

课程实验报告

课 程 名 称： 计算机网络

实验项目名称： 网络基础编程实验

专 业 班 级： 计科1902

姓 名： 樊一鸣

学 号： 2019028010212

指 导 教 师： 罗辉章

信息科学与工程学院

## 一、实验题目

　　通过本实验，学习采用Socket（套接字）设计简单的网络数据收发程序，理解应用数据包是如何通过传输层进行传送的。

## 二、实验内容

实验内容分为四个部分:

1. 采用TCP进行数据发送的简单程序
2. 采用UDP进行数据发送的简单程序
3. 多线程\线程池对比
4. 写一个简单的chat程序，并能互传文件，编程语言不限。

## 三、实验原理

Socket（套接字）是一种抽象层，应用程序通过它来发送和接收数据，就像应用程序打开一个文件句柄，将数据读写到稳定的存储器上一样。一个socket允许应用程序添加到网络中，并与处于同一个网络中的其他应用程序进行通信。一台计算机上的应用程序向socket写入的信息能够被另一台计算机上的另一个应用程序读取，反之亦然。

不同类型的socket与不同类型的底层协议族以及同一协议族中的不同协议栈相关联。现在TCP/IP协议族中的主要socket类型为流套接字（sockets sockets）和数据报套接字（datagram sockets）。流套接字将TCP作为其端对端协议（底层使用IP协议），提供了一个可信赖的字节流服务。一个TCP/IP流套接字代表了TCP连接的一端。数据报套接字使用UDP协议（底层同样使用IP协议），提供了一个"尽力而为"（best-effort）的数据报服务，应用程序可以通过它发送最长65500字节的个人信息。一个TCP/IP套接字由一个互联网地址，一个端对端协议（TCP或UDP协议）以及一个端口号唯一确定。

多线程原理：多线程的实现方法有两种一客户一线程和线程池。

每个新线程都会消耗系统资源：创建一个线程将占用CPU周期，而且每个线程都自己的数据结构（如，栈）也要消耗系统内存。另外，当一个线程阻塞（block）时，JVM将保存其状态，选择另外一个线程运行，并在上下文转换（context switch）时恢复阻塞线程的状态。随着线程数的增加，线程将消耗越来越多的系统资源。这将最终导致系统花费更多的时间来处理上下文转换和线程管理，更少的时间来对连接进行服务。那种情况下，加入一个额外的线程实际上可能增加客户端总服务时间。

我们可以通过限制总线程数并重复使用线程来避免这个问题。与为每个连接创建一个新的线程不同，服务器在启动时创建一个由固定数量线程组成的线程池（thread pool）。当一个新的客户端连接请求传入服务器，它将交给线程池中的一个线程处理。当该线程处理完这个客户端后，又返回线程池，并为下一次请求处理做好准备。如果连接请求到达服务器时，线程池中的所有线程都已经被占用，它们则在一个队列中等待，直到有空闲的线程可用。

## 实验步骤

### 采用TCP进行数据发送的简单程序

参考课本代码，使用Python进行编程，其中**客户端程序功能是接受一个字符串，并将其转化为大写的形式返回给用户**。

编程步骤：导入socket库，然后设置好服务器地址

服务器地址格式ADDR(IP,Port)，对于服务器端，我们设置好自己的端口号，并查看IP，就可以知道我们的地址。然后使用 socket.socke建立套接字，然后bind相应的端口，在调用listen进行监听，等待连接。 使用一个while循环，等待连接并将数据变为大写后发回给客户端。

|  |
| --- |
| Func: TCP connect  """  import socket  # 创建socket  serverPort = 12000  serverSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  # 激活socket  serverSocket.bind(('', serverPort))  serverSocket.listen(1) # 监听  print('The server(TCP) is ready to receive!')  while True:  connectionSocket, addr = serverSocket.accept()  sentence = connectionSocket.recv(1024).decode() # 接收1024字节  capitalizeSentence = sentence.upper() # 全部变为大写  connectionSocket.send(capitalizeSentence.encode())  connectionSocket.close() |

