

计算机通信网络大作业报告

项目四：聊天程序

学号: 517021911065 姓名: 刘浩文 班级: F1703603

联系方式: IssacLiewX@sjtu.edu.cn

2019 年 12 月 25 日

摘要

在这次大作业实践中，我使用 *python* 语言，利用 *Socket* 编程完成了一个功能较为齐全的聊天程序。该聊天程序分为服务器与客户端，服务器与客户端建立 *TCP* 连接。其界面与操作逻辑参考 *macOS* 版的聊天软件 *QQ*。其功能有：支持多个用户之间的一对一聊天（私聊）；支持群组聊天（默认模式），即多对多聊天；显示在线联系人列表，当有新用户上线时，其他用户实时更新在线联系人，用户下线时同理；支持文件传输，实现用户之间的文件传输，不限文件类型（通过服务器中的共享文件夹完成）；能够刷新连接，在服务器宕机重启或网络断开后不用重启客户端就能重新与服务器建立连接；支持用户名顶替登陆，如果用户名相同，后来的用户会把之前的同名用户顶替掉；支持收发图片表情；支持全屏截图。

关键词： 聊天程序 Socket 编程 TCP

1 概述

以下列出本次实践的基本调试环境：

- **机器:** *MacBook Pro (13 - inch, 2016, Four Thunderbolt 3 Ports)*
- **系统:** *macOS Catalina Version10.15.2*
- **运行环境:** *python3.7.0*
- **编译工具:** *Spyder4.0.0*
- **测试时服务器端系统:** *macOS Catalina Version10.15.2*
- **测试时客户端系统:** *macOS Catalina Version10.15.2* 与虚拟机软件 *VMware Fusion Pro Version 11.5.1 (15018442)* 中的虚拟机系统 *Ubuntu 18.04.3 LTS* 64 位, 网络连接为 *NAT* 模式
- **程序文件列表:**

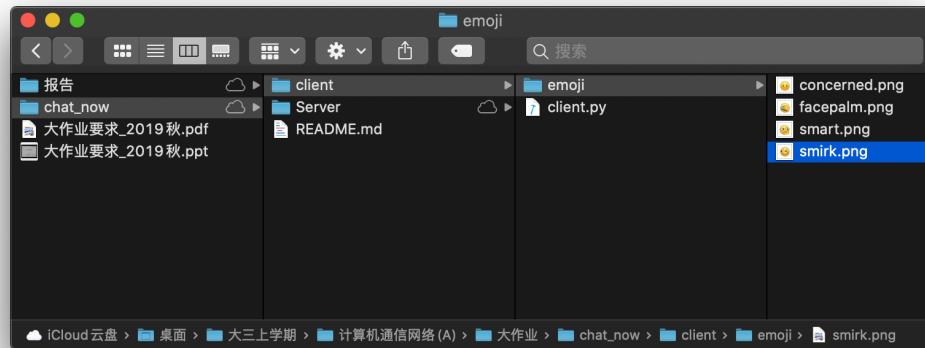


图 1: *client* 文件列表

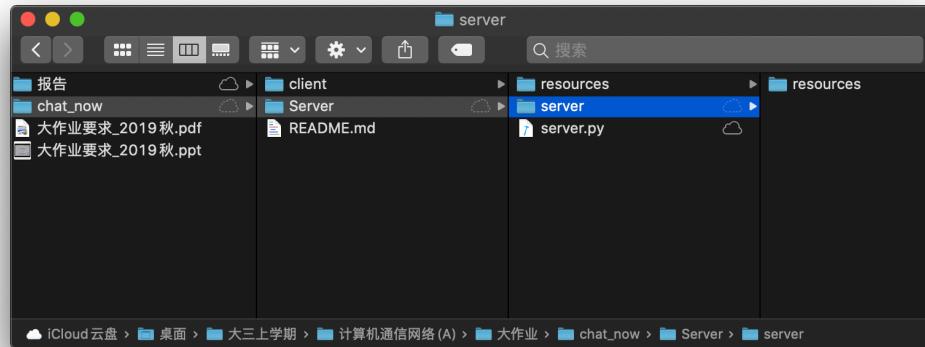


图 2: *Server* 文件列表

2 主要数据结构与主要算法

2.1 主要数据结构

- **Socket:** $IP : PORT$, IP 是用来标识互联网中的一台主机的位置, 而 $PORT$ 是用来标识这台机器上的一个应用程序, IP 与 $PORT$ 的绑定就标识了互联网中独一无二的一个应用程序
- **队列:** 缓冲器使用队列来读取、传输、输出各种消息, 或者使用信息如结束符 EOF 等进行判断与下一步执行。
- **字典:** 用户在服务器中以 $[connection, username, address(IP : PORT)]$ 形式存储, 便于添加与删除用户, 建立与断开对应用户的连接。

2.2 主要算法

2.2.1 基本框架: *Socket* 编程实现 *TCP* 连接

在了解 *Socket* 编程时, 在网上查找了大量资料, 对 *Socket* 编程有了一定了解。

可以对 *Socket* 编程下一个这样的定义: *Socket* 是应用层与 *TCP/IP* 协议族通信的中间软件抽象层, 它是一组接口。在设计模式中, *Socket* 其实就是一个门面模式, 它把复杂的 *TCP/IP* 协议族隐藏在 *Socket* 接口后面, 对用户来说, 一组简单的接口就是全部, 让 *Socket* 去组织数据, 以符合指定的协议。所以, 我们无需深入理解 *TCP/UDP* 协议, *Socket* 已经为我们封装好了, 我们只需要遵循 *Socket* 的规定去编程, 写出的程序自然就是遵循 *TCP/UDP* 标准的 [1]。

可以总结出使用 *Socket* 编程实现 *TCP* 连接对方法及流程。在 *Socket* 编程实现 *TCP* 连接时, 连接发起方为客户端, 连接响应方为服务器端。**其中客户端的实现流程为:**

1. 创建客户端的 *socket* 对象;
2. 建立与服务器之间的联系;
3. 发送请求;
4. 接收数据;
5. 关闭连接。

服务器端的实现流程为:

1. 创建服务器端的 *socket* 对象;
2. 绑定服务器端的地址;
3. 设置监听;
4. 等待客户端的连接;
5. 接受客户端的请求;
6. 返回处理的结果到客户端。

客户端与服务器端的流程及联系如图 3所示。

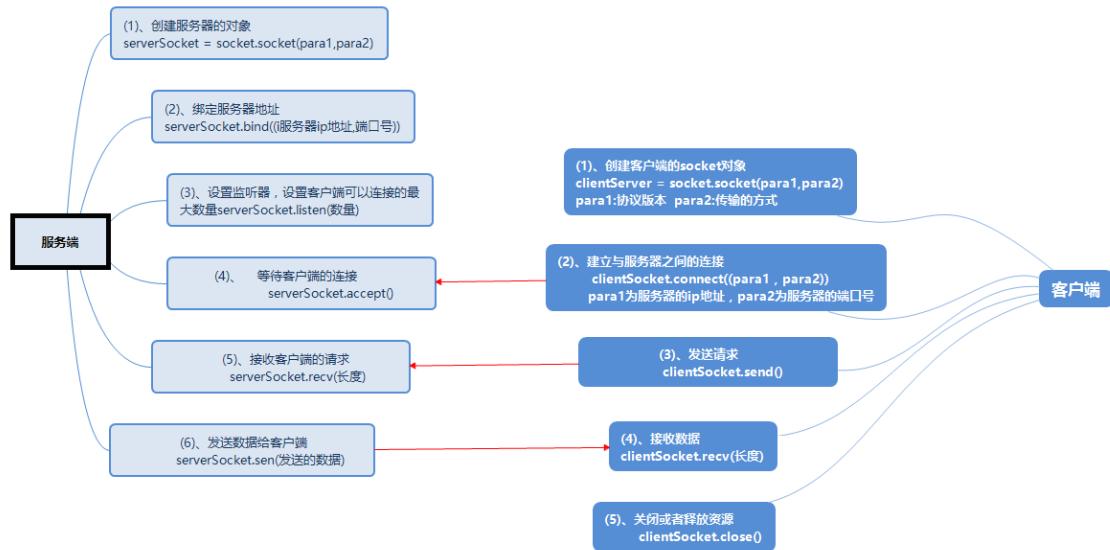


图 3: 客戶端与服務器端的流程及联系 [2]

