网络编程期末考

时间:2020,1,3

17级计算机科学与技术1-2班

本次考试共计五题，各题占分比例20%，每题请附上代码、执行结果截图及必要说明。

1 UDP的设计，为源端口标识了源机器上发送数据包的特定进程或程序，目标端口则标识了目标IP地址上进行该会话的特定应用程序。

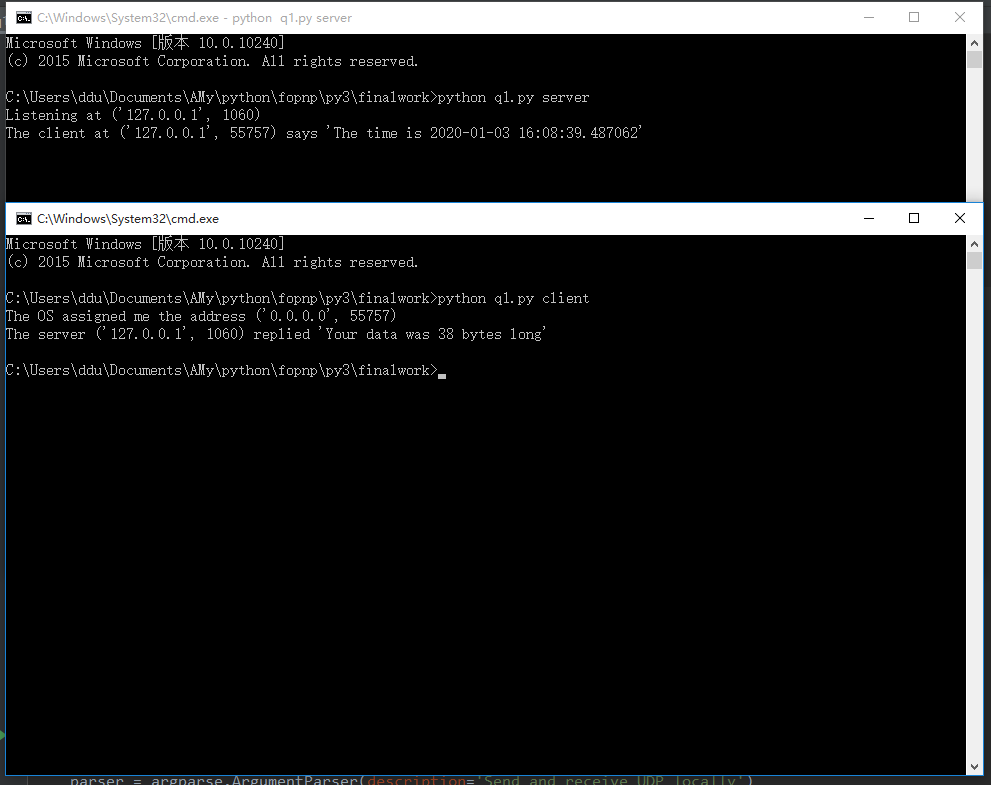
1. 请撰写一个简单的UDP服务器和UDP客户端代码并执行，请分别说明客户端和服务器执行之流程
2. 代码没有检查该数据报的源地址，没有验证该数据报是否确实是服务器发回的响应，所以容易遭受黑客攻击。请说明可以使用两个快速解决方案，以防止遭受黑客攻击。

（1）

代码：

import argparse, socket  
from datetime import datetime  
  
MAX\_BYTES = 65535  
  
  
def server(port):  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 sock.bind(('127.0.0.1', port))  
 print('Listening at {}'.format(sock.getsockname()))  
 while True:  
 data, address = sock.recvfrom(MAX\_BYTES)  
 text = data.decode('ascii')  
 print('The client at {} says {!r}'.format(address, text))  
 text = 'Your data was {} bytes long'.format(len(data))  
 data = text.encode('ascii')  
 sock.sendto(data, address)  
  
