



南开大学
Nankai University

《计算机网络》实验报告

(2023~2024 学年第一学期)

实验名称：基于套接字的网络程序设计

学 院：软件学院

姓 名：陈高楠

学 号：2112966

指导老师：张圣林

2023 年 12 月 17 日

目录

1 套接字基础与 UDP 通信	1
1 实验目的	1
2 实验条件	1
3 实验报告内容和原理	1
2 TCP 通信与 Web 服务器	4
1 实验目的	4
2 实验条件	4
3 实验报告内容和原理	4
3 SMTP 客户端实现	9
1 实验目的	9
2 实验条件	9
3 实验报告内容和原理	9
4 实验结论与心得体会	13
5 附录	14

实验一 套接字基础与 UDP 通信

1 实验目的

熟悉基于 Python 进行 UDP 套接字编程的基础知识，掌握使用 UDP 套接字发送和接收数据包，以 设置正确的套接字超时，了解 Ping 应用程序的基本概念，并理解其在简单判断网络状态，例如计算数据包丢失率等统计数据方面的意义。

2 实验条件

装有 python 环境的电脑两台；
局域网环境；

3 实验报告内容和原理

实验原理：

基于 UDP 的无连接客户/服务器在 Python 实现中的工作流程如下：

1. 首先在服务器端通过调用 `socket()` 创建套接字来启动一个服务器；
2. 服务器调用 `bind()` 指定服务器的套接字地址，然后调用 `recvfrom()` 等待接收数据。
3. 在客户端调用 `socket()` 创建套接字，然后调用 `sendto()` 向服务器发送数据。
4. 服务器接收到客户端发来的数据后，调用 `sendto()` 向客户发送应答数据，
5. 客户调用 `recvfrom()` 接收服务器发来的应答数据。
6. 一旦数据传输结束，服务器和客户通过调用 `close()` 来关闭套接字。

实验步骤：

- 1、运行服务器端代码。



```

1 from socket import *
2 import random
3
4 # 服务端通过调用socket创建套接字来启动服务器
5 serverSocket = socket(AF_INET, SOCK_DGRAM)
6 # 服务器调用 bind() 绑定服务器的套接字地址
7 serverSocket.bind(('', 7777))
8 while True:
9     rand = random.randint(0, 10)
10    # 服务端调用recvfrom()等待接收数据, 此时阻塞
11    message, address = serverSocket.recvfrom(1024)
12    # 成功接收消息后继续运行
13    print(message)
14    message = message.upper()
15    # 模拟丢失30%的客户端数据包
16    if rand < 4:
17        continue
18    # 服务器接收到客户端发来的数据后, 调用sendto()向客户端发送应答数据
19    serverSocket.sendto(message, address)
20
while True

```

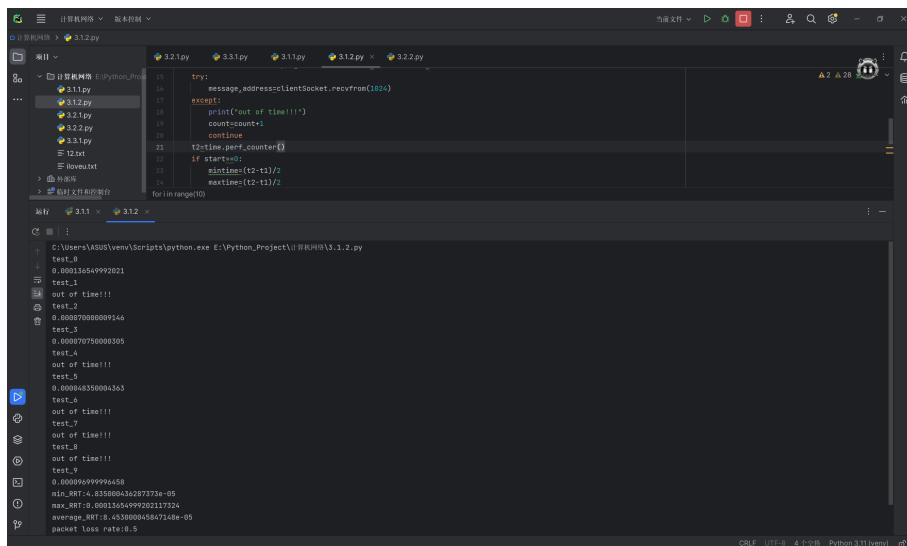
图 1.1: 运行服务器端代码

2、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。）：

3、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)：

4、否则，打印“请求超时”（中英文皆可）。

结果如下：



```

1 try:
2     message, address = clientSocket.recvfrom(1024)
3 except:
4     print('out of time!!!')
5     count=count+1
6     continue
7
8 timeLine_perf_counter()
9 if startFlag:
10    starttime=(t2-t1)/2
11    maxtime=(t2-t1)/2

