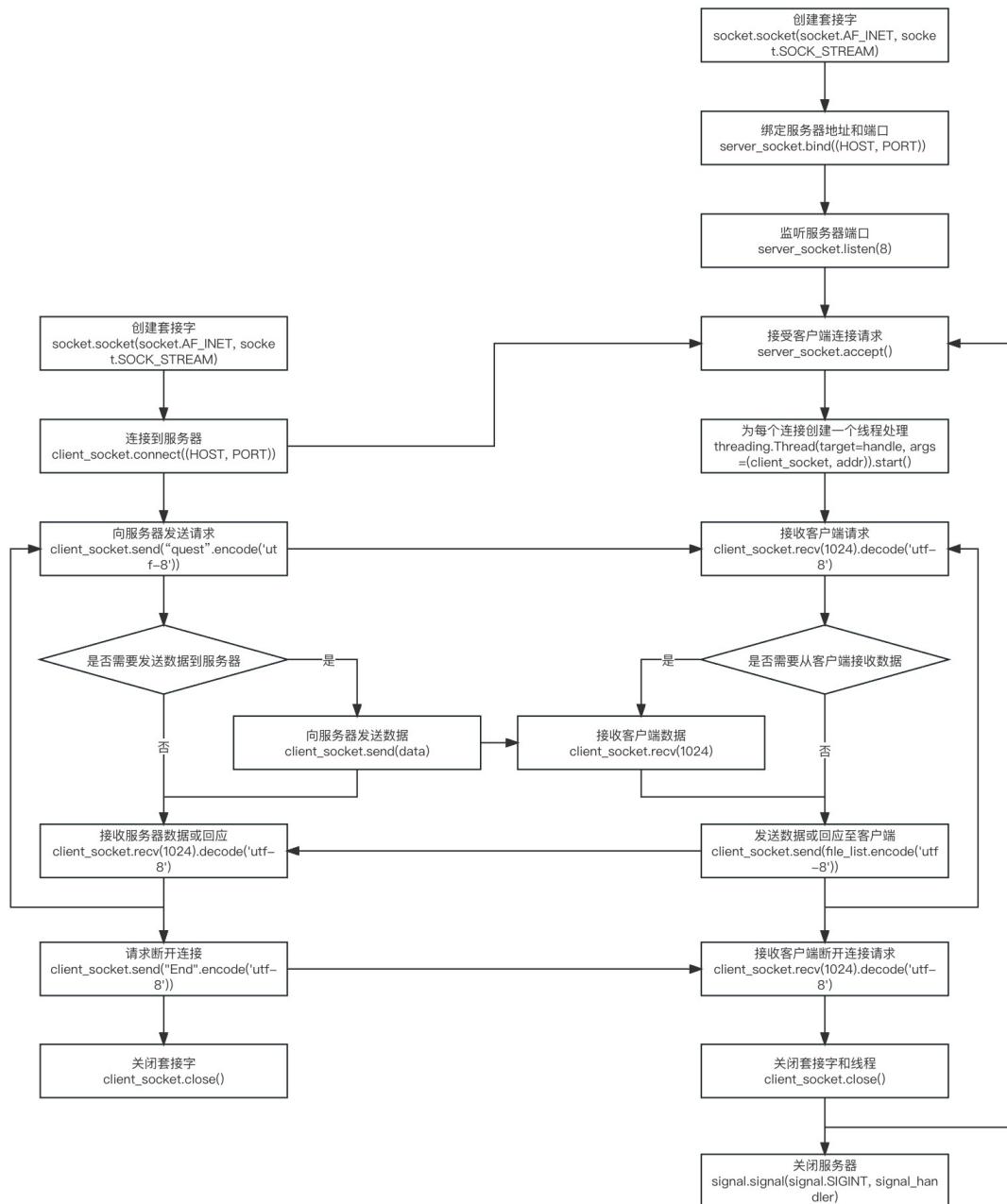


计算机网络 实验五

5.2 TCP 通信编程

一、软件设计流程图



二、源代码

1、客户端

```
import socket
import os

# 设置服务器地址和端口
HOST = '192.168.0.114'
PORT = 8080
folder_path = 'C:\\\\Users\\\\26396\\\\Desktop\\\\code_TCP\\\\client_file'

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# socket()函数用于创建套接字，用于进行网络通信。
# socket.socket(family, type, proto)
# family(地址族)：指定地址族，通常为：socket.AF_INET: IPv4 地址族，用于 Internet 协议。socket.AF_INET6: IPv6 地址族，用于 IPv6 协议。socket.AF_UNIX: 用于 Unix 域套接字通信。
# type (套接字类型)：指定套接字的类型，通常为：socket.SOCK_STREAM: TCP 套接字，提供可靠的流式连接。socket.SOCK_DGRAM: UDP 套接字，用于无连接、不可靠的数据传输。
# proto (协议)：通常可省略，根据 family 和 type 自动选择适当的协议。一般情况下，你不需要指定它。
client_socket.connect((HOST, PORT)) # 连接到服务器
# connect()用于建立到远程主机的连接，通常用于客户端与服务器之间的连接。
# socket.connect(address)
# address: 是一个包含远程主机地址和端口的元组，通常是一个由主机名和端口号组成的元组，例如 ('www.example.com', 80)

print("可选功能如下：")
print("1、查看文件列表")
print("2、上传文件")
print("3、下载文件")
print("4、断开连接并退出")

try:
    while True:
        choice = input("\n请输入编号选择功能：")

        if choice == '1': # 查看文件列表
            client_socket.send("List".encode('utf-8'))
            # send()用于将数据发送到连接的远程端点
            # socket.send(data)
            # data: 要发送的数据，通常是一个字节串 (bytes) 对象。

            # encode()用于将字符串 (string) 按照指定的编码方式转换为字节串 (bytes)，当需要在网络通信或文件读写等场景中处理文本数据时，encode()可以帮助将字符串编码为字节串，以便进行正确的传输或存储。
            # bytes = string.encode(encoding)
            # string: 要编码的字符串。
            # encoding: 指定的字符编码方式，如 'utf-8'、'latin-1' 等，用于将字符串转换为字节串的字符编码。

            # send()函数要求发送的数据必须是字节串 (bytes)，而不是普通的字符串 (string)。在 Python3 中，字符串默认是 Unicode 编码的，而套接字通常要求发送的数据是字节串，因此需要使用 encode()方法将字符串编码为字节串。

            file_list = client_socket.recv(1024).decode('utf-8')
            # recv()用于从已建立的套接字连接接收数据，常用于接收来自远程端点的数据。
            # data = socket.recv(buffer_size)
            # buffer_size: 一个整数，指定接收数据的缓冲区大小。通常它表示可以接收的最大数据量。

            print("当前服务器含有以下文件:")
            print(file_list)

        elif choice == '2': # 上传文件
            filename = input("请输入客户端文件夹中要上传的文件名: ")
            filepath = os.path.join(folder_path, filename)
```

```

if os.path.exists(filepath):
    client_socket.send(f"Upload {filename}".encode('utf-8'))
    with open(filepath, 'rb') as file: # 以二进制模式打开文件
        data = file.read() # 将文件的所有内容读入内存
        client_socket.send(data)
else:
    print("要上传的文件不存在")
    continue

response = client_socket.recv(1024).decode('utf-8')
print("服务器回应: ", response)

elif choice == '3': # 下载文件
    filename = input("请输入要从服务器下载的文件名: ")
    client_socket.send(f"Download {filename}".encode('utf-8'))

    data = client_socket.recv(1024)
    try:
        if data.decode('utf-8') == "文件不存在":
            print(f"服务器回应: 文件不存在")
        else:
            filepath = os.path.join(folder_path, filename)
            with open(filepath, 'wb') as file:
                file.write(data)
            print("文件已下载到客户端文件夹中")
    except:
        filepath = os.path.join(folder_path, filename)
        with open(filepath, 'wb') as file:
            file.write(data)
        print("文件已下载到客户端文件夹中")

elif choice == '4': # 断开连接并退出
    client_socket.send("End".encode('utf-8'))
    break

else:
    print("输入有误, 请重新选择")

except (ConnectionResetError, BrokenPipeError):
    print("服务器连接连接出错")

client_socket.close()
print("已关闭套接字")

```

2、服务器

```

import socket
import os
import threading
import signal
import logging

logger = logging.getLogger('ServerLogger') # 创建日志记录器
file_handler = logging.FileHandler('server.log') # 创建一个文件处理器
formatter = logging.basicConfig(level=logging.DEBUG, format='(时间:%(asctime)s 记录
器:%(name)s 线程:%(threadName)s) %(message)s') # 设置日志格式
file_handler.setFormatter(formatter)
file_handler.setLevel(logging.DEBUG) # 设置为 logging.DEBUG 保证了文件处理器会记录所有级别的日志信息
logger.addHandler(file_handler)

# 服务器配置
HOST = '192.168.0.114'
PORT = 8080 # 服务器端口
folder_path = 'C:\\\\Users\\\\26396\\\\Desktop\\\\code_TCP\\\\server_file'

def handle_client_connection(client_socket, addr): # 为每个连接创建一个新的线程

```

```

try:
    while True:
        request = client_socket.recv(1024).decode('utf-8') # 接收客户端请求
        print("")

        if request == 'List':
            logger.info(f"客户端{addr[0]}:{addr[1]}请求查看文件列表")

            files = os.listdir(folder_path) # 服务器的文件列表
            # os.listdir()用于获取指定目录中的文件和子目录列表，函数返回一个包含目录中所有文件和子目录名称的列表。
            file_list = os.listdir(path)
            # path: 是一个字符串，表示要列出文件和子目录的目标目录路径。
            file_list = "\n".join(files) # 将一个字符串列表 files 的各个元素连接组合成一个字符串，使用换行符分隔
            client_socket.send(file_list.encode('utf-8'))
            logger.info(f"已将文件列表传送给客户端{addr[0]}:{addr[1]}")

        elif request.startswith('Download'): # 处理文件下载请求
            filename = request.split(' ')[1]
            filepath = os.path.join(folder_path, filename)
            logger.info(f"客户端{addr[0]}:{addr[1]}请求下载文件{filename}")

            if os.path.exists(filepath):
                with open(filepath, 'rb') as file: # 以二进制模式打开文件
                    data = file.read()
                    client_socket.send(data)
                logger.info(f"已将文件{filename}传送给客户端{addr[0]}:{addr[1]}")
            else:
                client_socket.send(f"文件不存在".encode('utf-8'))
                logger.info(f"已回复客户端{addr[0]}:{addr[1]}文件{filename}不存在")

        elif request.startswith('Upload'): # 处理文件上传请求
            filename = request.split(' ')[1]
            filepath = os.path.join(folder_path, filename)
            logger.info(f"客户端{addr[0]}:{addr[1]}请求上传文件{filename}")

            data = client_socket.recv(1024)
            if not os.path.exists(filepath):
                with open(filepath, 'wb') as file: # 以二进制模式打开文件进行写入
                    file.write(data) # 将接收到的数据 data 写入到文件中
                client_socket.send(f"文件{filename}上传完成".encode('utf-8'))
                logger.info(f"已回复客户端{addr[0]}:{addr[1]}文件{filename}上传完成")
            else:
                with open(filepath, 'wb') as file:
                    file.write(data)
                client_socket.send(f"文件{filename}替换完成".encode('utf-8'))
                logger.info(f"已回复客户端{addr[0]}:{addr[1]}文件{filename}替换完成")

        elif request == 'End':
            logger.info(f"客户端{addr[0]}:{addr[1]}请求断开连接")
            break

        else:
            client_socket.send("请求有误请重新请求".encode('utf-8'))
            logger.info(f"客户端{addr[0]}:{addr[1]}请求有误")

except (ConnectionResetError, BrokenPipeError):
    logger.info(f"客户端{addr[0]}:{addr[1]}连接出错")

client_socket.close()
logger.info(f"已关闭客户端{addr[0]}:{addr[1]}的套接字")

def signal_handler(signum, frame): # 信号处理函数
    print("\n服务器程序结束")
    exit(0)

