

## 实验一 Socket 编程实验

### 1. 实验目的

- (1) 理解 Socket 套接字在网络模型中的位置与作用;
- (2) 掌握 Socket 接口的编程方式, 实现两台电脑之间的聊天功能。

### 2. 实验内容

- (1) 学习并理解 Socket 的原理与基本知识;
- (2) 掌握进程中调用 Socket 通信的基本方法;
- (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) 同组同学分别运行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 是网络中信息传递的唯一的标识符。Socket 通过主机 IP 与端口号所表示, 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>

运行在不同机器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、网络层等内容进行抽象, 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。

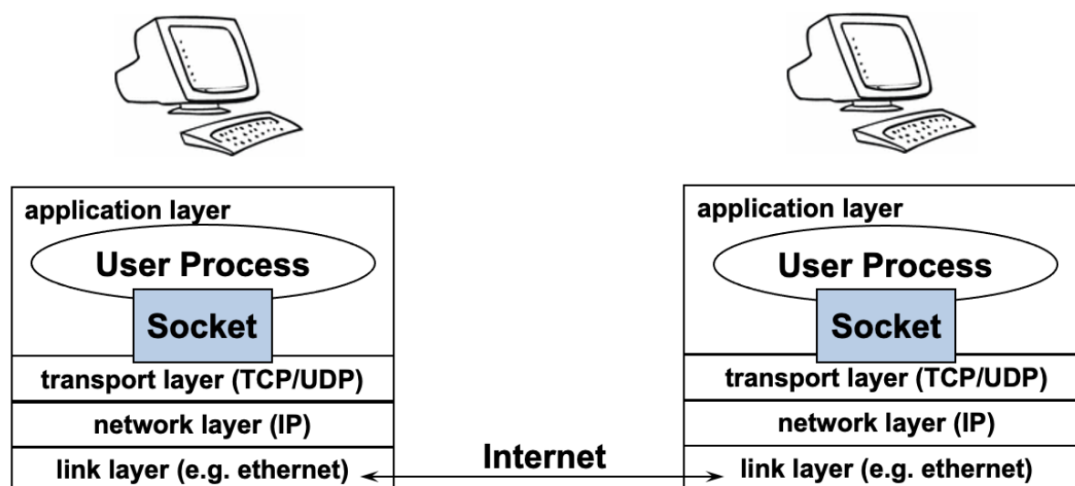


图 1 Socket 抽象

## 4. 实验环境和操作流程

### 4.1 实验环境配置

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<sup>1</sup>，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<sup>2</sup>。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 3 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<sup>3</sup>。

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。**

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

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

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

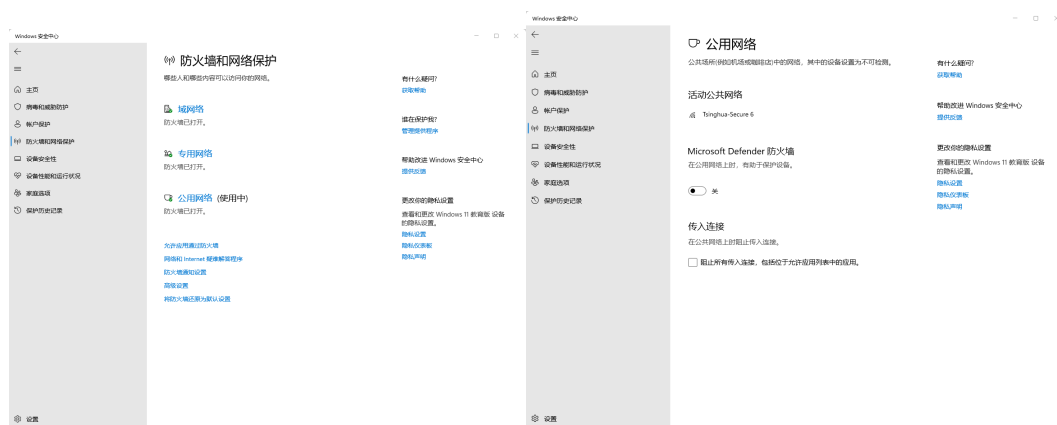


图 2 Windows 关闭防火墙（找到使用中的网络，将其防火墙关闭）

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。需注意的是，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 Linux 的平台使用（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<sup>4</sup>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）。**在此提供经测试的编译命令：

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
```

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

```

Last login: Tue Sep  6 11:00:47 on ttys000
(base) hanzhenyu@mbp13 ~$ ipython
Python 3.7.7 (default, Mar 23 2020, 17:31:31)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.13.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:

```

图 3 验证 Anaconda 与 Python 环境安装

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>	服务端代码，等待客户端连接并进行聊天通信（需补全）
<i>python_use1.py</i>	基础 python 语法示例
<i>mac_chat_client.cpp</i>	macOS 版本 C++客户端代码，连接服务端并进行聊天通信（需补全）
<i>mac_chat_server.cpp</i>	macOS 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>stdc++.h</i>	macOS 版本 C++所需额外 s 头文件
<i>linux_chat_client.cp</i> <i>p</i>	Linux 版本 C++服务端代码，连接服务端并进行聊天通信（需补全）
<i>linux_chat_server.cp</i> <i>p</i>	Linux 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）

## 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，推荐同学参考以下链接学习 Python 语法<sup>5 6</sup>。

