

# 实 验 报 告

组号： 5-6

姓名： 施炎江    学号： 2186113847    班级： 计算机 82

姓名： 高浩翔    学号： 2181411962    班级： 计算机 82

## 一、 实验名称

实现一个简单的聊天程序

## 二、 实验原理

### 1. Socket 网络编程原理

#### 1) 服务器

服务器先创建一个套接字，并将该套接字和特定端口绑定，然后服务器开始在此套接字上监听，直到收到一个客户端的连接请求，然后服务器与客户端建立连接，连接成功后和该客户端进行通信（相互接收和发送数据），进行用户信息验证，并返回验证信息。最后，服务器和客户端断开连接，继续在端口上监听。

#### 2) 客户端

客户端创建一个套接字，里面包含了服务器的地址和端口号，客户端的端口号由系统自动分配，不需要指明。和服务器建立连接，如果连接成功则 Socket 创建成功。通信结束后主动断开连接，释放资源。

### 2. 文件传输原理

文件传输基于 Socket 网络编程技术，主要分为 3 个阶段：文件传输前的控制信息对接阶段、文件数据传输阶段、文件传输后收尾阶段。在第一阶段，收发双方需要对接待传输文件的相关信息，如文件名、是否有断点文件，如果有，该从哪里继续传输等。在第二阶段，收发双方利用 socket 传输文件数据，主要包括利用 socket 收发数据和对文件进行读写两大子操作。在第三阶段，发送方应向接收方发送文件传输完毕的标志，然后断开二者的连接。

### 3. 语音通话原理

语音通话可分为语音的采集播放、语音的传输。语音的传输基于 socket 通信，和传输普通文件没有区别。语音的采集与播放基于 Windows 的低级音频处理函数 WAVEIN、WAVEOUT 系列 API。为了保证语音的连续性与实时性，采用多缓冲的思路构建多个缓冲轮流在硬件消息回调函数的处理下将音频从硬件缓冲中读出、写入，最终实现语音通话功能。

### 三、 实验目的

1. 掌握 Socket 的相关基础知识，学习 Socket 编程的基本函数和模式、框架
2. 掌握 UDP、TCP 协议及 Client/Server 和 P2P 两种模式的通信原理。
3. 掌握 Socket 编程框架。

### 四、 实验内容

#### 1. 基本功能

- 1) 实现用户认证功能，具体包括用户注册和用户登录两方面。
- 2) 实现文字聊天功能，要求一方能够连续发送任意多条消息。
- 3) 实现文件传输功能，要求能够传输较大二进制文件。

#### 2. 高级功能

- 1) 实现文件的断点续传功能，要求上传、下载均支持断点续传。
- 2) 实现语音通话功能。
- 3) 由于本程序是 CSC 架构，因此不需要 NAT 穿透也能在任何地方运行。

### 五、 实验实现

#### 1. 人员分工

施炎江：负责 CSC 聊天模型框架的基本搭建，参与用户认证功能的部分开发，负责文字聊天功能的开发，完善了文件传输开发，负责断点续传功能的开发。

高浩翔：完善用户认证功能，负责文件传输功能的开发，负责语音通话功能的开发。

#### 2. 实验设计

##### （一）协议

本程序的所有功能都是基于 TCP 协议实现的。整个程序的基本功能是客户端和服务端间的消息传递,其他所有功能都是通过识别字符串中的关键字衍生而成的。

本程序的架构是 CSC 模式,即客户端-服务器-客户端模式。我们将服务器代码(server.cpp)放在了一台云服务器上运行,然后两个用户分别在各自的主机上运行客户端代码(client.cpp)。这种方式下服务器有公网 IP,不需要 NAT 穿透就可以在不同地方访问服务器、运行程序。服务器在整个程序中起到了存储转发的功能,即对于文字聊天、在线文件传输、语音通话功能,服务器负责从一个客户端接收消息(这里的“消息”既包含了用户能看见、听见的文字消息、语音消息,也包含了用户看不见的各种控制信息,如文件传输前的各种控制信息等),再直接转发到另一客户端;对于离线文件传输功能,服务器负责存储离线文件,并提供下载途径。

具体来说,在客户端的发送消息到服务器的字符串中,如果程序识别到关键字“FTP”,则会启动发送文件的功能;如果识别到“FILELIST”,则会显示服务器上所有可供下载的文件名称;如果识别到“GET”,则会启动从服务器上下载文件的功能;如果识别到“VOICE”,则会启动语音通话功能。在客户端接收的来自服务器的字符串中,如果识别到“online”,则会启动文件在线传输功能;如果识别到“VOICE”,则会启动语音通话功能。每个功能的具体交互过程如下。

#### 1) 用户认证

客户端有两个函数 login、sign\_up 分别处理用户认证和用户注册,服务器端有一个函数 ID\_verify 来和客户端进行对接,下面分别说明。

当进行用户认证时,客户端会输出指导性语句,指导用户输入用户名和密码。然后,客户端会将用户名和密码拼接成一个字符串,用空格分隔,再调用 socket 的 send 函数将该字符串发送给服务器。服务器端有一个名为 password 的文本文件,所有用户的用户名和密码都存储在这个文件中。服务器收到密码后,会将字符串中的信息和 password 文件中的所有记录进行比较,如果成功匹配,则说明

用户身份认证成功，返回成功信息并允许用户进入聊天室；如果未找到匹配字符串，则说明用户身份认证失败，返回失败信息，并继续等待下一条认证信息，直到用户身份认证成功。

当进行用户注册时，客户端同样会输出指导性语句，指导用户设置新的用户名和密码，然后以同样的方式将其拼接成一个字符串发送给服务器，服务器在收到信息后会将新账户信息写入 password 文件，这样就完成了整个用户注册的功能。

## 2) 文字聊天

文字聊天的实现方式很简单，本质上就是利用 socket 的 send 和 recv 函数进行字符串的收发。由于我们的程序是 CSC 架构的，即客户端—服务器—客户端架构，因此需要服务器在两个客户端之间进行消息的转发功能。考虑到要实现一方可以连续发送任意条消息的要求，我们决定通过多线程的方式处理双方消息的收发。

## 3) 文件传输

正如上面所说，当用户键入“FTP”关键字时，客户端会首先让用户选择文件传输方式，在线传输或离线传输，然后会让用户输入想要传输的文件名。

如果选择在线传输，客户端会新建一个名为 ftp\_thread 的新线程，专门用于处理文件的传输。服务器收到消息后会及时将消息转发给接受方，然后接收方也会新建一个文件传输专用的线程，并和发送方一起分别新建连接到服务器的套接字。但是在新建好套接字后，服务器无法区分两个客户端哪个是发送方，哪个是接收方，因此此时服务器会对两个客户端分别新建一个 waiting 线程并阻塞，等待客户端发来各自的身份消息。具体来说，如果是文件发送方，它就会发送“I am sender.”字符串到服务器；如果是文件接收方，它就会发送“I am receiver.”字符串到服务器。当然，这些信息交互过程对于用户来说都是透明的。通过收到的字符串，服务器就能判断两个新建的套接字中，哪个是文件发送方。