客户端代码

思路： 根据服务器的IP和端口号，使用套接字连接服务器后，再发送数据即可收到回复。

|  |
| --- |
| Func:Client of TCP 发送给服务器数据，等待服务器返回  """  import socket  # 1. 创建socket  servername = '10.68.137.99'  serverPort = 12000  ClientSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  # 2. 连接成功  ClientSocket.connect((servername, serverPort))  sentence = input('Input lowercase sentence:')  ClientSocket.send(sentence.encode())  modifiedSentence = ClientSocket.recv(1024)  print('From Server: ', modifiedSentence.decode())  ClientSocket.close() |
| 运行结果： |
| 服务端不断等待数据到来    客户端与服务端创建连接，然后输入数据并得到大写的数据结果 |

### 采用UDP进行数据发送的简单程序

客户端：首先创建socket，然后直接发送数据到对应的地址

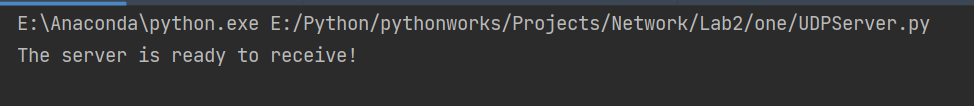
|  |
| --- |
| # -\*- coding: utf-8 -\*-  """  Created on 2022-04-05 19:46  @author: Fan yi ming  Func: UDP Client  """  import socket  # 1. 创建socket  serverPort = 12001  servername = '10.68.137.99'  ClientSocket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  # 2. 不需要连接，直接发送  sentence = input('Input lowercase sentence:')  ClientSocket.sendto(sentence.encode(), (servername, serverPort))  modifiedSentence = ClientSocket.recv(2048)  print('From Server: ', modifiedSentence.decode())  ClientSocket.close() |

服务端：与TCP不同的是，创建socket的时候选项为SOCK\_DGRAM，然后使用recvform可以获得数据。将其变成大写后返回给发送发，这个过程中并没有创建连接。

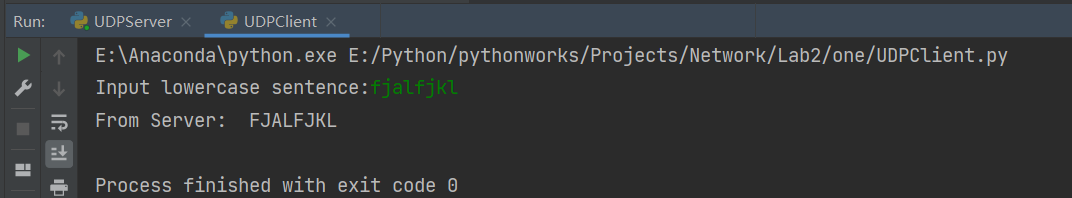
|  |
| --- |
| # -\*- coding: utf-8 -\*-  """  Created on 2022-04-05 19:42  @author: Fan yi ming  Func: UDP server  """  import socket  # 创建socket  serverPort = 12001  serverSocket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  # 激活socket  serverSocket.bind(('', serverPort))  print('The server is ready to receive!')  while True:  message, ClientAddress = serverSocket.recvfrom(2048)  modifiedMessage = message.decode().upper()  serverSocket.sendto(modifiedMessage.encode(), ClientAddress) # 发送给对应端口 |

实验结果：

服务器方运行截图



客户端运行截图



### 多线程\线程池对比

这一部分主要是对服务器端进行更改，我使用多线程是TCP的。

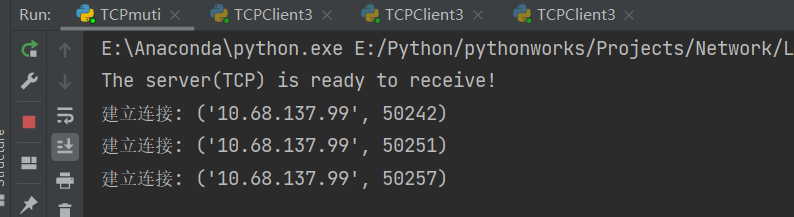
多线程：一客户一线程

实现方式：使用Python的threading库实现

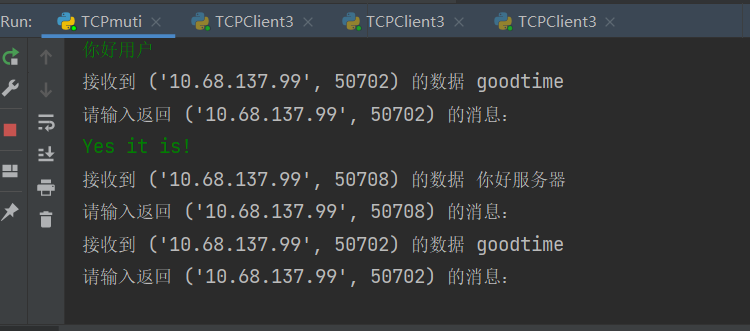
首先引入需要的功能 from threading import Thread, 然后将TCP建立连接并处理这一部分写成函数，每当通过serverSocket.accept()收到一个连接，就新建一个线程去运行他。

