套接字编程作业 2-1：Web 服务器

本作业要求掌握使用 Python 进行 TCP 套接字编程的基础知识：如何创建套接字，将其绑定到特定的地址和端口，发送和接收 HTTP 分组，以及一些 HTTP 首部格式的基础知识。

在这个编程作业中，你将用 Python 语言开发一个简单的 Web 服务器，它仅能处理一个 HTTP 请求。具体而言，你的 Web 服务器将：（1）当一个客户（浏览器）联系时创建一个 TCP 套接字；（2）从这个 TCP 连接接收 HTTP 请求；（3）解释该请求以确定所请求的特定文件；（4）从服务器的文件系统获得请求的文件；（5）创建一个由请求的文件组成的 HTTP 响应报文，报文前面有首部行；（6）经 TCP 连接向请求的浏览器发送响应。如果浏览器请求一个在该服务器中不存在的文件，服务器应当返回一个“404 Not Found”差错报文。

任务 1、编写 Web 服务器代码

请在“Web 服务器框架代码.py”中标有#Fill-in-start 和#Fill-in-end 的地方填写代码，每个地方都可能需要不止一行代码。

任务 2、运行 Web 服务器

自己编写一份简单的 HTML 文件，放在服务器程序所在的目录中。运行服务器程序。确认运行 Web 服务器的主机的 IP 地址，以及服务器代码中使用的端口号，建议关闭防火墙。

在另一个主机上打开浏览器（建议使用 Chrome 浏览器）并提供相应的 URL，例如：

http://***server_address*** : ***server_port*** / ***filename***

其中：***server_address*** 是 Web 服务器的 IP 地址，如果 Web 服务器和浏览器运行在同一个主机中，则可以是该主机的 IP 地址或 127.0.0.1；***server_port*** 是 Web 服务器正在监听的端口；***filename*** 是被请求对象在服务器上的路径。

然后用客户端尝试获取服务器上不存在的文件，你应该会得到一个“404 Not Found”消息。

任务 3、编写 HTTP 客户端代码

不使用浏览器，编写自己的 HTTP 客户端来测试你的 Web 服务器。客户端将使用一个 TCP 连接用于连接到服务器，向服务器发送 HTTP 请求，并将服务器响应显示出来。你可以假定发送的 HTTP 请求将使用 GET 方法。

要求客户端程序的命令格式：

client.py server_address server_port filename

其中：***client.py*** 是客户端程序文件名；***server_address*** 是 Web 服务器的 IP 地址，如果 Web 服务器和客户端运行在同一个主机中，则可以是该主机的 IP 地址或 127.0.0.1；***server_port*** 是 Web 服务器正在监听的端口；***filename*** 是被请求对象在服务器上的路径。

任务 4、运行 HTTP 客户端

先确定 Web 服务器已运行，然后在 CMD 窗口中输入客户端程序的运行命令，分别请求服务器上存在和不存在的文件，查看运行结果。

任务 5*、编写并运行多线程 Web 服务器（选做）

目前，这个 Web 服务器一次只处理一个 HTTP 请求。请实现一个能同时处理多个请求的多线程服务器。首先创建一个主线程，在固定端口监听客户端请求。当从客户端收到 TCP 连接请求时，它将通过另一个端口建立 TCP 连接，并在另外的单独线程中为客户端请求提供服务。这样在每个请求/响应对的独立线程中将有一个独立的 TCP 连接。

作业提交要求

任务 1~4 必做，任务 5 选做。

将以下内容压缩打包后提交，压缩文件的命名规则为“作业 2-1-学号-姓名”：

1）任务 1 的完整 Web 服务器代码文件 1 份

2）任务 3 的完整 HTTP 客户端代码文件 1 份

3）运行结果报告文件，包含：

- Python 版本、浏览器类型
- Web 服务器运行窗口截图
- 任务 2 的客户端浏览器窗口截图：分别请求服务器上存在的文件和不存在的文件
- 任务 4 的 HTTP 客户端程序运行窗口截图：分别请求服务器上存在的文件和不存在的文件