在这之后，万事俱备，剩下要做的就是开始正式的文件传输了。发送方会以

程序所在的目录为当前目录，根据文件名打开指定的文件。由于 TCP 传输有报文大小的限制，又考虑到传输的效率，我们规定每读 1024 字节就打包成一个报文，用 send 函数发送，重复这一过程直到整个文件被发送完毕。这个过程可以用一个 while 循环实现，逻辑并不复杂。而对于接受方来说过程也是类似的，它会在当前目录下新建一个名为“recv\_file”的文件，并将收到的数据循环写入这个文件。在这个过程中，服务器的功能和在文字聊天过程中的一样，也是一个单纯的转发功能。换句话说，对于两个客户端来说，服务器是透明的。当发送方所有数据后，它会向服务器发送“FTPfin”字段，以告知文件传输的结束。服务器会将这条消息转发，并且断开和客户端的连接，结束文件传输线程。接收方在收到结束标志后，会结束文件的写，断开连接，结束文件传输线程。此时，整个在线文件传输过程就结束了。

文件的离线传输涉及到断点续传功能，因此具体过程将在断点续传部分进行描述，这里只说明一下基本交互流程。离线传输首先将文件发送给服务器，服务器将接受到的文件保存名为“Downloads”的文件夹中，这一过程比较简单，可以说是消息收发和文件写入的结合，故具体过程不再赘述。然后，服务器发送一条消息给另一客户端，通知用户有新文件可供下载。用户输入“FILELIST”关键字后，服务器就会将“Downloads”文件夹下所有文件的文件名发送给客户端，用户就能根据这份文件名单选择想要下载的文件。如果用户想要下载服务器上的文件，只要使用“GET \$filename”即可，其中\$filename 是想要下载的文件名。服务器端会识别这个关键字，新建线程，新建套接字，完成文件传输。这一过程和文件的离线上传是对称的，故不再赘述。

#### 4) 断点续传

断点续传，最关键任务的就是确定已传文件的大小。该功能只在离线文件传输的上传、下载中有。由于文件的上传和下载是对称的，因此这里只描述文件的断点上传流程，断点下载类似。断点上传，就是客户端向服务器传输文件，而服务器端可能已有部分残缺的文件。这种情况下，首先需要客户端告知服务器，要

传输哪个文件？即首先客户端要向服务器发送待传输的文件名。服务器收到文件名后，在 Downloads 文件夹下扫描所有文件的文件名，看是否能匹配到。如果没有匹配到，说明服务器端没有对应的残缺文件，这是一个新文件，因此就向客户端返回一个值为 0 游标；如果匹配到了，说明已有部分文件在上次传输过程中被传输到了服务器，这时服务器就会计算这个残缺文件的大小（按字节计算），然后将这个值作为游标值发送给客户端。于是客户端便会收到这个游标值，它便会将读文件指针跳转到这个游标值的位置，然后继续传输文件。剩下的部分就和文件的在线传输一样了。这样，我们就实现了断点续传功能。

### 5) 语音通话

语音在网络传输过程中其实和其他数据并没有太大的区别，因此语音传输功能的关键在于调用计算机的麦克风与扬声器进行录音、播放，以及如何处理才能保证语音通话的实时性与连续性这两大问题。对于第一个问题，我们采用 Windows 配备的 WAVEIN、WAVEOUT 系列媒体 API 来对硬件设备以及相关硬件缓存进行调用。对于如何保证语音通话的实时性与连续性这一问题，在查阅大量资料之后，我们决定采用多缓存的思想来传输。在语音的采集部分开辟两块缓存，在回调函数的控制下轮流接受硬件设备的缓存并发送。在音频的接受与发送部分，我们开辟 50 个接受缓存循环接受通过套接字传输来的语音数据，同时开辟 8 个播放缓存轮流从接受缓存中读取数据加入硬件缓存。这三大缓存之间的协调通过指针的自增自减以及音频 API 的消息回调函数控制来实现。这样一来大大提高了语音通话的实时性与连续性。由于时间关系，我们并没有对音频进行压缩，而是直接传输 PCM 音频裸流，相信如果再采用压缩机制，实时语音的表现会更好。

## （二）UI 设计

考虑到开发时间较短，没有设计特别的 UI，所有操作均在命令行界面进行。

## （三）框架结构

服务器端采用多线程技术保证服务的高效。常驻线程两个，分别用来监听两个客户端发来的消息。临时线程有三种，第一种是用来向客户端发送消息的，它

只有在接收到一条消息后才会被创建，在消息发送完成后便会结束；第二种是用来进行文件传输的，它专门负责文件数据的发送和接受，只有在接收到文件传输请求时才会被创建，在文件传输完成后便会结束；第三种是用来发送语音数据的，它只有在接收到语音通话请求时才会被创建，在语音通话结束后才会结束。

客户端也采用多线程技术实现。常驻线程有两个，一个用来接收来自服务器的消息，并将其打印出在屏幕上；一个用来向服务器发送消息。临时线程有两种，第一种用于文件传输：当用户开始发送文件功能或者用户收到文件发送请求时，会建立相应的线程用来接受或者发送文件；第二种用于语音通话：当用户建立语音通话时会建立相应的线程来录制、发送、接受、播放语音数据。

### 3. 关键代码的描述

#### (一) 服务器端

##### 1) main 函数

功能描述：新建两个套接字，等待客户端连接，新建两个线程，用于接收来自客户端发来的消息。

是否借鉴：是，借鉴了网上 socket 的基本框架，由施炎江负责开发。

##### 2) ID\_verify 函数

功能描述：用于用户认证和用户注册。

是否借鉴：否，由高浩翔自主编写开发。

##### 3) send\_func、recv\_func 函数

功能描述：接收和转发来自客户端的消息，并根据不同的关键字做出不同的响应，执行不同的功能。

是否借鉴：“FILELIST”关键字下的“打印文件夹中所有文件名”部分借鉴了网上关于如何获得文件夹中所有文件名的实现，其余部分为自主编写。其中普通文字消息功能、文件传输功能（包括在线文件传输和离线断点续传）由施炎江负责开发，语音通话功能由高浩翔负责开发。

##### 4) ftp\_func 函数

功能描述：新建用于文件传输的套接字，并和客户端对接必要控制信息（如文件名、游标位置等），为正式的文件传输做准备。

是否借鉴：套接字部分借鉴了网上的写法，其余部分由施炎江负责自主开发。

#### 5) ftp\_offline\_recv、ftp\_offline\_send、ftp\_online 函数

功能描述：前两个函数负责文件离线传输的发送和接收功能，第三个函数负责文件的在线传输功能。

是否借鉴：文件的打开、写入和读取借鉴了网上文件操作的相关实现，其余部分由施炎江自主编写开发。

#### 6) voice\_func、voice\_send\_func、voice\_recv\_func 函数

功能描述：voice\_func 函数用于在服务器端建立专门用于语音通话功能的套接字。voice\_send\_func 函数用于进行语音数据的发送，voice\_recv\_func 函数用于语音数据的接受功能。

