

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		2023211616	詹冲		
2023211807		2023211617	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端 2 个程序，客户端发送文件传输请求，服务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。				
	见下方《实验报告》				
学生实验报告 (附页)					
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：

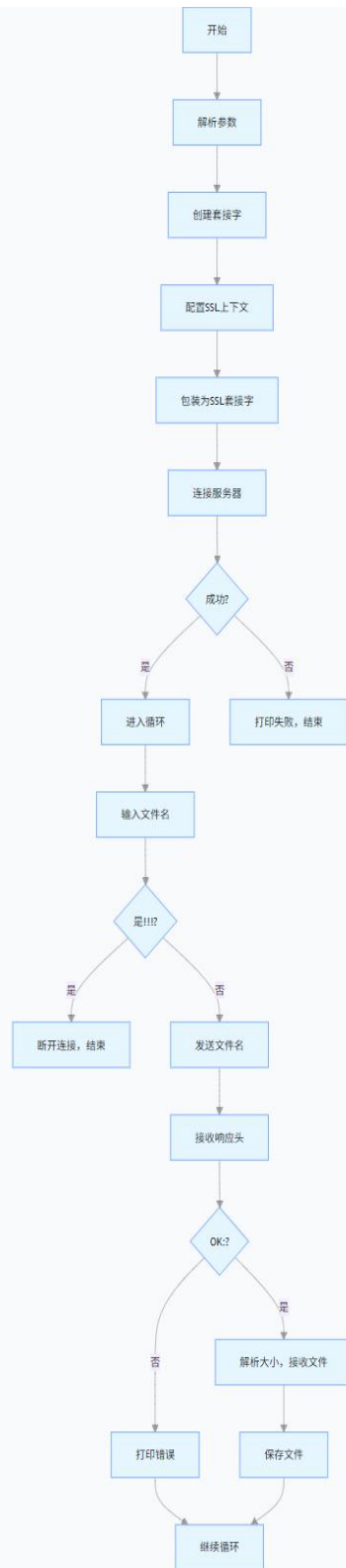
start\_client()（主流程）

服务器端：

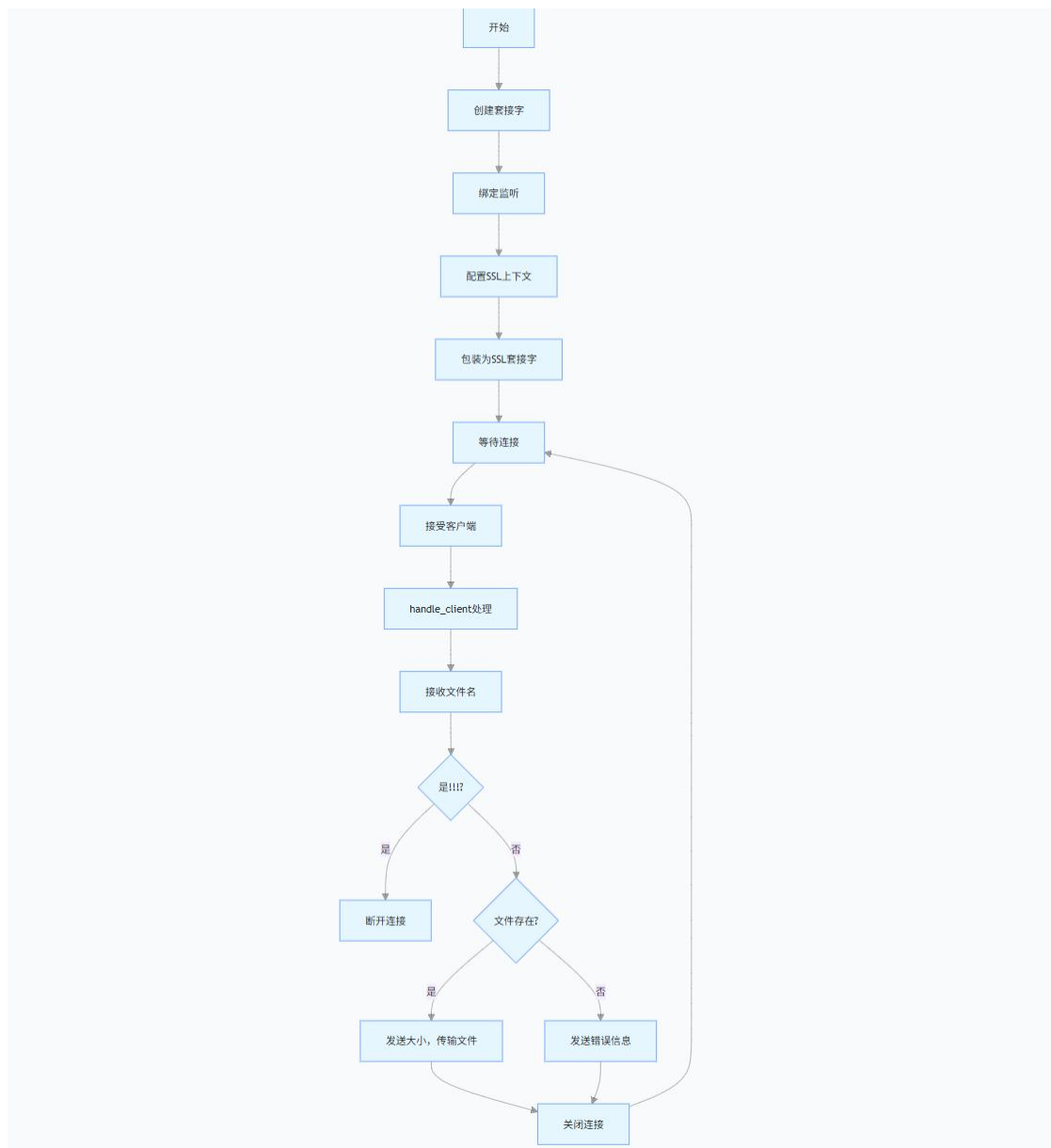
start\_server()→handle\_client()（每个客户端连接触发）

