实验内容

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）。

实验环境

- Windows 11
- Python

软件设计

数据结构

- 服务器端
  - server.py
    - 服务器端代码
  - a.txt
    - 待发送文本文件
  - b.jpg
    - 待发送图像文件
  - server.py 运行参数
    - `--port` socket server 监听端口号
    - `--txt_name` 待发送文本文件名
    - `--jpg_name` 待发送图像文件名
- 客户端
  - client.py
    - 客户端代码
  - a.txt
    - 用于检查是否传输出错的文本文件
  - b.jpg
    - 用于检查是否传输出错的图像文件
  - client.py 运行参数
    - `--host` 服务器 ip 地址
    - `--port` 服务器端口
    - `--txt_name` 待发送文本文件名
    - `--jpg_name` 待发送图像文件名

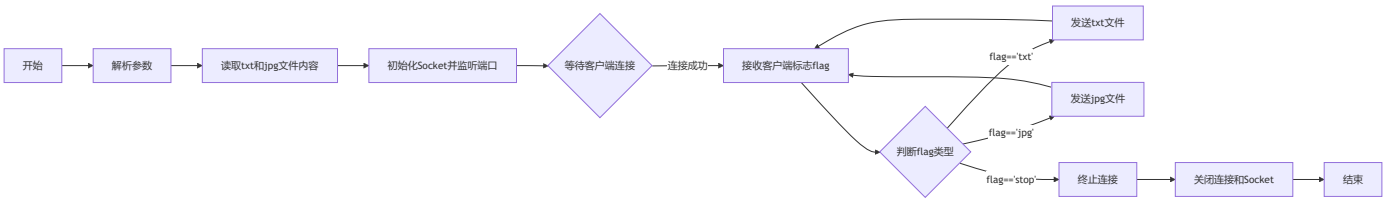
模块结构

- server.py
  - main
    - 程序入口
  - getArgs
    - 获得输入参数如监听端口、发送文本文件名、发生图像文件名等

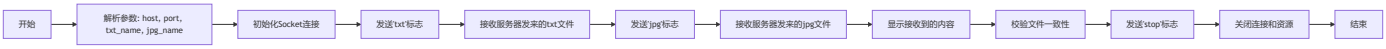
    - getFile
        - 读取文件
    - initialize
        - 初始化 socket
    - getFlag
        - 接收 Flag
        - Flag 用于指定发送的文件
        - Flag 为 `txt` 时发送 a.txt
        - Flag 为 `jpg` 时发送 b.jpg
    - sendFile
        - 发送文件
        - 约定在发送前发送待发送文件大小
    - terminate
        - 收尾处理
- client.py
    - main
        - 程序入口
    - getArgs
        - 获得输入参数如服务器 IP 与端口、文件名等
    - getFile
        - 读取文件
    - initialize
        - 初始化 socket
    - sendFlag
        - 发送 Flag
    - receiveFile
        - 先发送文件大小
        - 接收文件
    - displayFile
        - 展示文件
    - checkFile
        - 检查文件是否传输出错
    - terminate
        - 收尾处理

算法流程

- server.py



- client.py



主要功能模块的实现要点

- server.py
  - main
    a. 预加载两个文件到内存
    b. 阻塞等待客户端连接 accept()
    c. 循环处理请求：根据标志决定发送文本/图片 stop指令触发退出
  - sendFile
    a. 先发送固定64字节的文件大小信息（左对齐字符串）
    b. 再发送原始文件数据
    c. 使用 sendall() 发送
- client.py
  - receiveFile
    a. 先接收64字节的文件头（包含文件大小信息）
    b. 转换为整数 file_size 后开始循环接收
    c. 固定缓冲区大小（256字节）循环接收数据
    d. 通过 received_size 累计已接收数据量，直到等于 file_size

