深 圳 大 学 实 验 报 告

课程名称：计算机网络

实验项目名称：**Socket 网络编程**

学院：计算机与软件学院

专业：计算机科学与技术

指导教师：谢瑞桃

报告人：王云舒 学号：**2018152044** 班级：计科 **03**

实验时间：**2021/4/26**

实验报告提交时间：**2021/6/20**

教务处制

实验目的：

- 理解 UDP 与 TCP 套接字的区别；
- 掌握 UDP 和 TCP 套接字编程方法；
- 了解简单网络应用的编程思路；
- 了解网络编程相关的一些库。

实验环境：

- 具有 Internet 连接的主机；
- Windows 10 操作系统；
- Python3.8（base）+Anaconda+pycharm2020.3.5。

实验内容：

1. URL 请求程序；
2. 系统时间查询；
3. 网络文件传输；
4. 网络聊天室。

实验要求：参考讲义学习套接字编程的基础知识，了解网络编程的相关库；掌握编写简单网络应用的技能，依照步骤完成实验内容 1——4。

实验步骤：

1 URL 请求程序

要求：利用 Python 的 HTTP 库 **Requests** 实现一个简单的程序，它能够：

1）请求一个网页，并存储为 HTML 文件；

2）计算所请求网页的大小。

1.1 代码思路与编写

由实验的要求以及实验范例，编写的程序首先需要能够请求一个输入的 URL，这一点可以利用 requests 库中的 get 函数实现；接下来需要将其保存为 HTML 文件，这里可以编写一个程序 saveHtml(file_name,file_content)，file_name 负责将请求到的 URL 设置重命名的保存文件名，file_content 即刚才请求到的 URL，利用 Python 的 with open as f 和

f.write 写文件，保存到本地。最后利用 Python 的 os 库，os.path.getsize 函数可以得到本地文件的大小，os.getcwd 函数可以得到当前文件的路径，输出最后的提示结果。

从 Requests 库的官网 requests.readthedocs.io 可以得到关于其函数的一些说明，如图 1.1.1 所示；程序流程、核心代码与完整代码如图 1.1.2 和图 1.1.3 所示。

响应内容

我们可以读取服务器响应的内容。再次考虑 GitHub 时间线：

```
>>> import requests

>>> r = requests.get('https://api.github.com/events')
>>> r.text
'[{"repository":{"open_issues":0,"url":"https://github.com/...
```

请求将自动解码来自服务器的内容。大多数 unicode 字符集都是无缝解码的。

当您发出请求时，Requests 会根据 HTTP 标头对响应的编码进行有根据的猜测。访问时使用请求猜测的文本编码`r.text`。您可以使用以下`r.encoding`属性找出 Requests 正在使用的编码并对其进行更改：

```
>>> r.encoding
'utf-8'
>>> r.encoding = 'ISO-8859-1'
```

图 1.1.1 requests.get 的一些说明

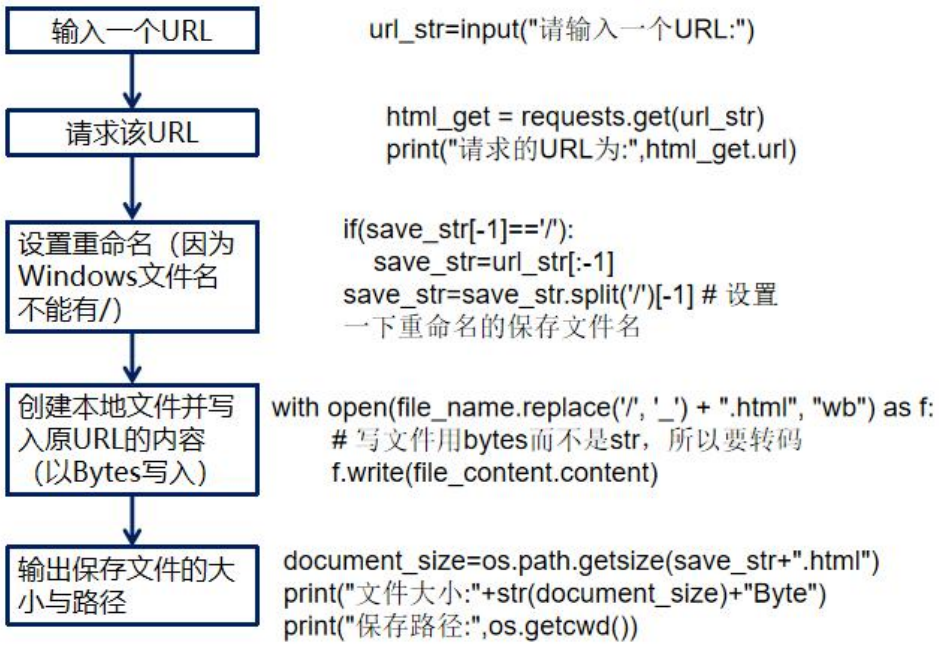


图 1.1.2 程序流程与核心代码

图 1.1.3 完整的程序代码

1.2 程序运行结果

在实验四程序所在的文件夹打开 Powershell，使用 python url_download.py 指令运行程序，请求网页以 baidu 为例，运行结果如图 1.2.1 所示：

图 1.2.1 URL 请求程序的运行结果

输出结果包括了范例中的所有内容，再去本地打开保存的 HTML 文件验证保存成功，如图 1.2.2 所示。可以看到保存的 HTML 成功打开。

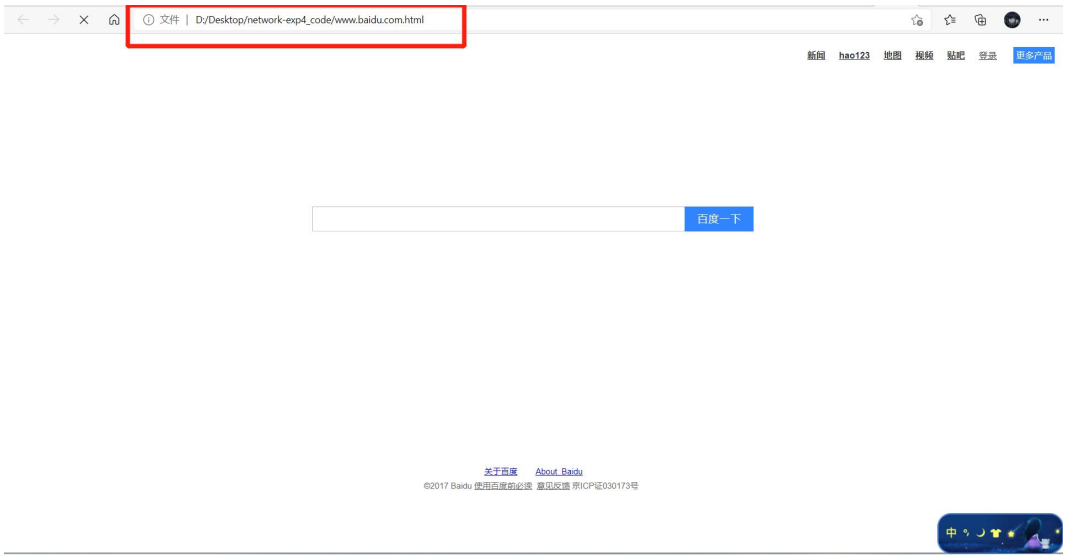


图 1.2.2 file:///D:/Desktop/network-exp4_code/www.baidu.com.html 保存 HTML 可以打开

2 系统时间查询

要求：实现一个基于客户/服务器的系统时间查询程序；传输层使用 TCP，将服务器与客户端的交互过程全部打印出来。

交互过程：

1）客户端向服务器端发送字符串“Time”；

2）服务器端收到该字符串后，返回当前系统时间；

3）客户端向服务器端发送字符串“Exit”；

4）服务器端返回“Bye”，然后结束 TCP 连接。

2.1 代码思路与编写

由实验的要求以及实验范例，编写的程序应该包括两部分，分别是客户端代码与服务器代码。

1）先编写客户端代码，使用 Python 的 socket 库，创建客户端的套接字：

```
clientSocket=socket.socket(socket.AF_INET,socket.SOCK_STREAM)    # 注意参数 socket 的第二个参数是 TCP
```

实现 TCP 连接利用 connect 函数，需要传递两个参数 serverIP 和 serverPort：

```
clientSocket.connect((serverIP, serverPort))
```

本次代码 IP 地址通过手动输入（本机可以 127.0.0.1，远程客户端可以输入服务器 IP），端口号指定为 12000。

```
serverPort=12000
serverIP=input("*Input the serverIP you wanna connect to: ")
print("*Connected to "+serverIP+":12000")
```

当 TCP 连接建立后，使用 While True 保持监听，发送的语句利用 input 输入，套接字 clientSocket.send 函数发送；接收的语句利用 clientSocket.recv(1024)监听。

```
while True:
 sentence = input('Send a request:')
 clientSocket.send(sentence.encode())
 modifiedSentence=clientSocket.recv(1024)
```

modifiedSentence 自动接收来自服务器发送到客户端的内容，根据返回的内容来作出应答；这里额外设置一个 Unknown meaning 作为对其他请求结果作出的应答，即如果发送的内容不是 Time.，服务器会发送会无法识别的语句作为应答。如果发送 Time.，服务器正常情况下会发回当前时间，客户端打印这个内容即可。

```
if(sentence=="Exit."): #Exit.请求结果作出应答
  print('Received a response:{}'.format(modifiedSentence.decode()))
  clientSocket.close()
  break
 elif(sentence=="Unknown meaning."):  #鲁棒性，对其他请求结果作出应答，说明无法识别
  print('Received a response:{}'.format(modifiedSentence.decode()))
 else: #对Time.请求结果作出应答
  print('Received the current system time on the server:{}'.format(modifiedSentence.decode()))
```

2）再编写服务器端的代码，服务器的端口号设为 12000，serverIP 设为 0.0.0.0，这个 IP 可以在客户端是本机的时候连接 127.0.0.1 或者远程的时候连接其 IP。创建服务器套接字并监听的代码：

```
serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)    # 注意参数 socket 的第二个参数是 TCP
serverSocket.bind((serverIP,serverPort))
serverSocket.listen(1)
```