在了解以上的知识后就可以搭建聊天程序的基本框架了。

```

1 #File:server.py
2 import socket
3
4 ip_port=('127.0.0.1',50007) #socket
5 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #创建服务器端的$socket$对象;
6 s.setsockopt(SOL_SOCKET,SO_REUSEADDR,1) #重用 ip 和端口
7 s.bind(ip_port) #绑定服务器端的地址
8 s.listen(5) #设置监听;
9
10 while True:
11     conn,addr=s.accept() #等待客户端的连接;
12     while True:
13         msg=conn.recv(1024) #接受客户端的请求;
14
15         conn.send(msg.upper()) #返回处理的结果到客户端。
16
17     conn.close() #关闭到该客户端的连接
18
19 s.close() #关闭服务器
  
```

```

1 #File:client.py
2 import socket
3
4 ip_port=('127.0.0.1',50007) #socket
5 s=socket.socket(socket.AF_INET,socket.SOCK_STREAM) #创建客户端的$socket$对象;
6
7 s.connect(ip_port) #建立与服务器之间的联系;
8
9 while True:
10     msg=input('>>>').strip()
11     if len(msg) == 0:continue
12
  
```

```
13 s.send(msg.encode('utf-8')) #发送请求;
14 feedback=s.recv(1024)    #接收数据;
15
16 s.close()   #关闭连接。
```

完成基本框架就可以继续进行聊天程序的下一步设计了。

2.2.2 服务器设计

服务器分为 *ChatServer* 和 *FileServer*。其中 *ChatServer* 负责记录连接的用户，转发用户发送的消息；而 *FileServer* 负责客户端与服务器之间文件的相互传送。*ChatServer* 与 *FileServer* 相互独立，*ChatServer* 需要用户名才能登陆，*FileServer* 不需要用户名就可以登陆。但在用户端，文件传输是聊天程序的一个附属功能，也就是说如果用户不先登陆 *ChatServer* 是无法连接 *FileServer* 的，但用户断开与 *ChatServer* 的连接后却仍然可以与服务器进行文件传输。

2.2.3 用户登陆及用户列表刷新设计

为了能够清楚连接到服务器端的用户的各属性，即连接 ($\text{socket}(IP : PORT)$)、用户名、及用户地址，采用字典的数据结构来存储用户信息： $\text{users}[\text{connection}, \text{username}, \text{address}(IP : PORT)]$ ，便于添加与删除用户，建立与断开对应用户的连接。

在客户端用户建立和服务器的连接后，将同时发送用户的用户名。服务器端记录连接、用户名、用户地址并一起存入用户字典中。在用户断开连接后，服务器将根据断开的连接在字典中找到该连接对应的条目并一并删除。之后在服务器的每一轮消息转发中，都将最先发送用户列表，以此达到客户端实时更新用户列表的功能。

2.2.4 聊天程序支持群聊与私聊设计

本聊天程序默认聊天模式为群聊，这是因为在服务器设计时由于时间不足只设计了群发的转发方式，而未设计指定对象转发的方式。在转发消息时，将用户字典遍历一遍，向所有用户发送相同的消息，以此实现群聊。私聊模式与群聊模式的切换是在客户端实现的。用户端在向服务器发送消息时，每条消息的开头带有模式选择与用户名，服务器将消息转发到所有客户端。客户端收到消息时，先检查每条消息队列的开头，若发现是群聊模式的消息则无条件显示；若发现是私聊模式的消息则检查之后的指向用户名，若是自己则显示，若不是自己则不显示，由此实现私聊。可见即使是开启了私聊模式，所有客户端仍会收到所有消息，只是显示与否的区别，所以这种设计不太安全。这也是后续改进的方向，即私聊模式下消息发送应该在服务器端选择性发送。

2.2.5 文件传输设计

FileServer 的设计是基于 *CSDN* 上的一个已实现的命令行文件传输工具实现的 [3]。服务器基于 *TCP* 连接传输文件，根据客户端传来的指令完成文件的收发。

2.2.6 同名用户顶替与刷新连接设计

服务器端建立一个新的连接后先判断该连接的用户名有没有在用户字典中存在。若存在则将原来存在于用户字典中的同名用户的连接断开并删除条目，更新以新的条目。这也为刷新连接创造了条件。刷新连接即断开当前连接后尝试重连服务器，不改变用户名；由于同名用户顶替的规则，原来的用户连接被服务器断开，用户条目被服务器删除，更新以新的连接，由此完成连接刷新。

3 程序测试截图及说明

3.1 开启服务器

运行 `server.py`, 如图 4。注意至少让 `resources` 文件夹与 `server.py` 在同一个工作目录下。



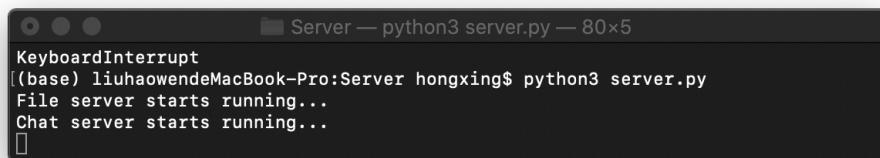
```
(base) liuhawendeMacBook-Pro:chat_now hongxing$ cd Server/
(base) liuhawendeMacBook-Pro:Server hongxing$ ls
resources      server.py
(base) liuhawendeMacBook-Pro:Server hongxing$ python3 server.py
```

图 4: 运行 `server.py`

点击弹出的 `Server` 窗口的 `Set Server` 按键, 然后关闭窗口, 启动服务器, 如图 5、6。注意一定要关闭 `Server` 窗口才能启动服务器, 因为 `Tkinter` 在 Mac 下工作不太流畅, 所以采用这个方法让窗口能够顺利关闭。



图 5: 开启服务器



```
KeyboardInterrupt
(base) liuhawendeMacBook-Pro:Server hongxing$ python3 server.py
File server starts running...
Chat server starts running...
```

图 6: 服务器开启

3.2 开启客户端

运行 `client.py`, 注意要让 `emoji` 文件夹与 `client.py` 在同一个工作目录下。弹出登陆窗口 `Log in`, 需要在 `Server address` 中输入服务器的 IP 与端口, 默认为 `127.0.0.1 : 50007`。然后输入 `Username`, 重名也没关

