

## 实验报告

### 要求：

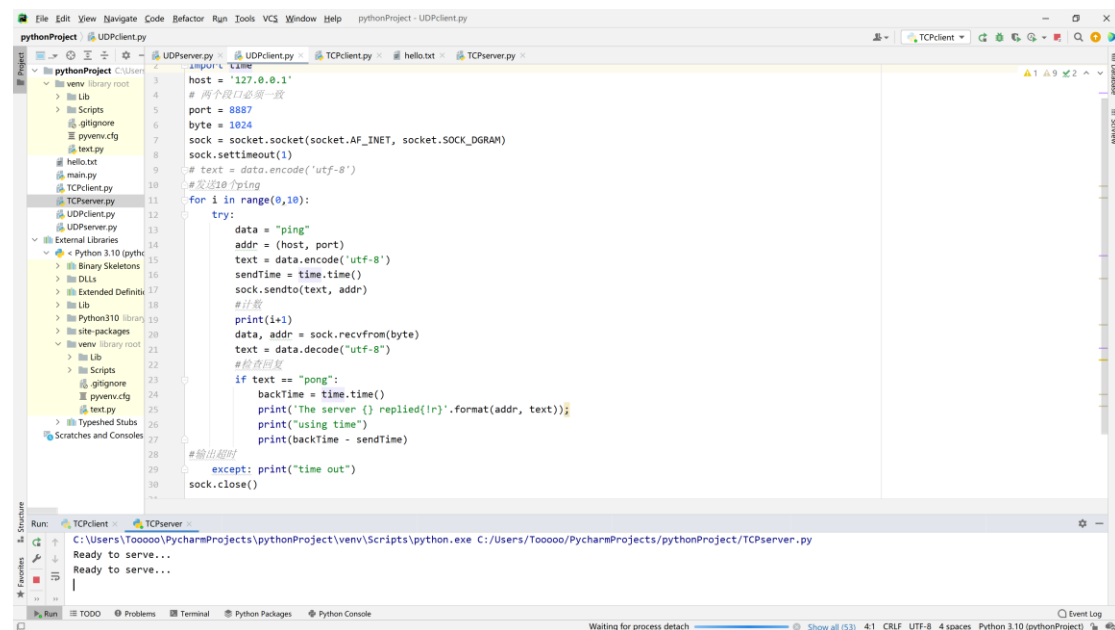
- 利用 python 编写简易 UDP 服务器和客户端，并实现两者间的通讯
- 利用 python 编写简易 TCP 服务器和客户端，并实现两者间的通讯

### 1. UDP:

#### 实验过程:

#### 客户端:

1. 使用 UDP 发送 ping 消息 (注意: 因为 UDP 是无连接协议, 不需要建立连接。);



2. 如果服务器在 1 秒内响应, 则打印该响应消息; 计算并打印每个数据包的往返时间 RTT (以秒为单位);

```
sock.settimeout(1)
```

```
#检查回复
```

```
if text == "pong":
    backTime = time.time()
    print('The server {} replied{!r}'.format(addr, text));
    print("using time")
    print(backTime - sendTime)
```

3. 否则, 打印“请求超时” (中英文皆可)。

```
#输出超时
```

```
except: print("time out")
sock.close()
```

