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。

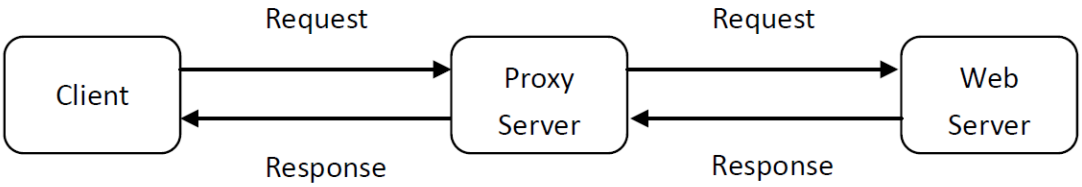

图 1 代理服务器原理

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()
```