系，会把更早的同名用户顶替，模拟 QQ 的异地登陆下线。然后点击 *Log in* 按键。如图 7 所示。

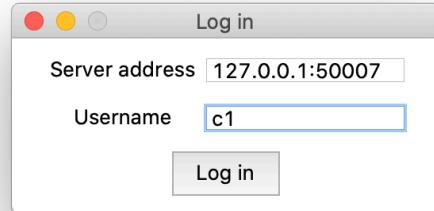


图 7: 用户登陆

登陆成功后弹出用户聊天程序窗口，如图 9 所示。依此步骤登陆多个用户以测试功能。测试时其他几个客户端是在 *Ubuntu* 虚拟机中创建的，如图 8。



图 8: 聊天程序用户窗口 (1)

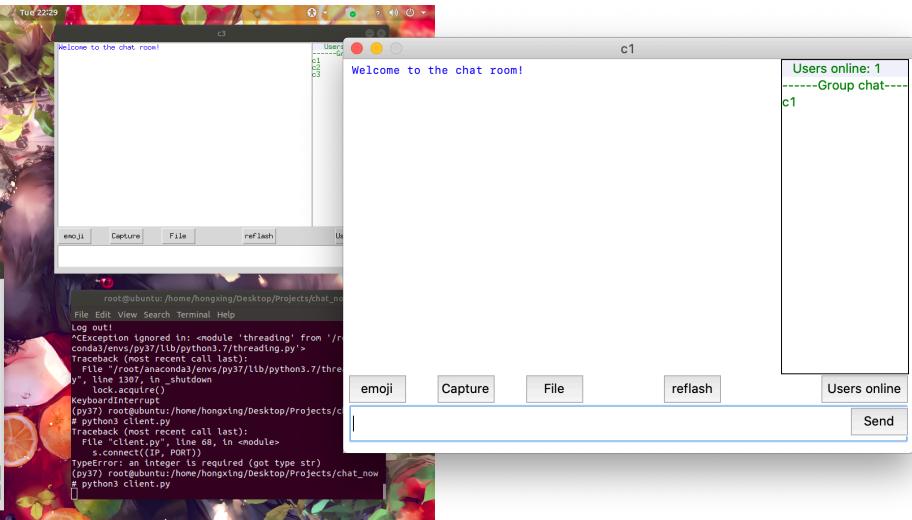


图 9: 聊天程序用户窗口 (2)

从这里也能看出该聊天程序可以正确显示并刷新用户列表。

3.3 用户同名顶替登陆

这个功能属于用户端登陆逻辑的一部分，在同名用户登录时会把之前登陆的同名用户顶替掉，即之前的同名用户被迫断开连接。当然被顶替掉的用户通过 *reflash* 刷新后也能重新建立连接，但又会把之前顶替了它的同名用户给顶替掉。这个逻辑是模仿了 QQ 的同账户异地登陆逻辑。由于其变化不容易通过静态的截图反应，可以观看演示录屏来了解这一过程。

3.4 群组聊天（聊天室）

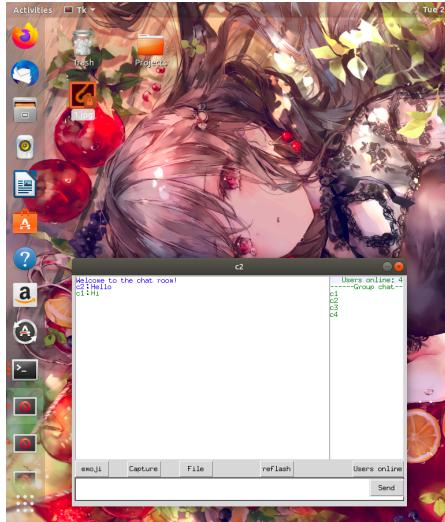


图 10: 群聊发送与接收消息 (1)

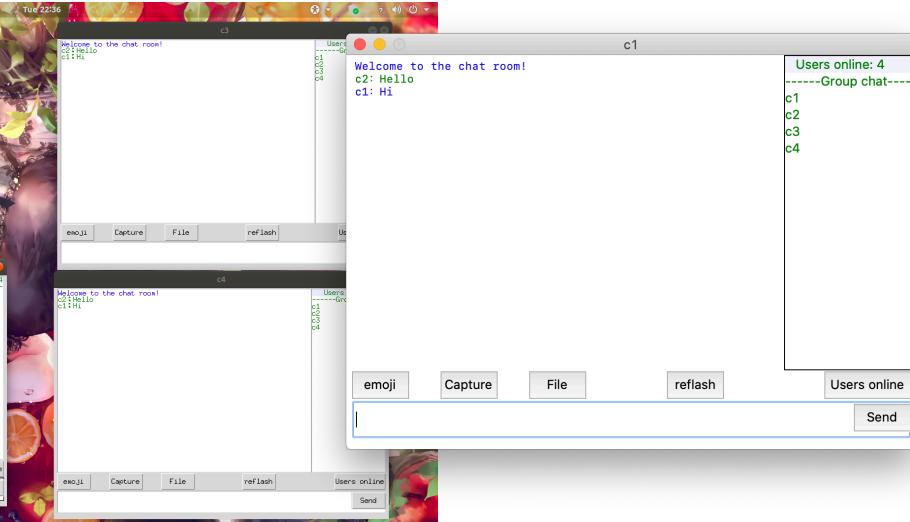


图 11: 群聊发送与接收消息 (2)

聊天程序客户端默认模式为群聊，或点击用户列表的 $--Group\ chat--$ 进入群聊模式。在最下面的输入框中输入一条消息，按回车或点击 *Send* 按钮后可以向所有在线用户发送消息，如图 11、10 所示。

3.5 多用户一对一聊天（私聊）

在聊天程序客户端中点击用户列表中的某位用户（不能是自己）进入私聊模式，发送方发给接收方的信息只有双方能看到，在线的其他用户都看不到，如图 12、13 所示（*c2* 私聊 *c1*，红色字体说明是私聊信息）。

