Socket 网络编程实验报告

无04 2019012137 张鸿琳

实验目的

- 理解 Socket 套接字在计算机网络中的位置与作用；
- 掌握 Socket 接口的编程方式，实现两台电脑之间的基于客户端-服务器模式的聊天应用。

实验内容

- 学习并理解 Socket 的编程原理与基本知识；
- 掌握进程中调用 Socket 相关接口的基本方法；
- 阅读并补全示例代码，实现基于 Socket 的命令行端对端聊天客户端程序；
- 将实现的客户端程序连接到课程准备的服务端程序，验证客户端是否正常；
- 阅读并补全示例代码，实现基于 Socket 的命令行端对端聊天服务端程序，并利用本机客户端进行测试；
- 同学们自由组队分别运行聊天应用的客户端与服务端程序，并测试是否能够正常工作；
- （选做）实现多对多 Socket 聊天服务器，将客户端发送的消息广播至其他连接的客户端。

代码展示

首先需要编写客户端程序，客户端 python 代码中实现了一个类 Client ，通过实例化一个 Client 对象生成一个客户端，此后运行 start_connection 来建立与服务端的连接。初始化（__init__）和连接（start_connection）实现比较简单，补上相应代码即可，在 start_connection 中建立连接后，开启了两个子线程，一个负责发送客户端发出的消息，一个负责接收服务端发出的信息，通过这两个线程保障了收发可以同时进行（即全双工），在负责发消息的 send_msg 中实现一循环结构即可（在发送信息时需要 encode ），当检测到输入“q”时关闭客户端连接，在负责收消息的 recv_msg 也要实现一循环结构（收到的信息需要 decode ），不过需要 try except 结构通过检测是否出错来判断连接是否断开。客户端补全后的代码如下：

```
import socket
from threading import Thread

BUFFER_SIZE = 1024

class Client():
    def __init__(self):
        #---------------------------------------------
        # TODO:  初始化客户端socket
        # self.client =
        #---------------------------------------------
        self.client = socket.socket()
        self.ip, self.port = self.set_ip_port()  # 通过命令行标准输入，设置服务器IP与端口


    def set_ip_port(self):
        print("请输入聊天服务器IP")
        ip = input()
        # ip = "127.0.0.1"
        print("请输入聊天服务器端口")
```

```
        port = input()
        return ip,int(port)

    def start_connection(self):
        #---------------------------------------------
        # TODO:  通过socket连接至对应IP与端口
        # self.client.XXXXXXX
        #---------------------------------------------
        self.client.connect((self.ip,self.port))
        print("与" + self.ip + "连接建立成功，可以开始聊天了！（输入q断开连接）")
        # 为接受消息和发送消息分别开启两个线程，实现双工聊天
        Thread(target=self.send_msg).start()
        Thread(target=self.recv_msg).start()

    def send_msg(self):
        #---------------------------------------------
        # TODO:  在本函数中实现Socket消息的接收，并实现输入q退出的功能
        # 提示：需要循环结构
        #---------------------------------------------
        msg = ''
        while msg != 'q'.encode('utf-8'):
            msg = input().encode('utf-8')
            self.client.send(msg)
        self.client.close()



    def recv_msg(self):
        #---------------------------------------------
        # TODO:  在本函数中实现Socket消息的接收
        # 提示：需要循环结构
        # 提示：send_msg子进程退出并关闭socket时会报错，因此需要用try except结构进行异常处理
        #---------------------------------------------
        while True:
            try:
                msg = self.client.recv(BUFFER_SIZE).decode('utf-8')
                print(msg)
            except:
                break



if __name__ == '__main__':
    client = Client()
    client.start_connection()
```

然后编写服务端程序，类中的初始化（__init__）比较简单，只需要补上所需函数即可（socket、bind、settimeout、listen等）， p2p_send_msg 和 p2p_recv_msg 的实现和客户端中相应代码类似，这样服务端的 p2p 模式就可以正常运行了，可以在服务端和客户端正常地收发消息。

