

**北 京 科 技 大 学**

**计算机网络课程设计报告**

学　　院： \_\_计算机与通信工程学院\_\_

班　　级： \_\_\_\_\_\_\_\_信安182\_\_\_\_\_\_\_\_\_

姓　　名： \_\_\_\_\_\_\_\_\_梁逸飞\_\_\_\_\_\_\_\_\_\_

学　　号： \_\_\_\_\_\_\_\_41824141\_\_\_\_\_\_\_\_\_

成　　绩： \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

指导教师签字： \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**年 月**

**北京科技大学实验报告**

学院：计算机与通信工程学院 专业：信息安全 班级： 信安182

姓名：梁逸飞 学号：41824141 实验日期：2021年 5月

**实验名称：**计算机网络课程设计 实验任务三——基于SOCKET的自动更新的可视化多人聊天室

**实验目的：**

基于SOCKET搭建一个可视化多人聊天室，并实现客户端的自动在线更新功能。

**实验仪器：**

实验硬件设备：华硕飞行堡垒， Intel(R) Dual Band Wireless-AC 8265

实验软件要求：win10，pycharm， python3.6

**实验原理：**

基于SOCKET可以实现聊天室的信息或文件传输，并自己新编写一个升级程序，每次启动客户端前会自动检测是否需要更新，利用tkinter将聊天室可视化。

**实验内容与步骤：**

**题目：基于SOCKET的自动更新的可视化多人聊天室**

1 需求分析

选择python语言，在实验一和二的基础框架上，实现一个并发双向通信的C/S架构体系，能够使得服务器和客户端之间传输信息和文件。同时客户端还能实现自动更新功能。

2 设计方案

2.1 配置文件

首先编写配置文件Setting.py，在里面定义客户端和服务器所需要的端口、IP等，文件传输允许的数据长度以及引入涉及到的所有依赖。

同时实现三个函数，一个set\_window\_center()，是将窗口置于屏幕中央，第二个是set\_main\_window()，设置启动窗口的格式，第三个missing\_conf()，是检测客户端和服务器配置文件的函数，如果缺失则会报错并退出程序。

2.2 服务器

2.2.1 配置文件

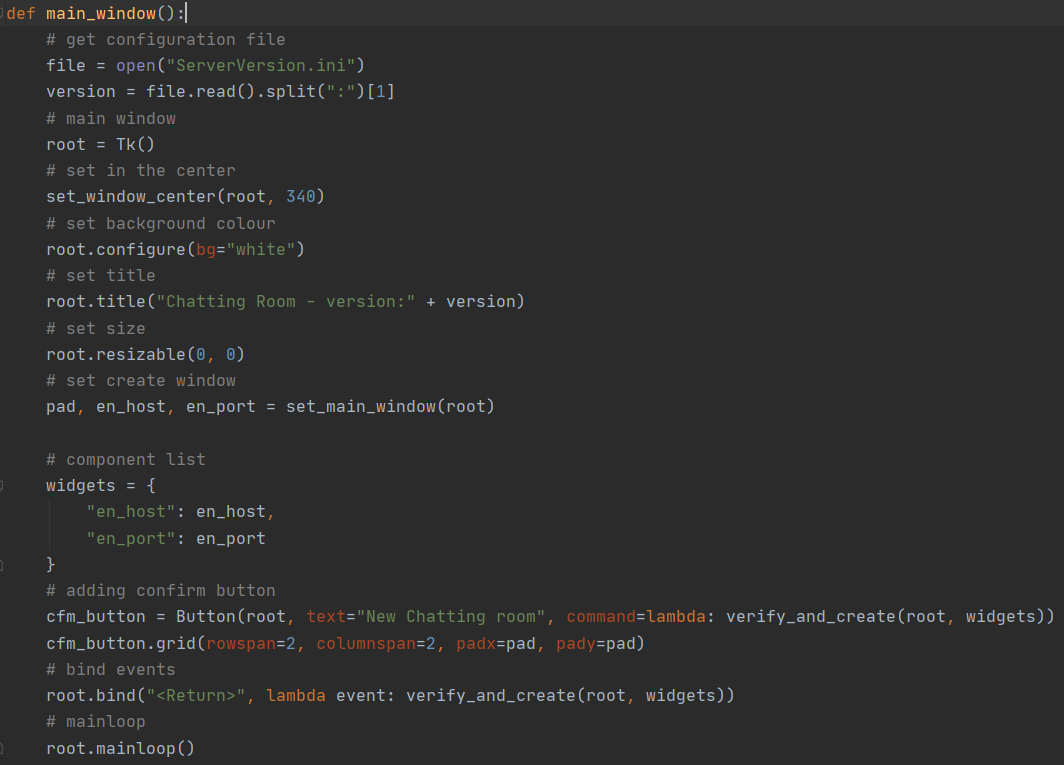
设置服务器的配置文件ServerVersion.ini，包含版本信息。

2.2.2 服务器设计

2.2.2.1 main\_window()函数

在main\_window()函数中，先获取配置文件，然后调用Setting.py里的函数设置窗口格式，设置创建聊天室的按钮并绑定事件函数verify\_and\_create()，一旦点击按钮则会调用该函数。

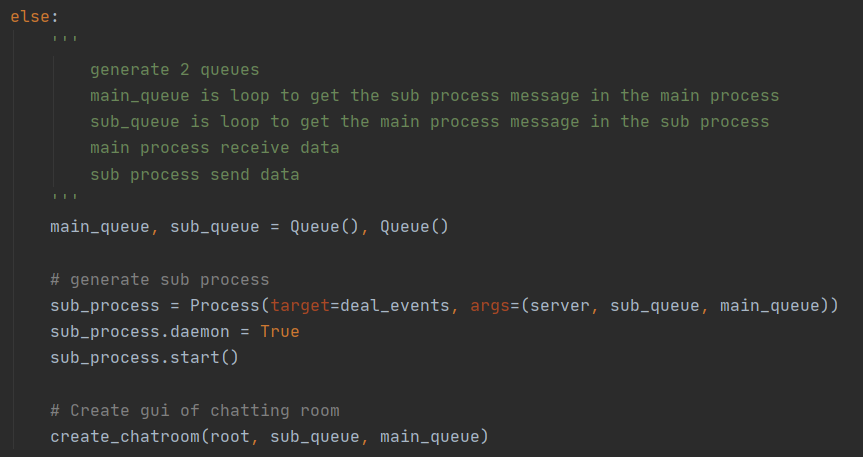
main\_window()也是主函数的入口点。



2.2.2.2 verify\_and\_create()函数

在verify\_and\_create()函数中主要功能是实现验证和创建的功能。他会验证端口和主机IP是否正确，如果正确，则会开始创建服务器socket并开始监听。创建成功后，创建两个队列，这里采用双进程主队列用于在主进程里循环获取子进程的信息，主要是用于服务器获取客户端消息，子队列用于在子进程里循环获取主进程的信息，主要是用于服务器发送消息。所有需要发送的消息以及接收到的消息都放到这两个队列中，之后统一处理。

生成子进程，调用函数deal\_events()，之后会说到，主要是用于处理消息文件的发送和接收以及转发。最后调用create\_chatroom()函数创建聊天室。

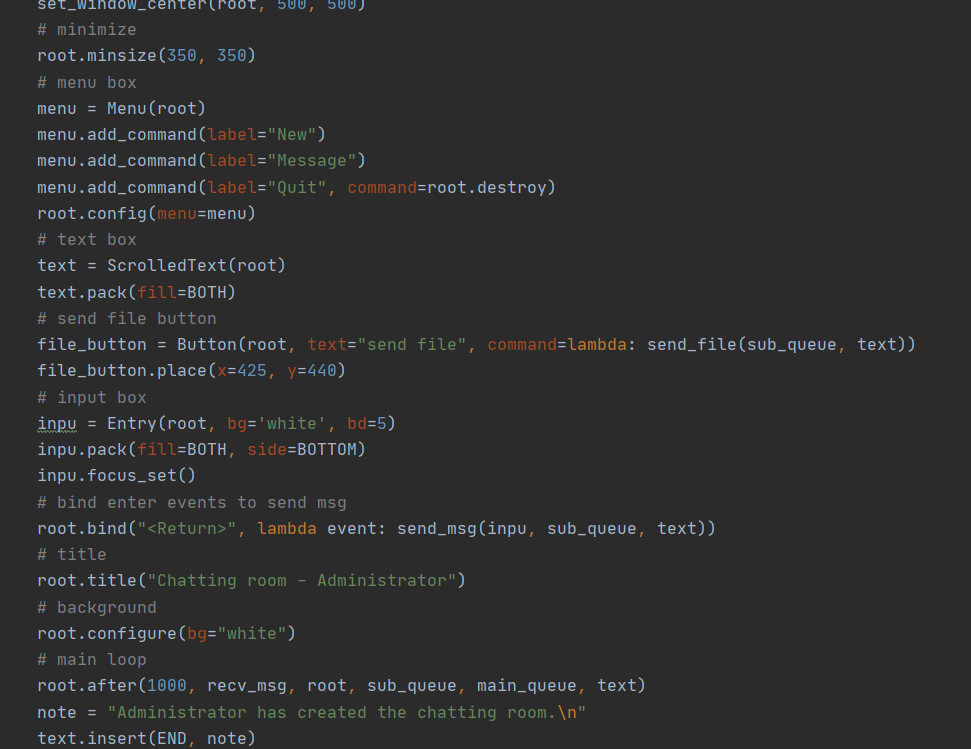


2.2.2.3 create\_chatroom()函数

该函数主要实现的功能是创建聊天室的窗口，包括显示的文本框、消息输入框以及文件发送按钮。文件发送按钮绑定事件send\_file()函数进行文件传输。

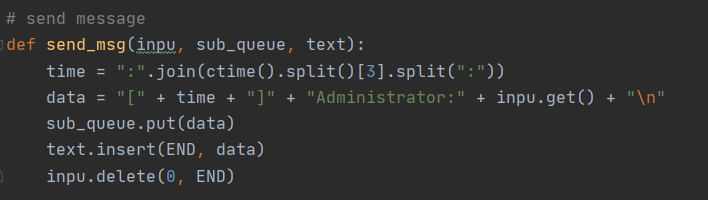
输入框里将回车按钮绑定事件send\_msg()，输入框里输入消息并按下回车时，则会调用send\_msg()函数发送消息。

