

基于套接字的网络程序设计

Author: 秦华谦 21312683

Date: April 30, 2023

实验1 套接字基础与UDP通信

1.1 代码部分（代码的说明包含在注释内）

服务器端（时间戳服务器）

```
from socket import *
from time import ctime
import time
import random

HOST = ''
PORT = 8080
BUFSIZ = 1024
ADDR = (HOST, PORT)

udpSerrverSocket=socket(AF_INET, SOCK_DGRAM) # 创建UDP连接
udpSerrverSocket.bind(ADDR) # 绑定服务器地址

while True: # 服务器无限循环
    print('等待连接...')
    data,addr=udpSerrverSocket.recvfrom(BUFSIZ) # 接受客户的连接
    # udpSerrverSocket.sendto(bytes('%s' %s' % (ctime(), data),encoding='utf-8'), addr) # 发送UDP 数据
    timestamp_ms = int(time.time() * 1000 + time.time_ns() % 1000000 // 1000)
    # 构造新的时间戳字符串
    timestamp_str = bytes('%d' %s' % (timestamp_ms, data), encoding='utf-8')
    # 一个调用random的随机函数，30%的概率重启循环
    if random.randint(0,9) < 3:
        continue
    udpSerrverSocket.sendto(timestamp_str, addr) # 反馈给客户端同样的时间
    print('连接地址:', addr)

udpSerrverSocket.close() # 关闭服务器连接
```

用户端

```
from socket import *
from datetime import datetime
import threading
import time

HOST = '172.28.63.133' # 服务器连接地址
PORT = 8080 # 服务器启用端口
BUFSIZ = 1024 # 缓冲区大小
ADDR = (HOST, PORT) # 目标地址
```

```

udpCliendSocket = socket(AF_INET, SOCK_DGRAM)

count = 0 # 计数器（用于计算送达的package个数）
time_start = '' # 第一个包的送达时间
time_end = 0 # 最后一个包的送达时间
ADDA = 0 # 目的地信息

def recv_timeout():
    pass # 为了配合函数调用变量需要设置的无用函数

def timestamp_diff_ms(timestamp1, timestamp2): # 两个时间戳时间差的函数，用于计算RTT
    dt1 = datetime.fromtimestamp(timestamp1 / 1000 + (timestamp1 % 1000) /
1000000)
    dt2 = datetime.fromtimestamp(timestamp2 / 1000 + (timestamp2 % 1000) /
1000000)
    diff_seconds = (dt2 - dt1).total_seconds()
    diff_ms = diff_seconds * 1000
    return diff_ms

print("正在ping", HOST, "端口号", PORT)

for i in range(10):
    data = '1' # 模拟传输的报文
    data = bytes(data, encoding='utf-8') # 将字符中解码为bytes类型
    # 创建一个发送时间的时间戳，小数点后6位级别的
    send_timestamp = int(time.time() * 1000 + time.time_ns() // 1000000 % 1000 +
time.time_ns() % 1000000 / 1000000)

    ADDR = (HOST, PORT) # 在每次循环中更新ADDR的值
    udpCliendSocket.sendto(data, ADDR) # 调用发送接口
    udpCliendSocket.settimeout(1) # 设置超时时间为1秒
    data, ADDR = None, None
    try: # 捕捉错误
        data, ADDR = udpCliendSocket.recvfrom(BUFSIZ) # 等待接受服务器回应
        if not time_start: # 如果当前的是第一个包
            time_start = data
        time_end = data
        count += 1 # 计数器加一
        # 创建接受时间时间戳
        recv_timestamp = int(time.time() * 1000 + time.time_ns() // 1000000 %
1000 +
            time.time_ns() % 1000000 / 1000000)
        print("连接成功...", "RTT =", timestamp_diff_ms(send_timestamp,
recv_timestamp), "ms")
    except timeout:
        print("请求超时。") # 超时输出
        continue # 超时后使用continue跳过本轮循环
    finally:
        udpCliendSocket.settimeout(None) # 恢复默认的超时时间

# 将服务器时间戳bytes字符串转换为字符串，并使用切片操作获取时间戳部分
if time_start:
    # 将时间戳bytes字符串转换为字符串，并使用切片操作获取时间戳部分
    time_start = time_start.decode('utf-8')[1:14]
    time_end = time_end.decode('utf-8')[1:14]

