

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 7 周

课程名称	网络编程	实验课时	5-7 节
实验项目	支持心跳检测的客户端	实验时间	2025. 04. 10
实验目的	学习如何使用多线程实现同时处理数据收发与心跳检测。		
实验环境	Visual Studio Code		
实 验 内 容 (算法、程 序、步骤和 方法)	<p>借助 Python 语言，分别实现：</p> <p>一、TCP 服务端程序</p> <p>1) 服务器等待接收来自客户端发送的字符串（不限次数），并输出到屏幕。</p> <p>2) 服务器内部记录每个客户端最新的心跳时间。</p> <p>3) 如果服务器在心跳间隔阈值（10 秒）内没有收到某个客户端的心跳，则判定连接异常。</p> <p>二、TCP 客户端程序</p> <p>1) 客户端等待键盘输入，并将输入信息以字符串形式发送给服务器。</p> <p>2) 每 5 秒发送一次心跳包给服务器（无须服务器在应用层应答这个心跳）。</p> <p>三、TCP 服务器端程序设计思路</p> <p>首先定义了 TCPServer 类,初始化时设置服务器的主机地址、端口和心跳检测阈值。服务器使用 socket 模块创建一个 TCP 套接字，并设置 SO_REUSEADDR 选项，以便在服务器重启时可以快速重新绑定端口。初始化时还创建了一个字典 client_info，用于存储客户端的连接信息，包括最后心跳时间、客户端套接字和连接时间：</p> <pre>def __init__(self, host='0.0.0.0', port=12345, heartbeat_threshold=10): self.host = host self.port = port self.heartbeat_threshold = heartbeat_threshold self.client_info = {} # 存储客户端地址和最后心跳时间</pre>		

```
self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
self.server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

```
self.running = False
```

在 `start` 方法中，首先将服务器套接字绑定到指定的主机和端口上，并开始监听客户端连接。打印服务器启动信息和心跳检测阈值。然后将 `running` 标志设置为 `True`，表示服务器开始运行。

为了实现心跳检测功能，启动了一个心跳检测线程 `heartbeat_thread`，该线程会定期检查客户端的心跳状态。心跳检测线程设置为守护线程，这样当主线程结束时，心跳检测线程也会随之结束。接下来，进入一个 `while` 循环，不断接受客户端的连接请求。当有客户端连接时，打印连接信息，并记录客户端的初始连接时间、套接字对象等信息到 `client_info` 字典中。然后为每个客户端创建一个接收线程 `client_thread`，用于处理该客户端的通信。接收线程同样设置为守护线程。如果在接收客户端连接时出现异常，会打印错误信息并继续循环：

```
def start(self):
```

```
    self.server_socket.bind((self.host, self.port))
```

```
    self.server_socket.listen(5)
```

```
    print(f"[{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}] 服务器启动，监听 {self.host}:{self.port}")
```

```
    print(f"[{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}] 心跳检测阈值: {self.heartbeat_threshold}秒")
```

```
    self.running = True
```

```
    # 启动心跳检测线程
```

```
    heartbeat_thread = threading.Thread(target=self.check_heartbeats)
```

```
    heartbeat_thread.daemon = True
```

```
    heartbeat_thread.start()
```

```
    try:
```

```
        while self.running:
```

```
            try:
```

```
                client_socket, client_address = self.server_socket.accept()
```

```
                print(f"[{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}] 新的客户端连接: {client_address}")
```

```
                # 记录客户端初始连接时间
```

```

        self.client_info[client_address] = {
            'last_heartbeat': time.time(),
            'socket': client_socket,
            'connect_time': datetime.now().strftime('%Y-%m
-%d %H:%M:%S')
        }

        # 为每个客户端创建接收线程
        client_thread = threading.Thread(
            target=self.handle_client,
            args=(client_socket, client_address)
        )
        client_thread.daemon = True
        client_thread.start()
    except Exception as e:
        print(f'[{datetime.now().strftime('%Y-%m-%d %H:%
M:%S')}] 接受客户端连接时出错: {str(e)}')
        continue
    except KeyboardInterrupt:
        self.stop()

    handle_client 方法是客户端接收线程的执行函数。它会不断接
    收客户端发送的数据。如果客户端发送的数据为空，说明客户端已
    经断开连接，此时退出循环。

