

实验一 Socket 网络编程实验

1. 实验目的

- (1) 理解 Socket 套接字在计算机网络中的位置与作用；
- (2) 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。

2. 实验内容

- (1) 学习并理解 Socket 的编程原理与基本知识；
- (2) 掌握进程中调用 Socket 相关接口的基本方法；
- (3) 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；
- (4) 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；
- (5) 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；
- (6) 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；
- (7) **【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

3. 实验原理

根据 RFC 147 的定义，**socket** 是计算机网络中报文发送和接收的唯一标识符。**Socket** 通过主机 IP 与端口号表示，如(166.111.4.98, 80)。

A socket is defined to be the unique identification to or from which information is transmitted in the network. The socket is specified as a 32 bit number with even sockets identifying receiving sockets and odd sockets identifying sending sockets. A socket is also identified by the host in which the sending or receiving processer is located.

<https://www.rfc-editor.org/rfc/rfc147.html>

运行在不同主机上的进程彼此通过向套接字发送报文来进行通信，实现信息交互。因此，**socket** 设计的目的是将更加底层的传输层和网络层等功能进行抽象，从而方便开发者进行调用，如图 1 所示。**Socket** 可以基于无连接的 UDP 协议，也可基于面向连接的 TCP 协议。在本实验中，我们将利用 **TCP socket** 实现进程间的通信，并完成两台计算机相互聊天的功能，实现基于客户端-服务器模式的网络应用。

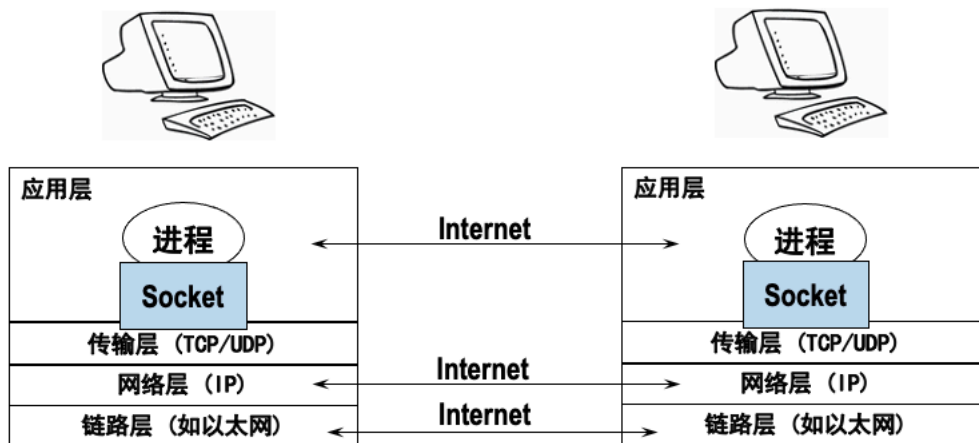


图 1 Socket 抽象

下面介绍 UDP socket 与 TCP socket 在编程上的主要流程。对于 UDP socket，服务端需要创建 socket 并绑定端口号，而 IP 一般为本机 IP。接下来，服务器等待用户请求，并在接收到用户请求后发送对应响应。客户端的 UDP socket 流程与服务端基本类似，首先创建 socket 并绑定端口号，之后向服务器发送请求，并等待接收服务器回复的响应。在此过程中，无需预先建立连接。而对于 TCP socket，服务端在创建 socket 并绑定端口号后，需持续等待客户端的连接请求，在接收到连接请求后建立与客户端的连接，之后等待客户端发送请求内容，并回复对应的响应。对于客户端，创建 socket 后无需绑定端口，直接向服务器发送连接请求，等待连接建立后即可发送请求内容，等待接收服务器响应。以 Unix 下的 socket API 为例，上述 UDP socket 与 TCP socket 的流程如下所示：

UDP Server:

socket() -> bind() -----> recvfrom() -> sendto()

UDP Client:

socket() -> bind() -> sendto() -----> recvfrom()

TCP Server:

socket() -> bind() -> listen() -----> accept() -----> recv() -> send()

TCP Client:

socket()-----> connect() -----> send()-----> recv()

