**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）【**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

（7）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

## 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。

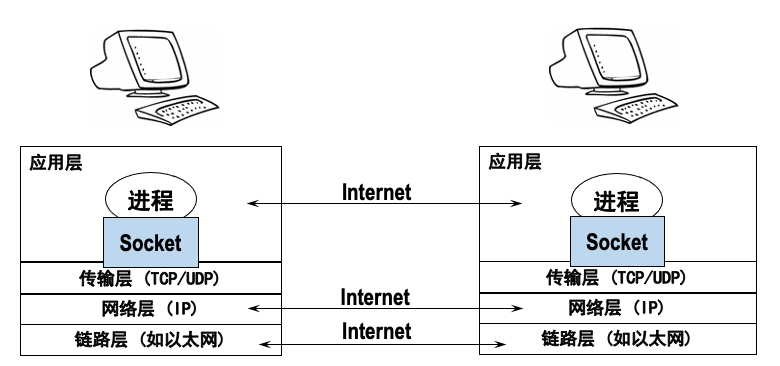


图 1 Socket抽象

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]](#footnote-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]](#footnote-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]](#footnote-3)。

图形用户界面, 文本, 应用程序

描述已自动生成

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**。

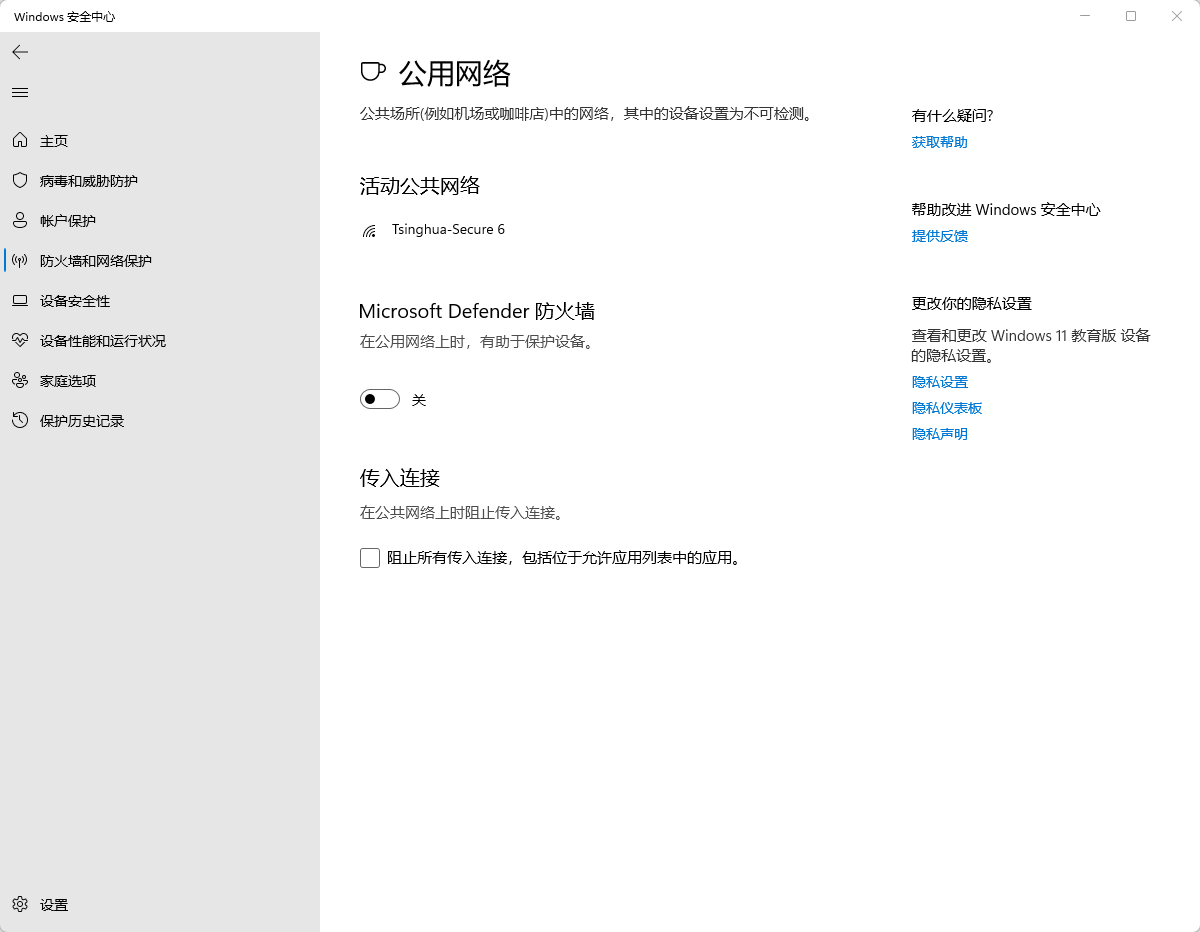
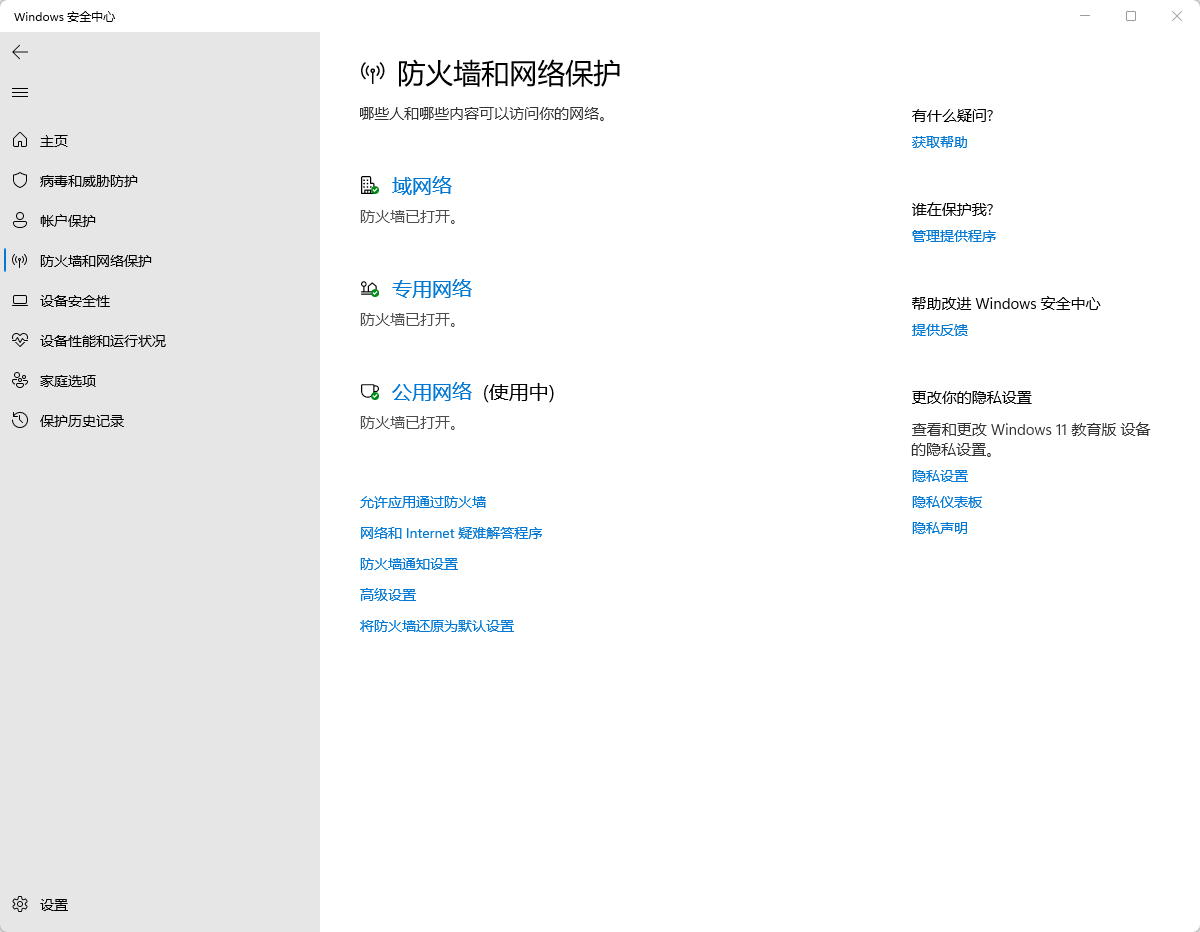