    如果收到的数据是心跳包 ("HEARTBEAT")，则更新客户端
    的最后心跳时间，并打印收到心跳包的信息。如果收到的是其他数
    据，则打印客户端发送的消息。

    如果在接收数据过程中出现 ConnectionResetError 异常，说明
    客户端异常断开连接，打印相关信息并退出循环。如果出现其他异
    常，也会打印错误信息并退出循环。

    无论何种原因退出循环，都会从 client_info 字典中删除该客户
    端的信息，并关闭客户端的套接字，最后打印客户端连接关闭的信
    息:
def handle_client(self, client_socket, client_address):
    try:
        while self.running:
            try:
                data = client_socket.recv(1024).decode('utf-8')
                if not data:
                    break

```

```

# 如果是心跳包，更新最后心跳时间
if data == "HEARTBEAT":
    self.client_info[client_address]['last_heartbeat'] =
time.time()

    print(f"[{datetime.now().strftime('%Y-%m-%d %
H:%M:%S')}] 收到来自 {client_address} 的心跳包")
else:
    print(f"[{datetime.now().strftime('%Y-%m-%d %
H:%M:%S')}] 来自 {client_address} 的消息: {data}")
except ConnectionResetError:
    print(f"[{datetime.now().strftime('%Y-%m-%d %H:%
M:%S')}] 客户端 {client_address} 异常断开")
    break
except Exception as e:
    print(f"[{datetime.now().strftime('%Y-%m-%d %H:%
M:%S')}] 处理客户端 {client_address} 时出错: {str(e)}")
    break

finally:
    if client_address in self.client_info:
        del self.client_info[client_address]
    try:
        client_socket.close()
    except:
        pass
    print(f"[{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}]
客户端 {client_address} 连接关闭")

    check_heartbeats 方法是心跳检测线程的执行函数。它会不断
    循环检查每个客户端的最后心跳时间。首先获取当前时间，然后遍
    历 client_info 字典中的所有客户端信息。

    对于每个客户端，计算从上次心跳到现在的时间间隔。如果时
    间间隔小于等于心跳检测阈值，则打印客户端状态为“正常”；如
    果时间间隔大于心跳检测阈值，则将该客户端添加到 disconnected
    _clients 列表中，并打印客户端状态为“超时”。

    在循环结束后，遍历 disconnected_clients 列表，关闭这些超时
    客户端的套接字，并从 client_info 字典中删除它们的信息，同时打
    印已断开连接的信息。

    最后，线程会休眠 3 秒，然后继续下一轮心跳检查。如果在心
    跳检测过程中出现异常，会打印错误信息并继续循环：
def check_heartbeats(self):
    while self.running:

```

```

try:
    current_time = time.time()
    disconnected_clients = []

    # 显示当前连接的客户端状态
    print("\n" + "="*50)
    print(f"[{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}] 当前客户端连接状态:")
    for client_addr, info in self.client_info.items():
        try:
            time_since_last_heartbeat = current_time - info['last_heartbeat']
            status = "正常" if time_since_last_heartbeat <= self.heartbeat_threshold else "超时"
            print(f"客户端 {client_addr} | 连接时间: {info['connect_time']} | "
                  f"最后心跳: {time_since_last_heartbeat:.1f}秒前 | 状态: {status}")

            if time_since_last_heartbeat > self.heartbeat_threshold:
                disconnected_clients.append(client_addr)

        except Exception as e:
            print(f"[{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}] 检查客户端 {client_addr} 心跳时出错: {str(e)}")
            disconnected_clients.append(client_addr)