4. 实验环境和操作流程

4.1 实验环境配置与准备

本次实验推荐基于 Python 进行，它具有简洁易用的特点，可以更加方便地实现 Socket 编程的核心操作。同时 Python 具有良好的跨平台兼容性，相同代码可方便地迁移到不同平台。首先我们需要在计算机上安装 Python。在这里，推荐使用 Anaconda，它融合了常用的 Python 虚拟环境管理器 conda，同时默认安装

了各类常用 Python 包，方便使用。

进入 Anaconda 官网¹，点击 Download 即可下载本机适配的 Anaconda 版本，也可通过课程提供的清华云盘链接进行下载²。云盘中提供了 Windows 版、Mac Intel 芯片版（x86-64）以及 Apple Silicon 芯片版（arm64），可按需下载。

安装完成后，打开 Windows PowerShell（左下角 Windows 徽标右键）或 Mac Terminal，输入 ipython 并回车，出现 Python 命令提示符即安装成功，如图 2 所示。关于 Anaconda 更复杂的使用本实验并不涉及，学有余力同学可参考以下链接进行学习³。

图 2 验证 Anaconda 与 Python 环境安装

注意，**Windows 系统的防火墙较为严格，需要在进行 socket 实验时暂时关闭，才能运行服务端程序。**

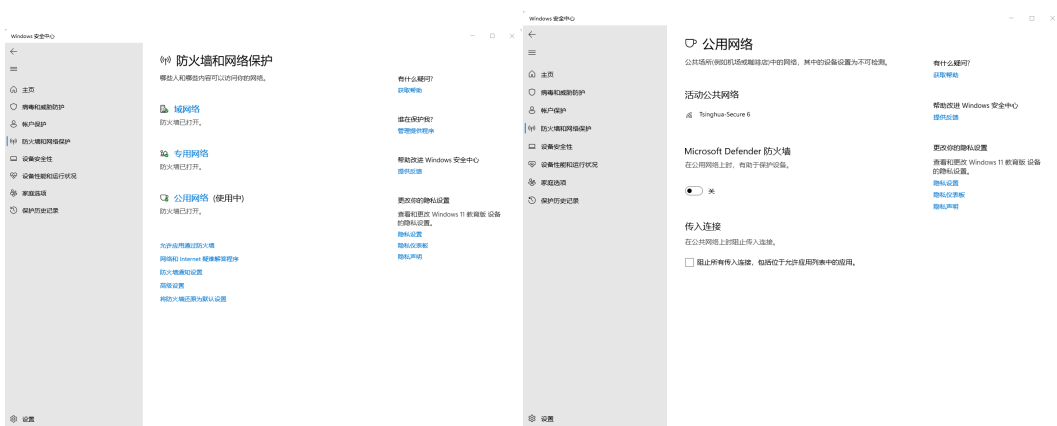


图 3 Windows 关闭防火墙（找到使用中的网络，将其防火墙关闭）

另外，本次实验还提供了 C++版本的代码框架，无法使用 Python 的同学可用 C++版本进行实验。需注意的，不同系统版本中的 socket 实现有所不同。基于 Unix/Linux 的系统采用的是 sys/socket.h 头文件，而 Windows 平台使用的是 winsock2.h 头文件，且必须依赖 Visual Studio 方可编译运行。考虑到可迁移性，本实验 C++版本实现基于 Unix/Linux 平台，提供的代码框架可直接在基于 Unix 的

¹ <https://www.anaconda.com/products/distribution#Downloads>

² <https://cloud.tsinghua.edu.cn/d/6af4682fe0d84bacb092/>

³ <https://www.jianshu.com/p/2f3be7781451>

平台使用（如 Ubuntu、macOS）。**Windows 平台同学可安装 Windows Subsystem for Linux (WSL)，或使用实验提供的 VMWare Ubuntu 虚拟机⁴进行 C++版本实验（对于未安装过 WSL 的同学，推荐使用课程提供的虚拟机，在后续实验中会继续使用；虚拟机安装方式见附录；仅使用 Windows 平台的 C++版本需要按照附录操作，Python 版本代码全平台通用）。**

注意，虚拟机内附带的代码框架可能较旧，请将网络学堂中的最新代码上传至虚拟机（鼠标拖拽即可）。

在此提供经测试的编译命令：

macOS 版：

```
clang++ chat_client.cpp -o chat_client
```

```
clang++ chat_server.cpp -o chat_server
```