最后通过tkinter的after()函数每隔1000ms调用函数recv\_msg()用于获取信息。



2.2.2.4 send\_msg()函数

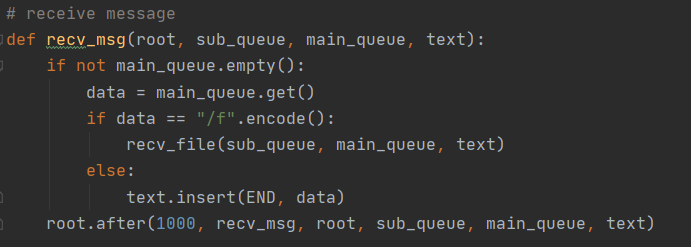
该函数就是用于发送消息的了。为了界面友好，发送消息前先获取当前时间，然后组合成格式为[time]who:msg的格式，然后放入子队列中。在deal\_events()函数中会对子队列的信息进行发送。



2.2.2.5 recv\_msg()函数

该函数主要是在主进程里获取主队列里的消息。在deal\_events()函数中会将客户端收到的信息放入主队列中，如果主队列不为空，则说明有客户端发来消息，则此时应显示在文本框中。

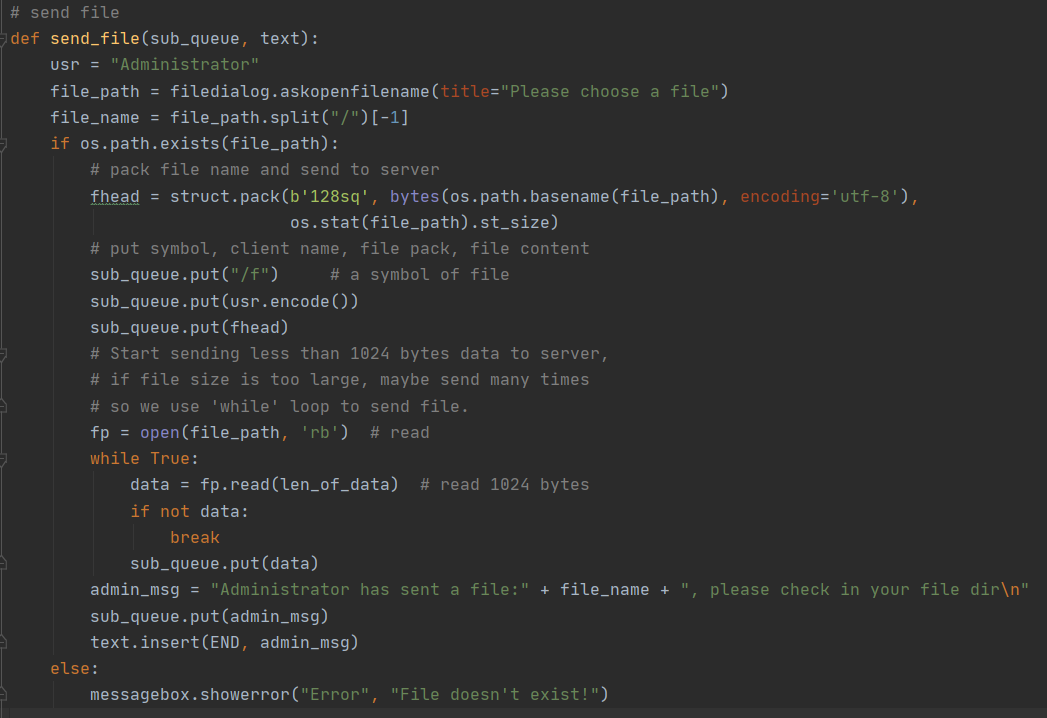
收到消息首先会判断是否是”/f”字符，由于消息传输和文件传输都是通过socket的send和recv函数，所以为了辨别我将”/f”设定为发送文件的标志，即发送文件之前我会首先发送”/f”字符用于标识。如果是该字符，则调用recv\_file()函数接受文件，否则就直接显示data。



2.2.2.6 send\_file()函数

通过tkinter的askopenfilename函数，能够浏览本地文件，返回值就是该文件的绝对路径。

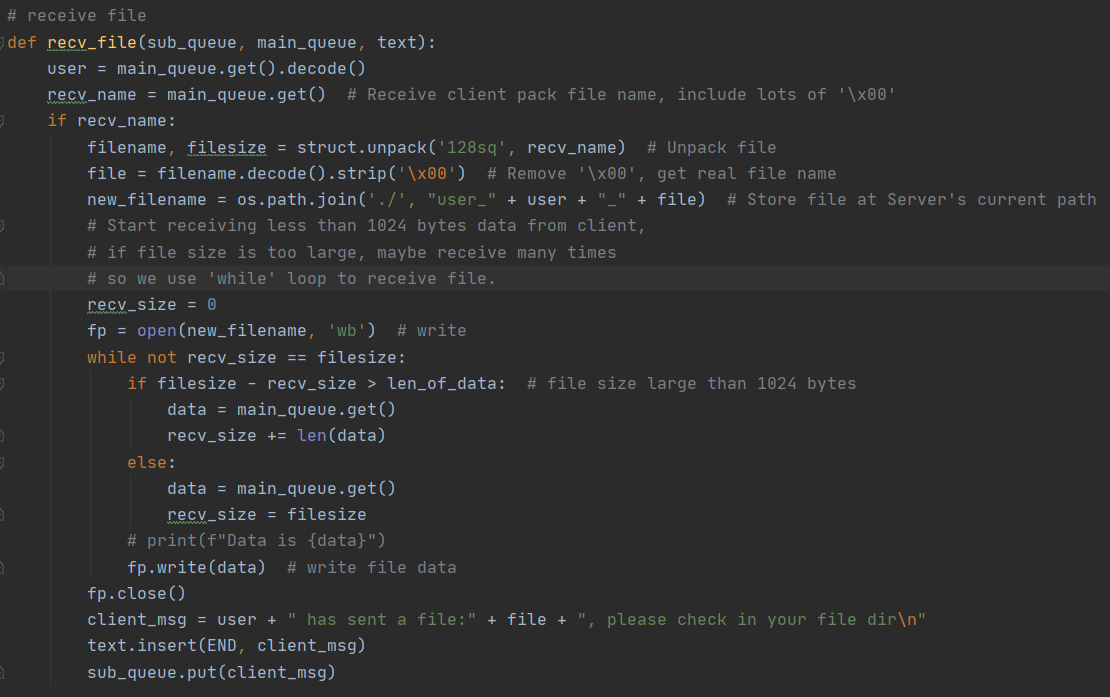
和实验二文件传送一样，通过struct库用128sq格式对文件进行打包，然后依次将字符”/f”，用户名(Administrator)，打包头放入子队列中。然后打开该文件，循环读取len\_of\_data长度的文件内容，并依次放入到子队列中。最后为了提示客户端服务器发送了文件，需要发送一个提示消息，格式也是[time]who:msg，放入到子队列中。客户端中发送文件的函数也是相似的，包括插入的字符和内容的顺序。



2.2.2.7 recv\_file()函数

该函数就是接受文件的了。原理和实验二中的一样，根据send\_file()中插入的内容，主队列中首先是”/f”字符，该字符在recv\_msg()函数中就被获取然后识别再跳转到该函数里，所以主队列之后的内容依次是用户名、打包头和文件内容。

文件接收也是一个循环过程，循环接收并写入len\_of\_data长度的内容，将其保存到当前目录下，格式是user\_name\_file，最后发送消息提示语，提示哪个客户端发送了一个文件，将其放入到子队列中。



2.2.2.8 deal\_events()函数

该函数是核心，用于处理消息的发送和接收，还有一个重要的功能，转发。

由于客户端发送的内容都是发送给客户端的，而多人聊天室是所有用户都能看见谁发送了消息，所以需要通过服务器将消息转发给其他的客户端。

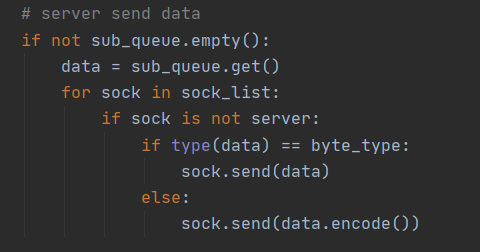
需要注意的是，无论是服务器自身发送消息还是转发客户端发来的消息，都是对子队列里的内容进行处理。

2.2.2.8.1 发送消息

在此之前创建了sock\_list，之后凡是加入聊天室的客户端，都会加入其socket。

发送消息就循环读取子队列里的内容，然后发送给sock\_list里所有不是server的socket。

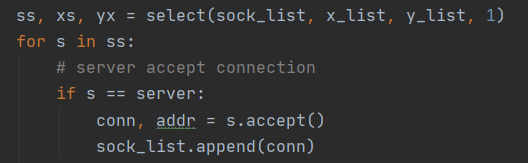
这里需要判断数据类型是不是byte类型。因为发送消息和发送文件不一样，发送消息需要编码成二进制类型，接收之后再解码，发送文件的话本身发送的就是二进制内容，收到之后不需要进行解码，否则就会报错。



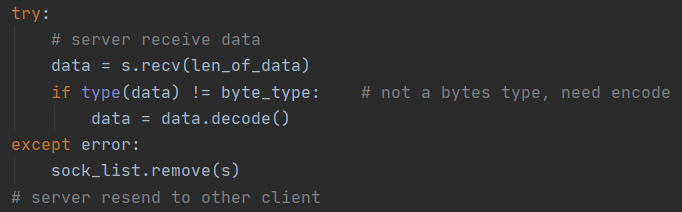
2.2.2.8.2 接受连接

这里采用select方法处理socket，只要有socket变动（不管连接还是断开连接还是其他），变动的内容就会放到首个位置，即ss。

如果s是server，则说明有客户端进入，此时接受连接并加入到sock\_list当中。

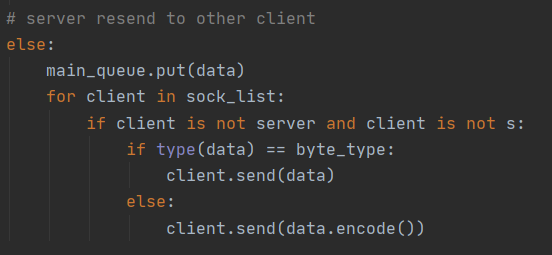


2.2.2.8.3 接受数据



同样在该循环当中，如果s不是server，说明有客户端发送了消息过来，服务器需要接受内容。

2.2.2.8.4 转发

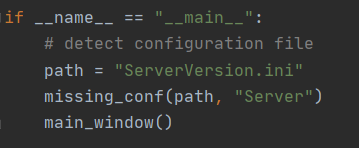


接受到了数据之后，需要转发给不是该客户端的其他客户端。

2.2.2.9 主函数

主函数就只需要进行函数调用就行了。

首先检测配置文件，然后调用main\_window()函数。



2.3 客户端

2.3.1 配置文件

同2.2.1，设置客户端的配置文件ClientVersion.ini，包含版本信息。

2.3.2 客户端设计

2.3.2.1 main\_window()函数

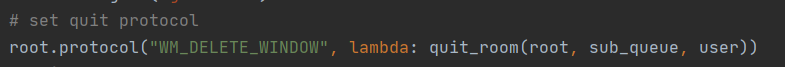
同2.2.2.1，不再赘述，只是在文字内容显示上有点区别，同时多了一个输入用户名的输入框。加入按钮绑定事件verify\_and\_create()。