是否借鉴：否，均由高浩翔自主开发编写。

## (二) 客户端

#### 1) main 函数

功能描述：新建套接字，连接至服务器，新建两个线程，用于接收和发送消息。

是否借鉴：是，借鉴了网上 socket 基本框架的写法，由施炎江负责开发。

#### 2) main\_UI 函数

功能描述：打印简单的 UI 界面，并提供用户登录和用户注册接口。

是否借鉴：否，由高浩翔负责编写开发。

#### 3) login 函数

功能描述：提供用户登录功能。

是否借鉴：否，由施炎江负责编写开发。

#### 4) sign\_up 函数

功能描述：提供用户注册功能。



是否借鉴：否，由高浩翔负责编写开发。

#### 5) send\_func、recv\_func 函数

功能描述：接收和转发来自客户端的消息，并根据不同的关键字做出不同的响应，执行不同的功能。

是否借鉴：否，由施炎江负责编写开发。

#### 6) ftp\_func 函数

功能描述：新建用于文件传输的套接字，并和服务端对接必要控制信息(如文件名、游标位置等)，为正式的文件传输做准备。

是否借鉴：套接字部分借鉴了网上的写法，其余部分由施炎江负责自主开发。

#### 7) ftp\_send、ftp\_recv\_online、ftp\_recv\_offline 函数

功能描述：第一个函数负责文件发送功能，第二三个函数负责文件的在线和离线接收功能。

是否借鉴：文件的打开、写入和读取借鉴了网上文件操作的相关实现，其余部分由施炎江自主编写开发。

#### 8) voice\_func、voice\_send\_func、voice\_recv\_func、vout 函数

功能描述：voice\_func 函数用于在客户端建立专门用于语音通话功能的套接字。voice\_send\_func 函数用于进行语音数据的收集与发送，voice\_recv\_func 函数用于语音数据的接受功能，vout 函数用于语音数据的播放功能。

是否借鉴：语音部分关于调用硬件设备以及相关 API 的实验参考了微软的官方文档。其余部分均由高浩翔自主开发编写。

#### 9) WaveInProc 、WaveCallback 函数

功能描述：这两个函数为 WAVEIN 与 WAVEOUT 相关媒体 API 中的消息处理回调函数。录音时录音设备的缓存存满之后、放音时设备的缓存播放完毕之后，都会向操作系统发回相关消息，这两个函数就是用于处理相关的消息。在这里主要功能是在当缓存满之后通过 socket 发送，缓存播放完毕之后加入新的缓存。

是否借鉴：是，参考了微软的官方文档，由高浩翔负责开发。

## 六、 测试及结果分析

### 1. 用户登录、注册功能

首先在云服务器开始运行服务器程序：

```
[root@kp-test01 14732]# ./server
Waiting for Connect...
```

之后在用户主机打开客户端程序，将会弹出登录界面：

```
D:\my_code\vs code\sock1.1\Debug\sock1.1.exe
-----
Input the operation:
1. Sign in.
2. Sign up.
-----
```

初次登录需要注册用户名与密码，即输入 2 进入注册功能：注册成功之后，用户会看到提示注册成功的语句。

```
D:\my_code\vs code\sock1.1\Debug\sock1.1.exe
-----
Please input your UserName: syj
Please input your PassWord: 123_
```

```
D:\my_code\vs code\sock1.1\Debug\sock1.1.exe
-----
Please input your UserName: syj
Please input your PassWord: 123
The account registration is successful. Please log in again.
_
```

注册成功 3 秒之后将会自动跳转到登录界面，用户也可以在初始登录界面输入 1 来到此界面，我们输入刚才注册的用户名与密码进行登录，将会提示登录成功，已经连接到聊天室：

```
D:\my_code\vs code\sock1.1\Debug\sock1.1.exe
-----
Please input your UserName: syj
Please input your PassWord: 123
Authentication succeeded, linked to ChatRoom.
_
```

服务器端在检测到用户登录之后会输出用户相关信息：

```
[root@kp-test01 14732]# ./server
Waiting for Connect...
User 202.117.43.84 Has Connect!
User 74.120.14.37 Has Connect!
ghx 123
syj 123
User syj 123 has login.
```

查看服务器端的密码文件可以看到我们刚才注册的用户信息：

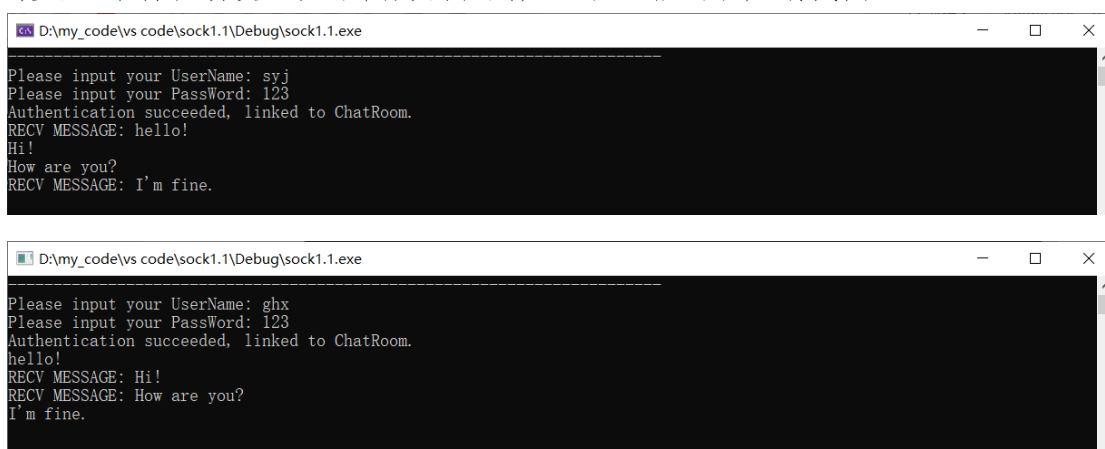


A screenshot of a WinSCP window showing a file named 'password.txt' located at '/14732/passwd.txt'. The file content is displayed as 'ghx 123' and 'syj 123', with the second line highlighted by a red rectangle.