4）可选提交：任务 5 的多线程服务器代码文件 1 份，以及运行结果报告 1 份（包含验证多线程正确运行的验证方案和结果截图）

Web 服务器框架代码

```
from socket import *
import sys

serverSocket = socket(AF_INET, SOCK_STREAM)

# Prepare a sever socket
#Fill-in-start
#Fill-in-end

while True:
# Establish the connection
print (' The server is ready to receive')

# Set up a new connection from the client
connectionSocket, addr =    #Fill-in-start    #Fill-in-end

try:
        # Receives the request message from the client
        message =    #Fill-in-start    #Fill-in-end
        # Extract the path of the requested object from the message
        # The path is the second part of HTTP header, identified by [1]
        filename = message.split()[1]

        # Because the extracted path of the HTTP request includes
        # a character '/', we read the path from the second character
        f = open(filename[1:])

        # Store the entire contenet of the requested file in a temporary buffer
        outputdata =    #Fill-in-start    #Fill-in-end

        # Send the HTTP response header line to the connection socket
        #Fill-in-start
        #Fill-in-end

        # Send the content of the requested file to the connection socket
        for i in range(0, len(outputdata)):
                connectionSocket.send(outputdata[i].encode())

        # Close the client connection socket
        connectionSocket.close()

except IOError:
        # Send HTTP response message for file not found
        #Fill-in-start
        #Fill-in-end

        # Close the client connection socket
        #Fill-in-start
        #Fill-in-end

serverSocket.close()

# Terminate the program after sending the corresponding data
sys.exit()
```

套接字编程作业 2-2：UDP ping 程序

本作业要求掌握使用 Python 进行 UDP 套接字编程的基础知识：如何使用 UDP 套接字发送和接收数据报，以及如何设置适当的套接字超时，并熟悉 Ping 应用程序及其在计算统计信息（如丢包率）中的作用。

在这个编程作业中，你需要研究一个用 Python 编写的简单的 ping 服务器程序，然后实现对应的客户端程序。该客户端程序将向 ping 服务器发送简单的 ping 报文，接收服务器返回对应的响应报文，并确定从该客户发送 ping 报文到收到响应报文为止的时延。该时延称为往返时延（RTT）。这些 ping 客户端程序和服务器程序提供的功能类似于现代操作系统中可用的标准 ping 程序功能，但是本作业的程序使用更简单的 UDP 协议，而不是标准 ping 程序使用的互联网控制消息协议（ICMP）。

任务 1、学习 ping 服务器代码

“UDPPingerServer.py”是 Ping 服务器的完整代码。

ping 服务器程序在一个无限循环中监听到来的 UDP 数据报。UDP 为应用程序提供了不可靠的传输服务，报文可能因为路由器队列溢出、硬件错误或其他原因而在网络中丢失。因此，ping 服务器程序使用一个随机整数来模拟网络丢包的影响。当数据报到达时，服务器将报文中的内容转为大写，并仅当生成的随机整数大于或等于 4 时才将其发送回客户端。

任务 2、编写 ping 客户端代码

根据 ping 服务器代码，编写一个 ping 客户端代码，向 ping 服务器发送简单的 ping 报文，接收服务器返回对应的响应报文，并确定从该客户发送 ping 报文到收到响应报文为止的时延。该时延称为往返时延（RTT）。

ping 客户端程序需具有如下功能：

1）客户端程序的命令格式：

client.py server_address server_port

其中：***client.py*** 是客户端程序文件名；***server_address*** 是 ping 服务器的 IP 地址，如果 ping 服务器和客户端运行在同一个主机中，则可以是该主机的 IP 地址或 127.0.0.1；***server_port*** 是 ping 服务器正在监听的端口。

2）客户端向服务器发送 10 个 ping 报文，每个报文的内容由以下格式的 ASCII 字符组成：

Ping ***sequence_number send_time***

其中：***sequence_number*** 是 ping 报文的序号（1~10）；***send_time*** 是以年月日

时分秒表示发送 ping 报文的时间。

3）因为 UDP 是不可靠的协议，所以从客户端发送到服务器的分组可能在网络中丢失。因此，客户端不能无限期地等待响应报文。客户端程序等待服务器回应的时间至多为 1 秒（请查找 Python 文档，以了解如何在 UDP 套接字上设置超时值）。