实验结果演示及分析

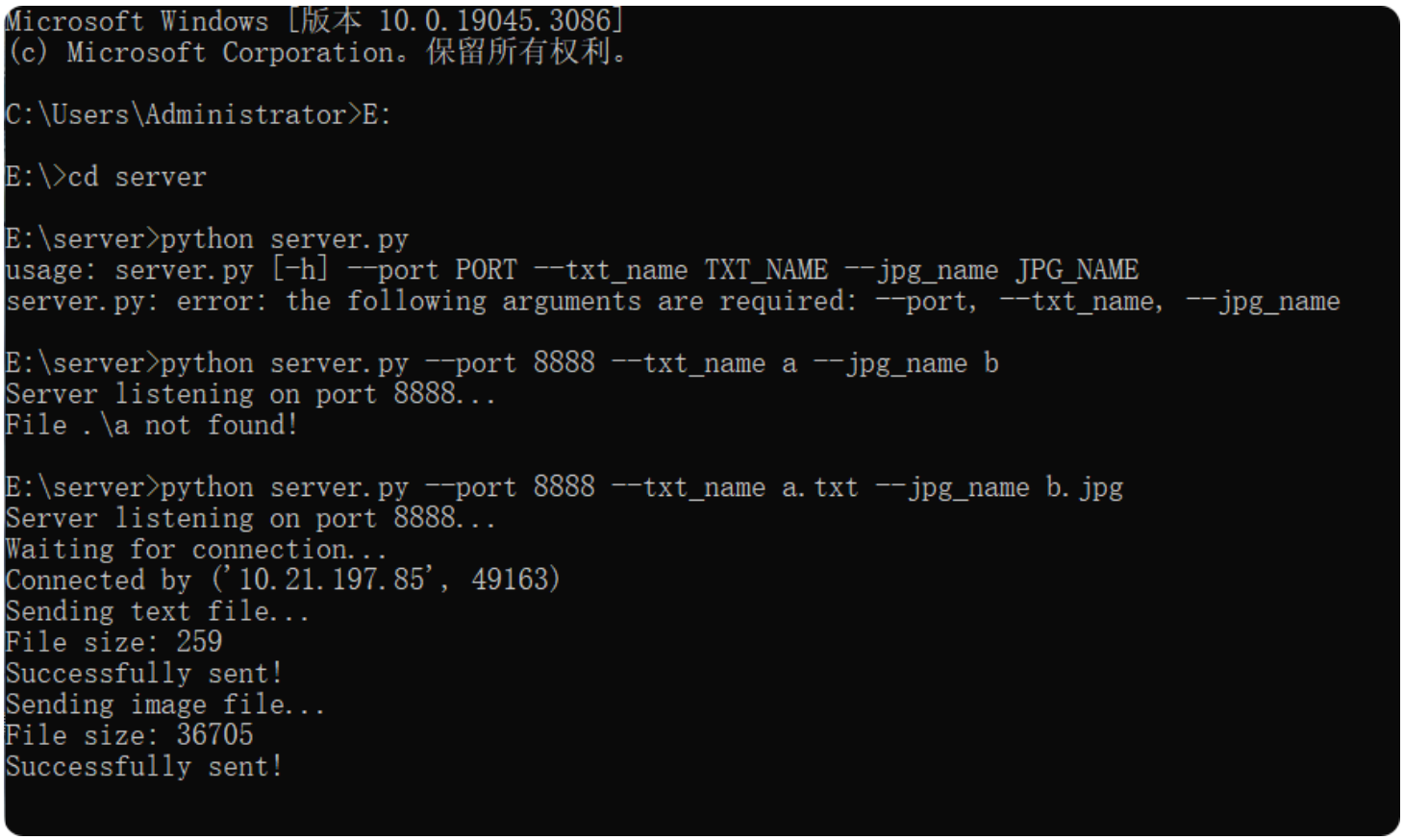
对于所实现的功能，逐个进行测试，并将输出截图

服务器端开始监听指定端口

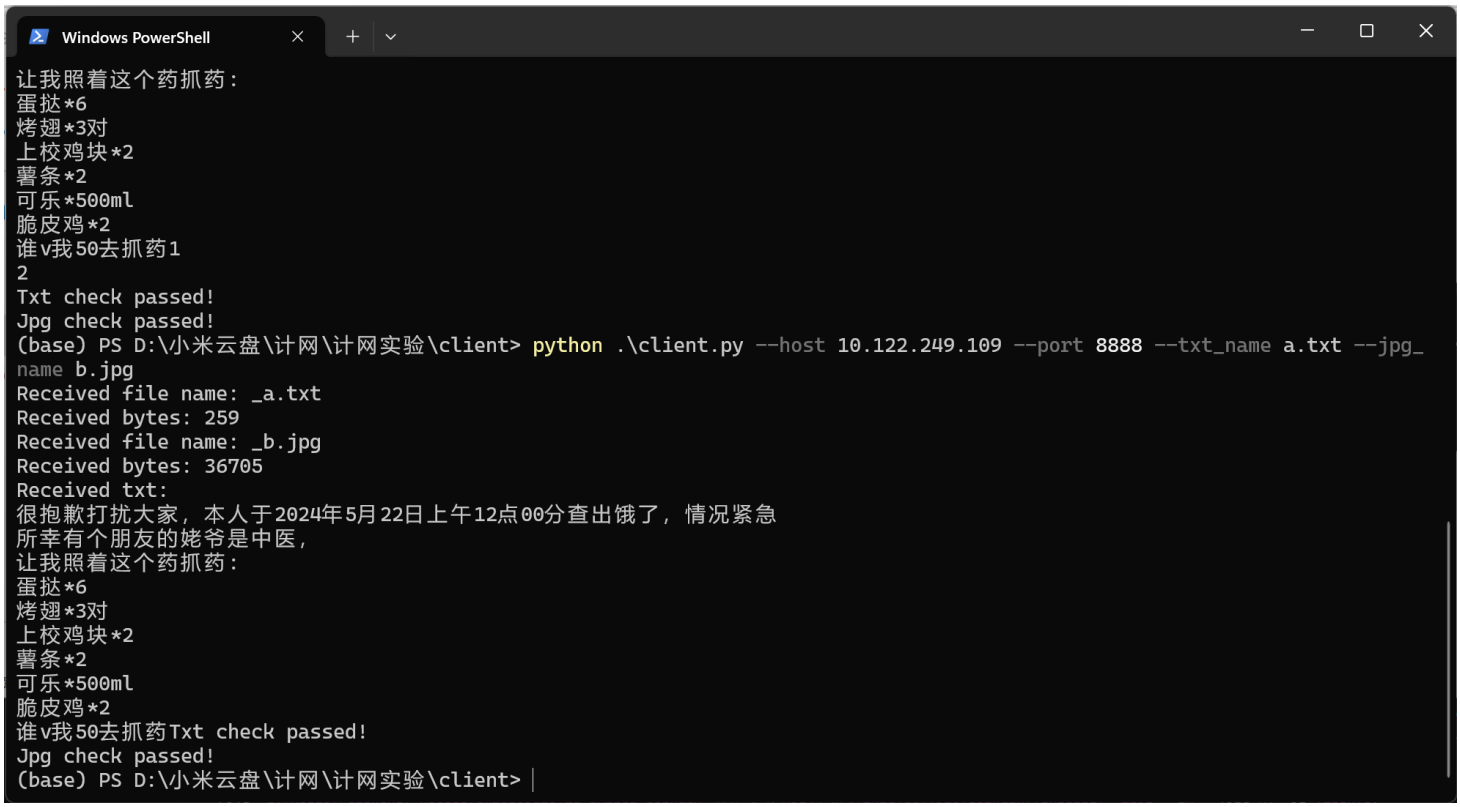
```
Microsoft Windows [版本 10.0.19045.3086]
(c) Microsoft Corporation。保留所有权利。

C:\Users\Administrator>E:

E:\>cd server

E:\server>python server.py
usage: server.py [-h] --port PORT --txt_name TXT_NAME --jpg_name JPG_NAME
server.py: error: the following arguments are required: --port, --txt_name, --jpg_name

E:\server>python server.py --port 8888 --txt_name a --jpg_name b
Server listening on port 8888...
File .\a not found!

E:\server>python server.py --port 8888 --txt_name a.txt --jpg_name b.jpg
Server listening on port 8888...
Waiting for connection...
```

