|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **实验名称** | 计算机网络实验二 | | **学 院** | 网络空间安全学院 | **指导教师** | 刘建毅 |
| **班 级** | **班内序号** | **学 号** | | **学生姓名** | **成绩** | |
| 2023211801  2023211807 |  | 2023211616  2023211617 | | 詹冲  蔡昱良 |  | |
| **实**  **验**  **内**  **容** | 本次实验主要包含下列内容：  1）使用Socket API通信实现文件传输客户端和服务器端2个程序，客户端发送文件传输请求，服务器端将文件数据发送给客户端，两个程序均在命令行方式下运行，要求至少能传输1个文本文件和1个图片文件；  2）客户端在命令行指定服务器的IP地址和文件名。为防止重名，客户端将收到的文件改名后保存在当前目录下。客户端应输出：新文件名、传输总字节数；或者差错报告；  3）服务器端应输出：客户端的IP地址和端口号；发送的文件数据总字节数；必要的差错报告（如文件不存在）。  4）可选功能：支持SSL安全连接。 | | | | | |
| **学生**  **实验**  **报告**  （附页） | 见下方《实验报告》 | | | | | |
| **实**  **验**  **成**  **绩**  **评**  **定** | **评语**:  **成绩**:  指导教师签名：    年 月 日 | | | | | |

**实验分工**

|  |  |
| --- | --- |
| 詹冲： | 客户端和服务端通信实现，代码编写；抓包验证；编写实验报告 |
| 蔡昱良： | 文件传输加密实现，代码编写；编写实验报告软件设计部分 |

**实验二 基于TCP的文件传输与Socket程序设计**

**一、实验目的**

深入理解传输层的TCP协议原理以及Socket套接字网络通信的流程。

**二、实验内容**

（1）使用SocketAPI通信实现文件传输客户端和服务器端2个程序，客户端发送文件传输请求，服务器端将文件数据发送给客户端，两个程序均在命令行方式下运行，要求至少能传输1个文本文件和1个图片文件；

（2）客户端在命令行指定服务器的IP地址和文件名。为防止重名，客户端将收到的文件改名后保存在当前目录下。客户端应输出：新文件名、传输总字节数；或者差错报告；

（3）服务器端应输出：客户端的IP地址和端口号；发送的文件数据总字节数；必要的差错报告（如文件不存在）。

（4）可选功能：支持SSL安全连接。

**三、实验环境**

（1）操作系统：Windows11

（2）编程语言：Python

（3）开发工具：Pycharm

**四、软件设计**

**（1）数据结构**

1.全局变量

|  |  |  |
| --- | --- | --- |
| 变量名 | 作用 | 类型 |
| HOST | 服务器监听地址，'0.0.0.0' 表示监听所有网络接口 | str |
| PORT | 服务器监听端口号，默认 34000 | int |

2.主函数变量

客户端：

|  |  |  |
| --- | --- | --- |
| 变量名 | 作用 | 类型 |
| SERVER\_HOST | 从命令行获取的服务器 IP 地址 | str |
| SERVER\_PORT | 服务器端口号，固定为 34000 | int |
| client\_socket | 客户端套接字对象 | socket.socket |
| context | SSL 上下文对象，配置加密参数 | ssl.SSLContext |
| REQUESTED\_FILENAME | 用户输入的文件名（含退出指令 !!!） | str |
| response\_header | 服务器返回的响应头（OK: 或 Error: 开头） | str |
| file\_size | 待接收文件的大小（从响应头解析） | int |
| local\_filename | 本地保存的文件名（原始名 + .bak） | str |
| bytes\_received | 已接收的文件字节数 | int |

服务器端：

|  |  |  |
| --- | --- | --- |
| 变量名 | 作用 | 类型 |
| server\_socket | 服务器套接字对象 | socket.socket |
| server\_address | 服务器地址元组（HOST, PORT） | tuple |
| context | SSL 上下文对象，加载服务器证书和私钥 | ssl.SSLContext |
| client\_socket | 与客户端通信的套接字对象 | socket.socket |
| client\_address | 客户端地址元组（IP, 端口） | tuple |
| filename | 客户端请求的文件名 | str |
| file\_size | 待发送文件的大小 | int |
| bytes\_sent | 已发送的文件字节数 | int |

