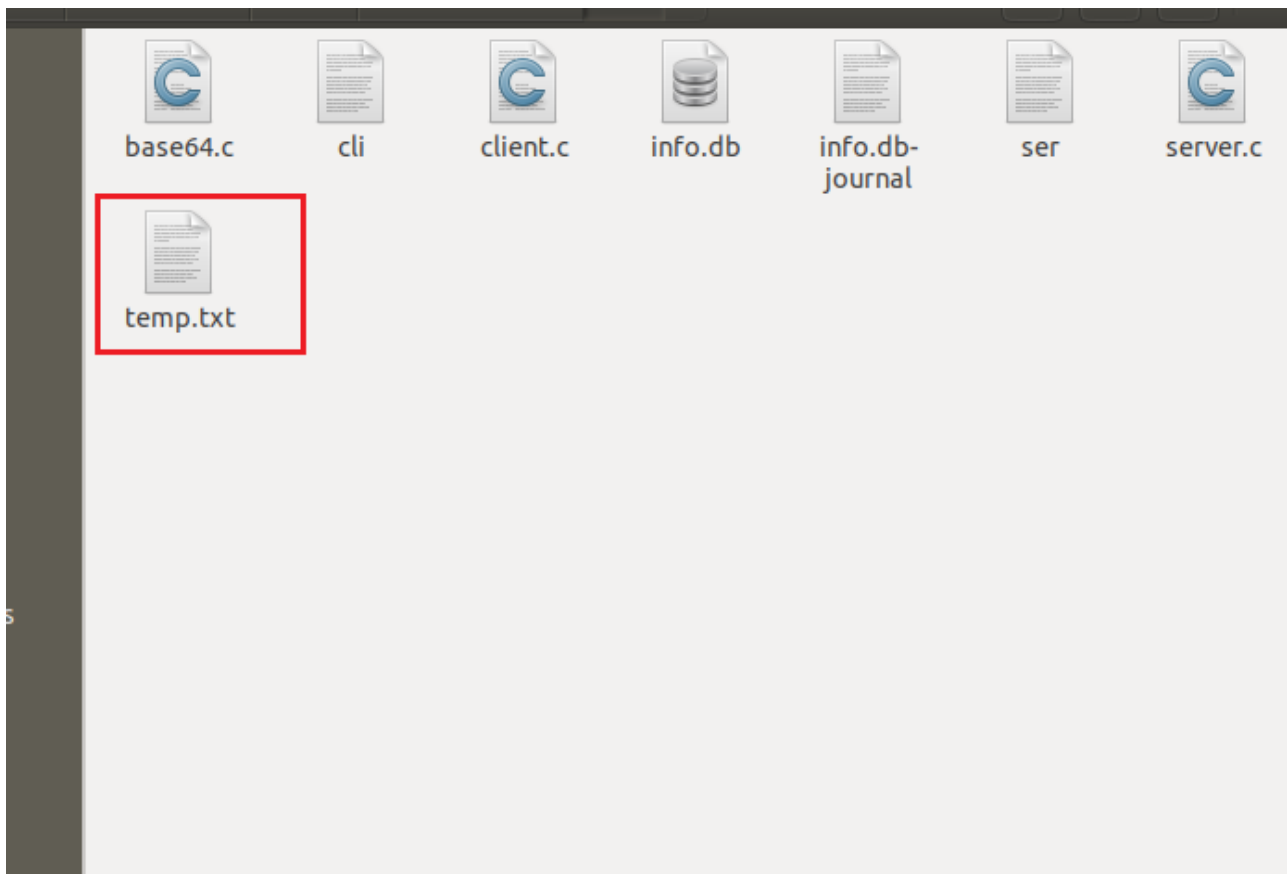
服务端终端打印信息如下：可以看到，base64.c文件是一段段地分部分上传的，总共有7部分，前6部分各有1024字节，第七部分有150字节。

```

There are 3 people in the Chatroom now!!
] to [ccc] 发送成功!
--- [!!ccc send you a file:base64.c
] to [aaa] 发送成功!
--- [!!ccc send you a file:base64.c
] to [bbb] 发送成功!
--- [!!ccc send you a file:base64.c
] to [ccc] 发送成功!
收到了 1024 字节
发送第 1 部分成功!
收到了 1024 字节
发送第 2 部分成功!
收到了 1024 字节
发送第 3 部分成功!
收到了 1024 字节
发送第 4 部分成功!
收到了 1024 字节
发送第 5 部分成功!
收到了 1024 字节
发送第 6 部分成功!
收到了 150 字节
发送第 7 部分成功!
成功发送所有部分!

