实 验 报 告

组号：[redacted]

姓名：[redacted] 学号：[redacted] 班级：计算机[redacted]

姓名：[redacted] 学号：[redacted] 班级：计算机[redacted]

一、 实验名称

Socket 网络编程实验，实现一个简单的聊天程序。

二、 实验原理

本实验基于 Client/Server 模式，主要使用 TCP 协议进行通信。

（1）服务器：

服务器通过多线程支持多个并发的 TCP 连接，主线程监听端口并等待接受客户端连接请求，连接建立后为每个客户端创建单独的线程，以处理客户端发来的消息并进行相应的转发工作。

（2）客户端：

客户端使用 python 的 tkinter 库进行 UI 设计，主要是通过文本框进行输入、输出，以及将按钮绑定到实现具体功能的函数。当客户端通过登录或注册认证后，将创建一个守护线程以接收服务器发来的消息，而主线程则负责响应客户端的操作，包括文字聊天、文件传输、语音通话等，此外还有部分子线程负责某些耗时长或者可能导致阻塞的操作，例如发送文件及语音、选择是否接收文件及语音等。

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注册与登录：账号只能含有数字或大小写字母且长度3~9，密码只能含有数字或大小写字母且长度6~9。用户注册时要求账号名不能重复，登录时只有账号密码与服务器存储的数据一致才能通过认证；

②文字聊天：支持多用户同时在线聊天、支持私聊与群聊、支持中英文以及多行文本输入与显示；

③文件传输：支持在用户间传输任意格式的大文件、支持文件传输的同时进行文字聊天。

2. 高级功能

①离线文件：当文件接收方离线时，服务器可以将文件暂存，等待接收方上线后再进行传输；

②断点续传：支持在线或离线文件的断点续传，当文件传输中断时，若服务器或接收方未删除已发送的临时文件，则恢复文件传输后，文件将从上次发送中断的地方开始传输；

③语音通话：支持语音通话的同时进行文字聊天，且语音清晰、延时较小；

④NAT穿透：通过UDP打洞实现全锥形NAT穿透、并能在此基础上进行P2P私聊。

五、 实验实现

1. 人员分工

[redacted]负责应用层协议设计、服务器及部分客户端编程、测试以及实验报告的撰写；

[redacted]负责部分客户端编程、测试以及实验报告的撰写。

2. 实验设计

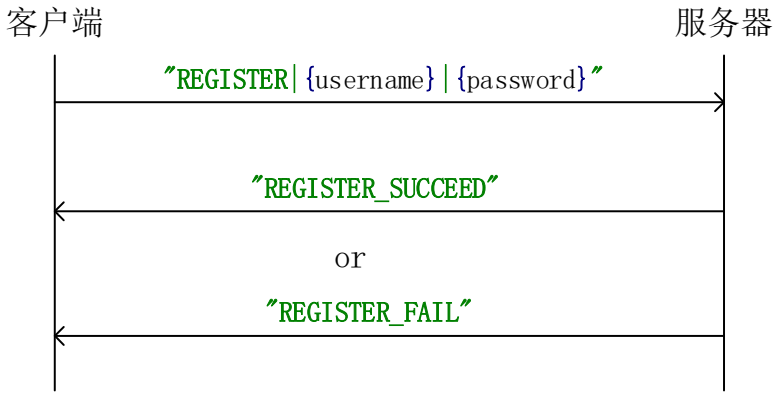
（一）协议

主要基于 Client/Server 模式，使用 TCP 协议进行通信。但在 NAT 穿透时，则是通过 UDP 打洞并进行 P2P 私聊。

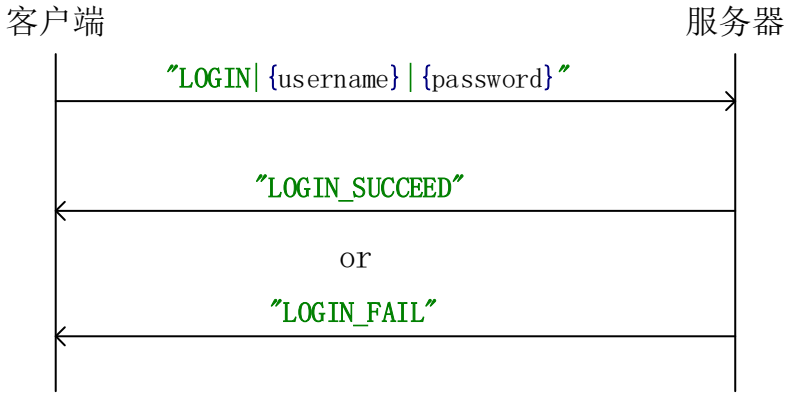
TCP 协议中消息的分割:每一条消息由消息头与主体两部分组成，消息头占 2 字节，用于表示主体长度（以字节为单位）。

主体的内容及客户端与服务器的交互过程如下（注：除文件和语音数据 data 外，其余消息均使用 UTF-8 编码，并通过“|”分隔各字段，接收时根据消息开头的关键字确定分隔符个数）：

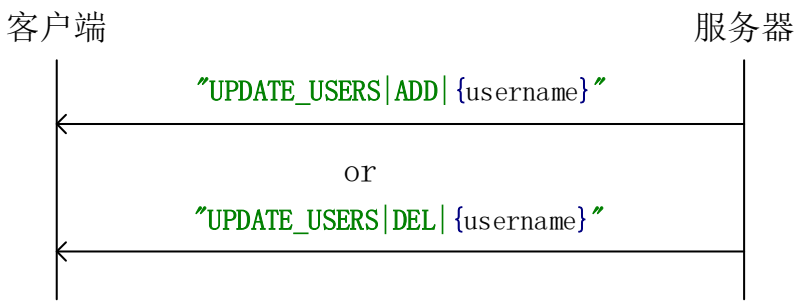
①注册：



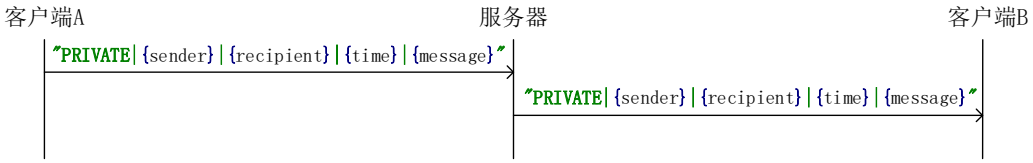
②登录：



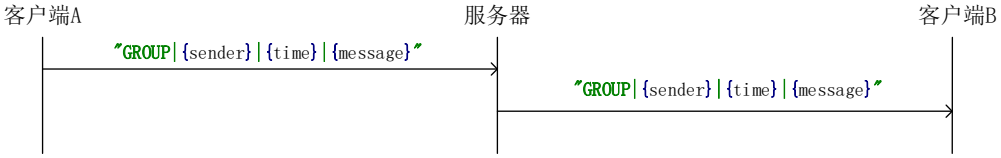
③更新在线用户列表：



④私聊：



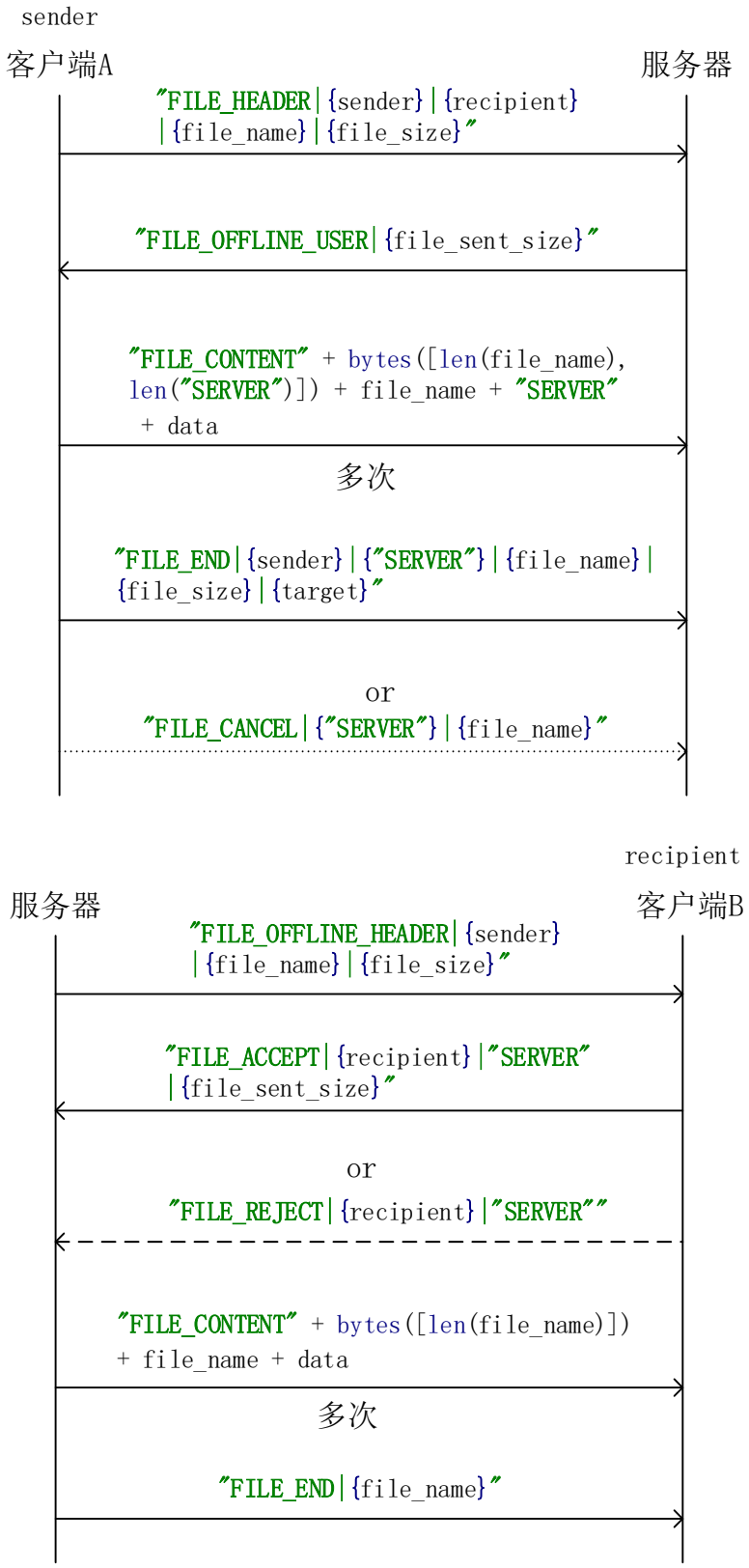
⑤群聊：



⑥在线文件传输：

说明：sent_size 字段用于断点续传，表示客户端 B 在之前传输中断时已接收到的文件大小（若有的话，否则其值为 0）。target 字段用于离线文件传输，表示实际接收方，这里 taget=recipient，在线文件传输时其值无意义。

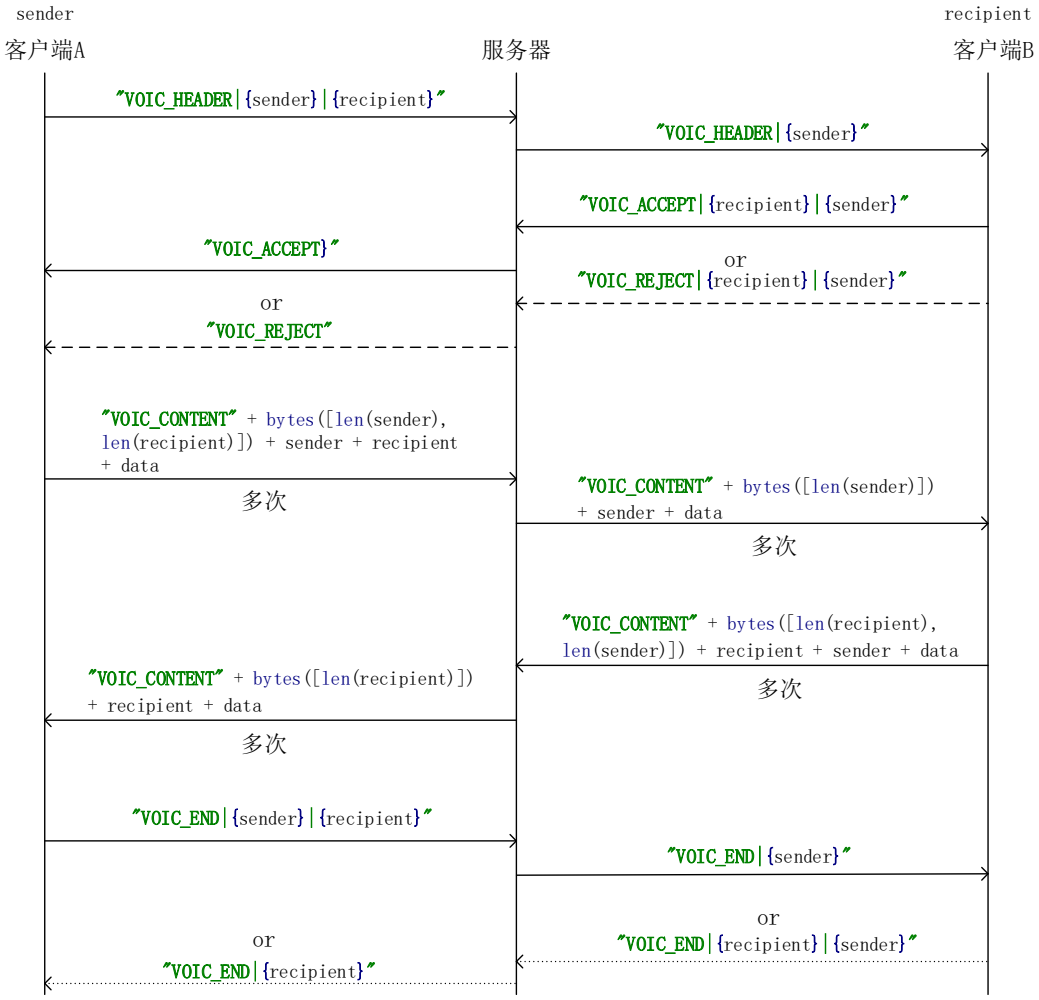
⑦离线文件传输：



说明：file_sent_size 字段用于断点续传，表示服务器在之前传输中断时已接收到的文件大小（若有的话，否则其值为 0）。target 字

段表示实际接收方，用于服务器记录离线文件接收方，这里taget=recipient。

⑧语音通话：

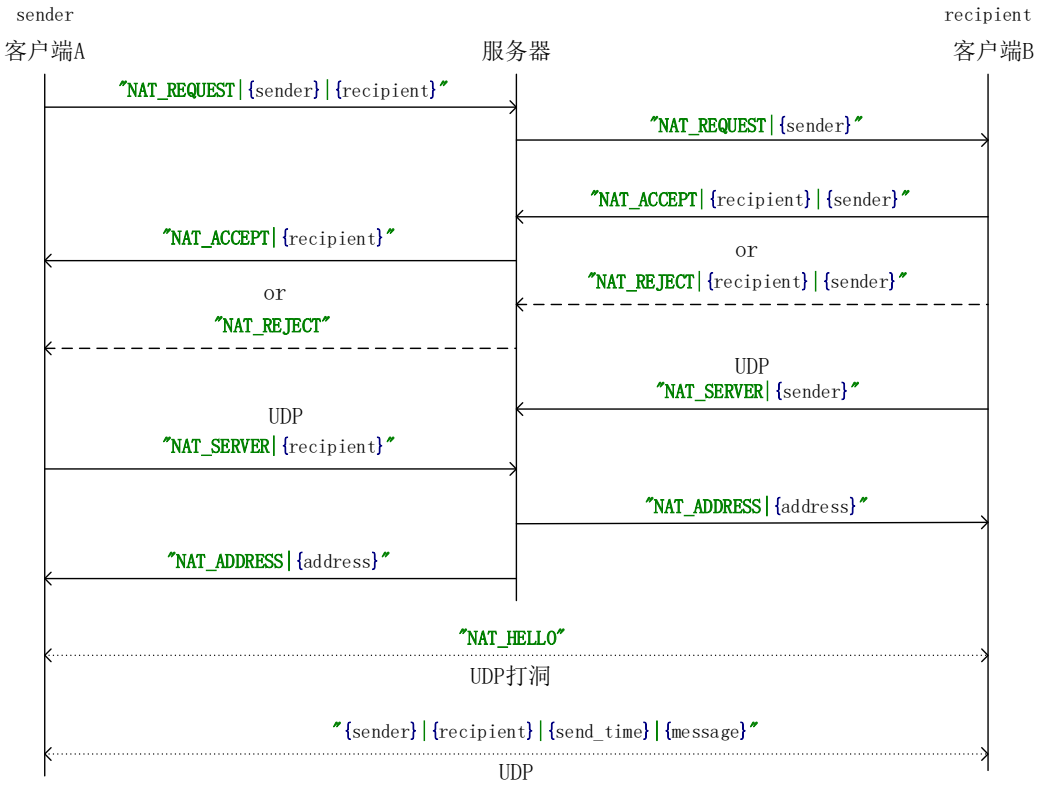


说明：语音通话双向进行，因此两个客户端都会将各自录制的语音数据发送给服务器，再由服务器进行转发。当任意一方关闭语音通话时，双方的的语音通话线程均结束。

⑨NAT 穿透：

说明：NAT 穿透时 TCP 与 UDP 配合完成通信，首先通过 TCP 询问对方是否进行 NAT 穿透，若同意则双方分别使用 UDP 向服务器发送消息，从而服务器可以获得双方的公网 IP 地址与端口号，然后服务器再将其转发给通信双方。客户端在获得对方地址后，通过 UDP 进行打

