计算机网络专题实验报告8（共6页）

实验名称：Socket网络编程实验　　时间： 2024 年5 月 24 日　早□ 午□ 晚☑

一.实验目的

1) 掌握 Sockets 的相关基础知识，学习 Sockets 编程的基本函数和数据类型
2) 掌握 UDP、TCP Client/Server 模式的通信原理。
3) 掌握 socket 编程命令

二.实验内容

1) 实现一个简单的客户机/服务器程序，基于 TCP 和 UDP 协议分别实现。

三.实验原理

互联网的 Client/Server 模式的工作原理如下，以 TCP 服务器为例说明，UDP 服务器略有不同。客户端也是如此。

1）服务器

服务器先创建一个套接字（Socket），并将该套接字和特定端口绑定，然后服务器开始在此套接字上监听，直到收到一个客户端的连接请求，然后服务器与客户端建立连接，连接成功后和该客户端进行通信（相互接收和发送数据），进行用户信息验证，并返回验证信息。最后，服务器和客户端断开连接，继续在端口上监听。

2）客户端

客户端创建一个套接字，里面包含了服务器的地址和端口号，客户端的端口号由系统自动分配，不需要指明。和服务器建立连接，如果连接成功则 socket 创建成功。然后客户端发送用户名和密码，等待验证。通信结束后主动断开连接，释放资源。

四.实验步骤

步骤 1：编写 server 端程序

步骤 2：编写 client 端程序

步骤 3：client 端和 server 端实现互联通信， 验证用户登录验证用户登录信息。例如，客户端发送用户名和密码，如若信息正确服务器端返回：送用户名和密码，如若信息正确服务器端返回：送用户名和密码，如若信息正确服务器端返

回：“*信息正确*”；否则，服务器端返回：“*用户名或密码错误请再次输入*”。（提示信息不唯一，可自由改变）

五.代码展示

我们选择 Python 作为编程语言。在客户端，我们使用 PyQt5 创建图形用户界面，使用户能够以更人性化的方式与软件交互。用户可以输入消息，通过点击 "发送" 按钮或按 enter 键来发送消息，此消息通过 socket 发送到服务器。服务器处理接收到的消息，并将其转发给所有在线的客户端。当服务器收到文件时，它会保存该文件，并将文件信息广播给所有客户端，其他客户端都可以告知要接收的文件，服务器会另启一个线程将文件发送给该客户端。

对于多用户同时在线的场景，使用了 Python 的多线程编程模式。每当有一个新的客户端连接到服务器时，创建一个新的线程来专门处理这个客户端的消息。这样，服务器可以同时处理多个客户端的请求。

TCP 实现的部分核心代码如下：

client 端：主要包含一个 Client 类，其主要组件包括 LoginDialog 以实现登录/注册功能，以及 add_ui 方法以创建用户界面。其中 send_msg，recv_msg 方法等实现消息的接收与发送。

```
    def handle_login(self):
        dialog = LoginDialog()
        if dialog.exec_() == QDialog.Accepted:
            mode, username, password = dialog.get_credentials()
            print(mode, username, password)
            self.client.send(str(mode).encode())
            time.sleep(0.1)
            self.client.send(username.encode())
            time.sleep(0.1)
            self.client.send(password.encode())
            time.sleep(0.1)
            response = self.client.recv(1024).decode()
            print(response)
            if "失败" in response:
                QMessageBox.warning(self, "登录失败", response)
                exit()
            QMessageBox.information(
                self, "成功", "登录成功!" if mode == 1 else "注册并登录成功!"
            )
            return True
        return False
class LoginDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("登录/注册")
        self.setGeometry(700, 400, 300, 200)

        layout = QGridLayout()

        # 添加选择按钮
```

```
        self.mode_combo = QComboBox()
        self.mode_combo.addItem("登录", 1)
        self.mode_combo.addItem("注册", 0)
        layout.addWidget(QLabel("选择模式:"), 0, 0)
        layout.addWidget(self.mode_combo, 0, 1)

        # 用户名和密码输入
        self.username = QLineEdit()
        self.password = QLineEdit()
        self.password.setEchoMode(QLineEdit.Password)
        layout.addWidget(QLabel("用户名:"), 1, 0)
        layout.addWidget(self.username, 1, 1)
        layout.addWidget(QLabel("密码:"), 2, 0)
        layout.addWidget(self.password, 2, 1)

        # 登录按钮
        login_button = QPushButton("确定")
        login_button.clicked.connect(self.accept)
        layout.addWidget(login_button, 3, 1)

        self.setLayout(layout)

    def get_credentials(self):
            return self.mode_combo.currentData(), self.username.text(),
self.password.text()
```

server 端：维护一个 Server 类，主要方法包括 authenticate 方法用于认证用户，get_conn，get_msg 方法用于接收客户端的连接请求和消息。其中运用了多线程的技术。

