

《网络与通信》课程实验报告

实验 2：Socket 通信编程

姓名	汪江豪	院系	计算机学院	学号	22121630
任课教师	何冰		指导教师	何冰	
实验地点	计算机楼 708		实验时间	周三 7-8	
实验课表现	出勤、表现得分(10)		实验报告 得分(40)		实验总分
	操作结果得分(50)				
实验目的：					
1. 掌握 Socket 编程过程； 2. 编写简单的网络应用程序。					
实验内容：					
利用你选择的任何一个编程语言，分别基于 TCP 和 UDP 编写一个简单的 Client/Server 网络应用程序。具体程序要求参见《实验指导书》。					
要求以附件形式给出：					
<ul style="list-style-type: none">● 系统概述：运行环境、编译、使用方法、实现环境、程序文件列表等；● 主要数据结构；● 主要算法描述；● 用户使用手册；● 程序源代码；					
实验要求：(学生对预习要求的回答) (10 分)					得分：
<ul style="list-style-type: none">● Socket 编程客户端的主要步骤<ol style="list-style-type: none">1. 创建 Socket：调用 socket 函数创建一个 Socket 对象，需要指定地址族(AF_INET 对应 IPv4, AF_INET6 对应 IPv6) 和 Socket 类型 (SOCK_STREAM 对应 TCP, SOCK_DGRAM 对应 UDP)。2. 连接服务器：使用 connect 函数连接到服务器，提供服务器地址和端口号。3. 发送数据：通过 send 或 write 函数发送数据到服务器，若是 TCP，函数会阻塞直到数据发送完毕，若是 UDP，数据会立即发送，不保证到达。4. 接受数据：使用 recv 从服务器接受数据。同样，在 TCP 中，是阻塞的，直到接收到数据或关闭连接。5. 关闭 Socket：通信完成后，调用 close 关闭连接，释放资源。● Socket 编程服务器端的主要步骤<ol style="list-style-type: none">1. 创建 Socket：同样指定地址族和 Socket 类型。2. 绑定地址和端口：使用 bind 函数将 Socket 绑定到一个地址和端口上，以便客户端通过该地址和端口找到服务器。3. 监听连接请求：调用 listen 函数使 Socket 进入监听状态，准备接受来自客户端的连接请求。4. 接受连接：使用 accept 函数接受客户端的连接请求，创建一个新的 Socket 用于与客户端通信。5. 发送/接收数据：与客户端通信，通过 send 或 write 函数发送数据，通过 recv 或 read 函数接收数据。					

6. 关闭Socket: 通信完成后, 调用close函数关闭与客户端的Socket连接。如果不再接受新的连接, 还需要关闭监听Socket。

实验过程中遇到的问题如何解决的? (10分)	得分:
------------------------	-----

问题1: 使用TCP协议进行通信的程序, 如何实现多用户连接通信?

答: 使用python编写服务端程序时, 可以import threading, 使用threading创建多线程。每当accept函数监听到了连接, 就调用threading函数, 创建一个线程供一个客户端使用。

```
while True:
    try:
        sock, addr = server.accept()
        client_sockets.append(sock)
        client_thread = threading.Thread(target=handle_socket, args=(sock, addr))
        client_thread.start()
    except OSError:
        break
```

问题2: 在多用户连接到服务器时, 如何指定两个客户端之间进行通信?

答: 在服务端程序中增加判断逻辑, 假设A想与B通信, A先发送信息“to 1234”, 服务器就将1234作为目标端口, 并返回A“sending to B的端口”消息。如果B存在, 向B发送“receiving from A的端口”消息, 否则, 发送信息给A, “目标客户端不存在, 无法连接”。

```
elif data.startswith("to "):
    # 客户端选择目标客户端
    target_port = data.split(" ")[1]
    client_targets[addr] = target_port
    sock.send(f"sending to:{target_port}".encode())

    # 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())
```

在服务器向B转发A的消息时, 从键值对数组client_targets中根据源地址addr找目标端口target_port找到后发送信息。如果没找到, 发送信息“目标客户端不存在, 无法发送”。

```
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())
```