**(2) 模块结构**

1.子程序功能描述

客户端：

start\_client()：

功能：解析参数、创建套接字、建立SSL连接、处理用户输入、收发文件请求及数据。

服务器端：

start\_server()：

功能：创建服务器套接字、绑定地址、监听连接、接受客户端请求并分配handle\_client处理。

handle\_client(client\_socket,client\_address)：

功能：处理单个客户端连接，接收文件名、校验文件、发送文件或错误信息。

参数：client\_socket：与客户端通信的套接字对象。

client\_address：客户端地址元组。

2.模块调用关系

客户端：

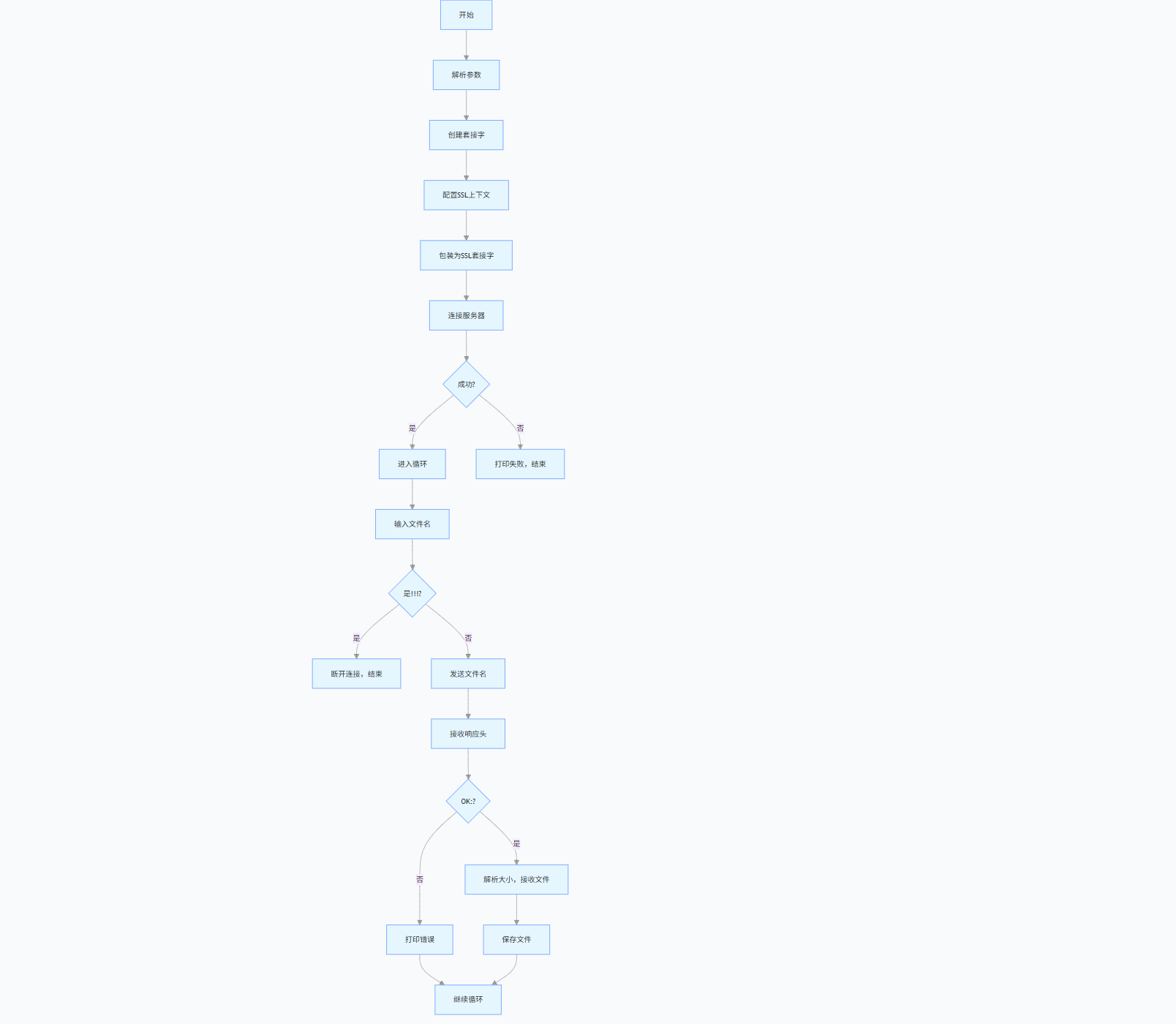
start\_client()（主流程）

服务器端：

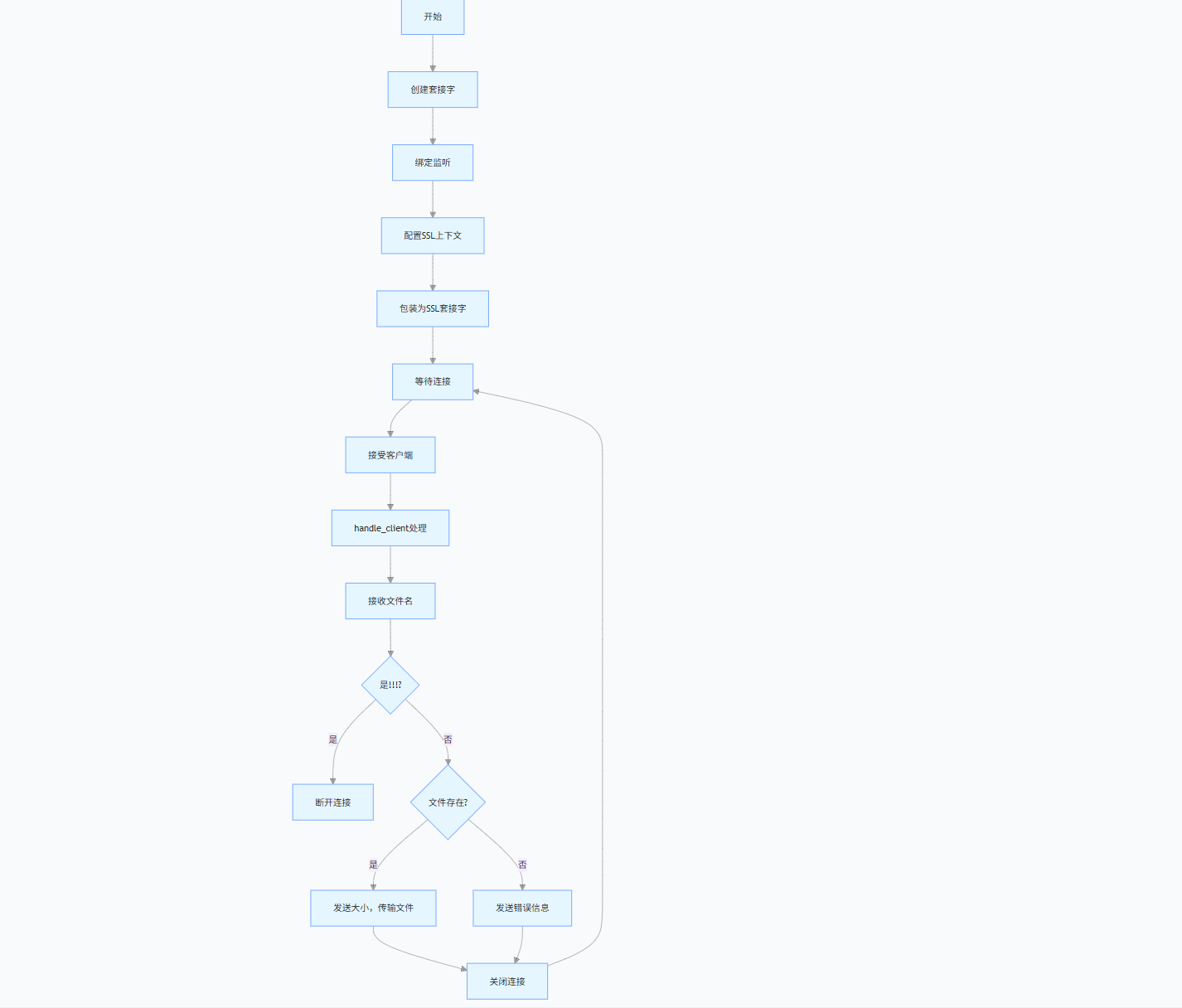
start\_server()→handle\_client()（每个客户端连接触发）

**（3）算法流程**

1.客户端流程



2. 服务器端流程



**（4）主要功能模块的实现要点**

1.SSL加密通信

服务器端：

使用ssl.SSLContext加载证书（cert.pem）和私钥（key.pem），通过wrap\_socket将普通套接字升级为SSL套接字。证书用于客户端验证服务器身份，私钥用于解密预主密钥、生成会话密钥。

客户端：

通过create\_default\_context创建SSL上下文。wrap\_socket完成SSL握手，后续数据自动加密传输。

2.文件传输协议

请求-响应格式：

客户端发送文件名，服务器返回响应头：

OK:{file\_size}：文件存在，file\_size为字节数。

Error:：文件不存在或处理失败。

分块传输：

服务器按4096字节分块读取文件，客户端按相同块大小接收，确保大文件传输的稳定性。

3.异常处理

客户端捕获Connection Refused Error、socket.gaierror等网络异常，服务器通过try-except处理文件操作和客户端断开异常，保证程序健壮性。