## （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。

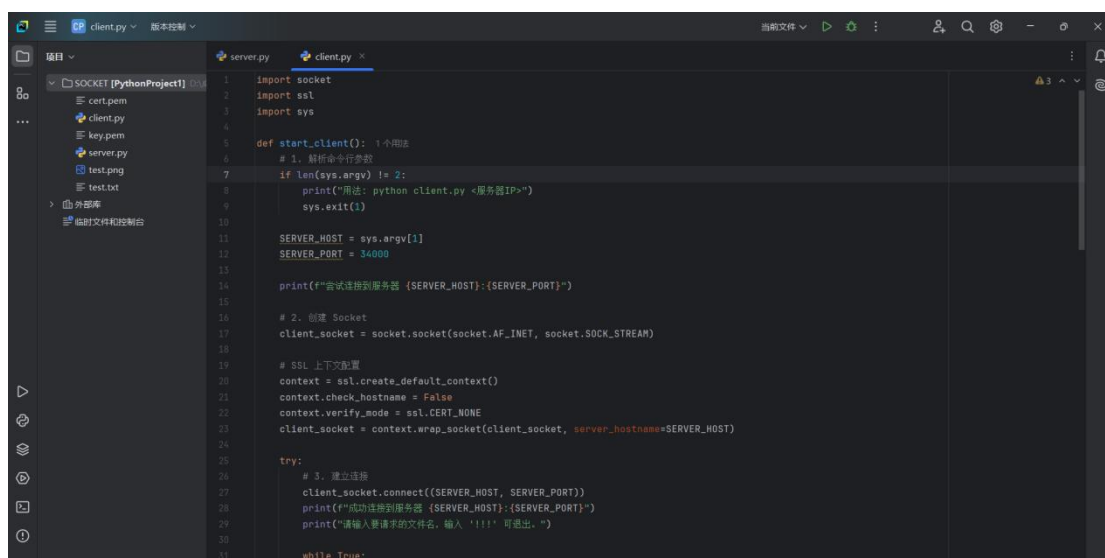
#### (5) 总结

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：客户端和服务端解耦，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