#### 服务器:

1. 创建套接字:

```
# 创建套接字
```

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

2. bind:

```
# 绑定
sock.bind(addr)
```

3. 设置丢包率

```
# 设置丢包率
if rand < 4:
    time.sleep(0.1)
    continue
```

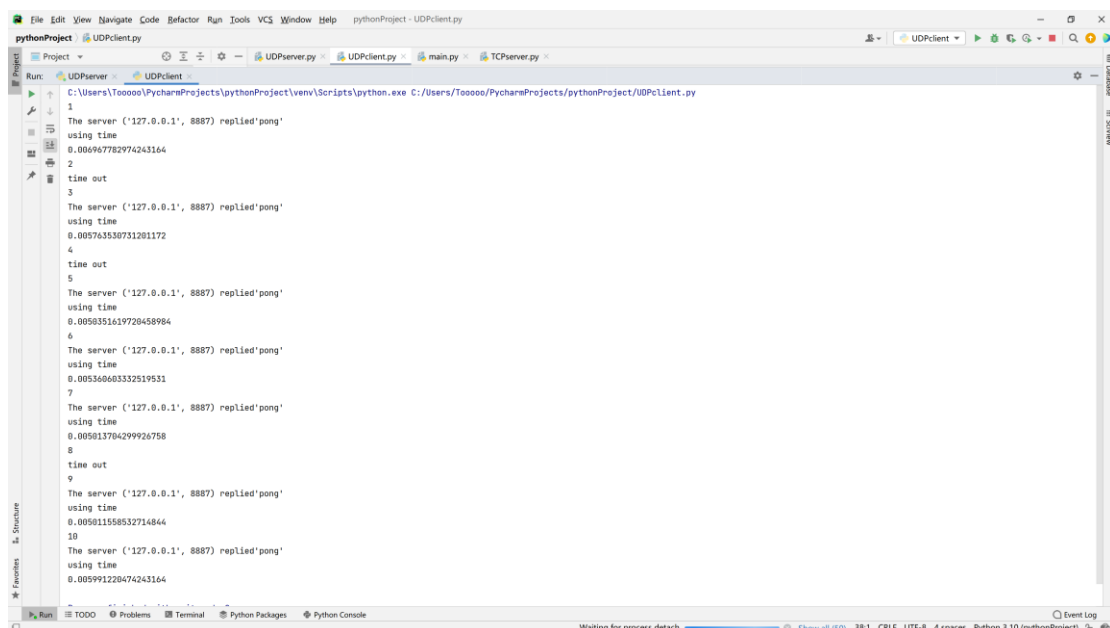
4. 设置响应时间

```
# 设置丢包率
if rand < 4:
    time.sleep(0.1)
    continue
print('The client says {!r}'.format(text))
if text == "ping":
    # 设置响应时间
    time.sleep(0.005)
    data = "pong".encode('utf-8')
    sock.sendto(data, addr)
```

5. 关闭套接字

```
# 关闭套接字
sock.close()
```

结果 客户端:



```
Run: UDPClient
C:\Users\T00000\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\T00000\PycharmProjects\pythonProject\UDPClient.py
1
The server ('127.0.0.1', 8887) replied'pong'
using time
0.006967782974243164
2
time out
3
The server ('127.0.0.1', 8887) replied'pong'
using time
0.005763530731201172
4
time out
5
The server ('127.0.0.1', 8887) replied'pong'
using time
0.0050351619720458984
6
The server ('127.0.0.1', 8887) replied'pong'
using time
0.00536060332519531
7
The server ('127.0.0.1', 8887) replied'pong'
using time
0.005013784299926758
8
time out
9
The server ('127.0.0.1', 8887) replied'pong'
using time
0.005011558532714846
10
The server ('127.0.0.1', 8887) replied'pong'
using time
0.005991228474243164
```

服务器:

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The title bar indicates the current project is 'pythonProject - UDPClient.py'. The main editor window displays the 'UDPClient.py' file, which contains the following code:

```

import socket
import sys

HOST = '127.0.0.1'
PORT = 8080

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(5)
print('waiting to receive messages...')

while True:
    c, addr = s.accept()
    print('The client says \'ping\'')
    c.send('pong')

```

The bottom panel shows the 'Run' tab, indicating the script is running. The terminal output shows the server waiting for messages and then receiving 20 'ping' messages from the client:

```

C:\Users\Toooso\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\Toooso\PycharmProjects\pythonProject\UDPserver.py
waiting to receive messages...
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'
The client says 'ping'

```

The status bar at the bottom indicates the current file is 'UDPClient.py', the encoding is 'UTF-8', and the interpreter is 'Python 3.10 (pythonProject)'.