4.退出机制

客户端输入!!!触发主动断开，服务器检测到空数据（not data）或退出命令时关闭连接，释放资源。

**（5）总结**

1.数据结构：以基础变量和网络字节流为主，通过字符串协议定义数据边界。

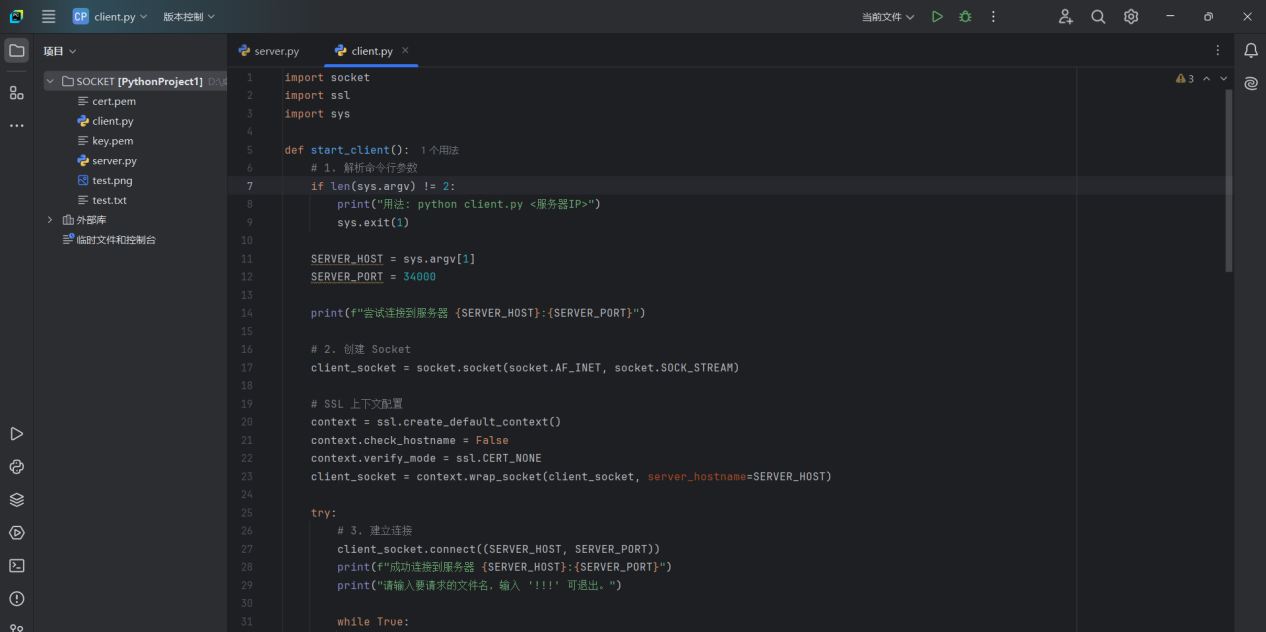
2.模块设计：客户端和服务器解耦，服务器通过多连接处理支持并发请求。

3.扩展方向：添加多线程处理客户端连接、实现文件断点续传、启用双向SSL认证（mTLS）。

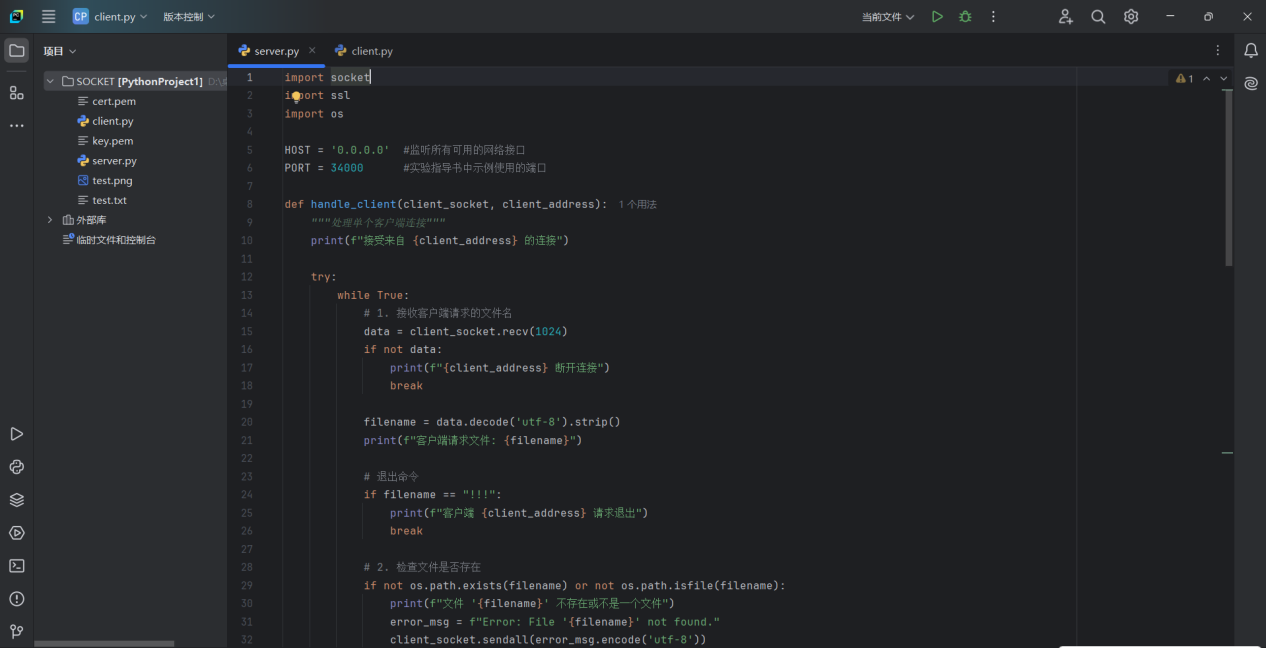
**五、实验步骤**

**（1）编写代码**

1.客户端代码：

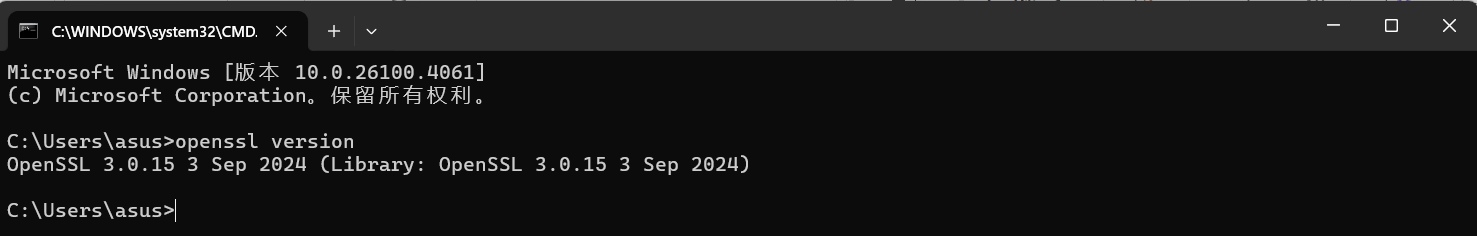


2.服务端代码：



**（2）生成SSL安全连接的证书和密钥**

1.下载openssl，首先至官网<https://slproweb.com/products/Win32OpenSSL.html>安装openssl，安装时选择“安装为系统环境变量”。安装完成后，打开终端，输入openssl version看是否正确安装。