    # 关闭超时的客户端连接
    for client_addr in disconnected_clients:
        if client_addr in self.client_info:
            try:
                self.client_info[client_addr]['socket'].close()

            except:
                pass

            try:
                del self.client_info[client_addr]

```

```

except:
    pass
    print(f"[{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}] 已断开与 {client_addr} 的连接(心跳超时)")

time.sleep(3) # 每 3 秒检查一次心跳并显示状态

except Exception as e:
    print(f"[{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}] 心跳检测线程出错: {str(e)}")
    continue

```

创建一个 `TCPClient` 类，初始化时指定服务器的主机地址 `host`（默认为 '127.0.0.1'）和端口号 `port`（默认为 12345）。初始化时创建了一个客户端套接字 `client_socket`，用于与服务器通信。同时，定义了一个 `running` 标志用于控制客户端的运行状态。此外，还定义了心跳相关的变量：心跳计数器 `heartbeat_count`、初始心跳间隔 `initial_interval`（前 3 次心跳间隔为 5 秒）、最终心跳间隔 `final_interval`（3 次后心跳间隔为 100 秒）以及最大初始心跳次数 `max_initial_heartbeats`（初始阶段心跳次数为 3 次）：

```

def __init__(self, host='127.0.0.1', port=12345):
    self.host = host
    self.port = port
    self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.running = False
    self.heartbeat_count = 0 # 心跳计数器
    self.initial_interval = 5 # 前 3 次心跳间隔(秒)
    self.final_interval = 100 # 3 次后心跳间隔(秒)
    self.max_initial_heartbeats = 3 # 初始阶段心跳次数

```

在 `connect` 方法中，首先尝试使用客户端套接字连接到指定的服务器主机和端口。如果连接成功，打印连接成功的消息，并将 `running` 标志设置为 `True`，表示客户端开始运行。

接下来，启动心跳线程 `heartbeat_thread`，用于定期发送心跳包到服务器。心跳线程设置为守护线程，这样当主线程结束时，心跳线程也会随之结束。同时，启动接收线程 `receive_thread`，用于接收服务器发送的消息，接收线程同样设置为守护线程。

然后，主线程进入一个循环，等待用户输入要发送的消息。用户输入消息后，将其发送到服务器。如果用户输入 'exit'，则调用 `disconnect` 方法断开与服务器的连接并退出循环。如果在发送消息时

出现异常，也会调用 `disconnect` 方法断开连接并退出循环。如果连接服务器时出现 `ConnectionRefusedError` 异常，说明无法连接到服务器，打印错误信息：

```
def connect(self):
    try:
        self.client_socket.connect((self.host, self.port))
        print(f"已连接到服务器 {self.host}:{self.port}")
        self.running = True

        # 启动心跳线程
        heartbeat_thread = threading.Thread(target=self.send_
heartbeats)
        heartbeat_thread.daemon = True
        heartbeat_thread.start()

        # 启动接收线程
        receive_thread = threading.Thread(target=self.receive
_messages)
        receive_thread.daemon = True
        receive_thread.start()

        # 主线程处理用户输入
        while self.running:
            message = input("请输入要发送的消息(输入'exit'退出): ")

            if message.lower() == 'exit':
                self.disconnect()
                break

            try:
                self.client_socket.sendall(message.encode('utf
-8'))

            except:
                print("发送消息失败")
                self.disconnect()
                break

        except ConnectionRefusedError:
            print(f"无法连接到服务器 {self.host}:{self.port}")
```

`send_heartbeats` 方法是心跳线程的执行函数。它会不断循环发送心跳包到服务器。每次发送心跳包时，心跳计数器 `heartbeat_count` 加 1，并打印发送心跳包的次数。

根据心跳计数器的值，决定下一次心跳的间隔时间。如果心跳次数小于最大初始心跳次数，则使用初始心跳间隔；否则，使用最终心跳间隔。然后线程休眠指定的时间间隔。如果在发送心跳包时出现异常，打印错误信息并调用 `disconnect` 方法断开连接，退出循环：

```
def send_heartbeats(self):
    while self.running:
        try:
            # 发送心跳包
            self.client_socket.sendall("HEARTBEAT".encode
('utf-8'))

            self.heartbeat_count += 1
            print(f"发送第 {self.heartbeat_count} 次心跳包")