在本次实验中我遇到了如下问题:

## 套接字的建立

- 丢包率的设置
- 响应时间的计算

解决方法:

在网上学习相应知识，解决问题，详细代码在上文实验过程中有写出。

分析实验结果:

满足 30%丢包率

满足不超时会显示回复和响应时间

满足超时 1s 有 timeout 提示

全部实现

改进措施：本代码已是改进后的代码，更加简洁，逻辑清晰

### 3. TCP

实验过程:

客户端:

1.

```
import socket
import time

MaxBytes = 1024*1024
host = '127.0.0.1'
port = 8083
# 创立套接字
socket_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket_client.settimeout(60)
socket_client.connect((host, port))
```

2. 输入文件和判断退出

```
while True:
    # 输入文件
    inputData = input("enter the name of file")
    # 判断退出
    if inputData == "quit":
        break
    socket_client.send(inputData.encode("utf-8"))
    recvData = socket_client.recv(MaxBytes)
    print(recvData.decode())

    if not recvData:
        print("no data received!")
        break
    recvData = socket_client.recv(MaxBytes)
    print(recvData.decode())
socket_client.close()
print("quit")
```

服务器:

1. 准备接受

```

# import socket module
from socket import *

serverSocket = socket(AF_INET, SOCK_STREAM)
# Prepare a sever socket
serverSocket.bind(('', 8083)) # 将TCP欢迎套接字绑定到指定端口
serverSocket.listen(1)
# 最大连接数为1

while True:
    # Establish the connection
    print('Ready to serve...')
    connectionSocket, addr = serverSocket.accept() # 接收到客户连接请求后, 建立新的TCP连接套接字

```

## 2. 根据客户端发送的消息获取文件名字

```

try:
    message = connectionSocket.recv(1024) # 获取客户发送的报文
    filename = message.split()[1]
    f = open(filename)
    outputdata = f.read();
    # Send one HTTP header line into socket
    header = ' HTTP/1.1 200 OK\r\nConnection: close\r\nContent-Type: text/html\r\nContent-Length: %d\r\n' % (
        len(outputdata))
    connectionSocket.send(header.encode())

    # Send the content of the requested file to the client
    connectionSocket.send(outputdata.encode())