洞，成功后便可通过 UDP 进行 P2P 私聊。



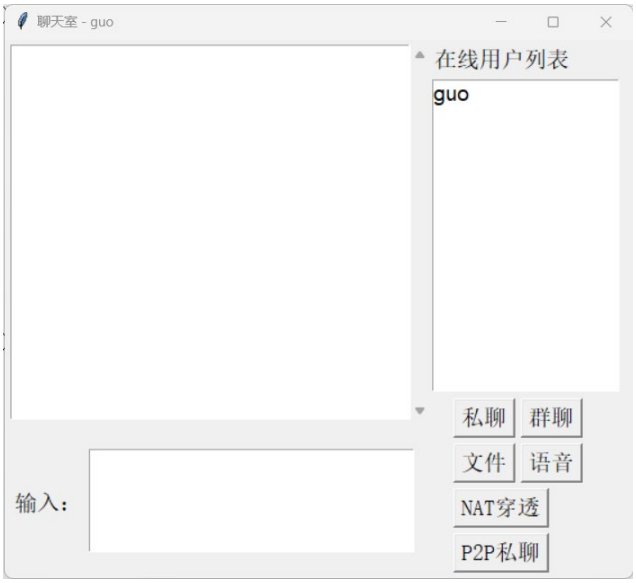
（二）UI 设计

使用 python 的 tkinter 库进行 UI 设计(下面展示部分 UI 界面)：

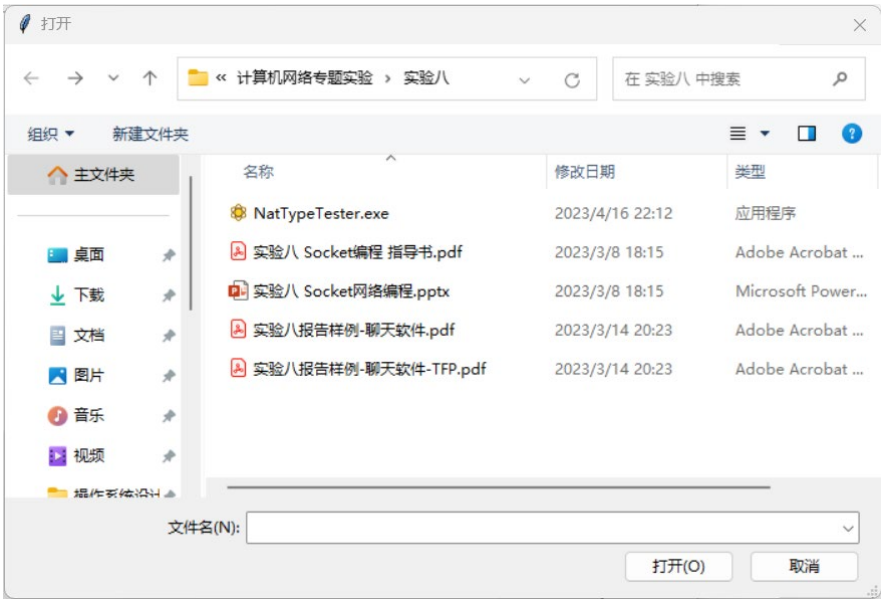
①登录界面：

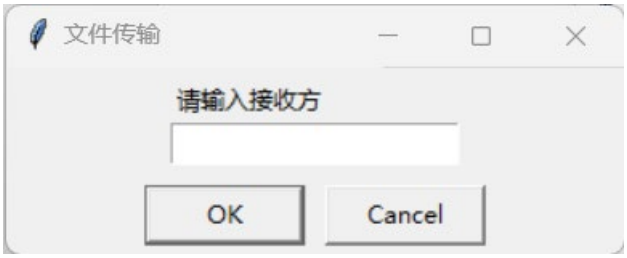
②聊天界面：

私聊时，点击右侧在线用户列表中的用户名，按下私聊按钮即可将消息发送给对方。



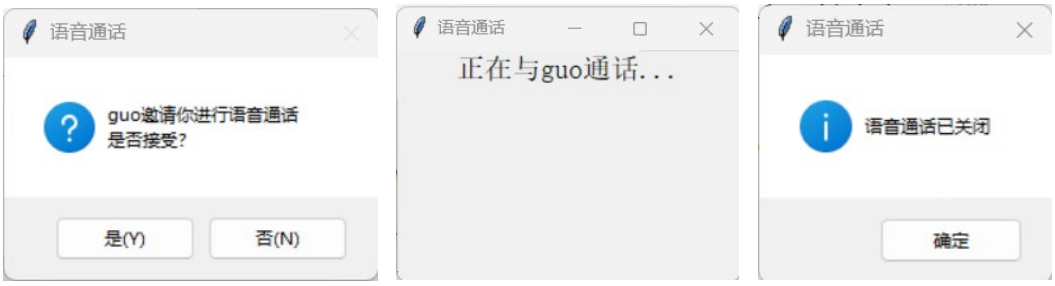
③文件传输：





④语音通话：

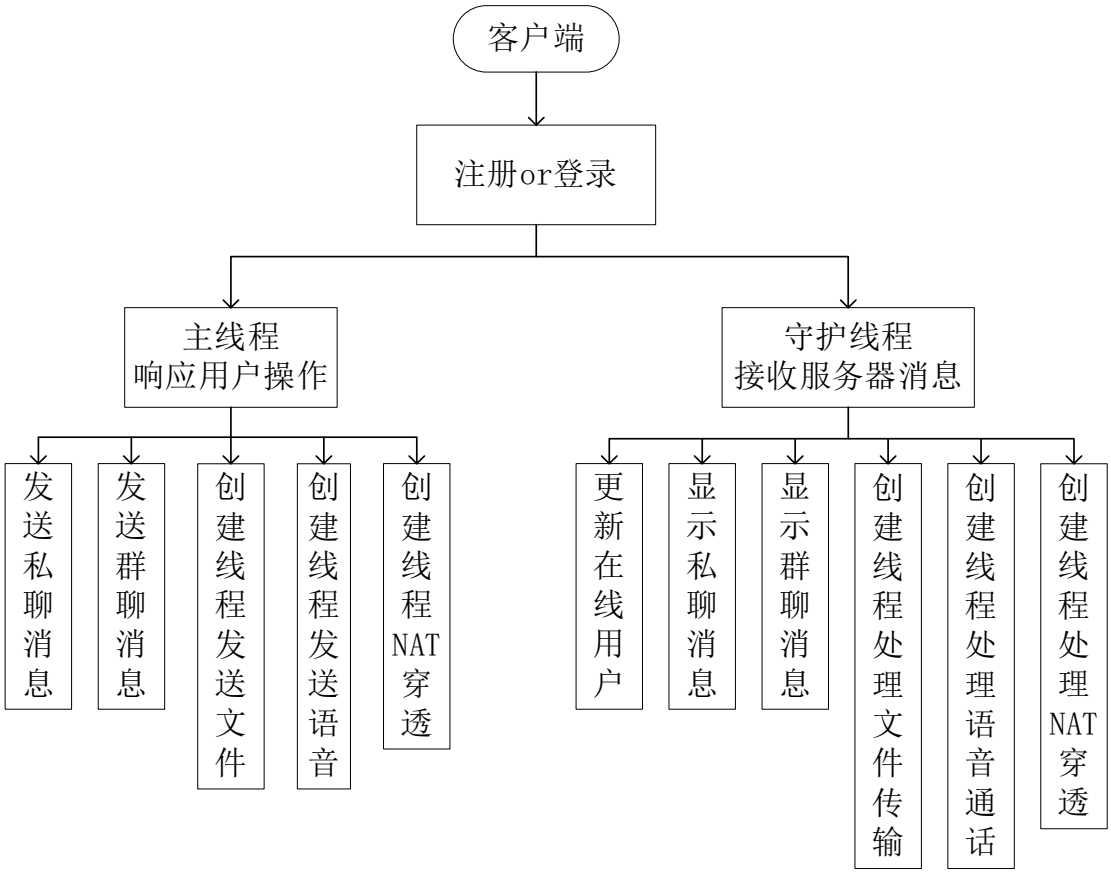
语音通话时，点击右侧在线用户列表中的用户名，按下语音按钮即可邀请对方进行语音通话。



⑤错误信息：

（三）框架结构

①客户端：



客户端程序中定义了 Client、LoginWindow、ChatWindow 三个类。其中，Client 类负责建立 TCP 连接、消息发送与接收；LoginWindow 类实现了登录界面 UI、注册以及登录功能；ChatWindow 类实现了聊

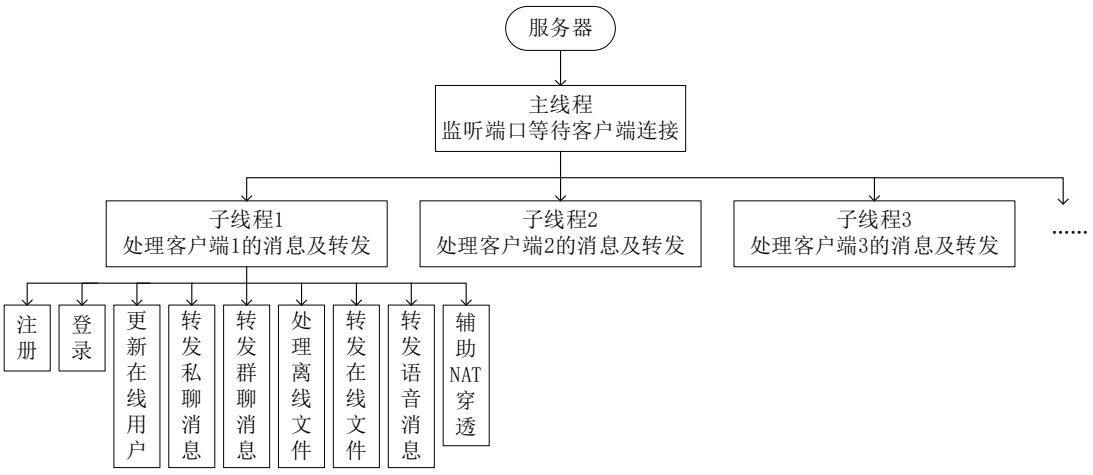
天界面UI、文字聊天、文件传输、语音通话以及NAT穿透功能。

在Client类中，发送消息时首先会计算数据包长度并将其封装在两个字节的消息头中，然后再将消息头与消息体一起发送给服务器；接收消息时首先会接收两个字节的消息头，然后再按照其中的数据包长度值接收指定大小的消息体，从而实现对不同消息的分割。

在LoginWindow类中，当用户输入账号密码后，客户端首先检查账号密码格式是否符合要求，若符合则将其发送给服务器，然后等待服务器发来登录(注册)成功或失败的消息，若成功则进入聊天界面，否则向用户报告失败原因。

在ChatWindow类中，主线程负责响应用户操作，包括读取文本框内容、选择私聊对象、执行按钮绑定的函数等。守护线程负责接收服务器发来的消息，并根据消息内容执行不同的处理函数。此外，在文件传输、语音通话、NAT穿透时，为了避免长时间等待对方应答而造成阻塞，客户端创建单独的线程执行这些操作，这使得客户端在进行文件传输或语音通话时能够正常收发消息。

②服务器：



服务器程序中定义了Server类，其主线程负责监听端口并等待客户端连接请求，每当服务器与客户端建立连接后，便新建一个子线程负责处理该客户端的消息并根据不同的消息执行相应的处理函数。

3. 关键代码的描述

（一）关键代码 1-注册

客户端代码编写：[redacted]。

当用户输入账号密码后，客户端首先检查格式是否符合要求，若符合则将其发送给服务器，然后等待服务器发来注册成功或失败的消息，若成功则进入聊天界面，否则向用户报告失败原因。

```
def register(self):
    self.client.username = self.entry_username.get()
    # 判断用户名是否合法
    if self.client.username.isalnum() and 2 <
len(self.client.username) < 10:
        password = self.entry_password.get()
        # 判断密码是否合法
        if password.isalnum() and 5 < len(password) < 10:
            # 发送注册请求

self.client.send_message(f"REGISTER|{self.client.username}|{passwor
d}")
            # 等待服务器发送注册确认
            message, data = self.client.receive_message()
            if message.startswith("REGISTER_SUCCEED"):
                self.window.destroy()
            elif message.startswith("REGISTER_FAIL"):
                messagebox.showerror('错误', "该账号已存在")
        else:
            messagebox.showerror('错误', "密码只能含有数字或大小写字母且长
度 6~9")
    else:
        messagebox.showerror('错误', "账号只能含有数字或大小写字母且长度
3~9")
```

服务器代码编写：[redacted]。

当服务器收到客户端发来的注册请求时，首先在记录用户信息的文件中查询注册的用户名是否存在，若存在则发送消息通知客户端注册失败，否则将新的用户名和密码记录在文件中，并向客户端发送注册成功消息，同时向在线用户发送消息以更新在线用户列表。

```
# 注册认证
def register(self, client_socket, message):
```

```
    parts = message.split("|", maxsplit=1)
    username = parts[0]
    password = parts[1]
    with open(self.user_file, "a+") as f:
        f.seek(0)
        lines = f.readlines()
        found = False
        # 查询用户是否存在
        for line in lines:
            parts = line.strip().split()
            if parts[0] == username:
                # 账号名已存在
                found = True
                self.send_message(client_socket, f"REGISTER_FAIL")
                break
        # 用户不存在则创建新用户
        if not found:
            # 服务器记录新账号与密码
            f.write(f"{username} {password}\n")
            # 记录用户套接字
            self.client_sock[username] = client_socket
            # 服务器日志记录用户注册
            print(f"{username} register at
{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
            # 向客户端发送注册成功消息
            self.send_message(client_socket, f"REGISTER_SUCCEED")
            # 更新在线用户列表
            self.update_online_users(username, "ADD")
```

（二）关键代码 2-登录

客户端代码编写：[redacted]。

当用户输入账号密码后，客户端首先检查格式是否符合要求，若符合则将其发送给服务器，然后等待服务器发来登录成功或失败的消息，若成功则进入聊天界面，否则向用户报告失败原因。

```
def login(self):
    self.client.username = self.entry_username.get()
    # 判断用户名是否合法
    if self.client.username.isalnum() and 2 <
len(self.client.username) < 10:
        password = self.entry_password.get()
        # 判断密码是否合法
        if password.isalnum() and 5 < len(password) < 10:
```

```
            # 发送登录认证
self.client.send_message(f"LOGIN|{self.client.username}|{password}"
)
            # 等待服务器发送登录确认
            message, data = self.client.receive_message()
            if message.startswith("LOGIN_SUCCEED"):
                self.window.destroy()
            elif message.startswith("LOGIN_FAIL"):
                messagebox.showerror('错误', "账号或密码错误")
        else:
            messagebox.showerror('错误', "密码只能含有数字或大小写字母且长
度 6~9")
    else:
        messagebox.showerror('错误', "账号只能含有数字或大小写字母且长度
3~9")
```