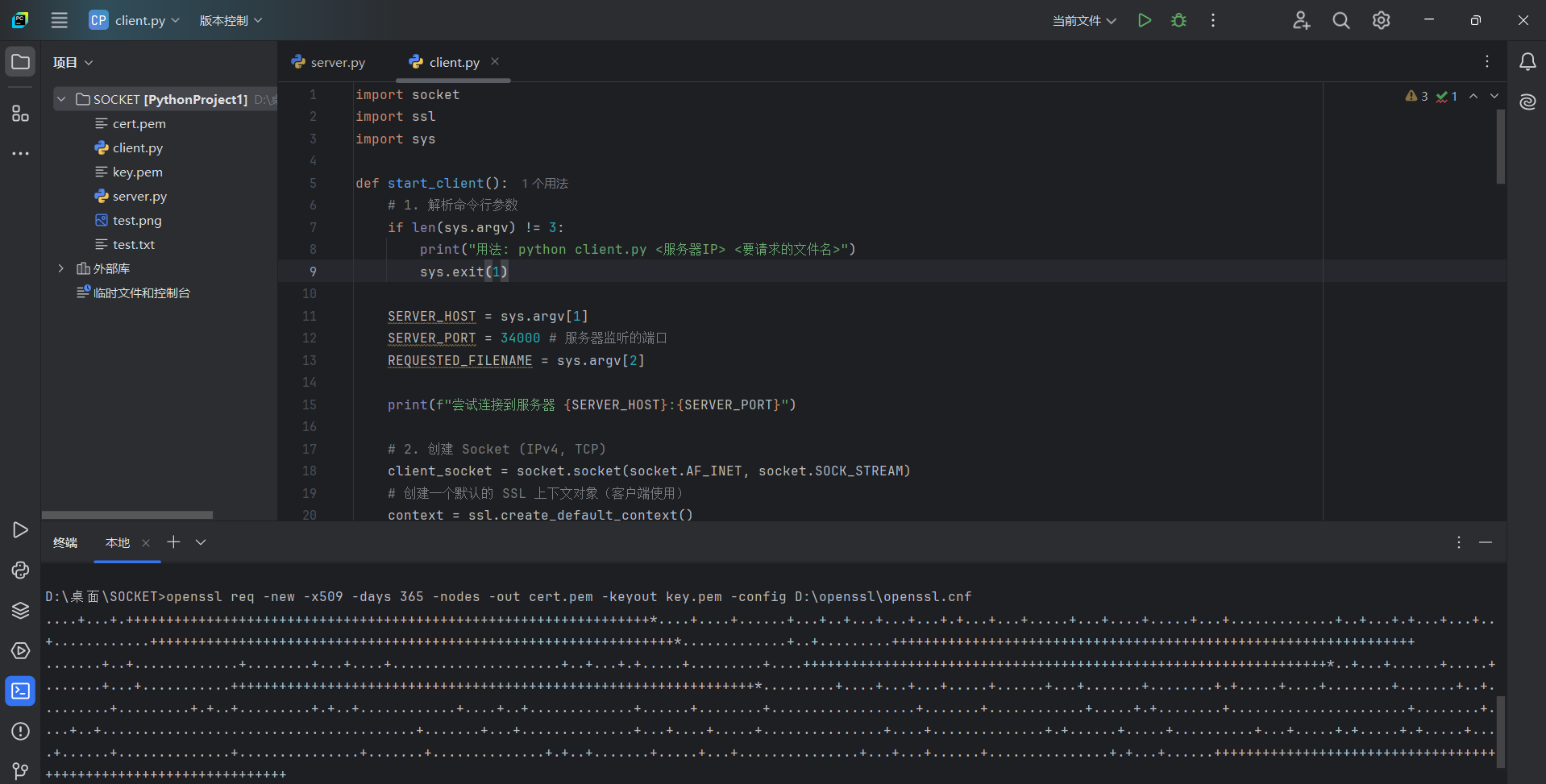
2.创建配置文件openssl.cnf，打开文本编辑器（推荐用 VS Code），将下面内容复制进去，选择一个容易找到的位置保存，比如：D:\openssl\openssl.cnf。

1. [ req ]
2. default\_bits       = 2048
3. distinguished\_name = req\_distinguished\_name
4. prompt             = no
5. x509\_extensions    = v3\_req
6. [ req\_distinguished\_name ]
7. C = CN
8. ST = Beijing
9. L = Beijing
10. O = MyOrganization
11. OU = MyUnit
12. CN = localhost
13. [ v3\_req ]
14. subjectAltName = @alt\_names
15. [ alt\_names ]
16. DNS.1 = localhost



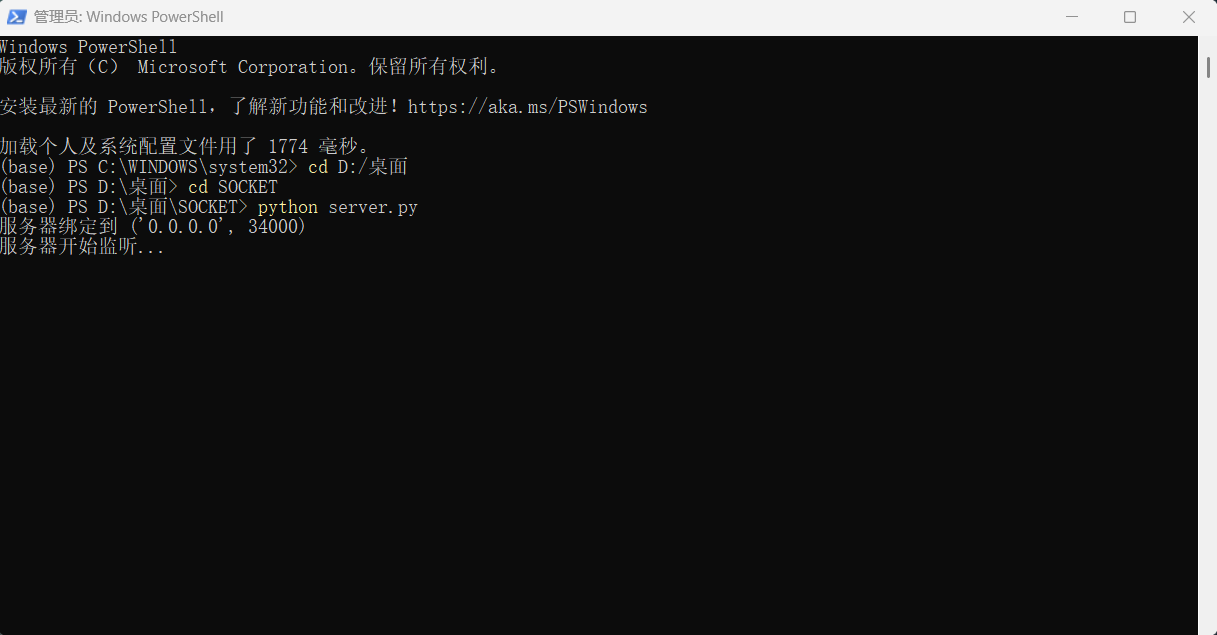
**3.在终端运行以下命令生成证书和私钥**

openssl req -new -x509 -days 365 -nodes -out cert.pem -keyout key.pem -config D:\openssl\openssl.cnf