问题3: 客户端如何实现UI逻辑

答: 使用python的tkinter库编写相关UI。包括登录窗口、好友列表、聊天记录显示区域、消息发送框、发送按钮等, 并对相应空间添加点击事件, 触发相关处理函数。

```

# 创建好友列表
self.friend_list = tk.Listbox(friend_frame, width=20, height=30)
self.friend_list.pack(side=tk.LEFT, fill=tk.Y)

# 创建滚动条并绑定到好友列表
self.friend_scrollbar = tk.Scrollbar(
    friend_frame, command=self.friend_list.yview
)
self.friend_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
self.friend_list.config(yscrollcommand=self.friend_scrollbar.set)

# 绑定好友列表选择事件
self.friend_list.bind("<>ListboxSelect>>", self.on_friend_select)

# 创建聊天记录显示区域, 添加滚动条
text_frame = tk.Frame(right_frame)
text_frame.pack(expand=True, fill=tk.BOTH)

self.text_display = tk.Text(
    text_frame, width=70, state=tk.DISABLED, wrap=tk.WORD
)
self.scrollbar = tk.Scrollbar(text_frame, command=self.text_display.yview)
self.scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

self.text_display["yscrollcommand"] = self.scrollbar.set
self.text_display.pack(side=tk.LEFT, expand=True, fill=tk.BOTH)

# 创建消息输入框
input_frame = tk.Frame(right_frame)
input_frame.pack(fill=tk.X, expand=False)

self.message_entry = tk.Entry(input_frame)
self.message_entry.pack(side=tk.LEFT, padx=10, fill=tk.X, expand=True)
self.message_entry.focus_set()

# 创建发送按钮
self.send_button = tk.Button(
    input_frame, text="发送", command=self.send_message
).pack(side=tk.LEFT)

# 将回车与发送消息绑定
self.message_entry.bind("<Return>", self.send_message_event)

```

问题 4: 如何主动关闭服务器?

答: 添加线程, 监听键盘输入, 如果输入 quit, 则主动关闭 sock, 并关闭服务器。

```

# 监听输入
input_thread = threading.Thread(target=input_listening, args=(server,))
input_thread.daemon = True
input_thread.start()

```

```
# 监听输入，如果输入quit则关闭服务器
def input_listening(server):
    while True:
        cmd = input()
        if cmd == "quit":
            print("服务器即将关闭...")
            for sock in connected_clients.values():
                sock.close()
            server.close()
            break
```

本次实验的体会（结论）（10分）

得分：

本次 socket 通信实验对我来说无疑是充满挑战且意义非凡的。整个实验过程从构想，思考，查阅资料，自己动手亲自实现。不断学习 socket 相关编程知识，学习程序调试知识，学习用 python 编写 UI 界面耗时很多天才终于独立完成，做出了自己的第一个基于 socket 的聊天通信程序。

在实验过程中，我更加深刻地意识到了计算机网络和操作系统的分层理念。下层向上层暴露接口，上层使用下层的服务，socket 也是如此。

本次实验克服了诸多困难。对于 UI 界面的编写，从一开始茫然无措，到后来不断查阅资料，不断排查 BUG，优化 UI 界面，优化代码逻辑，到最后都一一钻研解决。我选择了易于使用的 python 语言编写了程序。了解在通信过程中，如何解决多线程、身份验证，主动管理连接等内容。同时，我的 tkinter 库的 UI 编写有了更深刻的认识，知道了程序界面如何绑定键盘和鼠标事件，触发处理函数，如何对代码进行分模块编写等等，这些都极大地提高了我的代码编写能力和 DEBUG 能力。

同时，我对于 TCP 和 UDP 的认识更进一步。在传输层中，TCP 是面向连接的、可靠的通信协议，在数据传输前，需要建立连接，数据顺序到达接收方。UDP 则是无连接、不可靠的传输层协议，也因此，所需开销也比 TCP 小。

总之，本次 socket 通信实验，提升了我的自学能力、编程能力，DEGUB 能力，加深了我对于计算机网络知识的理解。是一次充满挑战且充实的实验。

思考题：（10分）

思考题 1：（4分）

得分：

你所用的编程语言在 Socket 通信中用到的主要类及其主要作用。

答：外部库：

```
1 ~import socket
2 import threading
3 import tkinter as tk
4 from tkinter import messagebox
5 import tkinter.font as tkFont
```