如果在 1 秒内没有收到服务器返回的响应报文，则客户端程序应该假定分组在网络传输期间丢失，并显示如下信息：

Request timed out

4）客户端收到服务器返回的响应报文时将显示如下信息：

Reply from ***server_address***: ***ping_message***, RTT=***rtt_value*** ms

其中：***server_address*** 是发送响应报文的 ping 服务器的 IP 地址；***ping_message*** 是响应报文的内容；***rtt_value*** 是以毫秒（ms）为单位的往返时间。

5）客户端要统计并显示丢包率（百分比），以及最小、最大和平均 RTT。

任务 3*、编写 UDP Heartbeat 客户端和服务器（选做）

UDP Ping 的另一个类似应用是 UDP Heartbeat（心跳）。心跳可用于检查应用程序是否已启动并运行，并报告单向丢包。客户端在 UDP 数据报中将一个序列号和当前时间戳发送给正在监听客户端心跳的服务器。服务器收到 UDP 数据报后，计算时差，报告丢包（若发生）。如果心跳数据报在指定的一段时间内丢失，就可以假设客户端应用程序已经停止。实现 UDP Heartbeat 客户端和服务器，你需要修改任务 2 中编写的 UDP ping 客户端代码和任务 1 中给出的 UDPPingerServer.py 代码。

作业提交要求

任务 1 和 2 必做，任务 3 选做。

将以下内容压缩打包后提交，压缩文件的命名规则为“作业 2-2-学号-姓名”：

1）任务 2 的完整 ping 客户端代码文件

2）运行结果报告文件，包含：

- Python 版本
- 任务 2 的 ping 客户端程序运行窗口截图

3）可选提交：任务 3 的 UDP Heartbeat 客户端和服务器代码文件各 1 份，以及运行结果报告 1 份（包含客户端和服务器程序运行窗口截图）

Ping 服务器的完整代码（UDPPingerServer.py）

```
# We will need the following module to generate randomized lost packets import random
from socket import *
import random

# Create a UDP socket
# Notice the use of SOCK_DGRAM for UDP packets
serverSocket = socket(AF_INET, SOCK_DGRAM)
# Assign IP address and port number to socket
serverSocket.bind(('', 12000))

while True:
    # Generate random number in the range of 0 to 10
    rand = random.randint(0, 10)
    # Receive the client packet along with the address it is coming from
    message, address = serverSocket.recvfrom(1024)
    # Capitalize the message from the client
    message = message.upper()
    # If rand is less is than 4, we consider the packet lost and do not respond
    if rand < 4:
        continue
    # Otherwise, the server responds
    serverSocket.sendto(message, address)
```

套接字编程作业 2-3：邮件客户

本作业要求了解 SMTP 协议，并使用 Python 实现 SMTP 标准协议。Python 提供了一个名为 smtplib 的模块，它内置了使用 SMTP 协议发送邮件的方法。但是我们不会在本实验中使用此模块，因为它隐藏了 SMTP 和套接字编程的细节。

在这个编程作业中，你要开发一个简单的邮件客户端软件，将邮件发送给任意收件人。你的邮件客户端将需要通过 TCP 连接到邮件服务器，使用 SMTP 协议与邮件服务器进行交互，经该邮件服务器向某接收方发送一封电子邮件，最后关闭与该邮件服务器的 TCP 连接。

为了限制垃圾邮件，一些邮件服务器不接受来源随意的 TCP 连接，建议使用你的大学邮件服务器和流行的 Webmail 服务器（如 QQ 或 163 邮件服务器）。

任务 1、编写邮件客户端代码

请在“SMTP 客户端框架代码.py”中标有#Fill-in-start 和#Fill-in-end 的地方填写代码，每个地方都可能需要不止一行代码。

框架代码中的“mailserver”变量值是发件人邮箱的 SMTP 服务器域名（学校邮箱为 mail.std.uestc.edu.cn，QQ 邮箱为 smtp.qq.com，163 邮箱为 smtp.163.com），SMTP 默认端口号是 25。

QQ 邮箱和 163 邮箱默认关闭 SMTP 服务，需要在设置中打开 SMTP 服务。另外，QQ 邮箱和 163 邮箱在打开 SMTP 服务后，会设置一个授权码，在程序使用这个授权码作为密码（代码中的“password”变量）登录，而不是平时使用的密码。

