****

Python网络编程课程设计

实 验 报 告

|  |  |
| --- | --- |
| **学 院** | 网络空间安全学院 |
| **专 业** | 网络工程 |
| **班 级** | 18272412 |
| **学 号** | 18041618 |
| **学生姓名** | 廖越强 |
| **教师姓名** | 胡伟通 |
| **完成日期** | 10.23 |
| **成 绩** |  |

**实验二 基于UDP的C/S程序设计**

1. **实验目的**
2. 熟悉UDP协议的原理和特点
3. 掌握基于UDP协议的应用层协议设计方法。
4. 掌握基于UDP协议的C/S网络应用程序设计和实现方法。
5. **实验内容**

**以下题目任选其一，或在征求指导老师同意的情况下，完成难度相似的基于UDP的网络应用。**

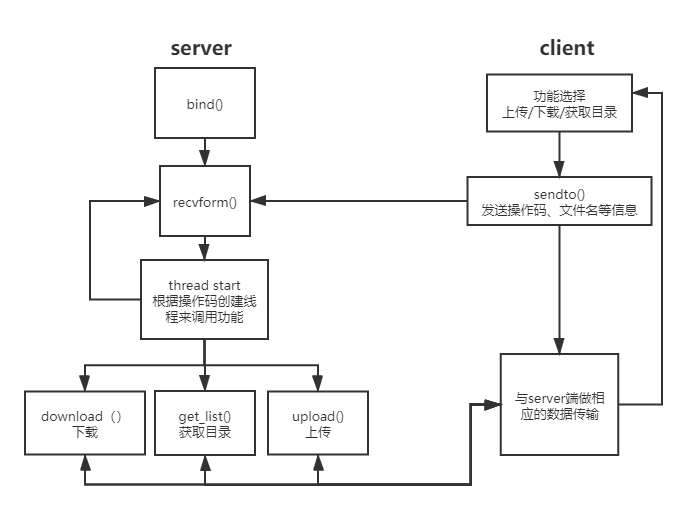
1. 编写基于UDP的聊天室程序，实现多人聊天功能。自己设计应用协议，要求实现以下功能：
2. 用户注册
3. 用户登录
4. 公聊
5. 私聊
6. 编写基于UDP的文件传输程序（可以参考TFTP协议，不能照抄），实现上传文件到服务器和从服务器下载文件，要求实现以下功能：
7. 获取文件列表
8. 上传文件
9. 下载文件

注意测试两个客户端同时进行上传和下载的情况

1. 基于UDP的屏幕广播程序，将服务器的桌面通过UDP的广播或多播，发送给对应的客户端程序，客户端接收并显示服务器的桌面信息（注意屏幕图片大小超过UDP的最大报文）。
2. **程序设计思路**
3. **所选题目说明：**