2.3.2.2 verify\_and\_create()函数

同2.3.2.2，不再赘述，在内容处理上多了一点就是用户名不能为空，然后创建了客户端的socket的同时立马发送一个消息提示聊天室里的所有人有谁进入了该聊天室。同样要创建两个队列，生成子进程，绑定deal\_events()事件，最后调用create\_chatroom()。

2.3.2.3 create\_chatroom()函数

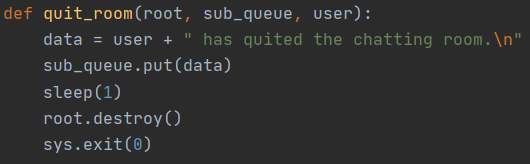
同2.2.2.3，绑定send\_file()、send\_msg()事件，此外定义了一个客户端退出规则，即点击右上角的×会退出，绑定了事件quit\_room。



2.3.2.4 quit\_room()函数

发送提示消息，告诉聊天室里的所有人有谁退出了，将其放到子队列中。

然后销毁窗口，退出程序。



2.3.2.5 send\_msg()函数

同2.2.2.4。

2.3.2.6 recv\_msg()函数

同2.2.2.5，多了个数据类型为404，在deal\_events()中会提到，是服务器关闭了。之后同样的需要判断”/f”标识字符。

2.3.2.7 send\_file()函数

同2.2.2.6，基本上一模一样。

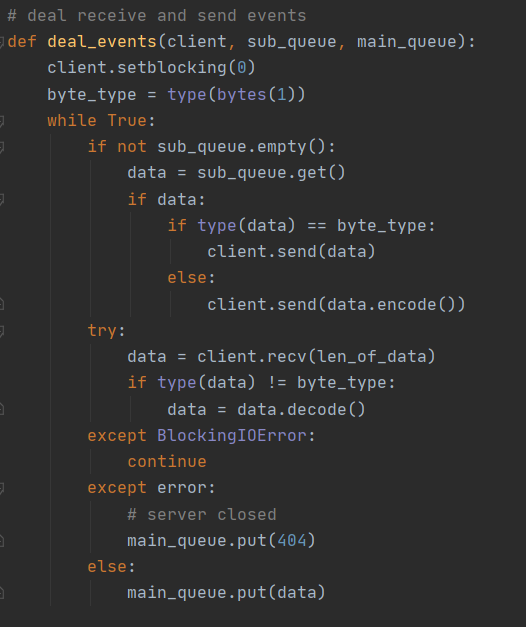
2.3.2.8 recv\_file()函数

同2.2.2.7，基本上一模一样。

2.3.2.9 deal\_events()函数

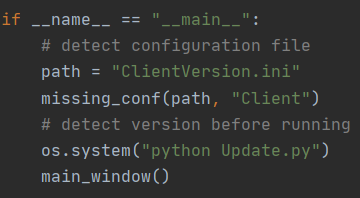
设置非阻塞，然后进行两个操作：发送消息和接收消息。

如果出现阻塞的I/O错误，则继续往下走，如果出现其他错误，则说明服务器关闭了连接，在主队列中加入404，用于报错提示客户端。



2.3.2.10 主函数

同2.2.2.9，在调用main\_window开始之前，中间多了个步骤，即启动升级程序，判断客户端是否需要升级更新。

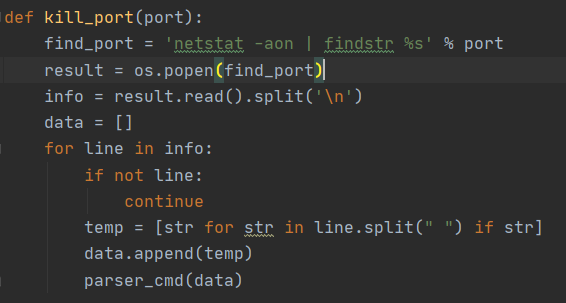


2.4 进程监控关闭程序

编写了killer.py文件，当客户端需要进行更新升级的时候，需要先关闭程序。由于win10下无法根据文件名获取其pid（所有python程序运行时的进程名都叫python.exe），所以killer.py文件用于关闭客户端所在端口的所有程序。

2.4.1 kill\_port()函数

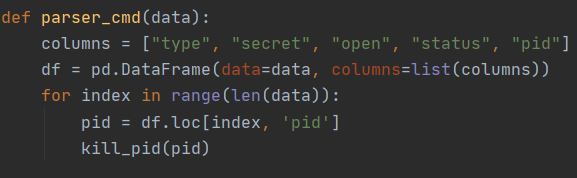
通过windows的cmd命令获取port端口下的所有进程信息，信息格式如下，然后调用parser\_cmd()函数获取pid。





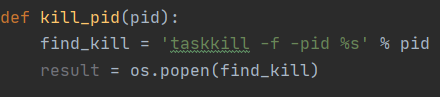
2.4.2 parser\_cmd()函数

通过上述格式获取每个进程的pid，然后调用kill\_pid()函数杀死进程。



2.4.3 kill\_pid()函数

直接通过windows的cmd命令执行操作。



2.5 升级程序

编写升级程序Update.py用于检测客户端版本是否是最新的，不是的话则需要更新。

2.5.1 detect\_update()函数

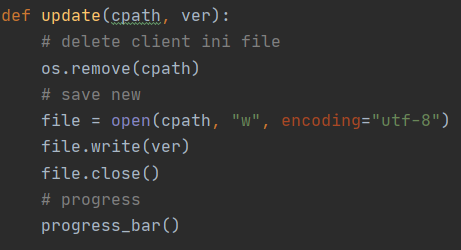
在该函数里获取客户端和服务器的配置文件里的版本信息，并进行比较，如果需要更新，则调用showinfo函数弹出确认框，是否需要进行升级，点击确认返回值为”ok”，点取消或×则返回值是”cancel”。

如果是”ok”，则先调用killer.py里的函数kill\_port()杀死端口下的所有进程，然后再调用update()函数进行升级。



2.5.2 update()函数

首先删除原有的配置文件，然后再将服务器的配置文件信息下载到客户端的对应当前路径下。



2.5.3 progress\_bar()函数

创建了更新的进度条窗口，使更新进度可视化。

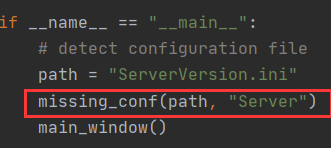
2.5.4 fill\_bar()函数

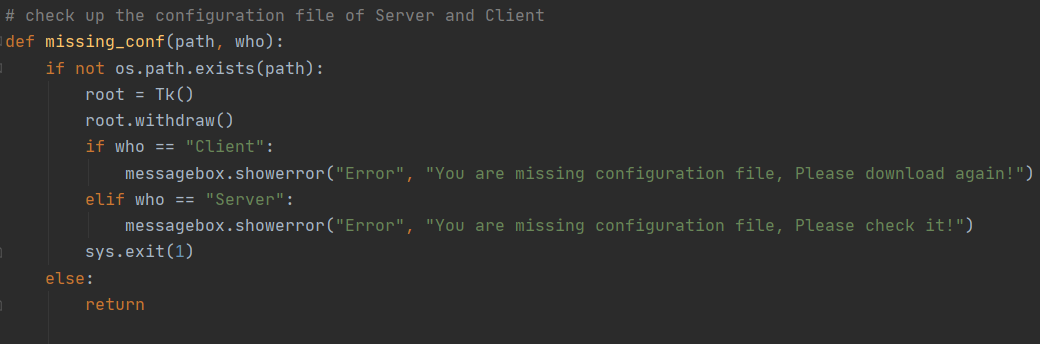
显示填充进度条过程。

3 异常处理

3.1 主函数中的异常处理

运行服务器程序或者客户端程序的时候都需检测他们的配置文件是否存在，此时调用Setting.py文件里的missing\_conf()函数进行判断，如果缺失，则报错提示用户，并退出程序。

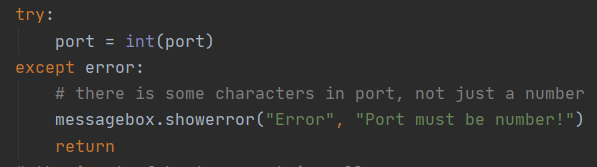




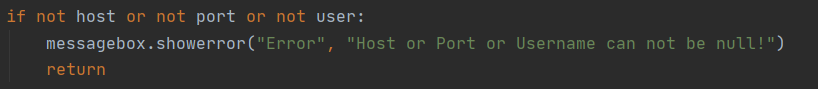
3.2 verify\_and\_create()中的异常处理

3.2.1 端口、IP、用户名处理

在客户端和服务器中，获取组件端口时有可能输入的不是一个整形数据，所以要进行异常处理，并弹出报错框。

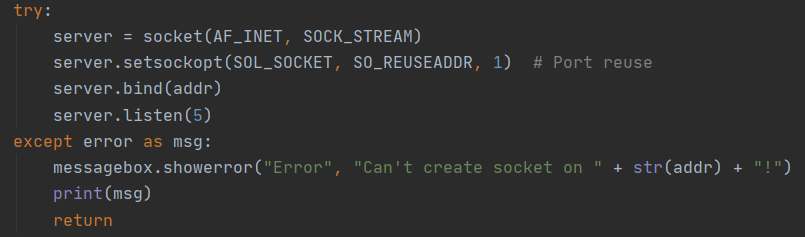


同时端口、IP、用户名不能为空。



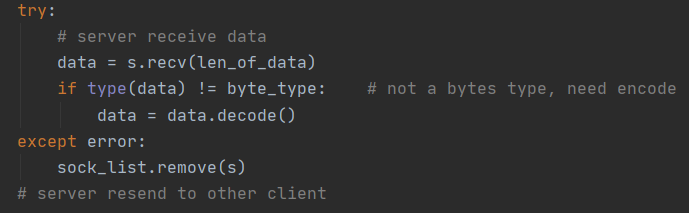
3.2.2 SOCKET创建

SOCKET创建、绑定时候可能会失败。

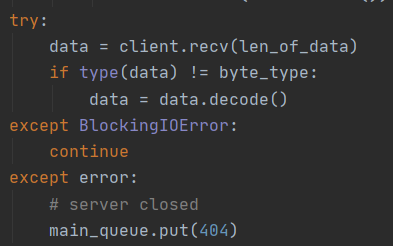


3.3 deal\_events()中的异常处理

在服务器中，获取客户端发送的消息可能会失败，则此时需要在sock\_list当中移除该客户端的socket。

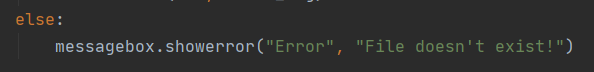


客户端中获取服务器发送的消息也可能失败，有可能是在非阻塞的套接字上执行阻塞操作时发生，则此时只需等待资源可用即可，若为其他，则是服务器关闭了连接，在主队列中放入404数据用于判断。