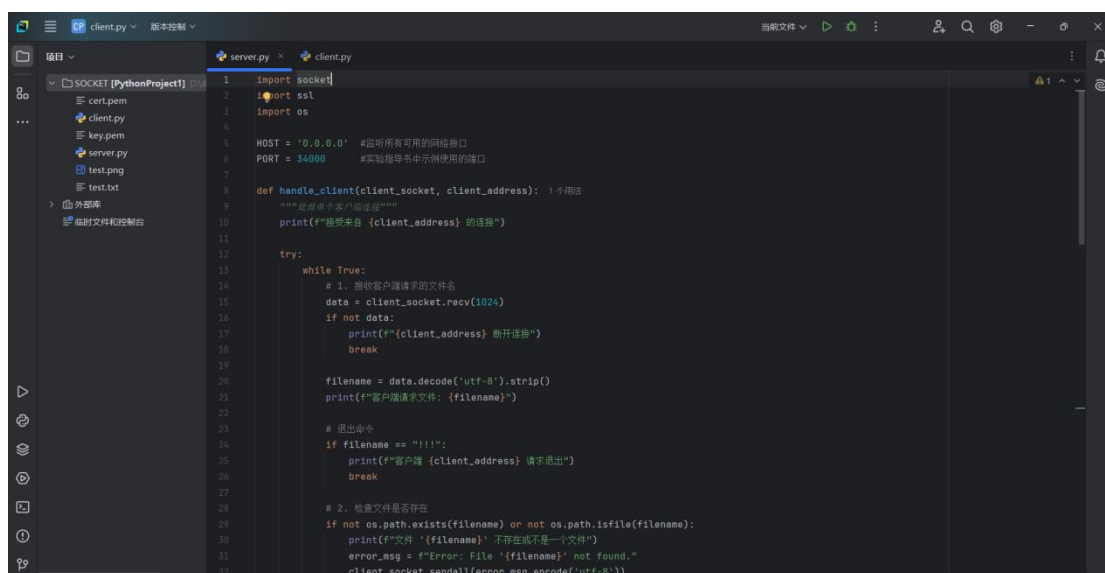
### (1) 编写代码

#### 1. 客户端代码：



```
1 import socket
2 import ssl
3 import sys
4
5 def start_client(): 1个用法
6     # 1. 解析命令行参数
7     if len(sys.argv) != 2:
8         print("用法: python client.py <服务器IP>")
9         sys.exit(1)
10
11     SERVER_HOST = sys.argv[1]
12     SERVER_PORT = 34000
13
14     print(f"尝试连接到服务器 {SERVER_HOST}:{SERVER_PORT}")
15
16     # 2. 创建 Socket
17     client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
18
19     # SSL 上下文配置
20     context = ssl.create_default_context()
21     context.check_hostname = False
22     context.verify_mode = ssl.CERT_NONE
23     client_socket = context.wrap_socket(client_socket, server_hostname=SERVER_HOST)
24
25     try:
26         # 3. 建立连接
27         client_socket.connect((SERVER_HOST, SERVER_PORT))
28         print(f"成功连接到服务器 {SERVER_HOST}:{SERVER_PORT}")
29         print("请输入要请求的文件名, 输入 '!!!' 可退出。")
30
31         while True:
```

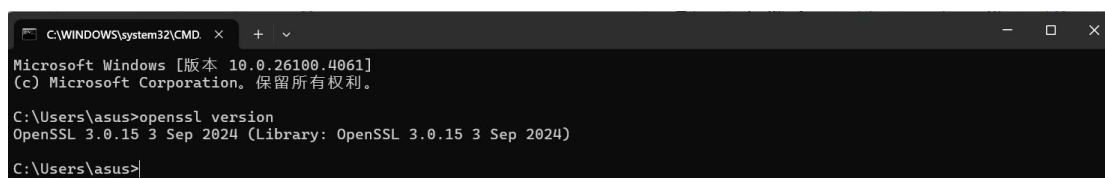
## 2. 服务端代码:



```
1 import socket
2 import ssl
3 import os
4
5 HOST = '0.0.0.0' # 监听所有可用的网络接口
6 PORT = 34000 # 实验指导书中示例使用的端口
7
8 def handle_client(client_socket, client_address): 1个用法
9     """处理单个客户端连接"""
10    print(f"接受来自 {client_address} 的连接")
11
12    try:
13        while True:
14            # 1. 接收客户端请求的文件名
15            data = client_socket.recv(1024)
16            if not data:
17                print(f"{client_address} 断开连接")
18                break
19
20            filename = data.decode('utf-8').strip()
21            print(f"客户端请求文件: {filename}")
22
23            # 退出命令
24            if filename == "!!!":
25                print(f"客户端 {client_address} 请求退出")
26                break
27
28            # 2. 检查文件是否存在
29            if not os.path.exists(filename) or not os.path.isfile(filename):
30                print(f"文件 '{filename}' 不存在或不是一个文件")
31                error_msg = f"Error: File '{filename}' not found."
32                client_socket.sendall(error_msg.encode('utf-8'))
```