此后要实现服务端的 hub 模式：在选择 hub 模式后， start_hub_listen 开始接收各个客户端的连接请求，每收到一个请求，则开启一个 hub_msg_process 线程，负责接收已连接客户端发来的信息，并将信息广播到其他客户端，同时在收到“q”后，利用 hub_close_client 关闭与该客户端的连接，并将断开连接信息广播至其他客户端，要特别注意的是，当已连接客户端数目达到设定最大值后，后续连接请求会被挂起，这些被挂起的客户端发送的消息不会被广播到其他客户端且其不会收到其他客户端的消

息，直到有已连接客户端退出，则被挂起的客户端依次自动进入聊天室，并收到一条“ xxx 已进入聊天室”（ xxx 为该客户端地址），此后就可以正常聊天了。服务端补全后的代码如下：

```python
import socket
from threading import Thread

BUFFER_SIZE = 1024

class Server():
    def __init__(self):
        #---------------------------------------------
        # TODO:  初始化服务端socket
        # self.server =
        #---------------------------------------------
        self.server = socket.socket()
        self.ip = "127.0.0.1"          # 服务器IP为local host，即本机
        self.port = self.set_port()    # 通过命令行标准输入，设置服务器端口

        #---------------------------------------------
        # TODO:  为服务socket绑定IP与端口
        # self.server.XXXXXXXX(params)
        #---------------------------------------------
        self.server.bind((self.ip,self.port))
        self.mode = self.get_mode()

        #---------------------------------------------
        # TODO:  设置服务端默认timeout时间(必须有)
        # self.server.XXXXXXXX(params)
        #---------------------------------------------
        self.server.settimeout(1000)
        self.max_clients = self.set_max_clients()

        #---------------------------------------------
        # TODO:  设置服务端最大连接的客户端数量(必须有)
        # self.server.XXXXXXXX(params)
        #---------------------------------------------
        self.server.listen(self.max_clients)


        if self.mode == 'p2p':
            # 必做：实现p2p聊天
            self.start_p2p_listen()
        else:
            # 选做：实现聊天室服务器功能
            self.ip_client = {}
            self.start_hub_listen()

    def set_port(self):
        print("请输入聊天服务器端口")
        port = input()
        return int(port)

    def get_mode(self):
        print("请输入服务器工作模式(p2p,hub)")
        mode = input()
        return mode