|  |
| --- |
| from threading import Thread  import socket  # 定义函数  def connect(new\_socket, clidentAddress):  print('建立连接:', clidentAddress)  while True:  sentence = new\_socket.recv(1024)  if sentence:  print('接收到', clidentAddress, '的数据', sentence.decode('gbk'))  print("请输入返回", clidentAddress, "的消息：")  send\_data = input()  new\_socket.send(send\_data.encode('gbk'))  else:  break  # 关闭连接  new\_socket.close()  print("来自", clidentAddress, "的连接关闭")  # 创建socket  serverPort = 12000  serverSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  # 激活socket  serverSocket.bind(('', serverPort))  serverSocket.listen(1) # 监听  print('The server(TCP) is ready to receive!')  try:  while True:  connectionSocket, addr = serverSocket.accept()  t = Thread(target=connect, args=(connectionSocket, addr))  t.start()  except:  print("something wrong0")  finally:  serverSocket.close() |

运行结果：可以知道建立了开起来三个客户端口，建立了三个连接，连接的IP都是相同的，因为都是同一个电脑，但是每个线程分配了不同的端口号



服务器可以同时收到多个客户端消息并回复



线程池方式：使用Python的ProcessPoolExecutor实现。

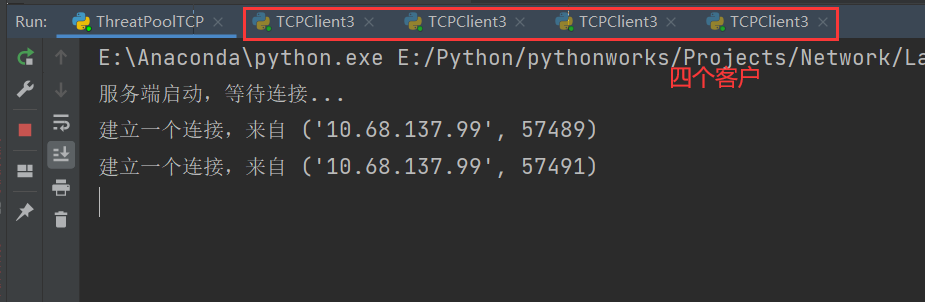
他的使用方式很简单，首先建立一个线程池：pool = ProcessPoolExecutor(2) ，参数表示线程的个数。

然后如果需要使用线程，就使用submit将任务提交。

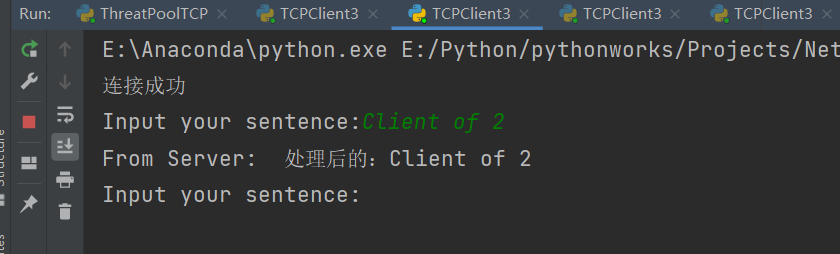
pool.submit(server.handle\_request, ConnectionSocket, clientAddress) #提交任务