socket:进行通信相关函数的实现。

Threading: 创建多个线程，如多客户端的连接，监听键盘输入等。

Tkinter: 用于编写 UI 的主要库。

Messagebox: 用于显示提示框信息。

tkFont: 设置界面相关字体。

自定义类: ClientSocket 封装了 socket 的一些函数，包括初始化，连接服务器，发送消息，接收消息，关闭连接等，便于使用。

```
# 该文件用于实现基于tcp方式的socket客户端socket通信逻辑
class ClientSocket:
    def __init__(self, server_ip, server_port):
        self.server_ip = server_ip
        self.server_port = server_port
        self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.client_socket.connect((self.server_ip, self.server_port))
        self.port = self.client_socket.getsockname()[1]

    # 发送消息
    def send_message(self, message):
        self.client_socket.send(message.encode())

    # 接收消息
    def receive_message(self):
        return self.client_socket.recv(1024).decode()

    # 关闭连接
    def close(self):
        self.client_socket.close()
        return
```

LoginWindow: 登录窗口界面逻辑，登陆后能触发启动通信窗口。

```
# 这里实现socket客户端登录到服务器的验证逻辑
class LoginWindow:
    def __init__(self, root):
        # 设置全局字体
        default_font = tkFont.Font(family="Microsoft YaHei", size=12)
        root.option_add("*Font", default_font)

        self.root = root
        self.root.title("登录窗口")

        # 设置窗口大小和位置
        width = 300
        height = 200
        top = 400
        left = 700

        # 设置窗口的大小和位置
        self.root.geometry(f"{width}x{height}+{left}+{top}")

        tk.Label(root, text="用户名:").pack(pady=5)
        self.username_entry = tk.Entry(root)
        self.username_entry.pack(pady=3)
        self.username_entry.focus_set()
```

ChatWindow: 通信界面 UI 和处理逻辑

```

# 该文件用于实现基于tcp方式的socket客户端聊天窗口逻辑
class ChatWindow:
    def __init__(self, root, username, server_ip, server_port):
        # 设置全局字体
        default_font = tkFont.Font(family="Microsoft YaHei", size=12)
        root.option_add("*Font", default_font)

        # 创建窗口
        self.root = root
        self.root.title(f"通信窗口 - {username}")

        # 设置窗口位置
        top = 270
        left = 480

        # 设置窗口的大小和位置
        self.root.geometry(f"+{left}+{top}")

```

思考题 2: (6 分)

得分:

说明 TCP 和 UDP 编程的主要差异和特点。

TCP:

面向连接: 数据传输前, 必须建立连接。连接建立和终止需要经过三次握手和四次挥手。

可靠: 保证数据按顺序传输, 如果发生丢包, 会重新传输。通过校验、确认应答、序列号等机制来保证数据的完整性。

流量控制: 使用滑动窗口进行流量控制, 避免发送方过快发送数据导致接收方来不及处理。

UDP:

无连接: 在发送数据前不需要建立连接。每次发送数据时, 都会封装成一个数据报, 独立传输。

不可靠: UDP 不保证数据包的顺序和完整性。如果数据包丢失或出错, UDP 不会自动重传。

无流量控制: 发送方发送数据了速率不会因为接收方的处理能力而受到限制。

差异:

- TCP 需要连接的建立和终止, UDP 不需要建立连接。
- TCP 保证数据顺序和可靠性, 适合传输大量数据, UDP 不保证数据顺序和可靠性, 适合传输少量数据, 如视频流和在线游戏。
- TCP 因为有更多机制, 性能开销较大, UDP 因为简单, 性能开销较小, 传输速度快。
- TCP 适用于需要可靠传输的场景, 如文件传输, 邮件传输等, UDP 适用于对实时性要求较高的场景, 如实时视频, 直播, 音频传输, 在线游戏等。