选择题目2（基于UDP的文件传输程序）

1. **网络应用拓扑结构**



1. **应用层协议设计**

1.操作码(short int，2字节)：

0：上传(upload)，

1：下载(download)，

2：获取目录

3：文件数据传输

4：错误代码

2.正文数据(data)：每次从文件里最多读取512字节

3.传输缓冲区大小设置为1024字节

1. **所选用的Python库介绍**

1.socket：一个操作网络接口的库。本实验中主要用于创建udp连接。

2.threading：该模块提供了操作多个线程的底层原语。这里用于创建线程，用于多线程处理客户端的请求。

3.struct：将字节串打包为二进制数据。实验中发送端把操作码和正文数据打包，做为一个整体进行传输；接收端进行解包读取数据。

4.os: 多种操作系统的接口。

1. **程序源代码**

**注意源代码要有详细的注释。 同学们提交的每个程序都应该遵循Honor Code（诚实代码保证）的要求。**

**请大家特别注意一定要在每个程序首部的注释中加上以下保证：**

**# 我真诚地保证：**

**# 我自己独立地完成了整个程序从分析、设计到编码的所有工作。**

**# 如果在上述过程中，我遇到了什么困难而求教于人，那么，我将在程序实习报告中**

**# 详细地列举我所遇到的问题，以及别人给我的提示。**

**# 在此，我感谢 XXX, …, XXX对我的启发和帮助。下面的报告中，我还会具体地提到**

**# 他们在各个方法对我的帮助。**

**# 我的程序里中凡是引用到其他程序或文档之处，**

**# 例如教材、课堂笔记、网上的源代码以及其他参考书上的代码段,**

**# 我都已经在程序的注释里很清楚地注明了引用的出处。**

**# 我从未没抄袭过别人的程序，也没有盗用别人的程序，**

**# 不管是修改式的抄袭还是原封不动的抄袭。**

**# 我编写这个程序，从来没有想过要去破坏或妨碍其他计算机系统的正常运转。**

**# <学生姓名> 廖越强**

1. **服务器端源码**
2. import socket
3. import threading
4. import struct
5. import os
6. import time
7. def upload(client\_addr, filename):
8. count = 0
9. # f = open(filename, 'wb')
10. f = open(os.getcwd()+'/file/'+filename, 'wb')
11. print("filename:", filename)
12. s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)
13. send\_data = struct.pack('!HH', 4, count)
14. s.sendto(send\_data, (client\_addr))  # 把将要用于进行数据传输的port发送给客户端
15. print("use random\_port:",client\_addr[1])
16. # 文件数据传输
17. while True:
18. data, client\_addr = s.recvfrom(1024)
19. count = struct.unpack('!H', data[2:4])  # 数据块编号
20. data = data[4:]
21. if str(data) != "b'end'":
22. f.write(data)
23. print('receive '+str(count[0])+' packets')
24. else:
25. break
26. s.sendto('ok'.encode('gb2312'), client\_addr)
27. f.close()
28. s.close()
29. print("finished!")
30. def download(client\_addr, filename):
31. count = 0
32. print("filename:", filename)
33. # f = open(filename, 'rb')
34. f = open(os.getcwd()+'/file/'+filename, 'rb')
35. s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)
36. while True:
37. data = f.read(512)
38. if str(data) != "b''":
39. data = struct.pack('!HH', 3, count) + data # count 块编号
40. s.sendto(data, client\_addr)
41. print(str(count)+'packets')
42. else:
43. data = struct.pack('!HH', 3, count) + 'end'.encode('gb2312')    # 传输结束
44. s.sendto(data, client\_addr)
45. break
46. data, client\_addr = s.recvfrom(1024)
47. count+=1
48. f.close()
49. s.close()
50. print("finished!")
51. def get\_list(ip, port):
52. path = os.getcwd()+"/file"  # 获取文件夹路径
53. dirs = os.listdir(path)
54. print("path:",path)
55. print("dirs:",dirs)
56. udp\_server = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)
57. dirs = ','.join(dirs).encode('gb2312')  # list转为string并用逗号分隔，再编码
58. send\_data = struct.pack('!H%ds' %(len(dirs)), 3, dirs)  # 将数据打包
59. udp\_server.sendto(send\_data, (ip, port))
60. print("finished!")
61. if \_\_name\_\_ == "\_\_main\_\_":
62. udp\_server = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)
63. msg\_server = ("127.0.0.1", 8877)
64. udp\_server.bind(msg\_server)# 绑定端口:
65. print("Bind UDP on:",msg\_server)
67. while True:
68. data, client\_addr = udp\_server.recvfrom(1024)
69. mode = struct.unpack('!H', data[:2])# 读取操作码
70. # 多线程处理
71. if(mode[0] == 0):
72. print('connected from %s:%s'%client\_addr)
73. filename = data[2:].split(b"\x00")[0].decode('gb2312') # 读取文件名
74. print('mode:',mode[0]," Upload..")
75. my\_thread = threading.Thread(target=upload, args=(client\_addr, filename))# 创建相应线程
76. my\_thread.start()# 执行
77. if(mode[0] == 1):
78. print('connected from %s:%s'%client\_addr)
79. filename = data[2:].split(b"\x00")[0].decode('gb2312')
80. print('mode:',mode[0]," Download..")
81. my\_thread = threading.Thread(target=download, args=(client\_addr, filename))# 创建相应线程
82. my\_thread.start()
83. if(mode[0] == 2):
84. print('connected from %s:%s'%client\_addr)
85. print('mode:',mode[0]," Get list..")
86. my\_thread = threading.Thread(target=get\_list, args=(client\_addr))# 创建相应线程
87. my\_thread.start()
88. **客户端源码**

import socket

import os

import struct   # 利用pack打包数据

import time

addr\_msg = ("127.0.0.1", 8877) # server's address

def upload():

    print('select: upload')

    s = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)

    print('input file name: ')