之后按照示例，输出一些相关信息如输出服务器端的 IP 地址、当前主机名、端口号等内容：

```
hostname = socket.gethostname()
IPinfo = socket.gethostbyname_ex(hostname)
LocalIP = IPinfo[2]

print("当前的主机名为:" + hostname + "\n 可供连接的 IP 地址为:"+str(LocalIP))
print("————————————————————————————————————")
print('The server is ready to connect')
print("The server address:("+serverIP+",12000)")
print("————————————————————————————————————")
```

服务器端也使用 while True 保持监听，使用函数 serverSocket.accept()来判断是否有

客户端连接上当前服务器。当有连接时先输出连接客户端的 IP 地址：

```
while True:  # 两层循环 一层连接 一层监听（实际上监听也可以封装成一个函数）
 connectionSocket,addr=serverSocket.accept()
 print("Accepted a new connection.")
 print("The connect address:" + str(connectionSocket.getsockname()))
 print("The client address:" + str(addr))
```

再一层循环作为监听函数，while True 保持监听，如果上面的 connectionSocket 即连接套接字接收到了内容，用 sentence=connectionSocket.recv(1024).decode()接收语句，并对 sentence 进行判断。如果是 Time.，就对请求作出应答，利用 Python 中 time 库的 asctime、localtime 函数返回当前时间，并将其封装成 str 类型，用 connectionSocket.send (responseSentence.encode())发回给客户端；如果是 Exit.，就发回字符串“Bye.”，之后关闭连接，即关闭套接字 connectSocket.close()；如果是其他请求，发送一个字符串“Unknown meaning.”表示无法识别。

```
while True:
 sentence=connectionSocket.recv(1024).decode()
 if(sentence=="Time."): #对 Time.请求作出应答
  print("Received a request:Time.")
  localtime = time.asctime(time.localtime(time.time()))
  responseSentence=str(localtime)
  print("Send a response:"+str(localtime))
  connectionSocket.send(responseSentence.encode())
 elif(sentence=="Exit."): #Exit.请求作出应答
  print("Received a request:Exit.")
  responseSentence = "Bye."
  print("Send a response:" + responseSentence)
  connectionSocket.send(responseSentence.encode())
  connectionSocket.close()
  print("————————————————————————————————————")
  break
 else: #鲁棒性，对其他请求作出应答，说明无法识别
  print("Received an unknown request.")
  responseSentence = "Unknown meaning."
  connectionSocket.send(responseSentence.encode())
```

由于该内容和之后内容代码量较多，实验完整代码全部以附件形式提交。

2.2 程序运行结果

先展示本地客户端与服务器交互，再展示远程客户端与服务器交互。本地客户端时与服务器使用同一台电脑，打开两个终端分别运行 TCP_inquiry_client.py 和 TCP_inquiry_server.py 即可。远程客户端时保持服务器开启，用另一台电脑运行 TCP_inquiry_client.py 文件，发送 Time.与 Exit.。下面展示结果的标注较多，不带标注的

实验截图都保存在实验结果与结论部分。

图 2.2.1 本地客户端下发送 Time.、Exit.等的交互过程

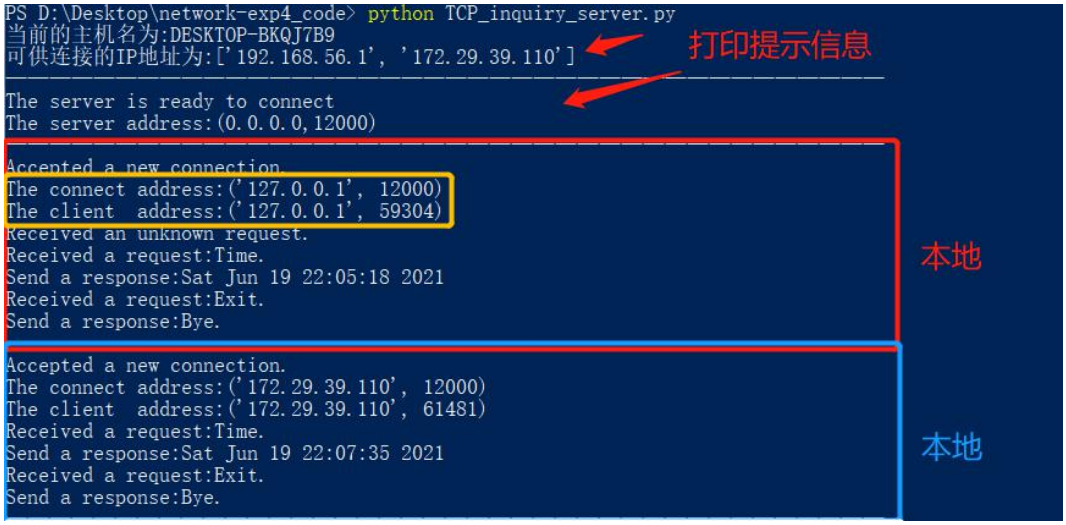


图 2.2.2 服务器与本地客户端的交互过程

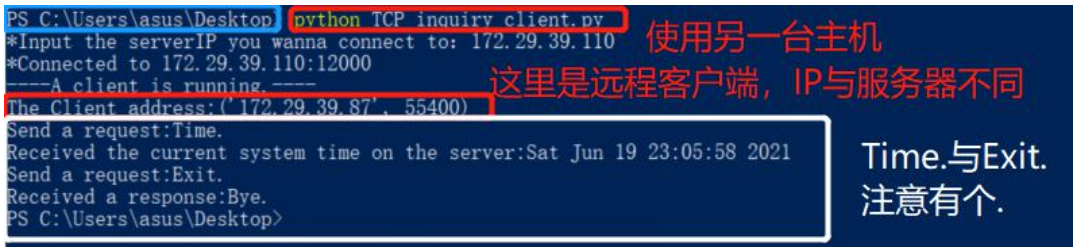


图 2.2.3 远程客户端下发送 Time.、Exit.等的交互过程

图 2.2.4 服务器与远程客户端的交互过程

3 网络文件传输

要求：实现一个基于客户/服务器的网络文件传输程序；传输层使用 TCP，将服务器与客户端的交互过程全部打印出来。

交互过程：

1）客户端从用户输入获得待请求的文件名；

2）客户端向服务器端发送文件名；

3）服务器端收到文件名后，传输文件；

4）客户端接收文件，重命名并存储在硬盘。

3.1 代码思路与编写

由实验的要求以及实验范例，编写的程序应该包括两部分，分别是客户端代码与服务器代码。

对比网络文件传输与系统时间查询，可以看到他们的 TCP 建立连接的步骤完全相同，在系统时间查询中客户端要向服务器发送 Time.和 Exit.，服务器监听接收这些内容后相对应的作出回答，发送回客户端时间或者退出字符。网络文件传输只需要将客户端发送给服务器的内容变成一个文件名，服务器得到这个文件名后，从自己的本地找到该文件并打开，以二进制字节流的方式（即调用 f = open(file_dir, 'rb')、data = f.read(64)）把 data 发送回客户端。客户端得到 data 再保存到自己本地，即可完成文件的传输与保存。比较起来唯一的区别就是 connectionSocket.send 的内容有一些不同，客户端接收到 data 以后还需要再调用 f = open(savefile, 'wb')、f.write(data)保存到自己的硬盘。

TCP_transfer_client 的代码编写与解释如图 3.1.1 所示：

```
1   #exp4-3
2   import socket
3
4   clientSocket=socket.socket(socket.AF_INET,socket.SOCK_STREAM)    # 注意参数 socket的第二个参数是TCP
5
6   serverName='localhost'
7   serverPort=12000
8   serverIP=input("*Input the serverIP you wanna connect to: ")
9   print("*Connected to "+serverIP+":12000")
10  clientSocket.connect((serverIP, serverPort))
11
12  print("----A client is running.----")
13  clientAddress=clientSocket.getsockname()
14  print("The Client address:"+str(clientAddress))
15
16  while True:
17      sentence = input('Input the requested file name:')  # 输入的文件名要在server的filehub存在，注意一定要带后缀
18      filename = sentence
19      clientSocket.send(sentence.encode())
20      file_size=int(clientSocket.recv(1024).decode())
21
22      recv_size = 0  # 已经接收文件的大小
```