### (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。



```
C:\WINDOWS\system32\CMD. x + v
Microsoft Windows [版本 10.0.26100.4061]
(c) Microsoft Corporation。保留所有权利。

C:\Users\asus>openssl version
OpenSSL 3.0.15 3 Sep 2024 (Library: OpenSSL 3.0.15 3 Sep 2024)

C:\Users\asus>
```

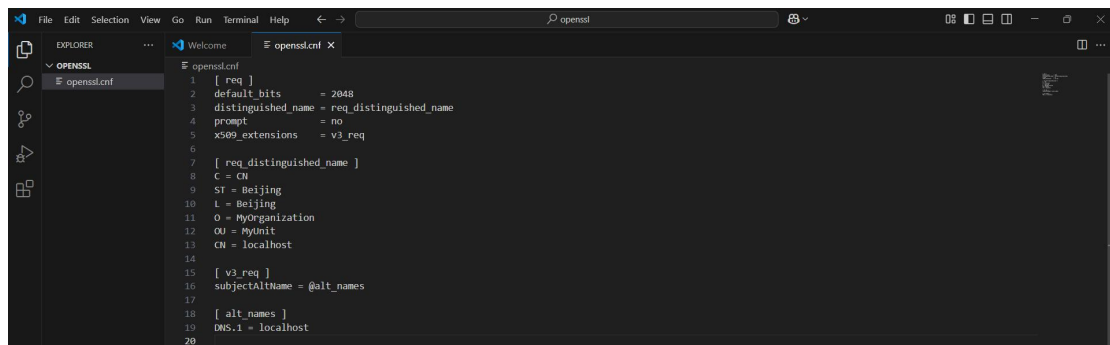
2. 创建配置文件 openssl.cnf, 打开文本编辑器 (推荐用 VS Code), 将下面内容复制进去, 选择一个容易找到的位置保存, 比如: D:\openssl\openssl.cnf。

1. [ req ]

```

2. default_bits          = 2048
3. distinguished_name    = req_distinguished_name
4. prompt                = no
5. x509_extensions       = v3_req
6.
7. [ req_distinguished_name ]
8. C = CN
9. ST = Beijing
10. L = Beijing
11. O = MyOrganization
12. OU = MyUnit
13. CN = localhost
14.
15. [ v3_req ]
16. subjectAltName = @alt_names
17.
18. [ alt_names ]
19. DNS.1 = localhost