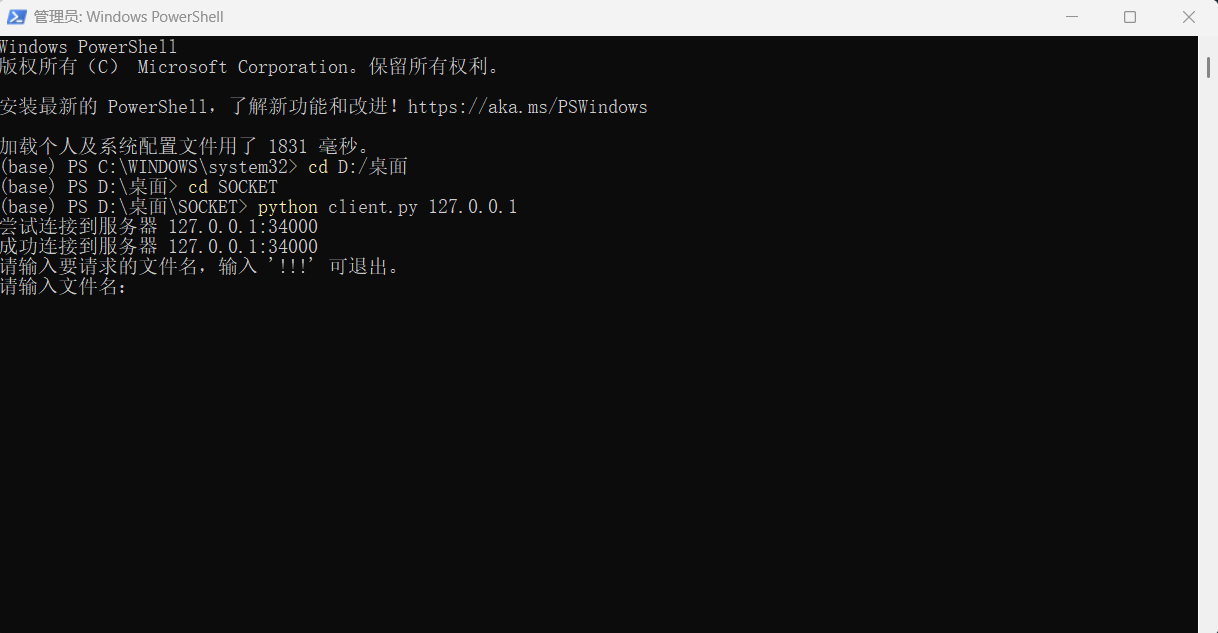


**（3）启动程序**

1.先以管理员身份打开一个终端，cd至项目所在目录，通过python server.py命令，启动服务端程序。



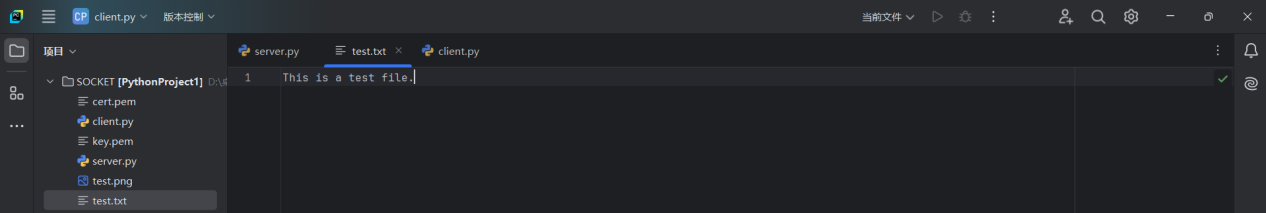
2.再以管理员身份打开另一个终端。cd至项目所在目录，通过规定的“python client.py <服务器IP>”格式运行客户端程序。