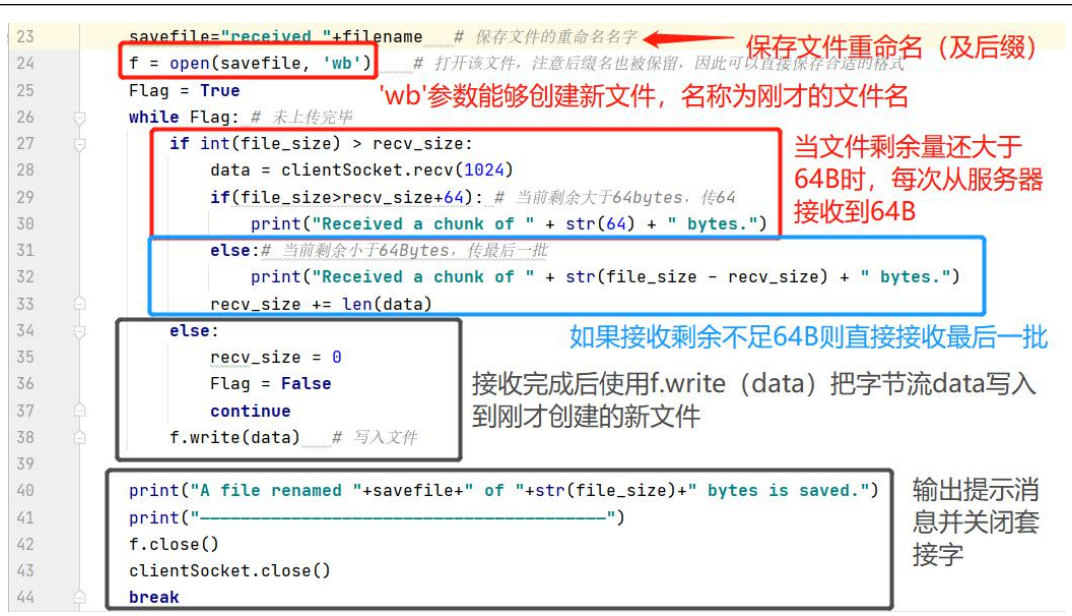


图 3.1.1 TCP_transfer_client 的代码编写与解释

TCP_transfer_server 的代码编写与解释如图 3.1.2 所示：

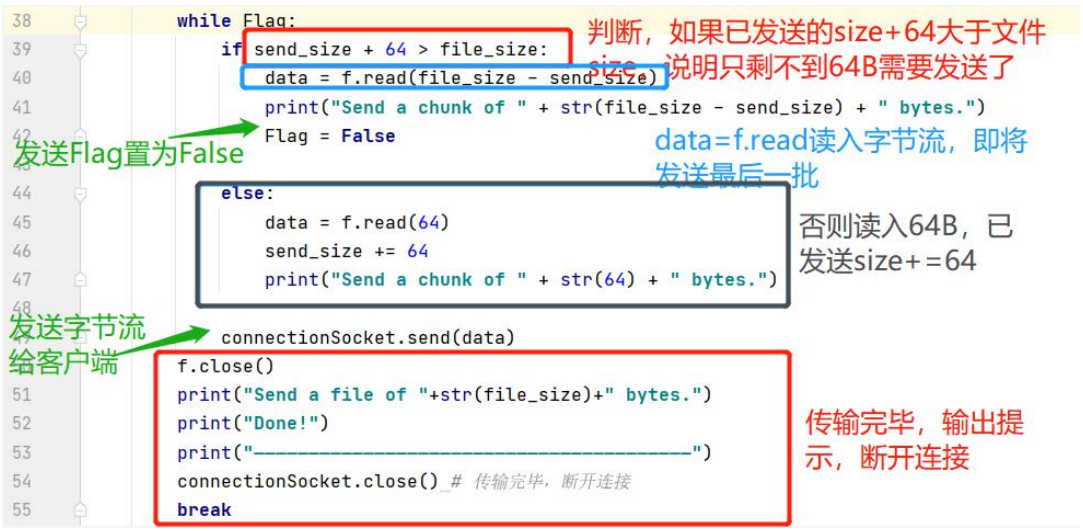


图 3.1.2 TCP_transfer_server 的代码编写与解释

3.2 程序运行结果

程序运行以两个文件为例，在服务器端有一个名为 long.doc 的 word 文件，有 19KB；还有一个名为 short.txt 的文本文档，只有 20B。客户端分别请求这两个文件，请求结束后到客户端的本地去查看这两个文件是否能以正确格式打开，内容是否正确。图 3.2.1、3.2.2、3.2.3 分别展示了客户端请求 long.doc、服务器端接收请求、请求完成后客户端打开本地的 received_long.doc，可以看到能够成功打开并得到正确的内容；

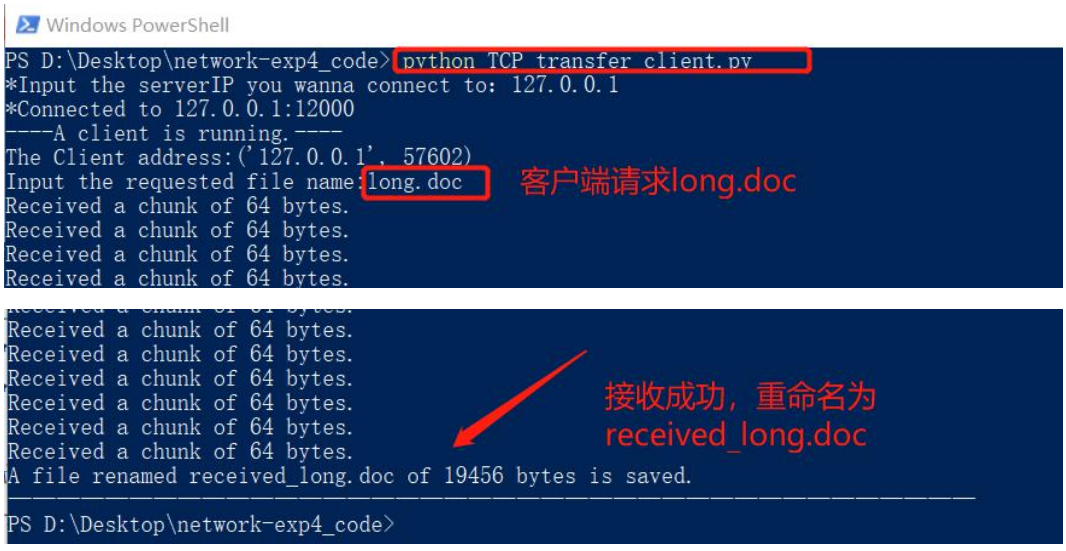


图 3.2.1 客户端请求 long.doc

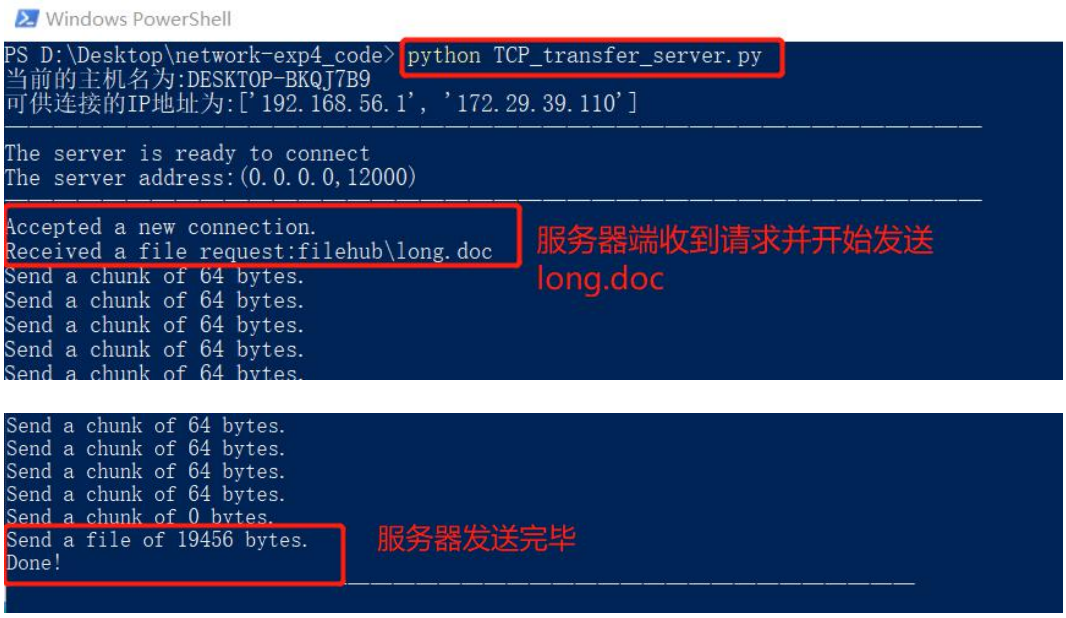


图 3.2.2 服务器向客户端发送文件的字节流

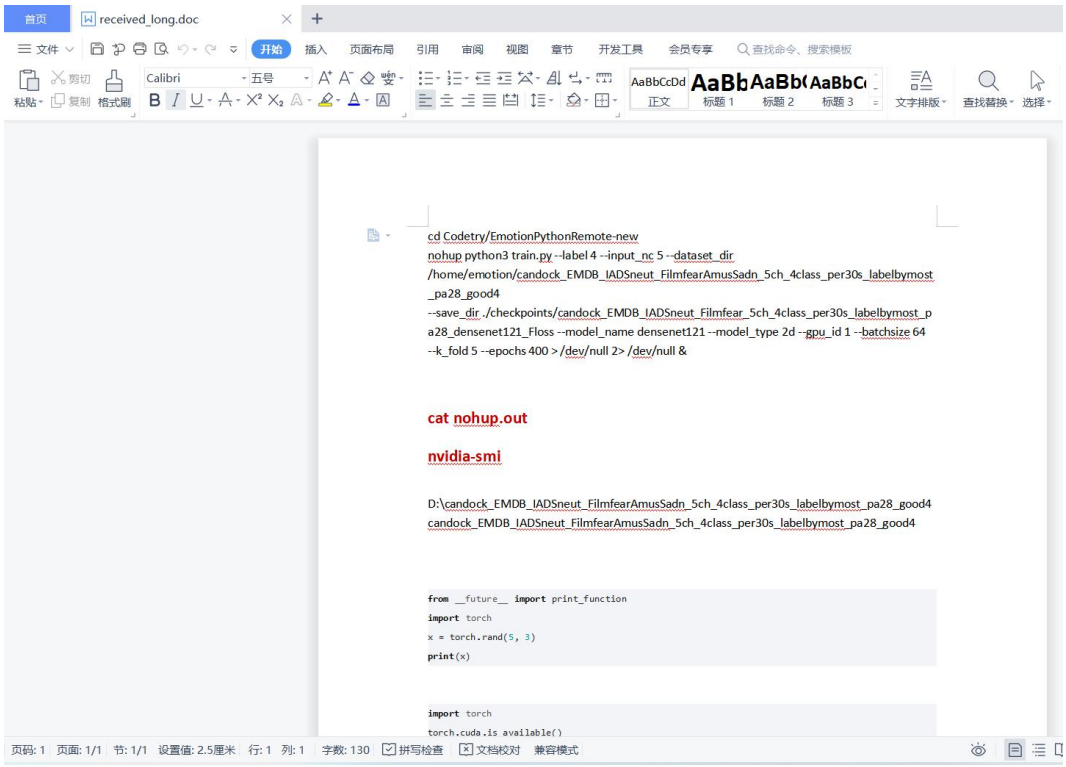


图 3.2.3 客户端打开保存在本地硬盘的重命名文件，可以打开，内容一致

图 3.2.4、3.2.5、3.2.6 分别展示了客户端请求 short.txt、服务器端接收请求、请求完成后客户端打开本地的 received_short.txt，可以看到也能够成功打开并得到正确的内容。说明字节流传输方式以及 f.open、f.write、f.read 组合可以得到各种形式后缀文件的正确格式。