```

```
signal.signal(signal.SIGINT, signal_handler) # 设置信号处理程序

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket: # 创建欢迎套接字

    server_socket.bind((HOST, PORT)) # 绑定服务器地址和端口
    # bind()用于将套接字绑定到特定的地址（主机地址和端口），以便服务器可以在特定的主机和端口上监听客户端的连接请求。
    # socket.bind()
    # address: 是一个包含远程主机地址和端口的元组，通常是一个由主机名和端口号组成的元组，例如
    ('www.example.com', 80)
    server_socket.listen(8) # 监听最多 8 个连接请求
    # listen()用于在服务器端套接字上启动监听模式，以便接受客户端的连接请求。它通常在服务器端套接字绑定到地址后调用。
    # socket.listen()
    # backlog: 是一个整数，指定在拒绝连接之前，可以挂起的最大连接数量。它决定了服务器可以同时处理的最大连接数。
    print(f"正在监听{HOST}:{PORT}")

try:
    while True:
        client_socket, addr = server_socket.accept() # 接受客户端连接请求
        # accept()用于在服务器端接受客户端的连接请求，并返回一个新的套接字对象以便与客户端进行通信。
        # accept()方法没有参数，它会阻塞程序执行，直到有客户端发起连接请求。
        # 一旦接收到客户端的连接请求，accept()方法会返回一个新的套接字对象 client_socket，以及客户端的地址信息 client_address，服务器可以使用这个新的套接字与客户端进行数据交换。
        print('')
        logger.info(f"接受到来自{addr[0]}:{addr[1]}的连接")
        # 为每个连接创建一个新线程
        threading.Thread(target=handle_client_connection, args=(client_socket,
        addr)).start()

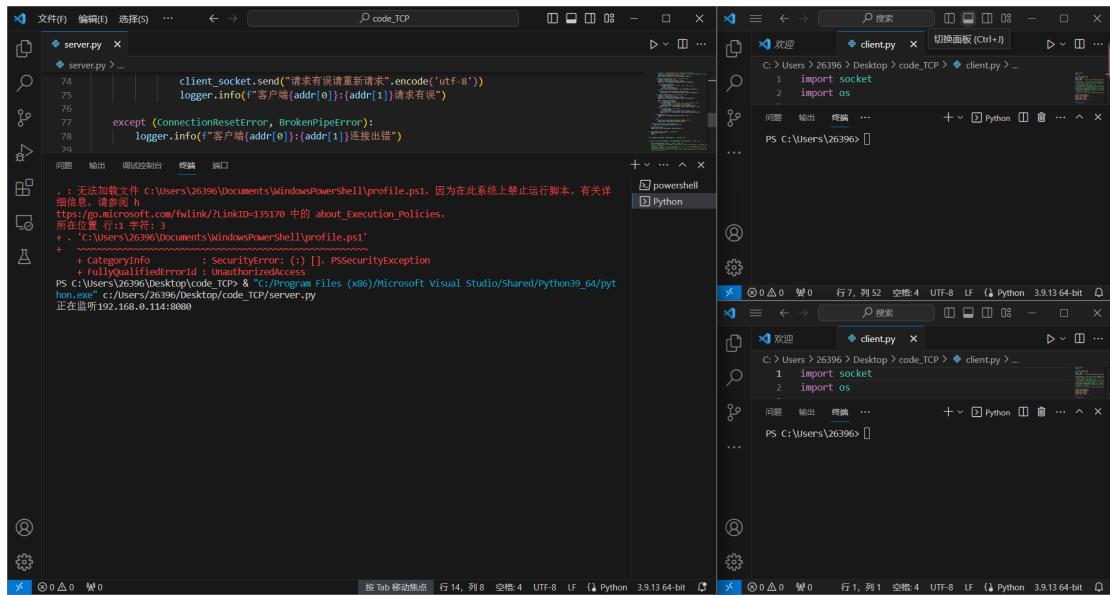
except KeyboardInterrupt: # 如果收到键盘中断信号（通常是 Ctrl+C），执行信号处理函数
    signal_handler(signal.SIGINT, None)
```



三、测试结果及运行日志

1、A 机器运行服务器软件

在左侧窗口运行服务器软件



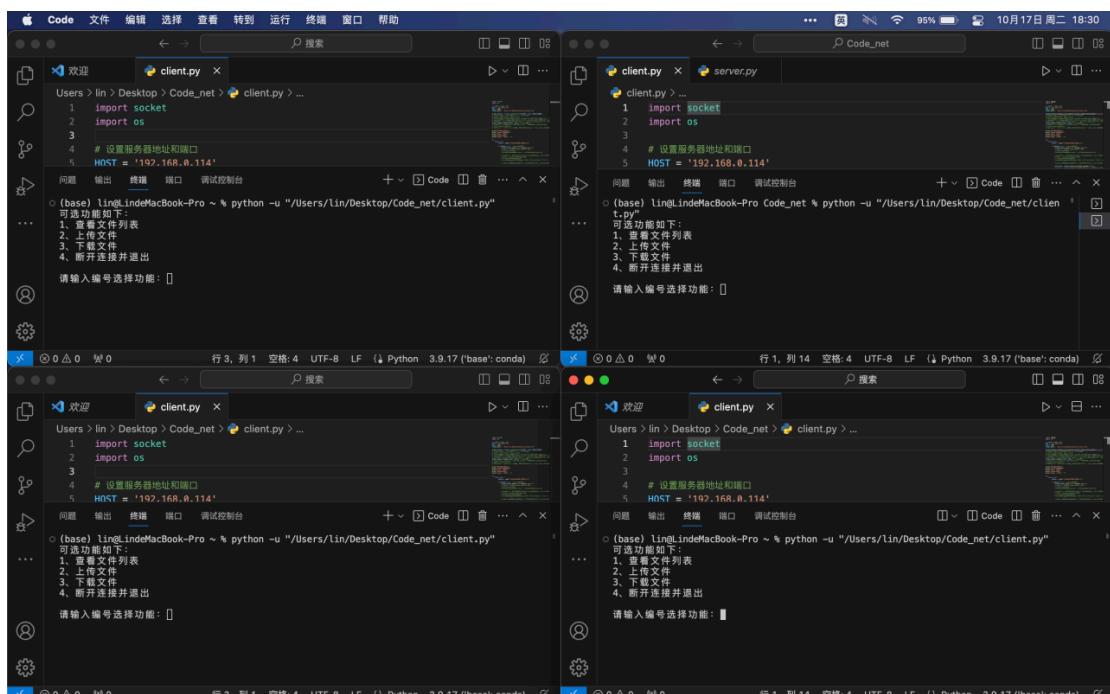
```
server.py
74     client_socket.send("请求有误请重新请求".encode('utf-8'))
75     logger.info("客户端[%s]:[%s]请求有误",addr[0],addr[1])
76 except (ConnectionResetError, BrokenPipeError):
77     logger.info("客户端[%s]:[%s]连接出错",addr[0],addr[1])
```

```
client.py
1 import socket
2 import os
```

可以看到程序输出正在监听服务器地址

2、在客户端 B 机器的命令行同时运行 4 个客户端

在 B 机器同时运行 4 个客户端：



```
client.py
1 import socket
2 import os
3
4 # 设置服务器地址和端口
5 HOST = '192.168.0.114'
```

```
client.py
1 import socket
2 import os
3
4 # 设置服务器地址和端口
5 HOST = '192.168.0.114'
```

```
client.py
1 import socket
2 import os
3
4 # 设置服务器地址和端口
5 HOST = '192.168.0.114'
```

```
client.py
1 import socket
2 import os
3
4 # 设置服务器地址和端口
5 HOST = '192.168.0.114'
```

可以看到 4 个客户端均连接成功

A 机器左侧服务器窗口输出：

The screenshot shows two side-by-side instances of Microsoft Visual Studio Code. The left instance is running a Python server application named 'server.py'. Its terminal output shows four connection requests from different clients. The right instance is running a Python client application named 'client.py'. Its terminal output shows the client connecting to the server.

```
server.py terminal output:
: 无法加载文件 C:\Users\26396\Documents\WindowsPowerShell\profile.ps1, 因为此系统上禁止运行脚本。有关详细信息，请参阅 https://go.microsoft.com/fwlink/?LinkId=135170 中的 about_Execution_Policies。
所在位置 行:1 字符:3
+ . 'C:\Users\26396\Documents\WindowsPowerShell\profile.ps1'
+ ~~~~CategoryInfo : SecurityError: () [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\26396\Desktop\code_TCP & "C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python39_64\python.exe" c:/Users/26396/Desktop/code_TCP/server.py
正在监听 192.168.0.114:8080

(时间:2023-10-17 18:30:28,918 记录器:ServerLogger 线程:MainThread) 接受到来自192.168.0.112:58516的连接
(时间:2023-10-17 18:30:30,974 记录器:ServerLogger 线程:MainThread) 接受到来自192.168.0.112:58520的连接
(时间:2023-10-17 18:30:36,301 记录器:ServerLogger 线程:MainThread) 接受到来自192.168.0.112:58539的连接
(时间:2023-10-17 18:30:37,163 记录器:ServerLogger 线程:MainThread) 接受到来自192.168.0.112:58528的连接

client.py terminal output:
欢迎
C:\Users>26396>Desktop>code_TCP>client.py ...
1 import socket
2 import os
问题 输出 终端 ... + v Python 假 ...
PS C:\Users\26396>[]
```

可以看到 A 机器服务器记录了 4 个客户端的连接请求。

3、在 A 机器也同时运行 2 个客户端软件

在 A 机器右侧窗口同时运行 2 个客户端：

The screenshot shows two side-by-side instances of Microsoft Visual Studio Code. The left instance is running a Python server application named 'server.py'. Its terminal output shows four connection requests from different clients. The right instance is running a Python client application named 'client.py'. Its terminal output shows the client connecting to the server.

```
server.py terminal output:
: 无法加载文件 C:\Users\26396\Documents\WindowsPowerShell\profile.ps1, 因为此系统上禁止运行脚本。有关详细信息，请参阅 https://go.microsoft.com/fwlink/?LinkId=135170 中的 about_Execution_Policies。
所在位置 行:1 字符:3
+ . 'C:\Users\26396\Documents\WindowsPowerShell\profile.ps1'
+ ~~~~CategoryInfo : SecurityError: () [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\26396\Desktop\code_TCP & "C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python39_64\python.exe" c:/Users/26396/Desktop/code_TCP/server.py
正在监听 192.168.0.114:8080

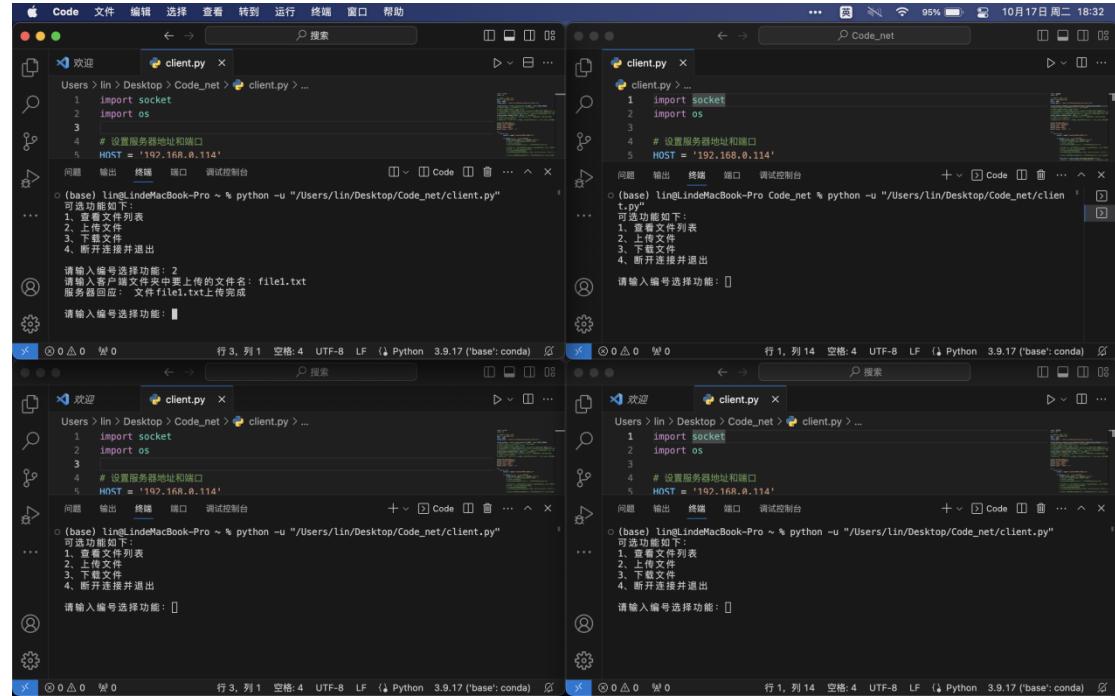
(时间:2023-10-17 18:30:28,918 记录器:ServerLogger 线程:MainThread) 接受到来自192.168.0.112:58516的连接
(时间:2023-10-17 18:30:30,974 记录器:ServerLogger 线程:MainThread) 接受到来自192.168.0.112:58520的连接
(时间:2023-10-17 18:30:36,301 记录器:ServerLogger 线程:MainThread) 接受到来自192.168.0.112:58539的连接
(时间:2023-10-17 18:30:37,163 记录器:ServerLogger 线程:MainThread) 接受到来自192.168.0.112:58528的连接

client.py terminal output:
欢迎
C:\Users>26396>Desktop>code_TCP>client.py ...
1 import socket
2 import os
问题 输出 终端 ... + v Python 假 ...
PS C:\Users\26396>[]
```