用户登录、注册功能测试结束。

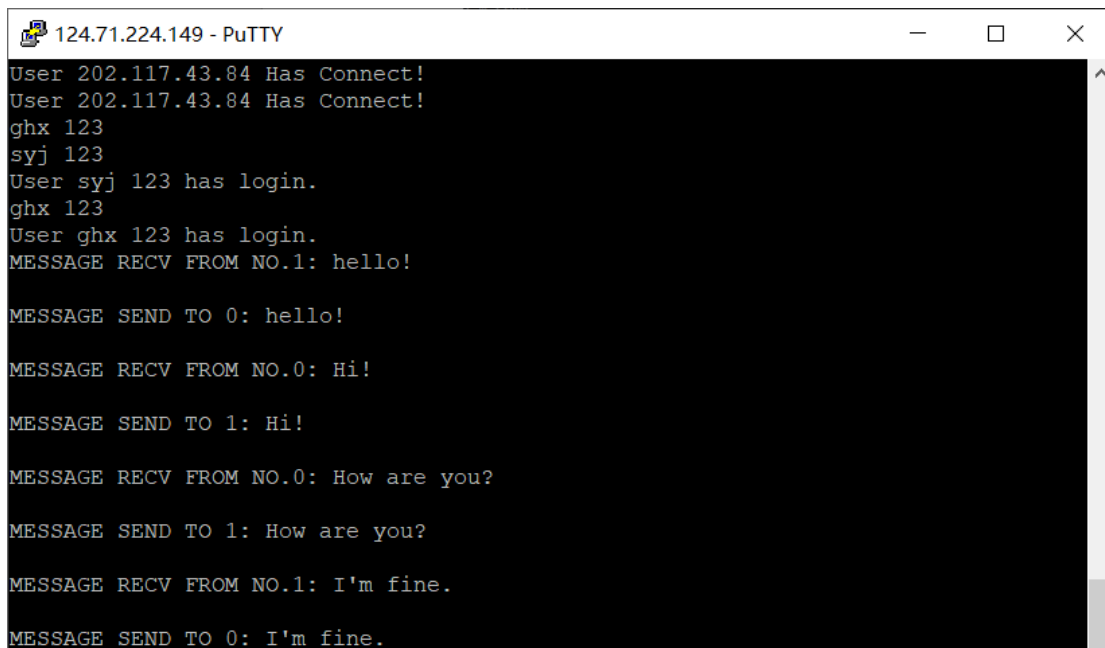
## 2. 文字聊天功能：

在成功登录之后用户将自动进入文字聊天功能，将想要发送的文字输入完毕敲击回车将自动发送，下图为两位用户互相通信的客户端截图：



Two screenshots of a client application window titled 'D:\my\_code\vs code\sock1.1\Debug\sock1.1.exe'. The first screenshot shows a user named 'syj' logging in with password '123' and sending a 'hello!' message. The second screenshot shows a user named 'ghx' logging in with password '123' and sending a 'Hi!' message, followed by a 'How are you?' message.

而在服务器端可以清楚地看到两人的对话以及转发方向：




A screenshot of a PuTTY window titled '124.71.224.149 - PuTTY' showing server logs. The logs display the login process for two users, 'ghx' and 'syj', and the subsequent chat messages being sent and received between them, including the forwarding direction (e.g., 'MESSAGE SEND TO 0: hello!').

文字聊天功能测试结束。



同时我们也可以在服务器端看到双方传输文件之前的相关对接信息:



```
124.71.224.149 - PuTTY
Waiting for Connect...
User 202.117.43.84 Has Connect!
User 202.117.43.84 Has Connect!
ghx 123
syj 123
User syj 123 has login.
ghx 123
User ghx 123 has login.
Receive FTP request.
method: online.
MESSAGE SEND TO 1: online
preparing for file transmission...
0User 202.117.43.84 is ready to transmit file!
1User 202.117.43.84 is ready to transmit file!
waiting for ID...
Sender confirm.
FTP online
recv signal: 1024
send signal: 1024
recv signal: 1024
send signal: 1024
recv signal: 1024
send signal: 1024
recv signal: 424
```

在文件传输结束之后，发送方客户端会进行如下提示：

```
D:\my_code\ws code\soc\sock1.1\Debug\soc1.1.exe
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:1024
send signal:636
sum bytes send: 120854140
send signal:6
Done.
Back to main thread.
```

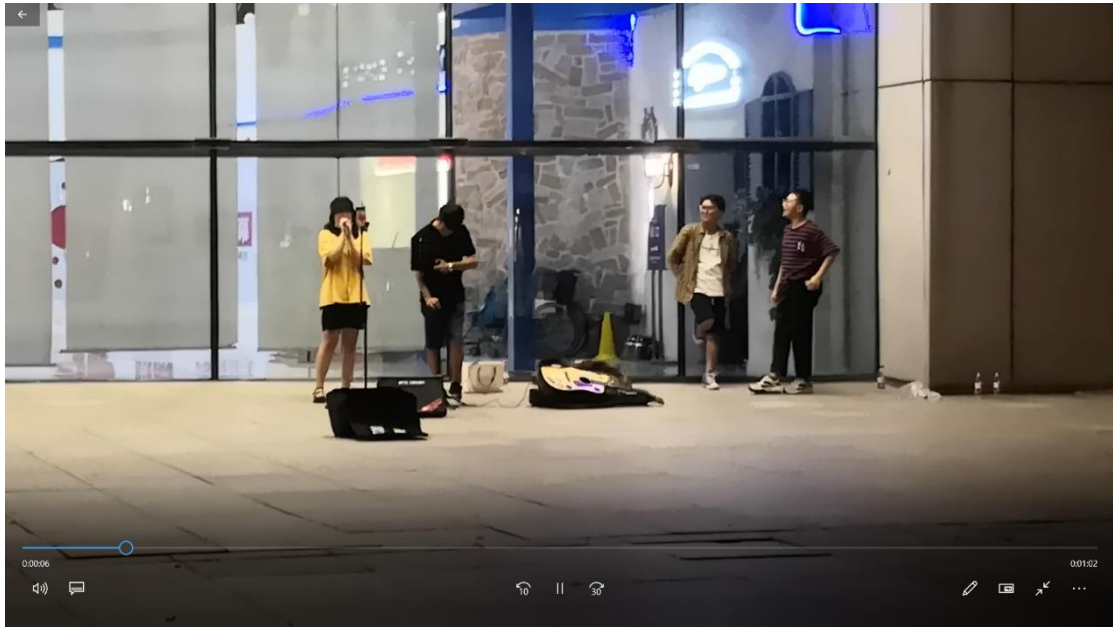
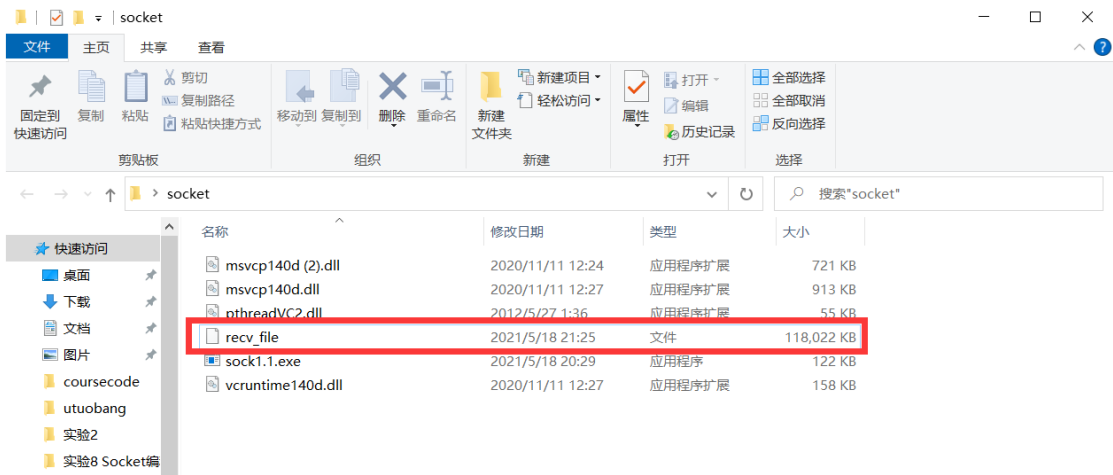
接收方的提示如下:

```
C:\Users\14732\Desktop\socket\sock1.1.exe
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 400
recv signal: 1024
recv signal: 224
recv signal: 1024
recv signal: 636
recv signal: 6
sum bytes received: 120854140
File transfer success!
```