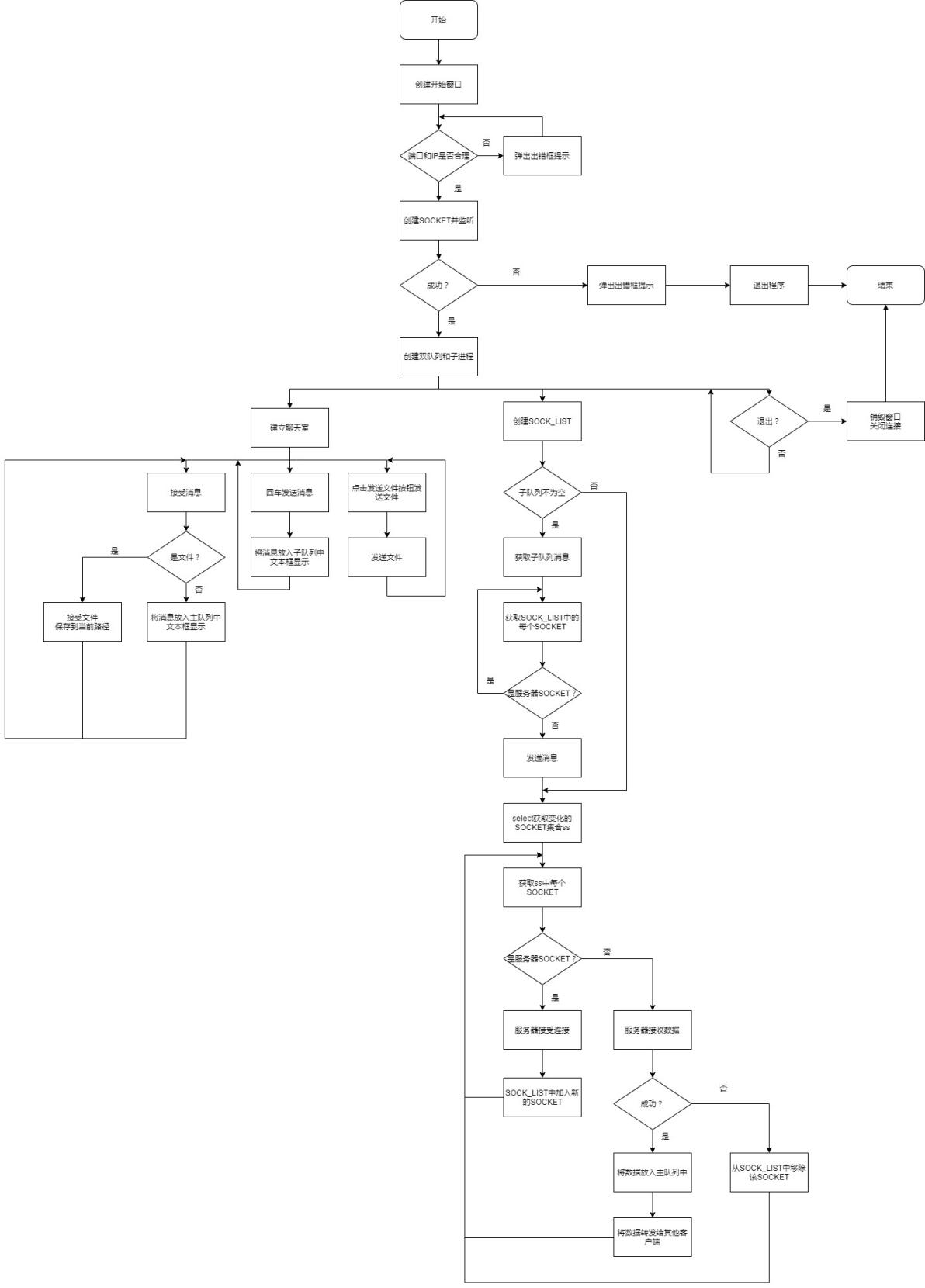
3.4 send\_file()中的异常处理

当用户选择的文件不存在（一般不可能）或者点击发送文件按钮又取消后，文件路径都是不存在的，需要异常处理。

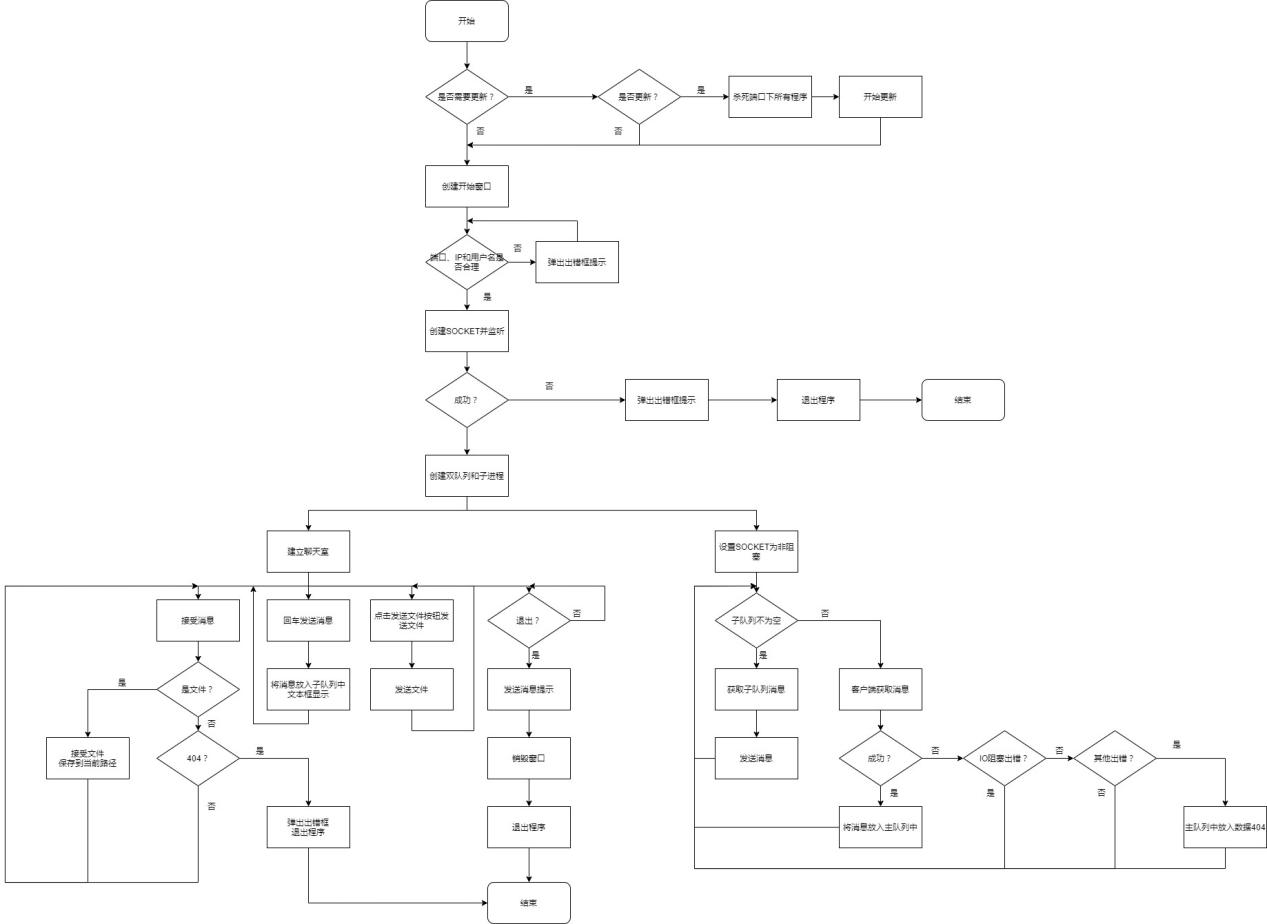


4 程序结构和流程图

4.1 服务器程序流程图



4.2 客户端程序流程图

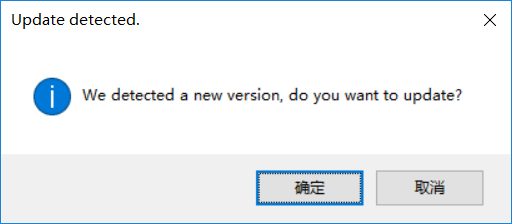


**实验结果与分析：**

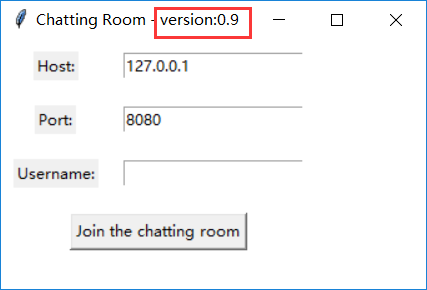
1 程序运行结果与分析

1.1 测试在线更新功能

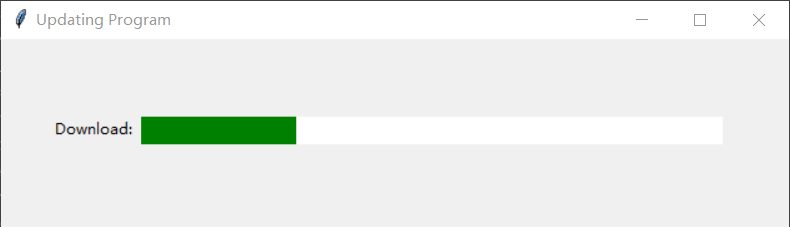
修改服务器版本为1.0，客户端版本为0.9，启动客户端，结果如下，会提示需要更新。



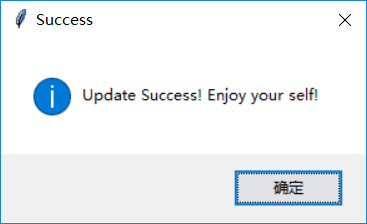
点击取消，则会正常运行程序，可以看到版本号为0.9。



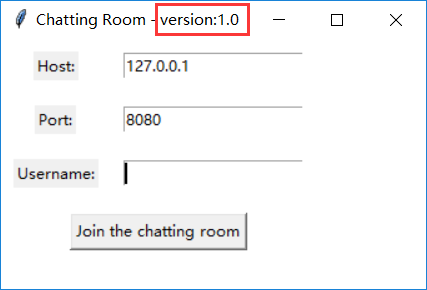
点击确定，开始更新，结果如下。



此时正在更新，稍等片刻后，更新结束。

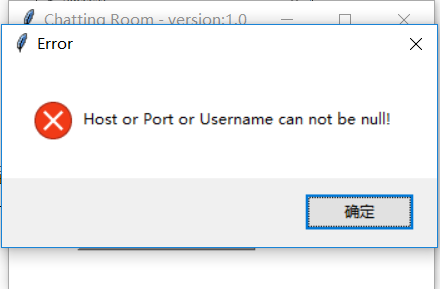


点击确定，启动程序，可以发现版本号变为了1.0。

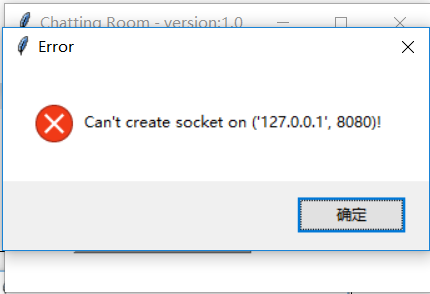


1.2 测试异常处理功能

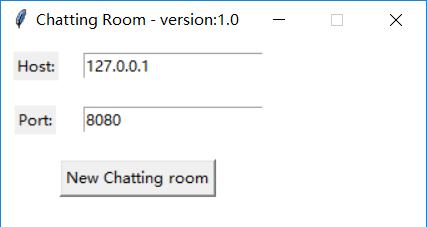
启动客户端，username为空，直接点击按钮，结果如下。

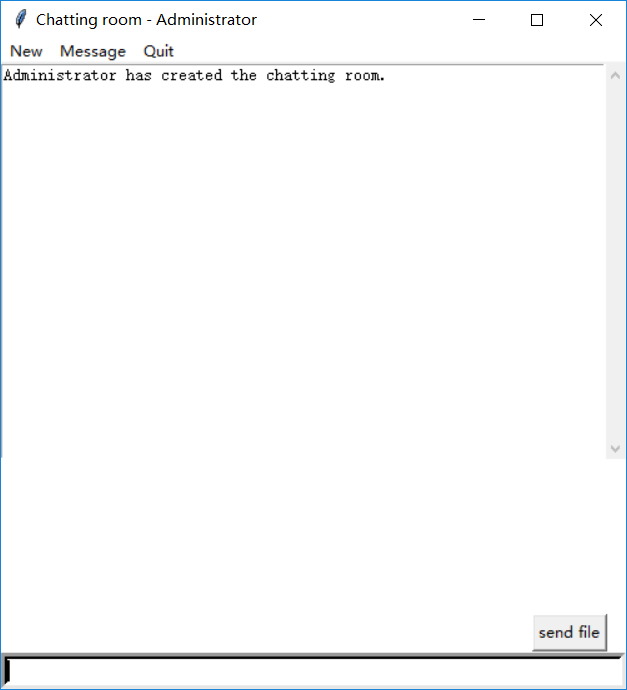


输入lyf，点击按钮，此时服务器未开启，结果如下。

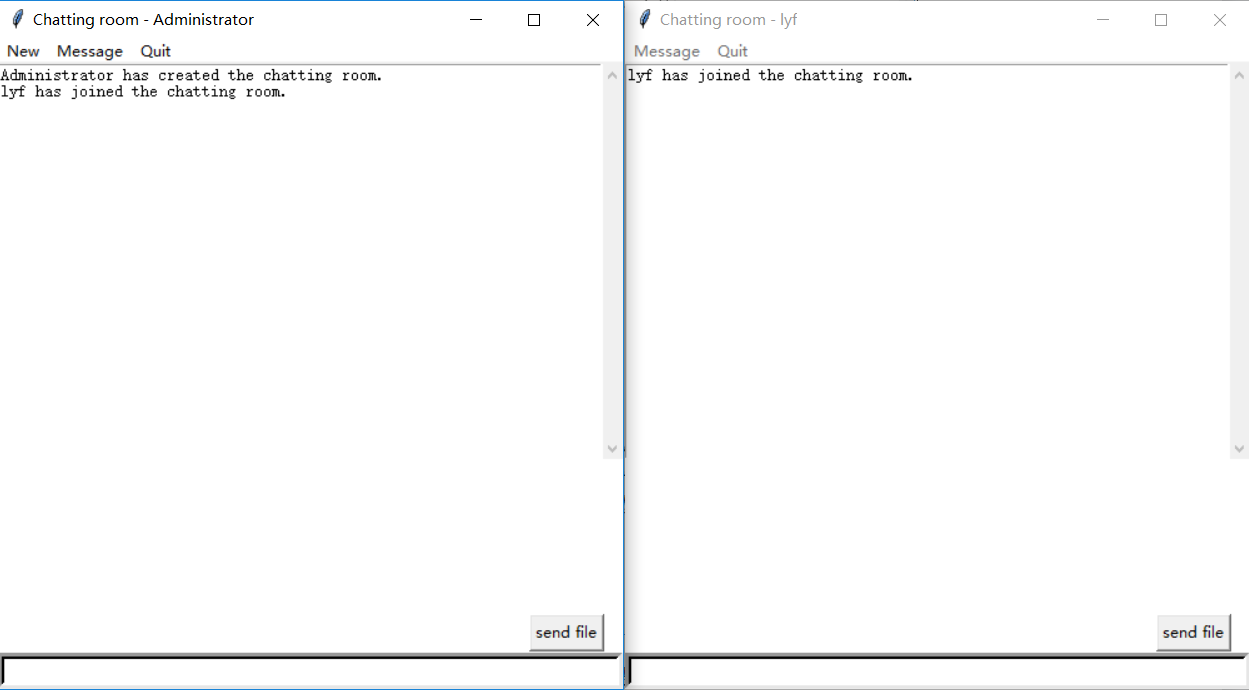


运行服务器，点击创建，结果如下。

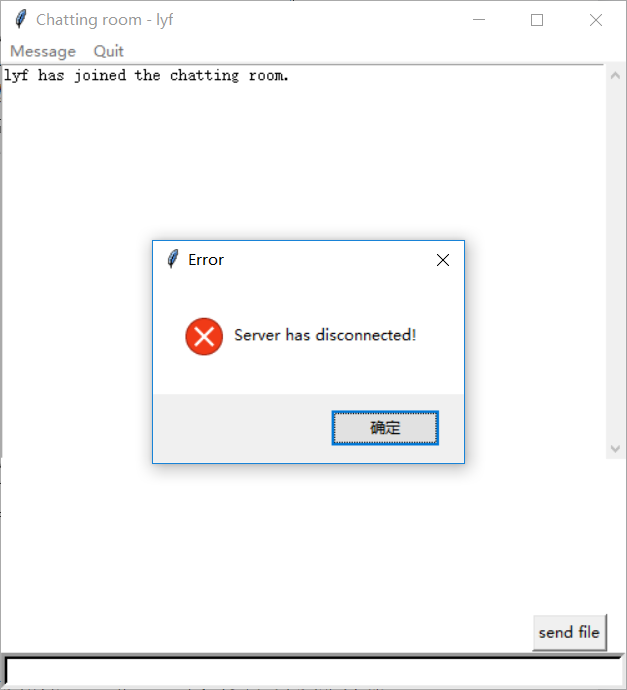




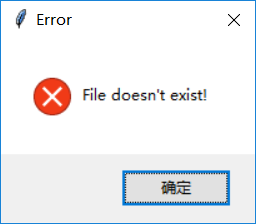
此时客户端再点击加入，结果如下。



关闭服务器，客户端结果如下。

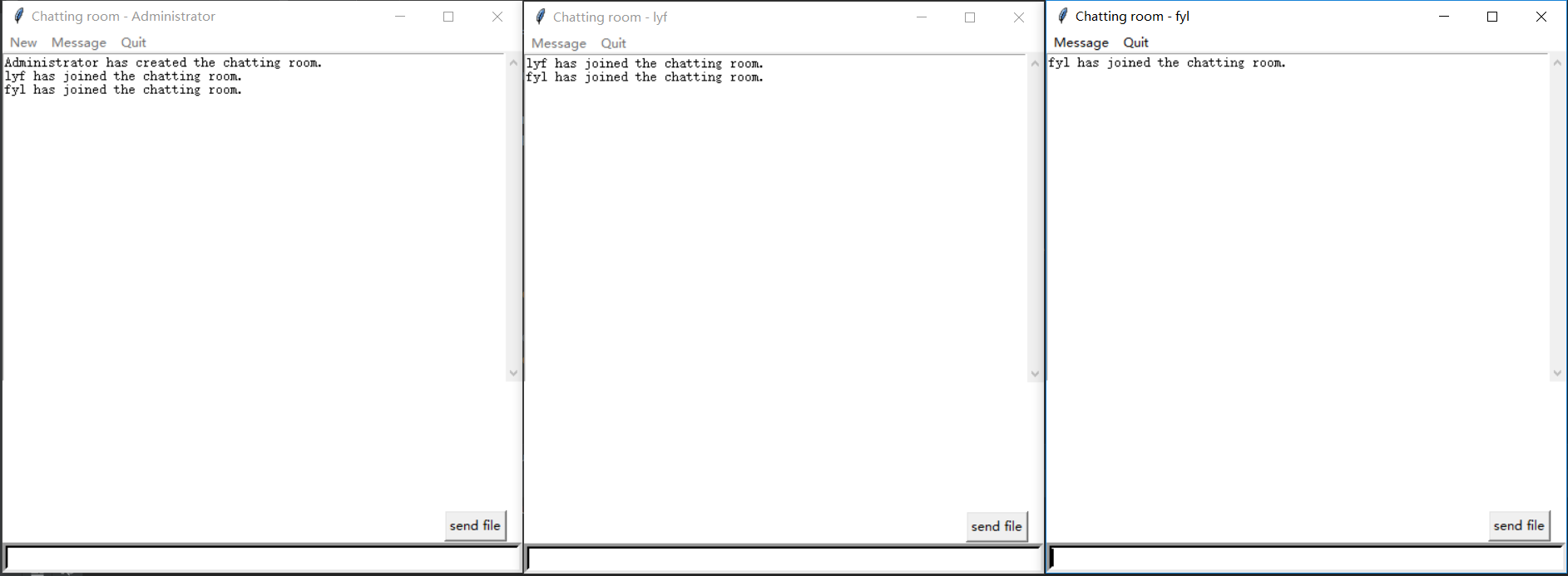


点击send file，然后取消，结果如下。