可以看到，右侧两个客户端均连接成功，左侧服务器软件的窗口记录了这两个客户端的连接记录。

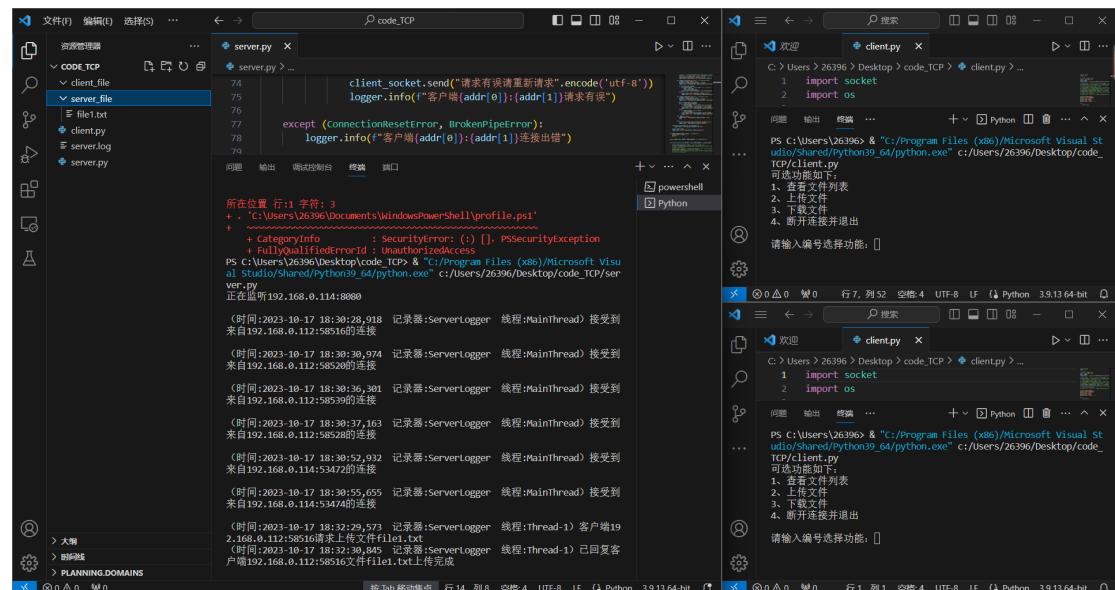
4、客户端运行测试

在 B 机器左上客户端窗口运行上传文件功能，将 file1.txt 上传至服务器：



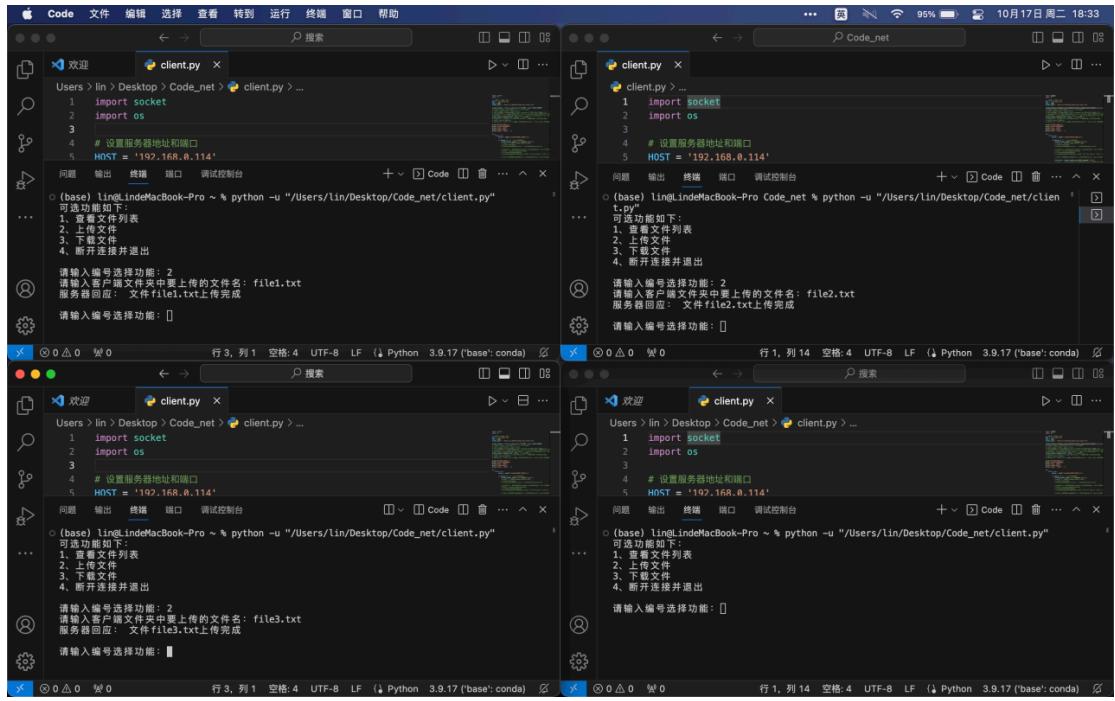
可以看到左上客户端窗口显示上传完成。

A 机器左侧服务器窗口输出：



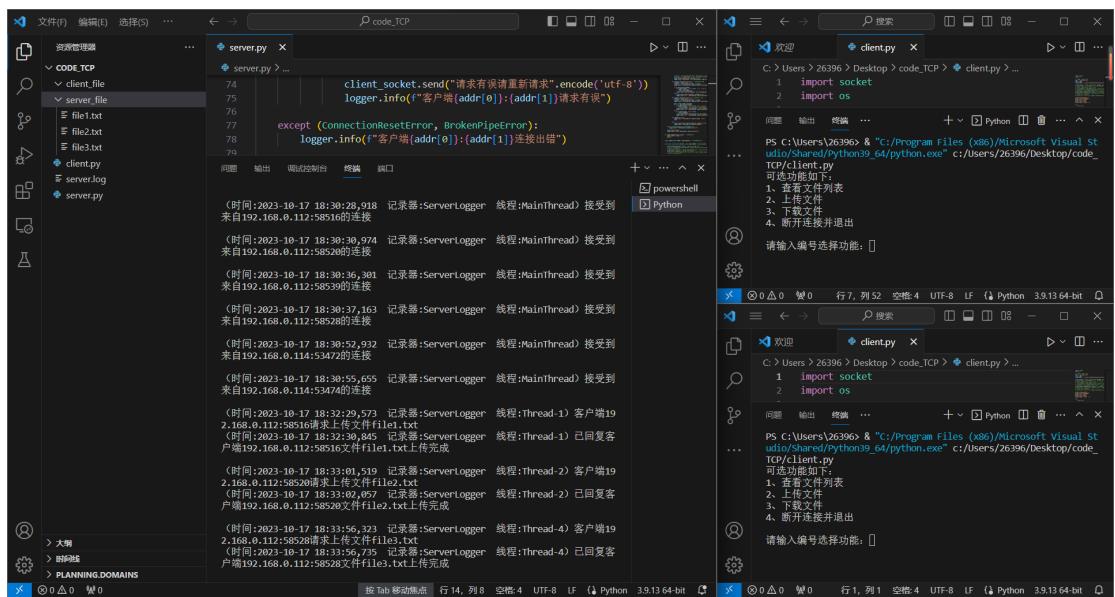
可以看到，服务器记录了客户端的文件上传请求，并在文件接收完成后对客户端回应文件上传完成。

在 B 机器右上客户端窗口运行上传文件功能，将 file2.txt 上传至服务器。在 B 机器左下客户端窗口运行上传文件功能，将 file3.txt 上传至服务器：



可以看到两个文件均上传成功。

A 机器左侧服务器窗口输出：



可以看到，两次上传均被服务器响应并记录。

在 B 机器右下客户端窗口请求查看服务器当前文件列表：

The screenshot shows five terminal windows arranged in a grid. The top row contains two windows, and the bottom row contains three windows. All windows are running Python code related to file transfer. The code in the windows includes imports for socket and os, and defines a HOST variable as '192.168.0.114'. The windows show various stages of file upload and download, with messages like '请输入编号选择功能' (Please enter number to select function) and '服务器回应：文件file1.txt上传完成' (Server response: file1.txt upload completed). The windows are titled 'client.py' and 'Code_ne...'.

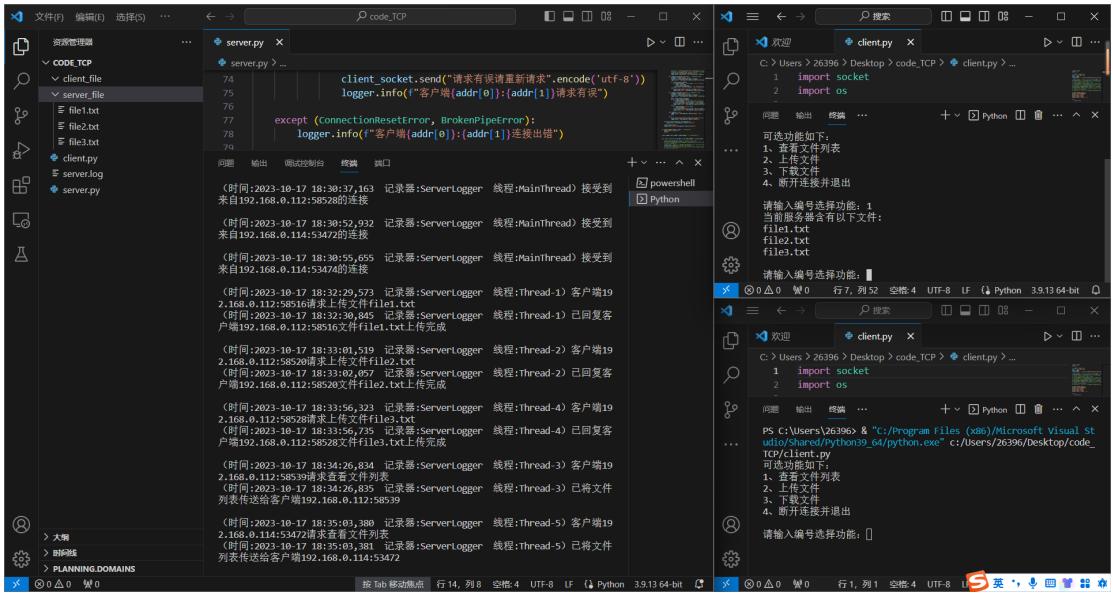
可以看到，刚才上传的三个文件均在服务器的文件列表中。

A 机器左侧服务器窗口输出：

The screenshot shows two terminal windows on machine A. The left window is a code editor for 'server.py' which handles file transfers. The right window is a terminal window showing the server's log output. The log shows multiple connections from the client IP '192.168.0.112' and the server's own IP '192.168.0.114'. It logs file transfers for 'file1.txt', 'file2.txt', and 'file3.txt', along with other connection-related messages. The terminal window title is '欢迎' (Welcome).