同时我们也可以在服务器观察到文件发送完毕的提示：

```
124.71.224.149 - PuTTY
recv signal: 1024
send signal: 1024
recv signal: 1024
send signal: 1024
recv signal: 424
send signal: 424
recv signal: 600
send signal: 600
recv signal: 1024
send signal: 1024
recv signal: 1024
send signal: 1024
recv signal: 1024
send signal: 1024
recv signal: 424
send signal: 424
recv signal: 600
send signal: 600
recv signal: 636
send signal: 636
recv signal: 6
send FTPfin.
File transmission success!
```

这时我们查看本地文件：就会发现一个名为 `recvfile` 的文件在客户端目录下，我们将文件名修改为原文件名，发现文件可以进行播放。



在线文件传输功能测试完毕。

#### 4. 文件传输基本功能（离线）：

我们选择离线传输，并输入相对路径，本次传输文件为一张图片，大小为2.44MB:

[illegible]

在服务器端也可以看到文件传输的相关标识与信息:

```
[root@kp-test01 14732]# ./server
Waiting for Connect...
User 202.117.43.84 Has Connect!
ghx 123
User ghx 123 has login.
Receive FTP request.
method: offline.
recv filename is: test.png
cur: 0
send cur: 0
send message fail
preparing for file transmission...
0User 202.117.43.84 is ready to transmit file!
offline_recv function.
recv signal: 1024
recv signal: 1024
recv signal: 1024
```



传输完毕之后客户端提示:

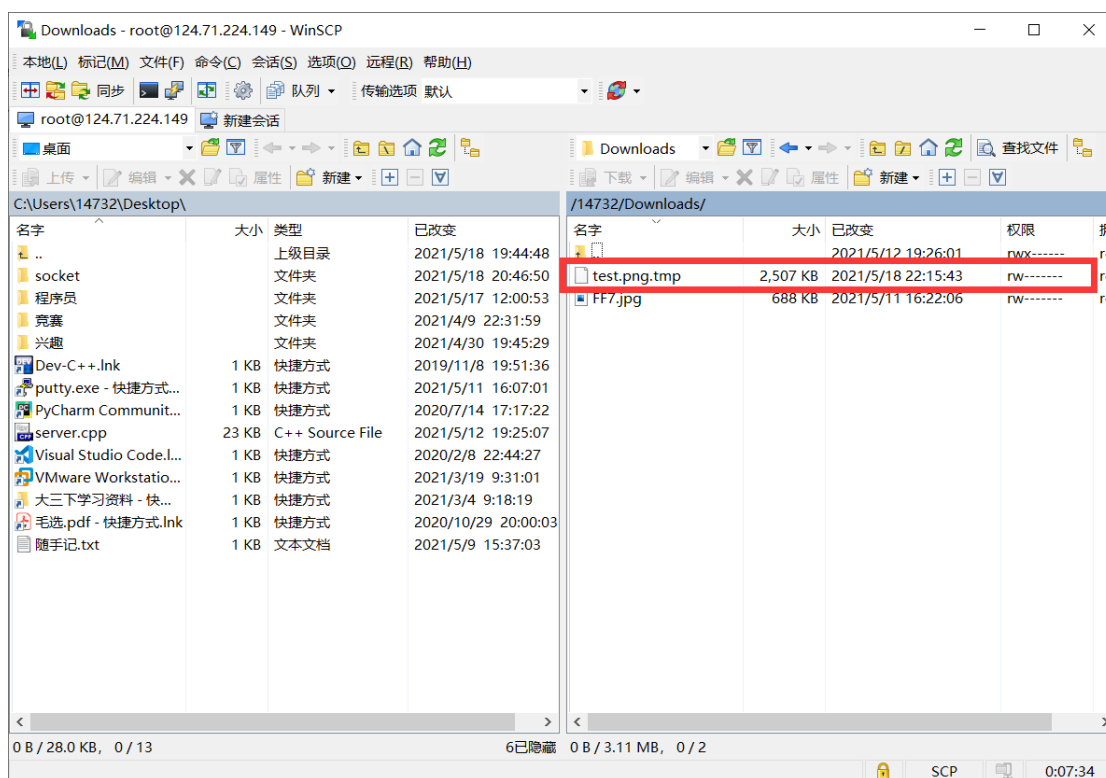
```
D:\my_code\vs code\sock1.1\Debug\sock1.1.exe  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:1024  
send signal:537  
sum bytes send: 2566681  
send signal:6  
Done.  
Back to main thread.
```

传输完毕服务器端的提示:

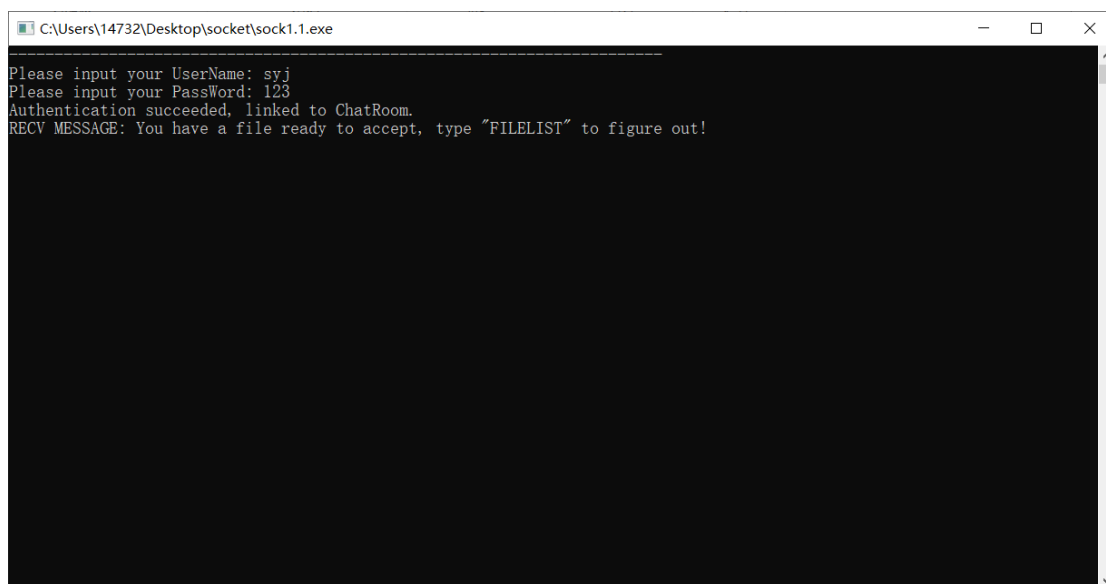


```
124.71.224.149 - PuTTY
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 424
recv signal: 600
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 424
recv signal: 600
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 424
recv signal: 600
recv signal: 1024
recv signal: 537
recv signal: 6
sum bytes received: 2566681
File transmission success!
```

此时我们查看服务器端的 Downloads 文件夹, 就可以发现相关文件:



之后当另一用户登录的时候, 客户端就会发出文件待接收提示:



我们输入 FILELISE 关键字获取服务器文件列表:

```
FILELIST
RCV MESSAGE: *****File List*****
RCV MESSAGE: -->test.png.tmp
RCV MESSAGE: -->FF7.jpg
```