```

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 ~ % 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<sup>7</sup>。

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

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

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

表2 Socket对象内建方法

函数	描述
服务器端套接字	
<b>s.bind()</b>	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。
<b>s.listen()</b>	开始 TCP 监听。 <b>backlog</b> 指定在拒绝连接之前, 操作系统可以挂起的最大连接数量。该值至少为 1, 大部分应用程序设为 5 就可以了。
<b>s.accept()</b>	被动接受 TCP 客户端连接,(阻塞式)等待连接的到来
客户端套接字	
<b>s.connect()</b>	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	
<b>s.recv()</b>	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。
<b>s.send()</b>	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。
s.recvfrom()	接收 UDP 数据, 与 recv()类似, 但返回值是 (data,address)。其中 data 是包含接收数据的字符串, address 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, address 是形式为 (ipaddr, port) 的元组, 指定远程地址。返回值是发送的字节数。

函数	描述
<b>s.close()</b>	关闭套接字
<b>s.getpeernam e()</b>	返回连接套接字的远程地址。返回值通常是元组 (ipaddr,port)。
<b>s.getsocknam e()</b>	返回套接字自己的地址。通常是一个元组(ipaddr,port)
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。
<b>s.settimeout(ti meout)</b>	设置套接字操作的超时期，timeout 是一个浮点数，单位是秒。值为 None 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 connect()）
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 None。
s.fileno()	返回套接字的文件描述符。
s.setblocking(fl ag)	如果 flag 为 False，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 recv() 没有发现任何数据，或 send() 调用无法立即发送数据，那么将引起 socket.error 异常。
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消息（即窗口命令行输入），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`。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、多线程感兴趣的同同学可通过以下资料自学<sup>8 9</sup>，本次实验无需对其内涵进行深刻理解。

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`；实验时需根据助教安排设置正确 IP）。在这里，服务端程序起到的作用是维护一个聊天室，将某用户发送至服务端的信息广播到其他所有用户。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。

```

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

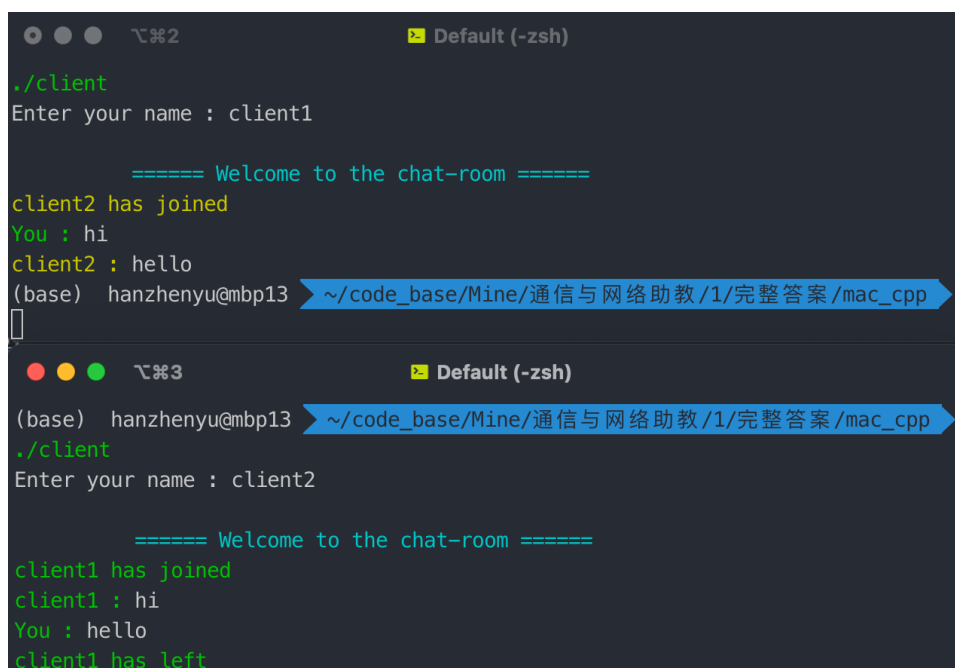
Last login: Tue Sep  6 16:17:19 on ttys001
(base) hanzhenyu@mbp13 ~ % cd ~/code_base/Mine/通信与网络助教/1/完整答案
(base) hanzhenyu@mbp13 ~ % python chat_client.py
请输入聊天服务器IP
127.0.0.1
请输入聊天服务器端口
1234
与 127.0.0.1 连接建立成功，可以开始聊天了！（输入 q 断开连接）
('127.0.0.1', 61750):input from client 1
input from client 2
q
(base) hanzhenyu@mbp13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

图 6 client 程序验证（Python 版本）

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

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