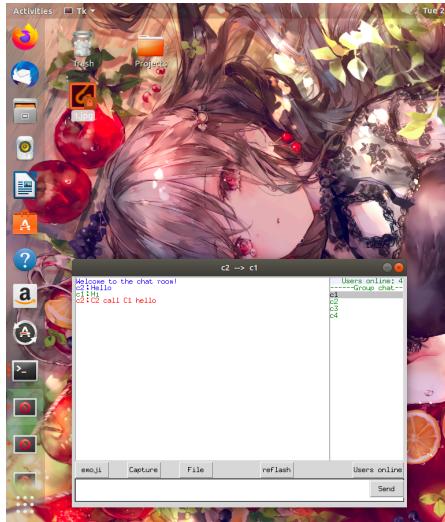


图 12: *c2* 私聊 *c1*，其他用户看不到

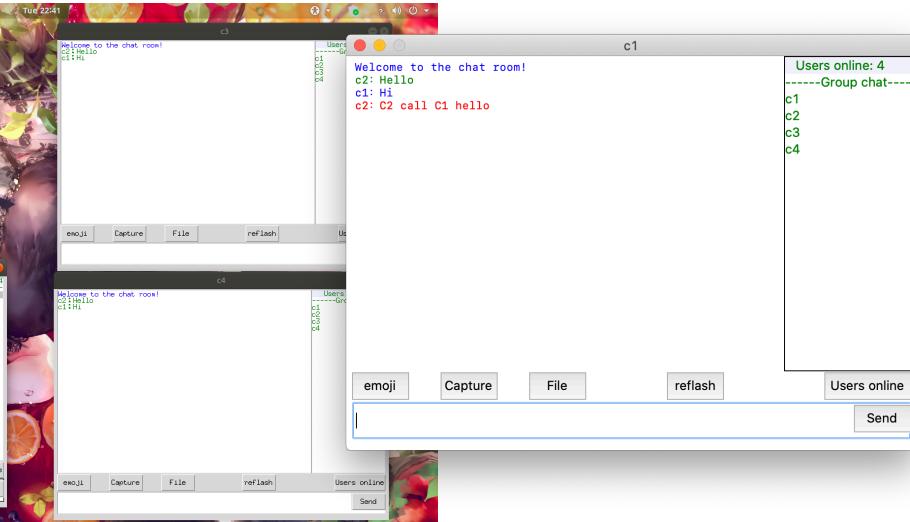


图 13: *c1* 接收 *c2* 的私聊信息

如果私聊模式时私聊自己则会显示警告“不能私聊自己”并拒绝请求，如图 14。

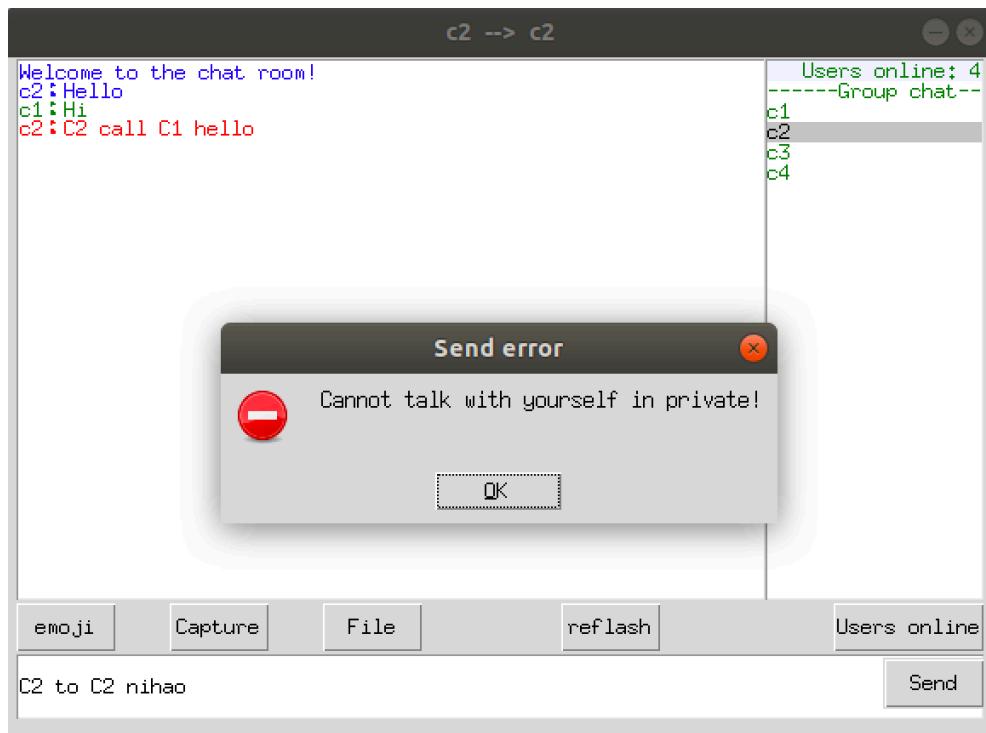


图 14: 私聊自己时发出警告

3.6 发送表情

点击 *emoji* 按钮选择表情发送。群聊模式中发送表情，所有在线用户均能收到。如图 15所示。

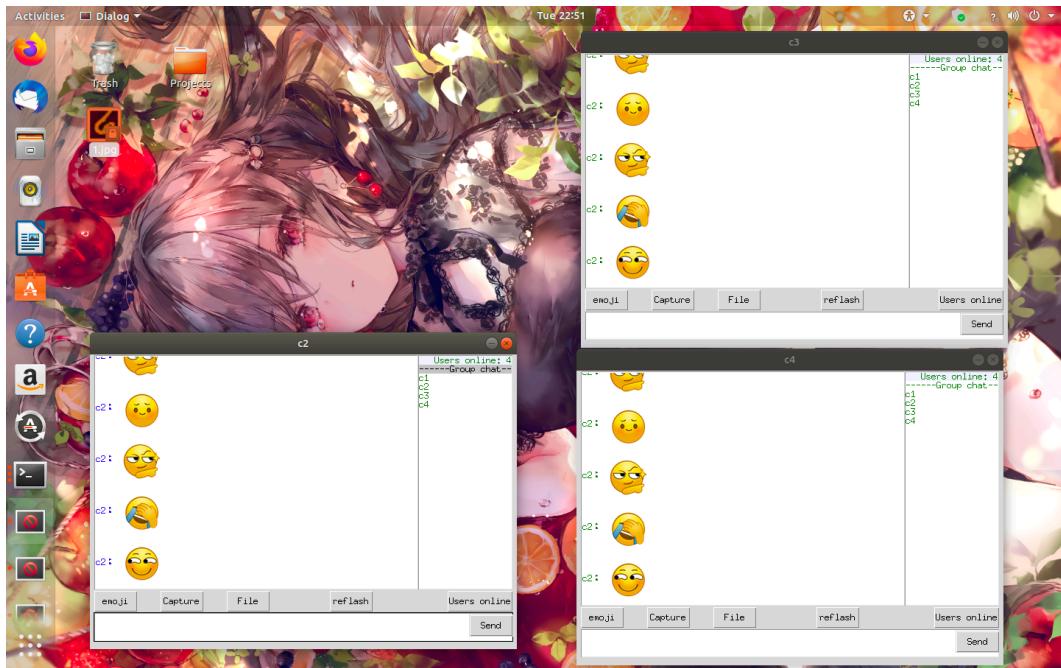


图 15: 群聊模式发送表情

私聊模式中发送表情，只有私聊的指定用户可以收到，其他用户都收不到。如图 16所示。

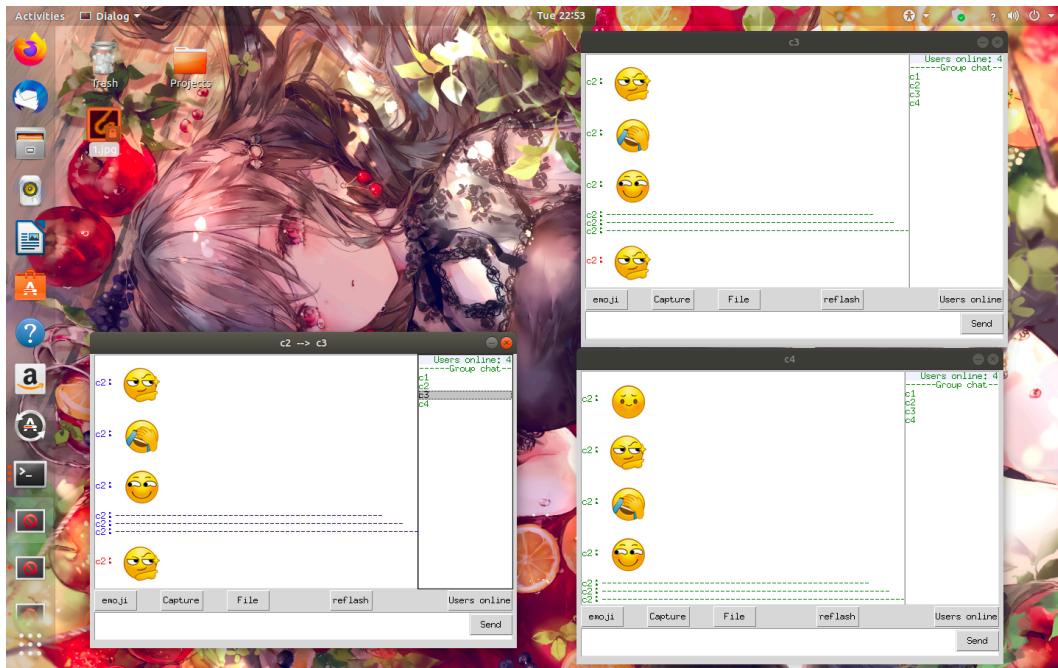


图 16: 私聊模式发送表情

3.7 文件传输