```

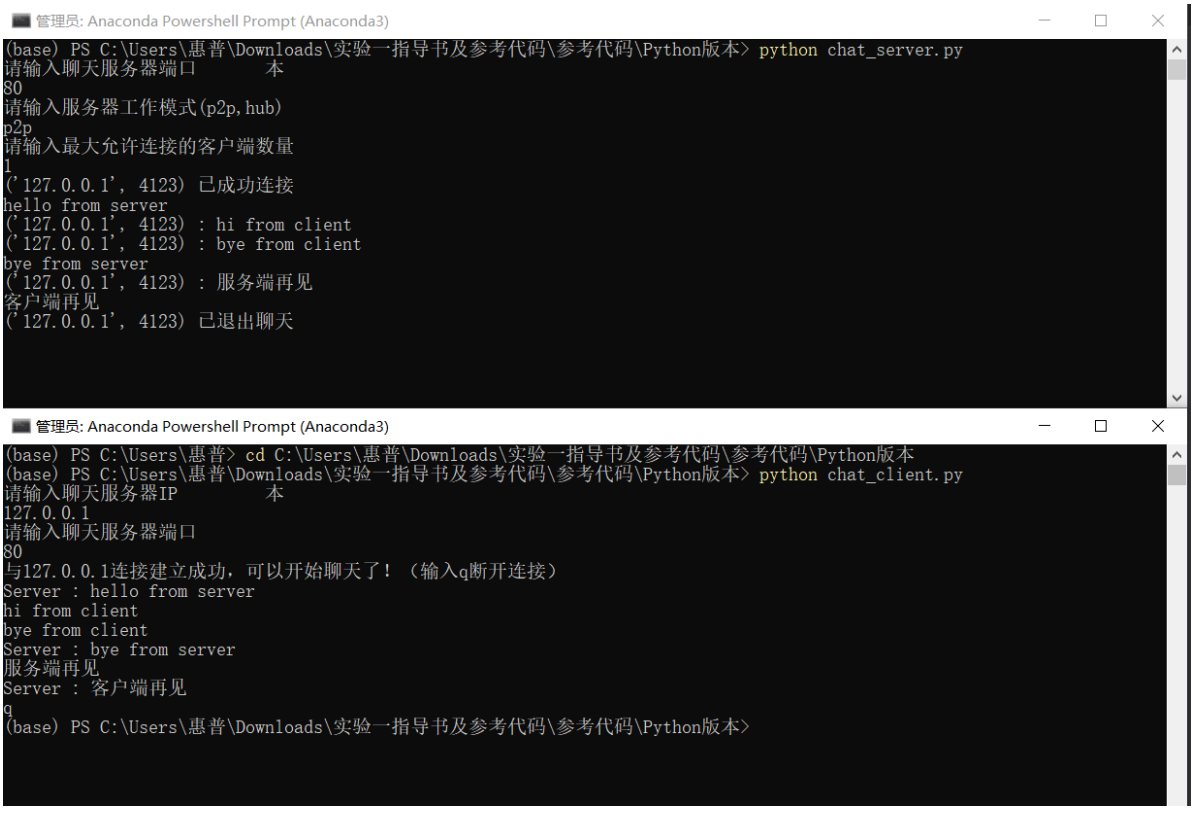
```
    def set_max_clients(self):
        print("请输入最大允许连接的客户端数量")
        max_clients = input()
        return int(max_clients)


################################################################################
    # 工作方式1：p2p连接服务器
    def start_p2p_listen(self):
        #-----------------------------------------
        # TODO： 等待建立连接(必须有)，当用户连接时打印消息，如(ip, port)已成功连接
        # 提示：需要循环结构
        # self.server.XXXXXXXX(params)
        #-----------------------------------------
        while True:
            client, cip_port=self.server.accept()
            if client:
                break
        print(cip_port,"已成功连接")

        #-----------------------------------------
        # TODO： 为接受消息和发送消息分别开启两个线程，实现双工聊天
        # 此处仅需替换param位置的参数；根据上一个位置的返回值仅需更改
        # Thread(target=self.p2p_send_msg,args=(param,)).start()
        # Thread(target=self.p2p_recv_msg,args=(param,)).start()
        #-----------------------------------------
        Thread(target=self.p2p_send_msg,args=(client,)).start()
        Thread(target=self.p2p_recv_msg,args=(client,)).start()

    def p2p_send_msg(self,client):
        #-----------------------------------------
        # TODO： 实现发送消息功能
        # 提示1：字符串必须先encode才能发送
        # 提示2：获得标准输入参考本例程其他函数
        # 提示3：需要循环结构
        # 提示4：当recv_msg收到用户退出通知，并关闭socket后，此子进程会报错，需要通过try except进行异常处理
        #-----------------------------------------
        msg = ''
        while True:
            try:
                msg = ("Server : "+input()).encode('utf-8')
                client.send(msg)
            except:
                break

    def p2p_recv_msg(self,client):
        #-----------------------------------------
        # TODO： 实现接受消息功能，客户端发送q则退出，并打印退出消息，如(ip, port)已退出聊天
        # 提示1：接收到的消息必须先decode才能转换为字符串
        # 提示2：打印到标准输出参考本例程其他函数
        # 提示3：需要循环结构
        #-----------------------------------------
        while True:
            msg = client.recv(BUFFER_SIZE).decode('utf-8')
            if msg == 'q':
```

```
                print(client.getpeername(),"已退出聊天")
                client.close()
                self.server.close()
                break
            else:
                print(client.getpeername(),":",msg)



################################################################################
    # 工作方式2：hub聊天室服务器（广播各个用户发送的信息）
    def start_hub_listen(self):
        #-------------------------------------------
        # 选做
        # TODO： 等待建立连接(必须有)，当用户连接时打印消息，如(ip, port)已成功连接
        # 提示1：需要循环结构
        # 提示2：推荐使用字典数据格式，利用self.ip_client将(ip,port)与client的键值对进行保存，方便管理多个用户
        # 提示3：在循环结构中，每个用户连接后利用此命令开启进程
        # Thread(target=self.hub_msg_process,args=(parm1,parm2 self.ip_client)).start()
        # 提示4：各个线程之间不会对传入参数进行拷贝，因此ip_client会由主线程动态更新
        # self.server.XXXXXXXX(params)
        #-------------------------------------------
        self.num = 0
        while True:
            while self.num >= self.max_clients:
                pass
            client, cip_port = self.server.accept()
            self.ip_client[cip_port] = client
            msg = str(cip_port)+'已进入聊天室'
            msg = msg.encode('utf-8')
            for key, value in self.ip_client.items():
                value.send(msg)
            Thread(target = self.hub_msg_process, args = (client,cip_port,self.ip_client)).start()
            print(cip_port,"已成功连接")
            self.num += 1

    def hub_msg_process(self,current_client, current_address, ip_client):
        #-------------------------------------------
        # 选做
        # TODO： 接受当前client发送的消息，并广播给其他所有client；
        # 当某一用户发送q时，退出该用户，并将其退出消息广播至其他所有用户
        # 提示1：需要循环结构
        # 提示2：需要调用self.hub_close_client函数退出用户线程并实现上述退出消息广播至其他所有用户的功能
        # 提示3：利用ip_client字典进行广播：for key, value in ip_client.items()；广播时，不能广播到自己
        #-------------------------------------------
        while True:
            msg = current_client.recv(BUFFER_SIZE).decode('utf-8')
            if msg == 'q':
                self.hub_close_client(current_client, current_address)
                self.num -= 1
                break
            else:
```