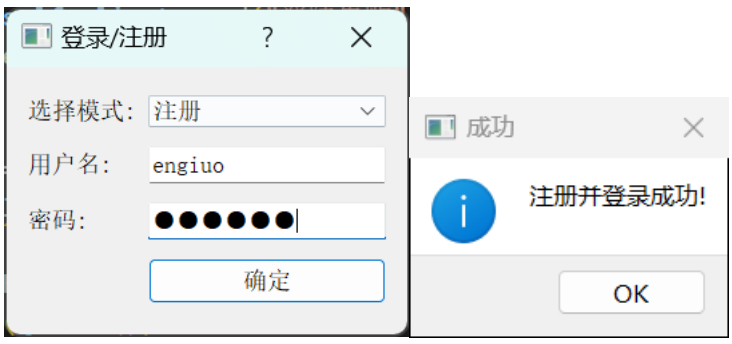
```
    def authenticate(self, client):
        mode = client.recv(1024).decode()
        print(mode)
        if mode == "0":
            username = client.recv(1024).decode()
            password = client.recv(1024).decode()
            if username in self.user_credentials:
                client.send("注册失败，已存在该用户名".encode())
                client.close()
                return None
            else:
                self.user_credentials[username] = password
                client.send("注册并登录成功".encode())
                return username
        username = client.recv(1024).decode()
        print(username)
        password = client.recv(1024).decode()
        print(password)
        print(mode, username, password)
        if (
            username in self.user_credentials
            and self.user_credentials[username] == password
        ):
            client.send("登录成功".encode())
            return username
        else:
            client.send("登录失败，请重新连接并尝试".encode())
```

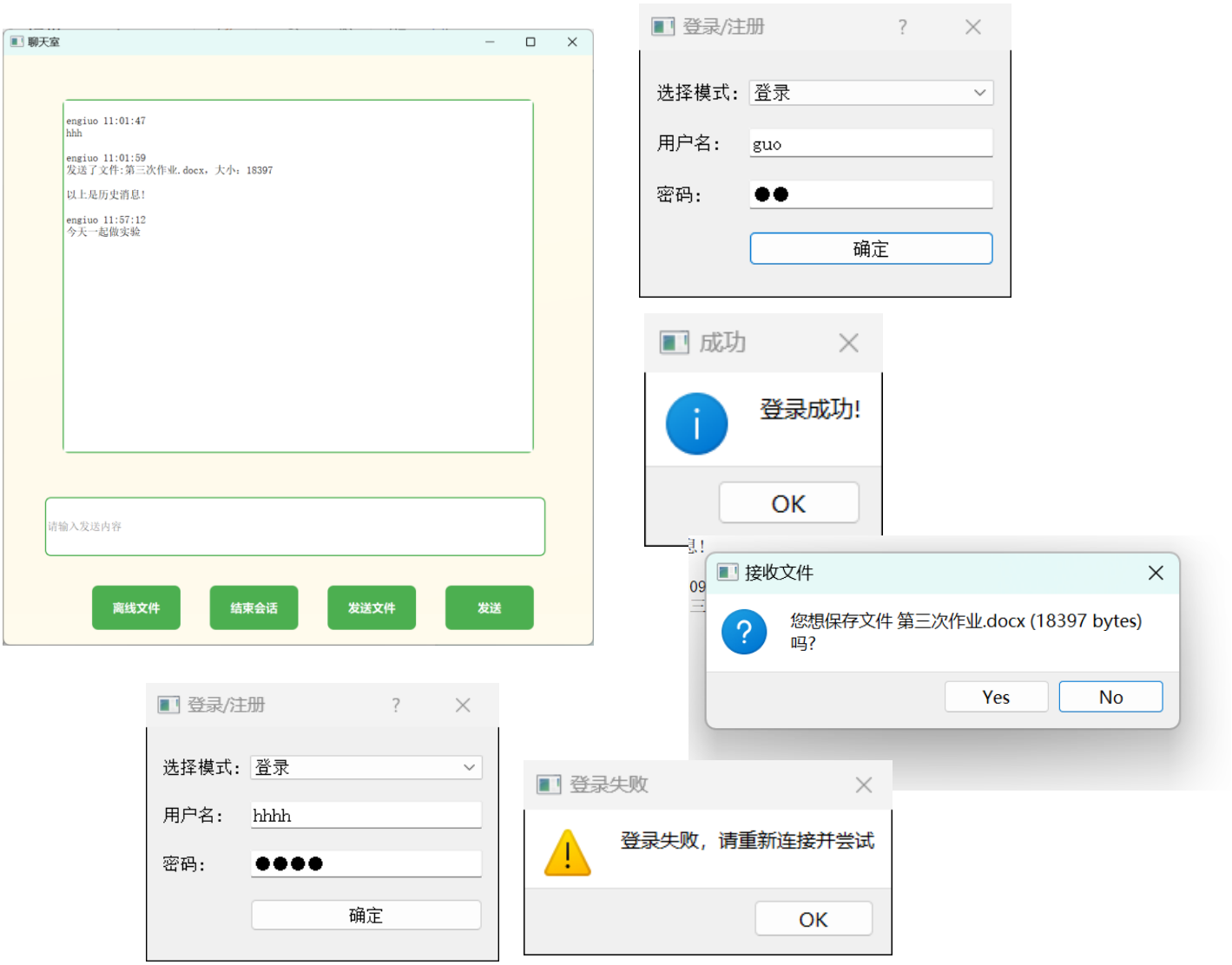
```
            client.close()
            return None

    # 监听客户端链接
    def get_conn(self):
        while True:
            client, address = self.server.accept()
            print(address)
            username = self.authenticate(client)
            is_login = client.recv(1024).decode()
            if username and "成功" in is_login:
                print(f"{username} ({address}) 已连接")
                self.clients.append(client)
                self.clients_name_ip[address] = username
                    Thread(target=self.get_msg, args=(client, username,
address)).start()
```

