四 川 大 学 计 算 机 学 院、软 件 学 院

实 验 报 告

学号：2023141460321 姓名：孙谦昊 专业：计算机科学与技术 班级：行政七班 第 12 周

课程名称	网络编程	实验课时	5-7 节
实验项目	Socket 编程练习：用户注册与登录	实验时间	2025.05.08
实验目的	1. User Registration - 用户注册 2. User Authentication - 用户登录认证		
实验环境	Visual Studio Code		
实验内容（算法、程序、步骤和方法）	借助 Python 语言，分别实现： **一、服务端程序** 1) 若注册请求通过，服务器在本地 passwd文件中保存新用户的用户名和密码的 hash 值，并返回 registration_response_msg，通知客户端注册成功。 2) 若注册请求未通过（用户重名），并返回 registration response msg，通知客户端注册失败。 3）服务器在本地 passwd文件中查找对应用户记录，并计算 login request msg 中密码的 hash 值，若 hash一致，则验证通过，服务器返回 login response msg，通知客户端登录成功。 4）若身份认证失败（用户名不存在或密码不正确），服务器返回 login response msg，通知 客户端注册失败，并在返回消息中描述失败原因。 **二、客户端程序** 1）客户端连接服务器，并发送 registration request msg 进行注册，然后客户端等待注册结果。 2) 客户端连接服务器，并发送 login request msg 进行登录，然后客户端等待登录结果。		

三、服务器端程序设计思路

首先定义了消息类型编号和服务器地址及端口号等信息，用于与客户端相关信息保持一致和确保链接：

```
COMMAND_ID_REG_REQUEST = 1
COMMAND_ID_REG_RESPONSE = 2
COMMAND_ID_LOGIN_REQUEST = 3
COMMAND_ID_LOGIN_RESPONSE = 4

SERVER_ADDRESS = "127.0.0.1"
SERVER_PORT = 12345
```

随后在主函数中将服务器启动并创建监听：

```
server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
server_socket.bind(("0.0.0.0", PORT))
server_socket.listen(5)
print(f"服务器启动，监听端口 {PORT}")
```

接下来，进入一个 while 循环，用于接受客户端的连接请求。当有一个客户端成功连接时，借助变量 data 接受客户端发过来的消息。

```
while True:
    client_socket, client_address =
server_socket.accept()
    print(f"客户端 {client_address} 已连接")

    while True:
        #最多接受 1024 字节数据
        data = client_socket.recv(2048)
        if not data:
            print(f"客户端 {client_address} 断开连接")
            break #没有接收到数据断开
    ……
    client_socket.close()
```

在接收到数据并放入 data 后，首先通过语句 total_length, command_id = struct.unpack("!II", data[:8])提取出注册或登录信息的总长度和消息类型编号。随后再借助语句 message_body = data[8:]提取出消息体 messgae_body。

根据上一步中提取出的消息类型编号，借助 if…else…语句判断客户端请求的操作是注册还是登录。无论是注册还是登录请求，都首先将消息体 message_dody 中的前 20 个字节在剔除补全字后的内容放入变量 username 中，将 messgae_dody 中的后 30 个字节在剔除补全字后的内容放入变量 password 中。随后分别调用 handle_registraion_request(username, password) 和 handle_login_request(username, password)函数来处理注册和登录请求。

在 handle_registraion_request(username, password)中，通过逐行比对的方式查找用户名是否已经注册。如果注册则返回注册失败和“用户名已存在”的描述。反之则将用户名和经过哈希变换后的密码写入存储用户注册信息的文件中。在 handle_login_request(username,password)中，首先将用户密码变换。随后也是逐行对比的方式查找用户是否有注册。如果用户信息存在但是密码不对则返回登录失败和“密码错误”描述。如果用户信息不存在则返回登录失败和“用户不存在”描述。