|  |
| --- |
| Func: use threat pool  """  import socket  from concurrent.futures import ProcessPoolExecutor  class MyTcpServer:  def \_\_init\_\_(self):  self.address = ('', 7788)  self.server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  self.server.bind(self.address)  self.server.listen(5)  print("服务端启动，等待连接...")  def wait\_accept(self):  new\_socket, clientAddr = self.server.accept()  return new\_socket, clientAddr  def handle\_request(self,new\_socket,clientAddr):  print("建立一个连接，来自", clientAddr)  while True:  try:  # 接收对方发送过来的数据  recv\_data = new\_socket.recv(1024) # 接收1024个字节  if recv\_data:  print('接收到', clientAddr, '的数据为:', recv\_data.decode('gbk'))  new\_socket.send(recv\_data.upper())  else:  break  except:  print("error!")  break  new\_socket.close()  print("来自", clientAddr, "的连接关闭")  if \_\_name\_\_ == '\_\_main\_\_':  server = MyTcpServer()  pool = ProcessPoolExecutor(2) # 创建一个2个线程的线程池  while True:  ConnectionSocket, clientAddress = server.wait\_accept()  pool.submit(server.handle\_request, ConnectionSocket, clientAddress) #提交任务 |

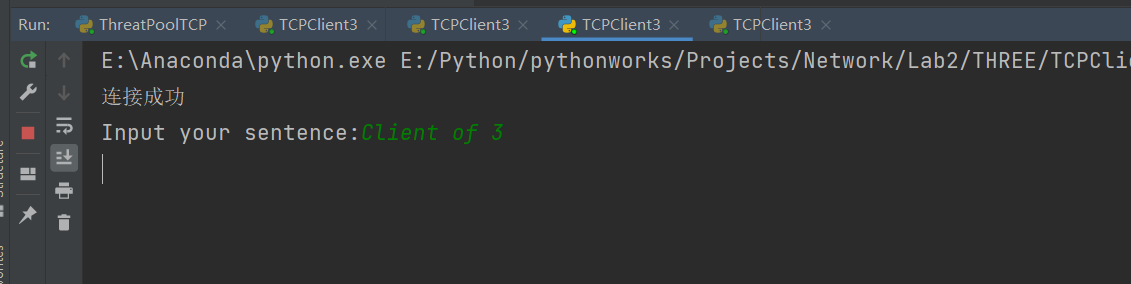
运行结果：建立一个服务器和4个客户端，但是因为线程池中只有两个线程，因此只建立了两个连接



