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()
```