

《计算机网络》实验报告

信息学院 计算机系统结构 专业 2024 级

实验时间 2024 年 10 月 20 日

姓名 李昊 学号 12024115051

实验名称 实验一 Socket 编程基础

实验成绩

一、实验目的

程序员可以使用不同的工具来调用操作系统 API 创建网络的服务器和客户端，这些工具的核心模块是 Socket。使用 Socket，你可以快速地编写 TCP 和 UDP 服务器和客户端，并且这些套接字应用并不困难。为了学习到如何实现两台计算机之间的信息通信基础，我们从创建一个简单的客户端和服务端开始，实现一个单次信息收发的小程序。

二、实验仪器设备及软件

实验设备：PC

软件：pycharm2024.2.3(Professional Edition),macos,python3.7.16

三、实验方案

1.搭建实验环境：

使用 anaconda 创建对应的 python 环境，并在 pycharm 里进行相应的配置。

2.撰写代码：

分别编写相应代码实现基于 TCP 连接的 server 和 client，基于 UDP 的 server 和 client，并生成相应对象测试结果的准确性；

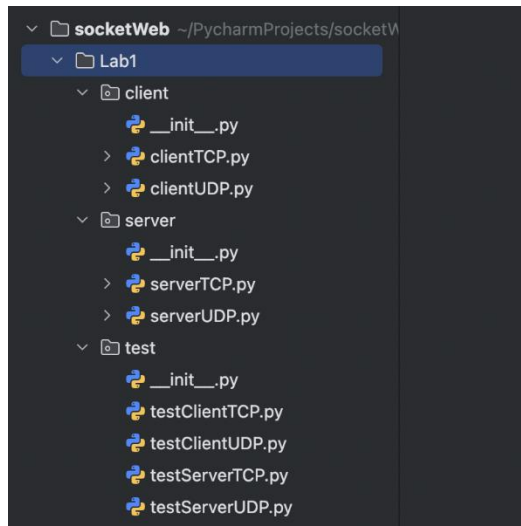
3.结果分析：

分析结果，观察各自的服务器与对应的客户端是否能正常收发数据；观察基于 tcp 的客户端和服务端当一方断开连接后另一方能否及时发现并做出相应的反

应；观察使用多线程进行编程，结果是否因为线程的特性导致阻塞使得结果出现错误的情况；

四、实验步骤

文件结构如下：



下列代码分析见对应函数下的注释：

serverClient:

```
def __init__(self, host="127.0.0.1", port=6666):  
    """  
    1. 设定客户端的相应socket信息，设置超时参数，设置标记位判断连接情况  
    2. 客户端自己的端口：通常是由操作系统自动分配的，不需要手动指定。  
    服务器通常监听一个固定的端口，而客户端则使用一个随机的临时端口进行通信。  
    :param host: 服务器ip  
    :param port: 服务器port  
    """  
    self.host = host  
    self.port = port  
    self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
    self.timeout = 30  
    self.client_socket.settimeout(self.timeout)  
    # 设置超时时间  
    # 注意使用的方法，是抛出异常  
    self.is_running = True  
    # 该标记位记录是否与服务器保持连接，listen_for_server中如果连接中断，将被设置为false
```

```
def handle_response_from_server(self, reponse): 1 usage  
    """  
    用于处理从服务器返回的信息  
    :param reponse:  
    :return:  
    """  
    print(reponse.decode("utf-8"))
```

```

def send_message(self): 1 usage
    """
    用于执行发送信息的函数，用input接收输入(会阻塞)
    :return:
    """
    while self.is_running:
        try:
            ready, _, _ = select.select([sys.stdin], [], [], timeout=1)
            if ready:
                message = input()

                # 因为使用input会阻塞，会妨碍client_socket.settimeout()的作用
                # 考虑用多线程接收数据

                # 注意使用byte object进行数据传输
                # 客户端要传输的信息怎么获取？

                self.client_socket.send(message.encode("utf-8"))

```

```

def listen_for_server(self): 1 usage
    """
    用新线程执行该函数，监听服务器状态，如果服务器断开/超时则切断与服务器的连接
    :return:
    """
    while self.is_running:
        try:
            ready, _, _ = select.select([self.client_socket], [], [], self.timeout)
            # 使用 select 来非阻塞监听
            if ready:
                response = self.client_socket.recv(1024)
                # 为什么服务器端关闭后仍然能收到ack---记得清理缓冲区
                if response:
                    print("上一条信息状态:", end=' ')
                    self.handle_response_from_server(response)
                    # while self.client_socket.recv(1024):
                    #     pass
                    # 清理client的socket的缓冲区
                else:
                    print("server is shutting down\nconnection has been closed by server")
                    self.is_running = False
                    self.client_socket.close()
                    break

```

```

def start(self): 1 usage
    """
    启动连接，并用一个新线程监听服务器状态，主线程执行发送信息
    :return:
    """
    self.connect()
    # 多线程，一个用于监听与服务器的连接，一个用于进行收数据
    listener_thread = threading.Thread(target=self.listen_for_server)
    listener_thread.start()
    # 主线程用于和服务器发送信息
    print("输入要传输的数据")
    self.send_message()
    # 等待监听线程结束
    listener_thread.join()