任务 2、运行邮件客户端代码

在命令行窗口运行你编写的邮件客户端程序代码，查看服务器返回的每条消息，其中包含每次操作后返回的状态码。

同时，登录发件人邮箱和收件人邮箱，在发件人的已发送文件夹中和收件人的收件箱中查看这封被发送的邮件。

在某些情况下，收件人邮件服务器可能会将你的电子邮件分类为垃圾邮件。如果你在收件人的收件箱中没有看到这封电子邮件，请检查垃圾邮件文件夹。

任务 3*、提供附件发送支持（选做）

任务 1 实现的 SMTP 邮件客户端只能在电子邮件正文中发送文本消息。修改你的邮件客户端代码，使其可以发送包含文本文件和图像文件的电子邮件。

作业提交要求

任务 1 和 2 必做，任务 3 选做。

将以下内容压缩打包后提交，压缩文件的命名规则为“作业 2-3-学号-姓名”：

1）任务 1 的完整邮件客户端代码文件，提交时请用***替换发件人邮件账户口令

2）运行结果报告文件，包含：

- Python 版本
- 根据任务 2 中邮件客户端程序运行中服务器返回的消息，画出本作业实现的邮件客户端与发件人邮箱的 SMTP 邮件服务器之间的交互时序图
- 任务 2 的邮件客户端程序运行窗口截图
- 任务 2 中发件人邮箱的已发送文件夹中这封被发送的邮件截图
- 任务 2 中收件人邮箱的收件箱中这封被发送的邮件截图。

3）可选提交：任务 3 的邮件客户端代码文件 1 份，以及运行结果报告 1 份（包含客户端和服务器程序运行窗口截图）

SMTP 客户端框架代码

```
from socket import *
import base64

# Mail content
subject = "I love computer networks!"
contenttype = "text/plain"
msg = "I love computer networks!"
endmsg = "\r\n.\r\n"

# Choose a mail server (e.g. Google mail server) and call it mailserver
mailserver =    #Fill-in-start    #Fill-in-end

# Sender and reciever
fromaddress =    #Fill-in-start    #Fill-in-end
toaddress =    #Fill-in-start    #Fill-in-end

# Auth information (Encode with base64)
username =      #Fill-in-start    #Fill-in-end
password =    #Fill-in-start    #Fill-in-end

# Create socket called clientSocket and establish a TCP connection with mailserver
#Fill-in-start
#Fill-in-end

recv = clientSocket.recv(1024) .decode()
print(recv)

# Send HELO command and print server response.
#Fill-in-start
#Fill-in-end

# Send AUTH LOGIN command and print server response.
clientSocket.sendall('AUTH LOGIN\r\n'.encode())
recv = clientSocket.recv(1024).decode()
print(recv)

clientSocket.sendall((username + '\r\n').encode())
recv = clientSocket.recv(1024).decode()
print(recv)

clientSocket.sendall((password + '\r\n').encode())
recv = clientSocket.recv(1024).decode()
print(recv)

# Send MAIL FROM command and print server response.
#Fill-in-start
#Fill-in-end

# Send RCPT TO command and print server response.
#Fill-in-start
#Fill-in-end

# Send DATA command and print server response.
#Fill-in-start
#Fill-in-end
```

```
# Send message data.
#Fill-in-start
#Fill-in-end

# Message ends with a single period and print server response.
#Fill-in-start
#Fill-in-end

# Send QUIT command and print server response.
#Fill-in-start
#Fill-in-end

# Close connection
clientSocket.close()
```

套接字编程作业 2-4：多线程 Web 代理服务器

本作业要求了解 Web 代理服务器的工作原理及其基本功能之一：缓存。

在这个编程作业中，你将开发一个能够缓存网页的小型 Web 代理服务器。这是一个很简单的代理服务器，它只能理解简单的 GET 请求，但能够处理各种对象——不仅仅是 HTML 页面，还包括图片。这个代理服务器将是多线程的，使其在相同时间能够处理多个请求。

