



上海大学

SHANGHAI UNIVERSITY

《Python 计算》期末综合报告

题 目 基于 Python 的多线程
通信程序

学 号 22121630
姓 名 汪江豪
日 期 2025 年 5 月 11 日

一、实验目的与要求

1. 项目背景:

在网络通信日益发达的今天，即时通讯工具越来越重要，本项目通过实现一个简单的多线程聊天小程序，帮助我深入理解网络编程和多线程编程的原理和实现方法。同时，该项目也可以作为一个基础框架，为后续开发更复杂的聊天应用提供参考。

2. 项目概述:

本项目旨在开发一个基于 Python 的多线程通信程序，实现多个客户端之间的全双工实时通信。该程序采用 TCP 协议，通过 socket 编程实现客户端与服务器之间的连接和数据传输。利用多线程技术，使得服务器能够同时处理多个客户端请求，客户端能够同时接收和发送消息。

二、实验环境

Threading	Python 的标准库，用于创建和管理线程。多线程允许程序同时执行多个任务。常见类包括 Thread 类（用于创建新线程）和 Lock 类（用于处理线程同步）
Socket	Python 的标准库，用于创建网络套接字。套接字允许程序通过网络发送和接收数据。常见的函数包括 socket()（创建套接字）、connect()（连接到服务器）、bind()（绑定地址和端口）、listen()（监听连接）和 accept()（接受连接）
Tkinter(简称 tk)	Python 的标准库，用于创建图形用户界面(GUI)。提供了丰富的空间和布局管理器，用于构建窗口、按钮、标签、输入框等界面元素。Tk 是 tkinter 的常用别名。其中 messagebox 用于显示消息框，向用户显示信息、警告、错误等；font(简称 tkFont)用于处理字体，创建和配置字体对象，设置控件中文本的字体样式、大小和颜色等属性。
Datetime	Python 标准库，用于处理日期和时间，并进行格式化和运算。

三、实验内容

1. 系统架构

- a. 服务器架构：采用 socket 的 TCP 通信方式，服务器端监听特定端口，等待客户端连接请求。建立连接后，为每个客户端创建独立线程进行通信处理，维护一个字典存储已连接客户端的套接字对象和连接信息，以便实现消息的准确转发。

- b. 客户端架构：客户端同样基于 socket 编程于服务器建立连接，图形界面使用 tkinter 搭建，包括聊天窗口、消息输入框、好友列表等组件。通过创建线程实现消息的实时接收与显示，不影响用户在界面的正常操作。

2. 功能实现

a. 客户端功能：

- 用户界面显示：主界面分为好友列表区和聊天区，好友列表展示在线用户端口号，聊天区用于显示聊天消息和进行消息输入。
- 消息发送与接收：用户在输入框输入消息后点击“发送”按钮或按回车键，消息发送至服务器，服务器转发给目标客户端并显示在聊天区。
- 好友管理：支持用户通过输入特定指令添加好友端口号到好友列表，方便后续聊天时选择目标对象。
- 主题切换：提供浅色、深色、蓝色三种主题，用户可通过菜单选择切换，满足不同用户的视觉喜好。

b. 服务器功能：

- 客户端连接管理：监听客户端连接请求，接受连接后为客户端创建线程，将其添加到已连接客户端列表中。当客户端断开连接时，从列表中移除并通知其他客户端。
- 消息转发：接收客户端发送的消息，根据消息内容和目标端口号，将消息转发给相应的目标客户端。
- 服务器关闭：接收用户输入的“quit”指令，关闭服务器，断开所有客户端连接。

3. 使用流程：

- a. 服务器启动：在服务器端运行 server.py 脚本，服务器开始监听指定端口（默认为 6666）
- b. 客户端连接：在客户端运行 main.py 脚本，启动图形化聊天界面，客户端自动连接到服务器（默认服务器 IP 为 127.0.0.1，端口为 6666）。
- c. 聊天操作：客户端用户通过好友列表选择聊天对象，输入消息进行发送和接收，可随时切换聊天对象，体验一对一聊天功能。
- d. 结束聊天：用户关闭聊天窗口，客户端与服务器断开连接。

四、实验设计与实现

服务端的消息转发机制：

该模块不仅实现了核心的通信功能，还通过巧妙的算法设计确保了消息的准确转发和客户端的动态管理。

1. 动态客户端管理：

服务器需要同时管理多个客户端连接，并能够动态地添加和移除客户端。通过一个 `connected_clients` 字典，来存储所有已连接的客户端，键是客户端的地址(addr)，值是对应的套接字对象(sock)。

