实验 1 套接字基础与 UDP 通信

1．实验过程

编写 UDP_Server.py 程序设置服务器地址及端口号，服务器接收 ping 报文，其中服务器程序中使用 random 函数来模拟丢包率，本程序中模拟丢包率为 30%，具体操作为在服务器接收请求时在 1-10 中随机生成一个整数，若该数小于 3 则判定为丢包，相应代码如下：

```
rand = random.randint(0, 10)        # 模拟丢包
if rand < 3:       # 丢包率 30%
    continue
```

在 UDP_Client.py 程序中确定与服务器所对应的地址及端口号，通过循环向服务器发送 10 个 ping 报文，每个 ping 报文包括请求序号及发送时间，内容如下：

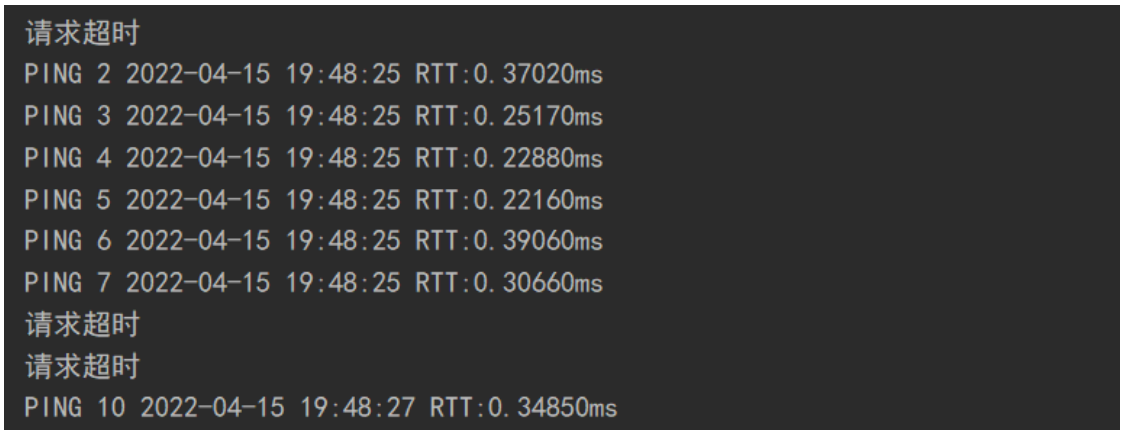
```
message = ('Ping %d %s' % (i+1, time_str))
```

服务器接收到客户端发送的 ping 报文后，通过 upper（）函数将报文转换为大写再发送回客户端。计算每个报文的往返时延 RTT，最高时延，最低时延，平均时延以及丢包率，在计算时延时使用的 time.time()函数读取时间的精度较低，故使用精度更高 time.perf_counter()函数替代，分别读取报文方式时间和接收到服务器反馈时间，差值即为报文往返时延 RTT，代码如下：

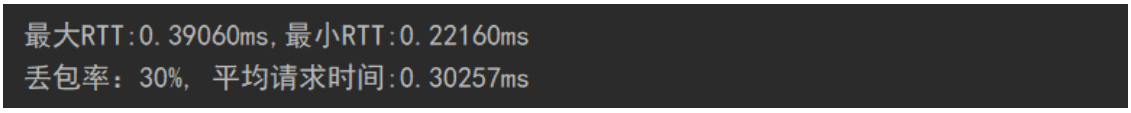
```
start_time = time.perf_counter()
…
end_time = time.perf_counter()
RTT = (end_time - start_time)*1000
```

2. 实验结果

客户端接收到的服务器响应如图 1，服务器成功响应的平均时延，最大时延，最小时延及丢包率计算结果如图 2，可以看出在客户端发送的 10 次 ping 报文中有 3 次服务器接收失败，每次成功的响应消息为均将报文转换为了大写，对应每个报文发送的响应时间及往返时延。



图表 1 UDP 传输结果



图表 2 时延计算及丢包率结果

实验 2 TCP 通信与 Web 服务器

1. 单线程

1.1 实验过程

在 TCP_Server1.py 文件中，编写程序接收客户端的 HTTP 请求，客户端请求成功后返回静态文件，请求失败则返回未找到文件的响应报文。服务器端使用 listen 函数设置服务器的监听数，指在客户端请求访问时的最大排队数，而非最大处理线程数。

由于编写的测试网站含有图片，因此若客户端访问图片需要将图片内容正确返回并显示，需要添加代码如下：

```
f = open('test.jpg', 'rb')
pic_content = b'''
HTTP/1.x 200 OK
Content-Type: image/jpg
'''
pic_content = pic_content+f.read()
f.close()
```

客户端发送请求报文时按照报文标准格式发送，发送内容如下：

```
request = b'GET /1.html HTTP/1.1 Host: 127.0.0.1:12000\r\n\r\n'
```

服务器接收到报文后读取客户端请求访问的相应文件，若文件存在，则服务器通过标准的返回报文格式将客户端请求的文件内容发送给客户端：

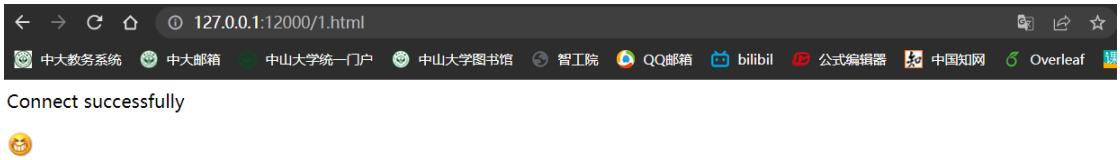
```
with open('.' + file, 'rb') as f:
    data = f.read()
response_line = 'HTTP/1.1 200 OK\r\n'
response_header = 'Server:pwb\r\n\r\n'
response_body = data
response = (response_line + response_header + '\r\n').encode() + response_body
connectionSocket.sendall(response)
```

客户端接收到后则接收数据存入本地文件。此时使用浏览器访问本机服务器页面可以获得相应的文件。若文件不存在则返回“404 NOT FOUND”页面，相应报文内容如下：

```
response_line = 'HTTP/1.1 404 NOT FOUND\r\n'
response_header = 'Server:pwb\r\n'
response_body = '404 NOT FOUND!'
```

1.2 实验结果

使用浏览器访问成功网页如图 3，访问失败的页面图 4。使用编写的 TCP_Client.py 客户端程序进行访问则会生成本地 html 文件。



图表 3 访问成功

图表 4 访问失败

本实验需要注意的是 HTTP 报文格式的书写需要按照标准书写，以确保报文能准确处理返回正确结果。

HTTP 请求报文格式：

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

HTTP 响应报文格式：

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
服务器: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html
```

2. 多线程

TCP_Server1.py 编写的服务器一次只能处理一个 HTTP 请求，在 TCP_Server2.py 文件中编写服务器处理多线程 HTTP 请求的程序，实现同时处理多线程任务。

与单线程处理服务器不同的是，多线程处理服务器需要用到 Threading 模块来创建子进程，将服务器接收 HTTP 请求并发送响应的过程封装为函数 deal_request(clientSocket)，再在主函数中利用 Threading 模块实现多线程的任务处理，步骤重要代码如下:

```
clientSocket, addr = serverSocket.accept()
#  创建子进程
sub_thread = threading.Thread(target=deal_request, args=(clientSocket,))
sub_thread.start()
```

多线程任务处理的服务器在运行时可以同时处理多个 HTTP 请求，访问网页结果与单线程相同。