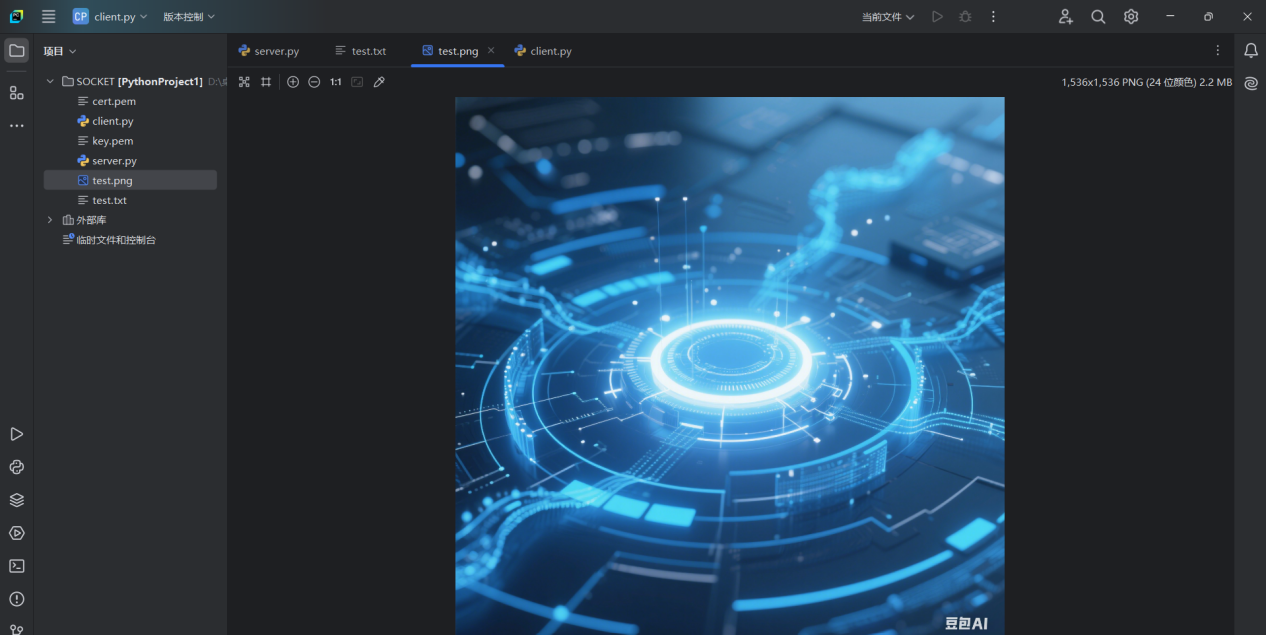
**（4）进行文件传输**

1.预先准备好两个待发送的文件，此处为test.txt和test.png。

test.txt：



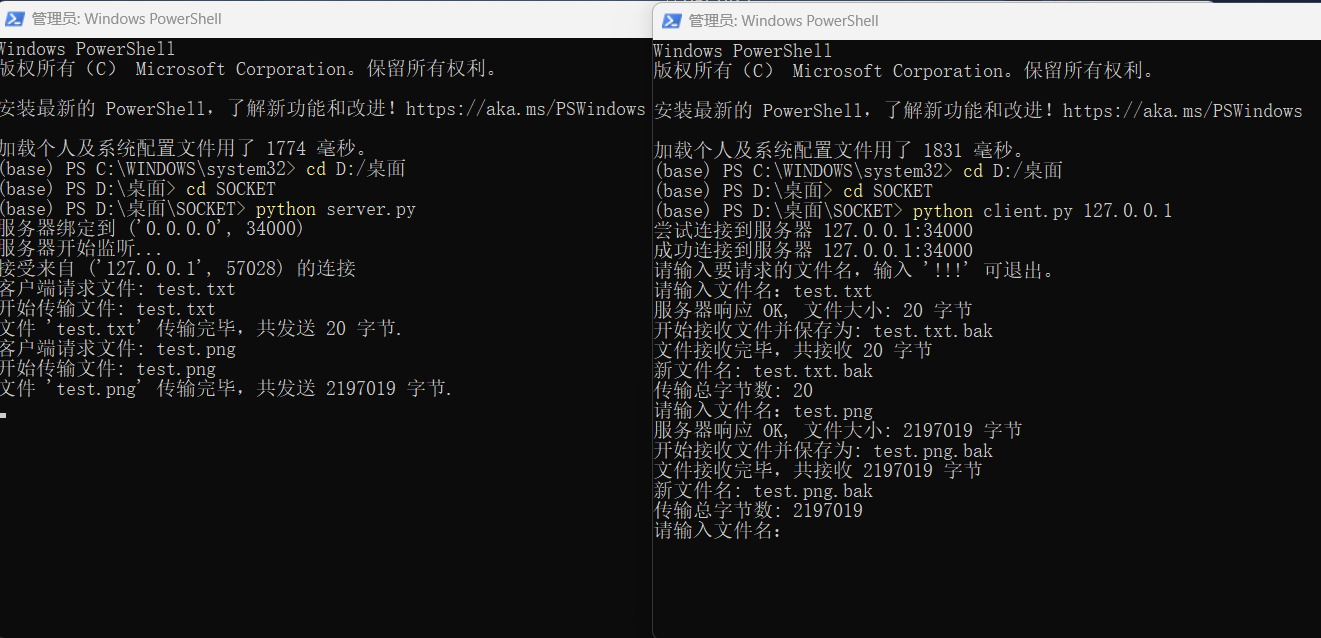
test.png：



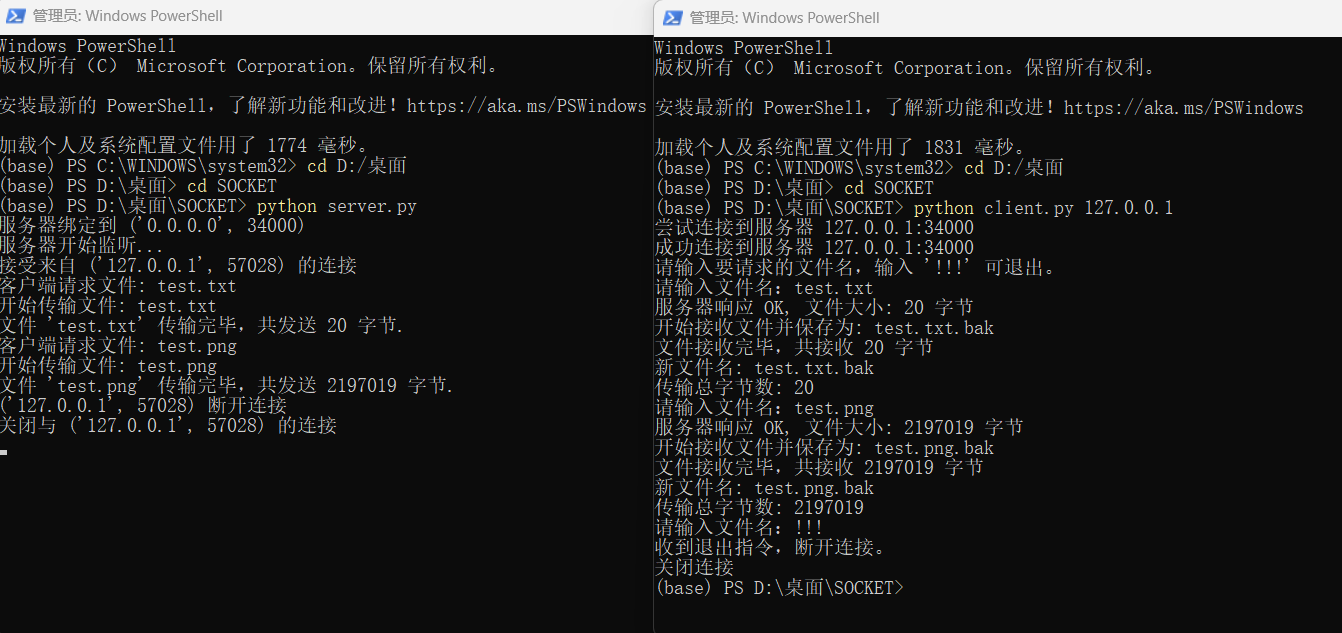
2.传输test.txt文件



3.传输test.png文件

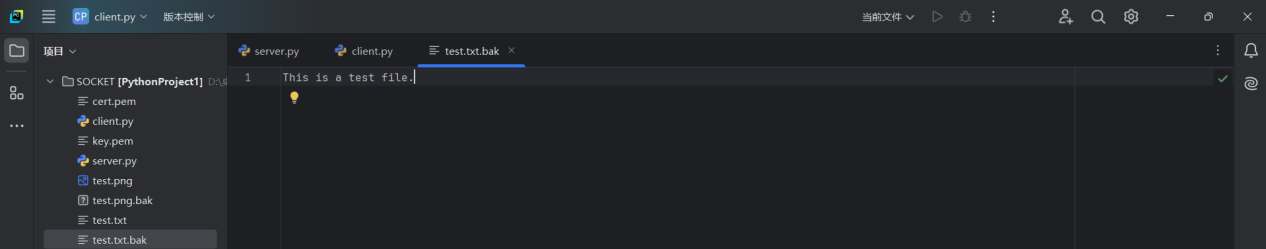


4.断开连接

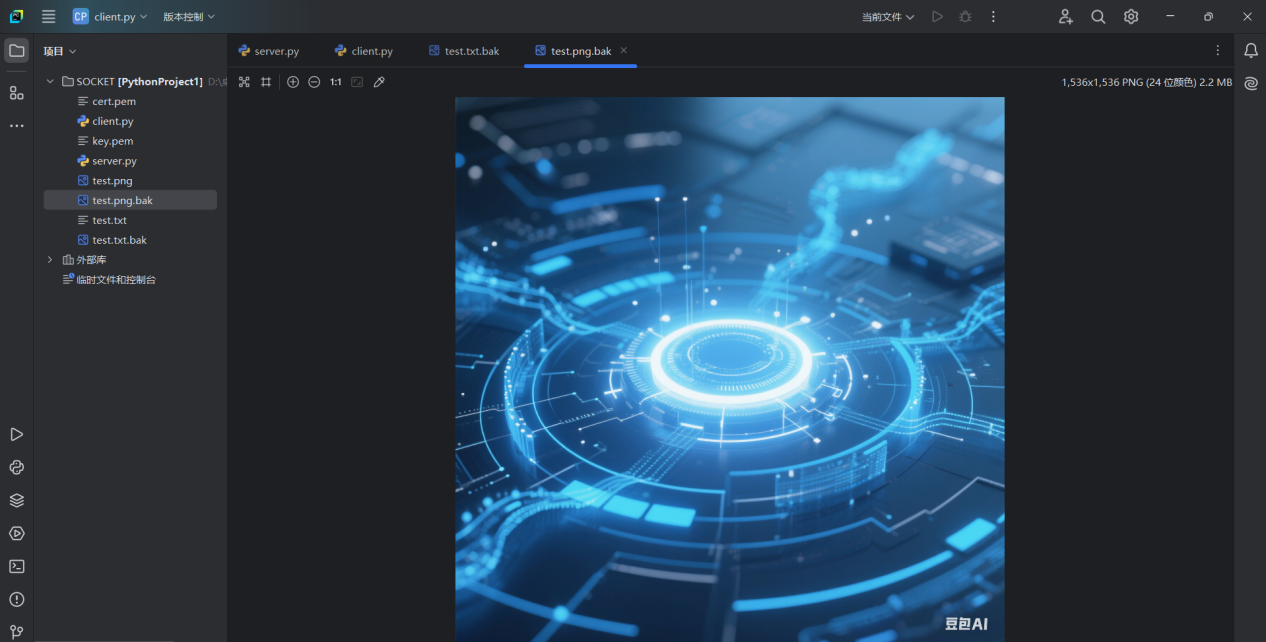


5.检查文件传输是否正确

检查test.txt.bak，发现与源文件相同。

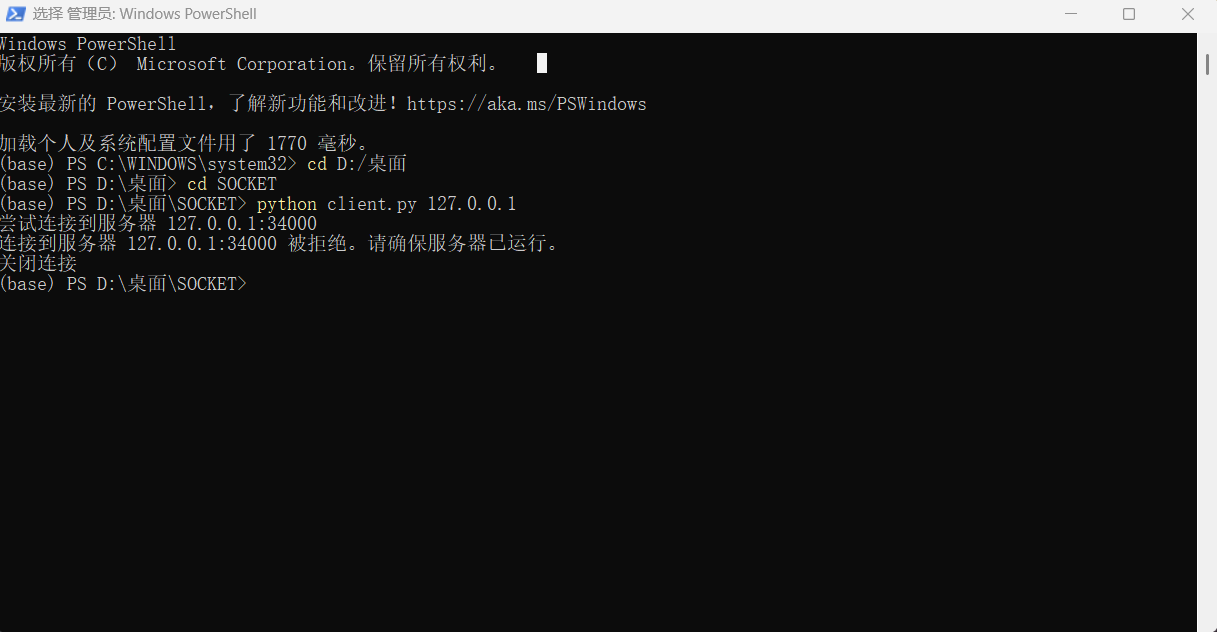


检查test.png.bak，发现与源文件相同。

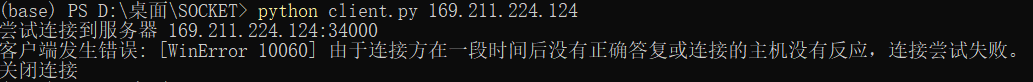


**（5）差错报告功能**

1.当服务器存在，但是没有开始运行时，发送如下差错报告。



2.当服务器不存在时，发送如下差错报告。



3.当客户端请求的文件不存在时，客户端和服务器均差错报告。

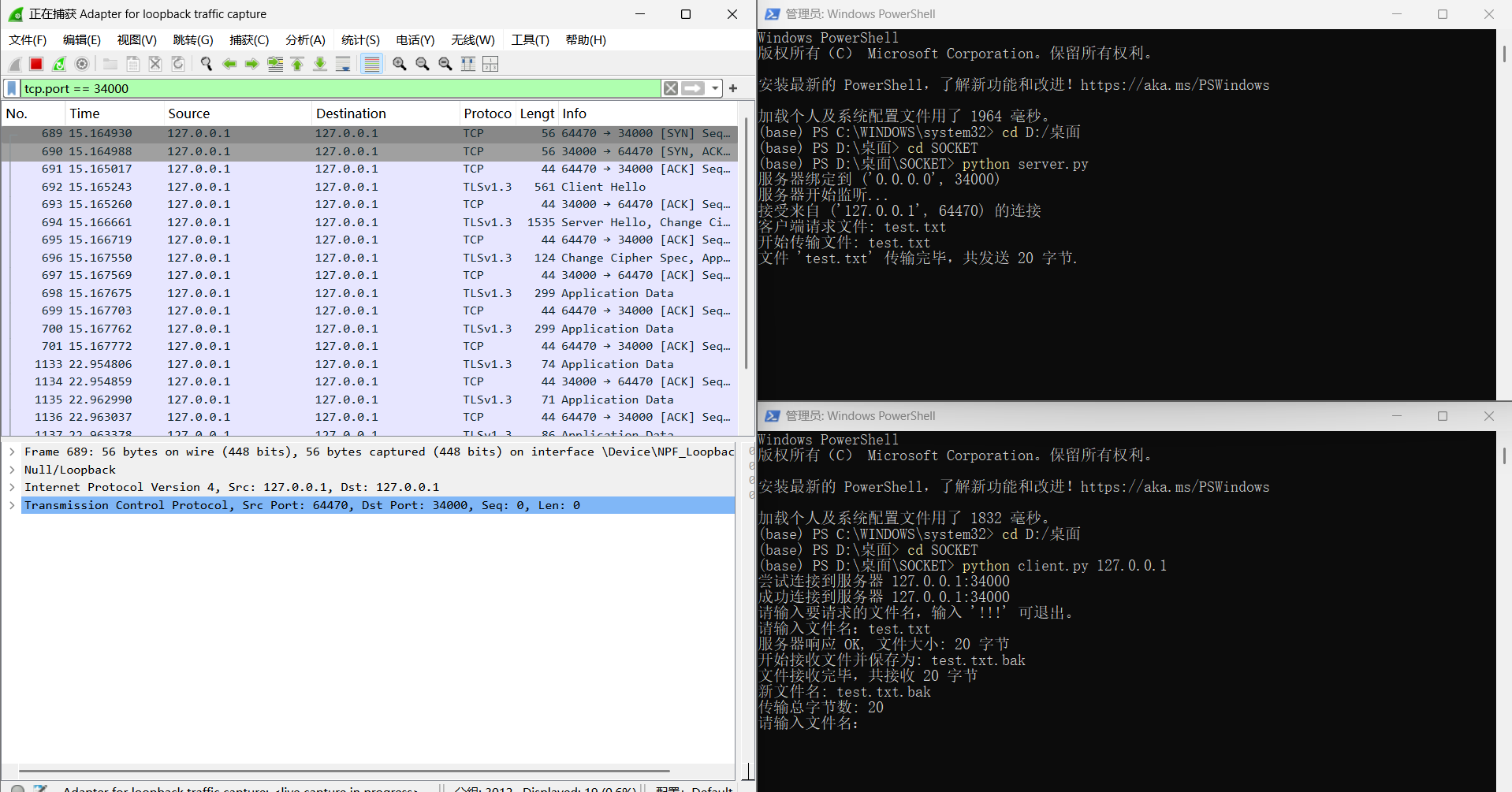
客户端：



服务器：



**（6）通过wireshark抓包，查看SSL安全连接建立成功，数据包均已被加密。**



**六、实验总结和心得体会**

(1)调试过程中都遇到的问题和解决的过程。

1.一开始进行抓包时，由于选择错误网络，导致一直无法抓取，后面查找资料得知，应选择adapter for loopback traffic capture网络。

2.对于SSL安全连接知识较少，后续通过查找相关资料，才得以实现。

(2)总结本次实验，在SOCKET机制方面、协议软件设计方面、理论学习方面、软件工程方面等哪些方面上有所收获和提高？

1.SOCKET机制方面：

深入理解了TCP的连接模型；掌握了Python中使用socket和ssl库实现安全加密通信；

熟悉了客户端与服务器通信过程的完整生命周期，包括连接、请求、响应、关闭。

2.协议软件设计方面：

了解了自定义通信协议的基本思路（如：先发送响应头OK:长度，再发送数据）；

实践了如何设计一个简单但可扩展的文件传输协议，并兼容错误信息处理。

3.理论学习方面：

将教材中的TCP、SSL协议知识和实际代码实现结合，加深了对通信协议的理解；通过错误分析过程，掌握了理论与实际之间的差异点，如连接拒绝与连接超时的本质区别。

4.软件工程方面：

锻炼了模块化设计能力，客户端和服务器清晰分工；学会了调试过程中的日志记录、异常处理和错误信息反馈机制；强化了“边写边测”的迭代式开发思想。

(3)设计的不足之处

1.无多线程/并发处理

服务器是单线程的，handle\_client()是串行执行，意味着同一时刻只能处理一个客户端，其他客户端会阻塞。

2.未对用户输入做超时/非法输入处理

没有设置socket的超时时间或异常输入保护机制，可能因恶意客户端或网络故障而被阻塞。

**附录：程序源代码**

**服务器代码：**

*#*

*#server.py*

*#*

import socket

import ssl

import os

HOST = '0.0.0.0'  *#监听所有可用的网络接口*

PORT = 34000      *#实验指导书中示例使用的端口*

def handle\_client(client\_socket, client\_address):

    """处理单个客户端连接"""

    print(f"接受来自 {client\_address} 的连接")

    try:

        while True:

*# 1. 接收客户端请求的文件名*

            data = client\_socket.recv(1024)

            if not data:

                print(f"{client\_address} 断开连接")

                break

            filename = data.decode('utf-8').strip()

            print(f"客户端请求文件: {filename}")

*# 退出命令*

            if filename == "!!!":

                print(f"客户端 {client\_address} 请求退出")

                break

*# 2. 检查文件是否存在*

            if not os.path.exists(filename) or not os.path.isfile(filename):

                print(f"文件 '{filename}' 不存在或不是一个文件")

                error\_msg = f"Error: File '{filename}' not found."

                client\_socket.sendall(error\_msg.encode('utf-8'))

                continue  *# 等待下一个请求*

*# 3. 发送文件*