```
Default (-zsh)
./client
Enter your name : client1

===== Welcome to the chat-room =====
client2 has joined
You : hi
client2 : hello
(base) hanzhenyu@mbp13 ~/code_base/Mine/通信与网络助教/1/完整答案/mac_cpp

Default (-zsh)
./client
Enter your name : client2

===== Welcome to the chat-room =====
client1 has joined
client1 : hi
You : hello
client1 has left
```

图 7 client 程序验证（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的构建。在这里，需要实现一个一对一的聊天服务，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`。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 部分选做内容。

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。

**若选择 C++ 版本，则可直接实现 4.5 的聊天室服务端，无需实现一对一聊天服务端程序。**



```
(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 (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

#### 4.5 【选做，C++版本必做】实现基于 Socket 的聊天室服务端程序

最后，学有余力的同学可以尝试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。

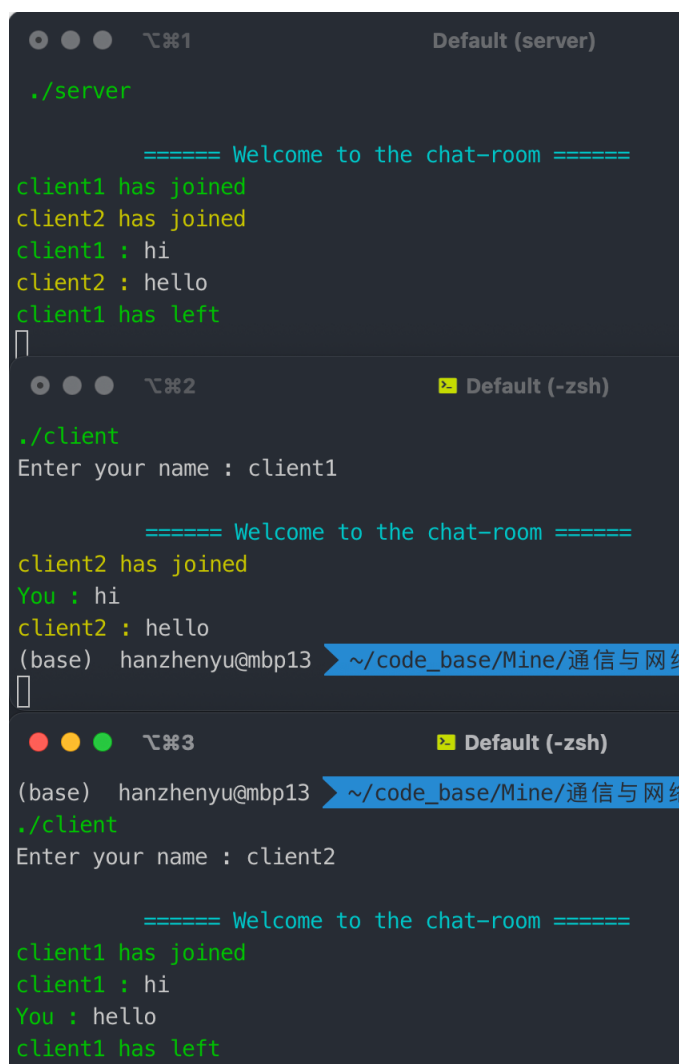
验证流程如图 9 所示，当多个用户连接到聊天室服务端后，每个用户输入消息被广播到其他所有用户；同时当用户退出时，其他用户会接收到退出消息。

```
(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）
abc
('127.0.0.1', 63887):def
q
(base) hanzhenyu@mbp13 ~ % python chat_client.py

(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', 63879):abc
def
('127.0.0.1', 63879)离开了
[]

(base) hanzhenyu@mbp13 ~ % 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）



```
Default (server)
./server
===== Welcome to the chat-room =====
client1 has joined
client2 has joined
client1 : hi
client2 : hello
client1 has left
[]

Default (-zsh)
./client
Enter your name : client1

===== Welcome to the chat-room =====
client2 has joined
You : hi
client2 : hello
(base) hanzhenyu@mbp13 ~/code_base/Mine/通信与网络

Default (-zsh)
(base) hanzhenyu@mbp13 ~/code_base/Mine/通信与网络
./client
Enter your name : client2

===== Welcome to the chat-room =====
client1 has joined
client1 : hi
You : hello
client1 has left
```

图 10 聊天室服务端功能验证（C++版本）

## 5. 实验考核

- (1) 理解并讲解 socket 的作用；
- (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（c++版可不做）；
- (4) 【选做，**c++版本必做**】根据提供的代码框架实现聊天室 server 功能。

## 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，桌面存放本次实验相关文件。
5. 打开 Terminal，进入桌面上的本次实验文件夹，完成代码补全后输入  
g++ linux\_chat\_server.cpp -lpthread -o server -std=c++11  
g++ linux\_chat\_client.cpp -lpthread -o client -std=c++11  
进行编译
6. 运行 server 与 client，测试功能  
./server  
./client