```

```

C:\Users\ASUS\venv\Scripts\python.exe E:\Python_Project\计算机网3.1.2.py
test_0
0.000136549992021
test_1
out of time!!!
test_2
0.00087088009146
test_3
0.00087075000305
test_4
out of time!!!
test_5
0.00084850004363
test_6
out of time!!!
test_7
out of time!!!
test_8
out of time!!!
test_9
0.00001699999458
min_RRT:0.00013654999202117324
max_RRT:0.00087088009146071494
average_RRT:0.00045000458471494
packet_loss_rate:0.5

```

图 1.2: 运行结果

主要代码逻辑：

服务器端：通过调用 `socket(AF_INET, SOCK_DGRAM)` 创建一个套接字。AF_INET 指定了使用 IPv4 地址，而 SOCK_DGRAM 表示这是一个基于数据报的 UDP 套接字。然后使用 `serverSocket.bind(('', 7777))` 绑定套接字到服务器的地址和端口号。这里的空字符串“”表

示服务器将接受任何地址的连接，而 7777 是指定的端口号。接着服务端进入循环并模拟丢失 30% 的客户端数据包。

客户端：客户端通过调用 `socket(AF_INET, SOCK_DGRAM)` 创建一个 UDP 套接字。AF_INET 指定使用 IPv4 地址，SOCK_DGRAM 表示使用 UDP 协议。然后使用 `settimeout(1)` 设置超时时间为 1 秒。接着开始发送信息。记录发送时间和返回时间，如果超时就输出 "out of time!!!"，等待全部发送后统计时间，输出最小的 RTT, 最大的 RTT, 平均的 RTT, 丢包率。

实验二 TCP 通信与 Web 服务器

1 实验目的

熟悉基于 Python 进行 TCP 套接字编程的基础知识，理解 HTTP 报文格式，能基于 Python 编写一个可以一次响应一个 HTTP 请求，并返回静态文件的简单 Web 服务器。

2 实验条件

装有 python 环境的电脑两台；
局域网环境；

3 实验报告内容和原理

实验原理：

基于 TCP 的面向客户端/服务器在 Python 实现中的的工作流程是：

1. 首先在服务器端通过调用 `socket()` 创建套接字来启动一个服务器；
2. 服务器调用 `bind()` 绑定指定服务器的套接字地址（IP 地址 + 端口号）；
3. 服务器调用 `listen()` 做好侦听准备，同时规定好请求队列的长度；
4. 服务器进入阻塞状态，等待客户的连接请求；
5. 服务器通过 `accept()` 来接收连接请求，并获得客户的 socket 地址。
6. 在客户端通过调用 `socket()` 创建套接字；
7. 客户端调用 `connect()` 和服务器建立连接。
8. 连接建立成功后，客户端和服务器之间通过调用 `read()` 和 `write()` 来接收和发送数据。
9. 数据传输结束后，服务器和客户各自通过调用 `close()` 关闭套接字。