通常，当浏览器发出一个请求时，请求将被直接发送到 Web 服务器。然后 Web 服务器处理该请求并将响应消息发回给浏览器。如图 1 所示，为了提高性能，我们在浏览器（Client）和 Web 服务器（Web Server）之间建立一个代理服务器（Proxy Server）。现在，浏览器发送的请求消息和 Web 服务器返回的响应消息都要经过代理服务器。换句话说，浏览器通过代理服务器请求对象，代理服务器将浏览器的请求转发到 Web 服务器。然后，Web 服务器将生成响应消息并将其传递给代理服务器，代理服务器又将其发送给浏览器。

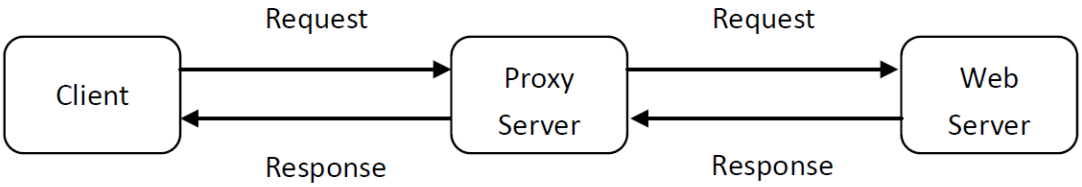

图 1 代理服务器原理

任务 1、编写 Web 代理服务器代码

请在“Web 代理服务器框架代码.py”中标有#Fill-in-start 和#Fill-in-end 的地方填写代码，每个地方都可能需要不止一行代码。

这份框架代码只能从服务器代理缓存大小不超过 4K 的文件，且不支持 HTTPS。

任务 2、运行代理服务器

使用命令行模式运行你编写的代理服务器程序。

从你的浏览器发送一个网页请求，将 IP 地址和端口号指向代理服务器，例如：

http:// *proxy_server_address* : *proxy_server_port* / *webpage_url*

其中：*proxy_server_address* 是代理服务器的 IP 地址，如果 Web 服务器和浏览器运行在同一个主机中，则可以是 localhost 或 127.0.0.1；*proxy_server_port* 是代理服务器程序中使用的端口号；*webpage_url* 是访问网页的 URL。

由于前大多数的网站均采用 HTTPS，且网页中大多会有超过 4K 大小的图片等文件，

所以在测试验证你的代理服务器程序时，建议 ***webpage_url*** 使用以下非 HTTPS 访问的网页 URL：

gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html

gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html

gaia.cs.umass.edu/wireshark-labs/alice.txt

浏览器自身也有缓存，所以测试中有时需要清空浏览器的缓存！

你还可以配置你的浏览器以使用你的代理服务。但是由于浏览器自身会发起一些后台请求，而本作业中的代理服务器不支持 HTTPS，且不能正常代理网页中超过 4K 的文件，因而会使得这些后台请求失败。虽不影响浏览器的运行，但是如果你的代理服务器代码没有做好异常处理故，则会影响代理服务器程序的调试和正常运行。故不建议配置浏览器的代理服务器选项来使用你的代理服务。

浏览器的代理服务器选项配置取决于浏览器。如果设置了浏览器的代理服务器选项，那么在浏览器的 URL 地址栏中就只需提供访问页面的 URL 地址即可，例如：

http:// gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html

任务 3*、提供其他功能（选做）

修改任务 1 中编写的 Web 代理服务器代码，尝试提供以下部分或所有功能：

3.1 支持 HTTP POST 方法

当前代理服务器只支持 HTTP GET 方法，请通过添加请求体来增加对 POST 的支持。

3.2 提供缓存有效性检验

每当客户端发出特定请求时，典型的代理服务器会缓存网页。本作业的代理服务器已提供了缓存的基本功能：当代理获得一个请求时，首先检查请求的对象是否已经在缓存中；如果是，则从缓存返回对象，从而不用联系服务器；如果对象未被缓存，则代理从服务器获取该对象，向客户端返回该对象，并缓存一个拷贝以备将来的请求。

在实际环境下，代理服务器必须验证被缓存的响应是否仍然有效，并且能对客户端正确响应。你可以在 RFC 2068 中阅读有关缓存及其在 HTTP 中实现方式的更多细节。（提示：HTTP 首部行中的 Last-Modified 和 Last-Modified-Since，以及 304 状态码。建议使用谷歌浏览器。）