常见 Linux 发行版：

```
g++ chat_client.cpp -o chat_client
```

```
g++ chat_server.cpp -o chat_server
```

表1 实验主要文件及功能

文件名	功能及说明
<i>chat_client.py</i>	客户端程序代码，连接服务端并进行聊天通信（需补全）
<i>chat_server.py</i>	服务端程序代码，等待客户端连接并进行聊天通信（需补全）；【选做】包含 python 版本聊天室代码框架
<i>python_use1.py</i>	基础 python 语法示例
<i>cpp_p2p/chat_client.cp</i> <i>p</i>	Unix 版本 C++服务端代码，实现一对一并聊天通信（需补全）
<i>cpp_p2p/chat_server.cp</i> <i>p</i>	Unix 版本 C++服务端代码，等待客户端连接并进行聊天通信（需补全）
<i>cpp_hub 目录</i>	【选做】包含 macOS 与 Linux 版聊天室代码框架

4.2 Python 下的 Socket 编程

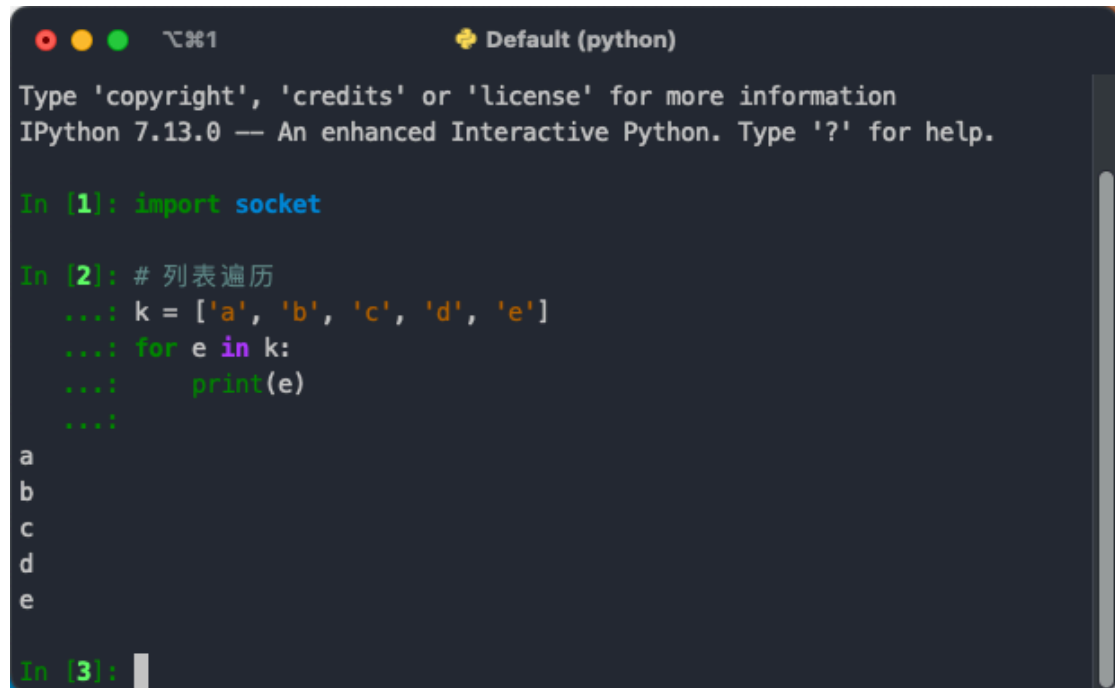
首先简要介绍 Python 的基本使用。在上述打开的命令行窗口中，我们可以交互式地执行命令，并得到其结果，无需编译过程，因此有所见即所得的特点，方便修改代码与调试。

本次实验中用到的主要语法均总结在 *python_use1.py* 中，可从中复制粘贴代

⁴ <https://cloud.tsinghua.edu.cn/f/ab6b3561edfe41a1ac85/>

码段到命令行窗口中尝试执行，简单掌握语法规则。如图 4 所示。对于编写好的.py 文件，也可在命令行中（交互式终端外）进行执行。首先，按 **control+d** 退出 **ipython** 交互式命令行回到终端，利用 **cd** 命令定位至代码存放的目录，输入 **python python_use.py** 执行代码，如图 5 所示。