```

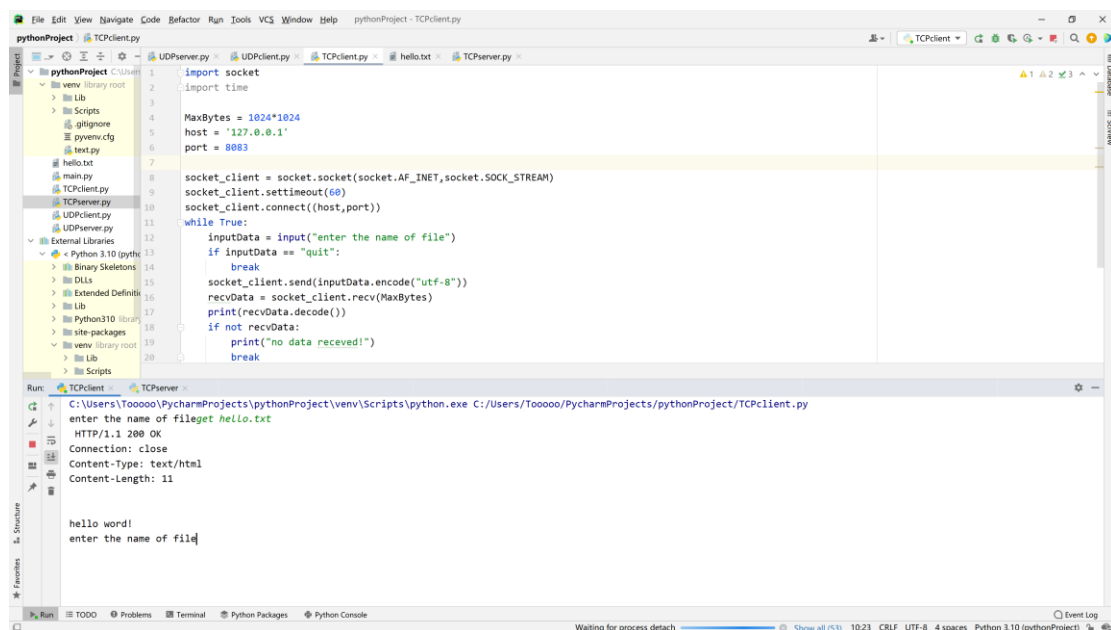
## 3. 找不到文件的报错

```

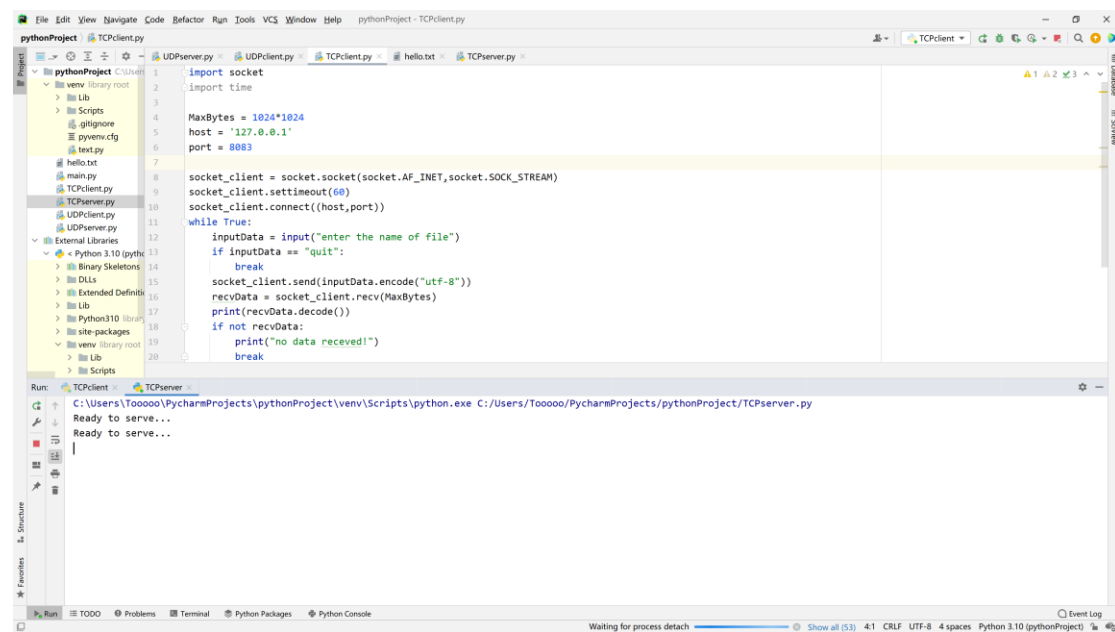
except IOError:
    # Send response message for file not found
    header = ' HTTP/1.1 404 not Found'
    connectionSocket.send(header.encode())

```

结果：客户端



服务器：



在本次实验中我遇到了如下问题：

连接的建立

文件的读写

Error 的写法

解决方法：

在网上学习相应知识，解决问题，详细代码在上文实验过程中有写出。

分析实验结果：

1. 实现服务器收到请求时能创建一个 TCP 套接字；
2. 可以通过这个 TCP 套接字接收 HTTP 请求；
3. 可以解析 HTTP 请求并在缓存中确定客户端所请求的特定文件；
4. 从服务器的文件系统读取客户端请求的文件；
5. 当被请求文件存在时，创建一个由被请求的文件组成的“请求成功”HTTP 响应报文（200OK）；
6. 当被请求文件不存在时，创建“请求目标不存在”HTTP （404 not found）响应报文；
7. 实现通过 TCP 连接将响应报文发回客户端；

全部实现

改进措施：本代码已是改进后的代码，改进服务器向客户端传文件信息的方式，更方便快捷  
其余代码逻辑清晰，简洁。

```
connectionSocket.send(outputdata.encode())  
#for i in range(0, len(outputdata)):  
#    connectionSocket.send(outputdata[i].encode())
```