这种数据结构使得服务器可以快速查找和访问任意客户端的套接字对象，便于消息的

发送和接收。

服务器使用另一个字典 `client_targets` 来存储每个客户端的目标端口号。当客户端发送消息时，服务器通过该字典确定消息的接收方。

这种设计使得客户端可以动态选择聊天对象，二服务器能够根据目标端口号准确地转发消息。

2. 消息转发逻辑：

服务器需要根据客户端发送的消息内容，判断消息的类型并执行相应的操作。首先要进行消息格式解析：客户端发送的消息是以特定的格式开头，服务器通过解析消息的开头部分，确定消息的类型并执行相应的操作，如：寻找目标客户端建立对话；客户端断开连接；聊天消息转发等。

3. 多线程处理：

服务器需要同时处理多个客户端的连接和消息，这是通过多线程实现的。每当有新的客户端连接时，服务器会启动一个新线程来处理该客户端的消息，以此确保服务器能够同时处理多个客户端的通信，而不会阻塞主线程。

五、测试用例（>=2 个用例说明）

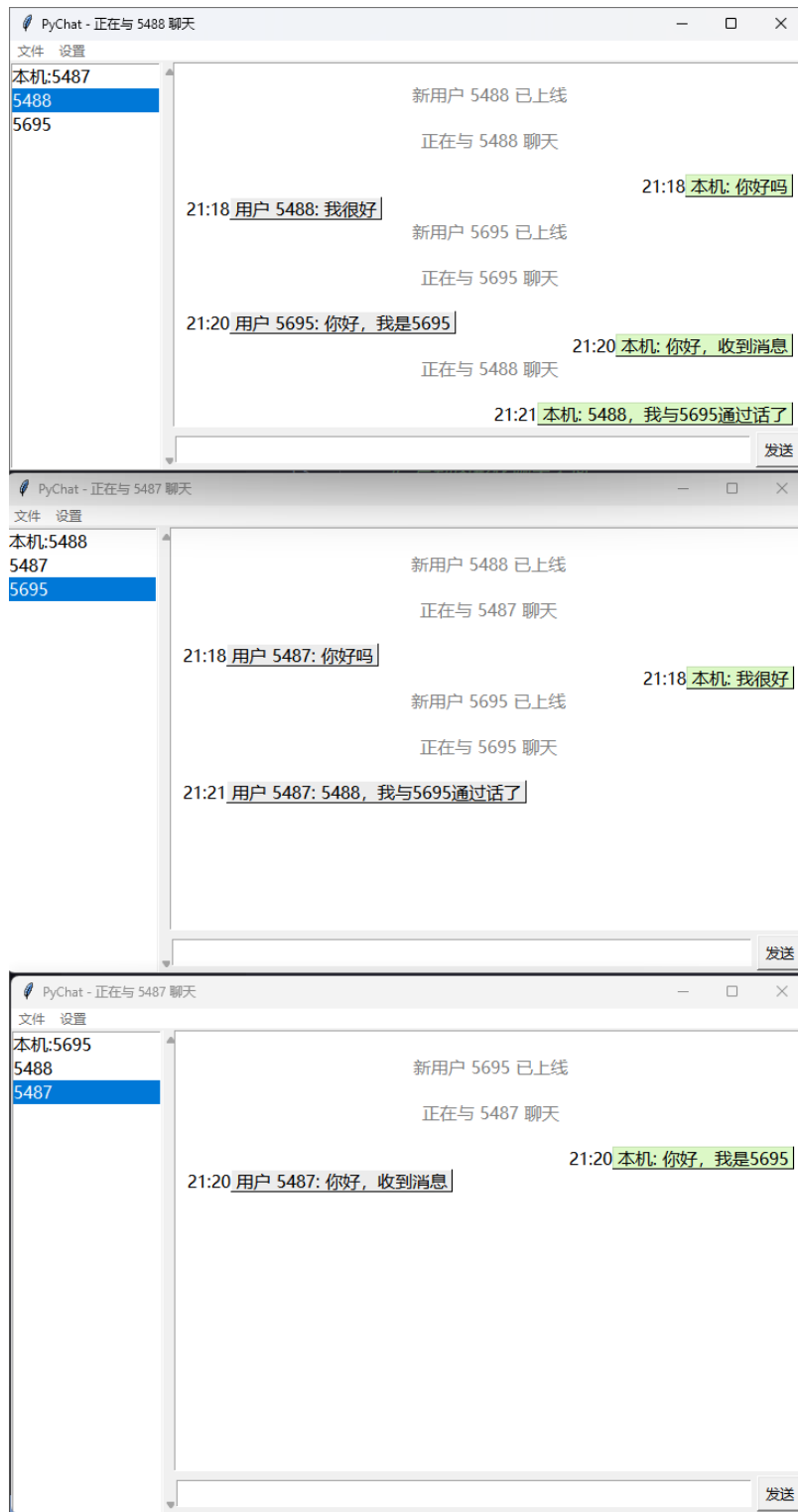
1. 双客户端间通信：

如下图，成功实现。



2. 多客户端间通信：

如下图，成功实现多线程间的全双工通信。



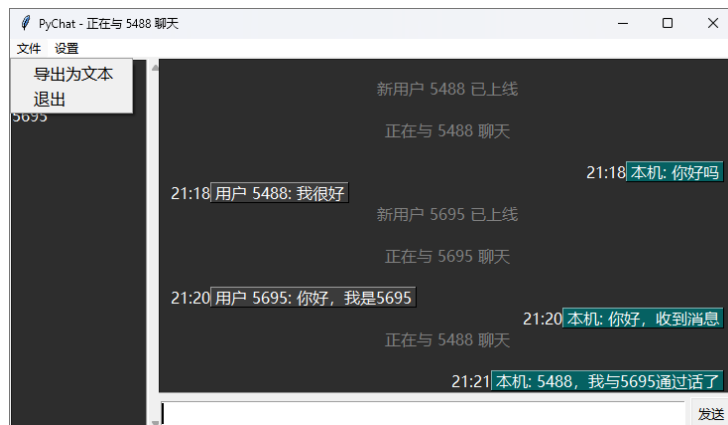
3. 主题切换和字号设置功能测试:

如下图, 1, 2 号客户端主题分别成功切换为了黑色和蓝色, 3 号客户端字号成功调大。



4. 聊天记录导出功能测试:

如下图, 通过点击菜单栏的文件, 导出为文本, 成功将聊天记录导出。



```
聊天记录_20250511_212530.txt
1
2 新用户 5487 已上线
3
4 新用户 5488 已上线
5
6 正在与 5488 聊天
7
8 21:18 本机: 你好吗
9 21:18 用户 5488: 我很好
10 新用户 5695 已上线
11
12 正在与 5695 聊天
13
14 21:20 用户 5695: 你好, 我是5695
15 21:20 本机: 你好, 收到消息
16 正在与 5488 聊天
17
18 21:21 本机: 5488, 我与5695通过话了
```

六、收获与体会

通过进行这个基于 Python 的多线程聊天小程序的项目开发, 我收获颇丰。

首先, 深入理解了 socket 编程。掌握了如何利用 socket 创建服务器和客户端, 实现基于 TCP 协议的网络通信。通过实际编写服务器和客户端的通信逻辑, 我理解了 socket 的连接、发送数据、接收数据以及关闭连接等操作了具体实现方式。

其次, 我掌握了多线程编程技巧。本项目中, 多线程技术是实现多客户端同时通信的关键。我学会了如何使用 threading 模块创建和管理线程。通过为每个客户端连接创建独立的线程, 服务器能够同时处理多个客户端的消息, 从而实现了高并发的通信。

再者, 我熟悉了图形用户界面开发。通过对 tkinter 库来构建客户端的图形用户界面, 我熟悉了 tkinter 的基本组件和布局方式。同时了解了通过事件绑定能够实现用户交互功能。

总之, 单人独立完成这个项目的过程并不容易, 但在项目的开发过程中, 我的编程能力和分析解决问题的能力得到了切实的提升, 这对我今后学习和工作大有裨益。

七、核心代码

1. 服务器转发消息逻辑:

```

# 接受客户端的连接, 创建 socket 连接对象, 并返回客户端的连接地址信息
def handle_socket(sock, addr):
    try:
        connected_clients[addr] = sock
        print("客户端" + str(addr[1]) + "已连接")

        # 处理客户端的消息
        while True:
            data = sock.recv(1024).decode()
            if data == "close connection":
                print("客户端" + str(addr[1]) + "已下线")
                del connected_clients[addr]
            elif data.startswith("to "):
                # 客户端选择目标客户端
                target_port = data.split(" ")[1]
                client_targets[addr] = target_port

                # 通知目标客户端连接到当前客户端
                target_sock = find_target_client(target_port)
                if target_sock:
                    target_sock.send(f"receiving from:{addr[1]}".encode())
                else:
                    sock.send(f"目标客户端不存在, 无法连接:{target_port}".encode())
            else:
                # 客户端发送消息给指定客户端
                target_port = client_targets.get(addr)
                if target_port:
                    target_sock = find_target_client(target_port)
                    if target_sock:
                        target_sock.send(f"{addr[1]}:{data}".encode())
                    else:
                        sock.send(f"目标客户端不存在, 无法发送:{target_port}".encode())
                else:
                    sock.send("请先选择目标客户端".encode())
    except OSError:
        return

```

2. 客户端发送消息逻辑:

```

self.send_button = tk.Button(input_frame, text="发送",
command=self.send_message, width=4, height=1, font=send_btn_font)

```

3. 客户端接收消息逻辑

```

threading.Thread(target=self.receive_message, daemon=True).start()

```