            # 根据心跳次数决定下次心跳间隔
            if self.heartbeat_count < self.max_initial_heartbeats:
                interval = self.initial_interval
            else:
                interval = self.final_interval


            time.sleep(interval)
        except:
            print("发送心跳失败")
            self.disconnect()
            break
```

`receive_messages` 方法是接收线程的执行函数。它会不断循环接收服务器发送的消息。如果接收到的消息为空，说明服务器已经断开连接，打印提示信息并调用 `disconnect` 方法断开连接，退出循环。

如果接收到的消息不是心跳包（"HEARTBEAT"），则打印来自服务器的消息。如果在接收消息时出现异常，打印错误信息并调用 `disconnect` 方法断开连接，退出循环。

`disconnect` 方法用于断开与服务器的连接。如果客户端正在运行，将 `running` 标志设置为 `False`，表示客户端停止运行。然后尝试关闭客户端套接字，最后打印已断开连接的消息：

```
def receive_messages(self):
    while self.running:
        try:
```


	<pre>data = self.client_socket.recv(1024).decode('utf-8') if not data: print("服务器断开连接") self.disconnect() break if data != "HEARTBEAT": print(f"来自服务器的消息: {data}") except: print("接收消息失败") self.disconnect() break def disconnect(self): if self.running: self.running = False try: self.client_socket.close() except: pass print("已断开与服务器的连接")</pre>
数据记录 和计算	<p>服务器端终端输出结果:</p>  <p>The screenshot displays a terminal window with the following content:</p> <ul style="list-style-type: none">Server startup message: [2025-04-10 15:32:32] 服务器启动, 监听 0.0.0.0:12345 服务器启动Heartbeat threshold: [2025-04-10 15:32:32] 心跳检测阈值: 10秒Initial connection status checks: [2025-04-10 15:32:32], [2025-04-10 15:32:35], and [2025-04-10 15:32:38] 当前客户端连接状态: (all empty)Client connection event: [2025-04-10 15:32:41] 当前客户端连接状态: 心跳包1New client connection: [2025-04-10 15:32:44] 新的客户端连接: ('127.0.0.1', 49948)Received heartbeat: [2025-04-10 15:32:44] 收到来自 ('127.0.0.1', 49948) 的心跳包Connection status update: [2025-04-10 15:32:44] 当前客户端连接状态: 客户端 ('127.0.0.1', 49948) 连接时间: 2025-04-10 15:32:44 最后心跳: 0.7秒前 状态: 正常Received message: [2025-04-10 15:32:45] 来自 ('127.0.0.1', 49948) 的消息: 1Client sends string: 127.0.0.1:49948客户端向服务器发送字符串Connection status update: [2025-04-10 15:32:47] 当前客户端连接状态: 客户端 ('127.0.0.1', 49948) 连接时间: 2025-04-10 15:32:44 最后心跳: 3.7秒前 状态: 正常Received heartbeat: [2025-04-10 15:32:49] 收到来自 ('127.0.0.1', 49948) 的心跳包Final status: [2025-04-10 15:32:50] 当前客户端连接状态: 心跳包2 客户端 ('127.0.0.1', 49948) 连接时间: 2025-04-10 15:32:44 最后心跳: 1.7秒前 状态: 正常