点击 *File* 按钮，可以在客户端与服务器另外建立文件传输的连接，并显示出服务器端端群文件夹目录。如图 18、17所示，图 18是服务器端的文件夹目录，图 17是客户端看到的群文件夹目录。

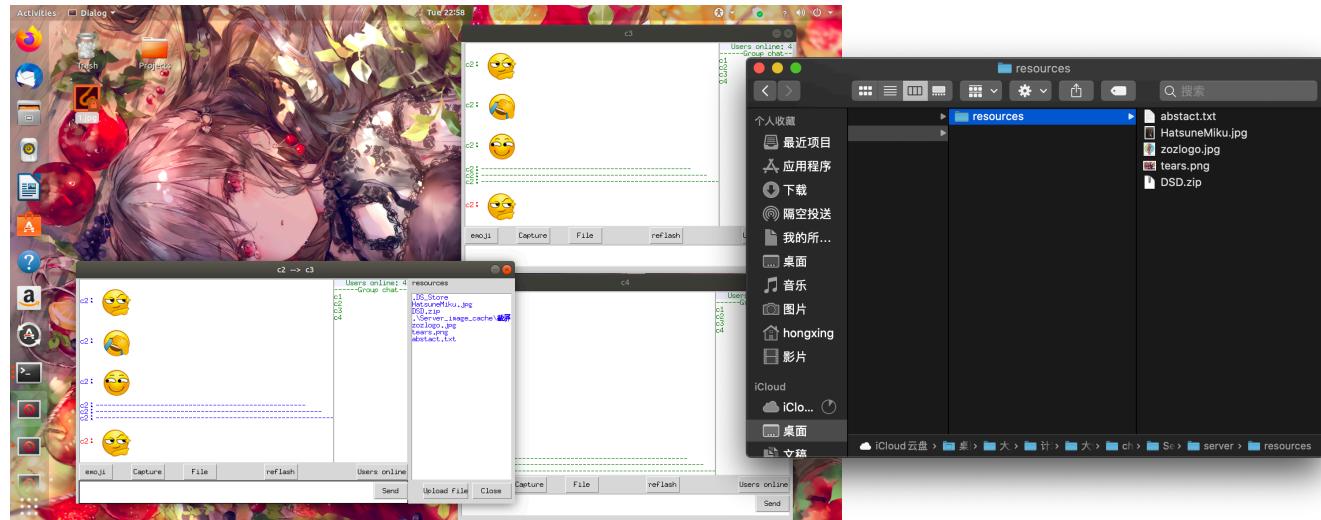


图 17: 客户端看到的群文件夹目录

图 18: 服务器端的文件夹目录

客户端点击显示出的群文件目录下方的 *Upload file* 按钮，将弹出文件选择窗口，选择本机的任意类型的文件，点击 *Open* 即可上传到服务器端（建议不要大于 10k，由于上传功能没时间优化了，容易卡住），其他用户打开群文件夹目录可以看到上传后到文件。如图 19、20所示。点击群文件目录下方的 *Close* 按键可以关闭群文件目录并关闭与服务器的文件传输连接。

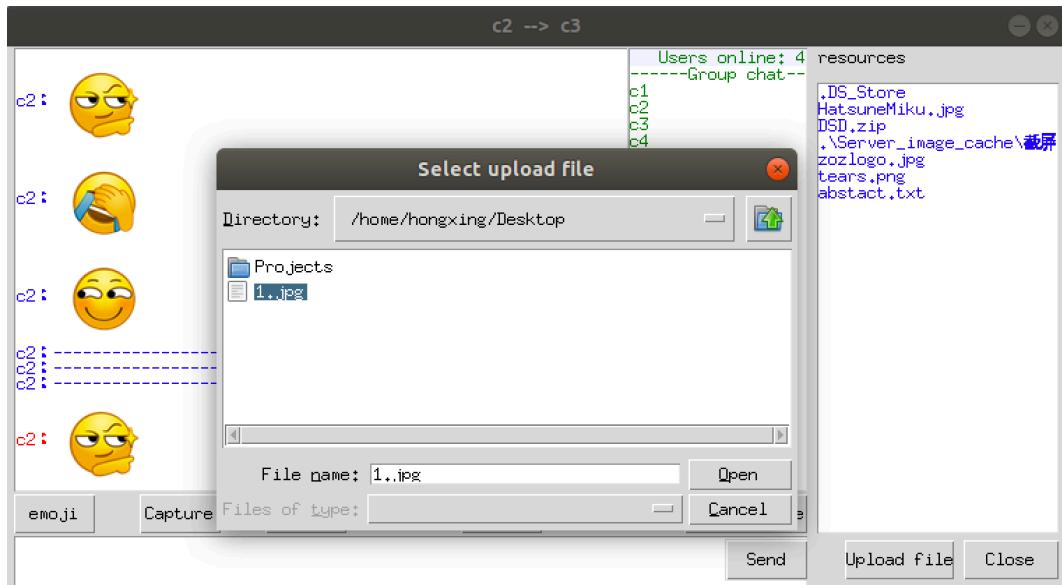


图 19: c2 选择本地桌面上的 1.jpg 上传



图 20: c1 可以看到群文件夹中多了 1.jpg

某客户向群文件夹上传文件后，其他用户便可从群文件夹中下载此文件。客户端点击群文件目录中的某个文件便会弹出下载窗口，选择一个路径点击 Open 按键即可下载到指定路径。如图 21、22所示。