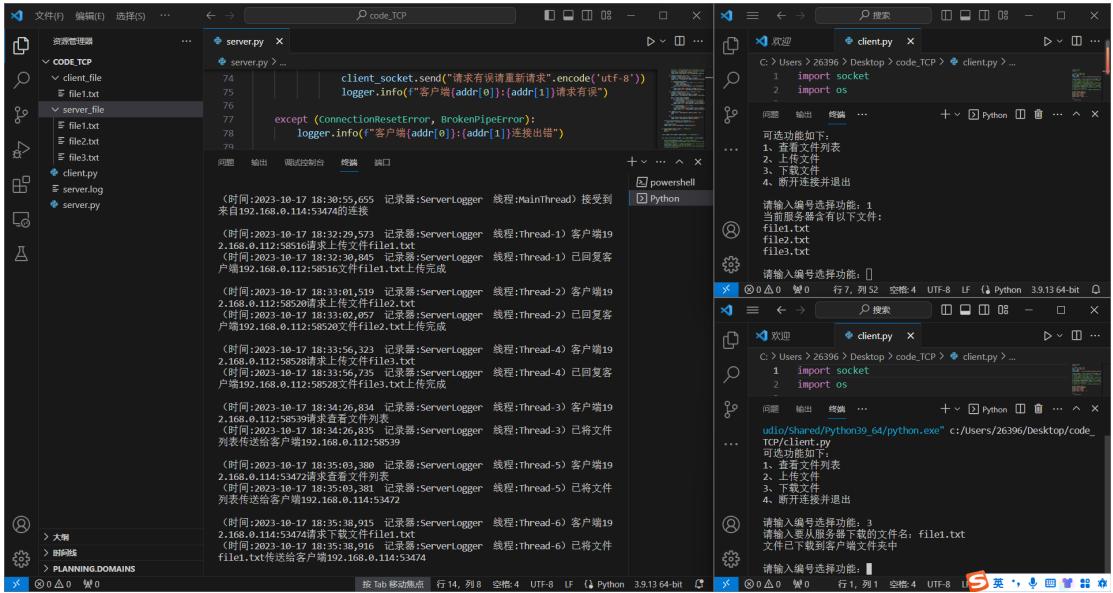
可以看到，服务器记录了客户端请求文件列表，并将文件列表传给了客户端。

在 A 机器右上客户端窗口请求查看服务器当前文件列表:



可以看到，右上客户端窗口成功查看了文件列表，左侧服务器窗口响应并记录了请求。

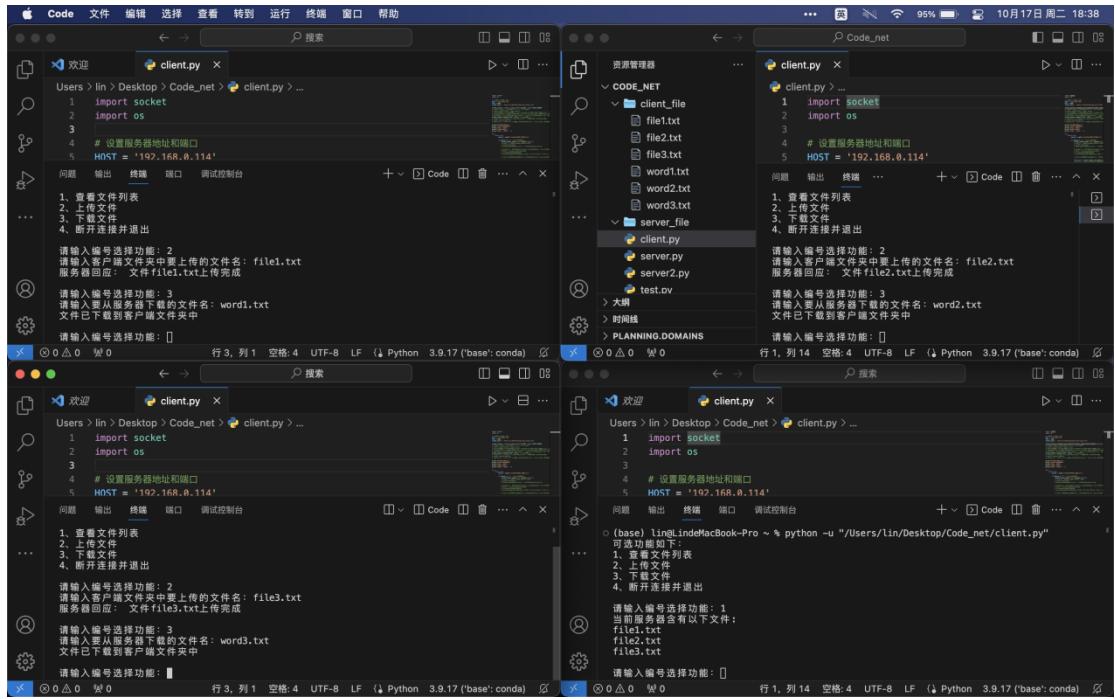
在 A 机器右下客户端窗口运行下载文件功能，将 file1.txt 下载至客户端。



可以看到，右下客户端窗口显示文件已下载到客户端文件夹中。在最左边资源管理器中可以看到，在 client_file 文件夹中出现了 file1.txt 文件。

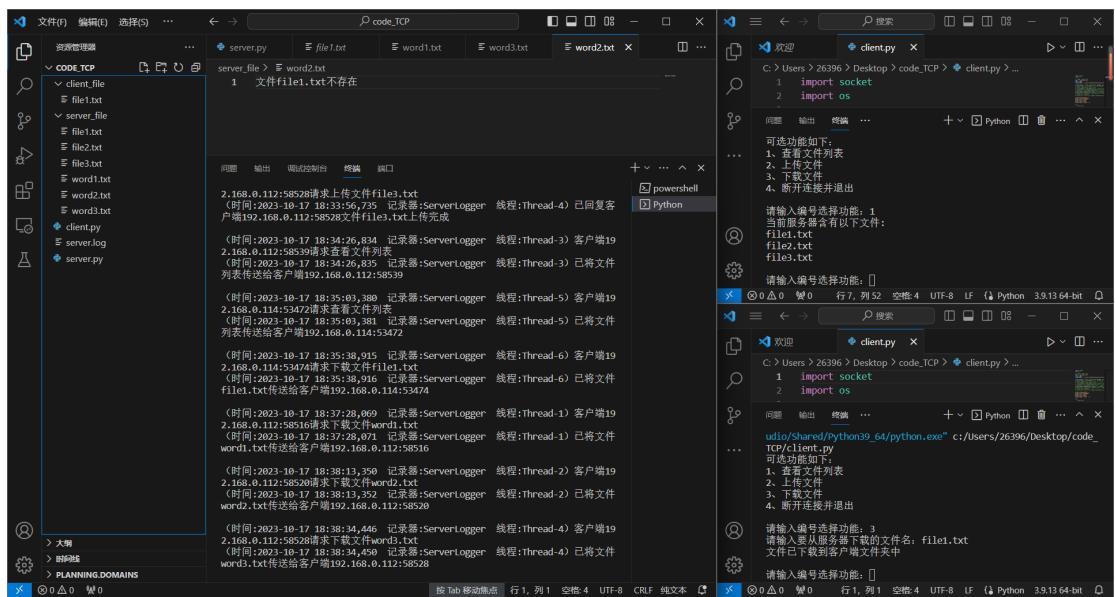
在左侧服务器窗口中，服务器记录了下载请求，并将文件传送给客户端。

在 B 机器左上、右上、左下客户端窗口运行下载文件功能，分别将 word1.txt、word2.txt、word3.txt 下载至客户端：



可以看到在 B 机器上，三个文件均下载成功，在资源管理器中也显示了相应文件。

A 机器左侧服务器窗口输出：



可以看到服务器响应并记录了上面三次文件下载请求。

在 A 机器右上客户端窗口运行上传文件功能，将 file1.txt 上传至服务器：

The screenshot shows two terminal windows. The left window is titled 'code_TCP' and shows logs from the server. It includes entries for file transfers between the client and server, such as 'file1.txt' being uploaded and replaced by the client. The right window is titled 'client.py' and shows the client's perspective. It displays a message asking for a file to be uploaded, which the client has done ('file1.txt replaced').

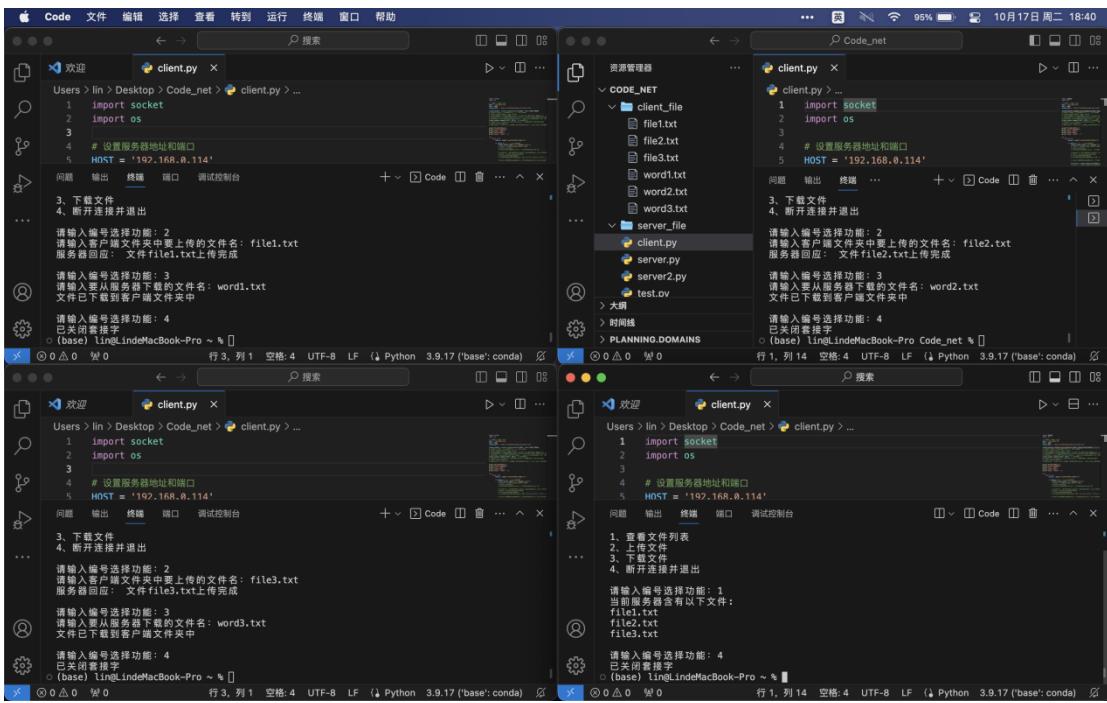
可以看到右上角客户端窗口显示文件替换完成，由于 file1.txt 之前已在客户端中存在，所以再次上传成功将文件替换。在左侧服务器窗口记录了上传请求并回应了文件替换成功。

在 A 机器右下客户端窗口运行下载文件功能，尝试下载不存在的文件 file4.txt：

The screenshot shows two terminal windows. The left window is titled 'code_TCP' and shows logs from the server. It includes entries for file requests from the client, such as 'file4.txt' being requested. The right window is titled 'client.py' and shows the client's perspective. It displays a message asking for a file to be downloaded, which the client has done ('file4.txt does not exist').

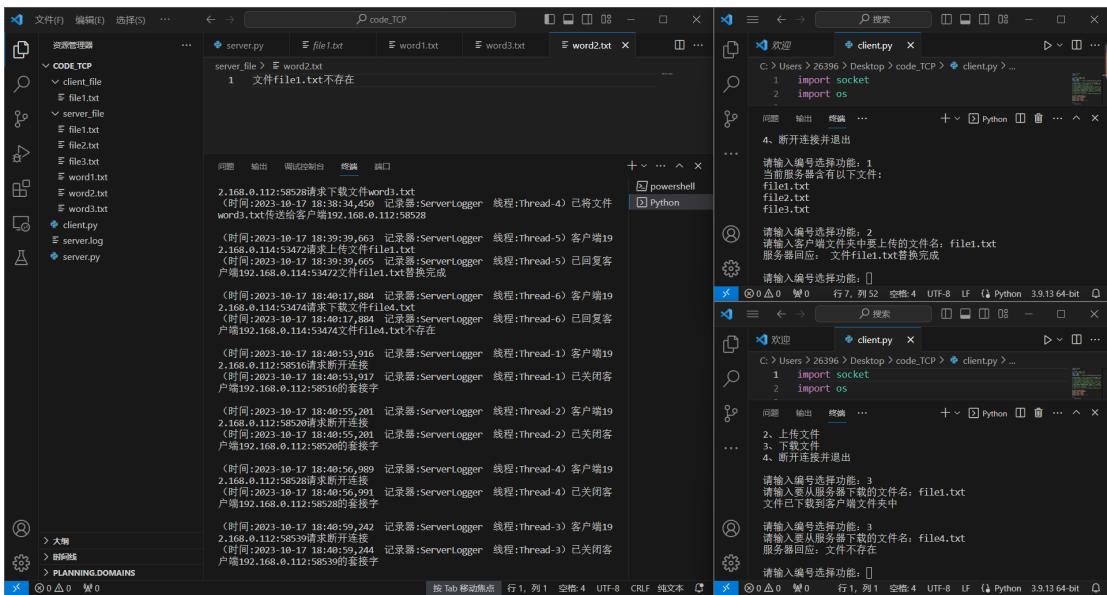
可以看到，在右下客户端窗口显示服务器回应文件不存在。在左侧服务器窗口记录了对 file4.txt 的下载请求，并回应了文件不存在。

关闭 B 机器的 4 个客户端：



可以看到 4 个客户端窗口均将连接断开并结束运行。

A 机器左侧服务器窗口输出：



可以看到左侧服务器窗口记录了四个断开连接的请求，并将对应套接字关闭。

关闭 A 机器的 2 个客户端：

The image shows two separate windows of the Windows Task Manager. The left window lists processes for 'code.TCP': 'server.py' (PID 26396), 'file1.txt' (PID 26396), 'word1.txt' (PID 26396), 'word3.txt' (PID 26396), and 'word2.txt' (PID 26396). The right window lists processes for 'client.py': 'client.py' (PID 26397), 'file1.txt' (PID 26397), 'file2.txt' (PID 26397), and 'file3.txt' (PID 26397). Both windows show the same system information at the bottom: CPU: 0.0%, RAM: 0.0%, Disk: 0.0%, and Network: 0.0%.