  
def client(port):  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 text = 'The time is {}'.format(datetime.now())  
 data = text.encode('ascii')  
 sock.sendto(data, ('127.0.0.1', port))  
 print('The OS assigned me the address {}'.format(sock.getsockname()))  
 data, address = sock.recvfrom(MAX\_BYTES) # Danger! See Chapter 2  
 text = data.decode('ascii')  
 print('The server {} replied {!r}'.format(address, text))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 choices = {'client': client, 'server': server}  
 parser = argparse.ArgumentParser(description='Send and receive UDP locally')  
 parser.add\_argument('role', choices=choices, help='which role to play')  
 parser.add\_argument('-p', metavar='PORT', type=int, default=1060,  
 help='UDP port (default 1060)')  
 args = parser.parse\_args()  
 function = choices[args.role]  
 function(args.p)

运行效果：



说明：

首先运行服务器，将其部署到本地的1060端口，然后运行客户端，客户端会发送一条当前的时间数据给服务器，然后服务器接收到消息后再将发送数据的长度返回给客户端

2）答：

方法1：设计或使用在请求中包含唯一标识符或请求ID的协议，在响应中重复特定请求的唯一标识符或请求ID

方法2：可以检查响应数据包的地址与请求数据包的地址是否相同，也口由使用connect()来阻止其他地址向客户端发送数据包

2 Python 3.4引进TLS函数，使得能在Python应用程序中安全地使用TLS，比起以前版本的Python实现起来要容易得多了。

1. 请创建及使用一个TLS默认上下文方法，写出一个简单的客户端和服务器通过TLS套接字进行安全通信的方法。
2. 并在代单中说明，为一个套接字提供安全通信所需要的3个步骤。