在第一个和第二个线程中输入信息，可以得到反馈信息



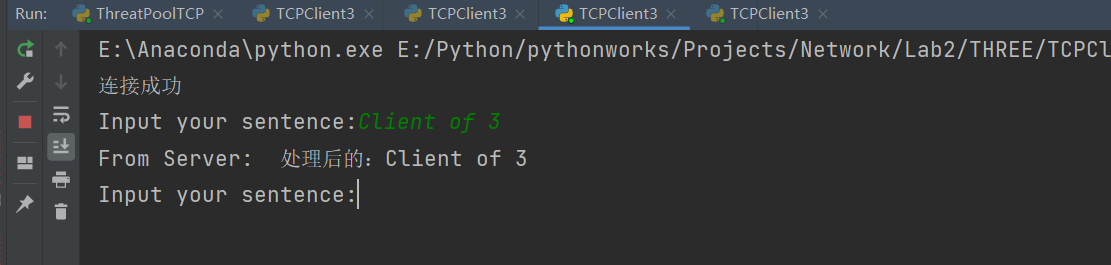
在第三个线程中输入信息，没有结果返回



如果关闭前两个客户端：则对第三第四个客户端建立了连接



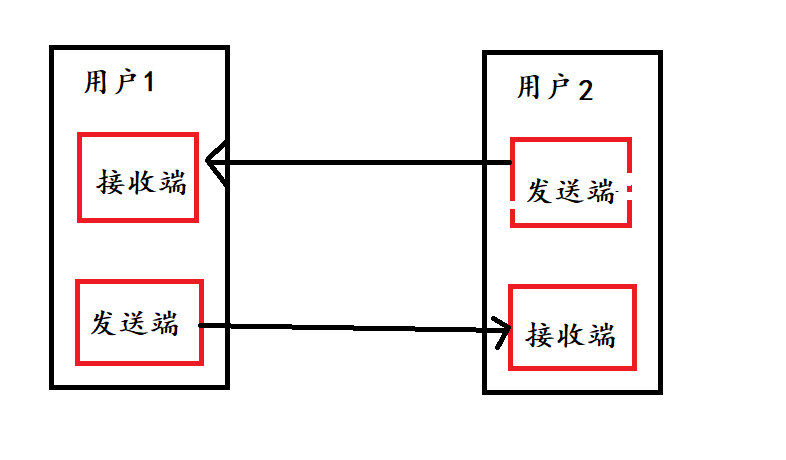
这时第三个客户端输入数据也能得到相应反馈。



### 聊天程序实现（可以传输文件） step1: 实现文字聊天

需求：客户可以找到指定的人进行文字聊天，同时可以及时收到对方的消息。

实现方式：为每个用户建立一个接收信息的线程（类似服务器）同时建立一个发送信息的线程，两个线程实现收发。

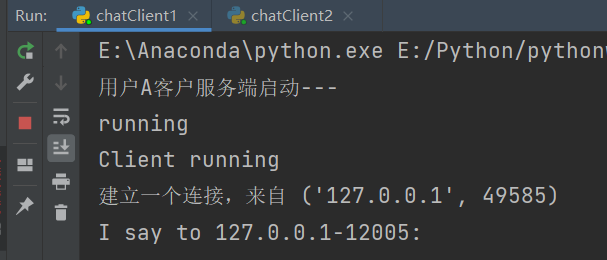


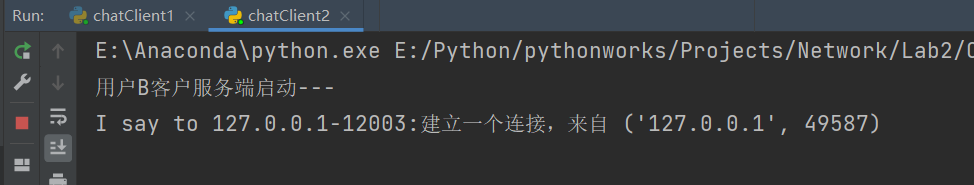
代码： 实现了两个类，分别是服务器端功能（接收对方信息）和客户端功能（接收缓冲区输入并发送）。

对于两个需要相互通信的用户，首先建立起双方的接收端线程（服务器），等待5s后建立起接收端线程，以确保双方的服务器已经建立好。然后接收线程接收到数据并打印出来，发送线程将发送数据给对方。

|  |
| --- |
| # -\*- coding: utf-8 -\*-  """  Created on 2022-04-05 21:35  @author: Fan yi ming  Func: 初始版本实现聊天127.0.0.1-12003  """  from threading import Thread  import socket  import os  import time  class ServerPart:  def \_\_init\_\_(self, Addr, name):  self.Addr = Addr  self.server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  self.server.bind(self.Addr)  self.server.listen(5)  print(name + "客户服务端启动---")  def wait\_accept(self):  new\_socket, clientAddr = self.server.accept()  return new\_socket, clientAddr  def handle\_request(self, new\_socket, clientAddr):  print("建立一个连接，来自", clientAddr)  while True:  try:  # 接收对方发送过来的数据  recv\_data = new\_socket.recv(1024).decode('gbk') # 接收1024个字节  if recv\_data:  print(clientAddr, '的信息:', recv\_data)  else:  break  except:  print("error!")  break  new\_socket.close()  print("来自", clientAddr, "的连接关闭")  def run(self):  print('running')  while True:  ConnectionSocket, clientAddress = self.wait\_accept()  t = Thread(target=self.handle\_request, args=(ConnectionSocket, clientAddress))  t.start()  self.server.close()  class ClientPart:  def \_\_init\_\_(self):  self.serverName = '127.0.0.1'  self.serverPort = 12005  self.ClientSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  def connect(self, Addr=None):  if not Addr:  Addr = (self.serverName, self.serverPort)  time.sleep(10) # 等待三秒  self.ClientSocket.connect(Addr)  while True:  sentence = input('I say to '+Addr[0]+'-'+str(Addr[1])+":")  if sentence == 'q': # 退出  break  self.ClientSocket.send(sentence.encode('gbk'))  self.ClientSocket.close()  def run(self):  t = Thread(target=self.connect)  t.run()  def main():  # 运行服务端  Addr = ('127.0.0.1', 12003) # 服务器地址  name = '用户A'  server = ServerPart(Addr, name)  Thread(target=server.run).start()  # 运行客户端  print('Client running')  client = ClientPart()  client.run()  if \_\_name\_\_ == '\_\_main\_\_':  main() |