实验步骤：

1. 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；

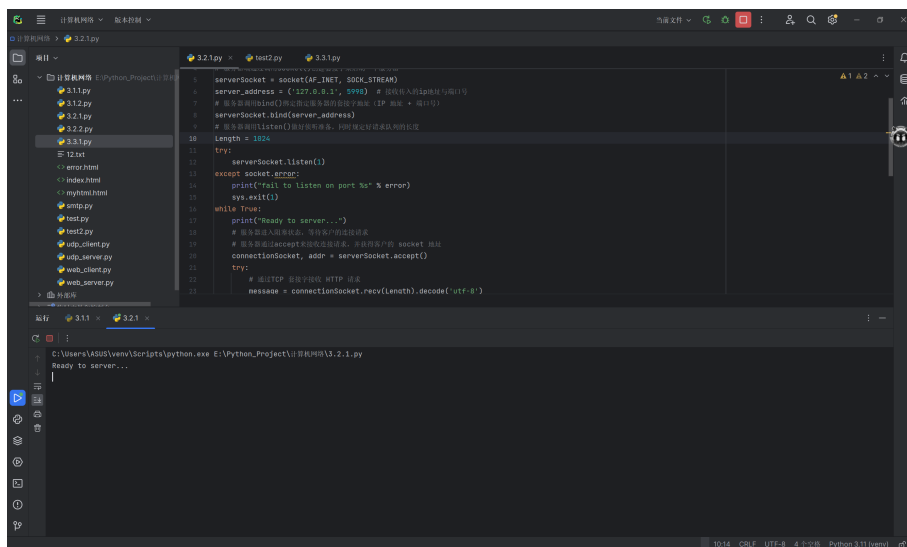


图 2.1: 运行服务器端代码

3. 解析 HTTP 请求并在操作系统中确定客户端所请求的特定文件；

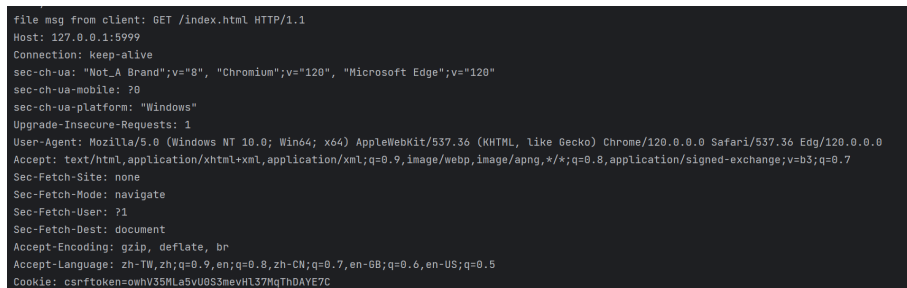


图 2.2: HTTP 请求

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；

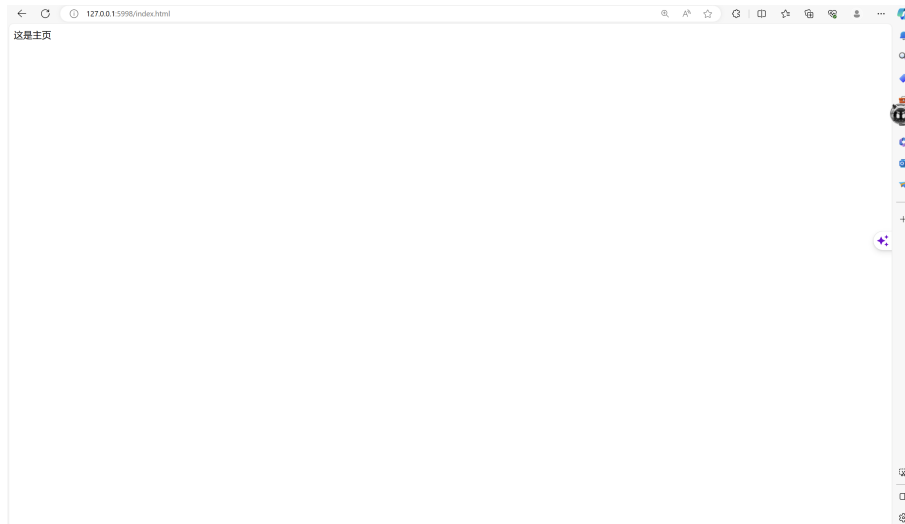


图 2.3: 请求文件存在

6. 当被请求文件不存在时，创建“请求目标不存在” HTTP 响应报文；

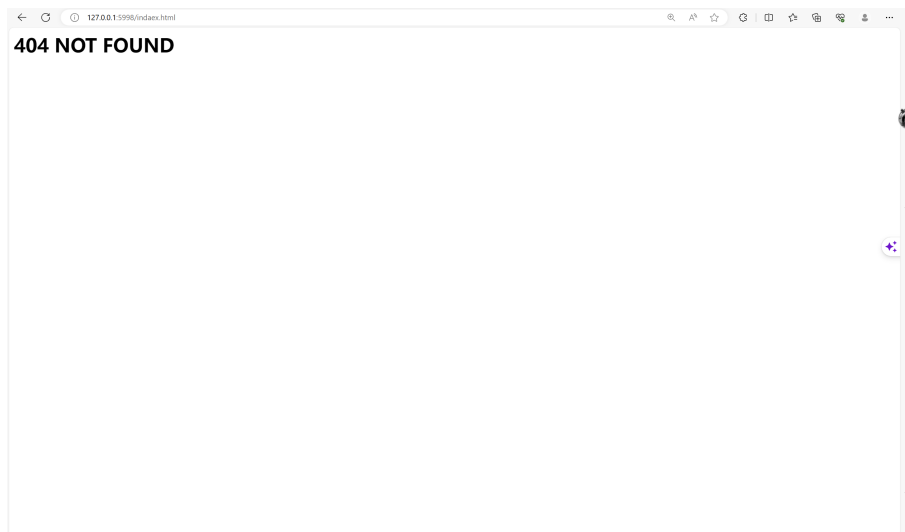


图 2.4: 请求文件不存在

7. 通过 TCP 连接将响应报文发回客户端；

以上过程也可以用代码模拟，将请求得到的信息存在本地的 myhtml.html 文件中，具体效果如下：


```
请求报文: Get index.html HTTP/1.1
Host:127.0.0.1:5998
User-agent:Mozilla/5.0
Connection:open
Accept-language:ch

请求报文发出时间:2023-12-11 23:10:40.57156538963318
响应报文收到时间:2023-12-11 23:10:40.57156538963318
RTT: 0.00028810000003431924 s
响应报文:
HTTP/1.1 200 OK
```

图 2.5: 请求成功

```
请求报文: Get index2.html HTTP/1.1
Host:127.0.0.1:5998
User-agent:Mozilla/5.0
Connection:open
Accept-language:ch

请求报文发出时间:2023-12-11 23:11:29.259548664093018
响应报文收到时间:2023-12-11 23:11:29.259548664093018
RTT: 0.00014409999857889488 s
响应报文:
HTTP/1.1 404 NOT FOUND
```

图 2.6: 请求失败

主要代码逻辑:

服务器端:

通过 `socket(AF_INET, SOCK_STREAM)` 创建一个 TCP 套接字。AF_INET 指定使用 IPv4 地址, SOCK_STREAM 表明这是一个基于流的 TCP 套接字。使用 `bind(server_address)` 绑定套接字到指定的 IP 地址和端口号 ('127.0.0.1', 5998)。接着服务器进入循环, 使用 `accept()` 接收客户端的连接请求, 该方法返回一个新的连接套接字 `connectionSocket` 和客户端的地址 `addr`。然后服务器通过 `connectionSocket.recv(Length)` 接收客户端发送的 HTTP 请求, 并解

码为 UTF-8 格式的字符串。如果收到的消息是”disconnect”，则跳出循环，结束服务。解析 HTTP 请求，从中提取请求的文件名。如果文件存在，则向客户端发送 HTTP 200 OK 响应，随后发送文件内容。如果文件不存在，则向客户端发送 HTTP 404 NOT FOUND 响应，以及 404 错误页面。

客户端：

使用 `socket(AF_INET, SOCK_STREAM)` 创建一个 TCP 套接字。AF_INET 指定 IPv4 地址,SOCK_STREAM 表明这是一个基于流的 TCP 套接字。客户端使用 `connect((serverName, serverPort))` 连接到服务器。这里的 `serverName` 和 `serverPort` 分别是服务器的 IP 地址和端口号。然后构建一个 HTTP GET 请求报文，发送请求报文到服务器。接着使用 `recv(1024)` 接收服务器的响应，此处两次调用 `recv` 分别获取响应头和响应体。打印接收响应的时间，计算并打印往返时间 (Round-Trip Time, RTT)。将响应报文解码并打印出来。并接收到的 HTML 内容写入本地文件 `myhtml.html`。

实验三 SMTP 客户端实现

1 实验目的

进一步理解和掌握基于 Python 进行 TCP 套接字编程的知识, 理解 SMTP 报文格式, 能基于 Python 编写一个简单的 SMTP 客户端程序。

2 实验条件

装有 python 环境的电脑两台;
局域网环境;

3 实验报告内容和原理

实验原理:

简单邮件传输协议 (Simple Mail Transfer Protocol, SMTP) 是实现电子邮件收发的主要应用层协议, 它基于 TCP 提供的可靠数据传输连接, 从发送方的邮件服务器向接收方的邮件服务器发送邮件。注意, 虽然一般情况下邮件总是从发送方的邮件服务器中发出, 但是工作在发送方邮件服务器上的发送程序是一个 SMTP 客户端, 因此一个完整的 SMTP 程序总有两个部分参与工作: 运行在发送方邮件服务器的 SMTP 客户端和运行在接收方邮件服务器的 SMTP 服务器。

实验步骤:

完成一个简单电子邮件客户端程序, 并通过向不同的账号发送电子邮件来测试程序。效果如下: 先输入要发送的邮箱以及发送的内容, 接着查看邮件是否发送成功。

```
请输入你要发送的邮箱: 2112966@mail.nankai.edu.cn
请输入你要发送的内容: 你好, 我今晚在写计算机网络的作业。借你邮箱一用。虽然你不是你, 你是我, 我是你, 我的意思是你邮箱本来就是我的, 你好像就是我。
220 126.com Anti-spam GT for Coremail System (126com[20140526])

HELO Alice

250 OK

AUTH LOGIN

334 dXNlcm5hbWU6

ZnVraW9zdG9uQDEyNi5jb20=

334 UGFzc3dvcmQ6
```

图 3.1: 发送邮件

输出的信息如下:

```
请输入你要发送的邮箱: 2112966@mail.nankai.edu.cn
请输入你要发送的内容: 你好, 我今晚在写计算机网络的作业。借你邮箱一用。虽然你不是你, 你是我, 我是你, 我的意思是你邮箱本来就是我的, 你好像就是我。
220 126.com Anti-spam GT for Coremail System (126com[20140526])

HELO Alice

250 OK

AUTH LOGIN

334 dXNlcm5hbWU6

ZnVraW9zdG9uQDEyNi5jb20=

334 UGFzc3dvcmQ6

SUFYTktsJS1JOUlNQVVFkUg==

235 Authentication successful

MAIL FROM: <fukioston@126.com>

250 Mail OK

RCPT TO: <2112966@mail.nankai.edu.cn>
```

图 3.2: 输出的信息 1

```
250 Mail OK

DATA

354 End data with <CR><LF>.<CR><LF>

from:fukioston@126.com
to:2112966@mail.nankai.edu.cn
subject:=?utf-8?B?5oiR5Zac5qyi6K6h566X5py6572R5uc77yB77yB77yB77yB?=?
Content-Type:text/plain; charset=UTF-8

你好, 我今晚在写计算机网络的作业。借你邮箱一用。虽然你不是你, 你是我, 我是你, 我的意思是你邮箱本来就是我的。你好像就是我。
250 Mail OK queued as zwqz-smtp-mta-g2-1,_____wC36zEBKXdLcaQKDA--.11302S2 1702308097

QUIT

221 Bye
```

图 3.3: 输出的信息 2

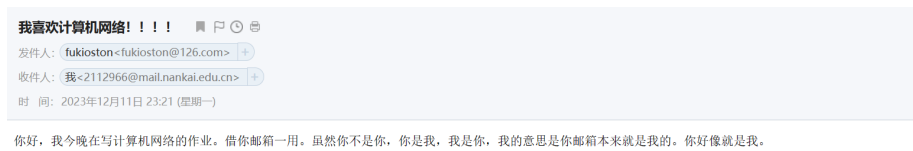


图 3.4: 邮件发送成功

更换邮箱地址再进行测试，



图 3.5: 发送邮件

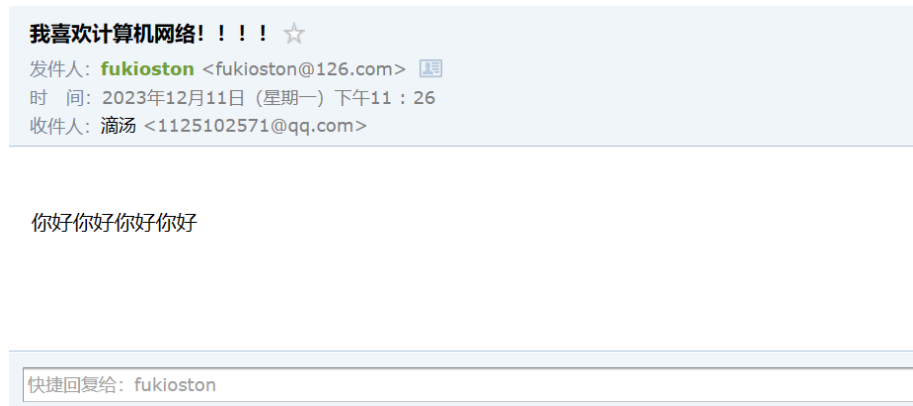


图 3.6: 邮件发送成功

主要代码逻辑：

设置邮件服务器地址和端口（smtp.126.com 和端口 25）。创建一个 TCP 套接字并连接到邮件服务器。接收来自服务器的初始响应，并检查状态码是否为 220（服务准备就绪）。向服

务器发送 HELO 命令，开始 SMTP 会话。接收并检查服务器响应，状态码应为 250（请求动作完成）。发送 AUTH LOGIN 命令请求登录。发送经过 Base64 编码的用户名和密码，检查登录是否成功，状态码应为 235。使用 MAIL FROM 命令指定发件人地址。使用 RCPT TO 命令指定收件人地址。然后发送 DATA 命令，表示开始发送邮件内容。编辑邮件内容，包括发件人、收件人、主题和正文。发送完后发送 QUIT 命令以结束 SMTP 会话。关闭套接字。

实验结论与心得体会

本次实验我学习了 UDP, TCP, SMTP 的相关知识, 并正确使用 UDP 套接字发送和接受数据包, 编写一个可以一次响应一个 HTTP 请求的 Web 服务器, 以及编写一个简单的 SMTP 客户端程序。在第三个实验中, 我遇到了一些困难。在使用 qq 邮箱的 SMTP 服务时一直连接不上 SMTP 的服务器, 好在后来更换了邮箱成功解决。在发送邮件的时候会出现乱码的问题, 后来将发送的信息加上“utf-8”编码便成功解决。本次实验加深了我对 UDP, TCP, SMTP 的认识, 并更加深刻地理解了这些协议的应用场景和工作原理。

附录

一. 套接字基础与 UDP 通信相关代码

服务器端

```
1 from socket import *
2 import random
3
4 # 服务端通过调用socket创建套接字来启动服务器
5 serverSocket = socket(AF_INET, SOCK_DGRAM)
6 # 服务器调用 bind( ) 指定服务器的套接字地址
7 serverSocket.bind(('', 7777))
8 while True:
9     rand = random.randint(0, 10)
10    # 服务端调用recvfrom()等待接收数据,此时阻塞
11    message, address = serverSocket.recvfrom(1024)
12    # 成功接收消息后继续运行
13    print(message)
14    message = message.upper()
15    # 模拟丢失30%的客户端数据包
16    if rand < 4:
17        continue
18    # 服务器接收到客户端发来的数据后,调用sendto()向客户发送应答数据
19    serverSocket.sendto(message, address)
```

客户端

```
1 # -*- coding: UTF-8 -*-
2 from socket import *
3 import time
4 #客户端调用socket()创建套接字
5 clientSocket=socket(AF_INET,SOCK_DGRAM)
6 #使用settimeout函数限制recvfrom()函数的等待时间为1秒
```



```

7 clientSocket.setTimeout(1)
8 count=0
9 start=0
10 totaltime=0
11 for i in range(10):
12     print("test_"+str(i))
13     t1=time.perf_counter()
14     clientSocket.sendto("ping".encode("utf-8"),("127.0.0.1",7777))
15     try:
16         message,address=clientSocket.recvfrom(1024)
17     except:
18         print("out of time!!!")
19         count=count+1
20         continue
21     t2=time.perf_counter()
22     if start==0:
23         mintime=(t2-t1)/2
24         maxtime=(t2-t1)/2
25         start=1
26     elif((t2-t1)/2<mintime):
27         mintime=(t2-t1)/2
28     elif((t2-t1)/2>maxtime):
29         maxtime=(t2-t1)/2
30     totaltime=totaltime+(t2-t1)/2
31     print('%.15f'%((t2-t1)/2))
32 print('min_RRT:'+str(mintime))
33 print('max_RRT:'+str(maxtime))
34 print('average_RRT:'+str(totaltime/(10-count)))
35 print('packet loss rate:'+str(count/10))

```

二.TCP 通信与 Web 服务器相关代码

服务器端

```

1 from socket import *
2 import sys
3
4 # 服务器端通过调用socket()创建套接字来启动一个服务器
5 serverSocket = socket(AF_INET, SOCK_STREAM)
6 server_address = ('127.0.0.1', 5998) # 接收传入的ip地址与端口号

```

```

7 # 服务器调用bind() 绑定指定服务器的套接字地址 (IP 地址 + 端口号)
8 serverSocket.bind(server_address)
9 # 服务器调用listen() 做好侦听准备, 同时规定好请求队列的长度
10 Length = 1024
11 try:
12     serverSocket.listen(1)
13 except socket.error:
14     print("fail to listen on port %s" % error)
15     sys.exit(1)
16 while True:
17     print("Ready to server...")
18     # 服务器进入阻塞状态, 等待客户的连接请求
19     # 服务器通过accept来接收连接请求, 并获得客户的 socket 地址
20     connectionSocket, addr = serverSocket.accept()
21     try:
22         # 通过TCP 套接字接收 HTTP 请求
23         message = connectionSocket.recv(Length).decode('utf-8')
24         print('file msg from client: ' + message)
25         # 收到客户端断开连接消息
26         if message == 'disconnect': break
27         # 从服务器的文件系统读取客户端请求的文件
28         print(message)
29         filename = message.split()[1]
30         filename = filename.lstrip('/')
31         file = open(filename, "rb")
32         content=file.read()
33         # 被请求文件存在, 创建一个由被请求的文件组成的“请求成功”HTTP 响应...
34         # 报文
35         response = "HTTP/1.1 200 OK\r\n\r\n"
36         connectionSocket.send(response.encode("utf-8"))
37         file.close()
38         connectionSocket.send(content)
39     except IOError:
40         response = "HTTP/1.1 404 NOT FOUND\r\n\r\n"
41         connectionSocket.send(response.encode("utf-8"))
42         connectionSocket.send("<h1>404 NOT FOUND</h1>".encode("utf-8"))
43         # 被请求文件不存在, 创建“请求目标不存在”HTTP 响应报文
44     print("finish test, close connect")
45     # 通过close() 关闭套接字
46     connectionSocket.close()
47     serverSocket.close()

```

客户端

```

1
2
3 import time
4 from socket import *
5 import numpy as np
6 serverName = "127.0.0.1" # 服务器主机
7 serverPort = 5998 # 端口号
8 clientsocket = socket(AF_INET,SOCK_STREAM) # 创建客户机套接字
9 clientsocket.connect((serverName,serverPort)) # 建立连接
10 ## 发送请求报文并接收服务器的回复
11 fetch_file = "index2.html" # 需要请求的文件
12 requestRow = "Get "+fetch_file+" HTTP/1.1\r\n" # 请求行
13 firstRow = "Host:127.0.0.1:5998\r\nUser-agent: Mozilla/5.0\r\nConnection:open...
            \r\nAccept-language:ch\r\n\r\n" # 首部行
14 requestMessages = requestRow+firstRow # 请求报文 (请求行+首部行)
15 # requestRow+firstRow
16 print("请求报文: "+requestMessages)
17 print("请求报文发出时间:",time.strftime("%Y-%m-%d %H:%M:", time.localtime(...
            time.time())) ,np.mod(time.time(),60),sep="")
18
19 start = time.perf_counter()
20 clientsocket.send(requestMessages.encode()) # 发送请求报文
21 responseMessage = clientsocket.recv(1024) # 接收服务器的回复
22 responseMessage2 = clientsocket.recv(1024)
23 print("响应报文收到时间:",time.strftime("%Y-%m-%d %H:%M:", time.localtime(...
            time.time())) ,np.mod(time.time(),60),sep="")
24 end = time.perf_counter()
25 print("RTT:",end-start,"s")
26
27 print("响应报文: \n",responseMessage.decode(),sep=" ")
28 ## 生成本地html文件
29 f = open('myhtml.html', 'w',encoding="utf-8")
30 message = responseMessage2.decode() # 报文解码
31 f.write(message) # 保存到本地文件中
32 f.close()
33 ## 关闭套接字
34 clientsocket.close()

```

三.SMTP 客户端实现相关代码

邮件发送代码

```

1 import base64
2 from socket import *
3 # 你好，我今晚在写计算机网络的作业。借你邮箱一用。虽然你不是你，你是我，我是...
   你，我的意思是你邮箱本来就是我的。你好像就是我。
4 endMsg = "\r\n.\r\n"
5 # 选择一个邮件服务
6 # IAXNKKRNRSPUQJR
7 mailServer = "smtp.126.com"
8 # 发送方地址和接收方地址，from 和 to
9 fromAddress = "fukioston@126.com"
10 toAddress = input("请输入你要发送的邮箱：")
11 msg = input("请输入你要发送的内容：")
12 msg="\r\n"+msg
13 # "2112966@mail.nankai.edu.cn"
14 # 1125102571@qq.com
15 encodefromAddress = base64.b64encode(fromAddress.encode('utf-8'))
16 username = str(encodefromAddress, 'utf-8') # 将发件人邮箱进行编码
17 passcode = "IAXNKKRNRSPUQJR"
18 encodepasscode = base64.b64encode(passcode.encode('utf-8'))
19 password = str(encodepasscode, 'utf-8') # 将授权码进行编码
20
21
22 # 创建客户端套接字并建立连接
23 serverPort = 25 # SMTP使用25号端口
24 clientSocket = socket(AF_INET, SOCK_STREAM)
25 clientSocket.connect((mailServer, serverPort))
26 # 从客户套接字中接收信息
27 recv = clientSocket.recv(1024).decode()
28 print(recv)
29 if '220' != recv[:3]:
30     print('220 reply not received from server.')
31
32 # 发送 HELO 命令并且打印服务端回复
33 # 开始与服务器的交互，服务器将返回状态码250,说明请求动作正确完成
34 helloCommand = 'HELO Alice\r\n'
35 print(helloCommand)
36 clientSocket.send(helloCommand.encode()) # 随时注意对信息编码和解码

```

```

37 recv1 = clientSocket.recv(1024).decode()
38 print(recv1)
39 if '250' != recv1[:3]:
40     print('250 reply not received from server.')
41
42 # 发送"AUTH LOGIN"命令, 验证身份. 服务器将返回状态码334 (服务器等待用户输入验证信息)
43 print('AUTH LOGIN\r\n')
44 clientSocket.sendall('AUTH LOGIN\r\n'.encode())
45 recv2 = clientSocket.recv(1024).decode()
46 print(recv2)
47
48 # 发送验证信息
49 print(username + '\r\n')
50 clientSocket.sendall((username + '\r\n').encode())
51 recvName = clientSocket.recv(1024).decode()
52 print(recvName)
53
54 print(password + '\r\n')
55 clientSocket.sendall((password + '\r\n').encode())
56 recvPass = clientSocket.recv(1024).decode()
57 print(recvPass)
58 # 如果用户验证成功, 服务器将返回状态码235
59
60
61 # TCP连接建立好之后, 通过用户验证就可以开始发送邮件。邮件的传送从MAIL命令开始, MAIL命令后面附上发件人的地址。
62 # 发送 MAIL FROM 命令, 并包含发件人邮箱地址
63 print('MAIL FROM: <' + fromAddress + '>\r\n')
64 clientSocket.sendall(('MAIL FROM: <' + fromAddress + '>\r\n').encode())
65 recvFrom = clientSocket.recv(1024).decode()
66 print(recvFrom)
67
68
69 # 接着SMTP客户端发送一个或多个RCPT (收件人recipient的缩写)命令, 格式为RCPT ... TO: <收件人地址>。
70 # 发送 RCPT TO 命令, 并包含收件人邮箱地址, 返回状态码 250
71 print('RCPT TO: <' + toAddress + '>\r\n')
72 clientSocket.sendall(('RCPT TO: <' + toAddress + '>\r\n').encode())
73 recvTo = clientSocket.recv(1024).decode() # 注意UDP使用sendto, recvfrom
74 print(recvTo)
75
76

```

```
77 # 发送 DATA 命令，表示即将发送邮件内容。服务器将返回状态码354（开始邮件输...
    入，以”.”结束）
78 print('DATA\r\n')
79 clientSocket.send('DATA\r\n'.encode())
80 recvData = clientSocket.recv(1024).decode()
81 print(recvData)
82
83
84 # 编辑邮件信息，发送数据
85 subject = "我喜欢计算机网络!!!!".encode('utf-8')
86 contentType = "text/plain; charset=UTF-8"
87
88 message = 'from:' + fromAddress + '\r\n'
89 message += 'to:' + toAddress + '\r\n'
90 message += 'subject:' + "?utf-8?B?" + base64.b64encode(subject).decode() + ...
    '=?\r\n'
91 message += 'Content-Type:' + contentType + '\r\n'
92 message += '\r\n' + msg
93 print(message)
94 clientSocket.sendall(message.encode())
95
96 # 以”.”结束。请求成功返回 250
97 clientSocket.sendall(endMsg.encode())
98 recvEnd = clientSocket.recv(1024).decode()
99 print(recvEnd)
100 if '250' != recvEnd[:3]:
101     print('250 reply not received from server')
102
103 # 发送"QUIT"命令，断开和邮件服务器的连接
104 print('QUIT\r\n')
105 clientSocket.sendall('QUIT\r\n'.encode())
106 recvEnd = clientSocket.recv(1024).decode()
107 print(recvEnd)
108 clientSocket.close()
```