```

serverTCP:

```
def __init__(self, host='127.0.0.1', port=6666):
    """
    指定服务器要监听的端口和host，维护一个连接到本服务器上的client列表方便后续操作
    :param host:
    :param port:
    """
    self.port = port
    self.host = host
    self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.client = []
    # 维护一个与本服务器连接的客户端列表
```

```
def handle_client(self, client_socket, client_address): 1 usage
    """
    从客户端接收信息
    :param client_socket:
    :param client_address:
    :return:
    """
    while True:
        try:
            message = client_socket.recv(1024)
            if message == b"":
                break
            # 断开连接的时候message为""
            print(f"从IP: {client_address[0]} Port: {client_address[1]} 处接受到的信息: ", end='')
            self.handle_request(message)
            client_socket.send("ACK".encode("utf-8"))
        except ConnectionResetError:
            print("oops!! something goes wrong!")
            break
```

```
def broadcastShutdown(self): 1 usage
    """
    服务器关闭，通知所有客户端并断开连接
    :return:
    """
    for client in self.client:
        # client.send(b'server is shutting down')
        client.close()
```

serverUDP:

```
def __init__(self, host="127.0.0.1", port=6666):
    """
    UDP 本身不需要建立连接，它可以处理来自不同客户端的数据包。
    实现多客户端支持的关键是在服务器端能够同时接收和回复来自多个客户端的请求，
    而不需要像 TCP 那样维护每个客户端的连接。

    因为无连接的方式，当服务器关闭时不需要切断和客户端的连接？发送一条数据通知所有连接过的客户端？
    :param host:
    :param port:
    """
    self.host = host
    self.port = port
    self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

def listen(self): 1 usage
    while True:
        data, client_address = self.server_socket.recvfrom(1024)
        print(f"-----来自IP: {client_address[0]} Port: {client_address[1]}-----")
        response = self.handle_request(data.decode())
        try:
            self.server_socket.sendto(response.encode(), client_address)
        except Exception as e:
            print(f"oops! something goes wrong when sending message to {client_address} :{e}")

def handle_request(self, data): 1 usage
    print(f"数据: {data}")
    return "check"

def shutdown(self): 1 usage
    if self.server_socket:
        self.server_socket.close()
    print("server is shutting down !")
```

clientUDP:

```
class clientUDP: 2 usages
    def __init__(self, host="127.0.0.1", port=6666):
        self.port = port
        self.host = host
        self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.timeout = 30
        self.client_socket.settimeout(self.timeout)
        self.is_running = True
```

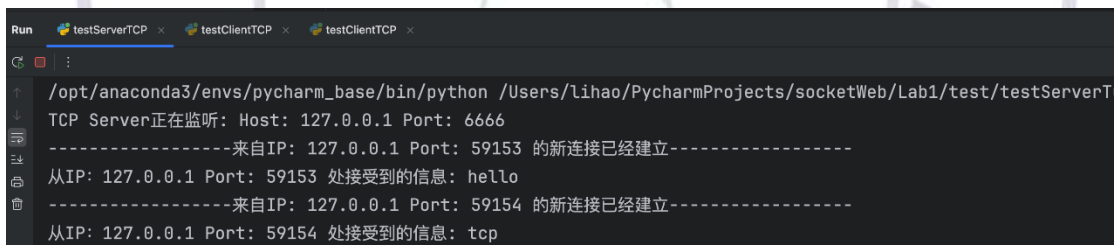
```
def handle_response_from_server(self, data, server_address): 1 usage
    print(f"收到{server_address} 处返回数据: {data.decode()}")

def close(self): 6 usages (4 dynamic)
    if self.client_socket:
        self.is_running = False
        self.client_socket.close()
```

```
def start(self): 1 usage
    while self.is_running:
        print("输入需要传输的信息: ")
        try:
            ready, _, _ = select.select([sys.stdin], [], [], self.timeout)
            if ready:
                message = input()
                self.sendMessage(message)
            else:
                print("client time out !")
                self.is_running = False
                break
        except Exception as e:
            print(f"oops! something goes wrong when sending message: {e}")
```

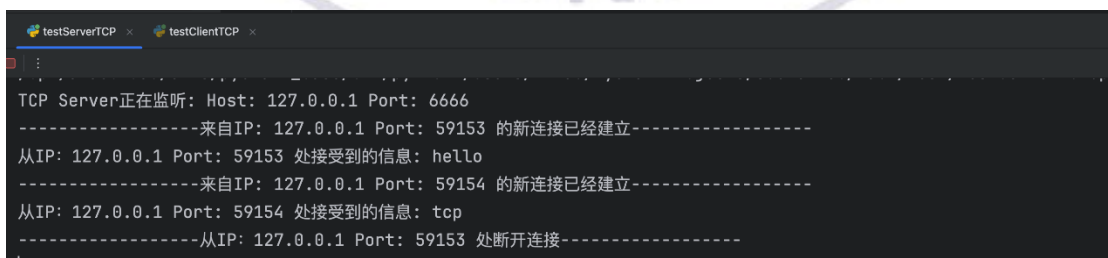
五、实验结果及分析

1. 基于 TCP 实现:



```
Run testServerTCP x testClientTCP x testClientTCP x
/opt/anaconda3/envs/pycharm_base/bin/python /Users/lihao/PycharmProjects/socketWeb/Lab1/test/testServerT
TCP Server正在监听: Host: 127.0.0.1 Port: 6666
-----来自IP: 127.0.0.1 Port: 59153 的新连接已经建立-----
从IP: 127.0.0.1 Port: 59153 处接受到的信息: hello
-----来自IP: 127.0.0.1 Port: 59154 的新连接已经建立-----
从IP: 127.0.0.1 Port: 59154 处接受到的信息: tcp
```