除提供的示例文件以外，不熟悉 Python 编程语言的同学推荐参考以下链接学习 Python 语法^{5 6}。



```

Type 'copyright', 'credits' or 'license' for more information
IPython 7.13.0 -- An enhanced Interactive Python. Type '?' for help.

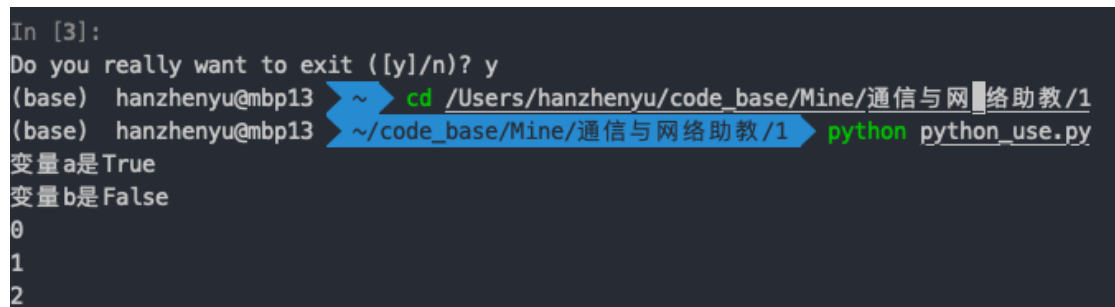
In [1]: import socket

In [2]: # 列表遍历
...: k = ['a', 'b', 'c', 'd', 'e']
...: for e in k:
...:     print(e)
...:
a
b
c
d
e

In [3]:

```

图 4 交互式窗口中执行代码



```

In [3]:
Do you really want to exit ([y]/n)? y
(base) hanzhenyu@mbp13 ~$ cd /Users/hanzhenyu/code_base/Mine/通信与网络助教/1
(base) hanzhenyu@mbp13 ~/code_base/Mine/通信与网络助教/1$ python python_use.py
变量a是True
变量b是False
0
1
2

```

图 5 完整执行 python 程序

接下来，介绍在 Python 中利用 Socket 编程的基础知识。在利用 Socket 编程时，首先需要导入 **socket** 包，并初始化一个 **Socket** 对象的实例。

```

import socket
s = socket.socket()

```

实例初始化后，我们需利用 **Socket** 对象的内建方法进行连接配置，以实现其

⁵ <https://www.runoob.com/python3/python3-basic-syntax.html>

⁶ <https://www.liaoxuefeng.com/wiki/1016959663602400/>

连接功能。常用的 Socket 内建方法总结如表 2 所示，其中在本次实验中着重需关注的 API 已标红。可参考 Python 官方 API 文档对各个方法进行详细了解⁷。

表2 Socket对象内建方法

函数	描述
服务器端套接字	
s.bind()	绑定地址（host,port）到套接字， 在 AF_INET 下,以元组（host,port）的形式表示地址。
s.listen()	服务器开始 TCP 监听。 backlog 指定在拒绝连接之前，操作系统可以挂起的最大客户端连接数量：该值至少为 1，大部分应用程序设为 5 就可以了。
s.accept()	被动接受 TCP 客户端连接,(阻塞式)等待客户端连接的到来
客户端套接字	
s.connect()	主动初始化 TCP 服务器连接，。一般 address 的格式为元组（hostname,port），如果连接出错，返回 socket.error 错误。
s.connect_ex()	connect()函数的扩展版本,出错时返回出错码,而不是抛出异常
公共用途的套接字函数	
s.recv()	接收 TCP 数据，数据以字符串形式返回， bufsize 指定要接收的最大数据量。 flag 提供有关消息的其他信息，通常可以忽略。
s.send()	发送 TCP 数据，将 string 中的数据发送到连接的套接字。返回值是要发送的字节数量，该数量可能小于 string 的字节大小。
s.sendall()	完整发送 TCP 数据。将 string 中的数据发送到连接的套接字，但在返回之前会尝试发送所有数据。成功返回 None ，失败则抛出异常。