```

```

# 将时间戳字符串转换为datetime对象
dt1 = datetime.fromtimestamp(int(time_start) / 1000)
dt2 = datetime.fromtimestamp(int(time_end) / 1000)

# 计算两个datetime对象之间的时间差
delta = dt2 - dt1

# 总共发送了十个数据包，count用于计数已经收到了的个数，输出丢失率
print("=====")
print(ADDR, "的统计信息：")
print("\t 数据包：已发送 = 10, 已接收 =", count, ", 丢失 =", 10 - count, "(", 100
* (10 - count) / 10, "% 丢失)")
# Max RTT in RTT_list
print("\t 最大RTT: ", max(RTT_list), "ms")
# Min RTT in RTT_list
print("\t 最小RTT: ", min(RTT_list), "ms")
# Average RTT in RTT_list
print("\t 平均值: ", sum(RTT_list) / len(RTT_list), "ms")
print("单趟行程的总时间：")
print("\t", delta)
else:
    print("未收到任何时间戳数据包")
    print("请检查连接（IP or Port）是否正确")

udpCliendSocket.close()

```

1.2 运行结果展示（不同环境下代码运行的结果）

服务器端（运行在同一台终端上）

```

PS E:\OneDrive -
mail2.sysu.edu.cn\Project\SYSU_Computer_Network_exp\No.1_TCP&UDP\UDP> python
Release_server.py
等待连接...
等待连接...
连接地址：('172.28.63.133', 53127)
等待连接...
连接地址：('172.28.63.133', 53127)
等待连接...
连接地址：('172.28.63.133', 53127)
等待连接...
连接地址：('172.28.63.133', 53127)
等待连接...
等待连接...
等待连接...
连接地址：('172.28.63.133', 53127)
等待连接...
连接地址：('172.28.63.133', 53127)
等待连接...
等待连接...

```

客户端

```
PS E:\OneDrive -
mail2.sysu.edu.cn\Project\SYSU_Computer_Network_exp\No.1_TCP&UDP> &
D:/Python/3.11/python.exe "e:/OneDrive -
mail2.sysu.edu.cn/Project/SYSU_Computer_Network_exp/No.1_TCP&UDP/UDP/Release_use
r.py"
正在ping 172.28.63.133 端口号 8080
请求超时。
连接成功... RTT = 2.002 ms
连接成功... RTT = 2.002 ms
连接成功... RTT = 2.002 ms
连接成功... RTT = 2.002 ms
请求超时。
请求超时。
连接成功... RTT = 0.0 ms
连接成功... RTT = 2.002 ms
请求超时。

=====
统计信息:
    数据包: 已发送 = 10, 已接收 = 6 , 丢失 = 4 ( 40.0 % 丢失)
    最大RTT:  2.002 ms
    最小RTT:  0.0 ms
    平均值:  1.6683333333333333 ms
单趟行程的总时间:
    0:00:01.856000
```

说明: 因为本人在完成代码的时候, 身边仅有本人自己的终端, 因此没有包含不同终端的展示结果。但是在实验课上本人已经完成了在不同终端上面的测试, 在同一局域网下可以正常运行。

1.3 分析与总结

1.3.1 实验思路

最开始我写了第一版程序, 仅仅测试了UDP的基础功能, 实现了报文的传输, 在服务器端也没有模拟报文丢失的情况, 如下:

```
from socket import *
port=12000
socket_server=socket(AF_INET,SOCK_DGRAM)
socket_server.bind(('',port))
message_server="I'm server!"
while True:
    message,address_client=socket_server.recvfrom(1024)
    print('The address of client is:',address_client)
    print('The message from client is:',message.decode())
    socket_server.sendto(message_server.encode(),address_client)

-----

from socket import *
server='192.168.2.24'
server_port=12000
socket_client=socket(AF_INET,SOCK_DGRAM)
for i in range(10):
```

```

message="this is NO."
socket_client.sendto(message.encode(),(server,server_port)) # 两台主机之间建立通信是通过进程的进行的，因此需要明确端口号(当然首先要明确主机地址)
message_receive,address_server=socket_client.recvfrom(1024) # 1024表示缓存长度
print("The address of server is:",address_server)
print("The message from server is:",message_receive.decode())
socket_client.close()