```

此时，在任意一个用户终端下都可以接收该文件，从服务端下载该文本文件，存入到本地指定的文本文件里，这里我们已经创建好了一个文本文件用来接收该文件。这里我们指定的本地文本文件是在同一目录下新建的temp.txt。



我们选择aaa用户来下载这个已经上传到服务端地文本文件

输入:`fs ./temp.txt`

aaa用户对应的终端显示信息如下：在下载接收地时候也是按部分接收的，我们设定，一个**buffer**最大容量是**1024**字节，所以要分7部分上传和下载。


```
zwb@ubuntu: ~/Documents/work/ChatroomV1.0/src
File Edit View Search Terminal Help
2 There are 2 people in the Chatroom now!!
:how are you?
aaa says:how are you?
:i am fine
aaa says:i am fine
bbb says:it is ok!!!
bbb says:have you eat lunch?
:oh,i am going now,bye
aaa says:oh,i am going now,bye
bbb says:see you again
User ccc has entered the Chatroom!
There are 3 people in the Chatroom now!!
!!ccc send you a file : base64.c
:fs ./temp.txt
你想把文件存储在 ./temp.txt
成功接收文件第 1 部分!
成功接收文件第 2 部分!
成功接收文件第 3 部分!
成功接收文件第 4 部分!
成功接收文件第 5 部分!
成功接收文件第 6 部分!
成功接收文件第 7 部分!
从服务器成功收到文件并存入 ./temp.txt!
```

temp.txt文件原本是空的,在接收文件后变成: 相当于将base64.c中的内容存入了temp.txt里面。

```
temp.txt
~/Documents/work/ChatroomV1.0/src
#include <stdio.h>
#include <string.h>

//
// base64.c
// base64
//
/**
 * 转解码过程
 * 3 * 8 = 4 * 6; 3字节占24位, 4*6=24
 * 先将要编码的转成对应的ASCII值
 * 如编码: s 1 3
 * 对应ASCII值为: 115 49 51
 * 对应二进制为: 01110011 00110001 00110011
 * 将其6个分组分4组: 011100 110011 000100 110011
 * 而计算机是以8bit存储,所以在每组的高位补两个0如下:
 * 00011100 00110011 00000100 00110011对应:28 51 4 51
 * 查找base64 转换表 对应 c z E z
 *
 * 解码
 * c z E z
 * 对应ASCII值为 99 122 69 122
 * 对应表base64_suffix_map的值为 28 51 4 51
 * 对应二进制值为 00011100 00110011 00000100 00110011
 * 依次去除每组的前两位,再拼接成3字节
 * 即: 01110011 00110001 00110011
 * 对应的就是s 1 3
 */

#include <stdlib.h>

// base64 转换表,共64个
```


查询在线人数

在某用户终端下，当群聊系统中出现用户登入或登出时，都会在终端打印显示当前系统中的人数。这个功能在前三个步骤中有涉及，就不再赘述。

其他小功能

用户名重复避免

打开一个客户端，用户名aaa登录；

```
zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$ ./ File Edit View Search Terminal Help
the server is ready and listening
你创建了一个叫做 'User' 的sqlite3 数据库!
User aaa has entered the Chatroom!
There are 1 people in the Chatroom now!!
--- [User aaa has entered the Chatroom!
There are 1 people in the Chatroom now!!
] to [aaa] 发送成功!
[]

zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$ ./cli
输入客户端IP(type <Enter> to use default):
输入用户名和密码，用空格隔开: aaa 123456
You have entered the chatroom, Start CHATTING Now!
User aaa has entered the Chatroom!
There are 1 people in the Chatroom now!!
```