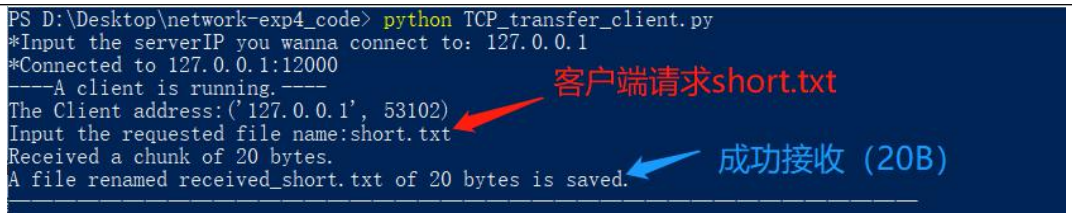


图 3.2.4 客户端请求 short.txt

图 3.2.5 服务器向客户端发送文件的字节流

图 3.2.6 客户端打开保存在本地硬盘的重命名文件，可以打开，内容一致

4 网络聊天室

要求：实现一个基于客户/服务器的网络聊天程序；传输层使用 UDP，能够实现多个用户的群聊，客户端打印聊天信息，服务器打印系统信息。

本次实验实现了一个 UI 界面，客户端部分聊天信息全部体现在 UI 界面中，服务器的系统信息仍打印在终端。

4.1 代码思路与编写

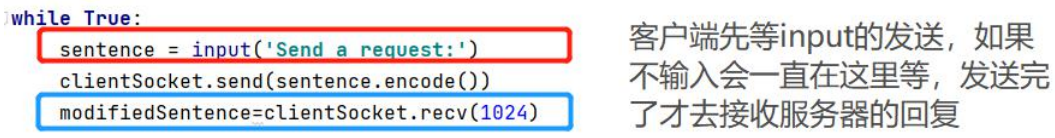
对于网络聊天室，首先要了解群聊的概念：一个用户发消息，所有用户都能收到。也就是**一个客户端把他想要发送的内容发给服务器，服务器接收到之后要把收到的消息转发给所有客户端。**

服务器为了知道聊天室里有哪些用户就需要维护一个聊天室用户列表。新用户到达时加进来，旧用户离开时删除。这个列表中的用户信息即其名称与 IP+端口。

因为这次使用 UDP 实现，而 UDP 是一种无连接的传输层协议，因此旧用户离开不能像 TCP 一样反映出来，需要应用层设计实现，例如退出时**客户端向服务器发送特殊字符串 exit，服务器收到消息时通过查看消息内容判断客户端是不是要终止聊天。**

在代码的实现上，这一问和 2、3 问有一个明显的区别，即客户端要从键盘获取用户输入并把消息发送给服务器，而且还要同时接受并显示服务器发送的其他用户的聊天信息。这和之前客户端向服务器发送一条信息，服务器接收一条信息并发回，客户端接

收有所不同：



2、3 问都是客户端向服务器发出请求，服务器做回复即可，写在循环中 input 在前

现在即使当前客户端不发送内容给服务器，服务器可能会主动的把其他客户端的消息转发给当前客户端。因此这里客户端要使用两个线程，利用 Python 的 threading 库，一个线程负责接收并显示消息，另一个线程负责获取输入和发送消息。

1）代码编写上先编写 UDP_chat_server.py，首先编写 server 类，需要包括其基本 IP、端口信息以及启动时调用的函数：

```
from socket import *
import time

# server 类，这个 py 文件需要先被运行才能执行 client
class Server():
 user_name = {}  # dict 用户名:ip 地址
 user_ip = {}  # dict ip 地址:用户名
 def __init__(self,post):
  self.ServerPort = post
  self.udp_socket = socket(AF_INET, SOCK_DGRAM)
  self.udp_socket.bind(self.ServerPort)

 # 运行
 def start(self):
  print(self.getTime(),'服务端已启动')
  self.recv_msg()
```

recv_msg()函数是 server 的主体，负责接收信息并进行发回操作。接收信息用到的语句：

```
recv_data,dest_ip = self.udp_socket.recvfrom(1024)
```

recv_data 是接收到的消息，dest_ip 是发送消息的客户端的 IP。之后用 print 语句输出 dest_ip 作为提示，并用 str.split()切割 recv_data，如果得到的是客户端发来的 exit 就会作为退出提示，如果包含了“-n”**（设定好的提示符，如果客户端向服务器第一次发送内容，会携带一个-n 标志）**就视作新用户注册，把该 IP 加入 server 中的 user_ip 中；其他情况都当做消息需要群发。

```
print(self.getTime(),"Receive from",dest_ip,recv_data)#打印提示信息
info_list = str(recv_data.decode("gb2312")).split() # 切割命令判断
```

详细的代码内容：

图 4.1.1 server 的详细代码内容与解释

2）client 类因为判断较少，代码相对更为简单，其中多线程需要从 Python threading 中 import Thread 库，函数使用的方法：

```
1.    self.thread_rece = Thread(target=self.recv_msg)#接收线程
```

```
    self.thread_send = Thread(target=self.send_msg)#发送线程

    self.thread_rece.start()
    self.thread_send.start()
    self.thread_rece.join()
    self.thread_send.join()
```

创建 client 类以后，两个成员 thread_rece 和 thread_send 调用 Thread（target=self.XXX），XXX 是编写的两个函数 recv_msg 和 send_msg，相当于为接收信息和发送信息两个函数各开一个线程。之后用 start 启动即可。

```
# client类，这个py文件也可以单独执行，其不带可视化界面
class Client():
    def __init__(self, name, client_post, server_post):
        # 初始化
        self.ClientPort = client_post
        self.ServerPost = server_post
        self.udp_socket = socket(AF_INET, SOCK_DGRAM)
        self.userName = name
        self.thread_rece = Thread(target=self.recv_msg)#接收线程
        self.thread_send = Thread(target=self.send_msg)#发送线程

    # 运行
    def start(self):
        self.udp_socket.bind(self.ClientPort)
        data_info = self.userName + ' -n'
        self.udp_socket.sendto(data_info.encode('gb2312'), self.ServerPost)#告诉服务器有新的用户加入了
        print(self.getTime(), '系统：您已加入聊天')
        self.thread_rece.start()
        self.thread_send.start()
        self.thread_rece.join()
        self.thread_send.join()
```

设置端口、创建套接字

注意参数和TCP不同

双线程

加入提醒

图 4.1.2 client 类的创建与多线程用法

之后编写接收信息函数 recv_msg 与发送信息函数 send_msg。发送代码核心：

```
    self.udp_socket.sendto(data_info.encode('gb2312'), self.ServerPost)
```

接收代码核心：

```
    recv_data, dest_ip = self.udp_socket.recvfrom(1024)
```

```
    # 接收信息
    def recv_msg(self):
        while True:
            try:
                recv_data, dest_ip = self.udp_socket.recvfrom(1024)  # 1024 表示接收最大字节  防止内存溢出

                if recv_data.decode("gb2312") == 'exit' and dest_ip == self.ServerPost:
                    print(self.getTime(), '客户端已退出')
                    break
                print(recv_data.decode("gb2312"))
            except:
                self.udp_socket.close()#维护可能的异常情况
                break
```

接收信息和打印，exit退出并break

```
41
42      # 发送信息
43      def send_msg(self):
44          while True:
45              data_info = input()
46              self.udp_socket.sendto(data_info.encode('gb2312'), self.ServerPost)
47              if data_info == 'exit':
48                  break
```

发送信息，发exit退出

图 4.1.3 client 的接收函数与发送函数

```
50      # 返回时间
51      def getTime(self):
52          return '[' + time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()) + ']'
53
54      # 发送已经不需要再输入的已有消息
55      def send_message(self, message):
56          self.udp_socket.sendto(message.encode('gb2312'), self.ServerPost)
57
58
59  if __name__ == '__main__':
60      name = input("Please input your name:")
61      port = int(input("Please input the port:"))
62      client = Client(name, ('127.0.0.1', port), ('127.0.0.1', 12000))
63      client.start()
```

发送不需要input的信息（之后编写GUI时可以调用该函数提示服务器有新加入的客户端）

主函数、提示与端口设置

图 4.1.4 client 的一些其他函数以及主函数，这里默认客户端与服务器都在本机了

到此客户端与服务器的代码都已经编写完毕，程序已经可以运行。之后再编写一个简易的 GUI，它可以增加一些提示颜色以及对话框发送。编写的 UDP_chat_GUI.py 作为客户端的主函数，从刚才 UDP_chat_client.py import 来 client 类，这里的 GUI 就只用编写一个框架，然后完全使用刚才编写的函数。其主窗口界面类 MainPanel：

```
8   # 主窗口界面类，这个py文件可以单独执行，其带可视化界面
9   class MainPanel:
10      def __init__(self, username, send_func, close_callback):
11          print("初始化主界面\n")
12          self.username = username
13          self.message_text = None
14          self.send_text = None
15          self.send_func = send_func
16          self.close_callback = close_callback
17          self.main_frame = None
18          self.thread_rece = Thread(target=self.recv_msg)  # 额外创建的接收线程
19
20      # 显示主窗口
21      def show(self):
22          global main_frame
23          main_frame = Tk()
24          main_frame.title("Network-4 王云舒-2018152044")
25          main_frame.configure(background="#333333")
26          main_frame.protocol("WM_DELETE_WINDOW", self.close_callback) # 设置窗口关闭按钮回调，用于退出时关闭socket连接
27          width = 975
28          height = 600
```