```

碰到的第一个错误是：[WinError 10054] 远程主机强迫关闭了一个现有的连接。，检查发现是因为IP地址和服务器端不匹配所致

以上是在课堂上实现的部分，课下，我重新思考了一下如何实现功能，需要实现以下功能

- 连续发送十个数据包并监测是否收到
- 判断每一个数据包的RTT
- 计算有几个数据包送达，有几个数据包丢失，计算丢包率

围绕这些功能，需要以下方法实现

- 有一个循环连续发送10次
- 要有一个时间戳，在发送时记录，在接收时记录，最后相减算出差值
- 要有一个计数器来为到达的数据包进行计数

然后开始具体实现，问题主要出现在时间戳部分，在进展的过程中主要遇到了以下问题

1.3.2 过程中遇到的问题

1. 对RTT认识错误

最开始我以为RTT是数据包从发送端到接收端的单程时间，因此大费周章的设计了一个时间戳服务器——当报文送达的时候，服务器会给报文加盖时间戳并进行回传。

客户端收到报文以后，将对报文进行解析，将时间戳拆解出来，与发送的时间相减即可得到单程时间。

但在实现以后我才发现，RTT是一个完整来回的时间，这样我只需要在客户端完成时间的计算即可，无需将时间再报文中来回传输。

不过我最终还是选择了将这个时间戳服务器保留了下来，数据用在了最后的单趟行程的总时间：0:00:01.856000 中，算是没有白干一场hhh

2. 时间精度不够

最开始我想着时间简单调用 `time.time()` 即可，但是这个函数返回的时间是秒级的数据，在报文传输的过程中都是毫秒级的数据，因此每次显示的都是 `RTT = 0s`。无奈，只得调整时间戳的数据，但是调整以后对时间戳的解读也变得比较复杂

最后的解决方案如下

生成时间

```

int(time.time() * 1000 + time.time_ns() // 1000000 % 1000 + time.time_ns() % 1000000 / 1000000)

```

解析时间

```
datetime.fromtimestamp(timestamp1 / 1000 + (timestamp1 % 1000) / 1000000)
```

这样时间的精度就被提升到了小数点后六位，可以显示毫秒级的数据了

3. recvfrom 的阻塞模式影响超时退出

因为 `socket_client.recvfrom` 是以阻塞模式运行的，因此当发生丢包的时候就没办法退出，卡住。

最开始考虑过使用 `timer` + 多线程 + 回调函数的方式实现，但是实践过程中发现这种方法性能开销过大并且我写出了调不玩的bug（主要原因），放弃

后来了解到 `udpCliendSocket.recvfrom(BUFSIZ)` 同样是以阻塞模式进行运行的，但是可以使用 `udpCliendSocket.settimeout(1)` 设定超时时间，当超时以后 `recvfrom` 函数会自动抛出 `timeout`，被 `except` 捕捉到以后，输出“请求超时”，因此使用这种方法，成功实现

实验2 TCP通信与Web服务器

2.1 代码展示

2.1.1 服务器

```
# -*- coding: utf-8 -*-
from socket import *

# 定义服务器地址和端口
serverHost = ''
# serverHost = '192.168.1.666'
serverPort = 8080

# 准备服务器端 socket
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind((serverHost, serverPort))
serverSocket.listen(1)

while True:
    print("Ready to serve...")

    # 接受客户端连接请求
    connectionSocket, addr = serverSocket.accept()

    try:
        # 接收客户端发送的 HTTP 请求消息
        message = connectionSocket.recv(1024).decode()
        if not message:
            continue

        # 获取客户端请求的文件名
        filename = message.split()[1].lstrip('/')