服务器代码编写：[redacted]。

当服务器收到客户端发来的登录请求时，首先在记录用户信息的文件中查询用户名是否存在以及密码是否正确，若符合则发送消息通知客户端登录成功，同时向在线用户发送消息以更新在线用户列表；否则发送消息通知客户端登录失败。此外，当登录成功时，服务器还需要检查是否有离线文件需要发送给该用户。

```
# 登录认证
def login(self, client_socket, message):
    parts = message.split("|", maxsplit=1)
    username = parts[0]
    password = parts[1]
    with open(self.user_file, "r") as f:
        f.seek(0)
        lines = f.readlines()
        found = False
        # 查询用户是否存在
        for line in lines:
            parts = line.strip().split()
            if len(parts) == 2 and parts[0] == username:
                found = True
                # 判断密码是否正确
                if parts[1] == password:
                    # 记录用户套接字
                    self.client_sock[username] = client_socket
```

```
                    # 服务器日志记录用户登录
                    print(f"{username} login at
{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
                    # 向客户端发送登录成功消息
                    self.send_message(client_socket, f"LOGIN_SUCCEED")
                    # 更新在线用户列表
                    self.update_online_users(username, "ADD")
                    # 查询是否有离线文件需要发送给该客户端
                    threading.Thread(target=self.offline_file).start()
                else:
                    # 账号与密码不匹配
                    self.send_message(client_socket, f"LOGIN_FAIL")
                break
        if not found:
            # 用户不存在
            self.send_message(client_socket, f"LOGIN_FAIL")
```

（三）关键代码 3-更新在线用户列表

客户端代码编写：[redacted]

当客户端收到服务器发来的更新用户列表消息时，若关键字为“ADD”，则在用户列表中插入消息中的用户名；若关键字为“DEL”时，则将用户列表中的对应用户名删除。

```
# 更新在线用户列表
def update_online_users(self, message):
    user = message[4:]
    # 用户上线
    if message.startswith("ADD"):
        self.user_list.insert(tk.END, user)
    # 用户离线
    elif message.startswith("DEL"):
        for i in range(self.user_list.size()):
            if self.user_list.get(i) == user:
                self.user_list.delete(i)
                break
```

服务器代码编写：[redacted]

当用户上线时，服务器调用下面的函数，令 Op=“ADD”，以通知在线用户新的用户上线，并向新上线用户发送全部在线用户的列表；当用户离线时，服务器调用下面的函数，令 Op=“DEL”，以通知在线用户有用户离线。

```
# 更新在线用户列表
def update_online_users(self, user, Op):
    # 向已在线用户广播更新消息
    for username, sock in self.client_sock.items():
        self.send_message(sock, f"UPDATE_USERS|{Op}|{user}")
    # 向新上线用户发送在线用户列表
    if Op == "ADD":
        for username, sock in self.client_sock.items():
            if user != username:
                self.send_message(self.client_sock[user],
f"UPDATE_USERS|{Op}|{username}")
```

（四）关键代码 4-私聊

客户端代码编写：[redacted]。

当用户选中私聊对象并按下按钮后，客户端便读取文本框中的内容并将其发送给服务器，再由服务器转发给私聊对象；当客户端收到服务器发来的私聊消息后，便将其显示在聊天界面中。

```
# 发送私聊消息
def send_private_message(self):
    # 获取选中的私聊用户
    recipient = self.user_list.get("anchor")
    if recipient:
        message = self.input.get("1.0", tk.END)
        if message:
            send_time =
datetime.datetime.now().strftime('%Y-%m-%d %H:%M')
            # 显示在聊天界面
            self.textbox.insert(tk.END, f"{self.client.username} ->
{recipient} ({send_time}):\n{message}\n")
            self.textbox.see(tk.END)
            # 向服务器发送消息

self.client.send_message(f"PRIVATE|{self.client.username}|{recipien
t}|{send_time}|{message}")
        else:
            messagebox.showerror("错误", "发送的消息不能为空")
    else:
        messagebox.showerror("错误", "未选择私聊对象")

# 接收私聊消息
def recv_private_message(self, message):
    parts = message.split("|", maxsplit=3)
```

```
    sender = parts[0]
    recipient = parts[1]
    send_time = parts[2]
    msg = parts[3]
    if recipient == self.client.username:
        self.textbox.insert(tk.END, f"{sender} -> {recipient}
({send_time}):\n{msg}\n")
        self.textbox.see(tk.END)
```

服务器代码编写：[redacted]。

当服务器收到客户端发来的私聊消息后，便提取消息中的接收方信息，将该私聊消息转发给对应接收方。

```
# 发送私聊消息
def private_message(self, client_socket, message):
    parts = message.split("|", maxsplit=2)
    sender = parts[0]
    recipient = parts[1]
    message = parts[2]
    if recipient in self.client_sock:
        recipient_socket = self.client_sock[recipient]
        if recipient_socket != client_socket:
            self.send_message(recipient_socket,
f"PRIVATE|{sender}|{recipient}|{message}")
```

（五）关键代码 5-群聊

客户端代码编写：[redacted]。

当用户按下群聊按钮后，客户端便读取文本框中的内容并将其发送给服务器，再由服务器转发给所有在线用户；当客户端收到服务器发来的群聊消息后，便将其显示在聊天界面中。

```
# 发送群聊消息
def send_group_message(self):
    message = self.input.get("1.0", tk.END)
    if message:
        send_time =
datetime.datetime.now().strftime('%Y-%m-%d %H:%M')
        # 显示在聊天界面
        self.textbox.insert(tk.END, f"{self.client.username} ->
public ({send_time}):\n{message}\n")
        self.textbox.see(tk.END)
        # 向服务器发送消息
```

```
self.client.send_message(f"GROUP|{self.client.username}|{send_time}
|{message}")
    else:
        messagebox.showerror("错误", "发送的消息不能为空")

# 接收群聊消息
def recv_group_message(self, message):
    parts = message.split("|", maxsplit=2)
    sender = parts[0]
    send_time = parts[1]
    msg = parts[2]
    self.textbox.insert(tk.END, f"{sender} -> public
({send_time}):\n{msg}\n")
    self.textbox.see(tk.END)
```

服务器代码编写：[redacted]。

当服务器收到客户端发来的群聊消息后，便该消息转发给当前所有在线用户。

```
# 发送群聊消息
def public_message(self, client_socket, message):
    parts = message.split("|", maxsplit=1)
    sender = parts[0]
    message = parts[1]
    for username, sock in self.client_sock.items():
        if sock != client_socket:
            self.send_message(sock, f"GROUP|{sender}|{message}")
```

（六）关键代码 6-文件传输

客户端代码编写：[redacted]。

用户首先选择要发送的文件及接收方，然后将相应信息发送给服务器，再由服务器转发给接收方，之后便等待接收方应答。当接收方同意接收文件后，发送方便将文件切分为 32KB 的段进行发送，在这期间，当文件传输中断或者发送完成时都会发送相应的信息给接收方。此外，为了在文件传输的同时仍能进行文字聊天，上述步骤都在新的线程中进行。

断点续传的实现方法：接收方收到文件传输请求时，首先检查本地是否已经存在待发送文件的部分，若存在则计算其大小并通知发送

方，发送方再从未发送的部分开始发送。

离线文件的实现方法：当文件接收方离线时，服务器将通知发送方，于是发送方便将文件发送给服务器。服务器临时存储离线文件，并记录该离线文件的接收方，每当有用户上线时，服务器便检查是否有离线文件需要发送给该用户，若有则进行文件传输，其步骤与在线文件传输类似。

```
# 发送文件
def send_file(self):
    # 选择文件
    file_path = filedialog.askopenfilename()
    if file_path:
        file_name = os.path.basename(file_path)
        file_size = os.path.getsize(file_path)
        # 选择接收方
        recipient = simpledialog.askstring("文件传输", "请输入接收方")
        if recipient:

self.client.send_message(f"FILE_HEADER|{self.client.username}|{reci
pient}|{file_name}|{file_size}")
            threading.Thread(target=self.send_file_thread,
args=(recipient, file_path, file_size, )).start()

def send_file_thread(self, recipient, file_path, file_size):
    # 等待应答
    while not self.file_state:
        continue
    # 接收方不存在
    if self.file_state == 1:
        messagebox.showerror("错误", "文件接收方不存在")
        self.file_state = 0
    # 对方拒绝接收
    elif self.file_state == 2:
        messagebox.showerror("错误", "对方拒绝接收文件")
        self.file_state = 0
    # 开始发送
    elif self.file_state == 3 or self.file_state == 4:
        file_name = os.path.basename(file_path)
        target = recipient
        if self.file_state == 4:
            recipient = "SERVER"    # 离线文件的接收方为服务器
```

```
        # UI 界面
        file_window = tk.Toplevel(self.window)
        file_window.title("文件传输")
        file_window.protocol("WM_DELETE_WINDOW", self.file_cancel)
        tk.Label(file_window, text=f"{file_name}文件上传中",
font=10).pack()
        # 控制信息
        message = f"FILE_CONTENT|{file_name}|{recipient}"
        with open(file_path, "rb") as f:
            # 定位到先前已发送的位置
            f.seek(self.file_sent_size)
            total_sent = self.file_sent_size
            # 发送文件数据
            while self.file_state != 0 and total_sent < file_size:
                data = f.read(2**15)
                total_sent += len(data)
                self.client.send_message(message, data)
        file_window.destroy()
        self.file_sent_size = 0
        if self.file_state != 0:
            # 发送完成

self.client.send_message(f"FILE_END|{self.client.username}|{recipie
nt}|{file_name}|{file_size}|{target}")
            messagebox.showinfo("文件传输", f"{file_name}文件发送完成")
            self.file_state = 0
        else:
            # 发送取消

self.client.send_message(f"FILE_CANCEL|{recipient}|{file_name}")

# 取消文件发送
def file_cancel(self):
    self.file_state = 0

# 接收文件
def recv_file(self, message, data):
    # 询问是否接收在线文件
    if message.startswith("HEADER"):
        parts = message.split("|", maxsplit=3)
        sender = parts[1]
        file_name = parts[2]
        file_size = int(parts[3])
        threading.Thread(target=self.recv_file_thread, args=(sender,
```

```
file_name, file_size, 1)).start()
    # 询问是否接收离线文件
    elif message.startswith("OFFLINE_HEADER"):
        parts = message.split("|", maxsplit=3)
        sender = parts[1]
        file_name = parts[2]
        file_size = int(parts[3])
        threading.Thread(target=self.recv_file_thread, args=(sender,
file_name, file_size, 2)).start()
    # 接收文件
    elif message.startswith("CONTENT"):
        parts = message.split("|", maxsplit=1)
        file_name = parts[1]
        with open(file_name+".tmp", "ab") as f:
            f.write(data)
    # 接收完成
    elif message.startswith("END"):
        parts = message.split("|", maxsplit=1)
        file_name = parts[1]
        os.rename(file_name + ".tmp", file_name)
        threading.Thread(target=self.recv_file_thread, args=("",
file_name, "", 3)).start()
    # 传输中断
    elif message.startswith("CANCEL"):
        parts = message.split("|", maxsplit=1)
        file_name = parts[1]
        threading.Thread(target=self.recv_file_thread, args=("",
file_name, "", 4)).start()
    # 接收方不存在
    elif message.startswith("USER_NO_EXIST"):
        self.file_state = 1
    # 对方拒绝接收
    elif message.startswith("REJECT"):
        self.file_state = 2
    # 通知发送线程开始发送在线文件
    elif message.startswith("ACCEPT"):
        self.file_sent_size = int(message[7:])
        self.file_state = 3
    # 通知发送线程开始发送离线文件
    elif message.startswith("OFFLINE_USER"):
        self.file_sent_size = int(message[13:])
        self.file_state = 4

def recv_file_thread(self, sender, file_name, file_size, mode):
```

```
    # 询问是否接收在线文件
    if mode == 1:
        if messagebox.askyesno("文件传输", f"{sender}给你发送了一个文件:
\n{file_name}({file_size}Bytes)\n 是否接收？"):
            file_sent_size = 0
            # 获取先前已接收的文件大小-断点续传
            if os.path.isfile(file_name+".tmp"):
                file_sent_size = os.path.getsize(file_name+".tmp")

self.client.send_message(f"FILE_ACCEPT|{self.client.username}|{send
er}|{file_sent_size}")
        else:

self.client.send_message(f"FILE_REJECT|{self.client.username}|{send
er}")
    # 询问是否接收离线文件
    elif mode == 2:
        if messagebox.askyesno("文件传输", f"{sender}给你发送了一个离线文
件: \n{file_name}({file_size}Bytes)\n 是否接收？"):
            file_sent_size = 0
            # 获取先前已接收的文件大小-断点续传
            if os.path.isfile(file_name+".tmp"):
                file_sent_size = os.path.getsize(file_name+".tmp")

self.client.send_message(f"FILE_ACCEPT|{self.client.username}|SERVE
R|{file_sent_size}")
        else:

self.client.send_message(f"FILE_REJECT|{self.client.username}|SERVE
R")
    # 文件接收完成
    elif mode == 3:
        messagebox.showinfo("文件传输", f"{file_name}文件接收完成")
    # 文件传输中断
    elif mode == 4:
        messagebox.showinfo("文件传输", f"{file_name}文件传输中断")
```