背景图片的载入：

```
1.  original1 = Image.open(r'image/snow1.jpg')
2.  resized1 = original1.resize((860, 80), Image.ANTIALIAS)
3.  img1 = ImageTk.PhotoImage(resized1)
4.  label_img1 = Label(main_frame, image=img1)
5.  label_img1.grid(row=0,column=0,sticky=W+E+N+S)
```

设置位置与大小：

```
    screen_width = main_frame.winfo_screenwidth()
    screen_height = main_frame.winfo_screenheight()
    gm_str = "%dx%d+%d+%d" % (width, height, (screen_width - width) / 2,
                                      (screen_height - 1.2 * height) / 2)
    main_frame.geometry(gm_str)
    main_frame.minsize(width, height)
```

滚动条：

```
    msg_sc_bar = Scrollbar(main_frame)
    msg_sc_bar.grid(row=1, column=0, sticky=E + N + S, padx=(0, 10))
    self.message_text = Text(main_frame, bg="Snow", height=1,font=("consolas"
    , 14),highlightcolor="Snow", highlightthickness=1)
```

接收消息窗口：

```
    self.message_text.config(state=DISABLED)
    self.message_text.tag_configure('redcolor', foreground='red')
    self.message_text.tag_configure('greencolor', foreground='green')
    self.message_text.tag_configure('bluecolor', foreground='blue')
    self.message_text.grid(row=1, column=0, sticky=W + E + N + S)
    msg_sc_bar["command"] = self.message_text.yview
    self.message_text["yscrollcommand"] = msg_sc_bar.set
```

发送消息窗口：

```
    send_sc_bar = Scrollbar(main_frame)
    send_sc_bar.grid(row=2, column=0, sticky=E + N + S, padx=(0, 10), pady=10
    )
    self.send_text = Text(main_frame, bg="Snow", height=9, highlightcolor="Sn
    ow",font=("consolas", 14),highlightbackground="#444444", highlightthickness
    =3)
    self.send_text.see(END)
    self.send_text.grid(row=2, column=0, sticky=W + E + N + S, pady=10)
    send_sc_bar["command"] = self.send_text.yview
    self.send_text["yscrollcommand"] = send_sc_bar.set
```

发送按钮调用发送函数，主界面使用 mainloop 维持：

```
    Button(main_frame, text="发送", bg="Ivory", font=("黑体
    ", 14), fg="DimGray", command=self.send_func) \.grid(row=3, column=0, pady=
    5, padx=28, sticky=E, ipady=2, ipadx=13)
    self.main_frame = main_frame
    main_frame.mainloop()
```

发送按钮按下后，判断文本窗口中已有信息是否为空，为空则提示空消息拒绝发送，不为空则去调用 client 的 send_message，把内容 content 传给 send_message 让其直接发送

即可。

```
# 发送文本窗口中已有的信息给服务器
def send_message():
    print("send message:")
    content = main_window.get_send_text()
    if content == "" or content == "\n":
        print("空消息，拒绝发送")
        return
    print(content)
    # 清空输入框
    main_window.clear_send_text()
    client.send_message(content)
```

初始化函数与主函数：

```
def start():
    global client
    global name
    name=input("Please input your name:")
    port=int(input("Please input the port:"))
    client = Client(name, ('127.0.0.1', port), ('127.0.0.1', 12000))
    client.udp_socket.bind(client.ClientPort)
    data_info = client.userName + ' -n'
    client.udp_socket.sendto(data_info.encode('gb2312'), client.ServerPost)
    print(client.getTime(), '系统：您已加入聊天')

    global main_window
    main_window = MainPanel(name, send_message, close_main_window)
    main_window.thread_rece.start()
    main_window.show()
    main_window.thread_rece.join()

# main函数
if __name__ == "__main__":
    start()
```

初始化名称、IP与端口，构造client类

向服务器端发送自己加入

创建主窗口以及多线程函数使用

图 4.1.5 带 GUI 的初始化函数与主函数

4.2 程序运行结果

程序运行结果主要展示带界面的客户端状态与服务器打印的内容。实际上不带界面的 UDP_chat_client.py 也可以单独运行，也能够给出各种提示信息。客户端输入的聊天内容可以为多种语言，支持英文、中文、日文等。