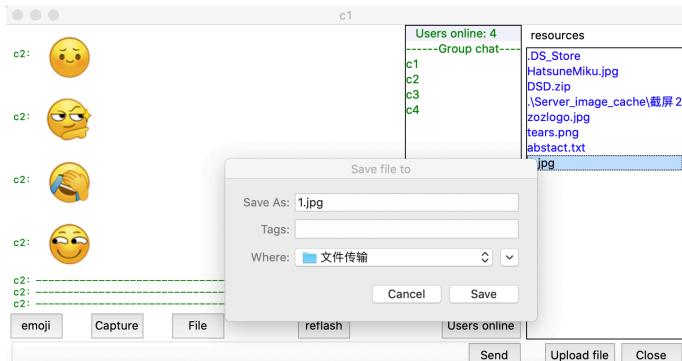


图 21: 弹出下载窗口

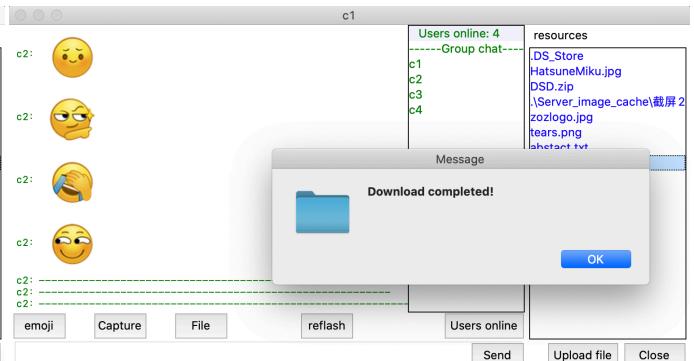


图 22: 客户端下载完成

3.8 全屏截图

客户端界面点击 *Capture* 按钮便可进行全屏截图，截好的图像将默认放入工作文件夹中。如图 23 所示。

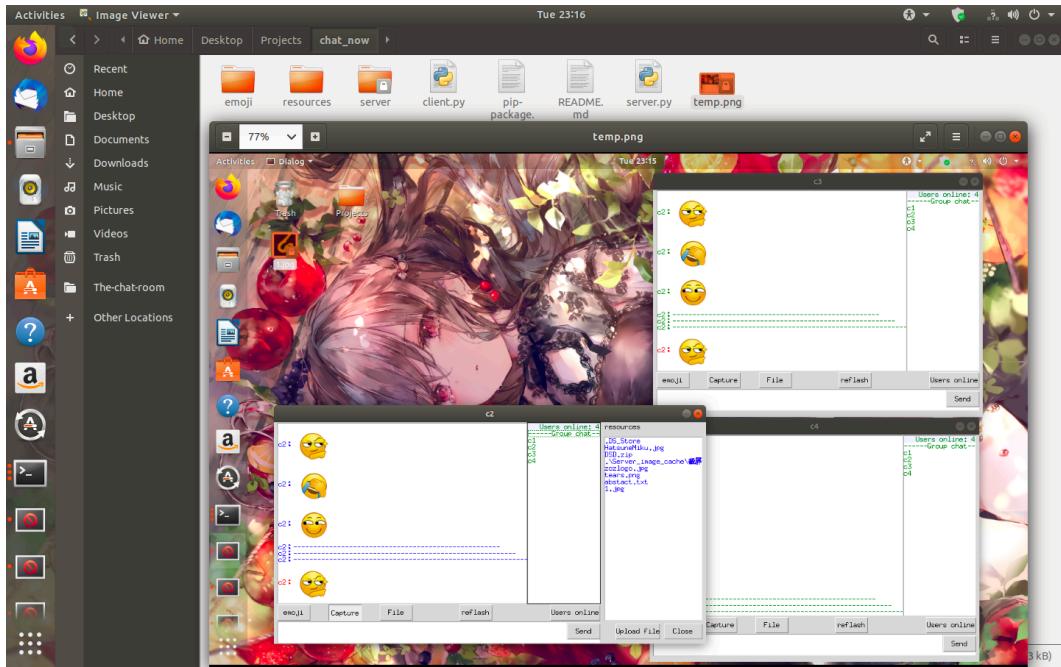


图 23: 全屏截图后可以在工作文件夹中找到截图 temp.png

3.9 刷新连接

在现实中经常发生网络环境差而断开连接，或服务器宕机重启的情况，导致客户端与服务器端的连接断开。如果没有很好的机制则只能重启客户端重新登陆。本聊天程序加入了刷新连接的功能，在客户端由于种种

原因与服务器端断开连接后，能够在不重启客户端的情况下以同一用户身份重新建立与服务器的连接，且之前互相往来的消息能够保留。在

点击 *reflash* 按键以刷新连接（Linux 下可能要多点几次，点一次可能只是断开了连接）。如图 24、25。

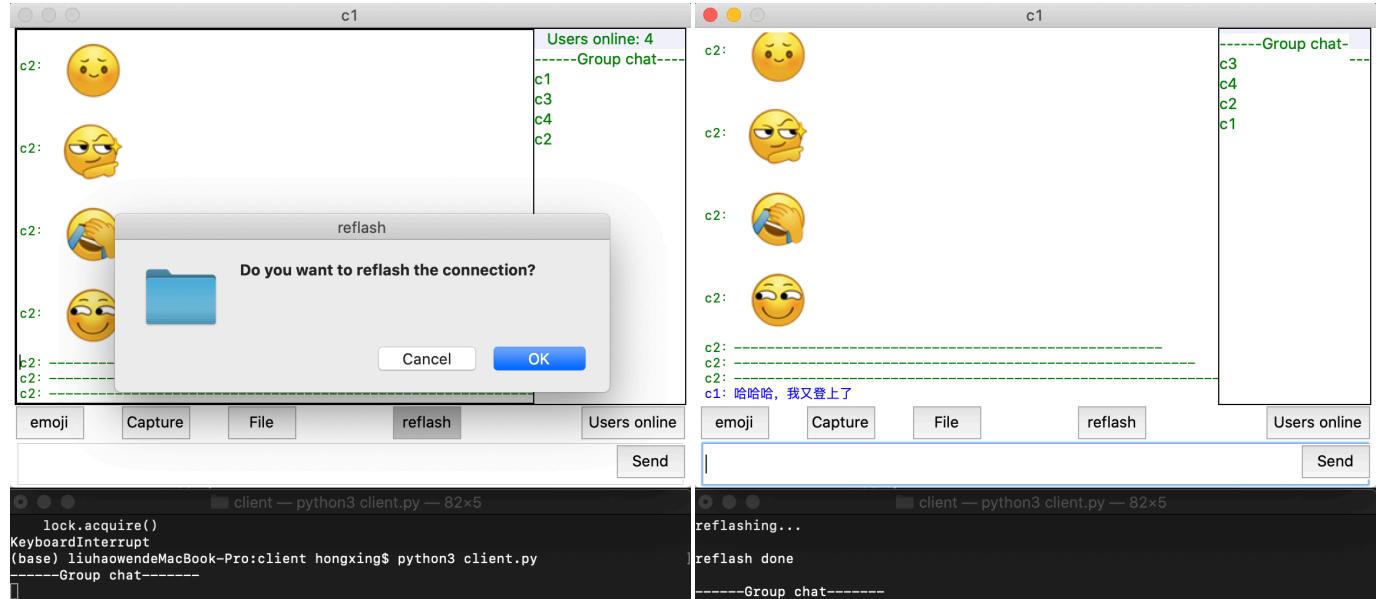


图 24: 刷新连接

图 25: 刷新连接成功

接下来做一个更为极端的实验：重启服务器，不重启客户端，用户通过刷新连接重新登陆。

重启服务器，如图 26 所示。由于没有自动重连功能，客户端与服务器全部断开连接，消息无法发送，如图 27 所示。



图 26: 服务器宕机

图 27: 客户端无法发送消息

重启服务器后，客户端点击 *reflash* 按键刷新连接，可以重新发送消息。可以看到用户列表中由于只有 *c1* 一个用户刷新了连接，所以目前只有 *c1* 一个用户在线，如图 28。这也反映出用户列表可以实时地反应在