⁷ <https://docs.python.org/3/library/socket.html>

函数	描述
s.recvfrom()	接收 UDP 数据，与 <code>recv()</code> 类似，但返回值是 <code>(data,address)</code> 。其中 <code>data</code> 是包含接收数据的字符串， <code>address</code> 是发送数据的套接字地址。
s.sendto()	发送 UDP 数据，将数据发送到套接字， <code>address</code> 是形式为 <code>(ipaddr, port)</code> 的元组，指定远程地址。返回值是发送的字节数。
s.close()	关闭套接字
s.getpeernam e()	返回连接套接字的远程地址。返回值通常是元组 <code>(ipaddr,port)</code> 。
s.getsocknam e()	返回套接字自己的地址。通常是一个元组 <code>(ipaddr,port)</code>
s.setsockopt(le vel,optname,va lue)	设置给定套接字选项的值。
s.getsockopt(le vel,optname[.b uflen])	返回套接字选项的值。
s.settimeout(ti meout)	设置套接字操作的超时期， <code>timeout</code> 是一个浮点数，单位是秒。值为 <code>None</code> 表示没有超时期。一般，超时期应该在刚创建套接字时设置，因为它们可能用于连接的操作（如 <code>connect()</code> ）
s.gettimeout()	返回当前超时期的值，单位是秒，如果没有设置超时期，则返回 <code>None</code> 。
s.fileno()	返回套接字的文件描述符。
s.setblocking(fl ag)	如果 <code>flag</code> 为 <code>False</code> ，则将套接字设为非阻塞模式，否则将套接字设为阻塞模式（默认值）。非阻塞模式下，如果调用 <code>recv()</code> 没有发现任何数据，或 <code>send()</code> 调用无法立即发送数据，那么将引起 <code>socket.error</code> 异常。

函数	描述
s.makefile()	创建一个与该套接字相关连的文件

4.3 实现基于 Socket 的聊天客户端程序

首先,本次实验将实现 Socket 聊天客户端的功能。在这里,客户端将通过命令行方式连接到课程提供的服务端程序上(服务端程序 IP 及端口将在课上发布),连接后可向服务端程序发送标准输入流中获取的文本消息(即命令行窗口的用户输入),该消息将被广播至所有其他用户。同时,其他用户的消息也将通过服务端程序转发至客户端,实现双向通信。

实验已经提供客户端的框架代码 `chat_client.py` 与 `chat_client.cpp`。同学需根据上述 Socket 使用方法,实现其中标记为 TODO 位置的函数部分,以完成所需功能。需要注意的是,为了避免进程阻塞,示例代码中通过多线程的方式实现了双工通信,两个子进程分别负责 Socket 消息的接收与发送。对 Python 多进程、多线程感兴趣的同学可通过以下资料自学^{8 9},本次实验无需对其内涵进行深刻了解。

下面演示正常工作的程序流程,如图 6 所示。在两个终端中分别运行两个 client,通过命令行输入助教提供的服务端程序 IP 及端口(需注意,此处示例在本机进行,故 IP 为 127.0.0.1 (Baidu 这个 IP 的具体含义);实验时需根据助教安排设置正确 IP)。在这里,助教将提供两个服务端程序用于同学测试客户端程序:

1) 服务器程序维护一个聊天室,将某用户发送至服务端的信息广播到其他所有用户;

2) 服务器程序原样返回用户输入。

以第一个服务端功能为例,首先,在 client2 中输入消息,可以在 client1 的窗口中看到;其次,在 client1 的窗口中输入消息,可以在 client2 的窗口中看到。上述两个步骤验证了程序信息收发功能的正常。最后,在 client2 窗口中输入 q, client2 退出, client1 中收到 client2 退出的消息,完成整个流程验证。

C++版本中的一对一聊天程序客户端在执行时需提供端口号,如图 7 所示。此外 C++版本并未采用多线程或多进程,在对方回复前无法继续发送信息。

⁸ <https://docs.python.org/3/library/multiprocessing.html?highlight=process#module-multiprocessing>