可以看到也均关闭成功。

5、观看服务器端运行的日志



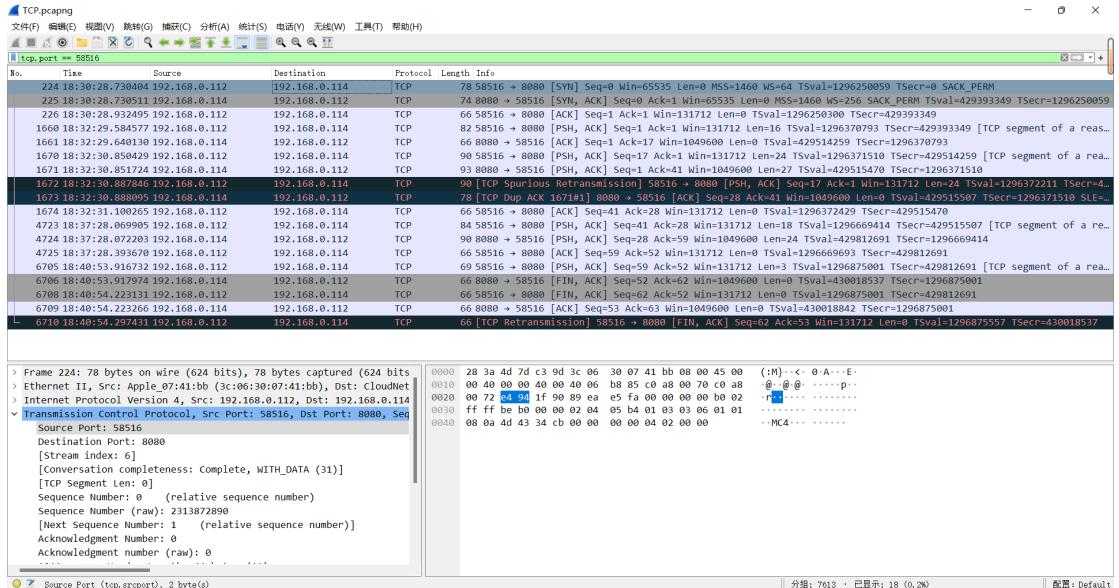
```
文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) ... ← → code_TCP
server.py server.log
21 接受到来自 192.168.0.112:57954 的连接
22 接受到来自 192.168.0.112:57968 的连接
23 接受到来自 192.168.0.112:57979 的连接
24 接受到来自 192.168.0.114:53410 的连接
25 接受到来自 192.168.0.114:53420 的连接
26 接受到来自 192.168.0.112:58516 的连接
27 接受到来自 192.168.0.112:58520 的连接
28 接受到来自 192.168.0.112:58530 的连接
29 接受到来自 192.168.0.112:58528 的连接
30 接受到来自 192.168.0.114:53427 的连接
31 接受到来自 192.168.0.114:53474 的连接
32 客户端 192.168.0.112:58516 请求上传文件 file1.txt
33 已回复客户端 192.168.0.112:58516 文件 file1.txt 上传完成
34 客户端 192.168.0.112:58520 请求上传文件 file2.txt
35 已回复客户端 192.168.0.112:58520 文件 file2.txt 上传完成
36 客户端 192.168.0.112:58528 请求上传文件 file3.txt
37 已回复客户端 192.168.0.112:58528 文件 file3.txt 上传完成
38 客户端 192.168.0.112:58539 请求查看文件列表
39 已将文件列表发送给客户端 192.168.0.112:58539
40 客户端 192.168.0.112:58542 请求查看文件列表
41 已将文件列表发送给客户端 192.168.0.112:58542
42 客户端 192.168.0.114:53410 请求下载文件 file1.txt
43 已将文件 file1.txt 传输给客户端 192.168.0.114:53474
44 客户端 192.168.0.112:58516 请求下载文件 file1.txt
45 已将文件 word1.txt 传输给客户端 192.168.0.112:58516
46 客户端 192.168.0.112:58520 请求下载文件 word2.txt
47 已将文件 word2.txt 传输给客户端 192.168.0.112:58520
48 客户端 192.168.0.112:58528 请求下载文件 word3.txt
49 已将文件 word3.txt 传输给客户端 192.168.0.112:58528
50 客户端 192.168.0.114:53472 请求下载文件 file1.txt
51 已回复客户端 192.168.0.114:53472 文件 file1.txt 替换完成
52 客户端 192.168.0.114:53474 请求下载文件 file4.txt
53 已回复客户端 192.168.0.114:53474 文件 file4.txt 不存在
54 客户端 192.168.0.112:58516 请求断开连接
55 已关闭客户端 192.168.0.112:58516 的套接字
56 客户端 192.168.0.112:58520 请求断开连接
57 已关闭客户端 192.168.0.112:58520 的套接字
```

可以看到，上面测试过程中，左侧服务器窗口的输出也被保存到了日志文件中。

6、打开 wireshark 观察、确认服务器端和客户端的传输层协议交互过程。

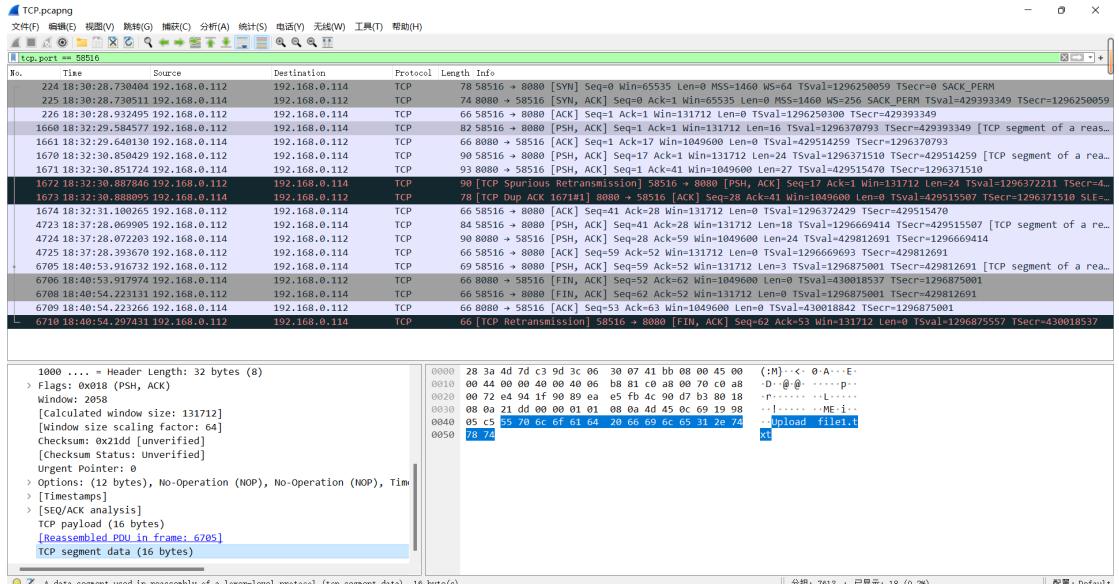
对 B 机器的第一个客户端进行分析，其端口号为 58516

(1) 建立 TCP 连接



在 18:30 时，我们在 B 机器上运行了第一个客户端程序，可以看到第 224 至 226 号 TCP 报文对应 TCP 连接的建立，即“三次握手”的过程。

(2) 上传文件



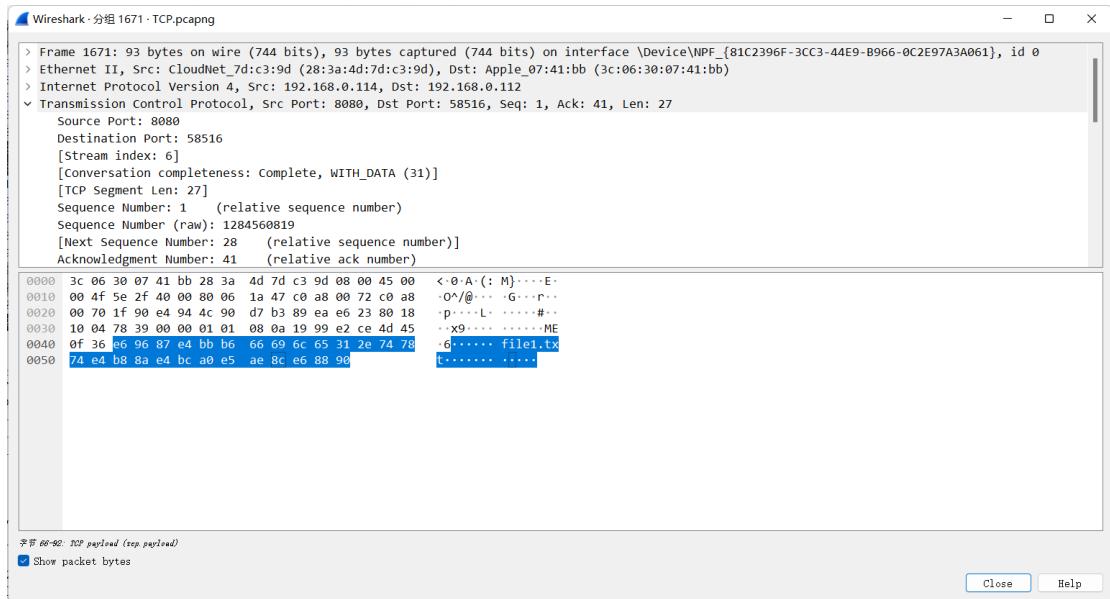
在 18:32 时，我们在 B 机器上请求上传文件 file1.txt 至服务器，可以看到第 1660 至 1674 号 TCP 报文对应文件上传过程。

可以看到，在上图显示的 1660 号报文中，数据段的内容为：Upload file1.txt，这与客户端程序的设计相对应。

1661 号报文为服务器对收到了客户端数据的确认，序列号和确认号与 1660 号报文的交互相
关联。

1670 号报文将 file1.txt 文件的二进制数据传输给服务器。

1671 号报文为对 1670 号报文的确认，并将服务器的回应发送给客户端。



可以看到 1671 号报文数据段的值为：

e69687e4bbb666696c65312e747874e4b88ae4bca0e5ae8ce68890

```
文件(F) 编辑(E) 选择(S) ... ← → 🔍 code_TCP
```

```
server.py test.py client.py
```

```
test.py > ...
1 # 给定的十六进制字符串
2 hex_string = "e69687e4bbb666696c65312e747874e4b88ae4bca0e5ae8ce68890"
3
4 # 将十六进制字符串转换为字节对象
5 byte_string = bytes.fromhex(hex_string)
6
7 # 解码字节对象为UTF-8编码的字符串
8 decoded_string = byte_string.decode('utf-8')
9
10 # 打印解码后的字符串
11 print(decoded_string)
12
```

```
问题 输出 调试控制台 终端 端口
```

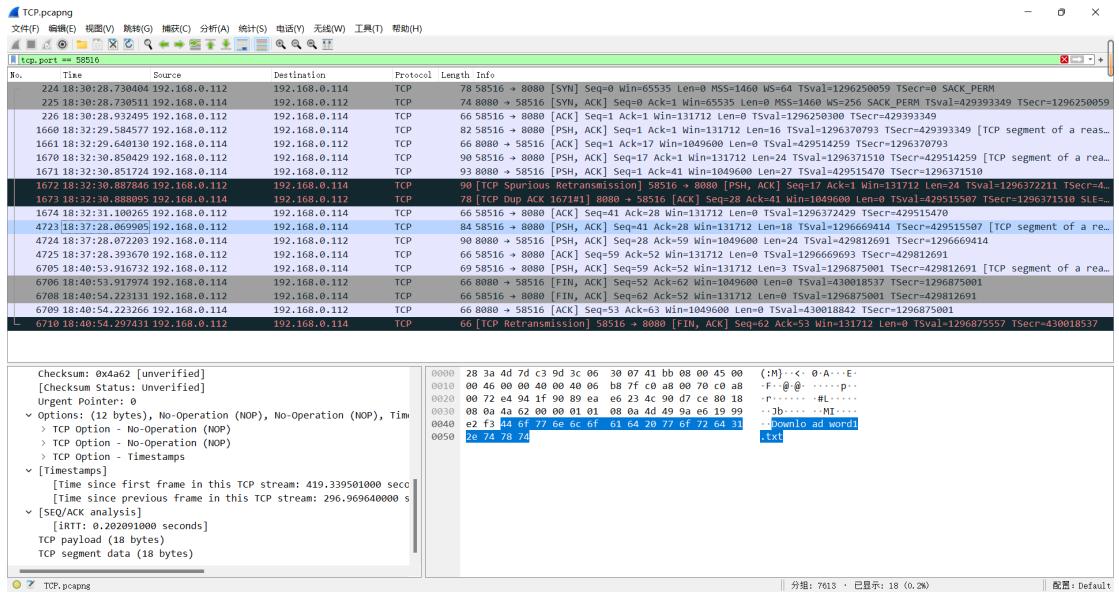
```
PS C:\Users\6396\Desktop\code_TCP> & "C:/Program Files (x86)/Microsoft Visual Studio/Shared/Python39_64/python.exe"
" c:/Users/26396/Desktop/code_TCP/test.py
文件file1.txt上传完成
PS C:\Users\6396\Desktop\code_TCP> [
```