3.3 可代理缓存网页中大小超过 4K 的文件

HTTP 首部行中的 Content-Length 用于描述 HTTP 消息实体传输长度。请找出任务 1 编写的代码中不能正常代理网页中超过 4K 文件的相关代码，尝试利用 Content-Length 修改完善你的代码。

3.4 支持 HTTPS

请自行查找资料，并利用 Wireshark 抓包等手段，学习了解 HTTPS 的工作原理。尝试修改完善你的代码以支持 HTTPS。

作业提交要求

任务 1 和 2 必做，任务 3 选做。

将以下内容压缩打包后提交，压缩文件的命名规则为“作业 2-4-学号-姓名”：

1）完整的 Web 代理服务器代码文件

2）运行结果报告文件，包含：

- Python 版本、浏览器类型
- 任务 2 中的 Web 代理服务器运行窗口截图
- 任务 2 中的客户端浏览器窗口截图：包括代理服务器上有和没有请求对象缓存的两种情况
- 任务 2 中代理服务器上包含缓存文件的文件夹截图

3）可选提交：任务 3 的服务器代码文件（可以每项功能单独 1 份代码文件，也可以在 1 份代码文件实现多个功能），以及运行结果报告 1 份（包含所实现的每项功能的验证结果截图）

Web 代理服务器框架代码

```
from socket import *
import threading
import os

# Define thread process
def Server(tcpClisock, addr):

    BUFSIZE = 1024
    print('Received a connection from:', addr)
    data = #Fill-in-start    #Fill-in-end
    print(data)

    if len(data):
        # Extract the filename from the received message
        getFile = data.split()[1]
        print('getFile:',getFile)

        # Form a legal filename
        filename =    #Fill-in-start    #Fill-in-end
        print('filename:',filename)

        # Check wether the file exist in the cache
        if os.path.exists(filename):
            print('File exist')
            # ProxyServer finds a cache hit and generates a response message
            f = open(filename,"r")
            CACHE_PAGE = f.read()
            # ProxyServer sends the cache to the client
            #Fill-in-start
            #Fill-in-end
            print('Send the cache to the client')
            tcpClisock.close()
        else:
            print('File not exist')
            # Handling for file not found in cache
            # Create a socket on the ProxyServer
            c =    #Fill-in-start    #Fill-in-end
            try:
                # Connect to the WebServer socket to port 80
                hostn = getFile.partition("/")[2].partition("/")[0]
                #Fill-in-start
                #Fill-in-end
                print('Connect to',hostn)

                # Some information in client request must be replaced
                # before it can be sent to the server
                #Fill-in-start
                #Fill-in-end

                # Send the modified client request to the server
                #Fill-in-start
                #Fill-in-end
```

```
                    # Read the response into buffer
                    buff = c.recv(4096)
                    print('recvbuff len:',len(buff))

                    # Send the response in the buffer to client socket
                    tcpClisock.send(buff)
                    print('Send to client\r\n')
                    # Create a new file to save the response in the cache
                    tmpFile = open("./" + filename,"w")
                    #Fill-in-start
                    #Fill-in-end
                except:
                    print("Illegal request")
                tcpClisock.close()

# Main process of    ProxyServer
if __name__ == '__main__':
    # Create a server socket, bind it to a port and start listening
    tcpSersock = socket(AF_INET, SOCK_STREAM)
    #Fill-in-start
    #Fill-in-end

    print("Ready to serve.....\n")
    while True:
        tcpClisock, addr = tcpSersock.accept()
        thread = threading.Thread(target=Server, args=(tcpClisock, addr))
        thread.start()
    tcpSersock.close()
```

套接字编程作业 5-1：ICMP ping 程序

本作业要求理解 ICMP 协议，掌握使用 ICMP 请求和响应消息实现 Ping 程序。

Ping 是一个流行的网络应用程序，用于测试 IP 网络中的某个特定主机是否可达。它也可用于测量客户主机和目标主机之间的网络延迟。它的工作过程是：向目标主机发送 ICMP“回显（Echo）”报文（即 ping 分组），并监听 ICMP“回显响应（Echo reply）”应答（有时也称为 pong 分组）。ping 程序测量往返时间（RTT），记录分组丢失，并计算多个 ping-pong 交换的统计汇总（往返时间的最小值、最大值、平均值和标准差）。