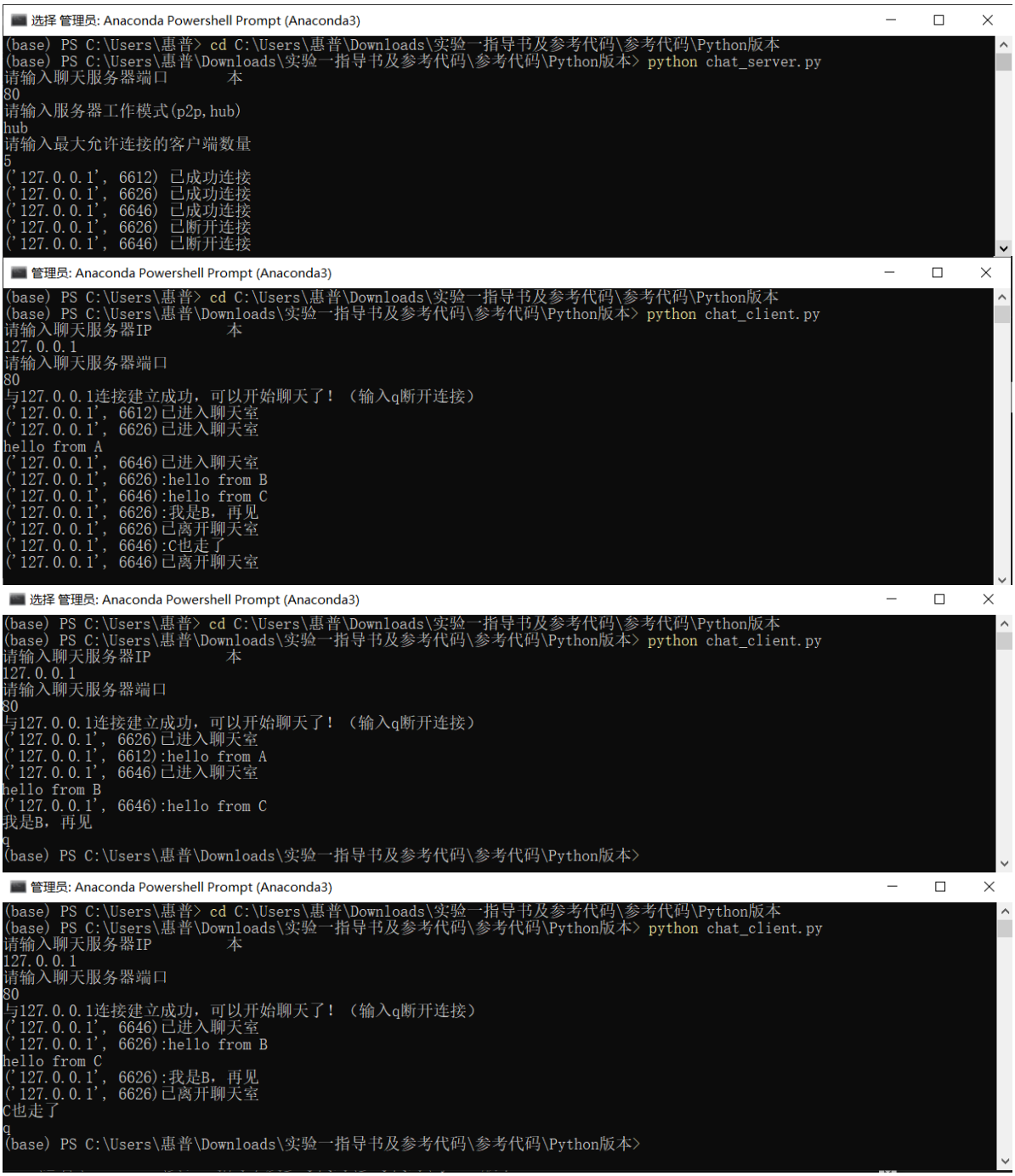
```
                msg = str(current_address)+':'+msg
                msg = msg.encode('utf-8')
                for key, value in ip_client.items():
                    if key != current_address:
                        value.send(msg)

    def hub_close_client(self, client, address):
        #-----------------------------------------------
        # 选做
        # TODO:  关闭该客户socket连接，将其退出消息广播至所有其他在线用户
        # 提示：从字典中删除元素:del(ip_client[key])
        #-----------------------------------------------
        client.close()
        del(self.ip_client[address])
        msg = str(address)+'已离开聊天室'
        msg = msg.encode('utf-8')
        for key, value in self.ip_client.items():
            value.send(msg)
        print(address,'已断开连接')



if __name__ == '__main__':
    server = Server()
```