图3 Windows关闭防火墙（找到使用中的网络，将其防火墙关闭）

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Unix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[[4]](#footnote-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** |
| *chat\_client.py* | 客户端程序代码，连接服务端并进行聊天通信（需补全） |
| *chat\_server.py*  *python\_use1.py* | 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  基础python语法示例 |
| *cpp\_p2p/chat\_client.cpp* | Unix版本C++服务端代码，实现一对一并聊天通信（需补全） |
| *cpp\_p2p/chat\_server.cpp* | 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） |
| *cpp\_hub目录* | 【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]](#footnote-5) [[6]](#footnote-6)。

文本

描述已自动生成

图4 交互式窗口中执行代码

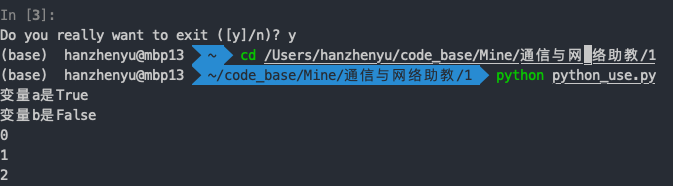


图5 完整执行python程序

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[[7]](#footnote-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。 | |
| 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。 | |
| **s.close()** | 关闭套接字 | |
| **s.getpeername()** | 返回连接套接字的远程地址。返回值通常是元组（ipaddr,port）。 | |
| **s.getsockname()** | 返回套接字自己的地址。通常是一个元组(ipaddr,port) | |
| s.setsockopt(level,optname,value) | 设置给定套接字选项的值。 | |
| s.getsockopt(level,optname[.buflen]) | 返回套接字选项的值。 | |
| **s.settimeout(timeout)** | 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(flag) | 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]](#footnote-8) [[9]](#footnote-9)，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（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clint1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，clint1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。



图6 client程序验证

文本

描述已自动生成

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器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。



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。

**文本

描述已自动生成**

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>

1. 下载提供的虚拟机平台：

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

1. 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。
2. 启动虚拟机后，将进入Ubuntu系统。用户名：cn，密码：12345678
3. 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

1. <https://www.anaconda.com/products/distribution#Downloads> [↑](#footnote-ref-1)
2. <https://cloud.tsinghua.edu.cn/d/6af4682fe0d84bacb092/> [↑](#footnote-ref-2)
3. <https://www.jianshu.com/p/2f3be7781451> [↑](#footnote-ref-3)
4. <https://cloud.tsinghua.edu.cn/f/ab6b3561edfe41a1ac85/> [↑](#footnote-ref-4)
5. <https://www.runoob.com/python3/python3-basic-syntax.html> [↑](#footnote-ref-5)
6. <https://www.liaoxuefeng.com/wiki/1016959663602400/> [↑](#footnote-ref-6)
7. <https://docs.python.org/3/library/socket.html> [↑](#footnote-ref-7)
8. <https://docs.python.org/3/library/multiprocessing.html?highlight=process#module-multiprocessing> [↑](#footnote-ref-8)
9. <https://www.liaoxuefeng.com/wiki/1016959663602400/1017627212385376> [↑](#footnote-ref-9)