在这个编程作业中，你的任务是使用原始套接字（RAW Socket）开发自己的 Ping 程序。你的程序将使用 ICMP，但为了保持简单，将不完全遵循 RFC 1739 中的官方规范。请注意，你只需要编写 ping 客户端程序，因为服务器侧所需的功能内置于所有的操作系统中。

任务 1、编写 ping 客户端代码

请在“ping 客户端框架代码.py”中标有#Fill-in-start 和#Fill-in-end 的地方填写代码，每个地方都可能需要不止一行代码。

你的 ping 程序需具有如下功能：

1）ping 程序的命令格式：

client.py destination_host [–n count]

其中：***client.py*** 是客户端程序文件名；***destination_host*** 是目标主机的域名地址；**–n *count*** 选项用来指定发送的 ping 分组个数，如不使用该选项则默认发送 4 个 ping 分组。

“ping 客户端框架代码.py”框架代码已给出了完整的 ping 命令解析代码。

2）Ping 程序发送的 ping 分组之间间隔大约一秒钟，每个 ping 分组的 ICMP 数据字段携带一个时间戳。每个 ping 分组发送完后，程序最多等待一秒，用于接收响应。

如果一秒后没有收到响应，那么应假设 ping 分组或 pong 分组在网络中丢失（或者目的主机已关闭），并显示如下信息：

Request timed out

如果收到目的主机返回的 pong 分组，则显示如下信息：

Reply from ***ip_address***: bytes=***data_bytes*** time=***rtt***ms TTL=***TTL_value***

其中：***ip_address*** 是发送响应报文的主机 IP 地址；***data_bytes*** 是响应报文中 ICMP 数据字段的字节长度；***rtt*** 是以毫秒（ms）为单位的往返时间（RTT），***TTL_value*** 是 pong 分组的 IP 首部中的生存时间（TTL）字段值。

框架代码中提供了发送一个 ping 分组的完整代码——“sendOnePing”方法，以及

计算 ICMP 报文校验和的完整代码——“checksum”方法。在接收一个 pong 分组的“receiveOnePong”方法中，你需要提取 ICMP 分组首部中的报文类型、校验和、ID 等信息来检查接收的 ICMP 分组是否有效；此外还需要提取 IP 分组首部中的 TTL、长度以及 ICMP 分组数据字段，以生成 ping 程序要显示的信息。

请在尝试完成“receiveOnePong”方法之前，先仔细研究“sendOnePing”方法和“checksum”方法。特别注意网络字节序问题！

3）ping 程序发送和接收完所有 ping-pong 分组后，要统计并显示（格式自定义）以下信息：

- 已发送的 ping 分组数量、已接收的 pong 分组数量、丢包数量和丢包率；
- 以毫秒（ms）为单位的最小、最大和平均 RTT。

任务 2、测试 ping 客户端程序

首先，通过发送分组到本地主机来测试你的客户端，目的主机地址是 127.0.0.1。

然后，分别 Ping 位于国内的 2 个目的主机和位于国外的 2 个目的主机，以测试你的 ping 程序。

这个作业要求使用原始套接字，因此在某些操作系统中，可能需要管理员/root 权限才能运行你的 Ping 程序。

任务 3*、解析并显示 ICMP 响应错误（选做）

任务 1 编写的 Ping 程序只能检测收到 ICMP 回显响应报文或没有收到响应。请修改 Ping 程序，解析 ICMP 的目的不可达错误代码，并向用户显示相应的错误结果。教材图 5-19 给出了常见的 ICMP 响应错误码。