UDP 实现基本同理，部分代码展示如下：

```
    def handle_login(self):
        dialog = LoginDialog()
        if dialog.exec_() == QDialog.Accepted:
            mode, username, password = dialog.get_credentials()
            print(f"Mode: {mode}, Username: {username}, Password: {password}")
            self.client.sendto(f"{mode}".encode(), (IP, MESSAGE_PORT))
            time.sleep(0.1)
            self.client.sendto(username.encode(), (IP, MESSAGE_PORT))
            time.sleep(0.1)
            self.client.sendto(password.encode(), (IP, MESSAGE_PORT))
            time.sleep(0.1)
            response, _ = self.client.recvfrom(1024)
            response = response.decode()
            print(response)
            if "失败" in response:
                QMessageBox.warning(self, "登录失败", response)
                exit()
            QMessageBox.information(
                self, "成功", "登录成功!" if mode == 1 else "注册并登录成功!"
            )
            return True
        return False
```

六.结果展示





支持注册登录、聊天室、文件传输等功能。

七.结果分析和互动讨论

1）比较 TCP UDP 两种协议的不同。

TCP 是面向连接的协议，提供可靠的数据传输。它确保数据按顺序到达，并且没有丢失或重复。这使得它适合用于登录系统和聊天系统等需要确认每次传输确实到达的应用。在数据传输之前，客户端和服务器必须建立连接（通过三次握手）。TCP 使用流量控制和拥塞控制机制来防止发送端传输速度过快，超过接收端的处理能力。

UDP 是一种无连接的协议，数据在发送前无需建立连接。每个数据包（称为数据报）是独立的。UDP 传输的数据没有保证到达目的地，也没有顺序或重传机制，可能会丢失或顺序混乱。由于没有连接建立和流量控制机制，UDP 传输速度更快，开销更小。

2）如何服务器能实现循环监听？

多线程处理：每次接受新的客户端连接时，启动一个新的线程来处理该客户端的消息。

无限循环：在 get_conn 方法中使用无限循环来持续监听新的客户端连接。在 get_msg 方法中使用循环来持续接收和处理客户端消息。

3）比较两种协议在代码层面的区别。

TCP：

使用 socket.SOCK_STREAM 类型。

需要调用 listen()和 accept()来处理连接。

使用 send()和 recv()函数传输数据。

UDP：

使用 socket.SOCK_DGRAM 类型。

无需建立连接，直接使用 sendto()和 recvfrom()函数传输数据。

4）在服务器端实现多线程的好处是什么？如何在服务端实现多线程？

多线程编程在服务器端具有多个显著的优势：

多线程允许服务器同时处理多个客户端连接。每个客户端连接都在一个单独的线程中处理，因此当一个客户端在等待或处理数据时，其他客户端的请求不会被阻塞。

由于多线程可以并发执行，因此服务器能更快速地响应客户端的请求。即使一个线程在进行长时间的操作（如文件上传或下载），其他线程仍然能够处理新的客户端请求。

多线程程序可以更好地利用多核处理器的性能。每个线程可以在不同的 CPU 核上运行，从而提高整个系统的效率。

借助 Python 的 threading 模块，可以很容易地创建和管理多个线程，使服务器在处理多个连接时变得更加高效和灵活。

5）TCP 协议和 HTTP 协议的区别和联系；

TCP 协议：

面向连接，提供可靠的数据传输。

传输层协议，确保数据包按序到达。

HTTP 协议：

应用层协议，基于请求-响应模型。

无状态协议，每次请求独立。

HTTP 通常在 TCP 之上运行，通过 TCP 提供可靠的传输。TCP 负责数据包的传输和完整性，HTTP 负责数据格式和交换。TCP 确保数据传输可靠，HTTP 构建于 TCP 之上用于网络通信和数据交换。