无论是失败还是成功，服务端程序都会创建一个响应体和响应头并发送给客户端。上述相关程序代码如下所示：

```
def handle_registration_request(username, password):
    with open(PASSWD_FILE, "a+") as f:
        f.seek(0)
        for line in f:
            if line.startswith(username + ":"):
                return False, "用户名已存在"
        hashed_password = hashlib.sha256(password.encode
()).hexdigest()
        f.write(f"{username}:{hashed_password}\n")
```

```
    return True, "注册成功"

def handle_login_request(username, password):
    hashed_password = hashlib.sha256(password.encode
()).hexdigest()
    with open(PASSWD_FILE, "r") as f:
        for line in f:
            #比对用户信息
            if line.startswith(username + ":"):
                stored_password = line.split(":")[1].stri
p()
                if stored_password == hashed_password:
                    return True, "登录成功"
                else:
                    return False, "密码错误"
    return False, "用户不存在"
……
while True:
        #最多接受 1024 字节数据
        data = client_socket.recv(2048)
        if not data:
            print(f"客户端 {client_address} 断开连接")
            break #没有接收到数据断开

        #解析消息头
        total_length, command_id = struct.unpack("!II",
data[:8]) #提取 message_length 和 command_id
        message_body = data[8:] #提取 message_body

        if command_id == COMMAND_ID_REG_REQUEST:
            username = message_body[:20].decode().strip
```

```
("\x00")  #解析出用户名
            password = message_body[20:50].decode().stri
p("\x00")  #解析出密码
            success, description = handle_registration_r
equest(username, password)
            status = 1 if success else 0 #失败 0 成功 1
            response_body = struct.pack("!B64s", status,
description.encode())
            response_header = struct.pack("!II", 8 + len
(response_body), COMMAND_ID_REG_RESPONSE)
            client_socket.send(response_header + respons
e_body) #返回状态
        elif command_id == COMMAND_ID_LOGIN_REQUEST:
            username = message_body[:20].decode().strip
("\x00") #解析出用户名
            password = message_body[20:50].decode().stri
p("\x00") #解析出密码
            success, description = handle_login_request
(username, password)
            status = 1 if success else 0 #失败 0 成功 1
           response_body = struct.pack("!B64s", status,
description.encode())
            response_header = struct.pack("!II", 8 + len
(response_body), COMMAND_ID_LOGIN_RESPONSE)
            client_socket.send(response_header + respons
e_body) #返回状态
```

四、客户端端程序设计思路

首先定义了消息类型编号和服务器地址及端口号等信息,用于与服务器端相关信息保持一致和确保链接:

```
COMMAND_ID_REG_REQUEST = 1
COMMAND_ID_REG_RESPONSE = 2
```

```
COMMAND_ID_LOGIN_REQUEST = 3
COMMAND_ID_LOGIN_RESPONSE = 4


SERVER_ADDRESS = "127.0.0.1"
SERVER_PORT = 12345
```

在客户端，创建好套接口后与服务器端建立链接。随后借助 while 循环等待用户选择注册或登录操作，并输入账号和密码。输入完账号和密码后，调用 send_registration_request(username, password)和 send_login_request(username, password)函数将相关信息按照规定的格式发送给服务器端。相关代码如下：

```
client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
   client_socket.connect((SERVER_ADDRESS,
SERVER_PORT))


   while True:
       choice = input("请输入操作（1：注册，2：登录）：")
       if choice == "1":
           username = input("请输入用户名：")
           password = input("请输入密码：")
           request_message =
send_registration_request(username, password)
       elif choice == "2":
           username = input("请输入用户名：")
           password = input("请输入密码：")
           request_message =
send_login_request(username, password)
       else:
           print("无效输入")
           continue
```

```
        client_socket.send(request_message)

        # 接收响应
        response_header = client_socket.recv(8)  # 接收消息头
        if not response_header:
            print("未收到响应")
            break
```

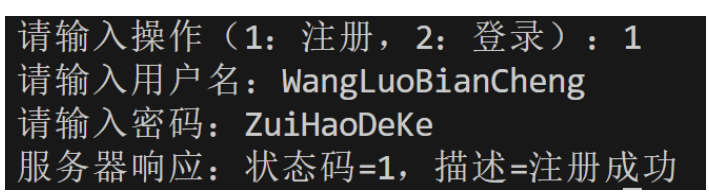
在 sen_registration_request(username, password) 和 send_login_request(username, password) 函数中，都是将 message_body 和 message_header 按照相应的格式要求进行组合好进行返回。相关代码如下：

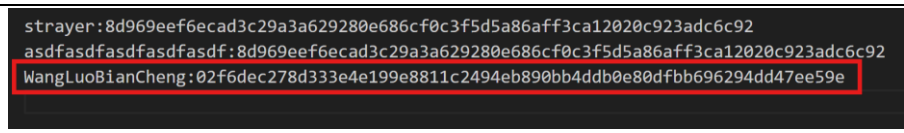
```
def send_registration_request(username, password):
    #注册
    message_body = struct.pack("!20s30s",
username.encode(), password.encode())
    message_header = struct.pack("!II", 8 +
len(message_body), COMMAND_ID_REG_REQUEST)
    return message_header + message_body

def send_login_request(username, password):
    #登录
    message_body = struct.pack("!20s30s",
username.encode(), password.encode())
    message_header = struct.pack("!II", 8 +
len(message_body), COMMAND_ID_LOGIN_REQUEST)
    return message_header + message_body
```