⁹ <https://www.liaoxuefeng.com/wiki/1016959663602400/1017627212385376>



```

Last login: Tue Sep  6 16:17:50 on ttys002
(base) hanzhenyu@mbp13 ~$ cd ~/code_base/Mine/通信与网络助教/1/完整答案
(base) hanzhenyu@mbp13 ~$ python chat_client.py
请输入聊天服务器IP
127.0.0.1
请输入聊天服务器端口
1234
与127.0.0.1连接建立成功，可以开始聊天了！（输入q断开连接）
input from client 1
('127.0.0.1', 61743):input from client 2
('127.0.0.1', 61743)离开了
q

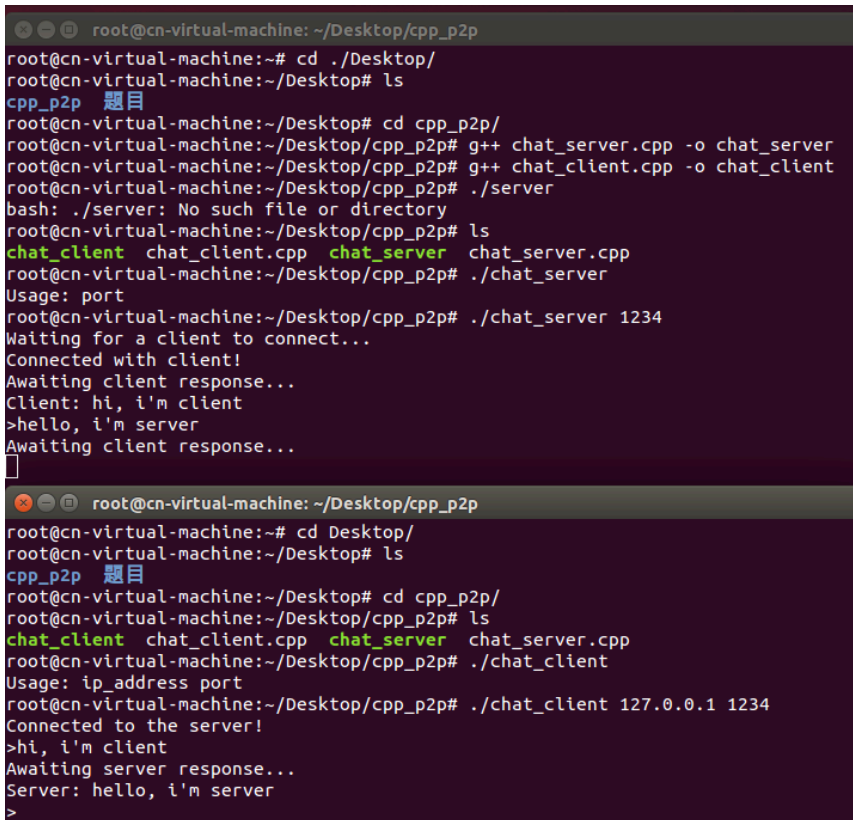
Last login: Tue Sep  6 16:17:19 on ttys001
(base) hanzhenyu@mbp13 ~$ cd ~/code_base/Mine/通信与网络助教/1/完整答案
(base) hanzhenyu@mbp13 ~$ python chat_client.py
请输入聊天服务器IP
127.0.0.1
请输入聊天服务器端口
1234
与127.0.0.1连接建立成功，可以开始聊天了！（输入q断开连接）
('127.0.0.1', 61750):input from client 1
input from client 2
q

```

能够发送消息到其他 client
能够实现退出功能

能够接收到其他 client 的消息

图 6 client 程序验证



```

root@cn-virtual-machine: ~/Desktop/cpp_p2p
root@cn-virtual-machine:~# cd ./Desktop/
root@cn-virtual-machine:~/Desktop# ls
cpp_p2p  题目
root@cn-virtual-machine:~/Desktop# cd cpp_p2p/
root@cn-virtual-machine:~/Desktop/cpp_p2p# g++ chat_server.cpp -o chat_server
root@cn-virtual-machine:~/Desktop/cpp_p2p# g++ chat_client.cpp -o chat_client
root@cn-virtual-machine:~/Desktop/cpp_p2p# ./server
bash: ./server: No such file or directory
root@cn-virtual-machine:~/Desktop/cpp_p2p# ls
chat_client  chat_client.cpp  chat_server.cpp
root@cn-virtual-machine:~/Desktop/cpp_p2p# ./chat_server
Usage: port
root@cn-virtual-machine:~/Desktop/cpp_p2p# ./chat_server 1234
Waiting for a client to connect...
Connected with client!
Awaiting client response...
Client: hi, i'm client
>hello, i'm server
Awaiting client response...

root@cn-virtual-machine:~/Desktop/cpp_p2p
root@cn-virtual-machine:~# cd Desktop/
root@cn-virtual-machine:~/Desktop# ls
cpp_p2p  题目
root@cn-virtual-machine:~/Desktop# cd cpp_p2p/
root@cn-virtual-machine:~/Desktop/cpp_p2p# ls
chat_client  chat_client.cpp  chat_server.cpp
root@cn-virtual-machine:~/Desktop/cpp_p2p# ./chat_client
Usage: ip_address port
root@cn-virtual-machine:~/Desktop/cpp_p2p# ./chat_client 127.0.0.1 1234
Connected to the server!
>hi, i'm client
Awaiting server response...
Server: hello, i'm server
>