```



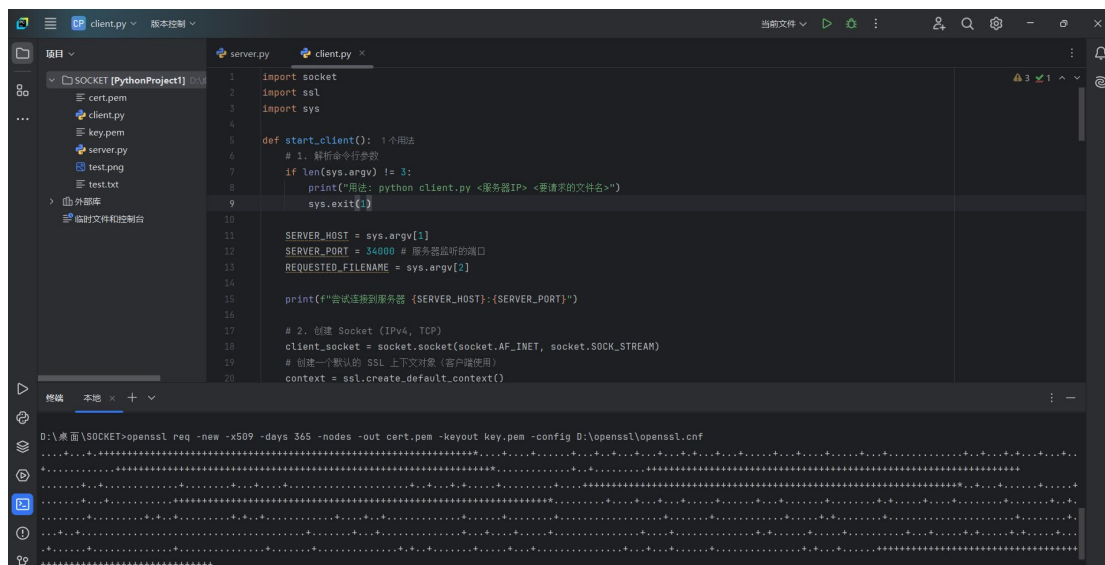
### 3. 在终端运行以下命令生成证书和私钥

```

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

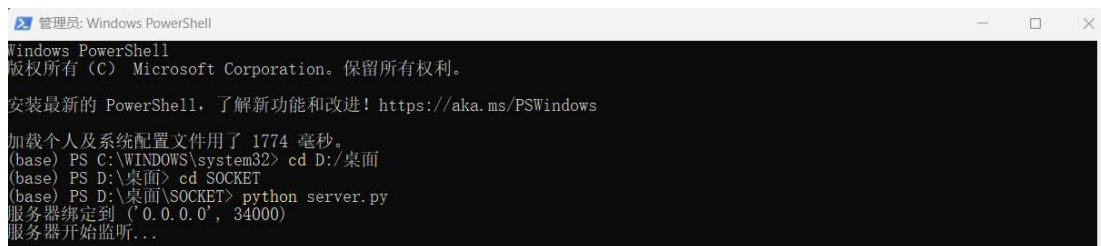
```