程序如下

* 服务器端依TLS协议进行连接，等待客户端要求连接。
* 客户端依TLS协议要求连接后，
* 服务器端传送

The quick brown fox jumps over a lazy dog. 文字

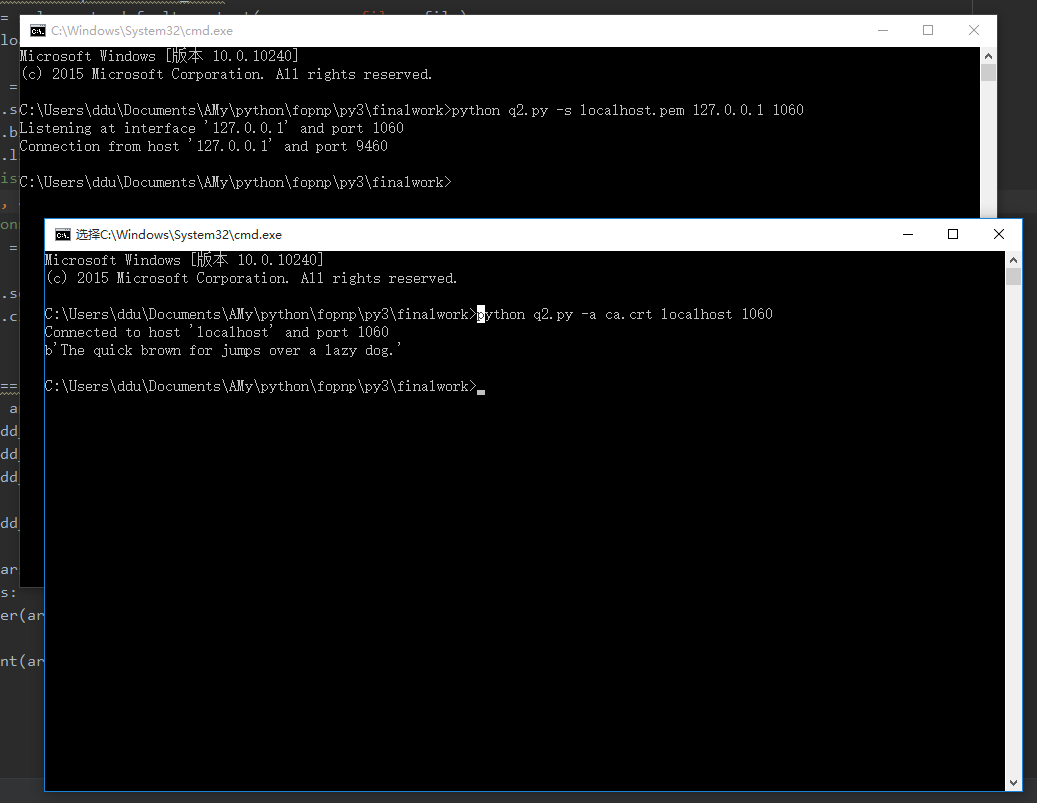
* 客户端于接收后将接收到的文字，将其打印出来后关闭连接。

服务器于传送所有文字后结束连接。

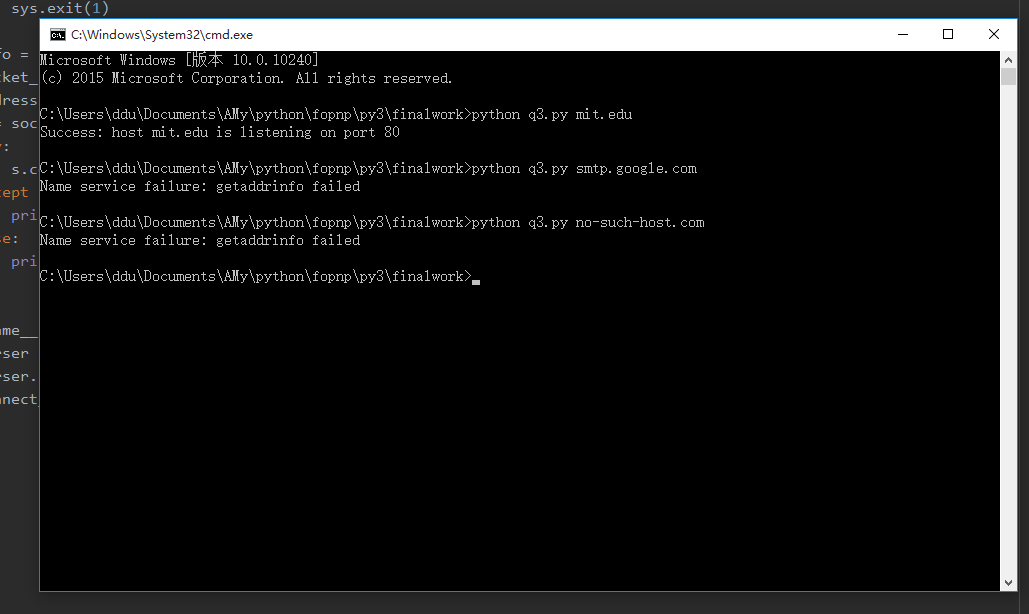
代码：

import argparse  
import socket  
import ssl  
  
  
def client(host, port, cafile=None):  
 purpose = ssl.Purpose.SERVER\_AUTH  
 # 创建TLS默认上下文方法  
 context = ssl.create\_default\_context(purpose, cafile=cafile)  
 raw\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 # 连接  
 raw\_sock.connect((host, port))  
 print('Connected to host {!r} and port {}'.format(host, port))  
 ssl\_sock = context.wrap\_socket(raw\_sock, server\_hostname=host)  
  
 while True:  
 data = ssl\_sock.recv(1024)  
 if not data:  
 break  
 print(repr(data))  
  
  
def server(host, port, certfile, cafile=None):  
 purpose = ssl.Purpose.CLIENT\_AUTH  
 # 创建TLS默认上下文方法  
 context = ssl.create\_default\_context(purpose, cafile=cafile)  
 # 加载授权  
 context.load\_cert\_chain(certfile)  
 listener = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 listener.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)  
 # 绑定端口  
 listener.bind((host, port))  
 # 监听  
 listener.listen(1)  
 print('Listening at interface {!r} and port {}'.format(host, port))  
 raw\_sock, address = listener.accept()  
 print('Connection from host {!r} and port {}'.format(\*address))  
 ssl\_sock = context.wrap\_socket(raw\_sock, server\_side=True)  
   