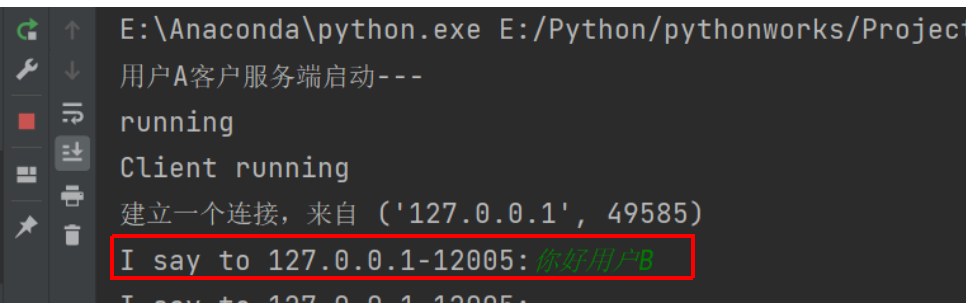
运行截图：模拟用户A和用户B相互通信，双方先建立好连接



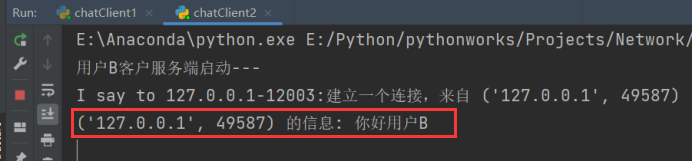


双方通信：用户A发送“你好用户B”，用户B发送“你好用户A”

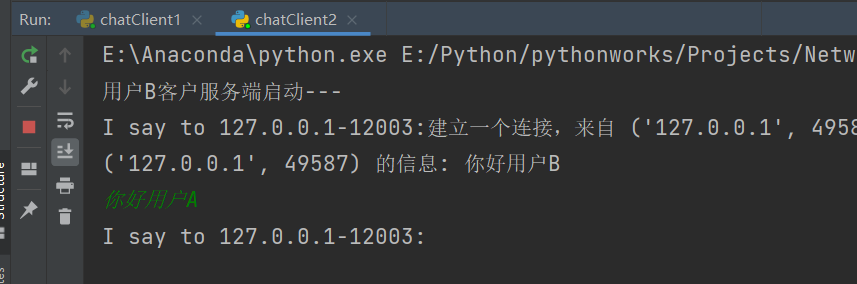
用户A端截图



用户B端截图：收到了相应信息



然后B再给A发送信息，A同样收到





Step2：文件传输功能实现：

通过查阅资料[https://zhuanlan.zhihu.com/p/144361212](https://zhuanlan.zhihu.com/p/144361212" \t "_blank) 文件传输可以通过分成文件发送和文件接收两个部分来实现。

**文件发送：**文件传送需要文件路径和传送数据的套接字（TCP连接）

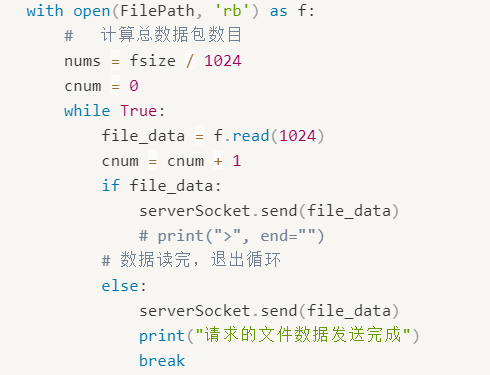
①判断是否存在此文件



②获取文件信息，将文件信息传送给对方，以便对方了解。

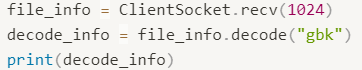


③文件传送：打开文件，每次读取相应的字节数（1024），然后将数据发送出去，直到读取结束，就关闭这个TCP连接。



文件接收:

①接收并打印文件信息。



②预先准备好接收路径, 然后打开文件，循环接收对方传送的数据，并写入到文件中，直到这个套接字关闭，说明此文件已经接收成功。



测试和调用：

仍然用两个用户进行测试：用户A和用户B，当用户A向用户发送文件时，首先发送File告知，然后用户A新建一个TCP连接到B的服务器（称为文件连接），A通过文件连接发送文件给B，B通过文件连接接收文件，传输文件结束后关闭连接。