然后再打开一个客户端，使用同样的用户名aaa登录,发现终端显示用户名已存在。换一个用户名bbb登录，成功。这个功能避免了用户名的冲突。

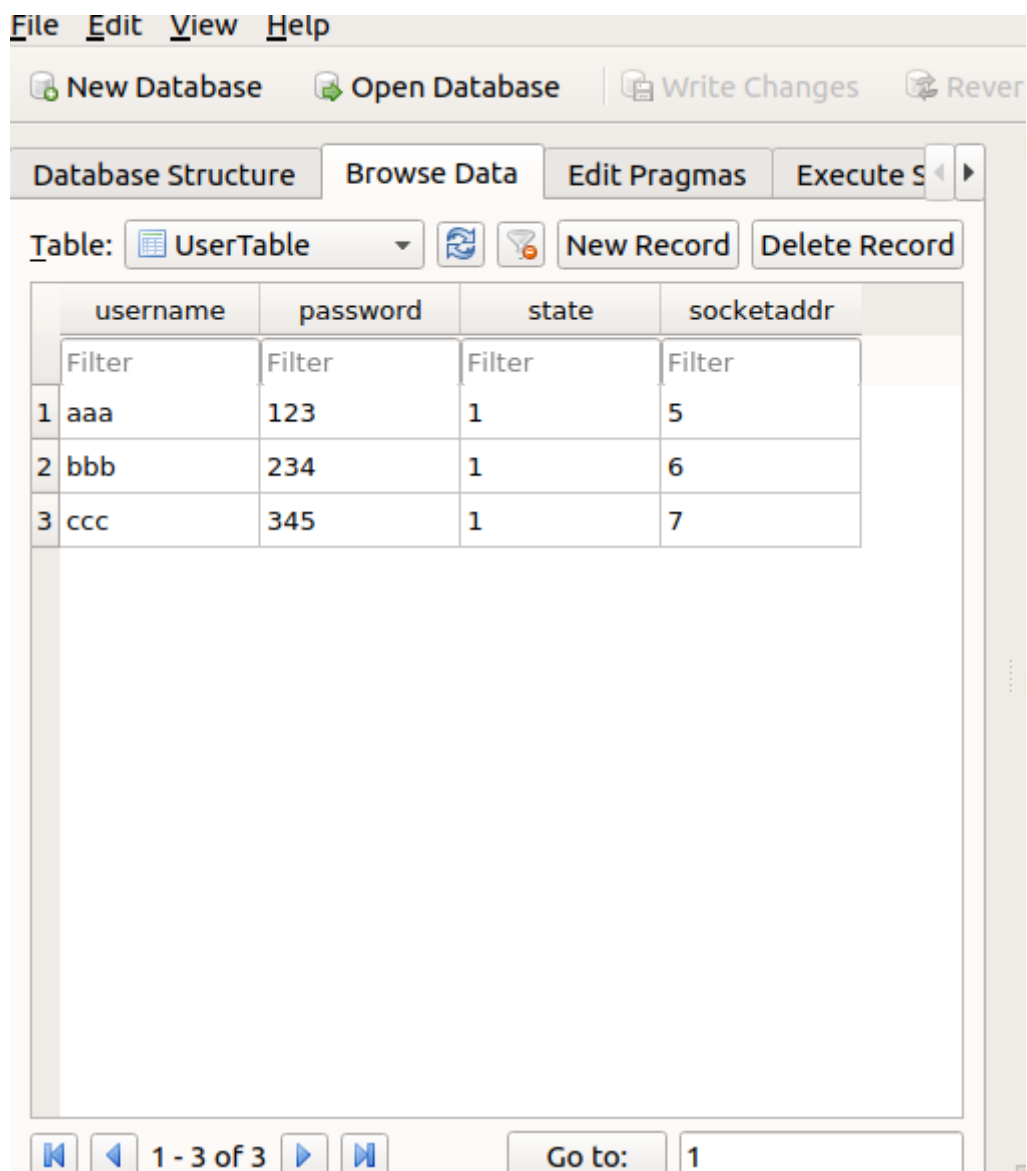
```
File Edit View Search Terminal Help
zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$ ./ser File Edit View Search Terminal Help
the server is ready and listening
你创建了一个叫做 'User' 的sqlite3 数据库!
User aaa has entered the Chatroom!
There are 1 people in the Chatroom now!!
--- [User aaa has entered the Chatroom!
There are 1 people in the Chatroom now!!
] to [aaa] 发送成功!
User bbb has entered the Chatroom!
There are 2 people in the Chatroom now!!
--- [User bbb has entered the Chatroom!
There are 2 people in the Chatroom now!!
] to [aaa] 发送成功!
--- [User bbb has entered the Chatroom!
There are 2 people in the Chatroom now!!
] to [bbb] 发送成功!
[]

zwb@ubuntu:~/Documents/work/ChatroomV1.0/src$ ./cli
输入客户端IP(type <Enter> to use default):
输入用户名和密码，用空格隔开: aaa 34567
用户名已存在，请换一个。
输入用户名和密码，用空格隔开: bbb 56789
You have entered the chatroom, Start CHATTING Now!
User bbb has entered the Chatroom!
There are 2 people in the Chatroom now!!
```

用户信息数据库存储与图形界面可视化

创建了三个客户端，用户名分别为aaa,bbb,ccc,登录完成后

用DB Broswer工具查看数据库记录



用户ccc所在终端输入:q!，退出

按下DB Browser更新键更新User表，显示用户ccc的status为0，说明他不在线了。

这样方便系统管理员对群聊系统中的用户进行管理。

DB Browser for SQLite - /home/zwb/De

File Edit View Help

New Database Open Database Write Changes Reveal

Database Structure Browse Data Edit Pragma Execute SQL

Table: UserTable New Record Delete Record

	username	password	state	socketaddr
	Filter	Filter	Filter	Filter
1	aaa	123	1	5
2	bbb	234	1	6
3	ccc	345	0	7

1 - 3 of 3 Go to: 1