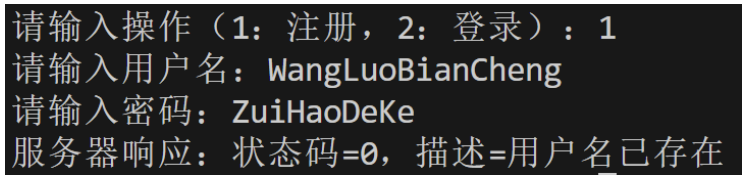
客户端还有一部分代码用于接收和解析服务器端的响应。这一部分既包含对于 TCP 的异常处理和服务器端响应的处理：

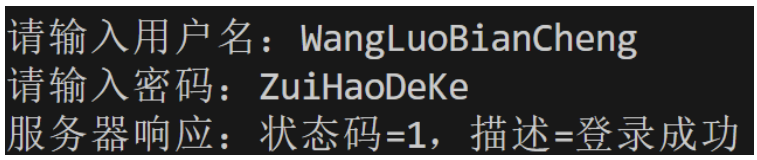
```
response_header = client_socket.recv(8)  # 接收消息头
        if not response_header:
```

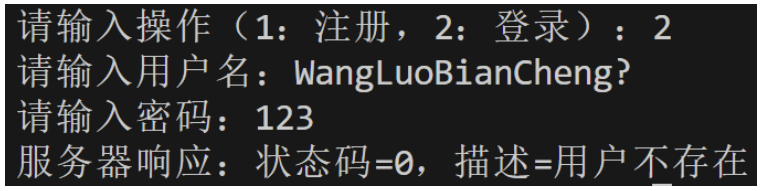
	print("未收到响应")
	break
	
	total_length, command_id = struct.unpack("!II", response_header)
	remaining_length = total_length - 8 # 消息体长度
	
	response_body = b""
	while remaining_length > 0:
	chunk = client_socket.recv(remaining_length)
	if not chunk:
	print("连接中断")
	break
	response_body += chunk
	remaining_length -= len(chunk)
	
	if remaining_length > 0:
	print("未收到完整的响应")
	break
	
	status, description = struct.unpack("!B64s", response_body)
	description = description.decode().strip("\x00")
	print(f"服务器响应：状态码={status}，描述={description}")

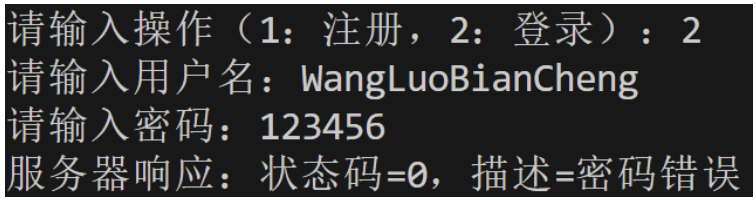
数据记录和计算	在客户端创建一个新用户。观察到成功注册： 

<table>
<tr><td></td><td>


重复注册同样的用户。观察到返回“用户名已存在”：


登录已注册的账户并输入正确的密码。观察到成功登录：


登录未注册的账户。观察到返回“用户名不存在”：


登录已注册账户但输入错误的密码。观察到返回“密码错误”：


</td></tr>
<tr><td>结　论
（结 果）</td><td>通过本次实验，服务器端和客户端均按照预期的结果进行了输出和运行。服务器能够成功启动并监听指定端口，接收客户端的连接请求，并正确处理客户端发送的注册请求和登录请求。客户端能够成功连接到服务器，发送注册和登录请求，并接收服务器返回的响应消息。同时，客户端可以通过输入退出指令正常断开连接，服务器能够及时检测到连接关闭并清理相关资源。整个通信过程稳定可靠，消息格式和协议设计有效地保障了客户端与服务器之间的数据交互，实验达到了预期的目标。</td></tr>
<tr><td>小　结</td><td>在服务器端，我实现了用户注册和登录的逻辑处理，包括用户信息的存储、密码的哈希处理以及对重复用户名的检测。通过文件操作和哈希算法，我确保了用户数据的安全性和唯一性。同时，服务器能够正确解析客户端发送的消息，并根据消息类型返回相应的响应。在客户端，我实现了用户输入界面，允许用户选择注册或登录操作，并发送相应的请求到服务器。客户端能够正确解析服务器返回的响应消息，并向用户反馈操作结果。通过本次实验，我掌握了如何设计和实现一个简单的网络通信协议，并理解了客户端与</td></tr>
</table>

	服务器之间的交互流程。
指导老师评　　议	成绩评定：　　指导教师签名：