我们用一个简单程序对其解码，可以看到解码后的内容为：文件 file1.txt 上传完成
这就是服务器对客户端上传 file1.txt 的回应，与服务器程序的设计相对应。

1672 号报文被怀疑是不必要的重传，因为服务器已经收到了客户端的确认。1673 号报文被
认为是一个重复的确认，这是由于 1672 号报文的重传引起的。

最后 1674 号报文为对 1671 号报文的确认，其 ACK 分析显示[This is an ACK to the segment in
frame: 1671]

(3) 下载文件



在 18:37 时，我们在 B 机器上请求下载文件 word1.txt，可以看到第 4723 至 4725 号 TCP 报文对应文件下载过程。

可以看到，在上图显示的 4723 号报文中，数据段的内容为：Download word1.txt，这与客户端程序的设计相对应。

4724 号报文将文件 word1.txt 的二进制数据传送给客户端。

4725 号报文为对 4724 号报文的确认。

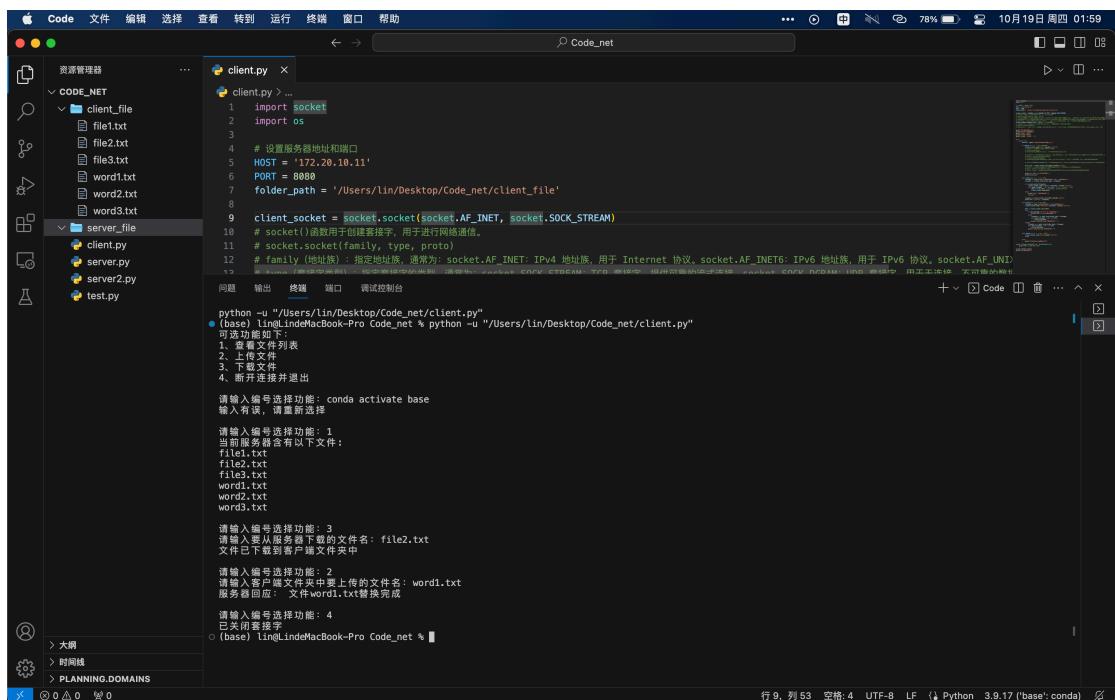


计算机网络 实验六

实验 6.1.2

一、在客户端、服务器端运行 实验 5.2 的程序

客户端：



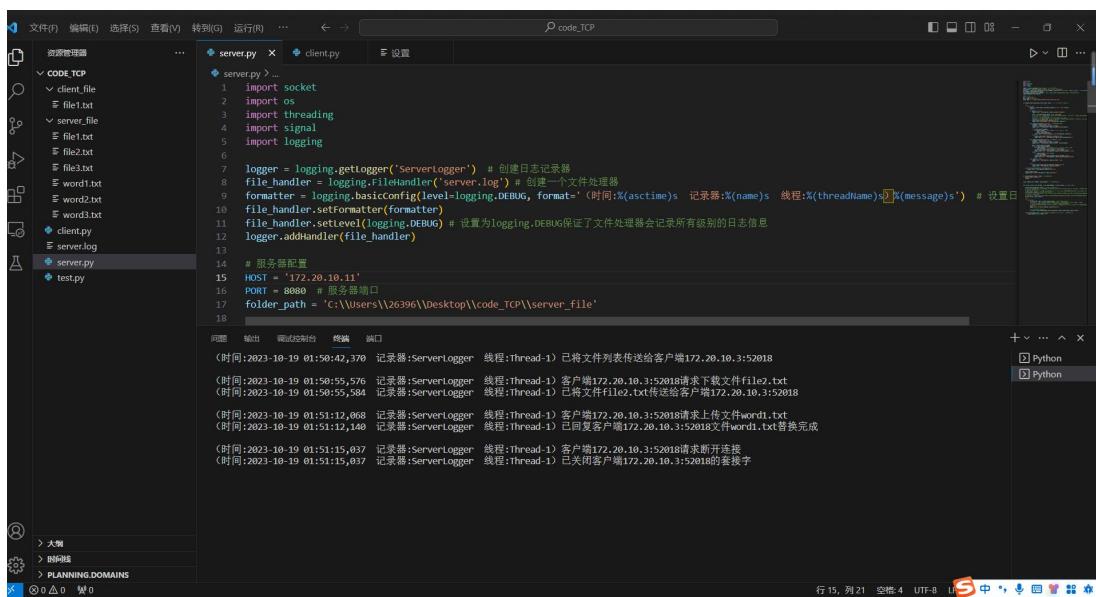
```
client.py > ...
1: import socket
2: import os
3:
4: # 设置服务器地址和端口
5: HOST = '172.20.10.11'
6: PORT = 8080
7: folder_path = '/Users/lin/Desktop/Code_net/client_file'
8:
9: client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10: # socket()函数用于创建套接字，用于进行网络通信。
11: # socket.socket(family, type, proto)
12: # family (地址族)：指定地址族，通常为：socket.AF_INET: IPv4 地址族，用于 Internet 协议。socket.AF_INET6: IPv6 地址族，用于 IPv6 协议。socket.AF_UNIX
13: # type (类型)：指定套接字的类型，通常为：socket.SOCK_STREAM - TCP 协议；socket.SOCK_DGRAM - UDP 协议；socket.SOCK_RAW - 无连接协议；socket.SOCK_RDM - 大对象的传输
14:
15: python -u "/Users/lin/Desktop/Code_net/client.py"
(base) lin@LindeMacBook-Pro Code_net % python -u "/Users/lin/Desktop/Code_net/client.py"
1: 选择功能如下:
2: 1. 查看文件列表
3: 2. 上传文件
4: 3. 下载文件
5: 4. 断开连接并退出
请输入编号选择功能: 1
当前服务器含有以下文件:
file1.txt
file2.txt
file3.txt
word1.txt
word2.txt
word3.txt

请输入编号选择功能: 3
请输入要从服务器下载的文件名: file2.txt
文件已下载到客户端文件夹中

请输入编号选择功能: 2
请输入客户端文件夹中要上传的文件名: word1.txt
服务器回应: 文件word1.txt替换完成

请输入编号选择功能: 4
已关闭连接
(base) lin@LindeMacBook-Pro Code_net %
```

服务器：

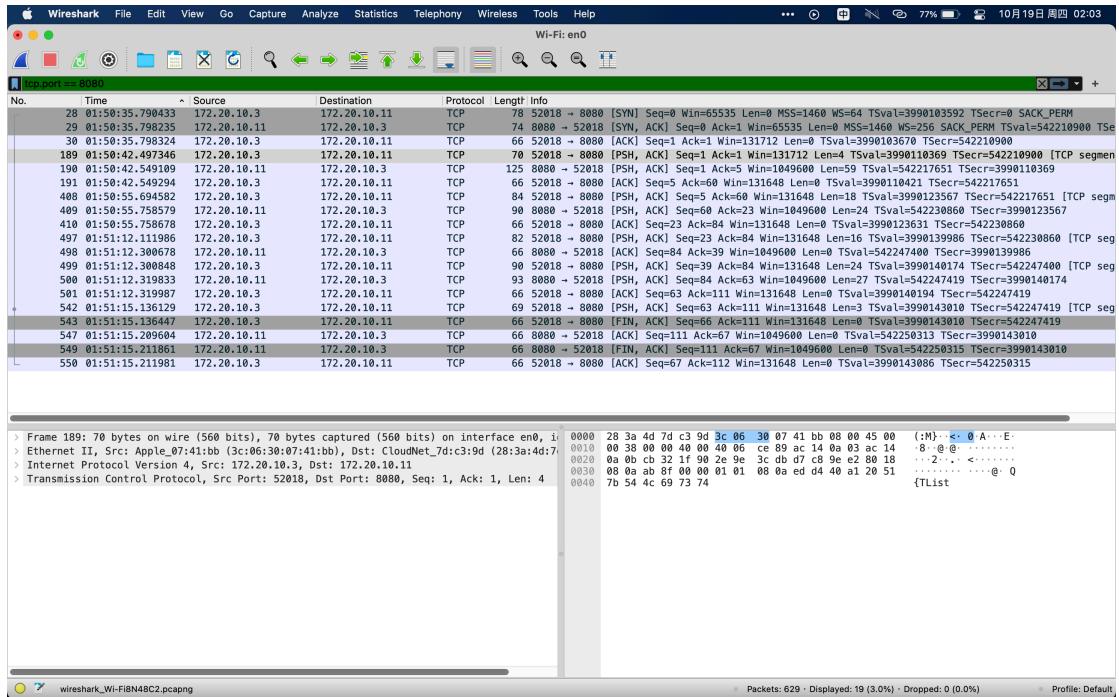


```
server.py > ...
1: import socket
2: import os
3: import threading
4: import signal
5: import logging
6:
7: logger = logging.getLogger('ServerLogger') # 创建日志记录器
8: file_handler = logging.FileHandler('server.log') # 创建一个文件处理器
9: formatter = logging.basicConfig(level=logging.DEBUG, format='[时间:%(asctime)s] 记录器:%(name)s 线程:%(threadName)s %(message)s') # 设置日志格式
10: file_handler.setFormatter(formatter)
11: file_handler.setLevel(logging.DEBUG) # 设置为logging.DEBUG保证了文件处理器会记录所有级别的日志信息
12: logger.addHandler(file_handler)
13:
14: # 服务端配置
15: HOST = '172.20.10.11'
16: PORT = 8080 # 服务端端口
17: folder_path = 'C:\Users\26396\Desktop\code_TCP\server_file'
18:

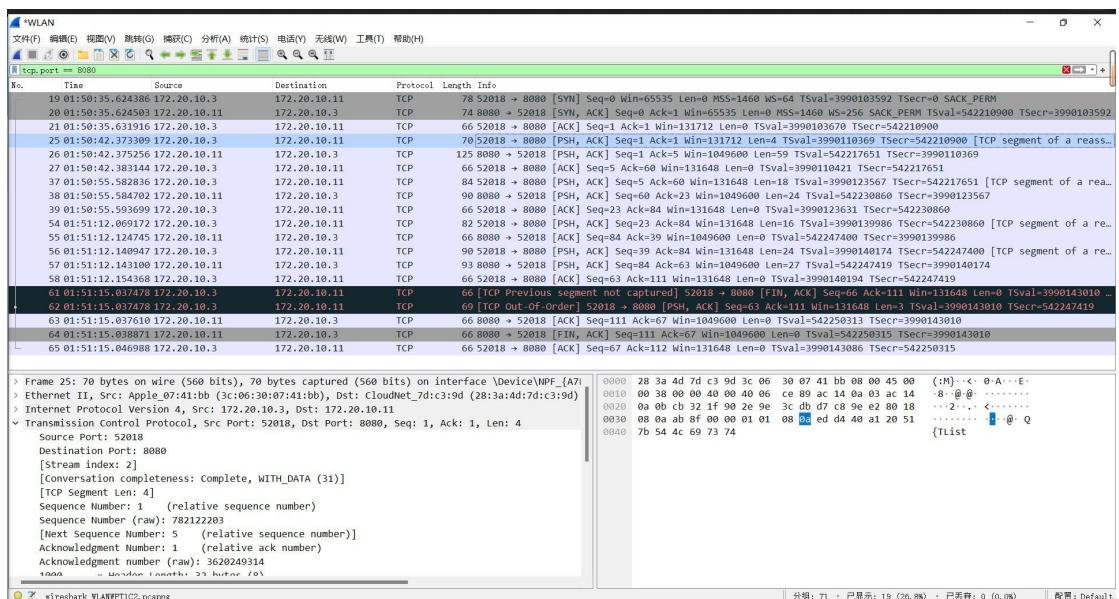
(base) lin@LindeMacBook-Pro code_TCP % python server.py
[时间:2023-10-19 01:50:42,370] 记录器:ServerLogger 线程:Thread-1 已将文件列表送给客户端172.20.10.3:52018
[时间:2023-10-19 01:50:55,576] 记录器:ServerLogger 线程:Thread-1 客户端172.20.10.3:52018请求下载文件file2.txt
[时间:2023-10-19 01:50:55,584] 记录器:ServerLogger 线程:Thread-1 已将文件file2.txt传送给客户端172.20.10.3:52018
[时间:2023-10-19 01:51:12,068] 记录器:ServerLogger 线程:Thread-1 客户端172.20.10.3:52018请求上传文件word1.txt
[时间:2023-10-19 01:51:12,148] 记录器:ServerLogger 线程:Thread-1 已回复客户端172.20.10.3:52018文件word1.txt替换完成
[时间:2023-10-19 01:51:15,037] 记录器:ServerLogger 线程:Thread-1 客户端172.20.10.3:52018请求断开连接
[时间:2023-10-19 01:51:15,037] 记录器:ServerLogger 线程:Thread-1 已关闭客户端172.20.10.3:52018的连接
```