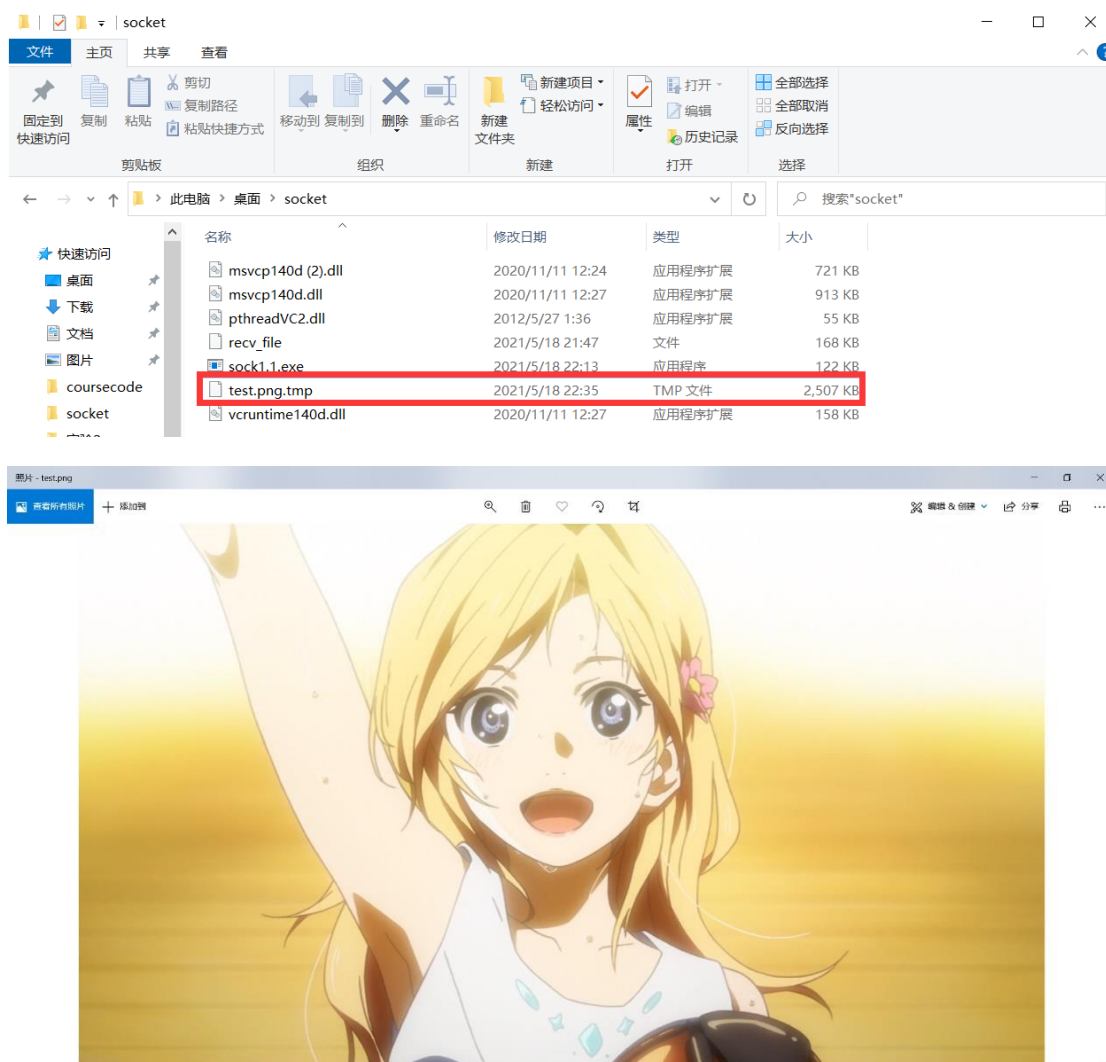
之后输入 GET 关键字以及文件的名称进行下载：

```
GET test.png.tmp
cur: 0
send cur: 0
Connecting to ftp service.
ftp recv
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 176
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
```

下载文件完毕之后结果如下：

```
选择C:\Users\14732\Desktop\socket\sock1.1.exe
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 1024
recv signal: 585
recv signal: 6
sum bytes received: 2566681
cmd.filename=test.png.tmp
newname=test.png.tmp
File transfer success!
```

我们将文件名改为 test.png 便可以正常打开文件：



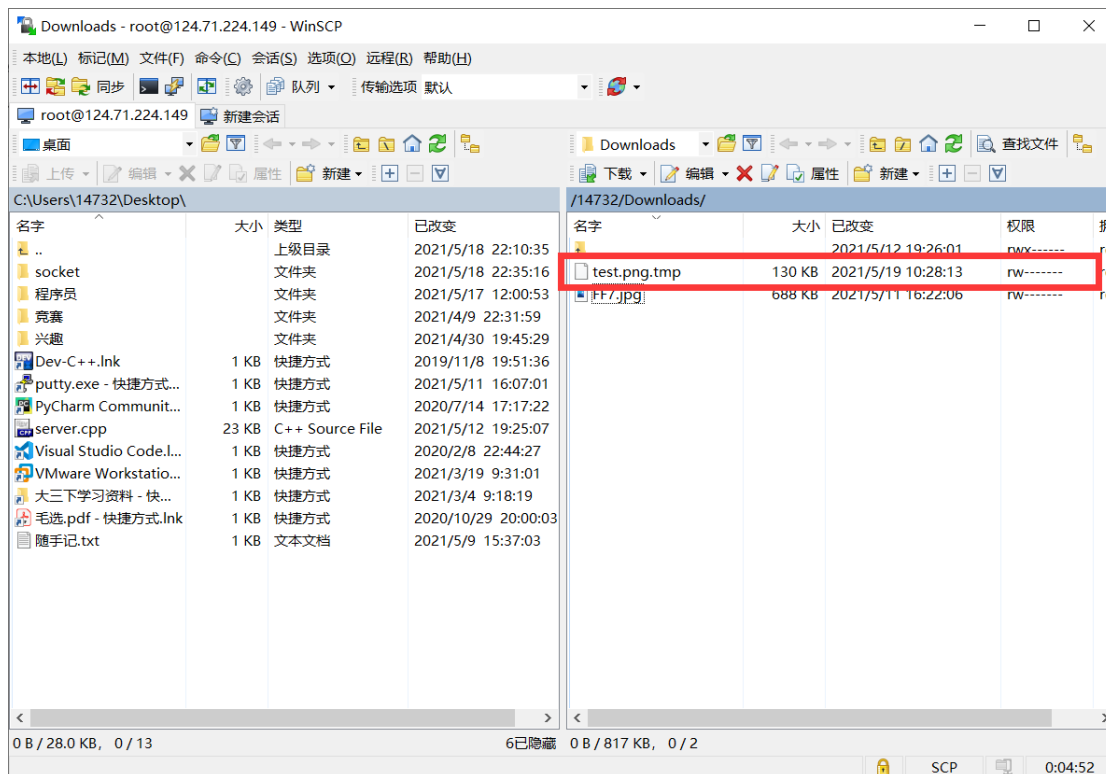
服务器端也可以看到文件传输的相关标识：

```
[root@kp-test01 14732]# ./server
Waiting for Connect...
User 202.117.43.84 Has Connect!
ghx 123
syj 123
User syj 123 has login.
MESSAGE SEND TO 1: *****File List*****
MESSAGE SEND TO 1: -->test.png.tmp
MESSAGE SEND TO 1: -->FF7.jpg
Receive FTP_GET request, filename= F7.jpg
test.png.tmp
cur: 0
preparing for file transmission...
```