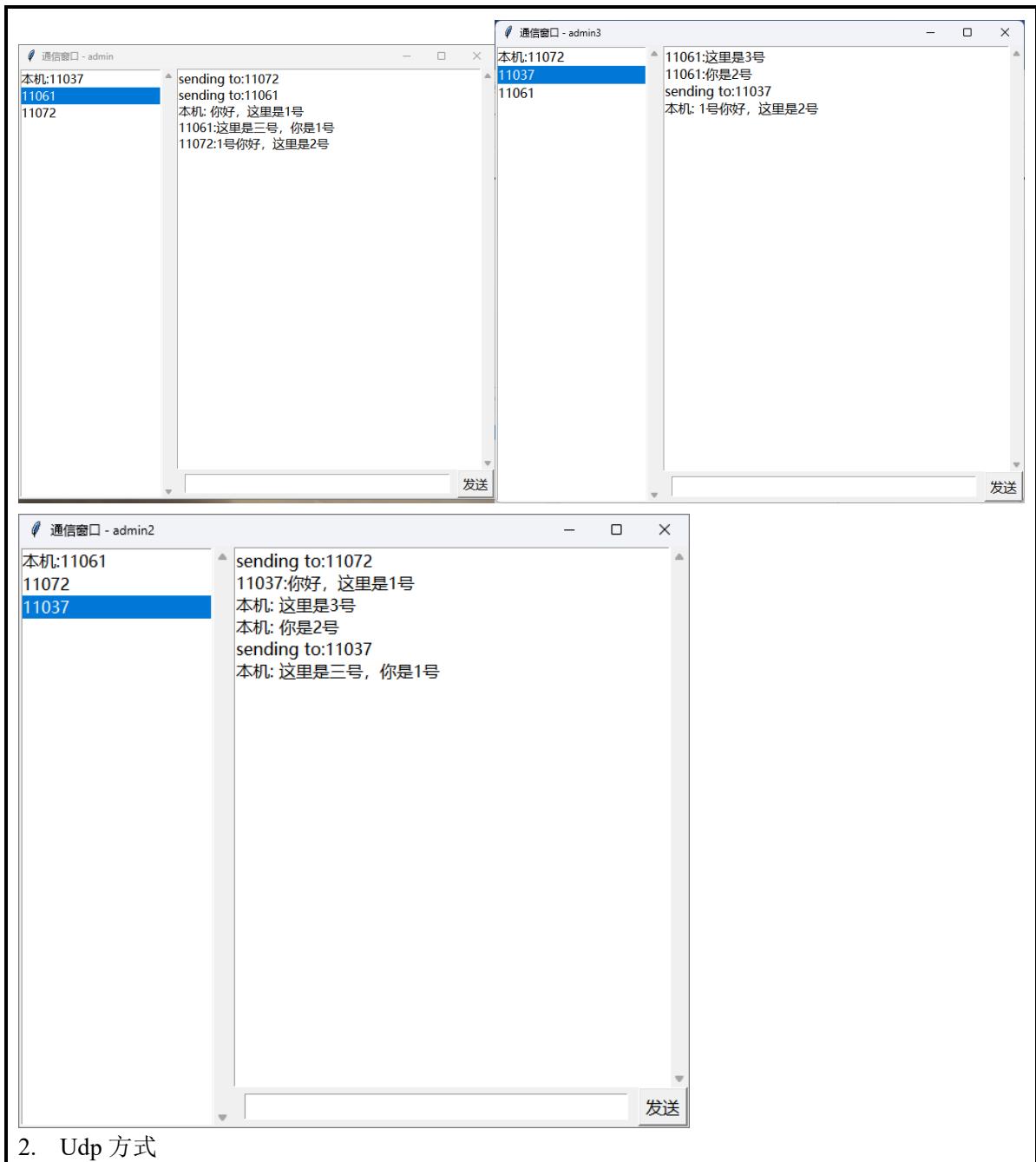
程序截图:

1. Tcp 方式

登录界面:



多客户端全双工通信:



2. Udp 方式

The image displays four terminal windows arranged in a 2x2 grid, illustrating a client-server communication process. The top-left window shows a client sending a message to a server. The top-right window shows the server receiving the message. The bottom-left window shows another client sending a message. The bottom-right window shows the server receiving this second message and then closing the connection.

Top Left Window (Client 1):

```
请输入要发送的内容: 这里是1号  
发送成功  
服务器已收到你的消息:这里是1号  
请输入要发送的内容: |
```

Top Right Window (Server):

```
请输入要发送的内容: 这里是2号  
发送成功  
服务器已收到你的消息:这里是2号  
请输入要发送的内容: |
```

Bottom Left Window (Client 2):

```
请输入要发送的内容: 这里是3号  
发送成功  
服务器已收到你的消息:这里是3号  
请输入要发送的内容: |
```

Bottom Right Window (Server):

```
服务器启动成功,端口:6666  
客户端60380发来消息:这里是1号  
客户端63028发来消息:这里是2号  
客户端62125发来消息:这里是3号  
quit  
服务器即将关闭...  
服务器已关闭  
请按任意键继续... . . .
```

指导教师评语:

日期: