实验报告

|  |  |
| --- | --- |
| 实验名称：基于Python Socket的聊天程序 | |
| 实验台号：2 | 实验时间：2019年11月29日 |
| 实验小组：第2组  成员及本次实验分工:於文卓 | |
| 实验目的：  1.进一步学习TCP协议的工作原理  2.学习SOCKET编程的基本方法  3.使用Python编程语言的socket包，实现多个服务器客户端之间一对多的通信  4.使用Tkinter包，实现一个聊天应用程序界面 | |
| 实验环境说明：  笔记本电脑一台(Win10系统)，Python（3.6） | |
| 实验过程、步骤（可另附页、使用网络拓扑图等辅助说明）及结果： 1.引言 在这个实验中，我使用Python和Python中的Socket包，实现了一个基于TCP的聊天小程序，在实现过程中使用了多线程技术，使得一个服务端能同时和多个客户端交流，客户端之间也能同时接受和发送消息。此外还是用Tkinter库，实现了一个GUI界面，相较于之前的控制台程序，GUI界面的程序使用起来更加方便。以下是实现过程介绍。 2.实验步骤  2.1 服务端实现 1）导入所需要的库  AF\_INET 采用ipv4的tcp或udp通信机制，我的小程序是基于ipv4实现的。  SOCK\_STREAM代表tcp通信，我顺便了解了微信的文本交流使用的是TCP，而QQ更多的是使用UDP，我的小程序基于的是TCP的协议。   |  | | --- | | #!/usr/bin/env python3  from socket import AF\_INET, socket, SOCK\_STREAM  #导入多线程库Thread from threading import Thread |   2）设定一些常量  定义了服务端的一些基本配置，比如主机地址和端口号，初始化了socket对象，并把主机地址和端口号绑定到了这个socket上。另外还设置了两个字典，client字典用来存放每个客户的名字，addresses字典用来保存每个客户端的地址信息。   |  | | --- | | clients = {} addresses = {}  HOST = '' PORT = 33000 BUFSIZ = 1024 ADDR = (HOST, PORT) SERVER = socket(AF\_INET, SOCK\_STREAM) SERVER.bind(ADDR) |   3）等待连接  使用socket中的accept函数，服务器等待客户端的连接。这个函数有两个返回值，一个是用来通信的socket，一个是客户端的地址。连接成功后，服务器会告诉客户端，可以在输入框中输入信息，提供自己的名字。   |  | | --- | | def accept\_incoming\_connections():  while True:  client, client\_address = SERVER.accept()  print("%s:%s has connected." % client\_address)  client.send(bytes("Now type your name and press enter!", "utf8"))  addresses[client] = client\_address  Thread(target=handle\_client, args=(client,)).start() |   4）处理客户端  这个函数用来处理客户端的相关操作。比如使用recv函数接受来自客户端的消息。用send函数对客户端发送消息。如果客户端发送了{quit}这样的消息，那么说明客户端退出了，需要关闭相关内容，包括用来通信的socket，要删除字典中对应客户端的信息等，并广播说明，这个客户端退出了。   |  | | --- | | def handle\_client(client):  name = client.recv(BUFSIZ).decode("utf8")  welcome = 'Welcome %s! If you ever want to quit, type {quit} to exit.' % name  client.send(bytes(welcome, "utf8"))  msg = "%s has joined the chat!" % name  broadcast(bytes(msg, "utf8"))  clients[client] = name  while True:  msg = client.recv(BUFSIZ)  if msg != bytes("{quit}", "utf8"):  broadcast(msg, name+": ")  else:  client.send(bytes("{quit}", "utf8"))  client.close()  del clients[client]  broadcast(bytes("%s has left the chat." % name, "utf8"))  break |   5）广播  广播顾名思义就是要对所有的客户端都发送同样的信息，这里遍历字典中保存的所有在线的客户端，发送同样的信息。在很多及时通讯软件中，有群组的概念。最简单的群消息处理就是讲一个客户端的消息传到中心服务器后，中心服务器将这个消息发给这个群里的每个人。   |  | | --- | | def broadcast(msg, prefix=""):  for sock in clients:  sock.send(bytes(prefix, "utf8")+msg) |   6）开启服务端  我的服务器实现了基本操作后，就可以开启等待客户端的访问了。这里listen表示服务器最多能被多少个客户端访问。使用Thread可以开启多线程，即当有新的客户端对这个服务器进行访问时，就开启一个线程处理这个客户端。   |  | | --- | | if \_\_name\_\_ == "\_\_main\_\_":  SERVER.listen(5)  print("Waiting for connection...")  ACCEPT\_THREAD = Thread(target=accept\_incoming\_connections)  ACCEPT\_THREAD.start()  ACCEPT\_THREAD.join()  SERVER.close() |  2.2 客户端实现 1）导入所需要的库  与服务端的操作类似，这里导入课tkinter，python自带的一个GUI图形界面包   |  | | --- | | #!/usr/bin/env python3  from socket import AF\_INET, socket, SOCK\_STREAM from threading import Thread import tkinter |   2）接收消息  客户端需要从服务端接受消息，使用recv函数，获得来自服务端的消息后插入到消息列表中，这个消息列表的信息会展示在图形界面的消息框中。   |  | | --- | | def receive():  while True:  try:  msg = client\_socket.recv(BUFSIZ).decode("utf8")  msg\_list.insert(tkinter.END, msg)  except OSError: # Possibly client has left the chat.  break |   3）发送消息  客户端通过图形界面，通过send函数将消息发送给服务端。msg消息是通过my\_msg.get()函数得到的，这里的my\_msg是一个消息输入框。如果发送的是{quit}，需要退出这个客户端。   |  | | --- | | def send(event=None):  msg = my\_msg.get()  my\_msg.set("") # Clears input field.  client\_socket.send(bytes(msg, "utf8"))  if msg == "{quit}":  client\_socket.close()  top.quit() |   4）Tkinter图形界面相关代码  这里设置了窗口的名字，大小。设置了相关组件包括Scrollbar，Listbox等，还有一些处理细节比如在输入框中敲回车，也可以激活发送命令，关比客户端窗口，会触发close命令等。   |  | | --- | | top = tkinter.Tk()  top.title("Chatter")  messages\_frame = tkinter.Frame(top)  my\_msg = tkinter.StringVar() # For the messages to be sent.  my\_msg.set("Type your messages here.")  scrollbar = tkinter.Scrollbar(messages\_frame)  msg\_list=tkinter.Listbox(messages\_frame,height=15,width=50,yscrollcommand=scrollbar.set)  scrollbar.pack(side=tkinter.RIGHT, fill=tkinter.Y)  msg\_list.pack(side=tkinter.LEFT, fill=tkinter.BOTH)  msg\_list.pack()  messages\_frame.pack()  entry\_field = tkinter.Entry(top, textvariable=my\_msg)  entry\_field.bind("<Return>", send)  entry\_field.pack()  send\_button = tkinter.Button(top, text="Send", command=send)  send\_button.pack() |   5）设置客户端信息  设置客户端的端口、主机号等信息，初始化客户端的socket，开启多线程，开启tkinter。   |  | | --- | | HOST = input('Enter host: ')  PORT = input('Enter port: ')  if not PORT:  PORT = 33000  else:  PORT = int(PORT)  BUFSIZ = 1024  ADDR = (HOST, PORT)  client\_socket = socket(AF\_INET, SOCK\_STREAM)  client\_socket.connect(ADDR)  receive\_thread = Thread(target=receive)  receive\_thread.start()  tkinter.mainloop() |  3.实验结果演示 这里我们开启了三个客户端一个服务端，服务端端口号为8888   每个client创建后输入自己的名字，会提示welcome XXX，当有新的client加入时，会提示xxx joinedt the chat。 用户可以发送消息，发送的消息会被显示在窗口中央。 | |
| 实验总结（遇到的问题及解决办法、体会）：  在这个实验中，实现了一个基于TCP的聊天小程序，在实现过程中使用了多线程技术，使得一个服务端能同时和多个客户端交流，客户端之间也能同时接受和发送消息。在实验中进一步了解了TPC协议在python中实现的流程，了解了即时通讯软件在实现过程中最基本的思想，比如群组消息如何实现。  在实现过程中使用了Tkinter这个库，这个库曾经接触不多，重新学习了下了解了GUI界面制作的基本流程，相较于控制台的程序，GUI更加直观使用起来也更加友好。  时间匆忙，代码还有很多可以改进的地方，比如可以用更加美观的GUI库来实现这个界面，比如Kivy等，比如实现Client之间P2P的交流等。我将会在后续空余时间改进代码。 | |
| 器材、工具领用及归还负责人：於文卓 | 实验记录人：於文卓 |
| 实验执笔人：於文卓 | 报告协助人：於文卓 |
| 小组成员签名：於文卓 | |
| 验收人： | 成绩评定： |