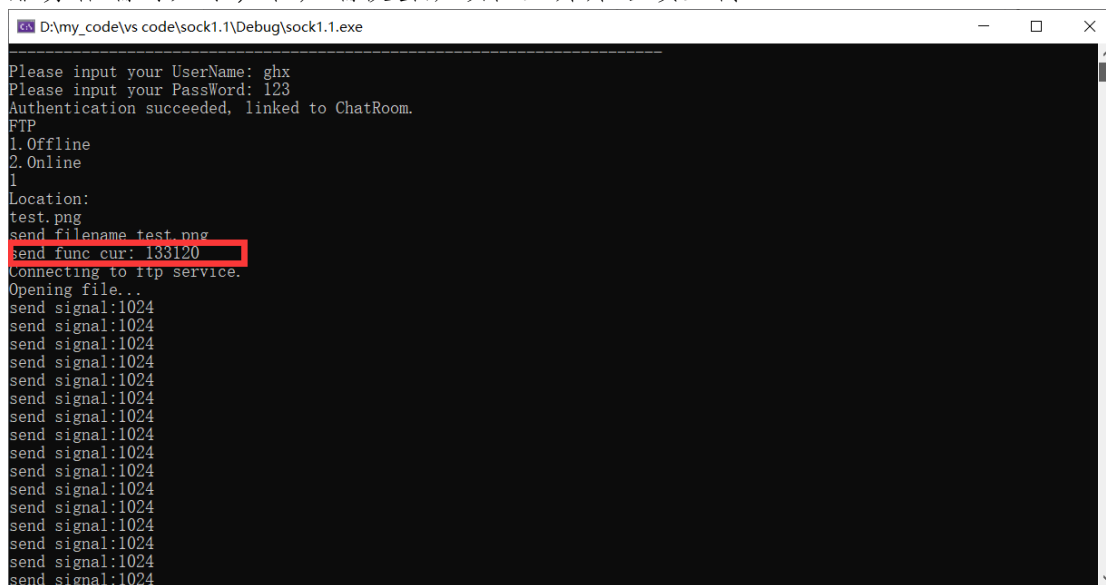
离线文件传输基本功能测试完毕。

## 5. 文件传输高级功能（离线传输断点上传）

在离线上传的时候，如果我们在传输中途关闭客户端，这个时候在服务器就会生成相应的 tmp 文件：

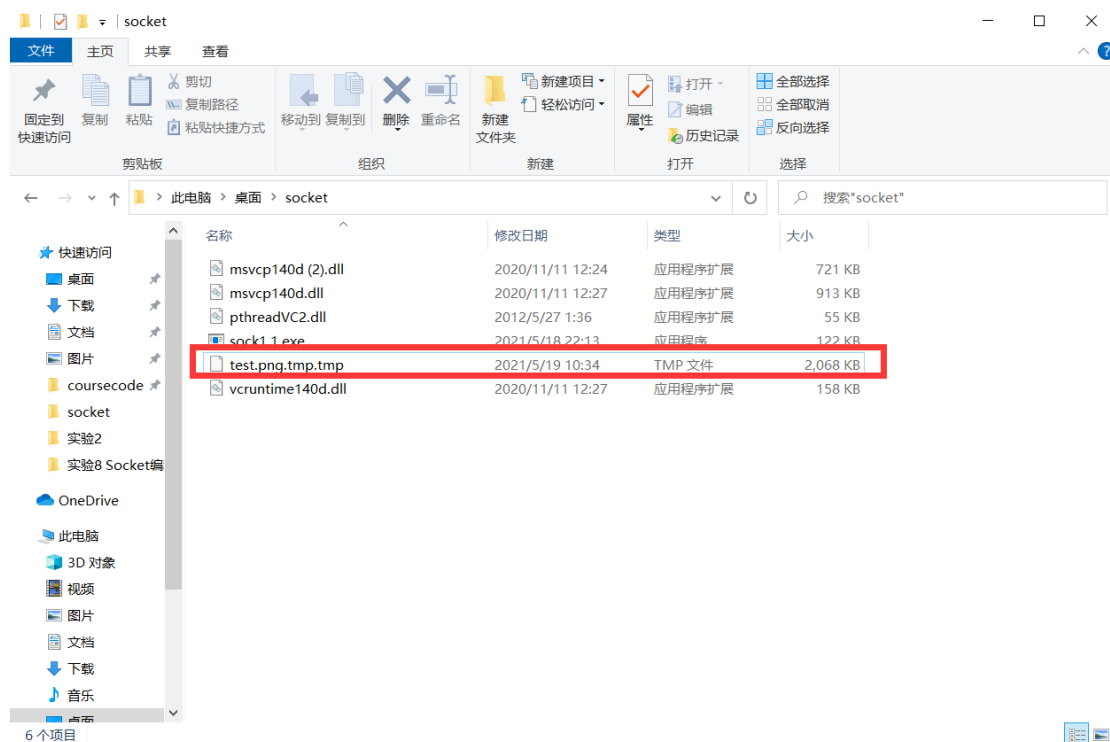


当用户重新登录开始上传，服务器就会返回游标值 `cur` 表示当前传输文件在服务器端的大小，客户端便会从该位置开始继续上传：

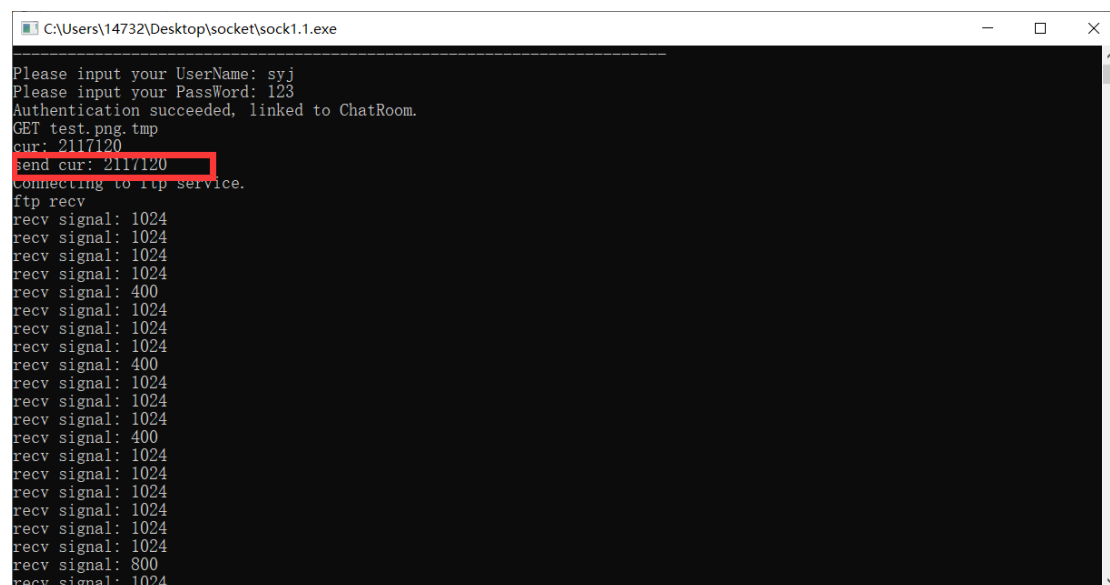


## 6. 文件传输高级功能（离线传输断点下载）

在离线传输功能的下载功能中，如果我们在传输中途关闭客户端，这个时候在客户本地就会生成相应的 tmp 文件：



当用户重新登录开始下载，客户端就会发送游标值 cur 表示当前传输文件在客户端的大小，服务器端便会从该位置开始继续传输：



服务器端也可以看到相关的标识：

```
[root@kp-test01 14732]# ./server
Waiting for Connect...
User 202.117.43.84 Has Connect!
ghx 123
syj 123
User syj 123 has login.
Receive FTP_GET request, filename= test.png.tmp

cur: 2117120
preparing for file transmission...
User 202.117.43.84 is ready to transmit file!
offline_send function.
cmd.filename= test.png.tmp

full name= ./Downloads/test.png.tmp
Opening file...
cur: 2117120
send signal:1024
send signal:1024
send signal:1024
```

## 7. 语音通话

在登录之后用户便可以输入 VOICE 关键字来进行语音通话，另一用户也会自动进入语音：

```
Microsoft Visual Studio 调试控制台
-----
Please input your UserName: syj
Please input your PassWord: 123
Authentication succeeded, linked to ChatRoom.
VOICE
Connecting to voice call service.
test
send:800send:800Bytes received: 800
Bytes received: 800
send:800Bytes received: 800
send:800Bytes received: 800
send:800Bytes received: 800
send:800Bytes received: 800
16
send:800Bytes received: 800
27
send:800Bytes received: 800
38
send:800Bytes received: 800
49
send:800Bytes received: 800
510
send:800610
Bytes received: 800
send:800Bytes received: 800
712
send:800Bytes received: 800
813
send:800Bytes received: 800
914
```

```
C:\Users\14732\Desktop\socket\sock1.1.exe
Please input your UserName: ghx
Please input your PassWord: 123
Authentication succeeded, linked to ChatRoom.
Voice call start.
Connecting to voice call service.
test
send:800send:800Bytes received: 800
send:800Bytes received: 800
Bytes received: 800
send:800Bytes received: 800
send:800Bytes received: 800
send:80015
Bytes received: 800
send:80026
Bytes received: 800
send:80037
Bytes received: 800
send:80048
Bytes received: 800
send:80059
Bytes received: 800
send:800610
Bytes received: 800
send:800711
Bytes received: 800
send:800812
Bytes received: 800
send:800913
Bytes received: 800
```

服务器端也可以观察到语音通话的相关标识：

```
[root@kp-test01 14732]# ./server