            print(f"开始传输文件: {filename}")

            file\_size = os.path.getsize(filename)

            client\_socket.sendall(f"OK:{file\_size}".encode('utf-8'))

            with open(filename, 'rb') as f:

                bytes\_sent = 0

                while True:

                    chunk = f.read(4096)

                    if not chunk:

                        break

                    client\_socket.sendall(chunk)

                    bytes\_sent += len(chunk)

            print(f"文件 '{filename}' 传输完毕，共发送 {bytes\_sent} 字节.")

    except Exception as e:

        print(f"处理客户端 {client\_address} 时发生错误: {e}")

        try:

            client\_socket.sendall(f"Error: Server processing failed - {e}".encode('utf-8'))

        except:

            pass

    finally:

        client\_socket.close()

        print(f"关闭与 {client\_address} 的连接")

def start\_server():

*# 1. 创建 Socket (IPv4, TCP)*

    server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

*# 2. 绑定地址和端口*

    server\_address = (HOST, PORT)

    server\_socket.bind(server\_address)

    print(f"服务器绑定到 {server\_address}")

*# 3. 监听连接请求*

    server\_socket.listen(5)

*# 创建一个用于 TLS 的 SSL 上下文对象（服务器端使用）*

    context = ssl.SSLContext(ssl.PROTOCOL\_TLS\_SERVER)

*# 加载服务器的证书和私钥，用于加密通信*

    context.load\_cert\_chain(certfile='cert.pem', keyfile='key.pem')

*# 用 SSL 封装原始 socket，使其支持安全通信*

    server\_socket = context.wrap\_socket(server\_socket, server\_side=True)

    print("服务器开始监听...")

*# 服务器主循环，持续接受连接*

    while True:

*# 4. 接受客户端连接 (阻塞式)*

        client\_socket, client\_address = server\_socket.accept()

*# 可以在这里创建新线程或进程来处理 handle\_client*

        handle\_client(client\_socket, client\_address)

*# 在程序入口调用启动服务器函数*

if \_\_name\_\_ == '\_\_main\_\_':

    start\_server()

**客户端代码：**

*#*

*#client.py*

*#*

import socket

import ssl

import sys

def start\_client():

*# 1. 解析命令行参数*

    if len(sys.argv) != 2:

        print("用法: python client.py <服务器IP>")

        sys.exit(1)

    SERVER\_HOST = sys.argv[1]

    SERVER\_PORT = 34000

    print(f"尝试连接到服务器 {SERVER\_HOST}:{SERVER\_PORT}")

*# 2. 创建 Socket*

    client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

*# SSL 上下文配置*

    context = ssl.create\_default\_context()

    context.check\_hostname = False

    context.verify\_mode = ssl.CERT\_REQUIRED

    context.load\_verify\_locations('cert.pem')  *# 指向你服务器的自签名证书*