线的用户。

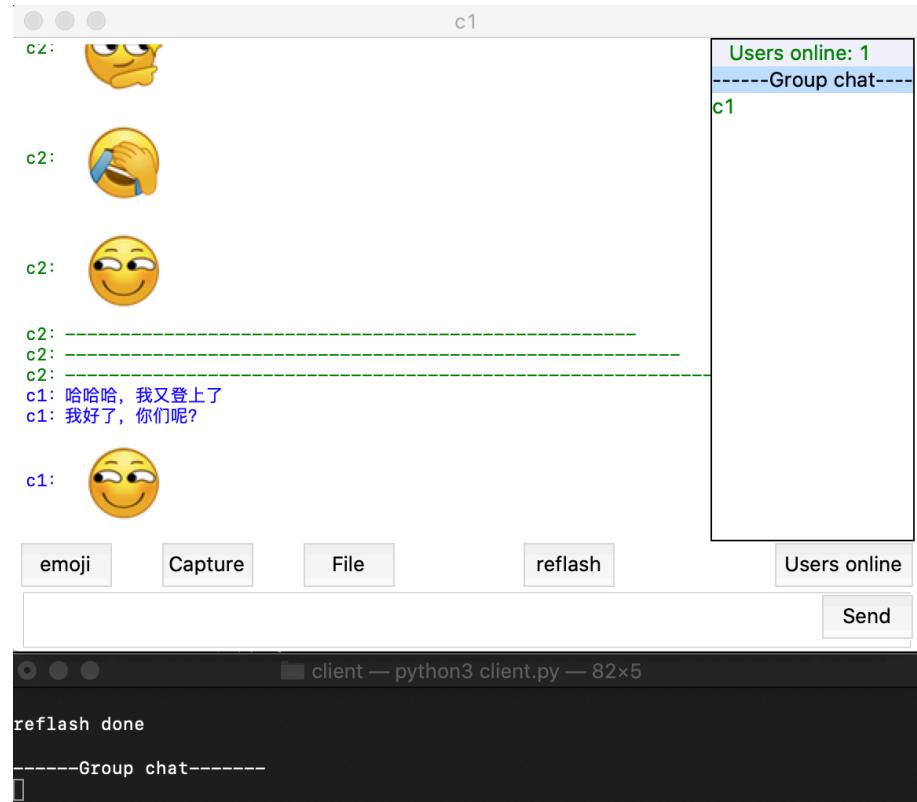


图 28: 用户 c_1 刷新连接, 只有 c_1 一个用户在线

c_2 、 c_3 、 c_4 依次刷新连接, 全部成功重新登陆到服务器。如图 29所示。

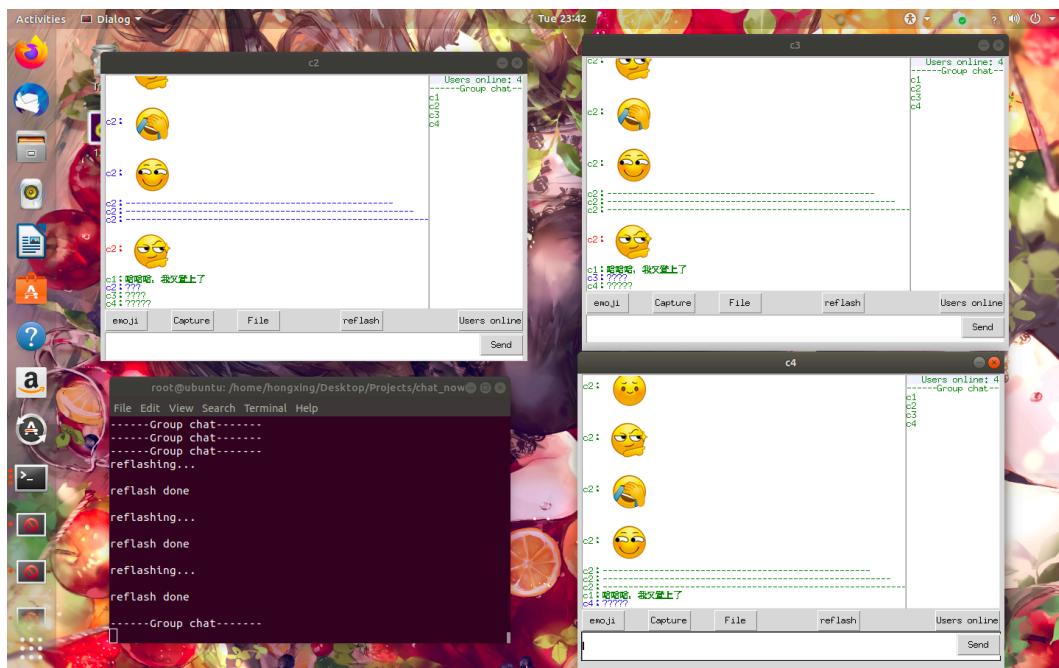


图 29: 所有用户全部刷新连接, 重新可以正常发送消息

在做其他测试没有得到正确响应时，也可通过点击 *reflash* 来获得正确响应。

3.10 显示与关闭用户列表

点击右下角的 *Users online* 按键可以显示或关闭用户列表，来获得更大的聊天消息显示空间，节约了客户端窗口占用的位置。如图 33所示。

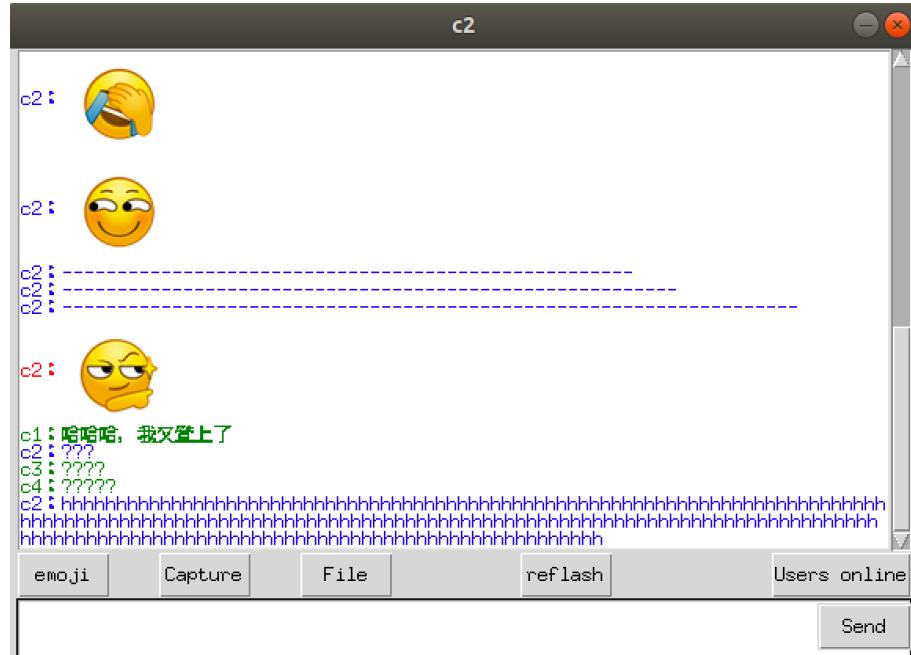


图 30: 关闭用户列表以获得更大的聊天消息显示空间

3.11 用户下线

关闭用户端窗口，点击确认后即可下线（如果使用命令行工具可能无法立即退出，此时采用 *control + c* 方法退出），可以看到用户列表显示的在线用户同步更新（如果没有更新请点击 *reflash*）。如图 31、32所示。