作业提交要求

任务 1 和 2 必做，任务 3 选做。

将以下内容压缩打包后提交，压缩文件的命名规则为“作业 5-1-学号-姓名”：

1）任务 1 的完整 ping 客户端代码文件

2）运行结果报告文件，包含：

- Python 版本
- 任务 2 的 ping 客户端程序运行窗口截图

3）可选提交：任务 3 的 ping 客户端代码文件 1 份，以及运行结果报告 1 份（包含客户端运行窗口截图）

ping 客户端框架代码

```
from socket import *
import os
import sys, getopt
import struct
import time
import select
import binascii

ICMP_ECHO_REQUEST = 8

def checksum(string):
    csum = 0
    countTo = (len(string) // 2) * 2
    count = 0

    while count < countTo:
        thisVal = string[count] * 256 + string[count+1]
        csum = csum + thisVal
        csum = csum & 0xffffffff
        count = count + 2

    if countTo < len(string):
        csum = csum + string[len(string) - 1]
        csum = csum & 0xffffffff

    csum = (csum >> 16) + (csum & 0xffff)
    csum = csum + (csum >> 16)
    answer = ~csum
    answer = answer & 0xffff
    answer = answer >> 8 | (answer << 8 & 0xff00)
    return answer

def receiveOnePong(mySocket, destAddr, ID, sequence, timeout):
    timeLeft = timeout

    while 1:
        startedSelect = time.time()
        whatReady = select.select([mySocket], [], [], timeLeft)
        howLongInSelect = (time.time() - startedSelect)
        if whatReady[0] == []: # Timeout
            return None
        timeReceived = time.time()
        recPacket, addr = mySocket.recvfrom(1024)

        #Fill in start

        #Fetch the ICMP header from the IP packet

        #Fill in end

        timeLeft = timeLeft - howLongInSelect
        if timeLeft <= 0:
            return None
```

```
def sendOnePing(mySocket, destAddr, ID, sequence):
    # Header is type (8), code (8), checksum (16), id (16), sequence (16)

    myChecksum = 0
    # Make a dummy header with a 0 checksum
    # struct -- Interpret strings as packed binary data
    header = struct.pack("!BBHHH", ICMP_ECHO_REQUEST, 0, myChecksum, ID, sequence)
    data = struct.pack("!d", time.time())

    # Calculate the checksum on the data and the dummy header.
    myChecksum = checksum(header + data)

    # Get the right checksum, and put in the header
    if sys.platform == 'darwin':
        # Convert 16-bit integers from host to network byte order
        myChecksum = htons(myChecksum) & 0xffff
    else:
        myChecksum = htons(myChecksum)

    header = struct.pack("!BBHHH", ICMP_ECHO_REQUEST, 0, myChecksum, ID, sequence)
    packet = header + data

    mySocket.sendto(packet, (destAddr, 1)) # AF_INET address must be tuple, not str
    # Both LISTS and TUPLES consist of a number of objects
    # which can be referenced by their position number within the object.

def doOnePing(destAddr, ID, sequence, timeout):
    icmp = getprotobyname("icmp")

    # SOCK_RAW is a powerful socket type. For more details:
    #http://sock-raw.org/papers/sock_raw
    #Fill in start

    #Create Socket here

    #Fill in end

    sendOnePing(mySocket, destAddr, ID, sequence)
    rtt = receiveOnePong(mySocket, destAddr, ID, sequence, timeout)

    mySocket.close()
    return rtt

def ping(dest, count):
    # timeout=1 means: If one second goes by without a reply from the server,
    # the client assumes that either the client's ping or the server's pong is lost
    timeout = 1

    myID = os.getpid() & 0xFFFF    # Return the current process i
    loss = 0

    # Send ping requests to a server separated by approximately one second
    for i in range(count) :
        result = doOnePing(dest, myID, i, timeout)

        #Fill in start
```

```
            #Print response information of each pong packet:
            #No pong packet, then display "Request timed out."
            #Receive pong packet, then display "Reply from host_ipaddr : bytes=… time=… TTL=…"

            #Fill in end

            time.sleep(1)# one second

        #Fill in start

        #Print Ping statistics

        #Fill in end

        return

if __name__ == "__main__":
        if len(sys.argv) < 2:
                print('IcmpPing.py dest_host [-n <count>]')
                sys.exit()
        host = sys.argv[1]
        try:
                dest = gethostbyname(host)
        except:
                print('Can not find the host "%s". Please check your input, then try again.'%(host))
                exit()

        count = 4
        try:
                opts, args = getopt.getopt(sys.argv[2:], "n:")
        except getopt.GetoptError:
                print('IcmpPing.py dest_host [-n <count>]')
                sys.exit(2)
        for opt, arg in opts:
                if opt == '-n':
                        count = int(arg)

        print("Pinging " + host + " [" + dest + "] using Python:")
        print("")
        ping(dest, count)
```