    client\_socket = context.wrap\_socket(client\_socket, server\_hostname=SERVER\_HOST)

    try:

*# 3. 建立连接*

        client\_socket.connect((SERVER\_HOST, SERVER\_PORT))

        print(f"成功连接到服务器 {SERVER\_HOST}:{SERVER\_PORT}")

        print("请输入要请求的文件名，输入 '!!!' 可退出。")

        while True:

*# 4. 读取用户输入的文件名*

            REQUESTED\_FILENAME = input("请输入文件名：").strip()

            if REQUESTED\_FILENAME == "!!!":

                print("收到退出指令，断开连接。")

                break

            if not REQUESTED\_FILENAME:

                print("文件名不能为空，请重新输入。")

                continue

*# 5. 发送文件名请求*

            client\_socket.sendall(REQUESTED\_FILENAME.encode('utf-8'))

*# 6. 接收响应头*

            response\_header = client\_socket.recv(1024).decode('utf-8')

            if response\_header.startswith("OK:"):

                try:

                    file\_size = int(response\_header.split(":")[1])

                    print(f"服务器响应 OK, 文件大小: {file\_size} 字节")

                    local\_filename = REQUESTED\_FILENAME + ".bak"

                    print(f"开始接收文件并保存为: {local\_filename}")

                    bytes\_received = 0

                    with open(local\_filename, 'wb') as f:

                        while bytes\_received < file\_size:

                            chunk = client\_socket.recv(min(4096, file\_size - bytes\_received))

                            if not chunk:

                                break

                            f.write(chunk)

                            bytes\_received += len(chunk)

                    print(f"文件接收完毕，共接收 {bytes\_received} 字节")

                    print(f"新文件名: {local\_filename}")

                    print(f"传输总字节数: {bytes\_received}")

                except Exception as e:

                    print(f"处理文件接收时发生错误: {e}")

                    print("输出：差错报告")

            elif response\_header.startswith("Error:"):

                error\_msg = response\_header[len("Error:"):]

                print(f"服务器返回错误: {error\_msg.strip()}")

            else:

                print(f"接收到未知响应头部: {response\_header.strip()}")

    except ConnectionRefusedError:

        print(f"连接到服务器 {SERVER\_HOST}:{SERVER\_PORT} 被拒绝。请确保服务器已运行。")

    except socket.gaierror:

        print(f"无法解析服务器地址: {SERVER\_HOST}")

    except Exception as e:

        print(f"客户端发生错误: {e}")

    finally:

        print("关闭连接")

        client\_socket.close()

if \_\_name\_\_ == '\_\_main\_\_':

    start\_client()