Waiting for Connect...
User 202.117.43.84 Has Connect!
ghx 123
syj 123
User syj 123 has login.
User 202.117.43.84 Has Connect!
ghx 123
User ghx 123 has login.
Receive VOICE CALL request.
MESSAGE SEND TO 1: VOICE
Waiting for Connect...
User 202.117.43.84 Has Connect!
User 202.117.43.84 Has Connect!
voice recive800
800
voice recive800
800
voice recive800
800
voice recive800
```

由于未对语音数据进行压缩处理，语音通话传输的数据量很大，在网络环境较差时可能导致语音断断续续，通信不佳。

语音通话功能测试完毕。

## 七、 实验结论

通过本次实验，本小组较深入地了解了有关 socket 编程的相关知识，对于 socket 通信有了较完整的认识，并基于 socket 原理成功实现了一个简单的 CSC 通信模式的聊天室程序。这个程序除了具备基本的用户验证、文字聊天功能外，还支持文件在线离线传输，更具备断点续传和语音通话功能。但即便如此，由于时间问题，该项目依旧还有很多不足，如服务器每次只能同时接入两



位用户，并且当用户退出之后服务器程序也会关闭，想要再次通信需要重新运行服务器程序；在语音通话功能中，如果网络环境较差可能会出现一方只能听到对方，却无法发出语音消息的现象。尽管如此，考虑到本实验旨在学习 socket 原理并掌握网络编程方法，因此我们认为本小组已经很好地完成了本次实验，达到了实验目的。但同时我们也意识到自己在项目分工、进度安排方面尚有很多不足之处，想要熟练掌握网络知识不能仅仅纸上谈兵，更需要自己亲自实践。

## 八、 总结及心得体会

本次实验大概是我们目前为止经历过历时最长，问题最多的实验之一了，从项目开始到结束，我们断断续续花了一个月的时间才写完这 1600 多行代码。在这个过程中出现了很多问题，也解决了很多问题。现在回过头来再看这一个月走过的脚印，我们认为可以分成两方面来总结。

第一部分是编程技术方面。通过这次实验，我们深刻认识到了代码规范和封装的重要性。在项目前期，网上关于 socket 的资料繁而不精，大部分情况下都要自己摸着石头过河。本来是想试着采用敏捷开发的方式进行迭代和完善，但随着功能的逐渐增加和封装维护的不及时，造成了代码的严重冗余和高度耦合，导致后面对代码的调试越来越困难，想要维护起来的时间成本也就越来越高。这也造成各种奇怪的问题层出不穷、难以找到问题根源，很多时候会因为一个小问题而花费大量时间去查错，并且由于代码耦合严重，经常会出现几个漏洞需要循环修复的现象。前期的编程不规范让我们在调试上花费了许多时间，以后如果再参与较大项目的开发，一定要注意编写的规范严谨。

第二部分是项目管理方面。由于开发周期长，又是两人的合作项目，因此项目进度和任务分配也很难做好。曾经和队友之间就因开发进度问题产生过矛盾，究其根本还是因为计划设置不明确、沟通交流不当导致的。今后无论在学习还是工作中团队合作肯定是必不可少的，这也提醒了我们在项目初期要合理规划进度安排，在开发过程中要注重团队成员之间的交流。

## 附件

### 1. 源码文件

见压缩包附件 server.cpp、client.cpp

### 2. 相关文档

README\_server: 服务器端程序的环境配置、编译、运行方法。

README\_client: 客户端程序的环境配置、编译、运行方法。

### 3. 参考资料

#### 1) Linux 环境下 Socket 编程:

[https://blog.csdn.net/weixin\\_42193813/article/details/105788066](https://blog.csdn.net/weixin_42193813/article/details/105788066)

#### 2) Windows 环境下 Socket 编程:

<https://docs.microsoft.com/en-us/windows/win32/winsock/creating-a-socket-for-the-client>

#### 3) Windows 下获取指定文件夹下所有文件名:

<https://blog.csdn.net/u012005313/article/details/50687297>

#### 4) 语音通话 mmeapi.h 头文件:

<https://docs.microsoft.com/en-us/windows/win32/api/mmeapi/nf-mmeapi-waveoutopen>

#### 5) 语音通话:

[https://blog.csdn.net/m0\\_38000900/article/details/73479799](https://blog.csdn.net/m0_38000900/article/details/73479799)