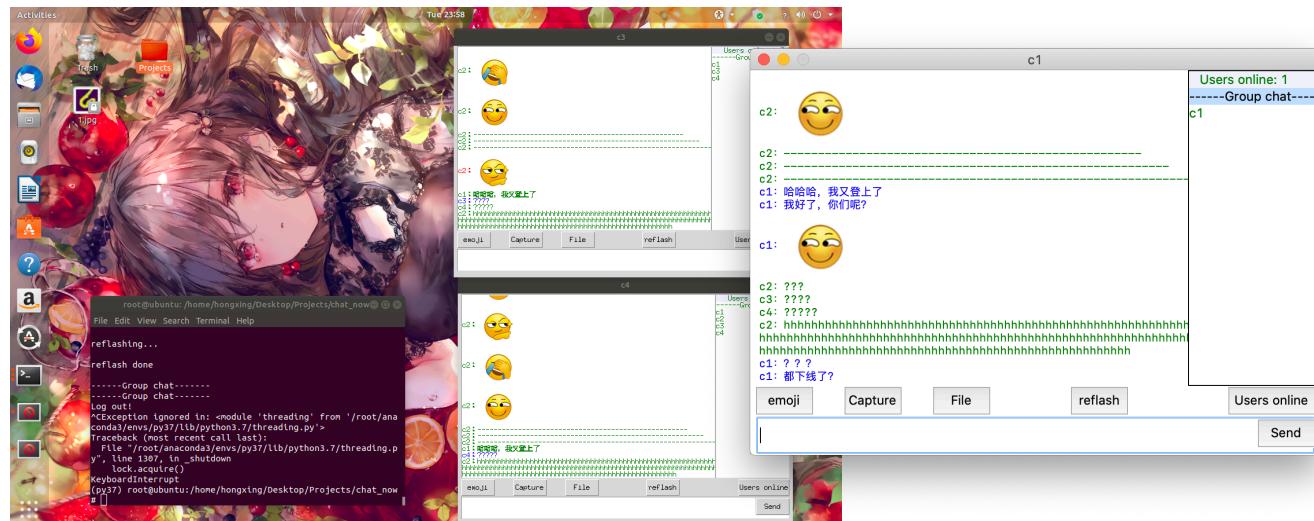


图 31: 用户 *c2* 退出

图 32: 用户 *c2*、*c3*、*c4* 退出

4 遇到的问题及解决方法

这次大作业由于工作量较大而时间略显不足，出现了很多问题。其中最大的问题就是不太理解 `socket.close()` 的机制，只是在客户端设置了主动断开连接的 `s.close()`（即断开连接的第一次挥手），但服务器端并没有设置断开连接的 `conn.close()`（即第三次挥手），导致服务器认为连接仍存在，继续向已经不存在的客户端转发消息（由于转发逻辑是全用户转发）。但此时连接已不存在，转发出错，导致服务器进程崩溃，必须重启服务器。这就造成了一旦有一个用户下线服务器就立即崩溃的情况，显然不能作为一个聊天工具使用。这个 *bug* 耗费了我大量的时间才发现，最后修复仅仅只是在 `delUsers` 函数中加一行代码 `conn.close()`。为了让程序可用性更强，我又加入了刷新连接的功能，大大提高了程序的可用性与稳定性。

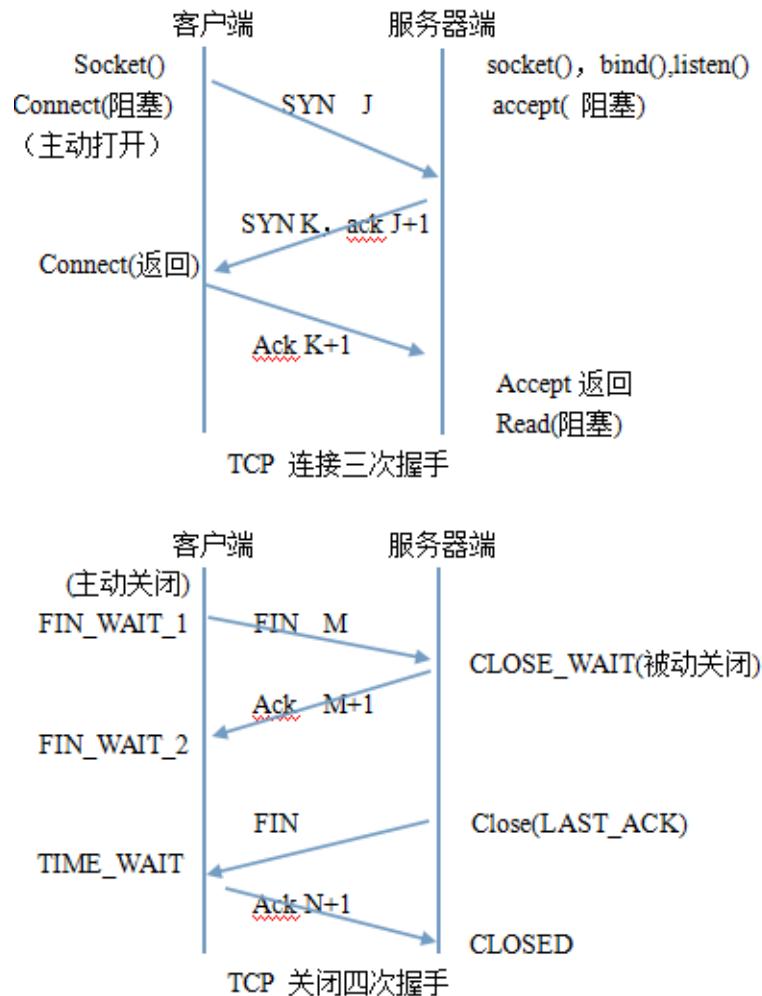


图 33: 三次握手与四次挥手 [4]

5 体会与建议

5.1 体会

这次大作业对没有应用程序编程经验、`python` 使用不熟练的我来说是一个巨大的挑战。之前编过的最复杂的程序也只是信息安全数学基础课程上编写的 `RSA` 公钥密码算法，是纯粹的数理逻辑编程。而这次编写聊

天软件是基于已有的库的编程，对我来说是全新的体验。任务刚上手是非常困难的，我去各大论坛、博客、官网上寻找相关的资料进行学习，去学习 *Socket* 编程的理论知识，去学习各种库的使用方法，去看各种已实现的代码进行学习，通过自己程序的报错去找错误的原因，才最终完成了这个比较完整的程序。这个过程虽然劳累，但让我学到了不少知识，巩固了课堂上学习的 *TCP/UDP* 连接的知识，也能作为我的应用程序编程的开端。这次大作业完成的聊天程序还有很多不足，比如说 *server* 的问题没有从根本上解决，私聊模式不安全等问题，一方面是由于时间不够，另一方面是由于自己对 *Socket* 编程的理论还不太了解。或许在考完试后的寒假，我能对这个程序做进一步改进吧。

5.2 建议

建议在布置大作业前系统介绍一下 *Socket* 编程的重点知识。网上的知识太零碎而不全面，且经常相互矛盾，需要有一个正确的导向才能有效地学习与编程。

6 Reference

- [1]CSDN.Dec 25.<https://www.cnblogs.com/-3866/articles/9187604.html>
- [2]CSDN.Dec 25.<https://blog.csdn.net/slavik/article/details/82430717>
- [3]CSDN.Dec 25.<https://blog.csdn.net/qq41742007/article/details/83377666>
- [4]CSDN.Dec 25.<https://blog.csdn.net/sdlcwangsong/article/details/21394665>
- [5]GITHUB.Dec 25.<https://github.com/11ze/The-chat-room>