```

```

# 打开文件并读取文件内容
with open(filename, 'rb') as f:
    content = f.read()
# 构造 HTTP 响应消息头部
response = 'HTTP/1.1 200 OK\r\n' + \
           'Content-Type: text/html; charset=UTF-8\r\n' + \
           'Content-Length: {}\r\n'.format(len(content)) + \
           '\r\n'

# 将响应消息头部和文件内容发送给客户端
connectionSocket.send(response.encode())
connectionSocket.send(content)
except IOError:
    # 如果文件不存在, 发送 404 Not Found 响应消息
    response = 'HTTP/1.1 404 Not Found\r\n\r\n'
    connectionSocket.send(response.encode())
finally:
    # 关闭客户端 socket
    connectionSocket.close()

# 关闭服务器 socket
serverSocket.close()

```

2.1.2 客户端

```

# -*- coding: utf-8 -*-

import socket

# 自动找出本机的IP地址以完成全自动访问操作
# 获取本机的主机名
hostname = socket.gethostname()

# 获取本机的 IP 地址
ip_address = socket.gethostbyname(hostname)

# 打印本机的 IP 地址
print("My IP address is:", ip_address)

# 将本机的 IP 地址作为服务器的地址
serverHost = ip_address
serverPort = 8080

# 创建客户端 socket
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 连接到服务器
clientSocket.connect((serverHost, serverPort))

# 发送 HTTP GET 请求
request = 'GET /helloWorld.html HTTP/1.1\r\nHost: {}\r\n\r\n'.format(serverHost)
clientSocket.send(request.encode())

```

```
# 接收服务器响应
response = clientSocket.recv(1024).decode()

# 输出服务器响应的内容
print(response)

# 关闭客户端 socket
clientSocket.close()
```

2.1.3 网页代码

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Simple HTML Page</title>
  <style>

    body {
      background-color: #f1f1f1;
      font-family: Arial, sans-serif;
    }

    h1 {
      font-size: 36px;
      color: #333;
      margin-bottom: 20px;
    }

    p {
      font-size: 18px;
      color: #666;
      line-height: 1.5;
      margin-bottom: 20px;
    }

    ul {
      list-style: none;
      margin: 0;
      padding: 0;
    }

    li {
      margin-bottom: 10px;
    }

    a {
      color: #0099cc;
      text-decoration: none;
    }

    a:hover {
      text-decoration: underline;
    }
  </style>
```



```

</head>
<body>
  <h1>welcome to My Website!</h1>
  <p>This is a simple HTML page, designed to demonstrate the basics of HTML
and CSS.</p>
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">About Us</a></li>
    <li><a href="#">Products</a></li>
    <li><a href="#">Contact Us</a></li>
  </ul>
</body>
</html>

```

2.2 运行结果展示

2.2.1 客户端——终端执行效果

```

PS E:\OneDrive -
mail2.sysu.edu.cn\Project\SYSU_Computer_Network_exp\No.1_TCP&UDP\TCP> &
D:/Python/3.11/python.exe "e:/OneDrive -
mail2.sysu.edu.cn/Project/SYSU_Computer_Network_exp/No.1_TCP&UDP/TCP/release_use
r.py"
My IP address is: 172.29.93.16
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 1200

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Simple HTML Page</title>
  <style>

    body {
      background-color: #f1f1f1;
      font-family: Arial, sans-serif;
    }