PS D:\Desktop\network-exp4_code> **python UDP_chat_server.py**

PS D:\Desktop\network-exp4_code> **python UDP_chat_GUI.py**

Please input your name:**cat**

Please input the port:**11000**

PS D:\Desktop\network-exp4_code> **python UDP_chat_GUI.py**

Please input your name:**dog**

Please input the port:**13000**

PS D:\Desktop\network-exp4_code> **python UDP_chat_GUI.py**

Please input your name:**pig**

Please input the port:**15000**

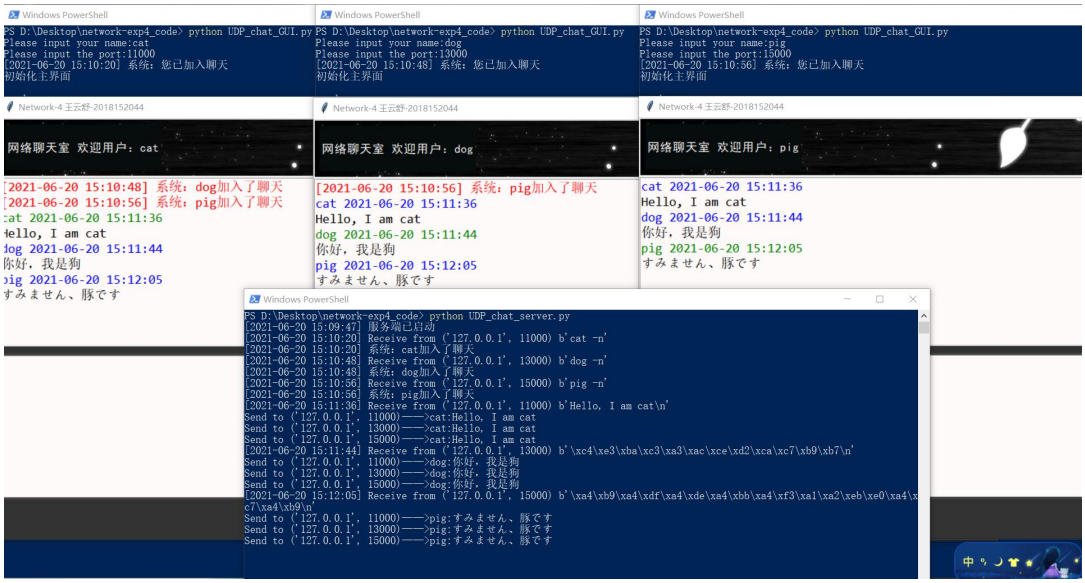


图 4.2.1 三个终端打开客户端，一个终端打开服务器，聊天室的外观总览

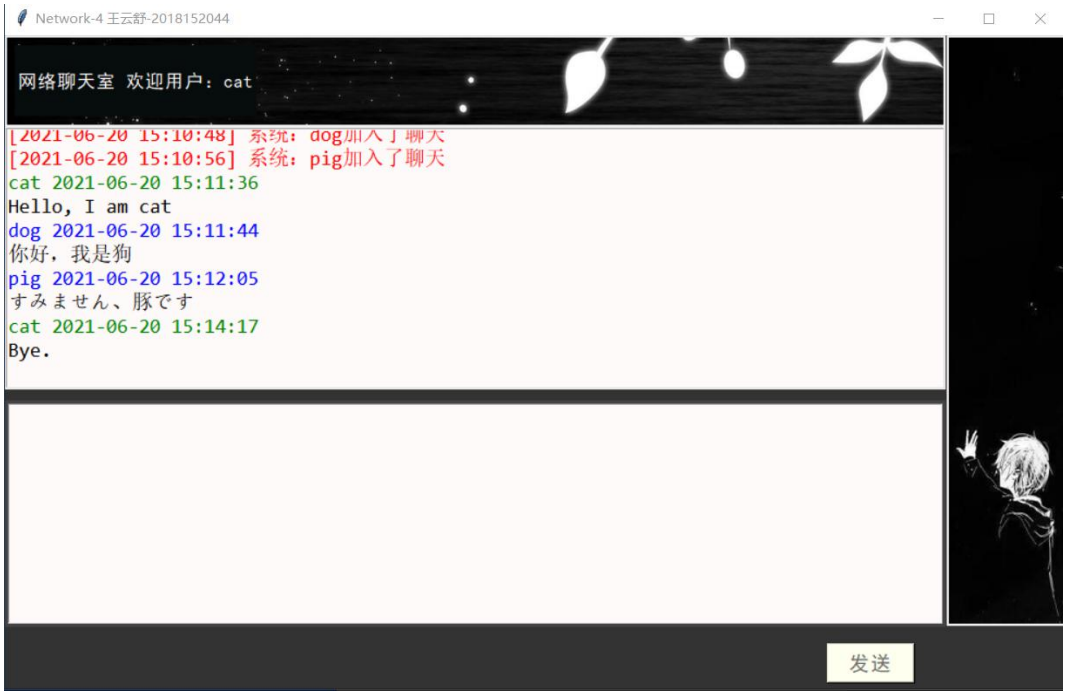


图 4.2.2 聊天室-客户端-cat 端口：11000

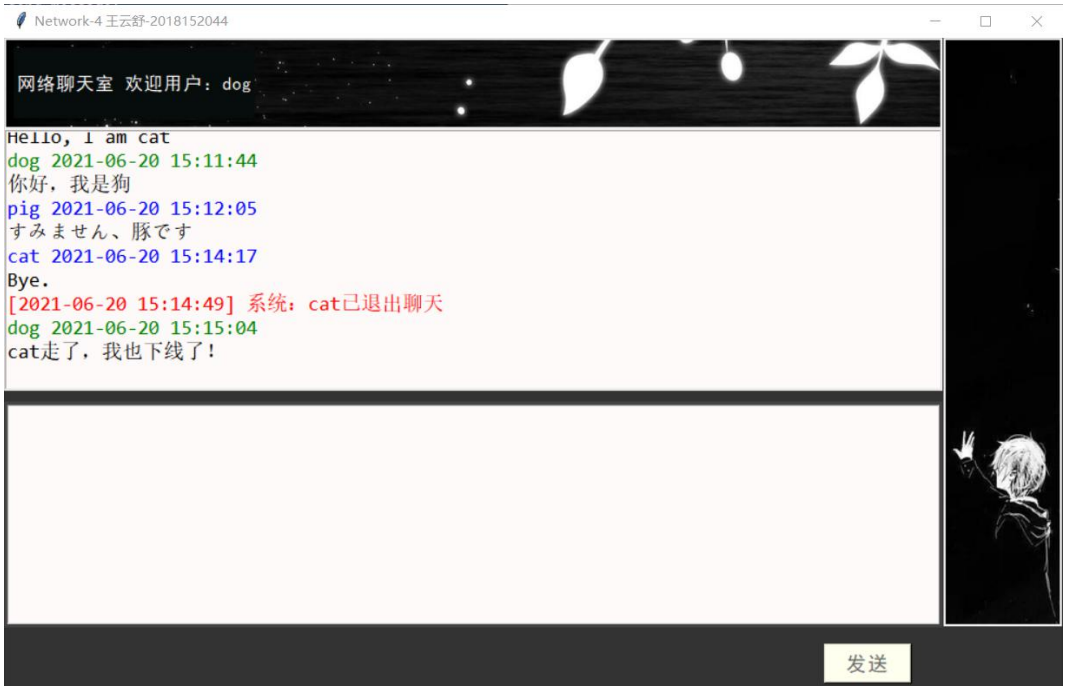


图 4.2.3 聊天室-客户端-dog 端口：13000

图 4.2.4 聊天室-客户端-pig 端口：15000

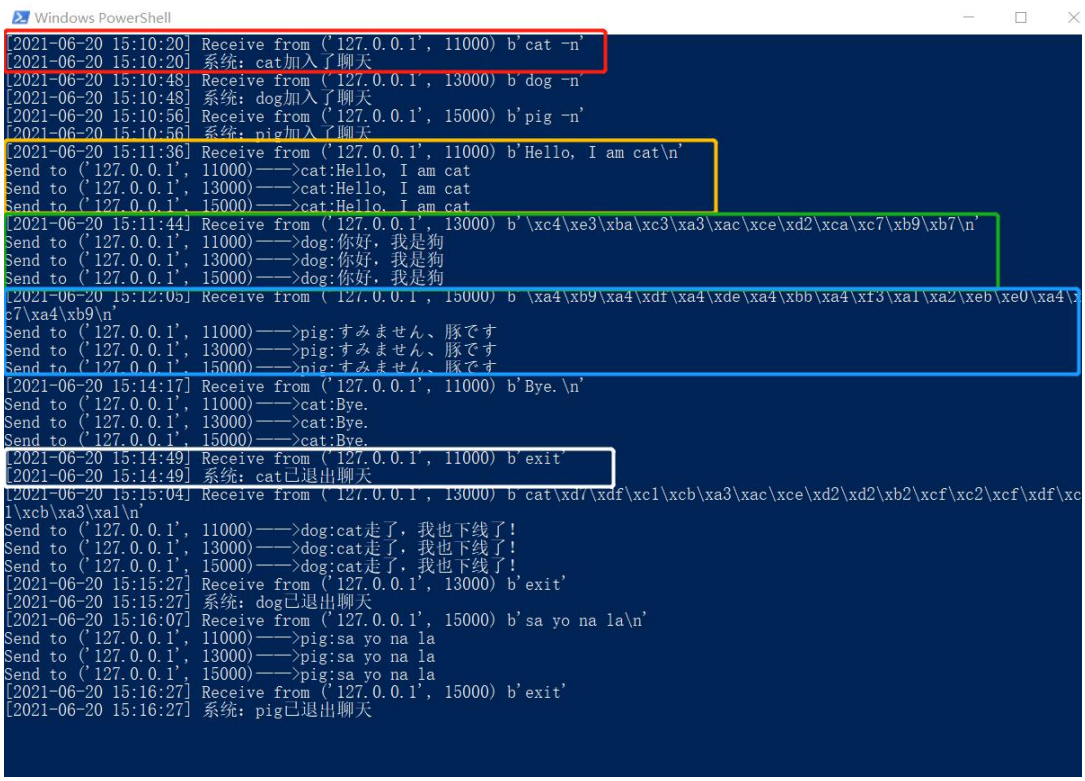


图 4.2.5 聊天室-服务器端

到此实验部分结束。

实验结果：

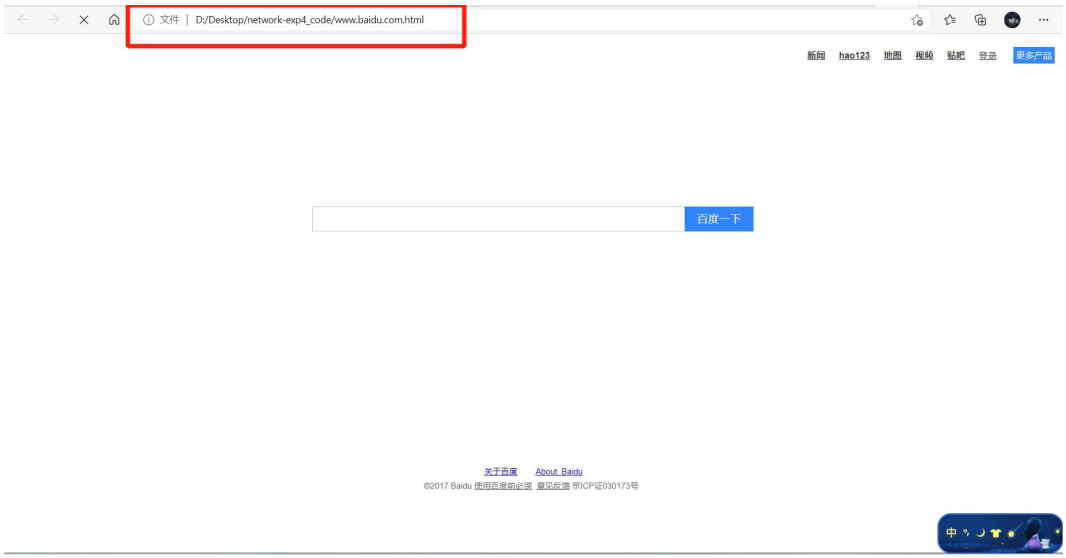
实验结果与分析、结论：

本次实验的所有实验结果在实验内容中都有展示以及详细的标注，这里再展示不带标注的结果图，可以更直接的看到每一步的实验结果。

1）URL 请求程序运行结果：

python url_download.py

URL 请求程序的运行结果



file:///D:/Desktop/network-exp4_code/www.baidu.com.html 保存 HTML 可以打开

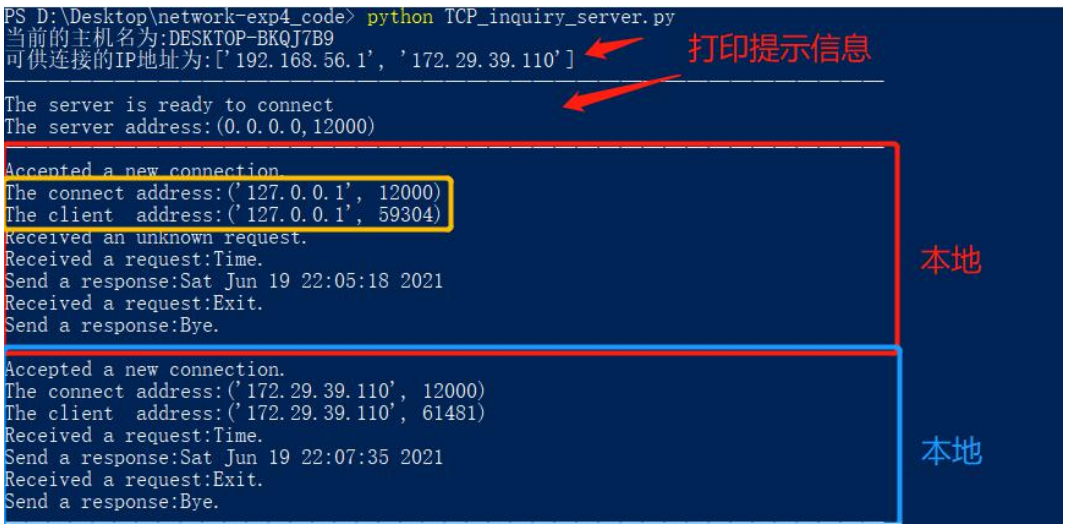
2）系统时间查询运行结果：

python TCP_inquiry_server.py

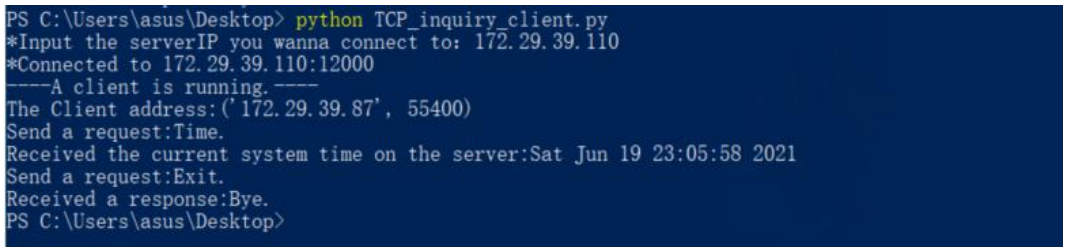
python TCP_inquiry_client.py

```
PS D:\Desktop\network-exp4_code> python TCP_inquiry_client.py
*Input the serverIP you wanna connect to: 127.0.0.1
*Connected to 127.0.0.1:12000
----A client is running.----
The Client address:('127.0.0.1', 59304)
Send a request:Hello?
Received the current system time on the server:Unknown meaning.
Send a request:Time.
Received the current system time on the server:Sat Jun 19 22:05:18 2021
Send a request:Exit.
Received a response:Bye.
PS D:\Desktop\network-exp4_code> python TCP_inquiry_client.py
*Input the serverIP you wanna connect to: 172.29.39.110
*Connected to 172.29.39.110:12000
----A client is running.----
The Client address:('172.29.39.110', 61481)
Send a request:Time.
Received the current system time on the server:Sat Jun 19 22:07:35 2021
Send a request:Exit.
Received a response:Bye.
PS D:\Desktop\network-exp4_code>
```