	<div data-bbox="403 211 1178 713"><pre>[2025-04-10 15:32:50] 当前客户端连接状态: (橙上留) 客户端 ('127.0.0.1', 49948) 连接时间: 2025-04-10 15:32:44 最后心跳: 1.7秒前 状态: 正常 ===== [2025-04-10 15:32:53] 当前客户端连接状态: 客户端 ('127.0.0.1', 49948) 连接时间: 2025-04-10 15:32:44 最后心跳: 4.7秒前 状态: 正常 [2025-04-10 15:32:54] 收到来自 ('127.0.0.1', 49948) 的心跳包 ===== [2025-04-10 15:32:56] 当前客户端连接状态: 心跳包3 客户端 ('127.0.0.1', 49948) 连接时间: 2025-04-10 15:32:44 最后心跳: 2.7秒前 状态: 正常 ===== [2025-04-10 15:32:59] 当前客户端连接状态: 客户端 ('127.0.0.1', 49948) 连接时间: 2025-04-10 15:32:44 最后心跳: 5.7秒前 状态: 正常 ===== [2025-04-10 15:33:02] 当前客户端连接状态: 客户端 ('127.0.0.1', 49948) 连接时间: 2025-04-10 15:32:44 最后心跳: 8.7秒前 状态: 正常 在最后一次心跳包发送后, 下一次的心跳包发送过来的时间在100秒后, 心跳包到服务器端所接收到的最后心跳时间不断累加, 超过10秒阈值后状 态更新为'超时' [2025-04-10 15:33:05] 当前客户端连接状态: 客户端 ('127.0.0.1', 49948) 连接时间: 2025-04-10 15:32:44 最后心跳: 11.7秒前 状态: 超时 [2025-04-10 15:33:05] 处理客户端 ('127.0.0.1', 49948) 时出错: [WinError 10053] 你的主机中的软件中止了一个已建立的连接。 [2025-04-10 15:33:05] 已断开与 ('127.0.0.1', 49948) 的连接(心跳超时) 服务器端断开与127.0.0.1:49948客户端的连接 [2025-04-10 15:33:05] 客户端 ('127.0.0.1', 49948) 连接关闭 ===== [2025-04-10 15:33:08] 当前客户端连接状态: 服务器断开与127.0.0.1:49948客户端的连接后继续保持监听状态</pre></div> <div data-bbox="403 736 1178 1183"><pre>[2025-04-10 15:52:05] 当前客户端连接状态: ===== [2025-04-10 15:52:08] 当前客户端连接状态: ===== [2025-04-10 15:52:11] 当前客户端连接状态: 服务器在监听状态下很久后, 重新与 [2025-04-10 15:52:14] 新的客户端连接: ('127.0.0.1', 50533) 127.0.0.1:50533客户端进行了链接 [2025-04-10 15:52:14] 收到来自 ('127.0.0.1', 50533) 的心跳包 ===== [2025-04-10 15:52:14] 当前客户端连接状态: 客户端 ('127.0.0.1', 50533) 连接时间: 2025-04-10 15:52:14 最后心跳: 0.7秒前 状态: 正常 [2025-04-10 15:52:16] 来自 ('127.0.0.1', 50533) 的消息: 2 ===== [2025-04-10 15:52:17] 当前客户端连接状态: 服务器依旧保持了接受来自客户端发送的字符串信息的能力 客户端 ('127.0.0.1', 50533) 连接时间: 2025-04-10 15:52:14 最后心跳: 3.7秒前 状态: 正常 [2025-04-10 15:52:19] 收到来自 ('127.0.0.1', 50533) 的心跳包 ===== [2025-04-10 15:52:20] 当前客户端连接状态: 客户端 ('127.0.0.1', 50533) 连接时间: 2025-04-10 15:52:14 最后心跳: 1.7秒前 状态: 正常</pre></div>
<div data-bbox="186 1671 284 1746">结 论 (结 果)</div>	<div data-bbox="375 1234 637 1266">客户端终端输出结果:</div> <div data-bbox="375 1279 1181 1552"><pre>ps/python3.10.exe c:/Users/s/Desktop/计算机学院/资料/网络编程/实验/实验2/hbclient.py 已连接到服务器 127.0.0.1:22345 发送第 1 次心跳包 请输入要发送的消息(输入'exit'退出): 1 客户端1与服务器建立连接后发送的字符串信息 请输入要发送的消息(输入'exit'退出): 发送第 2 次心跳包 发送第 3 次心跳包 接收消息失败 已断开与服务器的连接 心跳包超时后服务器断开与客户端的连接 c:/Users/s/AppData/Local/Microsoft/WindowsApps/python3.10.exe c:/Users/s/Desktop/计算机学院/资料/网络编程/实验/实验2/hbclient.py 发送消息失败 PS c:/Users/s/Desktop/计算机学院/资料/网络编程/实验/实验2> & c:/Users/s/AppData/Local/Microsoft/WindowsAp ps/python3.10.exe c:/Users/s/Desktop/计算机学院/资料/网络编程/实验/实验2/hbclient.py 已连接到服务器 127.0.0.1:22345 发送第 1 次心跳包 请输入要发送的消息(输入'exit'退出): 2 客户端2与服务器建立连接后发送的字符串信息</pre></div> <div data-bbox="330 1568 1250 1848"><p>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心跳包和消息。客户端能够成功连接到服务器，定期发送心跳包，并接收服务器发送的消息。同时，客户端可以通过输入 exit 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，心跳机制有效地保障了客户端与服务器之间的连接状态，实验达到了预期的目标。</p></div>