### (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。

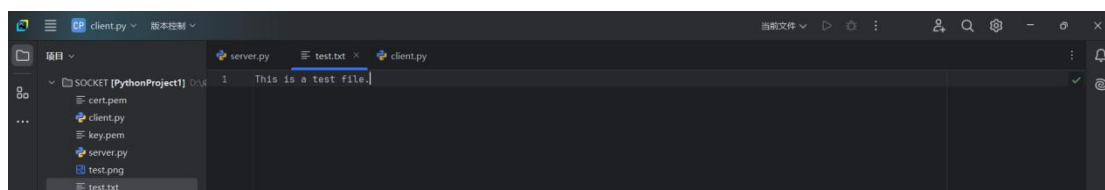


2. 再以管理员身份打开另一个终端。cd 至项目所在目录，通过规定的“python client.py <服务器 IP>”格式运行客户端程序。

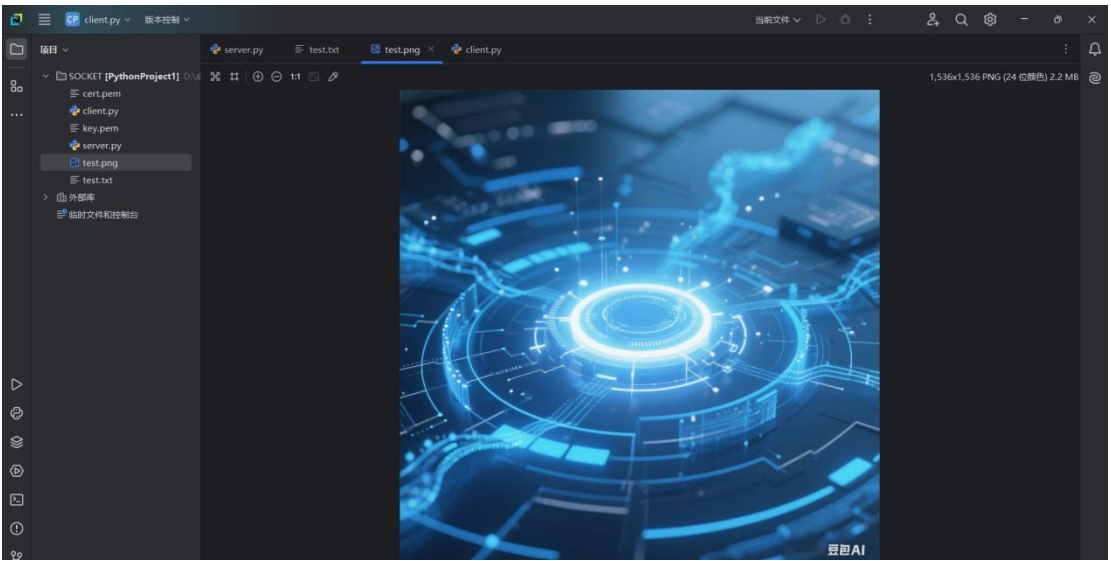


### (4) 进行文件传输

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

```
管理员: Windows PowerShell
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。
安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows
加载个人及系统配置文件用了 1774 毫秒。
(base) PS C:\WINDOWS\system32> cd D:/桌面
(base) PS D:\桌面> cd SOCKET
(base) PS D:\桌面\SOCKET> python server.py
服务器绑定到 ('0.0.0.0', 34000)
服务器开始监听...
接受来自 ('127.0.0.1', 57028) 的连接
客户端请求文件: test.txt
开始传输文件: test.txt
文件 'test.txt' 传输完毕，共发送 20 字节。

管理员: Windows PowerShell
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。
安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows
加载个人及系统配置文件用了 1831 毫秒。
(base) PS C:\WINDOWS\system32> cd D:/桌面
(base) PS D:\桌面> cd SOCKET
(base) PS D:\桌面\SOCKET> python client.py 127.0.0.1
尝试连接到服务器 127.0.0.1:34000
成功连接到服务器 127.0.0.1:34000
请输入要请求的文件名，输入 '!!!' 可退出。
请输入文件名: test.txt
服务器响应 OK，文件大小: 20 字节
开始接收文件并保存为: test.txt.bak
文件接收完毕，共接收 20 字节
新文件名: test.txt.bak
传输总字节数: 20
请输入文件名:
```

## 3. 传输 test.png 文件

```
管理员: Windows PowerShell
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。
安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows
加载个人及系统配置文件用了 1774 毫秒。
(base) PS C:\WINDOWS\system32> cd D:/桌面
(base) PS D:\桌面> cd SOCKET
(base) PS D:\桌面\SOCKET> python server.py
服务器绑定到 ('0.0.0.0', 34000)
服务器开始监听...
接受来自 ('127.0.0.1', 57028) 的连接
客户端请求文件: test.txt
开始传输文件: test.txt
文件 'test.txt' 传输完毕，共发送 20 字节。
客户端请求文件: test.png
开始传输文件: test.png
文件 'test.png' 传输完毕，共发送 2197019 字节。

管理员: Windows PowerShell
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。
安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows
加载个人及系统配置文件用了 1831 毫秒。
(base) PS C:\WINDOWS\system32> cd D:/桌面
(base) PS D:\桌面> cd SOCKET
(base) PS D:\桌面\SOCKET> python client.py 127.0.0.1
尝试连接到服务器 127.0.0.1:34000
成功连接到服务器 127.0.0.1:34000
请输入要请求的文件名，输入 '!!!' 可退出。
请输入文件名: test.txt
服务器响应 OK，文件大小: 20 字节
开始接收文件并保存为: test.txt.bak
文件接收完毕，共接收 20 字节
新文件名: test.txt.bak
传输总字节数: 20
请输入文件名: test.png
服务器响应 OK，文件大小: 2197019 字节
开始接收文件并保存为: test.png.bak
文件接收完毕，共接收 2197019 字节
新文件名: test.png.bak
传输总字节数: 2197019
请输入文件名:
```