 # 发送数据  
 ssl\_sock.sendall('The quick brown for jumps over a lazy dog.'.encode('ascii'))  
 # 关闭服务器  
 ssl\_sock.close()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 parser = argparse.ArgumentParser(description='Safe TLS client and server')  
 parser.add\_argument('host', help='hostname or IP address')  
 parser.add\_argument('port', type=int, help='TCP port number')  
 parser.add\_argument('-a', metavar='cafile', default=None,  
 help='authority: path to CA certificate PEM file')  
 parser.add\_argument('-s', metavar='certfile', default=None,  
 help='run as server: path to server PEM file')  
 args = parser.parse\_args()  
 if args.s:  
 server(args.host, args.port, args.s, args.a)  
 else:  
 client(args.host, args.port, args.a)

效果



3 getaddrinfo()函数是socket模块中涉及地址的众多操作之一。请撰写一个代码应用getaddrinfo()函数进行了一个简单的“你在吗?“ 的测试(可参考课本代码4-1)。可以在命令行提供任意连接网络服务器的名字，然后试通过一个流套接字快速连接80端口。请利用下列三个网络服务器作为测试使用: mit.edu，smtp.google.com，no-such-host.com。



4 请撰写一单线程服务器，使用一个最简单的TCP协议进行。在这个协议中，客户端可以询问2个问题，这2个问题都以纯文本的ASCII字符表示，在发出请求的问题后，客户端将等待服务器的回应。例如问题为**Every cloud has a silver** ?

服务器发送回的应答在结尾用句点表示响应信息的结束。

**lining**.

每个问题的结尾使用ASCII的问号字符表示问题的结束。(另二句为**Finding a needle in a? haystack. 及Don’t put all your eggs in one? basket.)**。

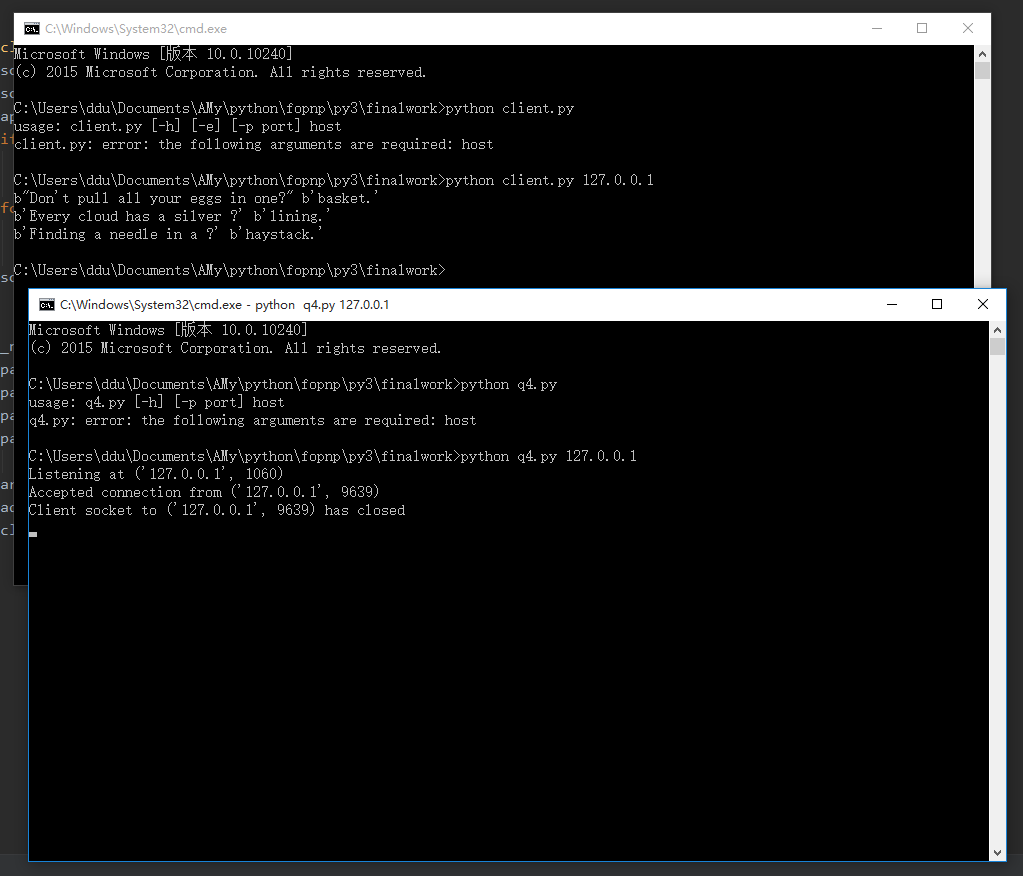
1. 请仿例7-4创建多个共享相同内存空间的多线程之代码。
2. 请仿例7-5 运用socketserver模块创建多个共享相同内存空间的线程代码。并请说明比较这二种多线程之代码之优缺点。