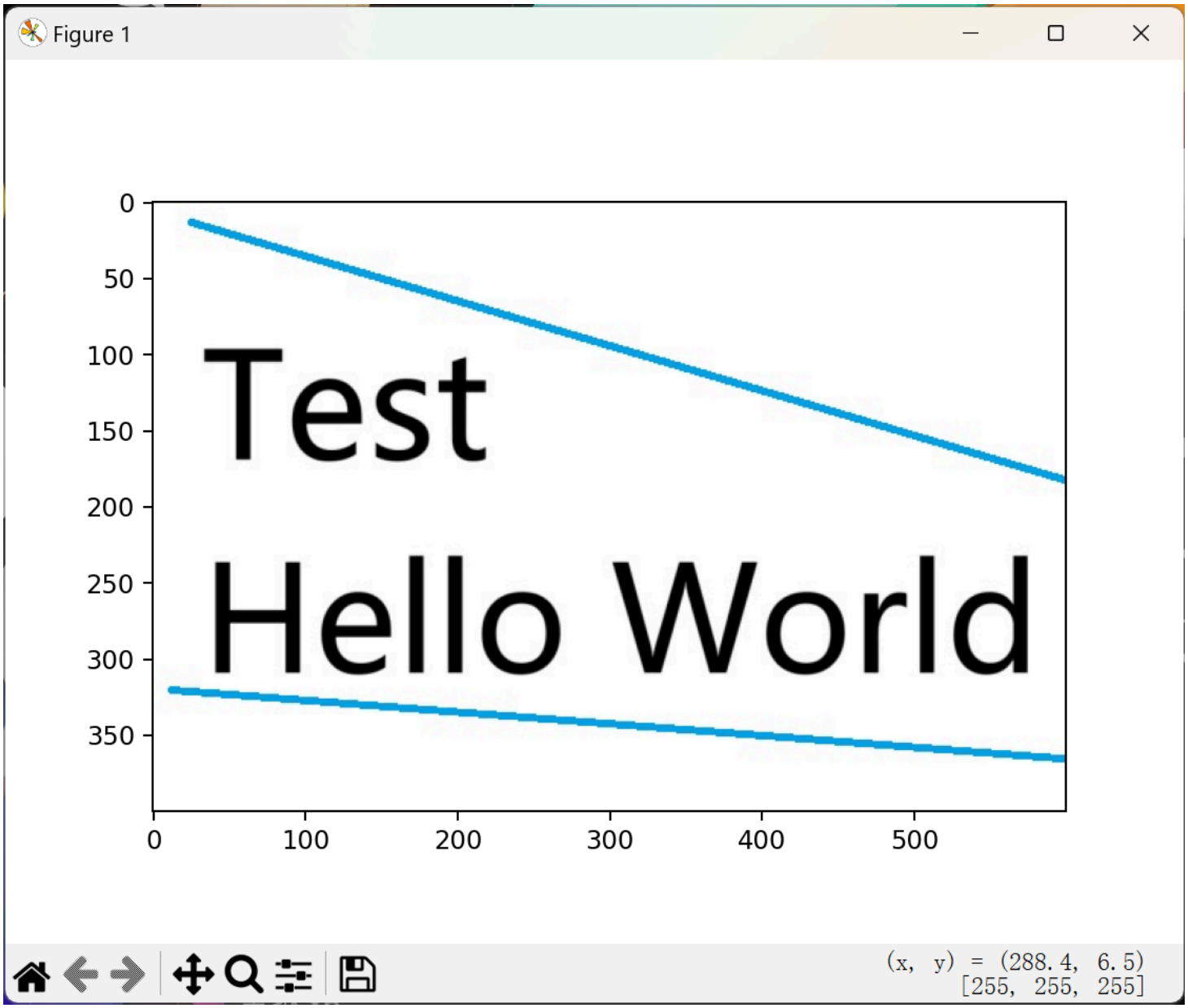
客户端连接服务器端并开始接收文件

服务器端输出客户端 IP 与端口号并发送文件



客户端显示接收到的 txt 文件和图片


Windows PowerShell
让我照着这个药抓药：
蛋挞*6
烤翅*3对
上校鸡块*2
薯条*2
可乐*500ml
脆皮鸡*2
谁v我50去抓药1
2
Txt check passed!
Jpg check passed!
(base) PS D:\小米云盘\计网\计网实验\client> python .\client.py --host 10.122.249.109 --port 8888 --txt_name a.txt --jpg_name b.jpg
Received file name: _a.txt
Received bytes: 259
Received file name: _b.jpg
Received bytes: 36705
Received txt:
很抱歉打扰大家，本人于2024年5月22日上午12点00分查出饿了，情况紧急
所幸有个朋友的姥爷是中医，
让我照着这个药抓药：
蛋挞*6
烤翅*3对
上校鸡块*2
薯条*2
可乐*500ml
脆皮鸡*2
谁v我50去抓药Txt check passed!
Jpg check passed!
(base) PS D:\小米云盘\计网\计网实验\client>



差错处理功能

服务端对于文件不存在的差错处理

```
E:\server>python server.py --port 8888 --txt_name a.txt --jpg_name b.jpg
Server listening on port 8888...
File .\a.txt not found!
```

实验总结和心得体会

完成本次实验的实际上机调试时间

- 编写代码
  - 3 小时左右
- 代码联调
  - 1 小时左右
- debug
  - 1 小时左右

调试过程中遇到的问题和解决的过程

- 客户端在接收文件时阻塞，无法执行下一步程序
  - 遇到了 TCP 粘包问题，自行设计了简单的协议
  - 服务器端先发送 64 字节的头用于传输待接收文件的大小
  - 原接收代码

```
with open(file_path, 'wb') as f:
    while True:
        data = s.recv(buffer_size)
        if not data:
            break
        f.write(data)
```

  - 改进后的代码

```
file_size_header = s.recv(64)
file_size = int(file_size_header.decode().strip())
received_size = 0
with open(file_path, 'wb') as f:
    while received_size < file_size:
        data = s.recv(buffer_size)
        if not data:
            break
        f.write(data)
        received_size += len(data)
```

总结本次实验

1. Socket 机制方面
    - 深入理解 TCP 协议
        - 通过实现客户端-服务器文件传输，掌握了 `socket.connect()` 、 `send()` 、 `recv()` 等核心方法，理解了 TCP 的可靠传输特性（如数据分块、按序到达）
    - 异常处理经验
        - 实践中学习了如何处理网络连接失败、数据传输中断等异常场景（如 `try-except` 包裹关键操作）
    - 协议设计能力
2. 协议软件设计方面
    - 设计了简单的应用层协议：
        - 用标志位（ `txt` 、 `jpg` 、 `stop` ）区分传输阶段
        - 通过固定长度头部（64 字节）传递文件大小，解决粘包问题
3. 理论学习方面
    - 网络分层模型应用
        - 实现了自定义的文件传输协议
4. 软件工程方面
    - 可维护性提升
        - 使用 `argparse` 管理参数，增强命令行工具的易用性
        - 通过函数封装（如 `getFile()` ）提高代码复用性
    - 文档与注释规范
        - 通过流程图和关键注释明确逻辑，便于后续维护。

本实验题目的设计和安排的不足

- 未能完成 ssl 的适配