本地客户端下发送 Time.、Exit.等的交互过程



服务器与本地客户端的交互过程



远程客户端下发送 Time.、Exit.等的交互过程

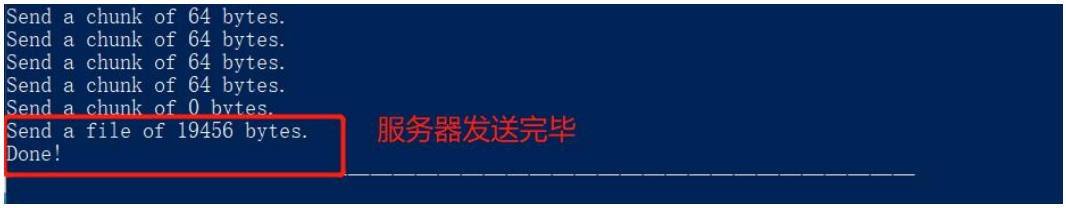
服务器与远程客户端的交互过程

3）网络文件传输运行结果：

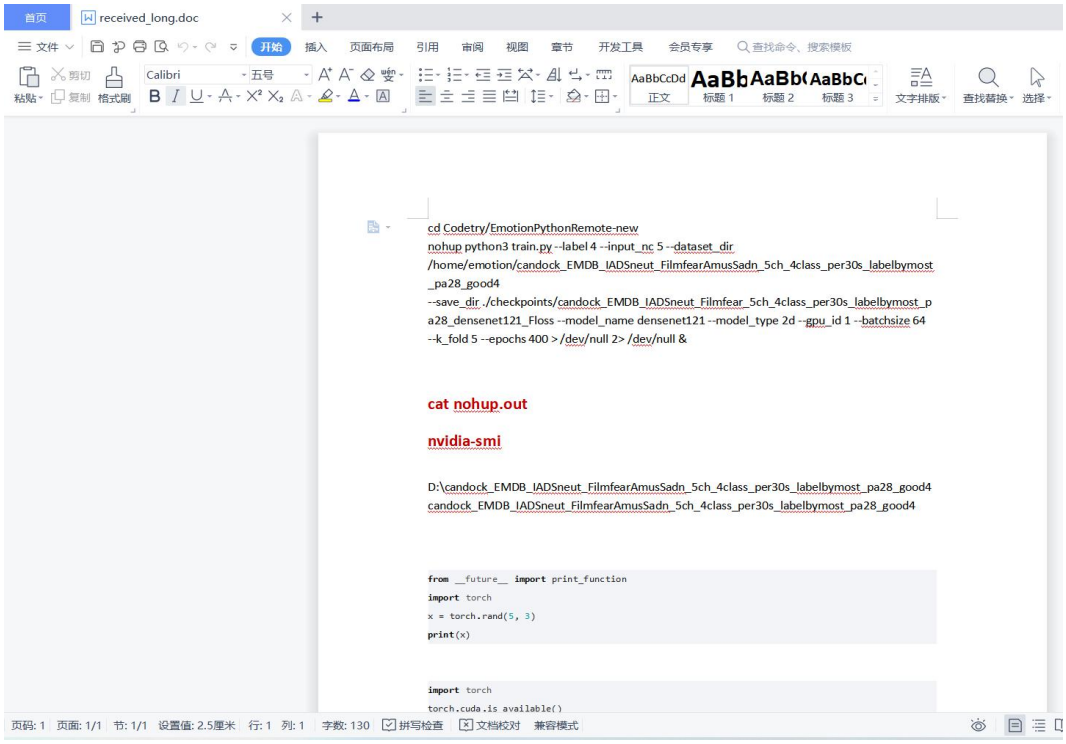
python TCP_transfer_server.py

python TCP_transfer_client.py

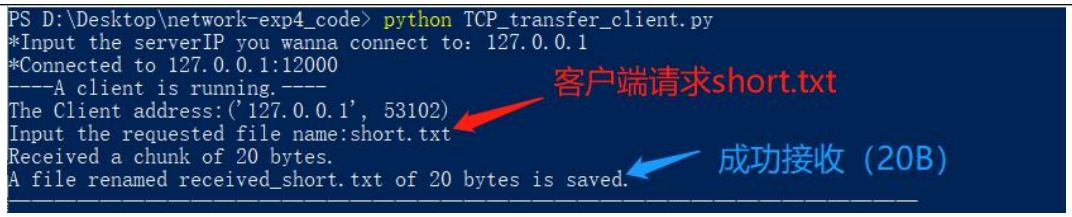
客户端请求 long.doc



服务器向客户端发送文件的字节流



客户端打开保存在本地硬盘的重命名文件，可以打开，内容一致



客户端请求 short.txt

服务器向客户端发送文件的字节流

客户端打开保存在本地硬盘的重命名文件，可以打开，内容一致

4）网络聊天室运行结果：

PS D:\Desktop\network-exp4_code> **python UDP_chat_server.py**

PS D:\Desktop\network-exp4_code> **python UDP_chat_GUI.py**

Please input your name:**cat**

Please input the port:**11000**

PS D:\Desktop\network-exp4_code> **python UDP_chat_GUI.py**

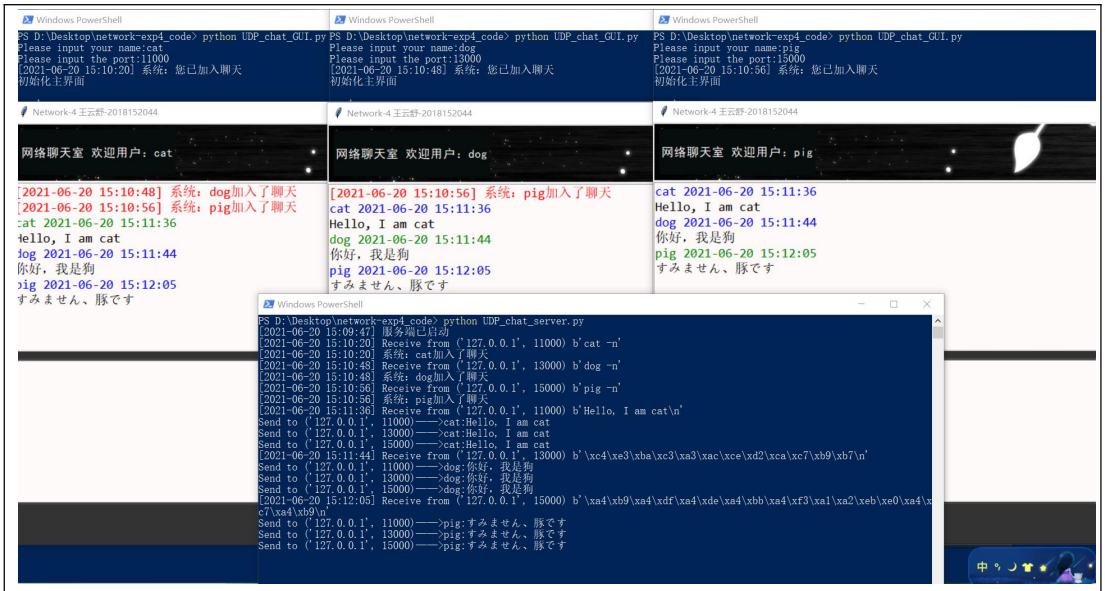
Please input your name:**dog**

Please input the port:**13000**

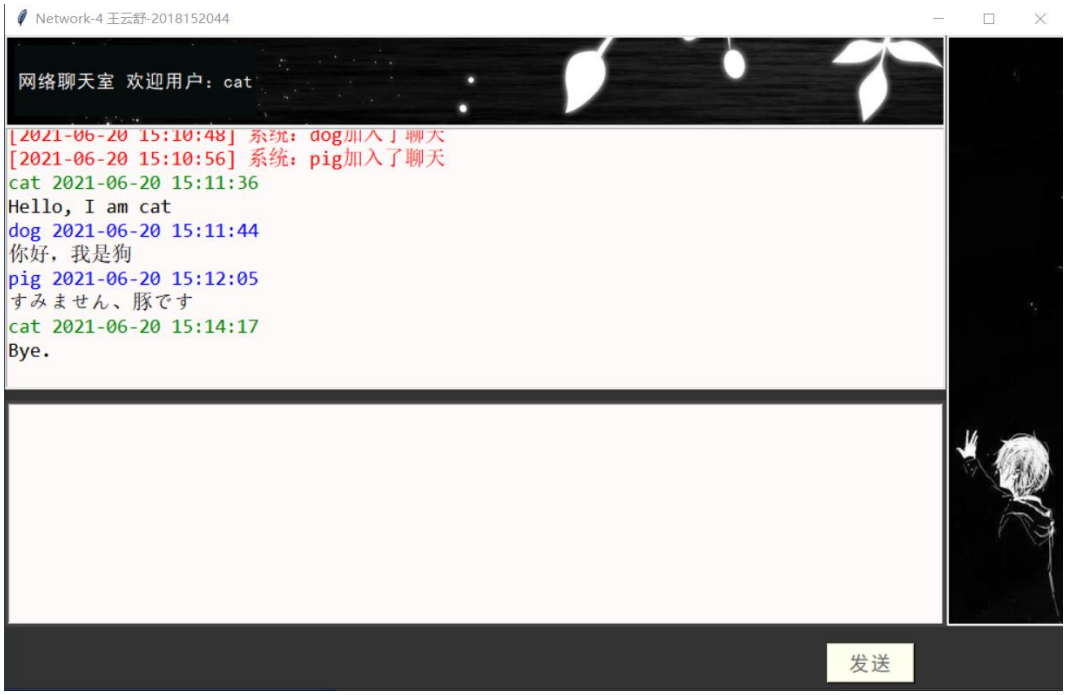
PS D:\Desktop\network-exp4_code> **python UDP_chat_GUI.py**