```

图 7 一对一聊天服务端验证（C++版本）

4.4 实现基于 Socket 的一对一聊天服务端程序

接下来，实验需要完成简单服务端程序的设计。在这里，需要实现一个包括客户端和服务端端的一对一的聊天服务，即服务端程序等待客户端连接，当有客户端连接成功后，实现客户端与服务端的聊天对话。

实验已提供服务端的框架代码 `chat_server.py` 与 `chat_server.cpp`。同学需根据上述 `Socket` 使用方法，实现其中标记为 `TODO` 位置的函数部分，以完成所需功能。需要注意的是，此代码包含两个功能，即一对一的聊天服务（`p2p`）与聊天室（`hub`）功能。本部分要求实现 `p2p` 聊天功能，`hub` 功能参考 4.5 部分内容。

下面演示正常工作的程序流程，如图 8 所示。开启服务端后，通过命令行设置端口与工作模式（`p2p`），并设置最大运行的客户端数量后，打开客户端进行连接。双方收发的消息应均正常工作，同时需正确关闭 `socket` 连接。

C++版本中的一对一聊天程序服务端在执行时需提供端口号与 IP 地址，如图 7 所示。此外 C++版本并未采用多线程或多进程，在对方回复前无法继续发送信息。



```
(base) hanzhenyu@mbp13 ~/code_base/Mine/通信与网络助教/1/完整答案 python chat_client.py
请输入聊天服务器IP
127.0.0.1
请输入聊天服务器端口
1234
与 127.0.0.1 连接建立成功，可以开始聊天了！（输入q断开连接）
hello
hi
q
(base) hanzhenyu@mbp13 ~/code_base/Mine/通信与网络助教/1/完整答案

(Default (-zsh))

(Default (python))
(base) hanzhenyu@mbp13 ~/code_base/Mine/通信与网络助教/1/完整答案 python chat_server.py
请输入聊天服务器端口
1234
请输入服务器工作模式 (p2p, hub)
p2p
请输入最大允许连接的客户端数量
1
与 ('127.0.0.1', 63467) 连接建立成功，可以开始聊天了！
('127.0.0.1', 63467):hello
hi
('127.0.0.1', 63467) 离开了
[]
```

图 8 一对一聊天服务端验证（Python 版本）

4.5 【选做】实现基于 `Socket` 的聊天室服务端程序

最后，可以实现更加复杂的服务端程序，即实现在 4.3 节中助教提供的聊天室服务端功能（`hub`）。在这里，需要服务端维护各个客户端的连接，每当新的客户端建立连接时开启新的子进程，并将客户端消息广播给其他所有客户端。当某个客户端退出连接时，需正确关闭 `socket` 并向其他所有用户广播通知。

验证流程如图 9 所示，当多个用户连接到聊天室服务端后，每个用户输入消息被广播到其他所有用户；同时当用户退出时，其他用户会接收到退出消息。

```

(Default (-zsh))
(base) hanzhenyu@mbp13 ~/code_base/Mine/通信与网络助教/1/完整答案 python chat_client.py
请输入聊天服务器IP
127.0.0.1
请输入聊天服务器端口
1234
与127.0.0.1连接建立成功，可以开始聊天了！（输入q断开连接）
abc
('127.0.0.1', 63887):def
q
(base) hanzhenyu@mbp13 ~/code_base/Mine/通信与网络助教/1/完整答案

(Default (python))
(base) hanzhenyu@mbp13 ~/code_base/Mine/通信与网络助教/1/完整答案 cd ~/code_base/Mine/通信与网络助教/1/完整答案
(base) hanzhenyu@mbp13 ~/code_base/Mine/通信与网络助教/1/完整答案 python chat_client.py
请输入聊天服务器IP
127.0.0.1
请输入聊天服务器端口
1234
与127.0.0.1连接建立成功，可以开始聊天了！（输入q断开连接）
('127.0.0.1', 63879):abc
def
('127.0.0.1', 63879)离开了

(Default (python))
(base) hanzhenyu@mbp13 ~/code_base/Mine/通信与网络助教/1/完整答案 python chat_server.py
请输入聊天服务器端口
1234
请输入服务器工作模式(p2p,hub)
hub
请输入最大允许连接的客户端数量
5
与('127.0.0.1', 63879)连接建立成功，可以开始聊天了！
与('127.0.0.1', 63887)连接建立成功，可以开始聊天了！
('127.0.0.1', 63879)离开了