服务器代码编写：[redacted]。

当服务器收到客户端的发送文件请求时，首先检查接收方是否存在，若不存在则通知发送方；否则检查其是否在线，若在线则服务器的主要工作就是在发送方与接收方之间转发消息。

当接收方离线时，服务器通知发送方，并临时存储离线文件以及

记录相应的文件与接收方信息。每当有用户上线时，服务器便检查是否有离线文件需要发送给该用户，若有则进行文件传输，并在结束后删除该文件以及对应的记录。

```
# 发送文件
def send_file(self, client_socket, message, data):
    # 处理发送文件请求
    if message.startswith("HEADER"):
        parts = message.split("|", maxsplit=4)
        sender = parts[1]
        recipient = parts[2]
        file_name = parts[3]
        file_size = parts[4]
        # 查看接收方是否存在
        with open(self.user_file, "r") as f:
            f.seek(0)
            lines = f.readlines()
            found = False
            for line in lines:
                parts = line.strip().split()
                if parts[0] == recipient:
                    found = True
                    if recipient in self.client_sock:
                        # 接收方在线
                        self.send_message(self.client_sock[recipient],
f"FILE_HEADER|{sender}|{file_name}|{file_size}")
                    else:
                        # 接收方离线
                        file_sent_size = 0
                        # 查看服务器是否存在发送未完成的临时文件
                        if os.path.isfile(file_name+".tmp"):
                            file_sent_size =
os.path.getsize(file_name+".tmp")
                        # 通知客户端发送离线文件
                        self.send_message(client_socket,
f"FILE_OFFLINE_USER|{file_sent_size}")
                    break
            if not found:
                # 接收方不存在
                self.send_message(client_socket, "FILE_USER_NO_EXIST")
    # 发送文件内容
    elif message.startswith("CONTENT"):
        parts = message.split("|", maxsplit=2)
```

```
            file_name = parts[1]
            recipient = parts[2]
            if recipient == "SERVER":
                # 服务器本地存储离线文件
                with open(file_name + ".tmp", "ab") as f:
                    f.write(data)
            else:
                # 向接收方转发在线文件
                self.send_message(self.client_sock[recipient],
f"FILE_CONTENT|{file_name}", data)
        # 文件发送完成
        elif message.startswith("END"):
            parts = message.split("|", maxsplit=5)
            sender = parts[1]
            recipient = parts[2]
            file_name = parts[3]
            file_size = parts[4]
            target = parts[5]
            if recipient == "SERVER":
                # 服务器日志记录离线文件任务
                os.rename(file_name + ".tmp", file_name)
                with open("offline_file.txt", "ab") as f:

f.write(f"{sender}|{target}|{file_name}|{file_size}|_READY\n".encod
e("utf-8"))
                threading.Thread(target=self.offline_file).start()
            else:
                # 通知接收方文件发送完成
                self.send_message(self.client_sock[recipient],
f"FILE_END|{file_name}")
        # 文件发送取消
        elif message.startswith("CANCEL"):
            parts = message.split("|", maxsplit=2)
            recipient = parts[1]
            file_name = parts[2]
            if recipient != "SERVER":
                self.send_message(self.client_sock[recipient],
f"FILE_CANCEL|{file_name}")
        # 对方拒绝接收文件
        elif message.startswith("REJECT"):
            parts = message.split("|", maxsplit=2)
            recipient = parts[2]
            if recipient == "SERVER":
                self.file_state = 1
```

```
            else:
                self.send_message(self.client_sock[recipient],
f"FILE_REJECT")
        # 对方同意接收文件
        elif message.startswith("ACCEPT"):
            parts = message.split("|", maxsplit=3)
            recipient = parts[2]
            file_sent_size = parts[3]
            if recipient == "SERVER":
                self.file_sent_size = int(file_sent_size)
                self.file_state = 2
            else:
                self.send_message(self.client_sock[recipient],
f"FILE_ACCEPT|{file_sent_size}")

    # 处理离线文件
    def offline_file(self):
        if os.path.isfile("offline_file.txt"):
            # 查询日志
            with open("offline_file.txt", "rb+") as f:
                f.seek(0)
                line = f.readline().decode("utf-8").rstrip()
                while line:
                    if line.endswith("FINISH"):
                        # 离线文件传输已完成
                        line = f.readline().decode("utf-8").rstrip()
                        continue
                    parts = line.split("|")
                    sender = parts[0]
                    recipient = parts[1]
                    file_name = parts[2]
                    file_size = parts[3]
                    if recipient not in self.client_sock:
                        # 接收方离线
                        line = f.readline().decode("utf-8").rstrip()
                        continue
                    # 接收方上线后，询问是否接收文件
                    self.send_message(self.client_sock[recipient],
f"FILE_OFFLINE_HEADER|{sender}|{file_name}|{file_size}")
                    # 等待应答
                    while not self.file_state:
                        continue
                    # 发送离线文件
                    if self.file_state == 2:
```

```
                message = f"FILE_CONTENT|{file_name}"
                with open(file_name, "rb") as f1:
                    # 定位到上次文件传输中断的位置
                    f1.seek(self.file_sent_size)
                    total_sent = self.file_sent_size
                    while total_sent < int(file_size):
                        data = f1.read(2 ** 15)
                        total_sent += len(data)

self.send_message(self.client_sock[recipient], message, data)
                self.file_sent_size = 0
                self.send_message(self.client_sock[recipient],
f"FILE_END|{file_name}")
            # 服务器本地删除文件
            os.remove(file_name)
            # 服务器日志记录离线传输完成
            f.seek(-7, 1)
            f.write("FINISH\n".encode("utf-8"))
            self.file_state = 0
            # 读取下一条记录
            line = f.readline().decode("utf-8").rstrip()
```

（七）关键代码 7-语音通话

客户端代码编写：[redacted]。

语音通话的流程与文件传输类似，不同点在于语音通话要求双方均在线且数据的传输是双向的，当用户 B 同意用户 A 发起的语音通话时，用户 B 也应该创建一个新线程用于录制并发送语音数据给用户 A。

```
# 发送语音
def send_voice(self):
    # 选择接收方
    recipient = self.user_list.get("anchor")
    if recipient:

self.client.send_message(f"VOIC_HEADER|{self.client.username}|{reci
pient}")
        # 创建发送线程
        threading.Thread(target=self.send_voice_thread,
args=(recipient, )).start()
    else:
        messagebox.showerror("错误", "未选择接听方")

def send_voice_thread(self, recipient):
```

```
    # 等待应答
    while not self.voice_state:
        continue
    # 对方拒绝接听
    if self.voice_state == 1:
        messagebox.showerror("错误", "对方拒绝接听")
        self.voice_state = 0
    # 开始发送
    elif self.voice_state == 2:
        # UI 界面
        voice_window = tk.Toplevel(self.window)
        voice_window.title("语音通话")
        tk.Label(voice_window, text=f"正在与{recipient}通话...",
font=10).pack()
        voice_window.protocol("WM_DELETE_WINDOW", self.close_voice)
        # 控制信息
        message = f"VOIC_CONTENT|{self.client.username}|{recipient}"
        # 发送语音数据
        while self.voice_state == 2:
            data = self.recording_stream.read(1024)
            self.client.send_message(message, data)
        # 语音通话结束
        voice_window.destroy()

self.client.send_message(f"VOIC_END|{self.client.username}|{recipie
nt}")

# 结束语音通话
def close_voice(self):
    self.voice_state = 0

# 接收语音
def recv_voice(self, message, data):
    # 询问是否进行语音通话
    if message.startswith("HEADER"):
        parts = message.split("|", maxsplit=1)
        sender = parts[1]
        threading.Thread(target=self.recv_voice_thread,
args=(sender, 1)).start()
    # 接听语音
    elif message.startswith("CONTENT"):
        self.playing_stream.write(data)
    # 对方关闭语音通话
    elif message.startswith("END"):
```

```
            parts = message.split("|", maxsplit=1)
            sender = parts[1]
            self.voice_state = 0
            threading.Thread(target=self.recv_voice_thread,
args=(sender, 2)).start()
        # 对方拒绝接听
        elif message.startswith("REJECT"):
            self.voice_state = 1
        # 通知发送线程开始发送语音
        elif message.startswith("ACCEPT"):
            self.voice_state = 2

    def recv_voice_thread(self, sender, mode):
        # 询问是否同意语音通话
        if mode == 1:
            if messagebox.askyesno("语音通话", f"{sender}邀请你进行语音通话\n
是否接受？"):

self.client.send_message(f"VOIC_ACCEPT|{self.client.username}|{send
er}")
                # 开启语音发送线程
                threading.Thread(target=self.send_voice_thread,
args=(sender,)).start()
                self.voice_state = 2
            else:

self.client.send_message(f"VOIC_REJECT|{self.client.username}|{send
er}")
        # 语音通话关闭
        elif mode == 2:
            messagebox.showinfo("语音通话", "语音通话已关闭")
```

服务器代码编写：[redacted]。

在语音通话的过程中，服务器的主要工作是在通话双方之间转发消息。

```
# 发送语音
def send_voice(self, message, data):
    # 处理发送语音请求
    if message.startswith("HEADER"):
        parts = message.split("|", maxsplit=2)
        sender = parts[1]
        recipient = parts[2]
        self.send_message(self.client_sock[recipient],
```

```
f"VOIC_HEADER|{sender}")
    # 发送语音内容
    elif message.startswith("CONTENT"):
        parts = message.split("|", maxsplit=2)
        sender = parts[1]
        recipient = parts[2]
        self.send_message(self.client_sock[recipient],
f"VOIC_CONTENT|{sender}", data)
    # 关闭语音通话
    elif message.startswith("END"):
        parts = message.split("|", maxsplit=2)
        sender = parts[1]
        recipient = parts[2]
        self.send_message(self.client_sock[recipient],
f"VOIC_END|{sender}")
    # 对方拒绝接收语音
    elif message.startswith("REJECT"):
        parts = message.split("|", maxsplit=2)
        recipient = parts[2]
        self.send_message(self.client_sock[recipient],
f"VOIC_REJECT")
    # 对方同意接收语音
    elif message.startswith("ACCEPT"):
        parts = message.split("|", maxsplit=2)
        recipient = parts[2]
        self.send_message(self.client_sock[recipient],
f"VOIC_ACCEPT")
```

（八）关键代码 8-NAT 穿透

客户端代码编写：[redacted]。

在利用 NAT 穿透进行 P2P 私聊时，用户 A 首先询问用户 B 是否进行 NAT 穿透（通过服务器转发该消息），若用户 B 同意，则双方分别向服务器发送 UDP 报文，从而使得服务器知道双方的公网 IP 地址与端口号，服务器再将该信息发送给双方。通信双方在知道对方的公网 IP 地址与端口号后便可以进行 UDP 打洞，从而实现 NAT 穿透以及后续的 P2P 私聊。

```
# NAT 穿透（全锥形）
def nat_request(self):
    # 选择连接方
    recipient = self.user_list.get("anchor")
```

```
        if recipient:

self.client.send_message(f"NAT_REQUEST|{self.client.username}|{reci
pient}")
        else:
            messagebox.showerror("错误", "未选择连接方")

    def nat_handle(self, message):
        # 询问是否同意 NAT 穿透
        if message.startswith("REQUEST"):
            parts = message.split("|", maxsplit=1)
            sender = parts[1]
            threading.Thread(target=self.nat_thread, args=(sender,
1)).start()
        # 同意 NAT 穿透
        elif message.startswith("ACCEPT"):
            parts = message.split("|", maxsplit=1)
            sender = parts[1]
            sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            # 向服务器发送消息以通知对方自己的公网 ip 地址与端口
            sock.sendto(f"NAT_SERVER|{sender}".encode("utf-8"),
("8.130.66.246", 6666))
            # 开启 P2P 通信线程
            threading.Thread(target=self.nat_send, args=(sock,
sender)).start()
            threading.Thread(target=self.nat_recv, args=(sock,)).start()
        # 拒绝 NAT 穿透
        elif message.startswith("REJECT"):
            threading.Thread(target=self.nat_thread, args=("",
2)).start()
        # 获取对方公网 ip 地址与端口
        elif message.startswith("ADDRESS"):
            parts = message.split("|", maxsplit=1)
            self.MyPeer = eval(parts[1])

    def nat_thread(self, sender, mode):
        # 询问是否同意 NAT 穿透
        if mode == 1:
            if messagebox.askyesno("NAT 穿透", f"{sender}请求建立连接\n 是否
接受？"):

self.client.send_message(f"NAT_ACCEPT|{self.client.username}|{sende
r}")
                sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
            time.sleep(1)
            # 向服务器发送消息以通知对方自己的公网 ip 地址与端口
            sock.sendto(f"NAT_SERVER|{sender}".encode("utf-8"),
("8.130.66.246", 6666))
            # 开启 P2P 通信线程
            threading.Thread(target=self.nat_send, args=(sock,
sender)).start()
            threading.Thread(target=self.nat_recv,
args=(sock,)).start()
        else:

self.client.send_message(f"NAT_REJECT|{self.client.username}|{sende
r}")
    # 对方拒绝 NAT 穿透
    elif mode == 2:
        messagebox.showerror("错误", "对方拒绝建立连接")

def nat_send(self, sock, recipient):
    # 等待服务器通知对方的公网 ip 地址与端口
    while not self.MyPeer:
        continue
    print(f"对方公网 ip 地址与端口号：{self.MyPeer}")
    # 尝试 UDP 打洞
    for i in range(5):
        sock.sendto("NAT_HELLO".encode("utf-8"), self.MyPeer)
        time.sleep(1)
    self.P2P = False
    # 发送 P2P 私聊消息
    while True:
        if self.P2P:
            message = self.input.get("1.0", tk.END)
            if message:
                send_time =
datetime.datetime.now().strftime('%Y-%m-%d %H:%M')
                # 显示在聊天界面
                self.textbox.insert(tk.END, f"{self.client.username}
-> {recipient} ({send_time}):\n{message}\n")
                self.textbox.see(tk.END)
                try:
                    # 发送消息

sock.sendto(f"{self.client.username}|{recipient}|{send_time}|{messa
ge}".encode("utf-8"), self.MyPeer)
                except socket.error as e:
```

```
                    # 连接出错
                    print(e)
                    sock.close()
                    self.MyPeer = None
                    self.P2P = False
                    raise SystemExit
            else:
                messagebox.showerror("错误", "发送的消息不能为空")
            self.P2P = False

def nat_recv(self, sock):
    while True:
        try:
            # 接收消息
            message = sock.recvfrom(1024)[0].decode("utf-8")
        except socket.error as e:
            # 连接出错
            print(e)
            sock.close()
            self.MyPeer = None
            self.P2P = False
            raise SystemExit
        else:
            if message.startswith("NAT_HELLO"):
                # UDP 打洞成功
                print("connect!")
                continue
            # 接收 P2P 私聊消息
            parts = message.split("|", maxsplit=3)
            sender = parts[0]
            recipient = parts[1]
            send_time = parts[2]
            msg = parts[3]
            if recipient == self.client.username:
                self.textbox.insert(tk.END, f"(P2P){sender} ->
{recipient} ({send_time}):\n{msg}\n")
                self.textbox.see(tk.END)

def P2P_chat(self):
    self.P2P = True
```

服务器代码编写：[redacted]。

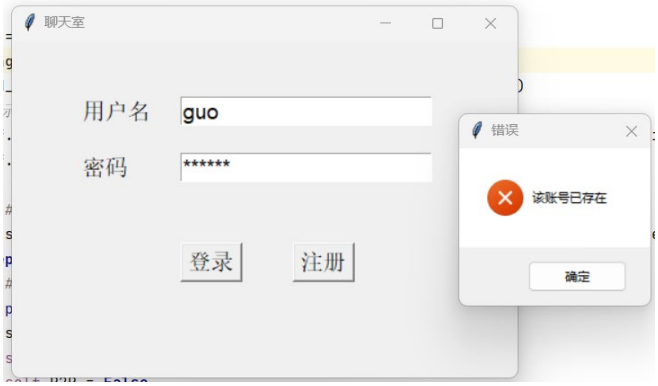
在 NAT 穿透的过程中，服务器的主要工作是转发通信双方之间的

协商消息（是否进行 NAT 穿透等消息），以及通过收到的 UDP 报文获取通信双方的公网 IP 地址与端口号并将其发送给通信双方，从而辅助实现 NAT 穿透。

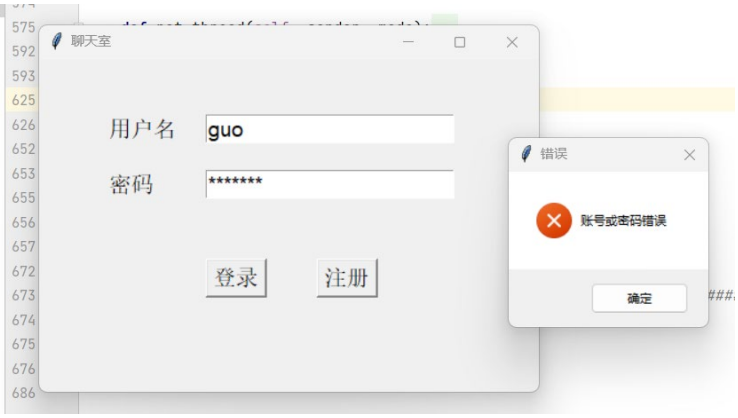
```
# NAT 穿透（全锥形）
def nat(self, message):
    # NAT 穿透请求
    if message.startswith("REQUEST"):
        parts = message.split("|", maxsplit=2)
        sender = parts[1]
        recipient = parts[2]
        self.send_message(self.client_sock[recipient],
f"NAT_REQUEST|{sender}")
    # 同意 NAT 穿透
    elif message.startswith("ACCEPT"):
        parts = message.split("|", maxsplit=2)
        sender = parts[1]
        recipient = parts[2]
        self.send_message(self.client_sock[recipient],
f"NAT_ACCEPT|{sender}")
        # 创建线程以获取客户端公网 ip 地址与端口号
        threading.Thread(target=self.nat_thread).start()
    # 拒绝 NAT 穿透
    elif message.startswith("REJECT"):
        parts = message.split("|", maxsplit=2)
        recipient = parts[2]
        self.send_message(self.client_sock[recipient], "NAT_REJECT")

def nat_thread(self):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(("172.18.152.17", 6666))
    for i in range(2):
        # 将客户端公网 ip 地址与端口号转发给对方
        message, address = sock.recvfrom(1024)
        message = message.decode("utf-8")
        parts = message.split("|", maxsplit=2)
        recipient = parts[1]
        self.send_message(self.client_sock[recipient],
f"NAT_ADDRESS|{address}")
    sock.close()
```

六、 测试及结果分析

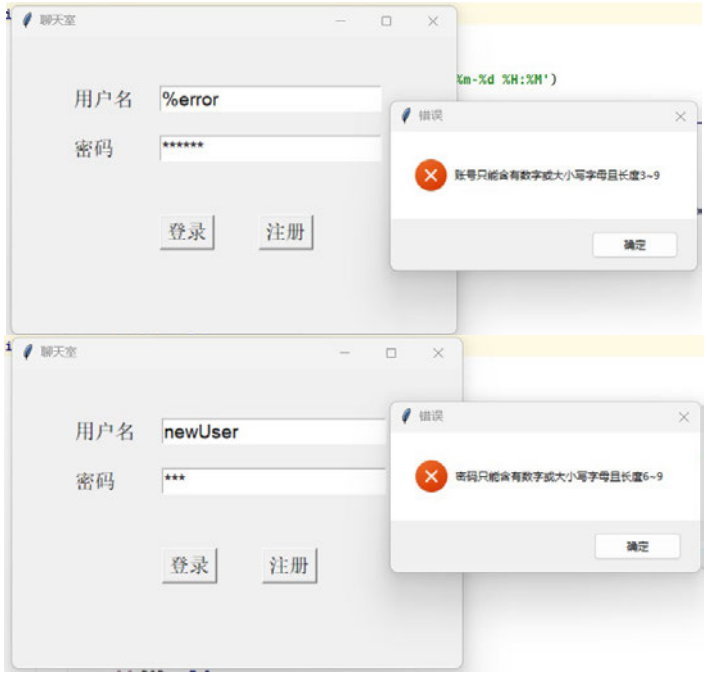
1. 测试 1-注册与登录



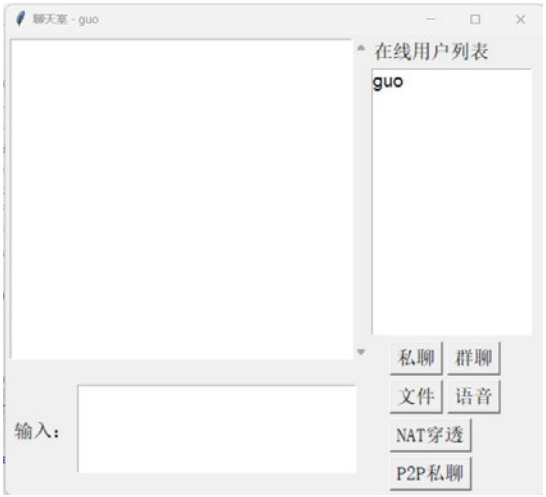
注册时，若用户名已存在则注册失败并提示错误信息。



登录时，若用户名不存在或者用户名与密码不匹配则登录失败并提示错误信息。



此外，对于用户名与密码的格式存在限制：账号只能含有数字或大小写字母且长度 3~9，密码只能含有数字或大小写字母且长度 6~9。

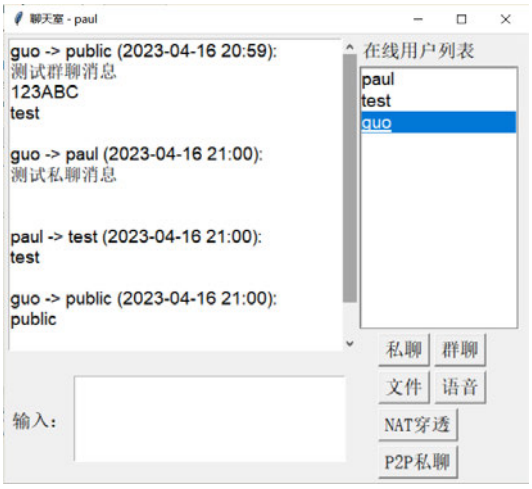


注册或登录成功后，便进入聊天界面。

2. 测试 2-文字聊天

客户端-guo

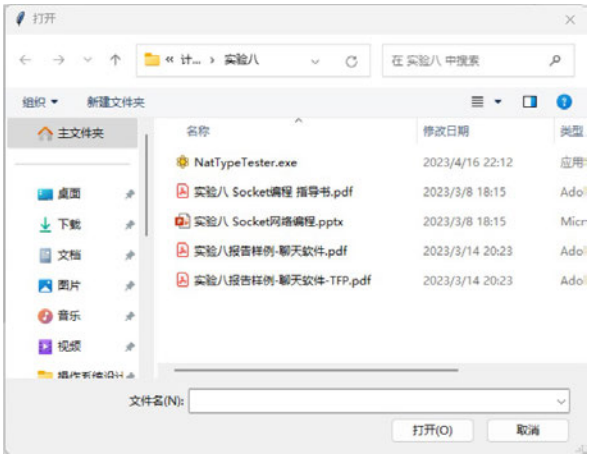
客户端-test

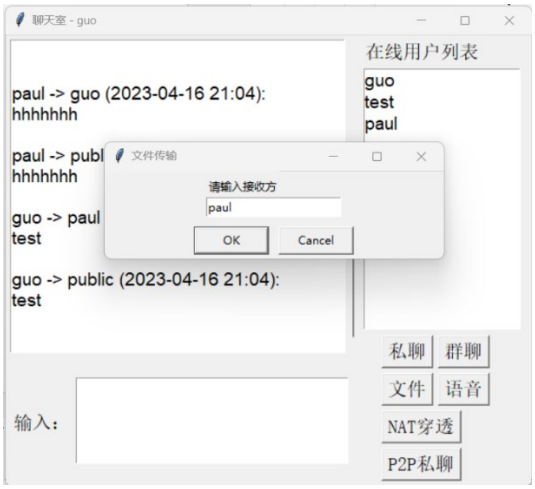


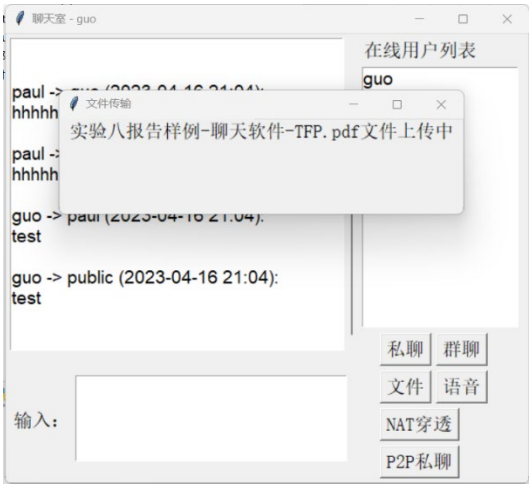
客户端-paul

文字聊天功能支持多用户同时在线、私聊、群聊、中英文、多行文字。从上图中可以发现，客户端 guo 发送给客户端 paul 的私聊消息只有二者可以看见而客户端 test 是看不见的，同时，客户端 guo 发送的群聊消息所有客户端都能看见。

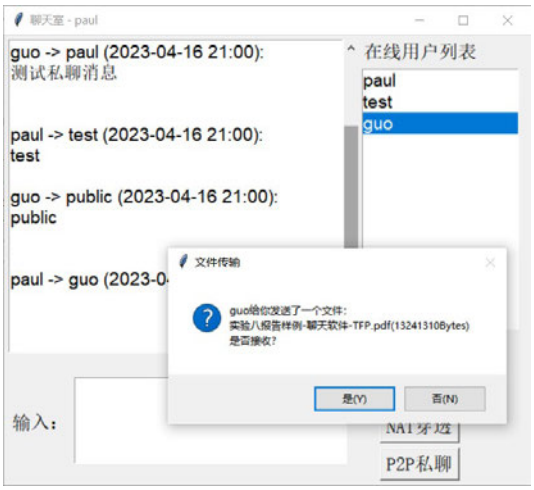
3. 测试 3-在线文件传输

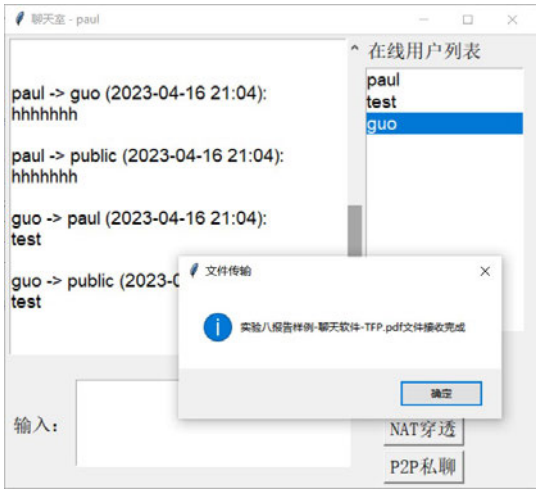


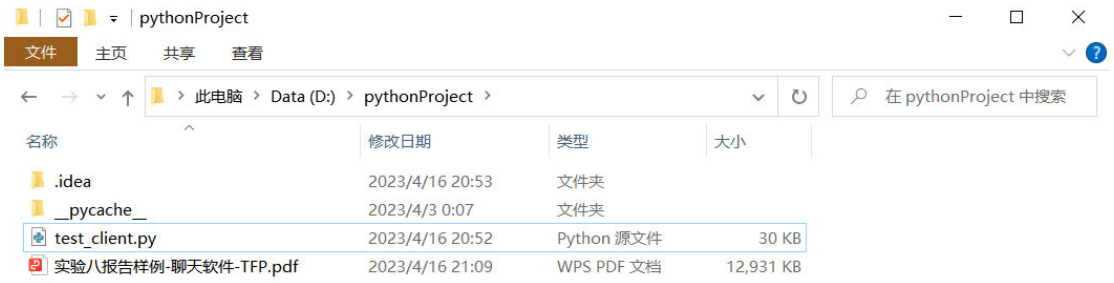




发送方



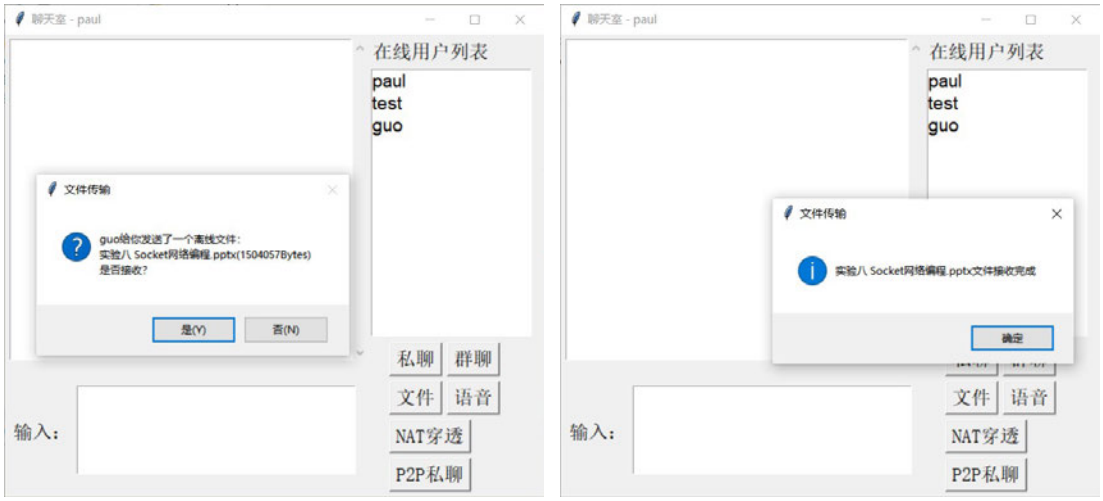


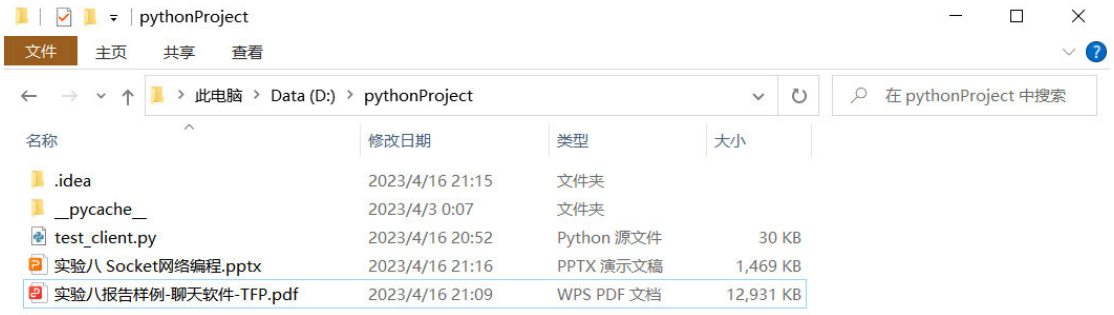


接收方

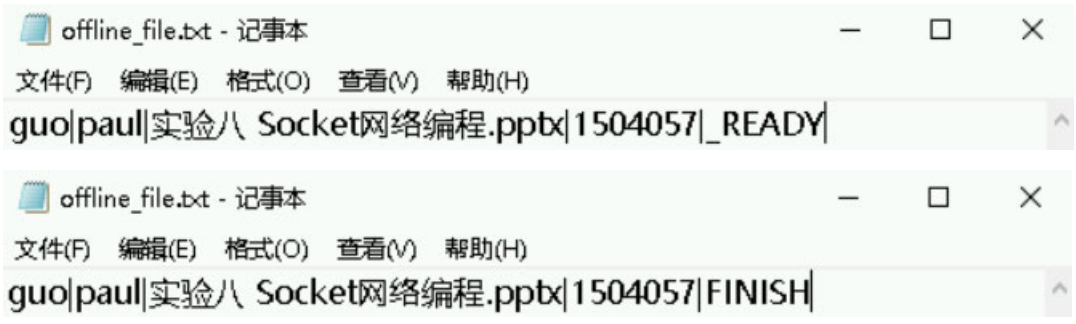
在测试 3 中，成功实现了 12MB 大小的在线文件传输。

4. 测试 4-离线文件传输





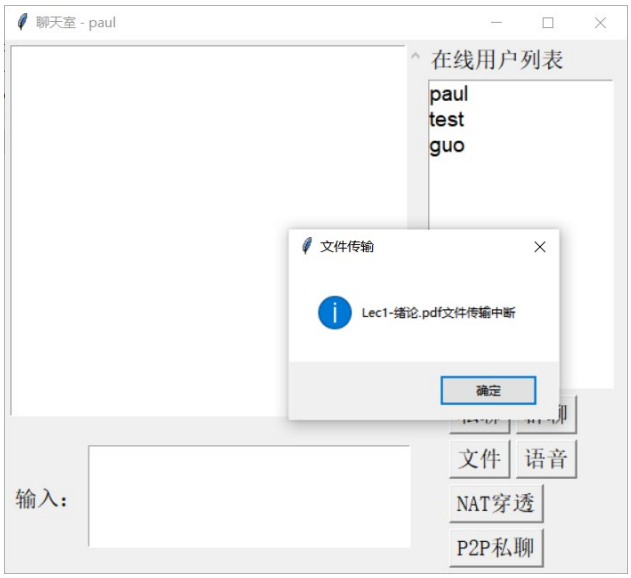
接收方

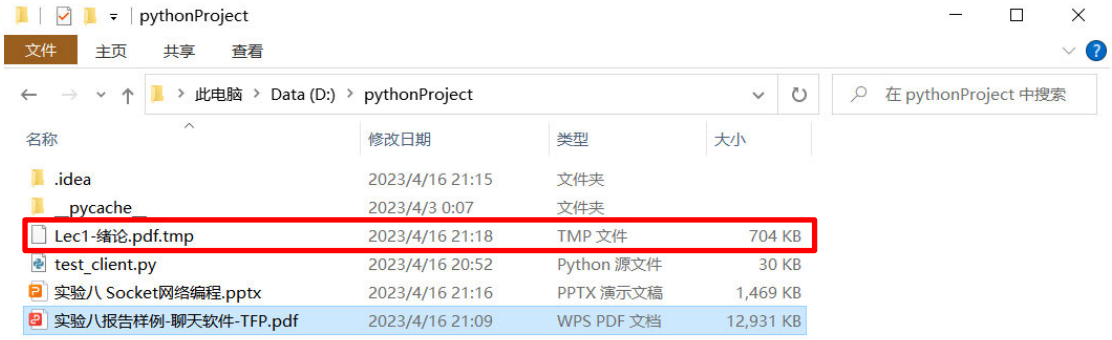


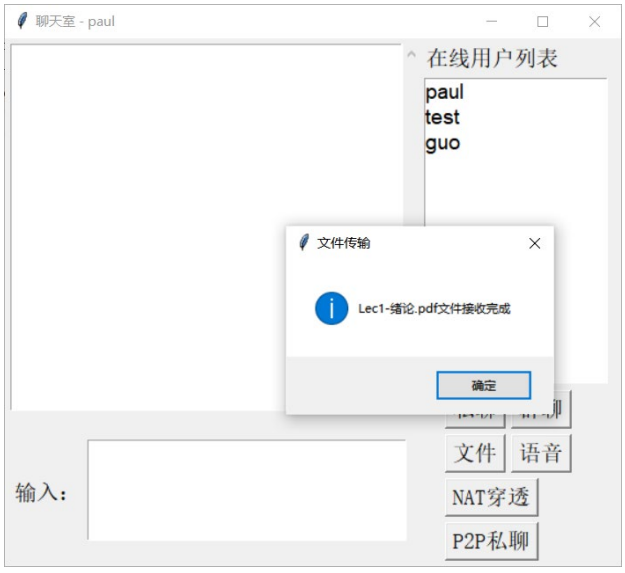
服务器离线文件记录

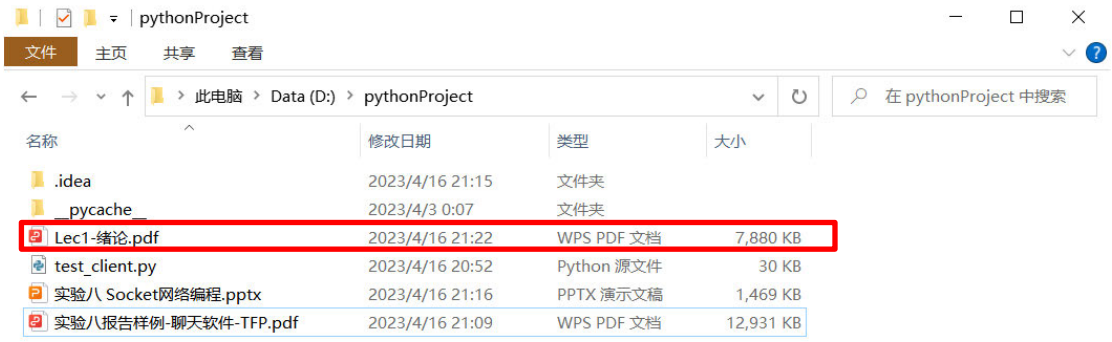
从上图中可以发现,客户端paul 在上线后成功接收了客户端guo发送的离线文件，且服务器的离线文件记录中，该离线文件的状态也由“_READY”变成了“FINISH”。

5. 测试5-断点续传





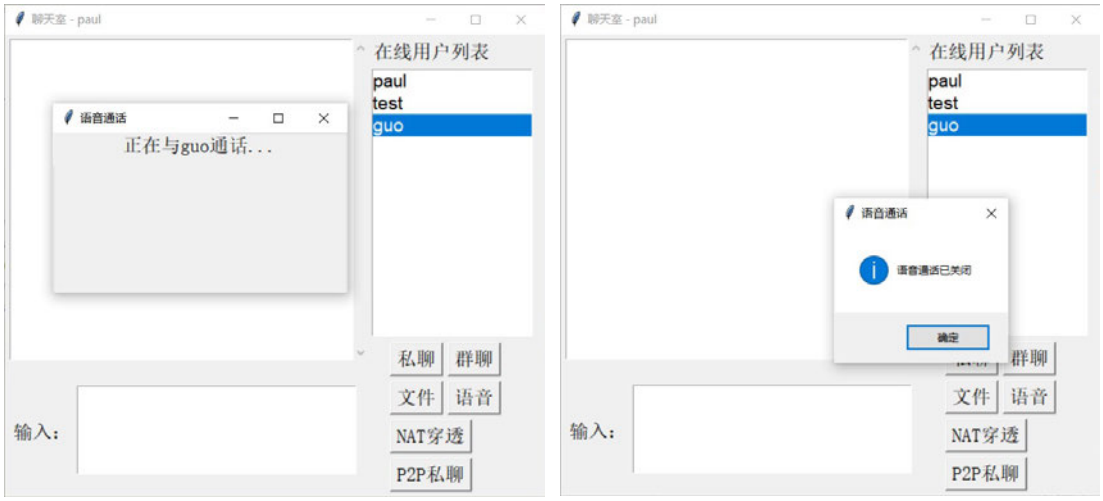




从上图中可以发现，当文件传输中断时，接收方接收到的文件大小为 704KB，且文件以“.tmp”结尾。当发送方重新发送文件时，将从先前中断的地方开始发送，最后接收方成功接收到了完整的文件。

6. 测试 6-语音通话

客户端-guo



客户端-paul

在测试 6 中，成功实现了低延时、清晰的语音通话。

7. 测试 7-NAT 穿透

本实验通过 UDP 打洞实现了全锥形 NAT 穿透，而由于组内成员的网络环境均为对称型 NAT，无法进行穿透。故该功能并未测试。

七、 实验结论

本次实验成功完成了全部基本功能，实现了客户端的注册与登录认证、多用户同时在线聊天（私聊与群聊）、用户间传输任意格式的文件。同时也完成全部高级功能，实现了全锥形 NAT 穿透、支持离线文件与断点续传、支持语音通话。

八、 总结及心得体会

通过本次实验，我们掌握了 socket 编程与多线程编程的基本方法，并成功实现了一个基于客户端/服务器模型的聊天程序。

在实验的过程中，我们也遇到了许多的问题，比如说：

①在实现客户端的收发消息时，刚开始我们将数据进行 UTF-8 编码后再传输，但在后来的文件传输及语音通话中却有一些比特流无法进行 UTF-8 编码，从而导致了数据传输的异常，最后的解决方法是将文件传输及语音通话的数据以比特流的方式直接发送；

②服务器与客户端在接收 TCP 报文时，如何区分不同的报文，解决方法是发送时在数据包的前面加上两个字节的数据长度字段，接收时先接收两个字节，在根据数据长度接收指定大小的报文；

总之，通过本次实验我们不仅掌握了网络编程的方法，而且更为重要的是，我们学会了团队配合，以及一个完整项目是如何从设计到实现的，这对于我们在将来的学习与工作都是极其有用的。

附件

1.参考资料

（1）如何使用 python 的 socket 包实现本地与云服务器通信

（2）python 实现实时语音对讲

（3）分别基于 UDP 和 TCP 打洞实现 P2P 连接-Python 实现

（4）Python 的 Socket 编程教程

2.源代码

（1）客户端

```
# 聊天程序客户端
from tkinter import messagebox
from tkinter import scrolledtext
from tkinter import filedialog
from tkinter import simpledialog
import tkinter as tk
import socket
import threading
import os
import time
import datetime
import pyaudio

###################################################################


class Client:
    def __init__(self, server_ip, server_port):
        self.username = ""
        self.is_connected = False
        self.close_window = False
        self.socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        try:
            # 建立连接
            self.socket.connect((server_ip, server_port))
        except socket.error as e:
            # 连接失败
            self.socket.close()
```

```
            tk.Tk().withdraw()
            messagebox.showerror('错误', str(e))
            raise SystemExit
        else:
            # 连接成功
            self.is_connected = True

    def send_message(self, message, data=b""):
        # 以二进制格式传输文件或语音
        if message.startswith("FILE_CONTENT") or
message.startswith("VOIC_CONTENT"):
            parts = message.split("|", maxsplit=2)
            name = parts[1].encode("utf-8")     # 文件名或语音发送方
            recipient = parts[2].encode("utf-8")
            sent_data = message[:12].encode("utf-8") +
bytes([len(name), len(recipient)]) + name + recipient + data
        # 传输普通字符串
        else:
            sent_data = message.encode('utf-8')
        # 计算数据包长度封装消息头
        length = len(sent_data)
        header = bytes([length // 256, length % 256])
        try:
            # 发送消息头和消息体
            self.socket.sendall(header + sent_data)
        except socket.error as e:
            # 连接断开
            if self.is_connected:
                self.is_connected = False
                self.socket.close()
                messagebox.showerror("错误", str(e))
                self.close_window = True
            raise SystemExit

    def receive_message(self):
        try:
            message = b""
            # 先接收两个字节的消息头，得到消息长度
            header = self.socket.recv(2)
            length = header[0] * 256 + header[1]
            # 根据消息长度接收消息体
            while length > len(message):
                message += self.socket.recv(length - len(message))
        except socket.error as e:
```

```
            # 连接断开
            if self.is_connected:
                self.is_connected = False
                self.socket.close()
                messagebox.showerror("错误", str(e))
                self.close_window = True
            raise SystemExit
        else:
            # 接收文件或语音
            if message[:12].decode('utf-8') == "FILE_CONTENT" or
message[:12].decode('utf-8') == "VOIC_CONTENT":
                name_len = message[12]
                name = message[13:13 + name_len].decode('utf-8')    #
文件名或语音发送方
                return f"{message[:12].decode('utf-8')}|{name}",
message[13 + name_len:]
            # 接收字符串
            else:
                return message.decode('utf-8'), b""

######################################################################


class LoginWindow:
    def __init__(self, _client):
        self.client = _client
        # 主窗口
        self.window = tk.Tk()
        self.window.geometry("450x300")
        self.window.title("聊天室")
        self.window.protocol("WM_DELETE_WINDOW", self.on_closing)
        self.window.after(1000, self.check_connect)
        # 用户名标签
        self.label_username = tk.Label(self.window, text="用户名",
font=20)
        self.label_username.place(x=60, y=50)
        # 用户名输入框
        self.entry_username = tk.Entry(self.window,
font=("Helvetica", 14))
        self.entry_username.place(x=150, y=50)
        # 密码标签
        self.label_password = tk.Label(self.window, text="密码",
font=20)
        self.label_password.place(x=60, y=100)
```

```
        # 密码输入框
        self.entry_password = tk.Entry(self.window,
font=("Helvetica", 14), show='*')
        self.entry_password.place(x=150, y=100)
        # 登录按钮
        self.button_login = tk.Button(self.window, text="登录",
font=25, command=self.login)
        self.button_login.place(x=150, y=180)
        # 注册按钮
        self.button_register = tk.Button(self.window, text="注册",
font=25, command=self.register)
        self.button_register.place(x=250, y=180)

        self.window.mainloop()

    def on_closing(self):
        self.client.socket.close()
        self.window.destroy()
        raise SystemExit

    def check_connect(self):
        if self.client.close_window:
            self.window.destroy()
            raise SystemExit
        else:
            # 每隔 1 秒检测连接是否断开
            self.window.after(1000, self.check_connect)

    def login(self):
        self.client.username = self.entry_username.get()
        # 判断用户名是否合法
        if self.client.username.isalnum() and 2 <
len(self.client.username) < 10:
            password = self.entry_password.get()
            # 判断密码是否合法
            if password.isalnum() and 5 < len(password) < 10:
                # 发送登录认证

self.client.send_message(f"LOGIN|{self.client.username}|{password}"
)
                # 等待服务器发送登录确认
                message, data = self.client.receive_message()
                if message.startswith("LOGIN_SUCCEED"):
                    self.window.destroy()
```

```
            elif message.startswith("LOGIN_FAIL"):
                messagebox.showerror('错误', "账号或密码错误")
            else:
                messagebox.showerror('错误', "密码只能含有数字或大小写字母
且长度 6~9")
        else:
            messagebox.showerror('错误', "账号只能含有数字或大小写字母且长
度 3~9")

    def register(self):
        self.client.username = self.entry_username.get()
        # 判断用户名是否合法
        if self.client.username.isalnum() and 2 <
len(self.client.username) < 10:
            password = self.entry_password.get()
            # 判断密码是否合法
            if password.isalnum() and 5 < len(password) < 10:
                # 发送注册请求

self.client.send_message(f"REGISTER|{self.client.username}|{passwor
d}")
                # 等待服务器发送注册确认
                message, data = self.client.receive_message()
                if message.startswith("REGISTER_SUCCEED"):
                    self.window.destroy()
                elif message.startswith("REGISTER_FAIL"):
                    messagebox.showerror('错误', "该账号已存在")
            else:
                messagebox.showerror('错误', "密码只能含有数字或大小写字母
且长度 6~9")
        else:
            messagebox.showerror('错误', "账号只能含有数字或大小写字母且长
度 3~9")

######################################################################


class ChatWindow:
    def __init__(self, _client):
        self.client = _client
        self.file_state = 0
        self.file_sent_size = 0
        self.voice_state = 0
        self.MyPeer = None
```

```
        self.P2P = False
        # 主窗口
        self.window = tk.Tk()
        self.window.geometry("560x480")
        self.window.title(f"聊天室 - {self.client.username}")
        self.window.protocol("WM_DELETE_WINDOW", self.on_closing)
        self.window.after(1000, self.check_connect)
        # 消息显示界面
        self.textbox = scrolledtext.ScrolledText(self.window,
font=("Helvetica", 14), width=32, height=15)
        self.textbox.place(x=5, y=5)
        self.textbox.bind("<Key>", self.disable_keyboard)
        # 在线用户列表选择框
        self.user_list = tk.Listbox(self.window, font=("Helvetica",
14), selectmode="single", width=15, height=12)
        self.user_list.place(x=380, y=35)
        # 消息输入框
        self.input = tk.Text(self.window, font=("Helvetica", 14),
width=26, height=4)
        self.input.place(x=75, y=365)
        # 在线用户列表标签
        self.label1 = tk.Label(self.window, text="在线用户列表",
font=10)
        self.label1.place(x=380, y=5)
        # 输入标签
        self.label2 = tk.Label(self.window, text="输入：", font=10)
        self.label2.place(x=6, y=400)
        # 私聊按钮
        self.button1 = tk.Button(self.window, text="私聊", font=10,
command=self.send_private_message)
        self.button1.place(x=400, y=320)
        # 群聊按钮
        self.button2 = tk.Button(self.window, text="群聊", font=10,
command=self.send_group_message)
        self.button2.place(x=460, y=320)
        # 发送文件按钮
        self.button3 = tk.Button(self.window, text="文件", font=10,
command=self.send_file)
        self.button3.place(x=400, y=360)
        # 语音通话按钮
        self.button4 = tk.Button(self.window, text="语音", font=10,
command=self.send_voice)
        self.button4.place(x=460, y=360)
        # NAT 穿透按钮
```

```
        self.button5 = tk.Button(self.window, text="NAT 穿透",
font=10, command=self.nat_request)
        self.button5.place(x=400, y=400)
        # P2P 私聊按钮
        self.button6 = tk.Button(self.window, text="P2P 私聊",
font=10, command=self.P2P_chat)
        self.button6.place(x=400, y=440)

        # 语音通话参数
        chunk_size = 1024                  # 设置每个音频缓冲区的大小为 1024
帧
        audio_format = pyaudio.paInt16      # 设置音频格式为 16 位整型
        channels = 1                       # 设置音频通道数为 1（单声道）
        rate = 20000                       # 设置音频采样率为 20000Hz
        # 播放音频的音频流对象
        self.playing_stream =
pyaudio.PyAudio().open(format=audio_format, channels=channels,
rate=rate, output=True,

frames_per_buffer=chunk_size)
        # 录制音频的音频流对象
        self.recording_stream =
pyaudio.PyAudio().open(format=audio_format, channels=channels,
rate=rate, input=True,

frames_per_buffer=chunk_size)

        # 创建线程用来接收服务器发来的消息
        self.receive_thread =
threading.Thread(target=self.receive_messages)
        self.receive_thread.setDaemon(True)
        self.receive_thread.start()

        self.window.mainloop()

    def on_closing(self):
        self.client.socket.close()
        self.window.destroy()
        raise SystemExit

    def check_connect(self):
        if self.client.close_window:
            self.window.destroy()
            raise SystemExit
```

```
        else:
            # 每隔 1 秒检测连接是否断开
            self.window.after(1000, self.check_connect)

    # 消息显示界面禁止键盘输入
    @staticmethod
    def disable_keyboard(event):
        return "break"

    # 发送私聊消息
    def send_private_message(self):
        # 获取选中的私聊用户
        recipient = self.user_list.get("anchor")
        if recipient:
            message = self.input.get("1.0", tk.END)
            if message:
                send_time =
datetime.datetime.now().strftime('%Y-%m-%d %H:%M')
                # 显示在聊天界面
                self.textbox.insert(tk.END, f"{self.client.username}
-> {recipient} ({send_time}):\n{message}\n")
                self.textbox.see(tk.END)
                # 向服务器发送消息

self.client.send_message(f"PRIVATE|{self.client.username}|{recipien
t}|{send_time}|{message}")
            else:
                messagebox.showerror("错误", "发送的消息不能为空")
        else:
            messagebox.showerror("错误", "未选择私聊对象")

    # 发送群聊消息
    def send_group_message(self):
        message = self.input.get("1.0", tk.END)
        if message:
            send_time =
datetime.datetime.now().strftime('%Y-%m-%d %H:%M')
            # 显示在聊天界面
            self.textbox.insert(tk.END, f"{self.client.username} ->
public ({send_time}):\n{message}\n")
            self.textbox.see(tk.END)
            # 向服务器发送消息

self.client.send_message(f"GROUP|{self.client.username}|{send_time}
```

```
|{message}")
        else:
            messagebox.showerror("错误", "发送的消息不能为空")

    # 发送文件
    def send_file(self):
        # 选择文件
        file_path = filedialog.askopenfilename()
        if file_path:
            file_name = os.path.basename(file_path)
            file_size = os.path.getsize(file_path)
            # 选择接收方
            recipient = simpledialog.askstring("文件传输", "请输入接收方
")
            if recipient:

self.client.send_message(f"FILE_HEADER|{self.client.username}|{reci
pient}|{file_name}|{file_size}")
                threading.Thread(target=self.send_file_thread,
args=(recipient, file_path, file_size, )).start()

    def send_file_thread(self, recipient, file_path, file_size):
        # 等待应答
        while not self.file_state:
            continue
        # 接收方不存在
        if self.file_state == 1:
            messagebox.showerror("错误", "文件接收方不存在")
            self.file_state = 0
        # 对方拒绝接收
        elif self.file_state == 2:
            messagebox.showerror("错误", "对方拒绝接收文件")
            self.file_state = 0
        # 开始发送
        elif self.file_state == 3 or self.file_state == 4:
            file_name = os.path.basename(file_path)
            target = recipient
            if self.file_state == 4:
                recipient = "SERVER"    # 离线文件的接收方为服务器
            # UI 界面
            file_window = tk.Toplevel(self.window)
            file_window.title("文件传输")
            file_window.protocol("WM_DELETE_WINDOW",
self.file_cancel)
```

```
            tk.Label(file_window, text=f"{file_name}文件上传中",
font=10).pack()
            # 控制信息
            message = f"FILE_CONTENT|{file_name}|{recipient}"
            with open(file_path, "rb") as f:
                # 定位到先前已发送的位置
                f.seek(self.file_sent_size)
                total_sent = self.file_sent_size
                # 发送文件数据
                while self.file_state != 0 and total_sent < file_size:
                    data = f.read(2**15)
                    total_sent += len(data)
                    self.client.send_message(message, data)
            file_window.destroy()
            self.file_sent_size = 0
            if self.file_state != 0:
                # 发送完成

self.client.send_message(f"FILE_END|{self.client.username}|{recipie
nt}|{file_name}|{file_size}|{target}")
                messagebox.showinfo("文件传输", f"{file_name}文件发送完成
")
                self.file_state = 0
            else:
                # 发送取消

self.client.send_message(f"FILE_CANCEL|{recipient}|{file_name}")

    # 取消文件发送
    def file_cancel(self):
        self.file_state = 0

    # 发送语音
    def send_voice(self):
        # 选择接收方
        recipient = self.user_list.get("anchor")
        if recipient:

self.client.send_message(f"VOIC_HEADER|{self.client.username}|{reci
pient}")
            # 创建发送线程
            threading.Thread(target=self.send_voice_thread,
args=(recipient, )).start()
        else:
```

```
            messagebox.showerror("错误", "未选择接听方")

    def send_voice_thread(self, recipient):
        # 等待应答
        while not self.voice_state:
            continue
        # 对方拒绝接听
        if self.voice_state == 1:
            messagebox.showerror("错误", "对方拒绝接听")
            self.voice_state = 0
        # 开始发送
        elif self.voice_state == 2:
            # UI 界面
            voice_window = tk.Toplevel(self.window)
            voice_window.title("语音通话")
            tk.Label(voice_window, text=f"正在与{recipient}通话...",
font=10).pack()
            voice_window.protocol("WM_DELETE_WINDOW",
self.close_voice)
            # 控制信息
            message =
f"VOIC_CONTENT|{self.client.username}|{recipient}"
            # 发送语音数据
            while self.voice_state == 2:
                data = self.recording_stream.read(1024)
                self.client.send_message(message, data)
            # 语音通话结束
            voice_window.destroy()

self.client.send_message(f"VOIC_END|{self.client.username}|{recipie
nt}")

    # 结束语音通话
    def close_voice(self):
        self.voice_state = 0

    # 更新在线用户列表
    def update_online_users(self, message):
        user = message[4:]
        # 用户上线
        if message.startswith("ADD"):
            self.user_list.insert(tk.END, user)
        # 用户离线
        elif message.startswith("DEL"):
```

```
            for i in range(self.user_list.size()):
                if self.user_list.get(i) == user:
                    self.user_list.delete(i)
                    break

    # 接收私聊消息
    def recv_private_message(self, message):
        parts = message.split("|", maxsplit=3)
        sender = parts[0]
        recipient = parts[1]
        send_time = parts[2]
        msg = parts[3]
        if recipient == self.client.username:
            self.textbox.insert(tk.END, f"{sender} -> {recipient}
({send_time}):\n{msg}\n")
            self.textbox.see(tk.END)

    # 接收群聊消息
    def recv_group_message(self, message):
        parts = message.split("|", maxsplit=2)
        sender = parts[0]
        send_time = parts[1]
        msg = parts[2]
        self.textbox.insert(tk.END, f"{sender} -> public
({send_time}):\n{msg}\n")
        self.textbox.see(tk.END)

    # 接收文件
    def recv_file(self, message, data):
        # 询问是否接收在线文件
        if message.startswith("HEADER"):
            parts = message.split("|", maxsplit=3)
            sender = parts[1]
            file_name = parts[2]
            file_size = int(parts[3])
            threading.Thread(target=self.recv_file_thread,
args=(sender, file_name, file_size, 1)).start()
        # 询问是否接收离线文件
        elif message.startswith("OFFLINE_HEADER"):
            parts = message.split("|", maxsplit=3)
            sender = parts[1]
            file_name = parts[2]
            file_size = int(parts[3])
            threading.Thread(target=self.recv_file_thread,
```

```
args=(sender, file_name, file_size, 2)).start()
        # 接收文件
        elif message.startswith("CONTENT"):
            parts = message.split("|", maxsplit=1)
            file_name = parts[1]
            with open(file_name+".tmp", "ab") as f:
                f.write(data)
        # 接收完成
        elif message.startswith("END"):
            parts = message.split("|", maxsplit=1)
            file_name = parts[1]
            os.rename(file_name + ".tmp", file_name)
            threading.Thread(target=self.recv_file_thread, args=("",
file_name, "", 3)).start()
        # 传输中断
        elif message.startswith("CANCEL"):
            parts = message.split("|", maxsplit=1)
            file_name = parts[1]
            threading.Thread(target=self.recv_file_thread, args=("",
file_name, "", 4)).start()
        # 接收方不存在
        elif message.startswith("USER_NO_EXIST"):
            self.file_state = 1
        # 对方拒绝接收
        elif message.startswith("REJECT"):
            self.file_state = 2
        # 通知发送线程开始发送在线文件
        elif message.startswith("ACCEPT"):
            self.file_sent_size = int(message[7:])
            self.file_state = 3
        # 通知发送线程开始发送离线文件
        elif message.startswith("OFFLINE_USER"):
            self.file_sent_size = int(message[13:])
            self.file_state = 4


    def recv_file_thread(self, sender, file_name, file_size, mode):
        # 询问是否接收在线文件
        if mode == 1:
            if messagebox.askyesno("文件传输", f"{sender}给你发送了一个文
件：\n{file_name}({file_size}Bytes)\n是否接收？"):
                file_sent_size = 0
                # 获取先前已接收的文件大小-断点续传
                if os.path.isfile(file_name+".tmp"):
                    file_sent_size = os.path.getsize(file_name+".tmp")
```

```
self.client.send_message(f"FILE_ACCEPT|{self.client.username}|{send
er}|{file_sent_size}")
            else:

self.client.send_message(f"FILE_REJECT|{self.client.username}|{send
er}")
        # 询问是否接收离线文件
        elif mode == 2:
            if messagebox.askyesno("文件传输", f"{sender}给你发送了一个离
线文件：\n{file_name}({file_size}Bytes)\n是否接收？"):
                file_sent_size = 0
                # 获取先前已接收的文件大小-断点续传
                if os.path.isfile(file_name+".tmp"):
                    file_sent_size = os.path.getsize(file_name+".tmp")

self.client.send_message(f"FILE_ACCEPT|{self.client.username}|SERVE
R|{file_sent_size}")
            else:

self.client.send_message(f"FILE_REJECT|{self.client.username}|SERVE
R")
        # 文件接收完成
        elif mode == 3:
            messagebox.showinfo("文件传输", f"{file_name}文件接收完成")
        # 文件传输中断
        elif mode == 4:
            messagebox.showinfo("文件传输", f"{file_name}文件传输中断")

    # 接收语音
    def recv_voice(self, message, data):
        # 询问是否进行语音通话
        if message.startswith("HEADER"):
            parts = message.split("|", maxsplit=1)
            sender = parts[1]
            threading.Thread(target=self.recv_voice_thread,
args=(sender, 1)).start()
        # 接听语音
        elif message.startswith("CONTENT"):
            self.playing_stream.write(data)
        # 对方关闭语音通话
        elif message.startswith("END"):
            parts = message.split("|", maxsplit=1)
            sender = parts[1]
```

```
                self.voice_state = 0
                threading.Thread(target=self.recv_voice_thread,
args=(sender, 2)).start()
        # 对方拒绝接听
        elif message.startswith("REJECT"):
            self.voice_state = 1
        # 通知发送线程开始发送语音
        elif message.startswith("ACCEPT"):
            self.voice_state = 2

    def recv_voice_thread(self, sender, mode):
        # 询问是否同意语音通话
        if mode == 1:
            if messagebox.askyesno("语音通话", f"{sender}邀请你进行语音通
话\n是否接受？"):

self.client.send_message(f"VOIC_ACCEPT|{self.client.username}|{send
er}")
                # 开启语音发送线程
                threading.Thread(target=self.send_voice_thread,
args=(sender,)).start()
                self.voice_state = 2
            else:

self.client.send_message(f"VOIC_REJECT|{self.client.username}|{send
er}")
        # 语音通话关闭
        elif mode == 2:
            messagebox.showinfo("语音通话", "语音通话已关闭")

    # NAT 穿透（全锥形）
    def nat_request(self):
        # 选择连接方
        recipient = self.user_list.get("anchor")
        if recipient:

self.client.send_message(f"NAT_REQUEST|{self.client.username}|{reci
pient}")
        else:
            messagebox.showerror("错误", "未选择连接方")

    def nat_handle(self, message):
        # 询问是否同意 NAT 穿透
        if message.startswith("REQUEST"):
```

```
            parts = message.split("|", maxsplit=1)
            sender = parts[1]
            threading.Thread(target=self.nat_thread, args=(sender,
1)).start()
        # 同意 NAT 穿透
        elif message.startswith("ACCEPT"):
            parts = message.split("|", maxsplit=1)
            sender = parts[1]
            sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            # 向服务器发送消息以通知对方自己的公网 ip 地址与端口
            sock.sendto(f"NAT_SERVER|{sender}".encode("utf-8"),
("8.130.66.246", 6666))
            # 开启 P2P 通信线程
            threading.Thread(target=self.nat_send, args=(sock,
sender)).start()
            threading.Thread(target=self.nat_recv,
args=(sock,)).start()
        # 拒绝 NAT 穿透
        elif message.startswith("REJECT"):
            threading.Thread(target=self.nat_thread, args=("",
2)).start()
        # 获取对方公网 ip 地址与端口
        elif message.startswith("ADDRESS"):
            parts = message.split("|", maxsplit=1)
            self.MyPeer = eval(parts[1])

    def nat_thread(self, sender, mode):
        # 询问是否同意 NAT 穿透
        if mode == 1:
            if messagebox.askyesno("NAT 穿透", f"{sender}请求建立连接\n
是否接受？"):

self.client.send_message(f"NAT_ACCEPT|{self.client.username}|{sende
r}")
                sock = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
                time.sleep(1)
                # 向服务器发送消息以通知对方自己的公网 ip 地址与端口
                sock.sendto(f"NAT_SERVER|{sender}".encode("utf-8"),
("8.130.66.246", 6666))
                # 开启 P2P 通信线程
                threading.Thread(target=self.nat_send, args=(sock,
sender)).start()
                threading.Thread(target=self.nat_recv,
```

```
args=(sock,)).start()
            else:

self.client.send_message(f"NAT_REJECT|{self.client.username}|{sende
r}")
        # 对方拒绝 NAT 穿透
        elif mode == 2:
            messagebox.showerror("错误", "对方拒绝建立连接")

    def nat_send(self, sock, recipient):
        # 等待服务器通知对方的公网 ip 地址与端口
        while not self.MyPeer:
            continue
        print(f"对方公网 ip 地址与端口号：{self.MyPeer}")
        # 尝试 UDP 打洞
        for i in range(5):
            sock.sendto("NAT_HELLO".encode("utf-8"), self.MyPeer)
            time.sleep(1)
        self.P2P = False
        # 发送 P2P 私聊消息
        while True:
            if self.P2P:
                message = self.input.get("1.0", tk.END)
                if message:
                    send_time =
datetime.datetime.now().strftime('%Y-%m-%d %H:%M')
                    # 显示在聊天界面
                    self.textbox.insert(tk.END,
f"{self.client.username} -> {recipient}
({send_time}):\n{message}\n")
                    self.textbox.see(tk.END)
                    try:
                        # 发送消息

sock.sendto(f"{self.client.username}|{recipient}|{send_time}|{messa
ge}".encode("utf-8"), self.MyPeer)
                    except socket.error as e:
                        # 连接出错
                        print(e)
                        sock.close()
                        self.MyPeer = None
                        self.P2P = False
                        raise SystemExit
                else:
```

```
                messagebox.showerror("错误", "发送的消息不能为空")
            self.P2P = False

    def nat_recv(self, sock):
        while True:
            try:
                # 接收消息
                message = sock.recvfrom(1024)[0].decode("utf-8")
            except socket.error as e:
                # 连接出错
                print(e)
                sock.close()
                self.MyPeer = None
                self.P2P = False
                raise SystemExit
            else:
                if message.startswith("NAT_HELLO"):
                    # UDP 打洞成功
                    print("connect!")
                    continue
                # 接收 P2P 私聊消息
                parts = message.split("|", maxsplit=3)
                sender = parts[0]
                recipient = parts[1]
                send_time = parts[2]
                msg = parts[3]
                if recipient == self.client.username:
                    self.textbox.insert(tk.END, f"(P2P){sender} ->
{recipient} ({send_time}):\n{msg}\n")
                    self.textbox.see(tk.END)

    def P2P_chat(self):
        self.P2P = True

    # 接收来自服务器的消息
    def receive_messages(self):
        while True:
            message, data = self.client.receive_message()
            if message.startswith("UPDATE_USERS"):
                self.update_online_users(message[13:])
            elif message.startswith("PRIVATE"):
                self.recv_private_message(message[8:])
            elif message.startswith("GROUP"):
                self.recv_group_message(message[6:])
```

```
            elif message.startswith("FILE"):
                self.recv_file(message[5:], data)
            elif message.startswith("VOIC"):
                self.recv_voice(message[5:], data)
            elif message.startswith("NAT"):
                self.nat_handle(message[4:])

########################################################################
##########


if __name__ == '__main__':
    Server_ip = "8.130.66.246"     # 服务器公网 ip 地址
    Server_port = 8888

    # 创建聊天室客户端
    client = Client(Server_ip, Server_port)
    # 打开登录窗口
    LoginWindow(client)
    # 打开聊天窗口
    ChatWindow(client)
```

（2）服务器

```
# 聊天程序服务器
import socket
import threading
import os
import datetime


class Server:
    def __init__(self, server_ip, server_port):
        self.client_sock = {}
        self.file_state = 0
        self.file_sent_size = 0
        self.user_file = "users.txt"
        self.socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        # 创建存储用户信息的文件
        if not os.path.isfile(self.user_file):
            with open(self.user_file, "w") as f:
                f.write("SERVER\n")
        # 将套接字绑定到指定的地址和端口
        self.socket.bind((server_ip, server_port))
        # 设置监听状态
```

```
        self.socket.listen(10)
        print(f"Server listening on {server_ip}: {server_port} ...")

    def start(self):
        while True:
            # 接受客户端连接请求
            client_socket, client_address = self.socket.accept()
            # 为每个客户端创建一个线程来处理请求
            threading.Thread(target=self.handle_client,
args=(client_socket, )).start()

    def send_message(self, client_socket, message, data=b""):
        # 以二进制格式传输文件或语音
        if message.startswith("FILE_CONTENT") or
message.startswith("VOIC_CONTENT"):
            parts = message.split("|", maxsplit=1)
            name = parts[1].encode("utf-8")    # 文件名或语音发送方
            sent_data = message[:12].encode("utf-8") +
bytes([len(name)]) + name + data
        # 传输普通字符串
        else:
            sent_data = message.encode('utf-8')
        # 计算数据包长度封装消息头
        length = len(sent_data)
        header = bytes([length // 256, length % 256])
        try:
            # 发送消息头和消息体
            client_socket.sendall(header + sent_data)
        except socket.error:
            # 连接断开
            self.offline(client_socket)

    def receive_message(self, client_socket):
        try:
            message = b""
            # 先接收两个字节的消息头，得到消息长度
            header = client_socket.recv(2)
            try:
                length = header[0] * 256 + header[1]
            except IndexError:
                raise SystemExit
            # 根据消息长度接收消息体
            while length > len(message):
                message += client_socket.recv(length - len(message))
```

```
            except socket.error:
                # 连接断开
                self.offline(client_socket)
            else:
                # 接收文件或语音
                if message[:12].decode('utf-8') == "FILE_CONTENT" or
message[:12].decode('utf-8') == "VOIC_CONTENT":
                    name_len = message[12]
                    recipient_len = message[13]
                    name = message[14:14 + name_len].decode('utf-8')      #
文件名或语音发送方
                    recipient = message[14 + name_len: 14 + name_len +
recipient_len].decode('utf-8')
                    return f"{message[:12].decode('utf-
8')}||{name}||{recipient}", message[14 + name_len + recipient_len:]
                # 接收字符串
                else:
                    return message.decode('utf-8'), b""

        # 用户离线
        def offline(self, client_socket):
            # 寻找套接字对应的用户名
            for username, sock in self.client_sock.items():
                if client_socket == sock:
                    # 删除套接字
                    del self.client_sock[username]
                    # 服务器日志记录用户离线
                    print(f"{username} leaves at
{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
                    # 向在线用户发送更新消息
                    self.update_online_users(username, "DEL")
                    break
            # 退出线程
            raise SystemExit

        # 更新在线用户列表
        def update_online_users(self, user, Op):
            # 向已在线用户广播更新消息
            for username, sock in self.client_sock.items():
                self.send_message(sock, f"UPDATE_USERS|{Op}||{user}")
            # 向新上线用户发送在线用户列表
            if Op == "ADD":
                for username, sock in self.client_sock.items():
                    if user != username:
```

```
                    self.send_message(self.client_sock[user],
f"UPDATE_USERS|{Op}|{username}")

    # 登录认证
    def login(self, client_socket, message):
        parts = message.split("|", maxsplit=1)
        username = parts[0]
        password = parts[1]
        with open(self.user_file, "r") as f:
            f.seek(0)
            lines = f.readlines()
            found = False
            # 查询用户是否存在
            for line in lines:
                parts = line.strip().split()
                if len(parts) == 2 and parts[0] == username:
                    found = True
                    # 判断密码是否正确
                    if parts[1] == password:
                        # 记录用户套接字
                        self.client_sock[username] = client_socket
                        # 服务器日志记录用户登录
                        print(f"{username} login at
{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
                        # 向客户端发送登录成功消息
                        self.send_message(client_socket,
f"LOGIN_SUCCEED")
                        # 更新在线用户列表
                        self.update_online_users(username, "ADD")
                        # 查询是否有离线文件需要发送给该客户端

threading.Thread(target=self.offline_file).start()
                    else:
                        # 账号与密码不匹配
                        self.send_message(client_socket, f"LOGIN_FAIL")
                    break
            if not found:
                # 用户不存在
                self.send_message(client_socket, f"LOGIN_FAIL")

    # 注册认证
    def register(self, client_socket, message):
        parts = message.split("|", maxsplit=1)
        username = parts[0]
```

```
        password = parts[1]
        with open(self.user_file, "a+") as f:
            f.seek(0)
            lines = f.readlines()
            found = False
            # 查询用户是否存在
            for line in lines:
                parts = line.strip().split()
                if parts[0] == username:
                    # 账号名已存在
                    found = True
                    self.send_message(client_socket, f"REGISTER_FAIL")
                    break
            # 用户不存在则创建新用户
            if not found:
                # 服务器记录新账号与密码
                f.write(f"{username} {password}\n")
                # 记录用户套接字
                self.client_sock[username] = client_socket
                # 服务器日志记录用户注册
                print(f"{username} register at
{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
                # 向客户端发送注册成功消息
                self.send_message(client_socket, f"REGISTER_SUCCEED")
                # 更新在线用户列表
                self.update_online_users(username, "ADD")

    # 发送私聊消息
    def private_message(self, client_socket, message):
        parts = message.split("|", maxsplit=2)
        sender = parts[0]
        recipient = parts[1]
        message = parts[2]
        if recipient in self.client_sock:
            recipient_socket = self.client_sock[recipient]
            if recipient_socket != client_socket:
                self.send_message(recipient_socket,
f"PRIVATE|{sender}|{recipient}|{message}")

    # 发送群聊消息
    def public_message(self, client_socket, message):
        parts = message.split("|", maxsplit=1)
        sender = parts[0]
        message = parts[1]
```

```
        for username, sock in self.client_sock.items():
            if sock != client_socket:
                self.send_message(sock, f"GROUP|{sender}|{message}")

    # 发送文件
    def send_file(self, client_socket, message, data):
        # 处理发送文件请求
        if message.startswith("HEADER"):
            parts = message.split("|", maxsplit=4)
            sender = parts[1]
            recipient = parts[2]
            file_name = parts[3]
            file_size = parts[4]
            # 查看接收方是否存在
            with open(self.user_file, "r") as f:
                f.seek(0)
                lines = f.readlines()
                found = False
                for line in lines:
                    parts = line.strip().split()
                    if parts[0] == recipient:
                        found = True
                        if recipient in self.client_sock:
                            # 接收方在线

self.send_message(self.client_sock[recipient],
f"FILE_HEADER|{sender}|{file_name}|{file_size}")
                        else:
                            # 接收方离线
                            file_sent_size = 0
                            # 查看服务器是否存在发送未完成的临时文件
                            if os.path.isfile(file_name+".tmp"):
                                file_sent_size =
os.path.getsize(file_name+".tmp")
                            # 通知客户端发送离线文件
                            self.send_message(client_socket,
f"FILE_OFFLINE_USER|{file_sent_size}")
                        break
                if not found:
                    # 接收方不存在
                    self.send_message(client_socket,
"FILE_USER_NO_EXIST")
        # 发送文件内容
        elif message.startswith("CONTENT"):
```

```
            parts = message.split("|", maxsplit=2)
            file_name = parts[1]
            recipient = parts[2]
            if recipient == "SERVER":
                # 服务器本地存储离线文件
                with open(file_name + ".tmp", "ab") as f:
                    f.write(data)
            else:
                # 向接收方转发在线文件
                self.send_message(self.client_sock[recipient],
f"FILE_CONTENT|{file_name}", data)
        # 文件发送完成
        elif message.startswith("END"):
            parts = message.split("|", maxsplit=5)
            sender = parts[1]
            recipient = parts[2]
            file_name = parts[3]
            file_size = parts[4]
            target = parts[5]
            if recipient == "SERVER":
                # 服务器日志记录离线文件任务
                os.rename(file_name + ".tmp", file_name)
                with open("offline_file.txt", "ab") as f:

f.write(f"{sender}|{target}|{file_name}|{file_size}|_READY\n".encod
e("utf-8"))
                threading.Thread(target=self.offline_file).start()
            else:
                # 通知接收方文件发送完成
                self.send_message(self.client_sock[recipient],
f"FILE_END|{file_name}")
        # 文件发送取消
        elif message.startswith("CANCEL"):
            parts = message.split("|", maxsplit=2)
            recipient = parts[1]
            file_name = parts[2]
            if recipient != "SERVER":
                self.send_message(self.client_sock[recipient],
f"FILE_CANCEL|{file_name}")
        # 对方拒绝接收文件
        elif message.startswith("REJECT"):
            parts = message.split("|", maxsplit=2)
            recipient = parts[2]
            if recipient == "SERVER":
```

```
                self.file_state = 1
            else:
                self.send_message(self.client_sock[recipient],
f"FILE_REJECT")
        # 对方同意接收文件
        elif message.startswith("ACCEPT"):
            parts = message.split("|", maxsplit=3)
            recipient = parts[2]
            file_sent_size = parts[3]
            if recipient == "SERVER":
                self.file_sent_size = int(file_sent_size)
                self.file_state = 2
            else:
                self.send_message(self.client_sock[recipient],
f"FILE_ACCEPT|{file_sent_size}")

    # 处理离线文件
    def offline_file(self):
        if os.path.isfile("offline_file.txt"):
            # 查询日志
            with open("offline_file.txt", "rb+") as f:
                f.seek(0)
                line = f.readline().decode("utf-8").rstrip()
                while line:
                    if line.endswith("FINISH"):
                        # 离线文件传输已完成
                        line = f.readline().decode("utf-8").rstrip()
                        continue
                    parts = line.split("|")
                    sender = parts[0]
                    recipient = parts[1]
                    file_name = parts[2]
                    file_size = parts[3]
                    if recipient not in self.client_sock:
                        # 接收方离线
                        line = f.readline().decode("utf-8").rstrip()
                        continue
                    # 接收方上线后，询问是否接收文件
                    self.send_message(self.client_sock[recipient],
f"FILE_OFFLINE_HEADER|{sender}|{file_name}|{file_size}")
                    # 等待应答
                    while not self.file_state:
                        continue
                    # 发送离线文件
```

```
                if self.file_state == 2:
                    message = f"FILE_CONTENT|{file_name}"
                    with open(file_name, "rb") as f1:
                        # 定位到上次文件传输中断的位置
                        f1.seek(self.file_sent_size)
                        total_sent = self.file_sent_size
                        while total_sent < int(file_size):
                            data = f1.read(2 ** 15)
                            total_sent += len(data)

self.send_message(self.client_sock[recipient], message, data)
                    self.file_sent_size = 0
                    self.send_message(self.client_sock[recipient],
f"FILE_END|{file_name}")
                # 服务器本地删除文件
                os.remove(file_name)
                # 服务器日志记录离线传输完成
                f.seek(-7, 1)
                f.write("FINISH\n".encode("utf-8"))
                self.file_state = 0
                # 读取下一条记录
                line = f.readline().decode("utf-8").rstrip()

    # 发送语音
    def send_voice(self, message, data):
        # 处理发送语音请求
        if message.startswith("HEADER"):
            parts = message.split("|", maxsplit=2)
            sender = parts[1]
            recipient = parts[2]
            self.send_message(self.client_sock[recipient],
f"VOIC_HEADER|{sender}")
        # 发送语音内容
        elif message.startswith("CONTENT"):
            parts = message.split("|", maxsplit=2)
            sender = parts[1]
            recipient = parts[2]
            self.send_message(self.client_sock[recipient],
f"VOIC_CONTENT|{sender}", data)
        # 关闭语音通话
        elif message.startswith("END"):
            parts = message.split("|", maxsplit=2)
            sender = parts[1]
            recipient = parts[2]
```

```
            self.send_message(self.client_sock[recipient],
f"VOIC_END|{sender}")
        # 对方拒绝接收语音
        elif message.startswith("REJECT"):
            parts = message.split("|", maxsplit=2)
            recipient = parts[2]
            self.send_message(self.client_sock[recipient],
f"VOIC_REJECT")
        # 对方同意接收语音
        elif message.startswith("ACCEPT"):
            parts = message.split("|", maxsplit=2)
            recipient = parts[2]
            self.send_message(self.client_sock[recipient],
f"VOIC_ACCEPT")

    # NAT 穿透（全锥形）
    def nat(self, message):
        # NAT 穿透请求
        if message.startswith("REQUEST"):
            parts = message.split("|", maxsplit=2)
            sender = parts[1]
            recipient = parts[2]
            self.send_message(self.client_sock[recipient],
f"NAT_REQUEST|{sender}")
        # 同意 NAT 穿透
        elif message.startswith("ACCEPT"):
            parts = message.split("|", maxsplit=2)
            sender = parts[1]
            recipient = parts[2]
            self.send_message(self.client_sock[recipient],
f"NAT_ACCEPT|{sender}")
            # 创建线程以获取客户端公网 ip 地址与端口号
            threading.Thread(target=self.nat_thread).start()
        # 拒绝 NAT 穿透
        elif message.startswith("REJECT"):
            parts = message.split("|", maxsplit=2)
            recipient = parts[2]
            self.send_message(self.client_sock[recipient],
"NAT_REJECT")

    def nat_thread(self):
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        sock.bind(("172.18.152.17", 6666))
```

```
        for i in range(2):
            # 将客户端公网 ip 地址与端口号转发给对方
            message, address = sock.recvfrom(1024)
            message = message.decode("utf-8")
            parts = message.split("|", maxsplit=2)
            recipient = parts[1]
            self.send_message(self.client_sock[recipient],
f"NAT_ADDRESS|{address}")
        sock.close()

    # 接收客户端消息
    def handle_client(self, client_socket):
        while True:
            message, data = self.receive_message(client_socket)
            if message.startswith("LOGIN"):
                self.login(client_socket, message[6:])
            elif message.startswith("REGISTER"):
                self.register(client_socket, message[9:])
            elif message.startswith("PRIVATE"):
                self.private_message(client_socket, message[8:])
            elif message.startswith("GROUP"):
                self.public_message(client_socket, message[6:])
            elif message.startswith("FILE"):
                self.send_file(client_socket, message[5:], data)
            elif message.startswith("VOIC"):
                self.send_voice(message[5:], data)
            elif message.startswith("NAT"):
                self.nat(message[4:])


if __name__ == "__main__":
    Server_ip = "172.18.152.17"     # 服务器私网 ip 地址
    Server_port = 8888

    server = Server(Server_ip, Server_port)
    server.start()
```