## 4. 断开连接

```
管理员: Windows PowerShell
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows

加载个人及系统配置文件用了 1774 毫秒。
(base) PS C:\WINDOWS\system32> cd D:/桌面
(base) PS D:\桌面> cd SOCKET
(base) PS D:\桌面\SOCKET> python server.py
服务器绑定到 ('0.0.0.0', 34000)
服务器开始监听...
接受来自 ('127.0.0.1', 57028) 的连接
客户端请求文件: test.txt
开始传输文件: test.txt
文件 'test.txt' 传输完毕，共发送 20 字节。
客户端请求文件: test.png
开始传输文件: test.png
文件 'test.png' 传输完毕，共发送 2197019 字节。
('127.0.0.1', 57028) 断开连接
关闭与 ('127.0.0.1', 57028) 的连接

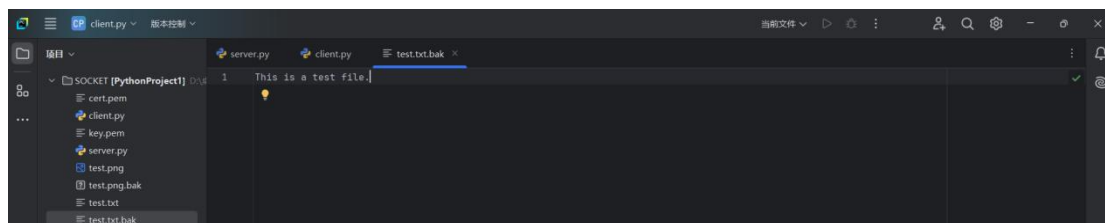
管理员: Windows PowerShell
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows

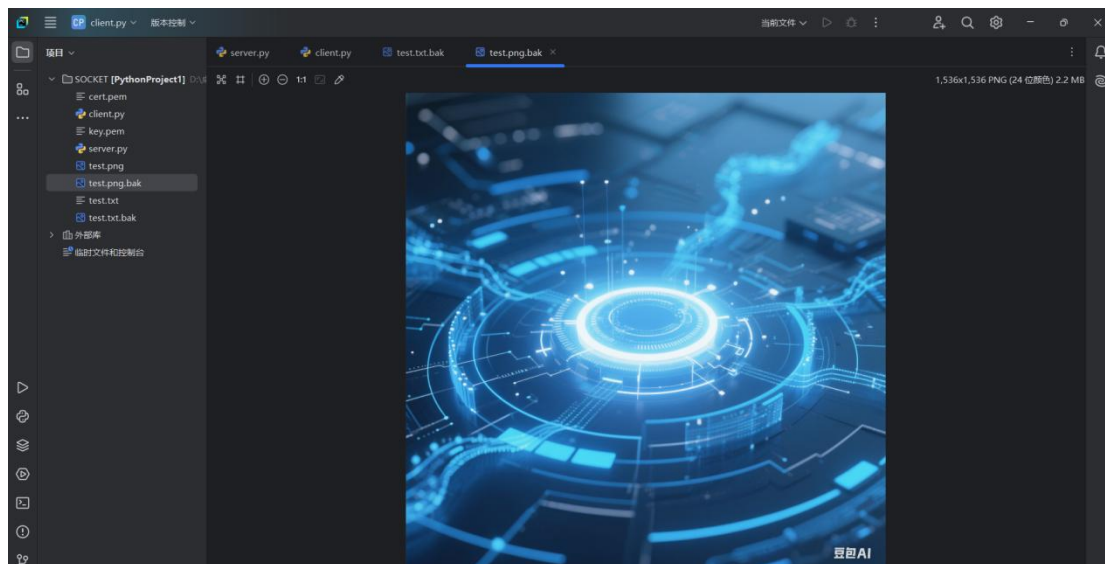
加载个人及系统配置文件用了 1831 毫秒。
(base) PS C:\WINDOWS\system32> cd D:/桌面
(base) PS D:\桌面> cd SOCKET
(base) PS D:\桌面\SOCKET> python client.py 127.0.0.1
尝试连接到服务器 127.0.0.1:34000
成功连接到服务器 127.0.0.1:34000
请输入要请求的文件名，输入 '!!!' 可退出。
请输入文件名: test.txt
服务器响应 OK，文件大小: 20 字节
开始接收文件并保存为: test.txt.bak
文件接收完毕，共接收 20 字节
新文件名: test.txt.bak
传输总字节数: 20
请输入文件名: test.png
服务器响应 OK，文件大小: 2197019 字节
开始接收文件并保存为: test.png.bak
文件接收完毕，共接收 2197019 字节
新文件名: test.png.bak
传输总字节数: 2197019
请输入文件名: !!!
收到退出指令，断开连接。
关闭连接
(base) PS D:\桌面\SOCKET>
```