    # filename = 'test.zip'

    filename = input()

    f = open(filename,'rb')

    count = 0

    while True:

        if count == 0:

            data = struct.pack('!H8s', 0, (filename).encode('gb2312'))

            s.sendto(data, addr\_msg)

            data = s.recvfrom(1024)

            new\_addr = data[1] # 把将要用于进行数据传输的port发送给客户端

        data = f.read(512)

        if str(data) != "b''":

            data = struct.pack('!HH', 3, count) + data # count 块编号

            s.sendto(data, new\_addr)

            print(str(count)+'packets')

        else:

            data = struct.pack('!HH', 3, count) + 'end'.encode('gb2312') # count 块编号

            s.sendto(data, new\_addr)

            break

        data, server\_addr = s.recvfrom(1024)

        count+=1

    print('finished!','\treceive from:', server\_addr,'\n')

    f.close()

    s.close()

def download():

    print('select: download')

    s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

    print('input file name: ')

    # filename = 's.zip'

    filename = input()

    f = open(filename, 'wb')

    print(filename)

    print("downloading..")

    print('start-time:',time.localtime().tm\_hour,'h',time.localtime().tm\_min,'m',time.localtime().tm\_sec,'s')

    data = struct.pack('!H8s', 1, (filename).encode('gb2312'))

    s.sendto(data, addr\_msg)

    count = 0

    while True:

        data, client\_addr = s.recvfrom(1024)

        count = struct.unpack('!H', data[2:4])

        data = data[4:]

        if str(data) != "b'end'":

            f.write(data)

            print('receive '+str(count[0])+' packets')

        else:

            break

        s.sendto('ok'.encode('gb2312'), client\_addr)

    f.close()

    s.close()

    print("finished!")

def get\_list():

    print('select: get\_list')

    addr\_msg = ("127.0.0.1", 8877) # server's address

    s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

    data = struct.pack('!H8s', 2, 'get\_list'.encode('gb2312'))

    s.sendto(data, addr\_msg)    # send request

    data, server\_addr = s.recvfrom(1024) # receive

    data = data[2:].decode('gb2312')

    print(data,'\treceive from:', server\_addr,'\n')

    s.close()

while True:

    print('####Select mode:')

    print('    0-upload   1-download   2-get files list   3-quit')

    mode = input()

    if mode == '0':

        upload()

    elif mode == '1':

        download()

    elif mode == '2':

        get\_list()

    elif mode == '3':

        print('goodbye!')

        break

    else:

        print('invalid input')

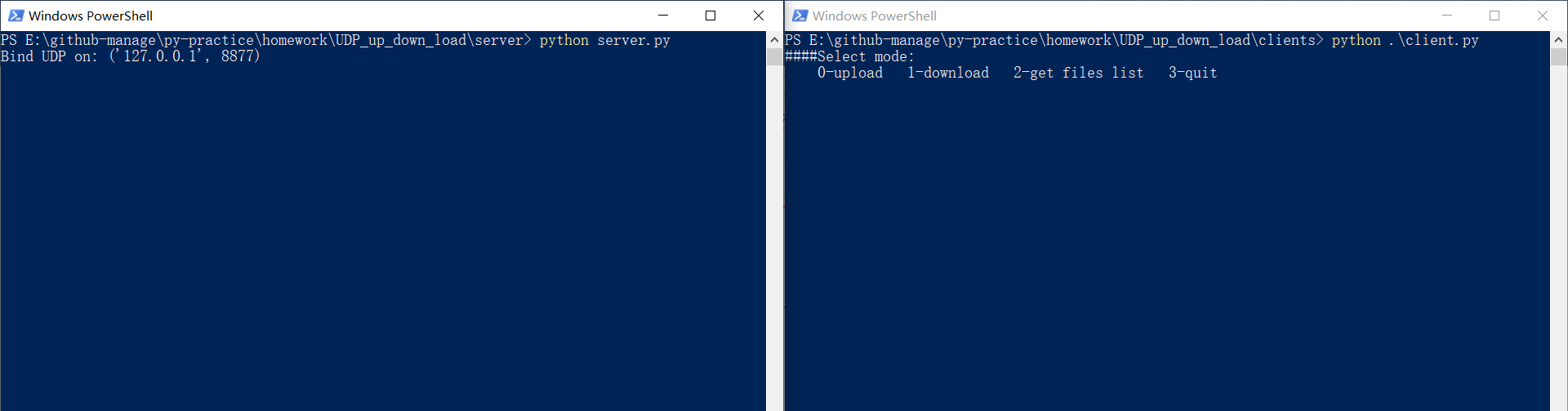
1. **程序测试方法及测试结果记录（不能光截图，要有相应的文字说明）**
2. **测试方法**

**本地运行server端，再运行客户端经行操作**

1. **测试流程**

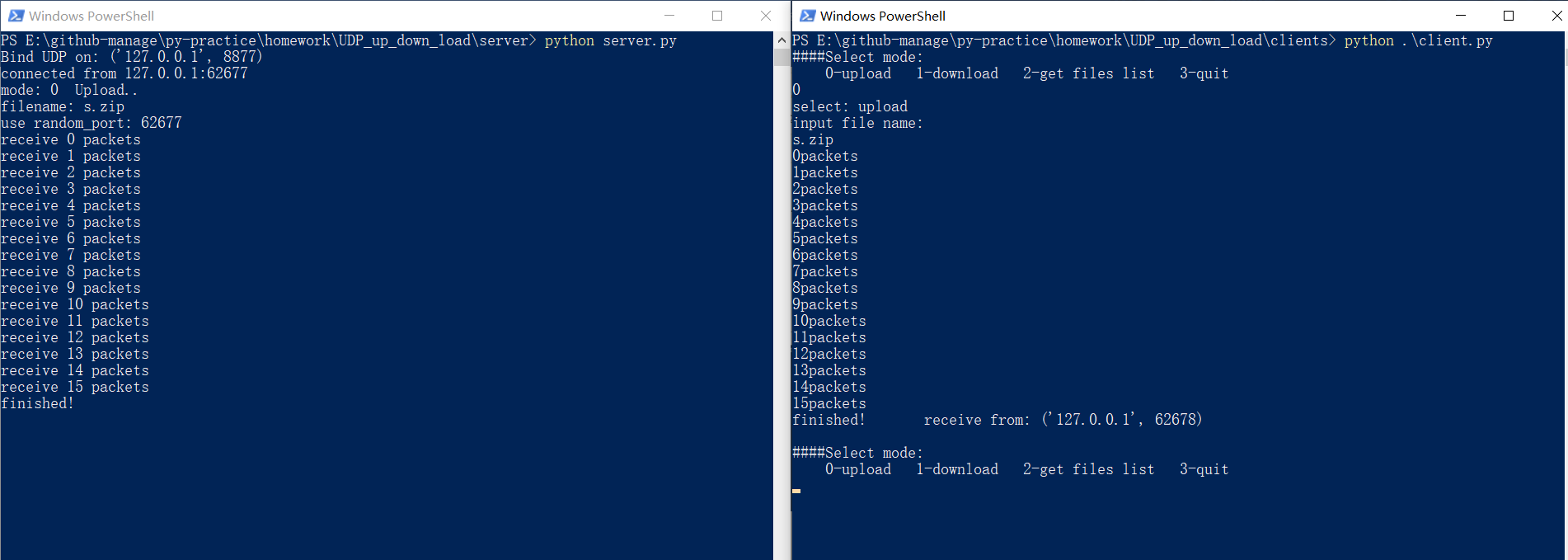
测试顺序：

1. 上传文件
2. 查看所有文件
3. 下载文件
4. 分别运行server端和client端

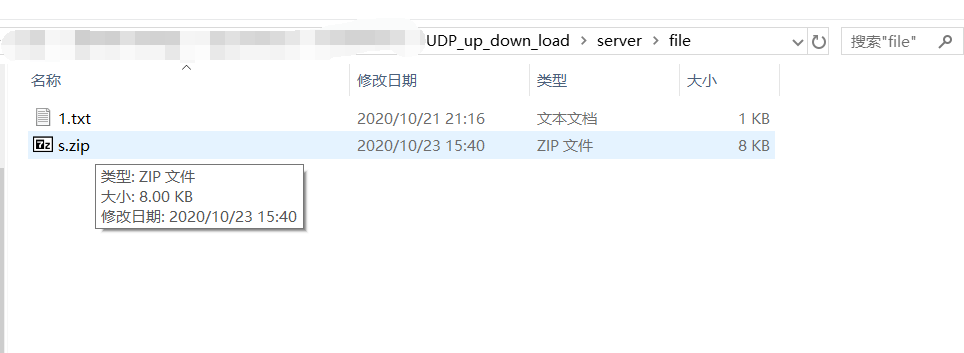


1. 上传文件测试

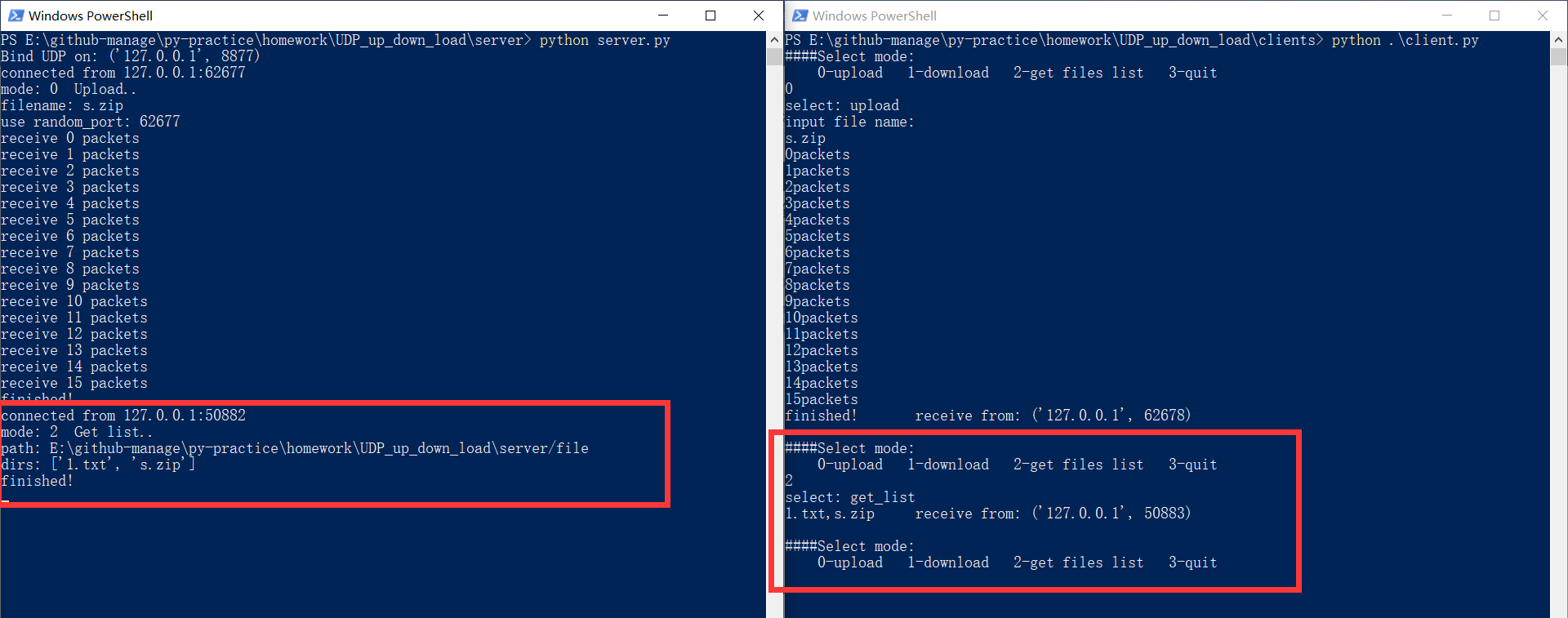
输入0进入文件传输模式，传输文件s.zip



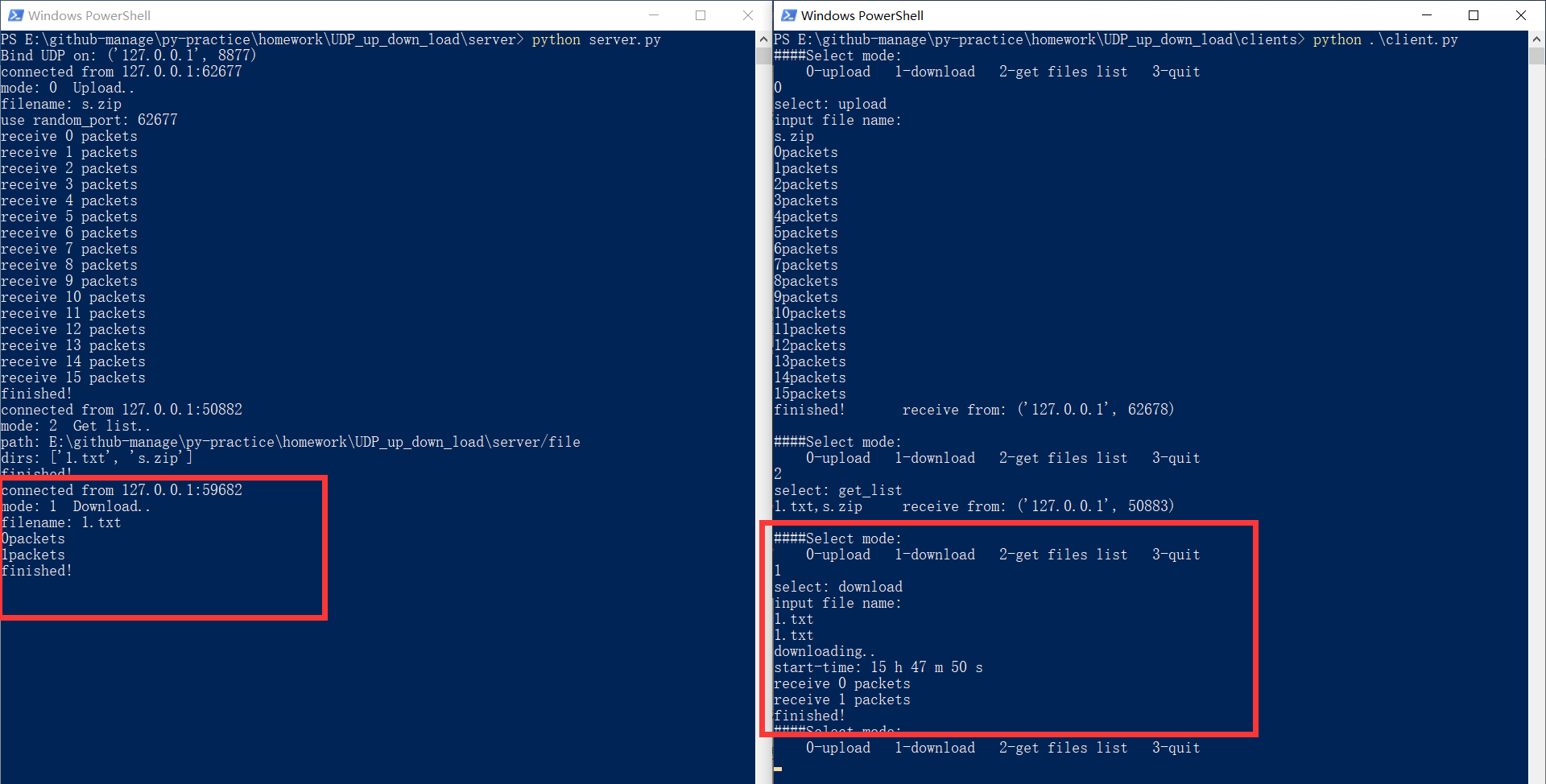
查看服务端存放文件的目录，成功传输文件s.zip



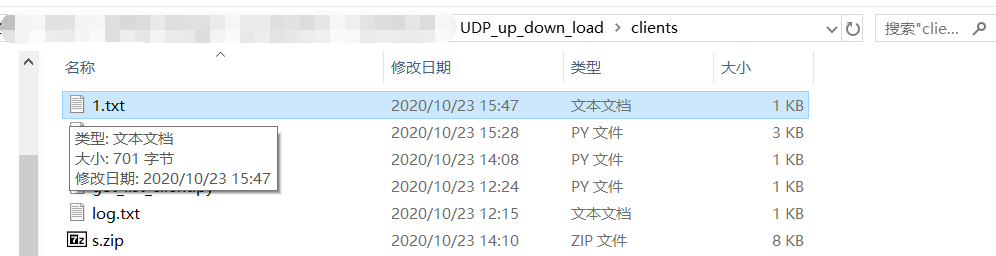
1. 选择mode 2，查看服务端目录下所有文件



1. 选择mode 1，把服务端上的文件1.txt下载到客户端



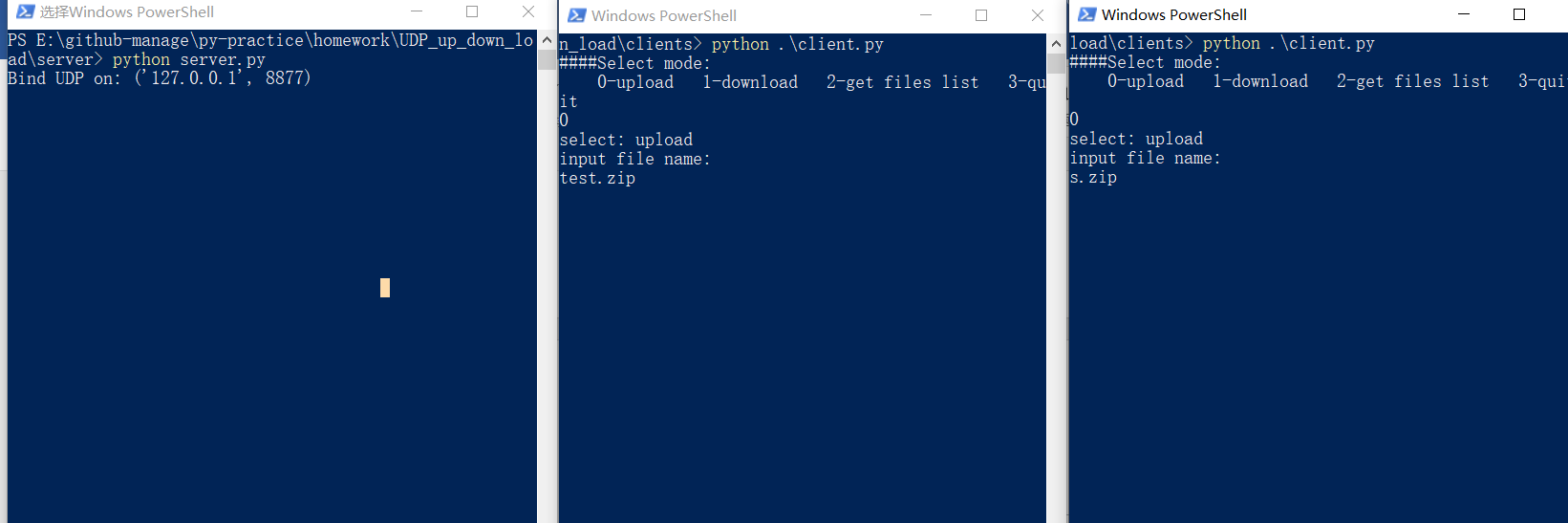
查看客户端本地文件，1.txt已经成功下载