```

图 9 聊天室服务端功能验证

Python 版本中聊天室程序框架位于 `chat_server.py` 中，而 C++版本则提供另一套代码框架，位于 `cpp_hub` 文件夹下。

C++版本聊天室程序编译方法如下：

macOS 版：

```
clang++ mac_chat_client.cpp -o client -std=c++11
```

```
clang++ mac_chat_server.cpp -o server -std=c++11
```

较新的 Linux 发行版：

```
g++ linux_chat_server.cpp -lpthread -o server
```

```
g++ linux_chat_client.cpp -lpthread -o client
```

本次实验提供的 Ubuntu 虚拟机：

```
g++ linux_chat_server.cpp -lpthread -o server -std=c++11
```

g++ linux_chat_client.cpp -lpthread -o client -std=c++11

5. 实验考核

- (1) 理解并能正确使用 `socket` 进行网络程序开发;
- (2) 根据提供的代码框架实现 `client` 功能;
- (3) 根据提供的代码框架实现一对一聊天的 `server` 功能;
- (4) 【选做】根据提供的代码框架实现聊天室 `server` 功能;
- (5) **网络学堂中提交实验报告+代码，报告内容合乎逻辑，表达清晰，有实验过程记录、截图及思考题回答。**

6. 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。
- (2) 除多线程外，有无其他方式实现 Socket 双工通信？
- (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？

7. 实验参考资料总结

- (1) RFC 147:
<https://www.rfc-editor.org/rfc/rfc147.html>
- (2) Anaconda 官方下载:
<https://www.anaconda.com/products/distribution#Downloads>
- (3) Anaconda 清华云盘备份:
<https://cloud.tsinghua.edu.cn/d/6af4682fe0d84bacb092/>
- (4) Anaconda 基本使用
<https://www.jianshu.com/p/2f3be7781451>
- (5) VMWare 虚拟机（用于 Windows 平台完成 C++ 版本代码）
<https://cloud.tsinghua.edu.cn/f/ab6b3561edfe41a1ac85/>
- (5) Python 基础语法学习
<https://www.runoob.com/python3/python3-basic-syntax.html>
<https://www.liaoxuefeng.com/wiki/1016959663602400/>
- (6) Python Socket 官方 API

<https://docs.python.org/3/library/socket.html>

(7) Python 多线程与多进程

<https://docs.python.org/3/library/multiprocessing.html?highlight=process#module-multiprocessing>

<https://www.liaoxuefeng.com/wiki/1016959663602400/1017627212385376>

8. 附录：安装VMware虚拟机

注意，此部分附录仅供 Windows 平台计算机进行 C++版本实验；其他平台机器或 Python 版本无需参考，按照指导书正文操作即可。

1. 下载并安装 VMware Workstation:

<https://www.vmware.com/products/workstation-pro/workstation-pro-evaluation.html>

2. 下载提供的虚拟机平台:

<https://cloud.tsinghua.edu.cn/f/ab6b3561edfe41a1ac85/>

3. 下载相关配置文件后，在 VMware 主界面中点击右上角“文件(F)”，在弹出来的下拉框中点击“打开(O)”，选择下载文件的目录，打开.vmx 文件，之后根据提示选择.vmdk 文件，点击“开启虚拟机”，选择“我已复制该虚拟机”，完成仿真环境搭建。

4. 启动虚拟机后，将进入 Ubuntu 系统。用户名：cn，密码：12345678

5. 将提供的 C++代码框架补全后拖拽到虚拟机中，完成编译并执行