## 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。

```
选择 管理员: Windows PowerShell
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows

加载个人及系统配置文件用了 1770 毫秒。
(base) PS C:\WINDOWS\system32> cd D:/桌面
(base) PS D:\桌面> cd SOCKET
(base) PS D:\桌面\SOCKET> python client.py 127.0.0.1
尝试连接到服务器 127.0.0.1:34000
连接到服务器 127.0.0.1:34000 被拒绝。请确保服务器已运行。
关闭连接
(base) PS D:\桌面\SOCKET>
```

2. 当服务器不存在时，发送如下差错报告。

```
(base) PS D:\桌面\SOCKET> python client.py 169.211.224.124
尝试连接到服务器 169.211.224.124:34000
客户端发生错误: [WinError 10060] 由于连接方在一段时间后没有正确答复或连接的主机没有反应，连接尝试失败。
关闭连接
```

3. 当客户端请求的文件不存在时，客户端和服务端均差错报告。

客户端：

```
(base) PS D:\桌面\SOCKET> python client.py 127.0.0.1
尝试连接到服务器 127.0.0.1:34000
成功连接到服务器 127.0.0.1:34000
请输入要请求的文件名，输入 '!!!' 可退出。
请输入文件名: aasdha.txt
服务器返回错误: File 'aasdha.txt' not found.
请输入文件名:
```

服务器：

```
(base) PS D:\桌面\SOCKET> python server.py
服务器绑定到 ('0.0.0.0', 34000)
服务器开始监听...
接受来自 ('127.0.0.1', 57828) 的连接
客户端请求文件: aasdha.txt
文件 'aasdha.txt' 不存在或不是一个文件
```

(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。

The image shows a Wireshark packet capture of an SSL connection between a client and a server. The client is running a Python script (client.py) and the server is running a Python script (server.py). The connection is established on port 34000. The packets show the SSL handshake process, including the Client Hello, Server Hello, and the exchange of certificates and keys. The file 'test.txt' is then transferred over the encrypted connection. The Wireshark interface shows the packet list, packet details, and packet bytes panes. The packet list pane shows the following packets:

No.	Time	Source	Destination	Protocol	Length	Info
689	15.164930	127.0.0.1	127.0.0.1	TCP	56	64470 → 34000 [SYN] Seq=...
690	15.164988	127.0.0.1	127.0.0.1	TCP	56	34000 → 64470 [SYN, ACK] Seq=...
691	15.165017	127.0.0.1	127.0.0.1	TCP	44	64470 → 34000 [ACK] Seq=...
692	15.165243	127.0.0.1	127.0.0.1	TLV1.3	561	Client Hello
693	15.165268	127.0.0.1	127.0.0.1	TCP	44	34000 → 64470 [ACK] Seq=...
694	15.166661	127.0.0.1	127.0.0.1	TLV1.3	1535	Server Hello, Change C...
695	15.166719	127.0.0.1	127.0.0.1	TCP	44	64470 → 34000 [ACK] Seq=...
696	15.167558	127.0.0.1	127.0.0.1	TLV1.3	124	Change Cipher Spec, App...
697	15.167560	127.0.0.1	127.0.0.1	TCP	44	34000 → 64470 [ACK] Seq=...
698	15.167675	127.0.0.1	127.0.0.1	TLV1.3	299	Application Data
699	15.167703	127.0.0.1	127.0.0.1	TCP	44	64470 → 34000 [ACK] Seq=...
700	15.167762	127.0.0.1	127.0.0.1	TLV1.3	299	Application Data
701	15.167772	127.0.0.1	127.0.0.1	TCP	44	64470 → 34000 [ACK] Seq=...
1133	22.954896	127.0.0.1	127.0.0.1	TLV1.3	74	Application Data
1134	22.954859	127.0.0.1	127.0.0.1	TCP	44	34000 → 64470 [ACK] Seq=...
1135	22.962990	127.0.0.1	127.0.0.1	TLV1.3	71	Application Data
1136	22.963037	127.0.0.1	127.0.0.1	TCP	44	64470 → 34000 [ACK] Seq=...
1137	23.063370	127.0.0.1	127.0.0.1	TLV1.3	86	Application Data

## 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，客户端和服务端清晰分工；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()
```