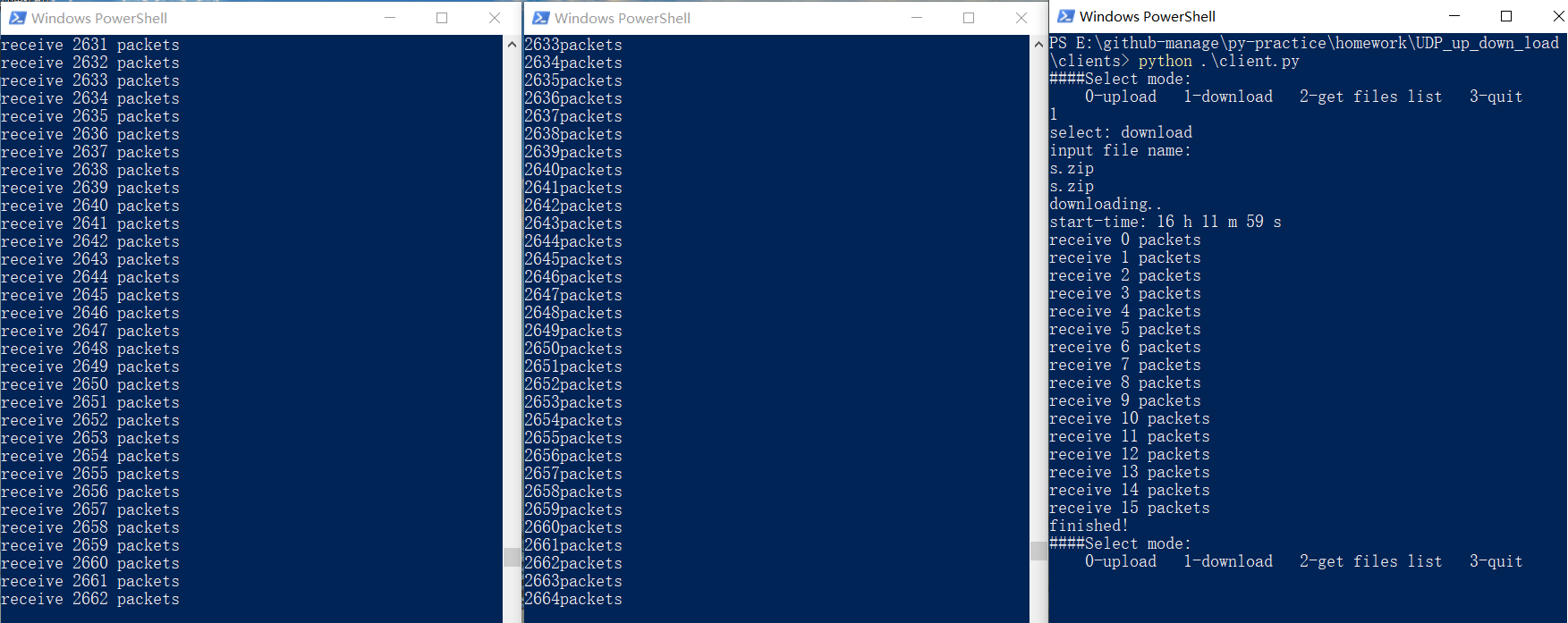
1. 两个客户端同时上传/下载测试

1.运行一个server端，两个client端

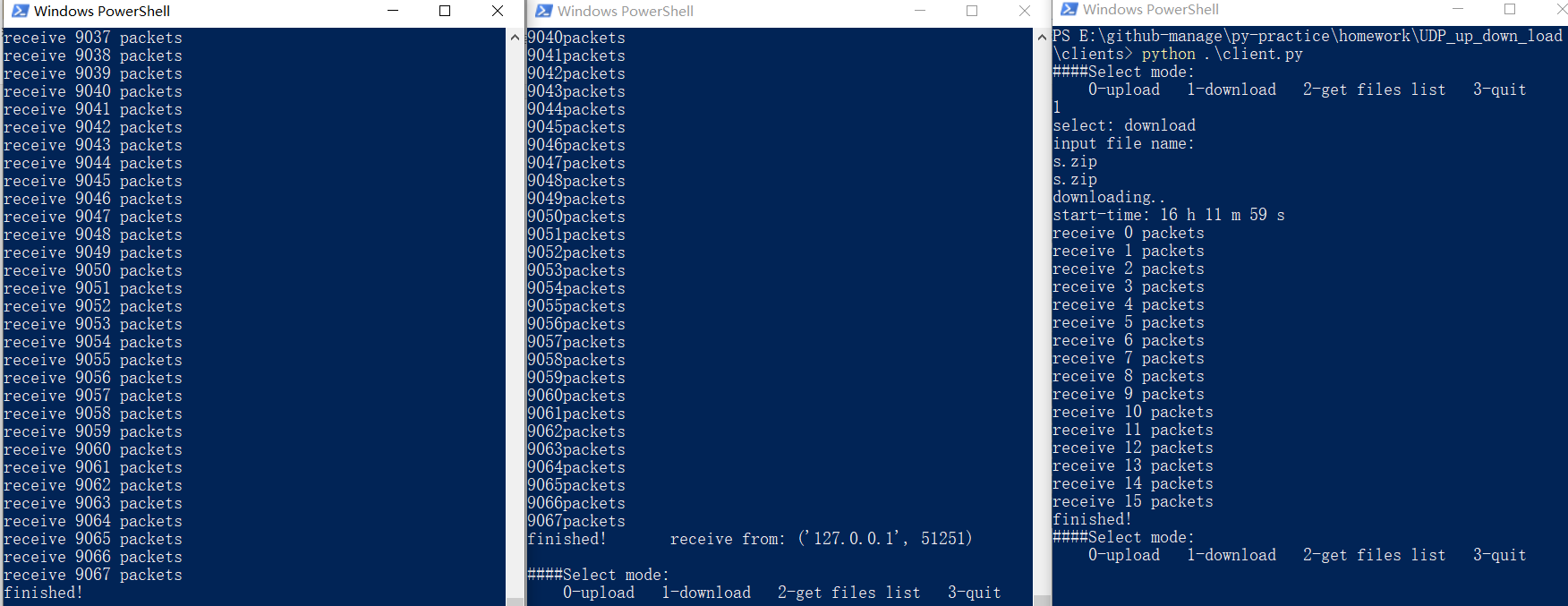
2.一个client上传大文件test.zip，上传期间另一个client下载一个小文件s.zip



可以看到大文件test.zip还在传输，s.zip已经下载完成了



最终，两个客户端都完成传输任务



1. **实验分析总结及心得（该部分也是评分的一个重点）**

（结合所学知识对实验过程中观察到的实验结果进行分析总结，以便加深对知识的理解，并总结通过实验学到的知识或技术）

特别推荐学生写出做实验时遇到的问题，并描述从原理分析得到解决方案的过程。

1.犯了个很蠢的错误，打开一个文件(open)使用参数的应该是文件名，但在传数据的时候忘记传文件名了，open()直接把文件内容的作为参数，导致报错“传入了一个无效的参数”。解决办法是在发送文件内容之前先把文件名传输过去，作为open()打开/创建文件的参数。

2.数据包长度的计算问题。刚开始每次读取文件的大小设置为1024bytes，然后和操作码（2bytes）拼接起来，没有想太多缓冲区也设置成了1024bytes，导致报错。解决办法是缩小每次读取文件的长度，把1024bytes缩小到512bytes，这样和操作码拼接后就不超过缓冲区大小了。

3.第三个遇到的问题是两个客户端如何同时进行传输操作。仔细想了后决定还是使用创建线程来处理客户端请求；这个比较麻烦，虽然已经学过线程，但实践起来还是遇到了许多小错误，但都能根据报错提示解决吧，这里就不记录了。