二、在支持 5.2 实验的 2 个机器上都启动 wireshark 抓包

客户端：

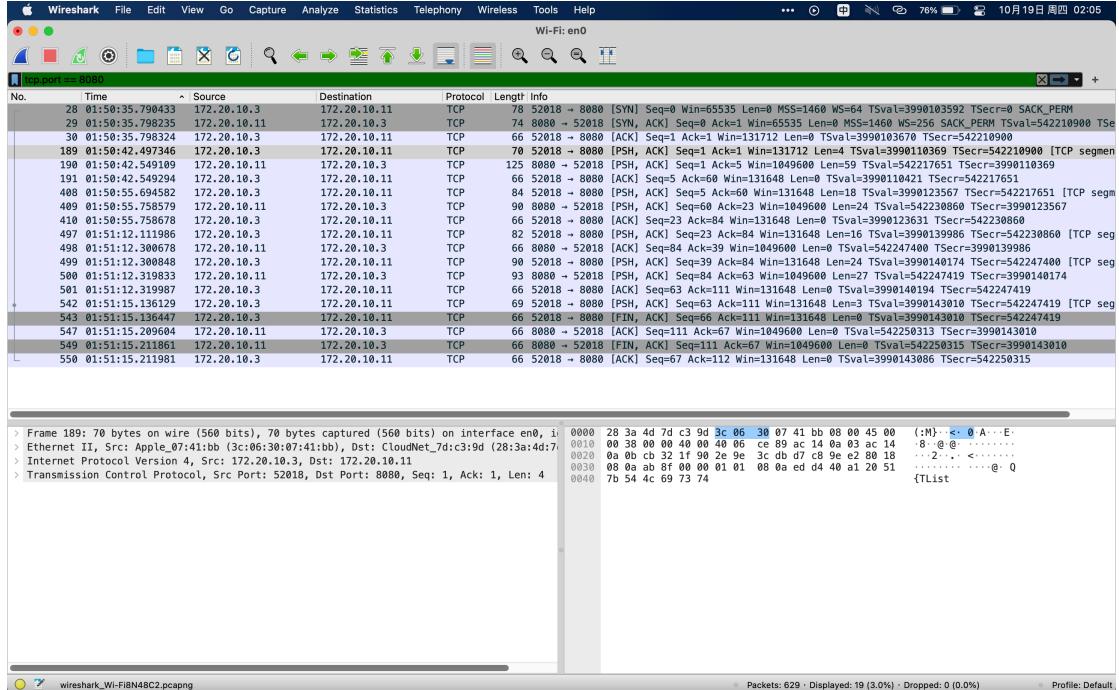


服务器：



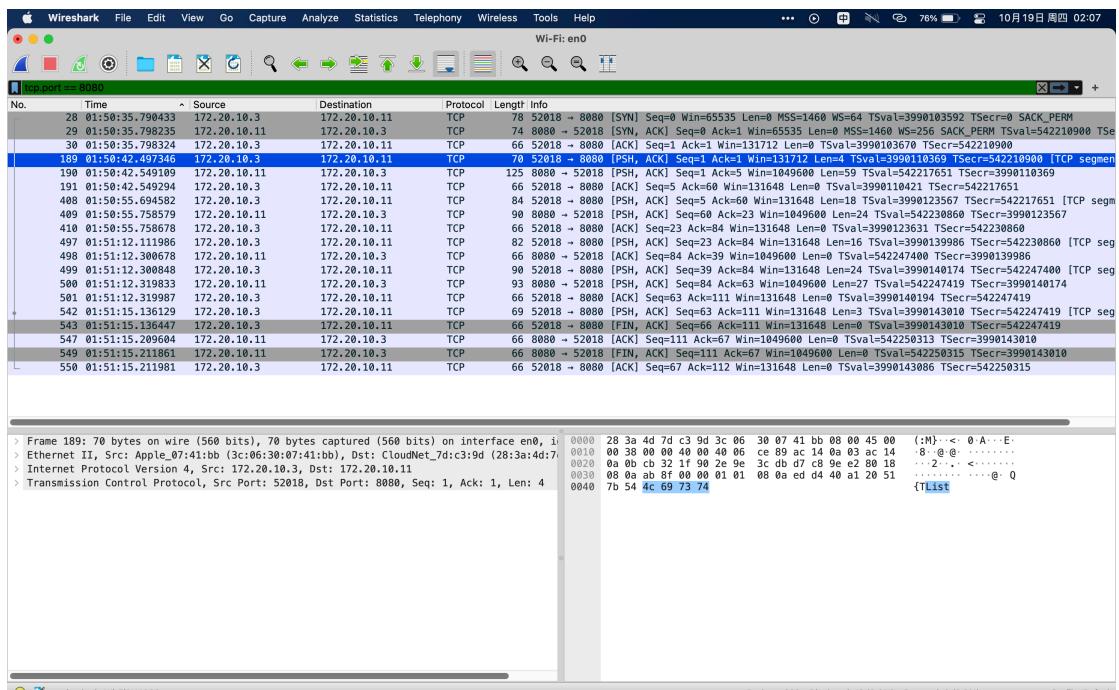
三、显示传输层相关的连接建立及关闭的过程；观察期间数据传输；分析并解释实验结果

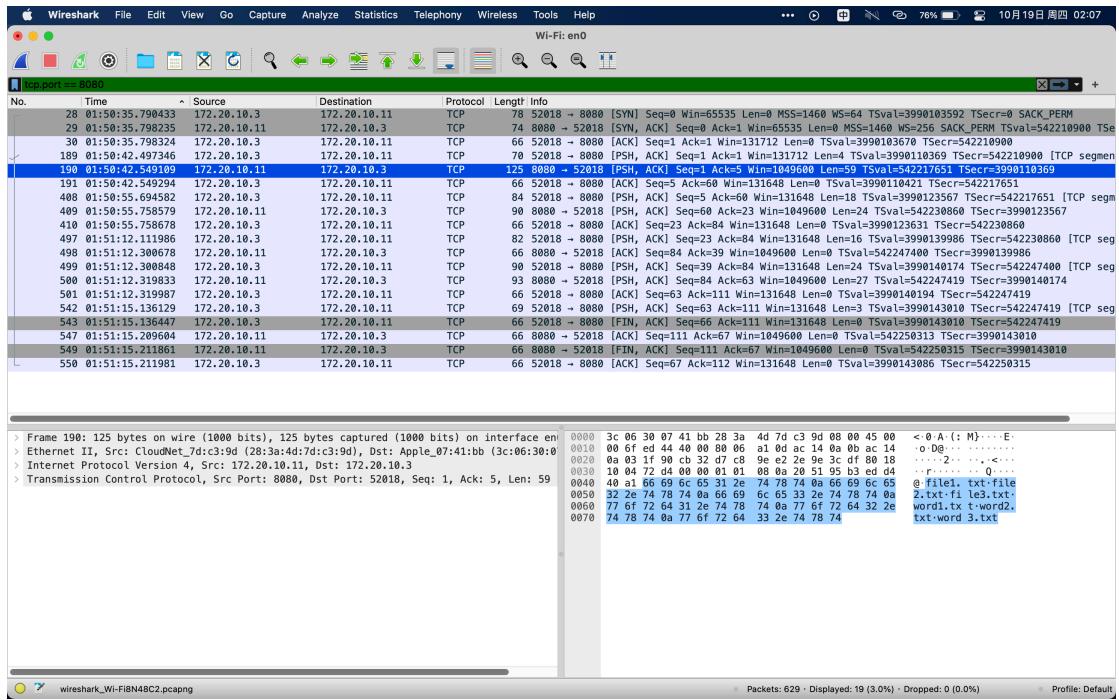
在客户端的抓包数据上进行分析



可以看到，客户端上，28 至 30 号报文为建立 TCP 连接的三次握手

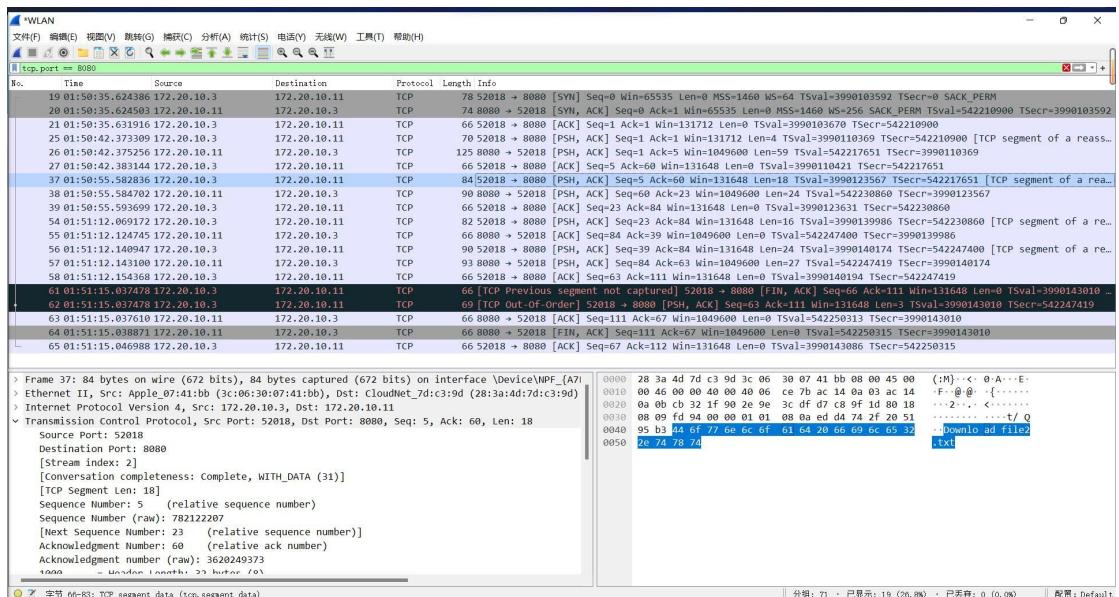
客户端首先请求查看服务器的当前文件列表：





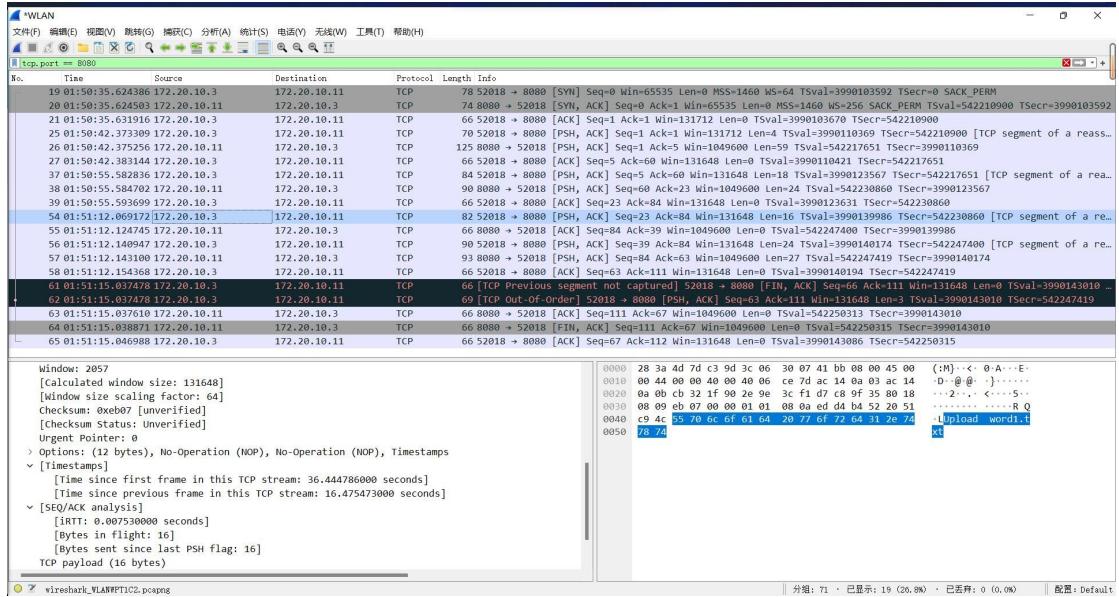
可以看到，在1:50时，客户端向服务器请求查看文件列表。189号报文数据段内容为List，为请求报文，与客户端程序的设计相对应。可以看到，190号报文数据段内容即为传送来的文件列表数据。191号报文为对190号报文的确认。