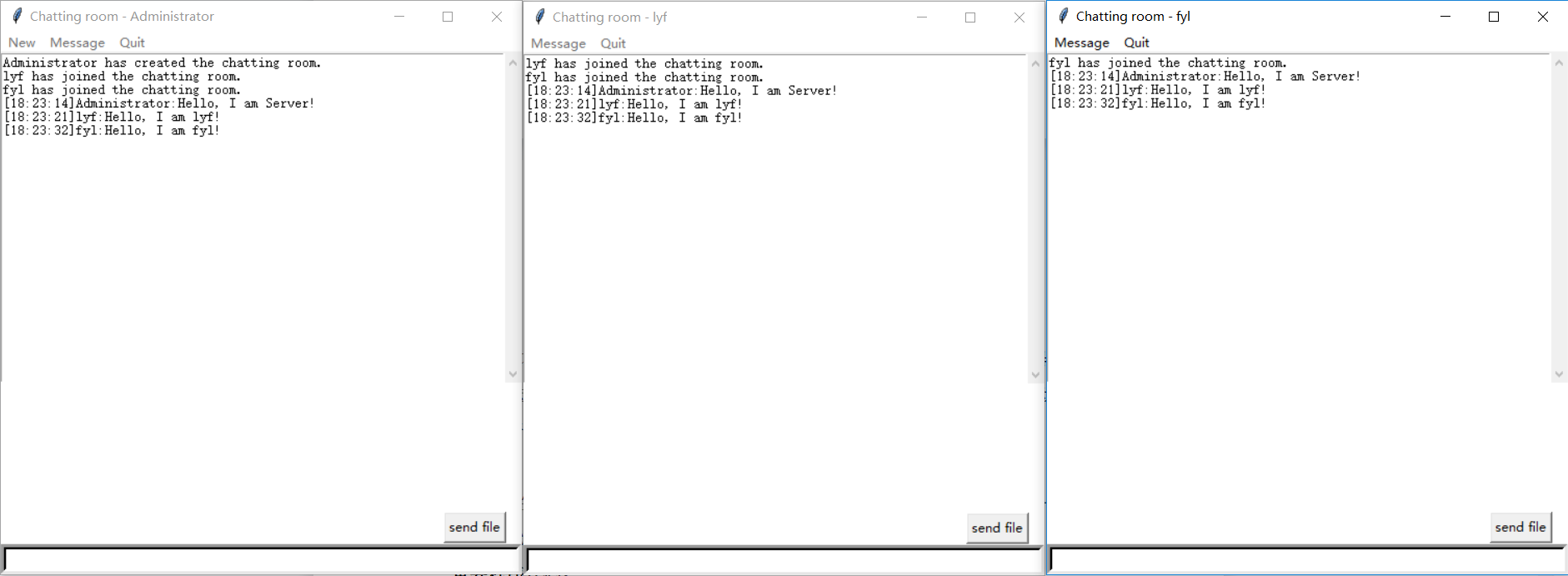


1.3 测试多人聊天功能

启动服务器创建聊天室，开启两个客户端，一个为lyf，另一个为fyl。



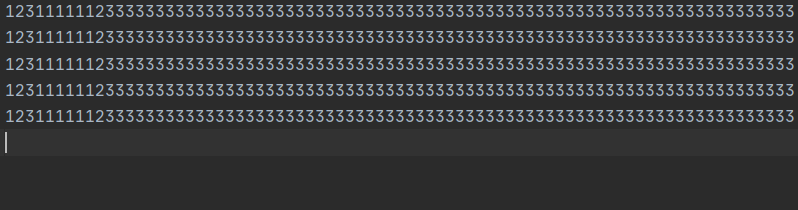
服务端中输入”Hello, I am Server!”，客户端1中输入”Hello, I am lyf!”，客户端2中输入”Hello, I am fyl!”结果如下。



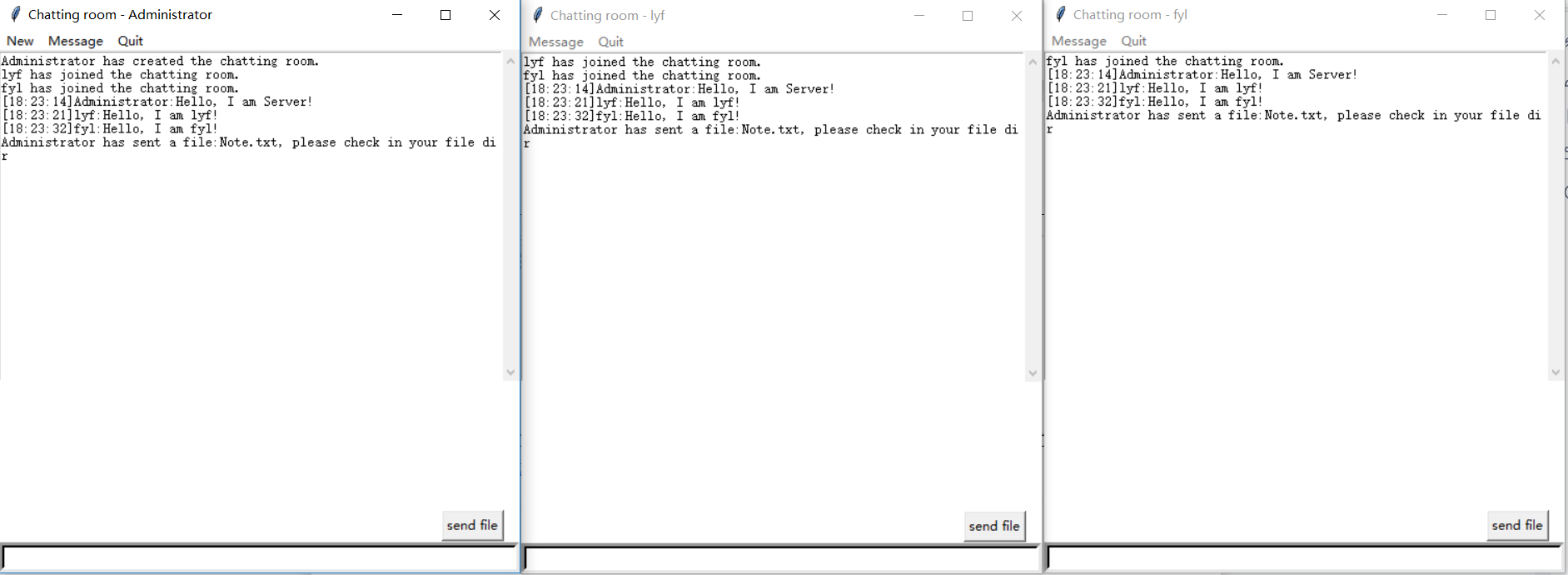
1.4 测试文件传输功能

1.4.1 服务端发送txt文件

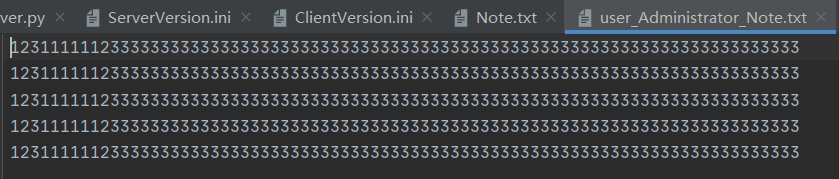
服务器中点击send file按钮，选择文件Note.txt，内容如下。



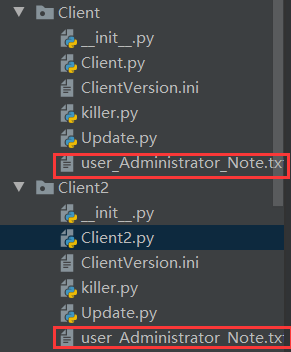
点击确定，结果如下。



都出现了提示消息。然后在客户端1和客户端2的当前路径下多出了一个user\_Administrator\_Note.txt文件。点击打开，内容如下。



可以发现和Note.txt中的文件内容一样。

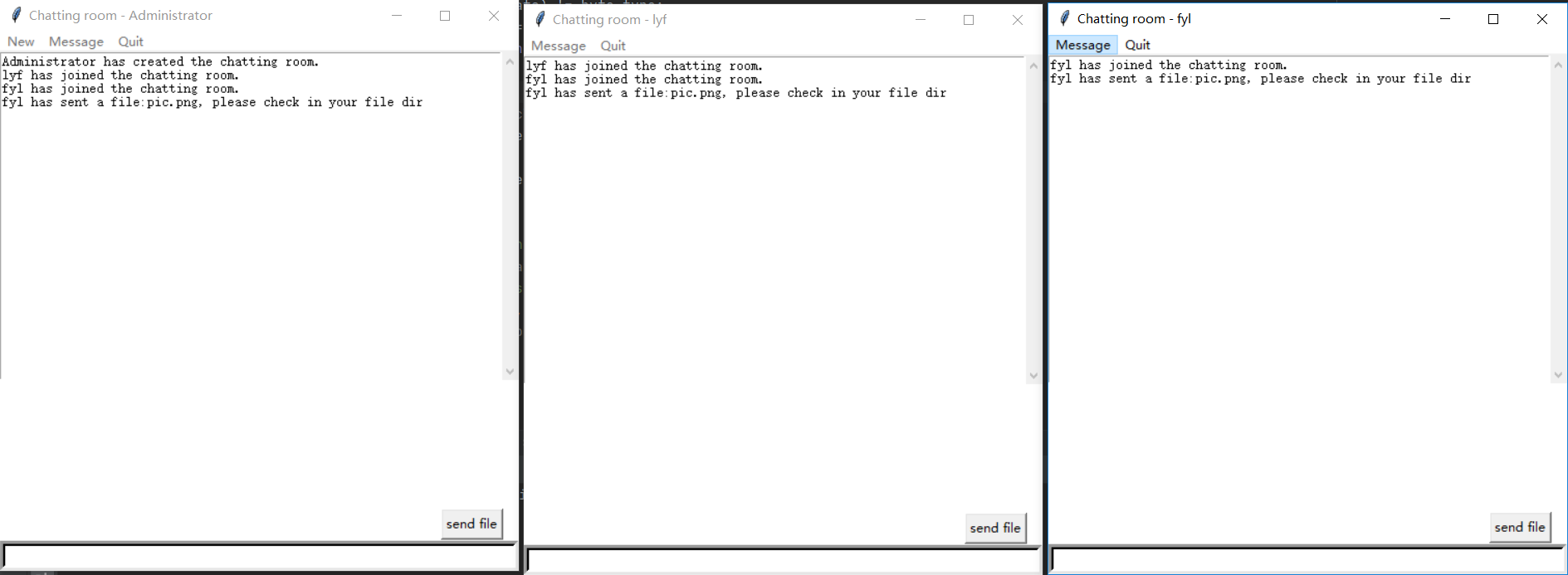


1.4.2 客户端发送图片

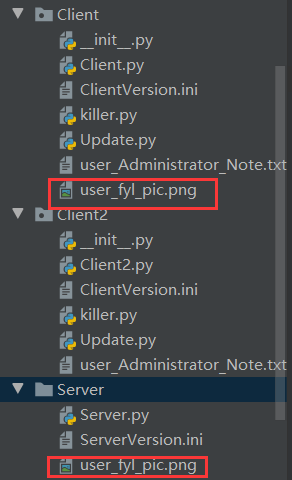
在客户端2（或客户端1）中点击send file按钮，选择pic.png图片，图片如下。