上图为一个 tcp 服务器连接多个客户端的情况，控制台打印出连接的客户端的相关信息 and 从对应客户端收到的数据信息；



```
testServerTCP x testClientTCP x
TCP Server正在监听: Host: 127.0.0.1 Port: 6666
-----来自IP: 127.0.0.1 Port: 59153 的新连接已经建立-----
从IP: 127.0.0.1 Port: 59153 处接受到的信息: hello
-----来自IP: 127.0.0.1 Port: 59154 的新连接已经建立-----
从IP: 127.0.0.1 Port: 59154 处接受到的信息: tcp
-----从IP: 127.0.0.1 Port: 59153 处断开连接-----
```

上图为当一个客户端因为超时或其他原因断开连接的情况，此时 server 端检测到客户端断开连接，它将调用对应的 client_socket 的 close 方法关闭连接，并将这个客户端从维护的连接客户端列表中移除；


```
testClientTCP x
tcp
上一条信息状态: ACK
server is shutting down
connection has been closed by server
```

上图为当服务器因为某些原因关闭时的情况，服务器退出前会调用 `broadcastshutdown` 方法，向连接列表中的每一个客户端发送一个断开连接的信号，并依次切断与其的连接。同时客户端检测到服务器关闭前发送的相应信息，于是客户端自己设置相应的标记位，并调用对应的方法关闭自己的 `socket` 后打印相应的信息并结束进程。

```
Run testServerTCP x testClientTCP x
/opt/anaconda3/envs/pycharm_base/bin/python /Users/lihao/PycharmProjects/socketWeb/Lab1/test/testClientTCP.p
与IP: 127.0.0.1 PORT: 6666 成功建立连接!
输入要传输的数据
oops! something goes wrong!!(probably timeout)
```

上图为当服务器检测到客户端一段时间内没有发送任何信息的情况，此时服务器判断客户端因为某些情况无法继续通信，所以主动且切断与 `client` 的连接，并将其从维护的连接列表中移除；

2.基于 UDP 实现:

```
Run testServerUDP x testClientUDP x
/opt/anaconda3/envs/pycharm_base/bin/python /Users/lihao/PycharmProjects/socketWeb/Lab1/te
UDP Server正在监听: Host: 127.0.0.1 Port: 6666
-----来自IP: 127.0.0.1 Port: 61978-----
数据: hello
-----来自IP: 127.0.0.1 Port: 61978-----
数据: testClientUDP
```

上图为 `server` 端接收到发自多个客户端的情况，因为 `udp` 为无连接协议，所以服务器仅打印对应的信息；

```
Run testServerUDP x testClientUDP x
hello
收到('127.0.0.1', 6666) 处返回数据: check
输入需要传输的信息:
testClientUDP
收到('127.0.0.1', 6666) 处返回数据: check
输入需要传输的信息:
```

上图为 client 端发送信息的情况，同时收到一条来自 server 的确认信息。

六、实验总结及体会

1. 遇到的问题：

- a) clientTCP 发送的数据由 cmd 输入，使用 input 方法会导致阻塞，这会导致监听服务器连接的功能收到妨碍；
- b) 监听服务器连接的功能里，如何判断到服务器已经断开，且当检测到服务器断开后，如何将这一信息告诉发送信息的函数，并相应的结束客户端的连接；
- c) 服务器关闭时，如何通知所有客户端关机信息；

2. 解决的方法：

- a) 对于第一个问题：单独实现一个发送信息的函数，并在客户端的 start 方法内新建一个线程，单独用于监听服务器的连接状态，并在主线程中实现数据信息的发送和接收；
- b) 对于第二个问题：使用 select 函数监听 socket 信息，同时设置一个超时时间，当到达超时时间而没有任何信息到来，表明可能服务器已经切断连接，此时 client 设置自己的连接状态位并主动断开与服务器的连接；实现发送数据的线程也会相应检测这个连接状态位，当发现断开连接时，主动退出。
- c) 对于第三个问题：维护一个列表，记录所有连接到该服务器的客户端，当出现客户端断开时从列表中移除相应的客户端，当服务器关机时，遍历这个列表并发送关机信息；

3. 心得：

学习了如何基于 python 实现多线程的 socket 通信，并体会到了 TCP 是有连接协议，通信时要维护一个连接，UDP 为无连接协议，通信时只需处理接收到的数据即可；

七、教师评语