Please input your name:**pig**

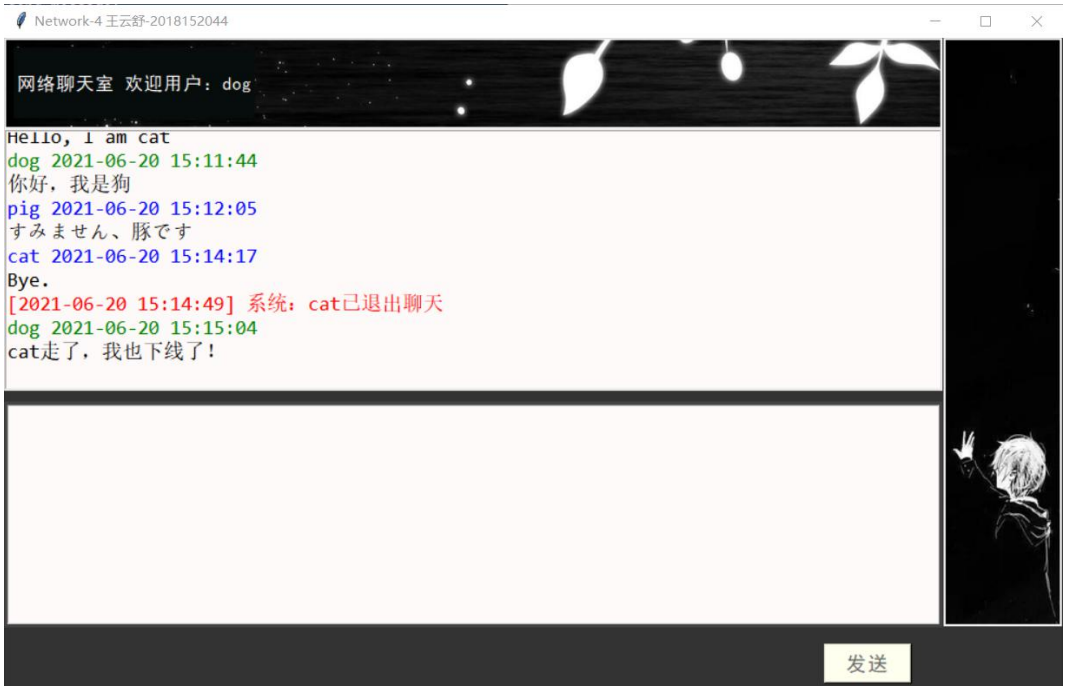
Please input the port:**15000**



三个终端打开客户端，一个终端打开服务器，聊天室的外观总览

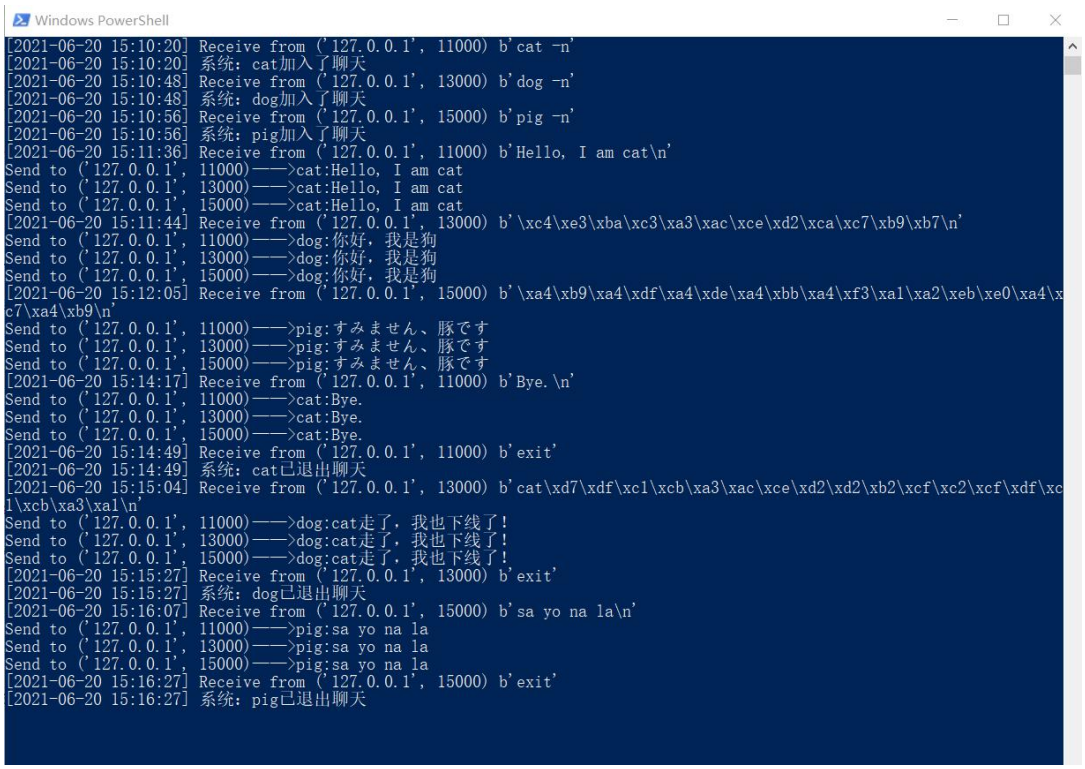


聊天室-客户端-cat 端口：11000



聊天室-客户端-dog 端口：13000

聊天室-客户端-pig 端口：15000



聊天室-服务器

对于实验的分析：

本次实验涉及到了非常多关于 TCP 连接与 UDP 的函数，以及套接字的使用。

socket(AF_INET, SOCK_DGRAM)和 socket(socket.AF_INET,socket.SOCK_STREAM)分别对应 UDP 和 TCP 套接字的创建；send 和 recv 等负责发送与接收。注意 TCP 需要建立连接，有专门对应的函数 clientSocket.connect((serverName, serverPort)) 和 connectionSocket, addr = serverSocket.accept()；当客户端请求时，在客户与服务器之间创建一个 TCP 连接。connect() 方法的参数是这条连接中服务器端的地址。这行代码执行完后，执行三次握手，并在客户和服务器之间创建起一条 TCP 连接。serverSocket 调用 accept() 方法，这在服务器中创建了一个称为 connectionSocket 的新套接字，由这个特定的客户专用。客户和服务器则完成了握手，在客户的 clientSocket 和服务器的 serverSocket 之间创建了一个 TCP 连接。借助于创建的 TCP 连接，客户与服务器现在能够通过该连接相互发送字节。使用 TCP，从一侧发送的所有字节不仅确保到达另一侧，而且确保按序到达。这个 connectionSocket 就对应了实验中的 connect address：在实验第 2、3 步的三个 IP，serverIP 0.0.0.0，connectaddress 会根据连接时 client 的请求，如果 client 使用 127.0.0.1，connectaddress 就是 127.0.0.1；client 请求实际 IP（172.29.39.110），connectaddress 就是 172.29.39.110。

总结客户端与服务器的交互流程：

服务器：1）创建套接字，即实例化。server = socket.socket()

2）绑定地址，且地址是一个元组，里面包括 ip 和端口，为自己创建了一个地址，用于客户端的连接。server.bind(('127.0.0.1',12000))

3）开始监听，此时的套接字 server 才被真正叫做监听套接字，自此之前，客户端是无法连接过来的。代码中的 1 表示最大能同时连接到客户端的数量。server.listen(1)

4）收到连接请求就建立与客户端连接，返回结果由两个变量接收，第一个变量是对等连接套接字，第二个是客户端的地址（ip 和端口）a,b = server.accept()。

5）利用对等连接套接字开启接收信息状态。若接到空值，表示客户端已主动断开连接。这里也会产生一次阻塞，客户端是无法发送空值的。代码的 1024 表示可以接收的最大字节数。a.recv(1024)

6）信息传递讲究一收一发，一发一收。若收到信息，应给客户端一个回复。这里要注意的是信息的传递是以字节的形式。a.send(date)

7）若收到空值，最后一步是断开连接。a.close()

客户端：1）创建套接字，即实例化生成客户端套接字。client = socket.socket()

2）向服务端发送连接请求，连接成功后，原本的客户端套接字实际上就变成了对等连接套接字。代码中的 ip 和端口是服务端的 ip 和端口。client.connect(('127.0.0.1',XXXX))

3）向服务端发送信息。client.send(message)

4）向服务端接收信息，这里会发生一次阻塞。client.recv(1024)

5）主动断开与服务端的连接，这时客户端会发送一个空值。client.close()

本次实验的代码全部作为附件形式上传。

实验小结：

本次实验是一个 Socket 网络编程实验，通过这次实验，我理解了 UDP 与 TCP 套接字的区别，掌握 UDP 和 TCP 套接字编程方法；了解了简单网络应用的编程思路以及网络编程相关的库。在实验中，我完成了 URL 请求程序、系统时间查询、网络文件传输和网络聊天室，并且设计了一个简易的界面。

实验中 Python 在 socket 库的调用时经常会出现一些问题，import socket 和 from socket import *有所区别，可能因为一些函数和 python 内置函数重名的原因，在有的时候不能 import *，调用时只能用 socket.socket()。在最后网络聊天室的实现中，为了解决所有出现的 bug，例如服务器异常关闭、界面显示不全、发送按钮无效等问题，我也修改了很长时间的代码。此外，第 2 步远程客户端需要注意关闭服务器电脑的防火墙。

实验四的代码部分虽然我完成的比较早，但是报告是在学期末才撰写，学完课程后对实验的内容也有了更好的了解。总之，本次实验使我受益匪浅。

指导教师批阅意见：

成绩评定：

指导教师签字：

年　　月　　日

备注：