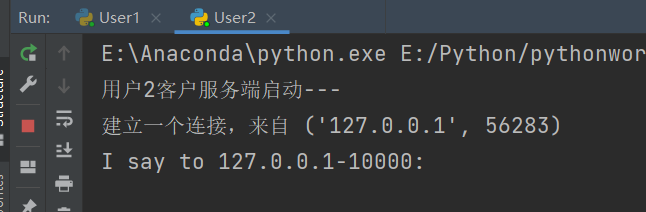
实现： 发送方发送File后，新建一个socket，连接到接收方的服务器，然后传输文件。

接收方接收文件File后，等待一个连接，得到这个连接后就通过这个连接接收文件。

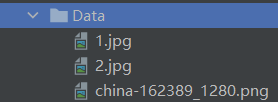
**截图**：

首先建立两个客户端，并相互连接



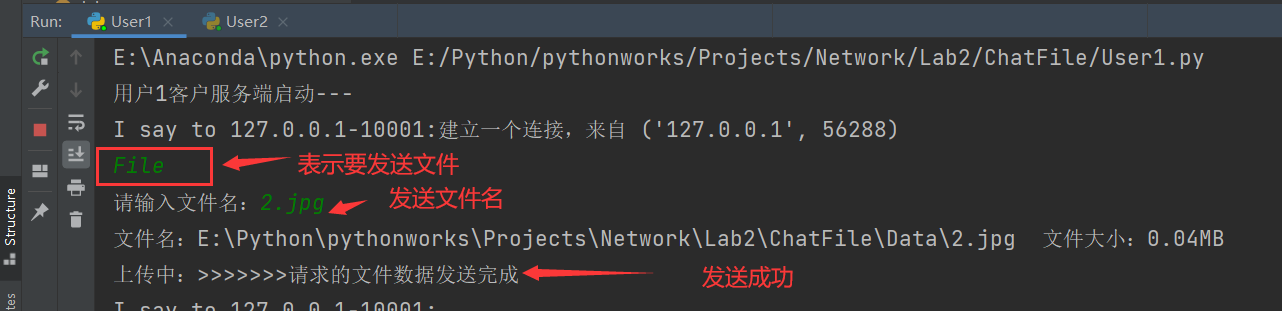


用户1向用户2发送一个文件：2.jpg，然后用户2将文件存放到datareceive文件夹中（）

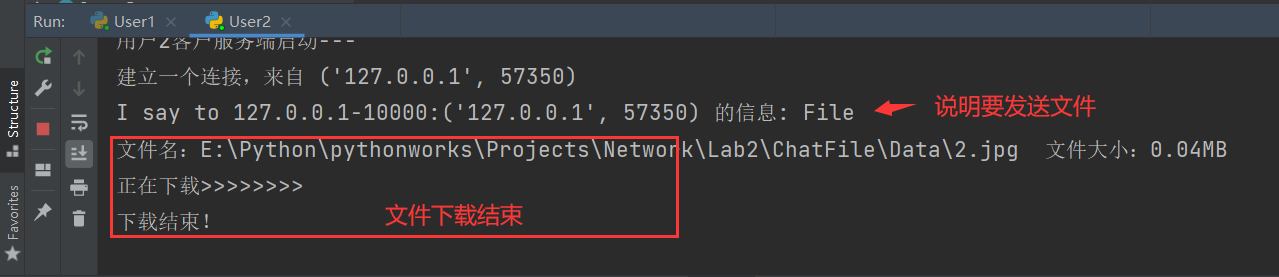


 可以看到目前接收文件夹仅有一张图片。

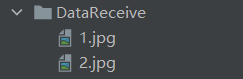
用户1向用户2发送文件2.jpg



这时用户2收到文件



这时文件夹中已经有了这个文件



## 实验心得与不足

心得：1. 学会了python的TCP和UDP连接方式。

1. 学会了Python的简单多线程和线程池操作
2. 学会了文件传输的方式，最简单的方法就是可以要发送文件时为这个文件单独创建一个TCP连接，当文件传输结束的时候关闭这个连接，这样接收方接收完文件后就会自动停止接收，而不需要人为计算发送的块数据，等到接收足够块数目的时候再停止。

不足： 没有学习Python的锁等控制并发的功能，实际上在开发聊天程序的时候是需要用到锁和其他的协调功能的，但是由于我设计的聊天工具比较简单，并没有用到这方面功能，只是保证每个线程不会用到冲突的资源。