pic

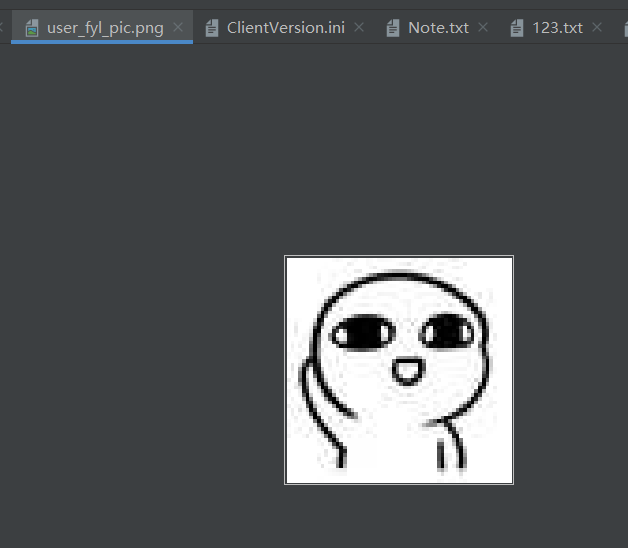
结果如下。



可以在服务器和客户端1中的路径下发现多了user\_fyl\_pic.png文件。

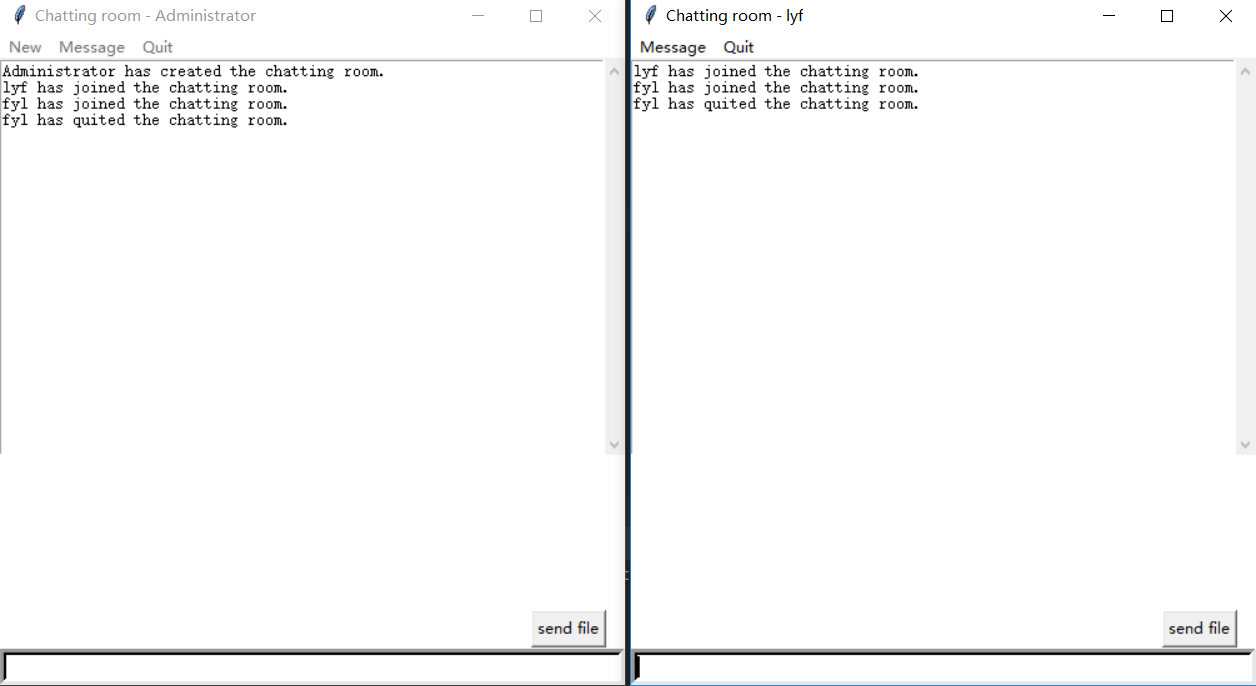


打开图片，结果如下。

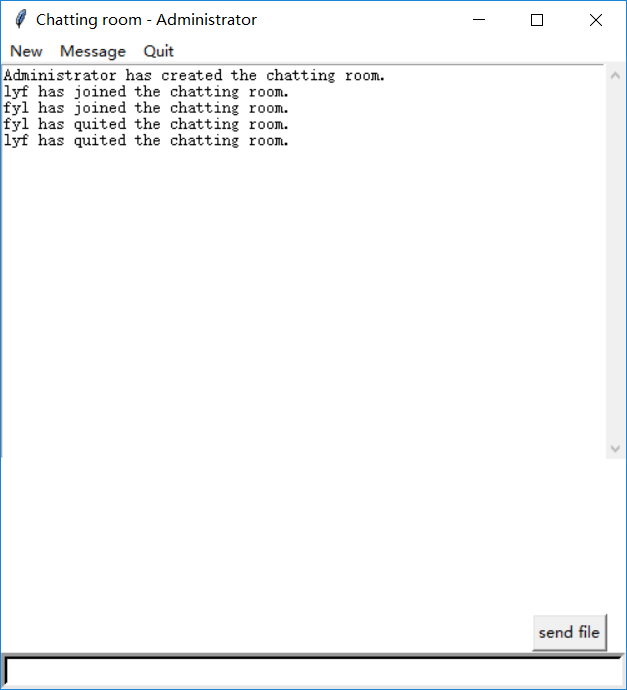


1.4.3 测试客户端退出功能

客户端2退出，结果如下。



客户端1退出，结果如下。



2 实验结论

通过本次实验实现了一个基于SOCKET并且能自动更新的多人聊天室，并且通过tkinter实现了聊天室的可视化界面，使得界面友好。

3 讨论

本次实验完成的C/S架构体系已经算是完善的了，继承了实验一和二的基本功能和特点，同时拓展了新的功能并且弥补了实验一和二存在的一些不足之处。

**重要程序代码：**

1 Server.py

|  |
| --- |
| *'''  Server.py  Using multithreading concurrent blokcing '''* from MultiChattingRoom.Setting import \*   def main\_window():  # get configuration file  file = open("ServerVersion.ini")  version = file.read().split(":")[1]  # main window  root = Tk()  # set in the center  set\_window\_center(root, 340)  # set background colour  root.configure(bg="white")  # set title  root.title("Chatting Room - version:" + version)  # set size  root.resizable(0, 0)  # set create window  pad, en\_host, en\_port = set\_main\_window(root)   # component list  widgets = {  "en\_host": en\_host,  "en\_port": en\_port  }  # adding confirm button  cfm\_button = Button(root, text="New Chatting room", command=lambda: verify\_and\_create(root, widgets))  cfm\_button.grid(rowspan=2, columnspan=2, padx=pad, pady=pad)  # bind events  root.bind("<Return>", lambda event: verify\_and\_create(root, widgets))  # mainloop  root.mainloop()   # verify the button event, checking the input is right or wrong def verify\_and\_create(root, widgets):  host, port = widgets["en\_host"].get(), widgets["en\_port"].get()  # Port may be a string instead of number, you need to convert  try:  port = int(port)  except error:  # there is some characters in port, not just a number  messagebox.showerror("Error", "Port must be number!")  return  # the input of host or port is null  if not host or not port:  messagebox.showerror("Error", "Host or Port can not be null!")  return  # start to create server socket  addr = (host, port)  try:  server = socket(AF\_INET, SOCK\_STREAM)  server.setsockopt(SOL\_SOCKET, SO\_REUSEADDR, 1) # Port reuse  server.bind(addr)  server.listen(5)  except error as msg:  messagebox.showerror("Error", "Can't create socket on " + str(addr) + "!")  print(msg)  return  # no error  else:  '''  generate 2 queues  main\_queue is loop to get the sub process message in the main process  sub\_queue is loop to get the main process message in the sub process  main process receive data  sub process send data  '''  main\_queue, sub\_queue = Queue(), Queue()   # generate sub process  sub\_process = Process(target=deal\_events, args=(server, sub\_queue, main\_queue))  sub\_process.daemon = True  sub\_process.start()   # Create gui of chatting room  create\_chatroom(root, sub\_queue, main\_queue)   # Create chatting room def create\_chatroom(r, sub\_queue, main\_queue):  # interface  r.destroy()  root = Tk()  # center  set\_window\_center(root, 500, 500)  # minimize  root.minsize(350, 350)  # menu box  menu = Menu(root)  menu.add\_command(label="New")  menu.add\_command(label="Message")  menu.add\_command(label="Quit", command=root.destroy)  root.config(menu=menu)  # text box  text = ScrolledText(root)  text.pack(fill=BOTH)  # send file button  file\_button = Button(root, text="send file", command=lambda: send\_file(sub\_queue, text))  file\_button.place(x=425, y=440)  # input box  inpu = Entry(root, bg='white', bd=5)  inpu.pack(fill=BOTH, side=BOTTOM)  inpu.focus\_set()  # bind enter events to send msg  root.bind("<Return>", lambda event: send\_msg(inpu, sub\_queue, text))  # title  root.title("Chatting room - Administrator")  # background  root.configure(bg="white")  # main loop  root.after(1000, recv\_msg, root, sub\_queue, main\_queue, text)  note = "Administrator has created the chatting room.\n"  text.insert(END, note)   # receive message def recv\_msg(root, sub\_queue, main\_queue, text):  if not main\_queue.empty():  data = main\_queue.get()  if data == "/f".encode():  recv\_file(sub\_queue, main\_queue, text)  else:  text.insert(END, data)  root.after(1000, recv\_msg, root, sub\_queue, main\_queue, text)   # receive file def recv\_file(sub\_queue, main\_queue, text):  user = main\_queue.get()  user = user.decode()  recv\_name = main\_queue.get() # Receive client pack file name, include lots of '\x00'  if recv\_name:  filename, filesize = struct.unpack('128sq', recv\_name) # Unpack file  filename = filename.decode()  file = filename.strip('\x00') # Remove '\x00', get real file name  new\_filename = os.path.join('./', "user\_" + user + "\_" + file) # Store file at Server's current path  # Start receiving less than 1024 bytes data from client,  # if file size is too large, maybe receive many times  # so we use 'while' loop to receive file.  recv\_size = 0  fp = open(new\_filename, 'wb') # write  while not recv\_size == filesize:  if filesize - recv\_size > len\_of\_data: # file size large than 1024 bytes  data = main\_queue.get()  recv\_size += len(data)  else:  data = main\_queue.get()  recv\_size = filesize  # print(f"Data is {data}")  fp.write(data) # write file data  fp.close()  client\_msg = user + " has sent a file:" + file + ", please check in your file dir\n"  text.insert(END, client\_msg)  sub\_queue.put(client\_msg)   # send message def send\_msg(inpu, sub\_queue, text):  time = ":".join(ctime().split()[3].split(":"))  data = "[" + time + "]" + "Administrator:" + inpu.get() + "\n"  sub\_queue.put(data)  text.insert(END, data)  inpu.delete(0, END)   # send file def send\_file(sub\_queue, text):  usr = "Administrator"  file\_path = filedialog.askopenfilename(title="Please choose a file")  file\_name = file\_path.split("/")[-1]  if os.path.exists(file\_path):  # pack file name and send to server  fhead = struct.pack(b'128sq', bytes(os.path.basename(file\_path), encoding='utf-8'),  os.stat(file\_path).st\_size)  # put symbol, client name, file pack, file content  sub\_queue.put("/f") # a symbol of file  sub\_queue.put(usr.encode())  sub\_queue.put(fhead)  # Start sending less than 1024 bytes data to server,  # if file size is too large, maybe send many times  # so we use 'while' loop to send file.  fp = open(file\_path, 'rb') # read  while True:  data = fp.read(len\_of\_data) # read 1024 bytes  if not data:  break  sub\_queue.put(data)  admin\_msg = "Administrator has sent a file:" + file\_name + ", please check in your file dir\n"  sub\_queue.put(admin\_msg)  text.insert(END, admin\_msg)  else:  messagebox.showerror("Error", "File doesn't exist!")   # deal receive and send events def deal\_events(server, sub\_queue, main\_queue):  sock\_list = [server]  x\_list = []  y\_list = []  # data type  byte\_type = type(bytes(1))  while True:  # server send data  if not sub\_queue.empty():  data = sub\_queue.get()  for sock in sock\_list:  if sock is not server:  if type(data) == byte\_type:  sock.send(data)  else:  sock.send(data.encode())   ss, xs, yx = select(sock\_list, x\_list, y\_list, 1)  for s in ss:  # server accept connection  if s == server:  conn, addr = s.accept()  sock\_list.append(conn)  else:  try:  # server receive data  data = s.recv(len\_of\_data)  if type(data) != byte\_type: # not a bytes type, need encode  data = data.decode()  except error:  sock\_list.remove(s)  # server resend to other client  else:  main\_queue.put(data)  for client in sock\_list:  if client is not server and client is not s:  if type(data) == byte\_type:  client.send(data)  else:  client.send(data.encode())   if \_\_name\_\_ == "\_\_main\_\_":  # detect configuration file  path = "ServerVersion.ini"  missing\_conf(path, "Server")  main\_window() |