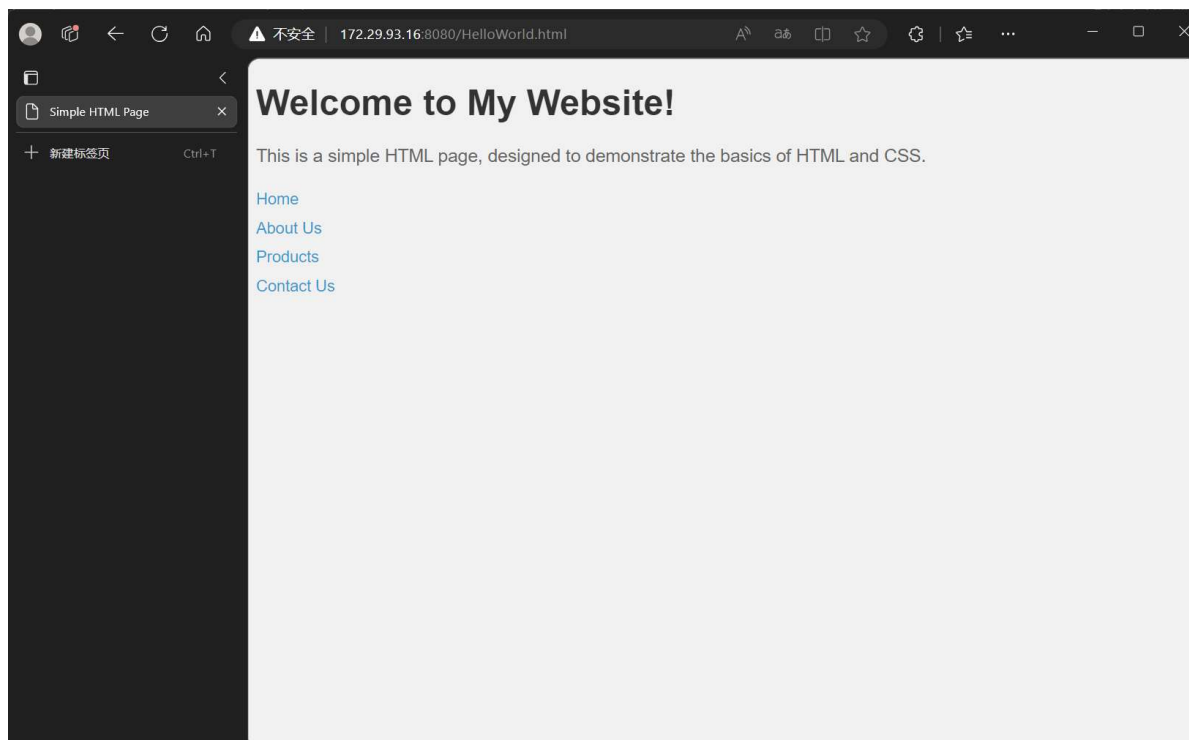
    h1 {
      font-size: 36px;
      color: #333;
      margin-bottom: 20px;
    }

    p {
      font-size: 18px;
      color: #666;
      line-height: 1.5;
      margin-bottom: 20px;
    }
  </style>

```

```
ul {  
    list-style: none;  
    margin: 0;  
}  
</head>  
<body>  
    <h1>welcome to My Website!</h1>  
    <p>This is a simple HTML page, designed
```

2.2.2 客户端——浏览器执行效果



2.2.3 服务器执行效果

```
PS E:\OneDrive -  
mail2.sysu.edu.cn\Project\SYSU_Computer_Network_exp\No.1_TCP&UDP\TCP> python  
release_server.py  
Ready to serve...
```

2.3分析与总结

2.3.1 实验过程

经过查询，在Python程序中，如果想要完成一个tcp服务器的功能，需要的流程如下：

- socket创建一个套接字
- bind绑定ip和port
- listen使套接字变为可以被动链接
- accept等待客户端的连接
- recv接收数据

按照以上思路，具体的实验过程在代码的注释中都有详细地给出

想要实现多个进程并行连接只需要修改 `serverSocket.listen(1)` 即可，例如 `serverSocket.listen(128)`，即可实现128个进程并行

2.3.2 一些思考

1) 客户端执行一次只发送一次数据就将接口close了还是会保持连接？

因为TCP相较于UDP的一个显著的区别就在于能够建立一次连接以后，持续性的进行沟通。但是对于这次的任务来讲，服务器只需要响应一次，将html的内容回传即可，所以这个程序是单词连接以后，在 `finally` 语句中就将接口进行了关闭。

我尝试写了一个报文发送服务器，能够在建立一次连接以后保持住，发送多条报文，主要代码如下