（1）

代码

import zen\_utils  
from threading import Thread  
  
  
def start\_threads(listener, workers=4):  
 t = (listener,)  
 for i in range(workers):  
 Thread(target=zen\_utils.accept\_connections\_forever, args=t).start()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 address = zen\_utils.parse\_command\_line('multi-threaded server')  
 listener = zen\_utils.create\_srv\_socket(address)  
 start\_threads(listener)

效果

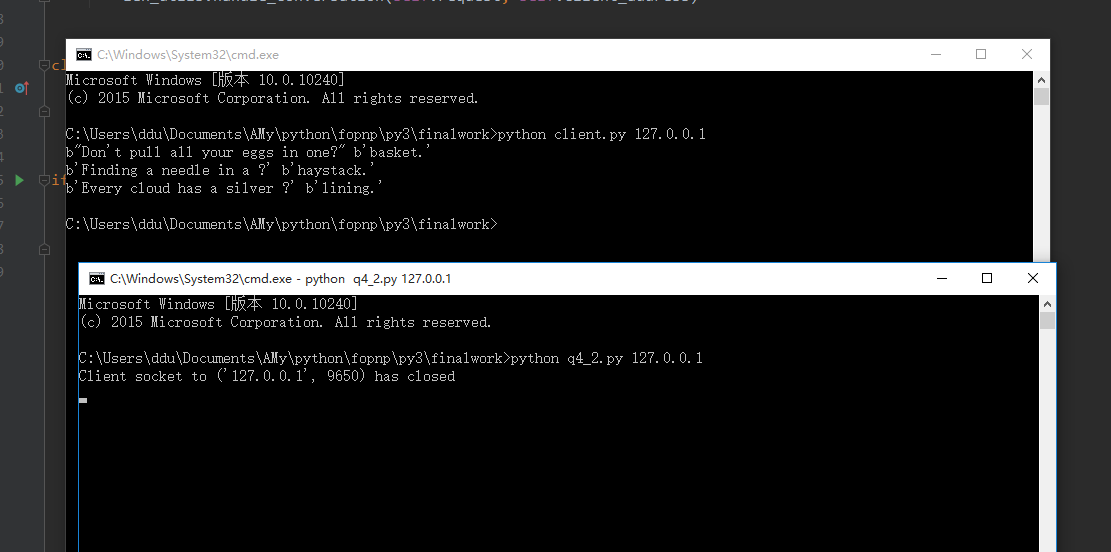


（2）

代码

from socketserver import BaseRequestHandler, TCPServer, ThreadingMixIn  
import zen\_utils  
  
  
class ZenHandler(BaseRequestHandler):  
 def handle(self):  
 zen\_utils.handle\_conversation(self.request, self.client\_address)  
  
  
class ZenServer(ThreadingMixIn, TCPServer):  
 allow\_reuse\_address = 1  
 # address\_family = socket.AF\_INET6 # uncomment if you need IPv6  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 address = zen\_utils.parse\_command\_line('legacy "SocketServer" server')  
 server = ZenServer(address, ZenHandler)  
 server.serve\_forever()