接下来从服务器的抓包数据进行分析



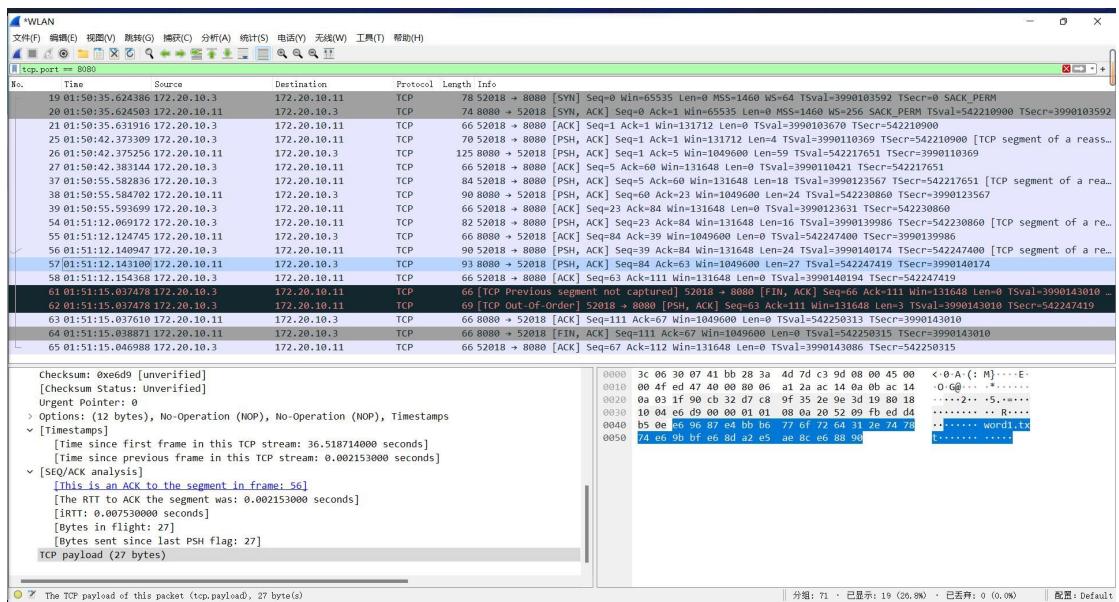
19至27号报文与前面的过程对应，从37号报文开始进入客户端下载请求，可以看到37号报文数据段内容为Download file2.txt，与客户端程序的设计相对应。之后38号报文将文件file2.txt的二进制数据发送给客户端。39号报文为对38号报文的确认。

之后服务器处理客户端上传文件的请求：



从 54 至 58 号报文对应这个过程。可以看到，54 号报文数据段内容为 Upload world1.txt，与客户端程序的设计相对应。55 号报文为对 54 号报文的确认。之后 56 号报文将文件 world1.txt 的二进制数据从服务器传到客户端。

57 号报文为服务器接收文件后的响应：



```

    # 给定的十六进制字符串
hex_string = 'e69687e4bbb6776f7264312e747874e69bbfe68da2e5ae8ce68890'

# 将十六进制字符串转换为字节对象
byte_string = bytes.fromhex(hex_string)

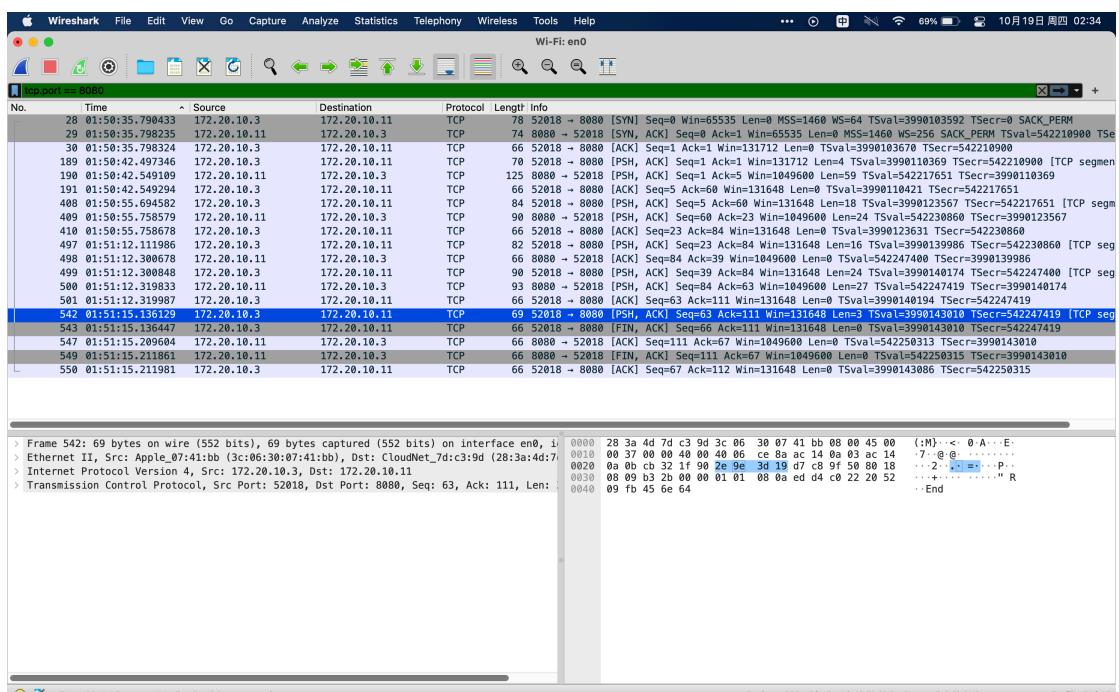
# 解码字节对象为UTF-8编码的字符串
decoded_string = byte_string.decode('utf-8')

# 打印解码后的字符串
print(decoded_string)

```

我们将 57 号报文的内容进行解码，可以看到其数据段的内容为：文件 word1.txt 替换完成。这里由于 word1.txt 已在服务器文件夹中存在，所以再次上传设计为将其替换。之后，58 号报文为对 57 号报文的确认。

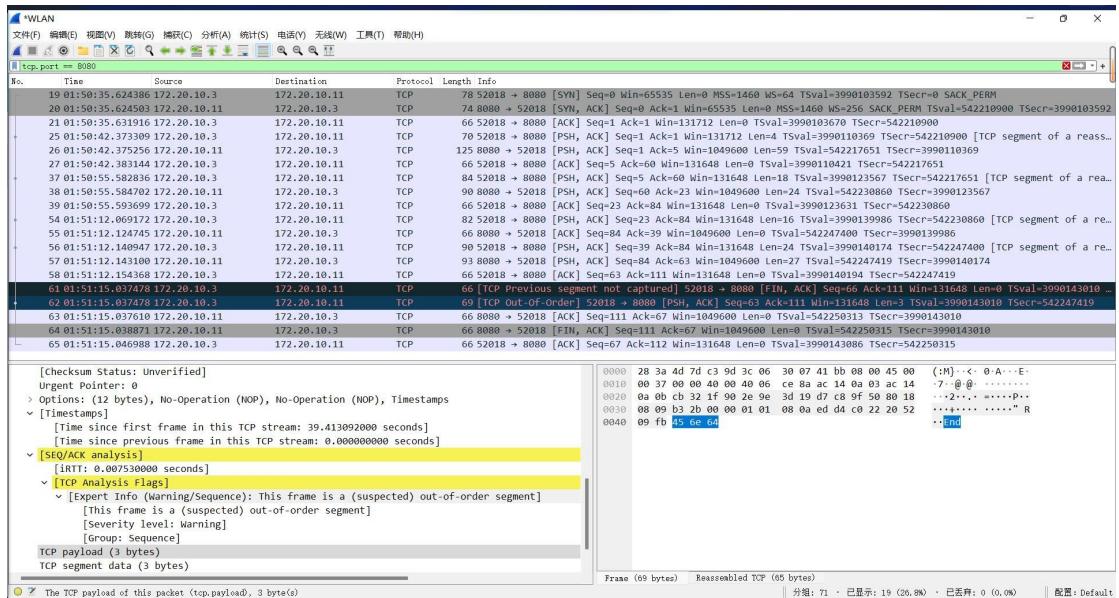
最后我们从客户端的抓包数据观察连接结束：



542 号报文是客户端请求断开连接，可以看到其数据段内容为 End，与客户端程序设计相对应。之后客户端套接字被关闭，客户端发送 543 号报文通知服务器。543 号报文标志为 FIN 和 ACK，表示这是一个结束连接的报文。

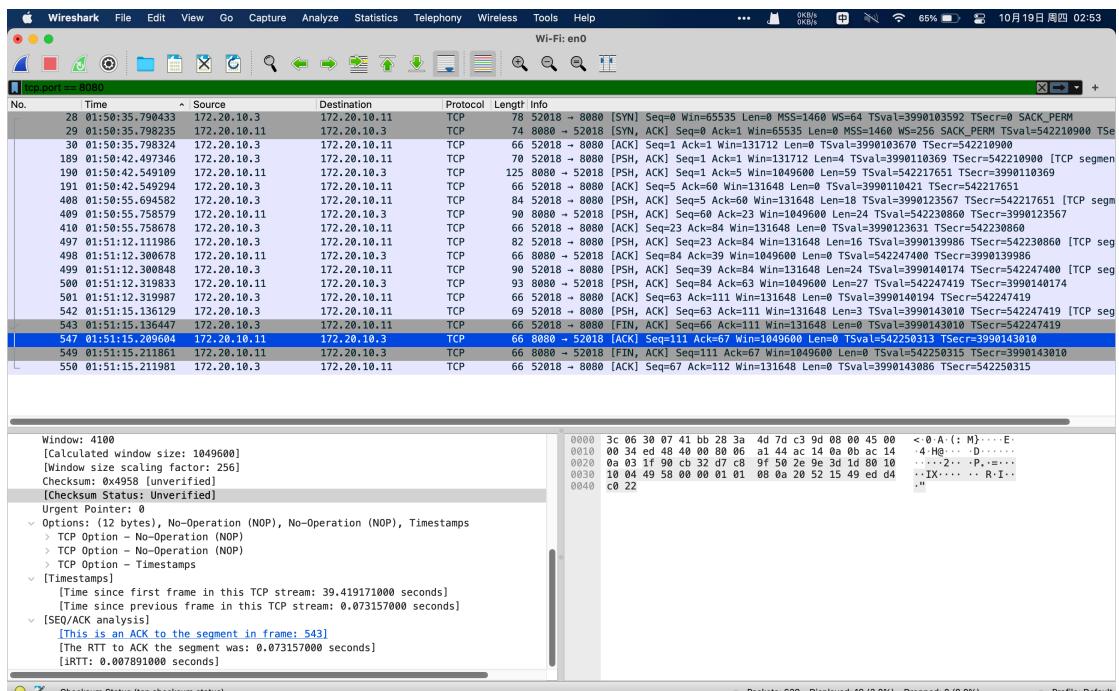
这里序列号为 63 的 542 号报文先被发送，之后序列号为 66 的 543 号报文再被发送。

我们查看服务器这边：



可以看到先发送的序列号为 63 的报文后一步才到达，后发送的序列号为 66 的报文反而先到到达，所以服务器抓包软件将这两个报文标识为黑色表示出现错误，并标识了 Out-Of-Order。

回到客户端视角：



547 号报文是对 543 号报文的确认。

549 号报文该报文也具有 FIN 和 ACK 标志，表示服务器希望关闭连接。

550 号报文是对 549 号报文的确认。