```
# 本地信息
address = ('', 7788)
# 绑定
tcp_server_socket.bind(address)
tcp_server_socket.listen(128)
while True:
    # 等待新的客户端连接
    client_socket, client_addr = tcp_server_socket.accept()
    while True:
        # 接收对方发送过来的数据
        recv_data = client_socket.recv(1024) # 接收1024个字节
        if recv_data:
            print('接收到的数据为:', recv_data.decode('gbk'))
        else:
            break
    client_socket.close()

tcp_server_socket.close()
```

`recv`函数工作在阻塞模式下，`recv`解堵塞有两种情况，对方发过来数据或者对方调用`close`，当对方调用`close`的时候，也就是发送报文为空的情况下，就能将这个连接关闭。

2) 关于listen与accept

- 当一个tcp客户端连接服务器时，服务器端会有1个新的套接字，这个套接字用来标记这个客户端，单独为这个客户端服务。
- `listen`后的套接字是被动套接字，用来接收新的客户端的链接请求的，而`accept`返回的新套接字是标记这个新客户端的。
- 关闭`listen`后的套接字意味着被动套接字关闭了，会导致新的客户端不能够链接服务器，但是之前已经链接成功的客户端正常通信。
- 关闭`accept`返回的套接字意味着这个客户端已经服务完毕。

3) 遇到的问题：IP地址的变化导致的代码调试麻烦

因为在我的电脑连接在校园网中，而校园网是由多个路由器搭建的无线信号，每当链接到一个新的路由器的时候，我就会分配到一个新的IP地址，意味着我客户端中写的那个自己的地址就不再可用了，为此经常出现报错如：远程主机关闭了一个连接

因此，经过查找，我选择了调用这段代码，自动查找本机的IP地址，大大方便了代码的调试过程

```
ip_address = socket.gethostbyname(hostname)
```

4) 对运行结果的改进

最开始我只是在html文件里面写了一个简单的helloworld字符，后来我研究了一下，简单的写了一个网页，这样使得在浏览器中调用的时候能够更加美观

然后我考虑了一下，如果我修改一下http响应头部是否能够获取一个互联网上面的网页的信息呢？

于是我进行里如下的尝试

```
# 导入socket库：
import socket
# 创建一个socket：
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 建立连接：
s.connect(('www.baidu.com', 80))

# 发送数据：
s.send(b'GET / HTTP/1.1\r\nHost: www.baidu.com\r\nConnection: close\r\n\r\n')

# 接收数据：
buffer = []
while True:
    # 每次最多接收1k字节：
    d = s.recv(1024)
    if d:
        buffer.append(d)
    else:
        break
data = b''.join(buffer)

# 关闭连接：
s.close()
print(data)
header, html = data.split(b'\r\n\r\n', 1)
print(header.decode('utf-8'))
# 把接收的数据写入文件：
with open('baidu.html', 'wb') as f:
    f.write(html)
```

成功获取到了百度的主页的html代码，如下（过长，截取部分）

```
<!DOCTYPE html><html><head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8"><meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"><meta content="always" name="referrer"><meta name="description" content="全球领先的中文搜索引擎、致力于让网民更便捷地获取信息，找到所求。百度超过千亿的中文网页数据库，可以瞬间找到相关的搜索结果。"><link rel="shortcut icon" href="//www.baidu.com/favicon.ico" type="image/x-icon"><link rel="search" type="application/opensearchdescription+xml" href="//www.baidu.com/content-search.xml" title="百度搜索"><title>百度一下，你就知道</title><style type="text/css">body{margin:0;padding:0;text-align:center;background:#fff;height:100%}html{overflow-y:auto;color:#000;overflow:-moz-scrollbar;height:100%}body,input{font-size:12px;font-family:"PingFang SC",Arial,"Microsoft YaHei",sans-serif}a{text-decoration:none}a:hover{text-decoration:underline}img{border:0;-ms-interpolation-mode:bicubic}input{font-size:100%;border:0}body,form{position:relative;z-index:0}#wrapper{height:100%}#head_wrapper.s-ps-islite{padding-bottom:370px}#head_wrapper.s-ps-islite .s_form{position:relative;z-index:1}#head_wrapper.s-ps-islite .fm{position:absolute;bottom:0}#head_wrapper.s-ps-islite .s-p-Date,year=date.getFullYear();document.getElementById("year").innerText="@"+year+"Baidu "</script></body></html>
```

在浏览器中打开的效果如下