效果



比较：

第一种方法启动了固定数量的线程，线程的数量可以由服务器的管理员根据特定服务器和操作系统能够高效管理的控制线程数量来指定。

第二种方法则由服务器的客户端连接池来决定启动的线程数量——不限制服务器最终启动的线程数量，这使得攻击者可以很容易令服务器过载，因此在开发用于生产环境以及面向客户的服务时，不推荐使用这个标准库模块

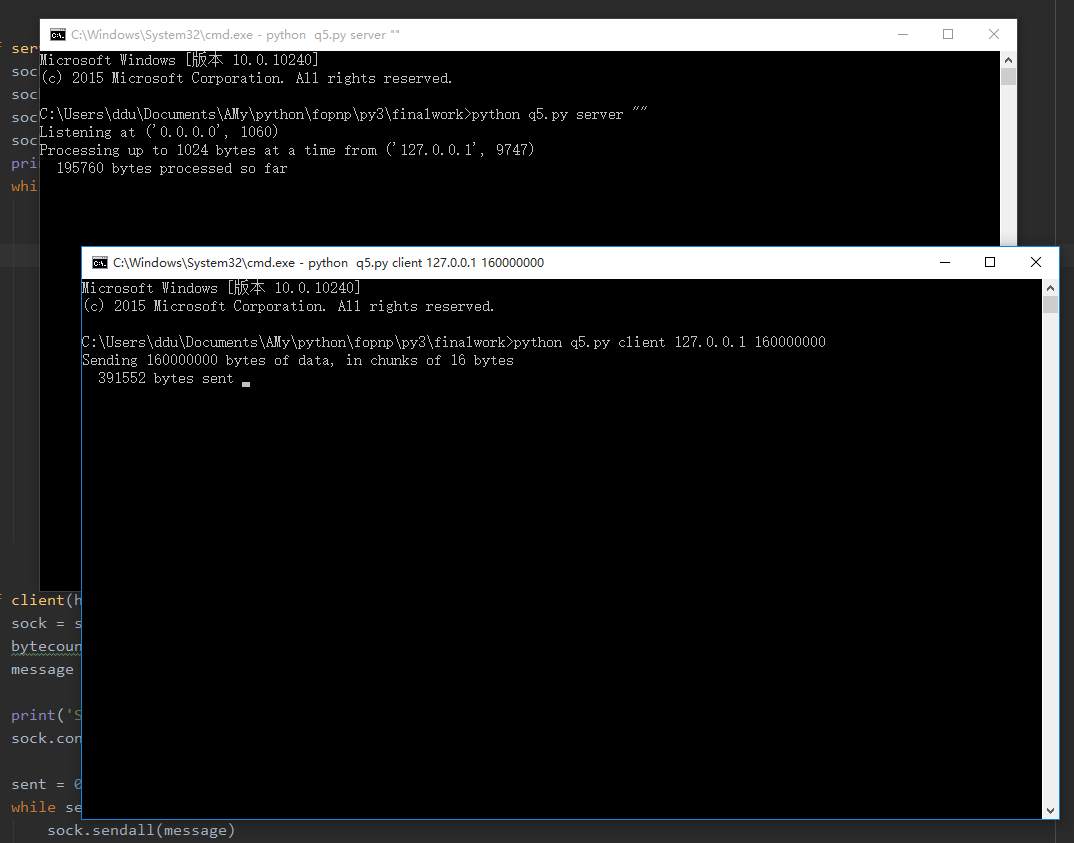
5 在计算机科学中，当两个程序共享有限的资源时，由于不好的计划，只能一直等待对方结束资源占用，这种情况称为死锁 (deadlock)。