实验结果

首先测试 p2p 模式，运行上述代码，开启一个服务端和客户端并将二者连接，互相发送消息并退出，实验截图如下：



再测试 hub 模式，运行上述代码，开启一个服务端和三个客户端，各个客户端发消息并退出，实验截图如下：



可以看到无论是 p2p 还是 hub 模式，都实现了正常工作。

实验思考题

(1) 本实验中提供的代码框架使用多线程分别处理消息接收与消息发送,若取消代码中的多线程部分，会出现什么现象？请分析现象原因。

对于p2p模式，两个子线程分别对应于信息的接收和发送，该多线程实现了双工通信，即可以接收消息和发送消息同时进行，因此若取消代码中的多线程部分，则程序会被堵塞在接收信息处（因为 recv 函数是阻塞式的），每发送一个信息都会等待对方答复，收到答复后才能发送下一条信息。

对于 hub 模式，多线程还保证了对多个客户端的同时监听，若取消了多线程，（不采用特殊的设计时）则各个客户端也只能依次发言了，同样是因为 recv 函数会产生阻塞。

(2) 除多线程外，有无其他方式实现 Socket 双工通信?

可以采用轮询的方式，在循环中先检测是否有收到对方的消息，再检测己方是否有输入消息，只要轮询频率够快，就可以实现双工通信的效果，要特别注意的是，此处使用的 recv 函数应该为非阻塞式的（根据网上资料，貌似可以通过 setblocking(0) 来设置）。

(3) 若使用基于 UDP 的 socket，聊天软件是否能正常工作？二者在使用上有何不同?

若直接把本实验中的 TCP 连接替换为 UDP 连接，应该是不能正常工作的，因为会出现丢失和乱序的问题（不清楚发生这种情况的严重程度）。但是如果采用了合理的优化算法（比如客户端使用UDP协议发出消息后，服务器收到该包，需要使用UDP协议发回一个应答包）弥补了 UDP 的这些缺点，则聊天软件可以正常工作，资料显示 QQ 和微信都用到了 UDP 通信协议，证明了基于 UDP 的聊天软件的可行性。

在使用上， TCP 协议需要先建立服务端和客户端之间的连接才能收发信息，以字节流形式按序传送，通过可靠数据传输原理保障了通信可靠，而 UDP 协议无需连接，每个数据报作为独立的包，没有对数据传输的保障机制，通信不可靠，因此 UDP 更容易出现丢包和乱序。