2 Client.py

|  |
| --- |
| *'''  Client.py '''* from MultiChattingRoom.Setting import \*   def main\_window():  # get configuration file  # if there is not conf file, tips error  file = open("ClientVersion.ini", "r", encoding="utf-8")  version = file.read().split(":")[1]  root = Tk()  set\_window\_center(root, 340, 200)  # set background colour  root.configure(bg="white")  # set title  root.title("Chatting Room - version:" + version)  # set create window  pad, en\_host, en\_port = set\_main\_window(root)  Label(root, text="Username:").grid(row=2, column=0, padx=pad, pady=pad)  en\_user = Entry(root)  en\_user.grid(row=2, column=1, padx=pad, pady=pad)  en\_user.focus\_set()   # component list  widgets = {  "en\_host": en\_host,  "en\_port": en\_port,  "en\_user": en\_user  }  # adding confirm button  cfm\_button = Button(root, text="Join the chatting room", command=lambda: verify\_and\_create(root, widgets))  cfm\_button.grid(rowspan=2, columnspan=2, padx=pad, pady=pad)  # bind events  root.bind("<Return>", lambda event: verify\_and\_create(root, widgets))  # mainloop  root.mainloop()   # verify the button event, checking the input is right or wrong def verify\_and\_create(root, widgets):  host, port, user = widgets["en\_host"].get(), widgets["en\_port"].get(), widgets["en\_user"].get()  # Port may be a string instead of number, you need to convert  try:  port = int(port)  except error:  # there is some characters in port, not just a number  messagebox.showerror("Error", "Port must be number!")  return  # the input of host or port or user is null  if not host or not port or not user:  messagebox.showerror("Error", "Host or Port or Username can not be null!")  return  # start to create server socket  addr = (host, port)  try:  client = socket(AF\_INET, SOCK\_STREAM)  client.connect(addr)  except error as msg:  messagebox.showerror("Error", "Can't create socket on " + str(addr) + "!")  print(msg)  return  else:  '''  generate 2 queues  main\_queue is loop to get the sub process message in the main process  sub\_queue is loop to get the main process message in the sub process  main process receive data  sub process send data  '''  main\_queue, sub\_queue = Queue(), Queue()   # send message of successful connection  join\_msg = str(user) + " has joined the chatting room.\n"  client.send(join\_msg.encode())   # generate sub process  sub\_process = Process(target=deal\_events, args=(client, sub\_queue, main\_queue))  sub\_process.daemon = True  sub\_process.start()   # Create gui of chatting room  create\_chatroom(root, sub\_queue, main\_queue, user, join\_msg)   # Create chatting room def create\_chatroom(r, sub\_queue, main\_queue, user, join\_msg):  # interface  r.destroy()  root = Tk()  # center  set\_window\_center(root, 500, 500)  # minimize  root.minsize(350, 350)  # menu box  menu = Menu(root)  menu.add\_command(label="Message")  menu.add\_command(label="Quit", command=root.destroy)  root.config(menu=menu)  # text box  text = ScrolledText(root)  text.pack(fill=BOTH)  text.insert(END, join\_msg)  # input box  inpu = Entry(root, bg='white', bd=5)  inpu.pack(fill=BOTH, side=BOTTOM)  inpu.focus\_set()  # send file button  file\_button = Button(root, text="send file", command=lambda: send\_file(sub\_queue, user))  file\_button.place(x=425, y=440)  # bind events  root.bind("<Return>", lambda event: send\_msg(inpu, sub\_queue, text, user))  # title  root.title("Chatting room - " + str(user))  # background  root.configure(bg="white")  # set quit protocol  root.protocol("WM\_DELETE\_WINDOW", lambda: quit\_room(root, sub\_queue, user))  # main loop  root.after(1000, recv\_msg, root, main\_queue, text)   def quit\_room(root, sub\_queue, user):  data = user + " has quited the chatting room.\n"  sub\_queue.put(data)  sleep(1)  root.destroy()  sys.exit(0)   # receive message def recv\_msg(root, main\_queue, text):  if not main\_queue.empty():  data = main\_queue.get()  # Server closed  if data == 404:  messagebox.showerror("Error", "Server has disconnected!")  sys.exit(0)  # receive file  elif data == "/f".encode():  recv\_file(main\_queue)  else:  text.insert(END, data)  root.after(1000, recv\_msg, root, main\_queue, text)   def recv\_file(main\_queue):  user = main\_queue.get()  user = user.decode()  recv\_name = main\_queue.get() # Receive client pack file name, include lots of '\x00'  if recv\_name:  filename, filesize = struct.unpack('128sq', recv\_name) # Unpack file  filename = filename.decode()  file = filename.strip('\x00') # Remove '\x00', get real file name  new\_filename = os.path.join('./', "user\_" + user + "\_" + file) # Store file at Server's current path  # Start receiving less than 1024 bytes data from client,  # if file size is too large, maybe receive many times  # so we use 'while' loop to receive file.  recv\_size = 0  fp = open(new\_filename, 'wb') # write  while not recv\_size == filesize:  if filesize - recv\_size > len\_of\_data: # file size large than 1024 bytes  data = main\_queue.get()  recv\_size += len(data)  else:  data = main\_queue.get()  recv\_size = filesize  # print(f"Data is {data}")  fp.write(data) # write file data  fp.close()  else:  messagebox.showerror("Error", "File doesn't exist!")   # send message def send\_msg(inpu, sub\_queue, text, user):  time = ":".join(ctime().split()[3].split(":"))  data = "[" + time + "]" + user + ":" + inpu.get() + "\n"  text.insert(END, data)  sub\_queue.put(data)  inpu.delete(0, END)   def send\_file(sub\_queue, user):  file\_path = filedialog.askopenfilename(title="Please choose a file")  if os.path.exists(file\_path):  # pack file name and send to server  fhead = struct.pack(b'128sq', bytes(os.path.basename(file\_path), encoding='utf-8'),  os.stat(file\_path).st\_size)  # put symbol, client name, file pack, file content  sub\_queue.put("/f") # a symbol of file  sub\_queue.put(user.encode())  sub\_queue.put(fhead)  # Start sending less than 1024 bytes data to server,  # if file size is too large, maybe send many times  # so we use 'while' loop to send file.  fp = open(file\_path, 'rb') # read  while True:  data = fp.read(len\_of\_data) # read 1024 bytes  if not data:  break  sub\_queue.put(data)  else:  messagebox.showerror("Error", "File doesn't exist!")   # deal receive and send events def deal\_events(client, sub\_queue, main\_queue):  # set non-blocking I/O  client.setblocking(0)  byte\_type = type(bytes(1))  while True:  if not sub\_queue.empty():  data = sub\_queue.get()  if data:  if type(data) == byte\_type:  client.send(data)  else:  client.send(data.encode())  try:  data = client.recv(len\_of\_data)  if type(data) != byte\_type:  data = data.decode()  except BlockingIOError:  continue  except error:  # server closed  main\_queue.put(404)  else:  main\_queue.put(data)   if \_\_name\_\_ == "\_\_main\_\_":  # detect configuration file  path = "ClientVersion.ini"  missing\_conf(path, "Client")  # detect version before running  os.system("python Update.py")  main\_window() |

其余代码见源代码。

**指导教师评语：（空白，半页A4）**