1. 请撰写一个代码可能会产生死锁的服务器和TCP客户端代码并执行(可参考代码清单3-2)。
2. 请说明可以使用何种解决方案，以防止死锁现象。

代码

import argparse, socket, sys  
  
  
def server(host, port):  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)  
 sock.bind((host, port))  
 sock.listen(1)  
 print('Listening at', sock.getsockname())  
 while True:  
 sc, sockname = sock.accept()  
 print('Processing up to 1024 bytes at a time from', sockname)  
 n = 0  
 while True:  
 data = sc.recv(1024)  
 if not data:  
 break  
 output = data.decode('ascii').upper().encode('ascii')  
 sc.sendall(output) # send it back uppercase  
 n += len(data)  
 print('\r %d bytes processed so far' % (n,), end=' ')  
 sys.stdout.flush()  
 print()  
 sc.close()  
 print(' Socket closed')  
  
  
def client(host, port, bytecount):  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 bytecount = (bytecount + 15) // 16 \* 16 # round up to a multiple of 16  
 message = b'capitalize this!' # 16-byte message to repeat over and over  
  
 print('Sending', bytecount, 'bytes of data, in chunks of 16 bytes')  
 sock.connect((host, port))  
  
 sent = 0  
 while sent < bytecount:  
 sock.sendall(message)  
 sent += len(message)  
 print('\r %d bytes sent' % (sent,), end=' ')  
 sys.stdout.flush()  
  
 print()  
 sock.shutdown(socket.SHUT\_WR)  
  
 print('Receiving all the data the server sends back')  
  
 received = 0  
 while True:  
 data = sock.recv(42)  
 if not received:  
 print(' The first data received says', repr(data))  
 if not data:  
 break  
 received += len(data)  
 print('\r %d bytes received' % (received,), end=' ')  
  
 print()  
 sock.close()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 roles = ('client', 'server')  
 parser = argparse.ArgumentParser(description='Get deadlocked over TCP')  
 parser.add\_argument('role', choices=roles, help='which role to play')  
 parser.add\_argument('host', help='interface the server listens at;'  
 ' host the client sends to')  
 parser.add\_argument('bytecount', type=int, nargs='?', default=16,  
 help='number of bytes for client to send (default 16)')  
 parser.add\_argument('-p', metavar='PORT', type=int, default=1060,  
 help='TCP port (default 1060)')  
 args = parser.parse\_args()  
 if args.role == 'client':  
 client(args.host, args.p, args.bytecount)  
 else:  
 server(args.host, args.p)

运行效果

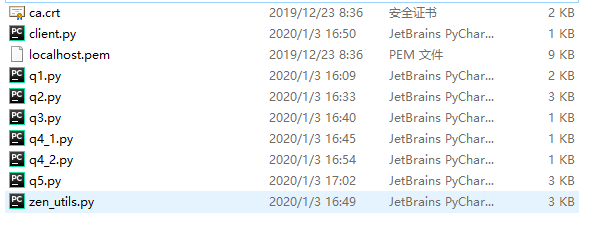


解决方法：

方法1：客户端和服务器可以通过套接字选项将阻塞关闭。这样一来，像send()和recv()这样的调用在得知还不能发送数据时就会立即返回。

方法2：程序可以使用某种技术同时处理来自多个输入的数据。可以采用多个线程或进程来处理（比如，一个用来向套接字发送数据，另一个可能就负责从套接字读取数据）；也可以运行select()或poll()等操作系统调用，使得程序在发送套接字和接受套接字繁忙时等待，当它们当中任意一个空闲时就做出响应。

代码附录：



Q代表问题

Q1.py 为第一题代码

Q2.py 为第二题代码

Q3.py 为第三题代码

Q4\_1.py 为第四题第一问代码

Q4\_2.py 为第四题第二问代码

Client.py为测试第四题代码的客户端

Q5.py 为第五题代码