

Computer Network Homework3

20161190 이민재

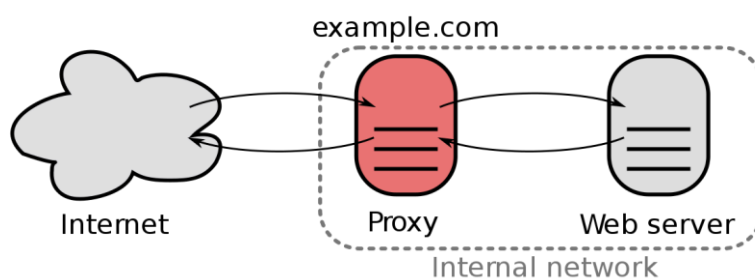
1. Introduction / Enviroment

My homework is implementing an HTTP proxy that can enable I/O multiplexing and can store cache. I setting web proxy (HTTP) at network setting (web proxy server is 127.0.0.1 and port number is 8123). I performed using python source code at same port number with network setting. My execution environment is as follows.

- Source code : Python 2.7.16
- OS : MacBook Pro(13-inch, 2017, Two thunderbolt 3 ports)
- Processor : 2.3GHz dual core intel Core i5
- Memmory : 8GB 2133 MHZ LPDDR3
- Graphic : Intel Iris Plus Graphics 640 1536 MB
- Library : socket, os, sys, urlparse, select

2. Concept of proxy server

Proxy server is server that forward client request to the remote server and return response to the client. Proxy server is used for security reasons and efficiency reasons. Security reason is to prevent client from directly accessing to origin server. In terms of efficiency, Proxy server has the information that client frequently requests. Then if client request information that has already proxy server, proxy server doesn't request to origin server and directly response to client. This concept is cache. Whenever I send a request to a new server. I saved the cache by creating a new file in the proxy.py folder. And when the same request came in, it printed out that it hit the cache, and the response result was also printed.



Proxy server also can I/O multiplexing. I make my proxy server handle multiple requests simultaneously. These features make clients convenient.

3. Explanation of functions

Before explaining the source code in detail, I summarize representative socket-related functions.

1) `socket(socket.AF_INET, socket.SOCK_STREAM)`

This function can be used to create socket objects. In order to perform networking using sockets in the same way as servers or clients, it is necessary to create sockets first. This function receives two factors, one family and the other type.

`Socket.AF_INET` is intended to represent an IPv4 address scheme, and `socket.SOCK_STREAM` means creating a socket using TCP.

2) `setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)`

Sockets can adjust details by manipulating socket options, and for this purpose, a `setsockopt()` function can be used. When a server program using a socket is operated, there are cases where it is forcibly terminated or abnormally terminated. For testing purposes, it is especially often necessary to force termination, and if the program is forced to end and executed again, the existing program was terminated, but the kernel still maintains bind information with abnormal termination. This may often be associated with timeout in a TCP connection, which usually disappears after a minute, but is inserted to prevent connection delays during that time.

3) `bind("127.0.0.7", 8080)`

In general, server sockets use fixed port numbers. And the client's connection request is accepted with that port number. Therefore, the operating system must bind the socket and the port number to make the server socket use a specific port number, and the API used at this time is `bind()`. Currently, "127.0.0.7" means that the IP address of the local host and 8080 means the port number.

4) listen(5)

When bind() is finished, it is a function for making the client wait for connection. The listen() function receives the number of clients to be requested for connection as a parameter. In this case, it means that it will allow connection to five clients.

5) connect(host, port)

The client uses socket.connect() to connect to the server, and the factor used at this time is the same as bind(). This method ends when a connection is established, and there is no value to return. Since the client must be connected to the server, receive the local host address through sys and enter the same port number previously created on the server side.

6) accept()

This function is used for connection-oriented sockets. It takes the first connection from the queue where the connections that have not yet been processed are waiting and creates a new connection socket. And assign a file designator pointing to the socket and return it. In this case, the information on the socket, the IP address, and the Port number are returned.

7) close()

When data transmission/reception is no longer required, use it to close the socket.

8) select()

It is very inefficient when multiple sockets are created and clients are received. As the number of sockets increases, the waiting time for the client to receive the response increases. However, using select() can solve this problem. Select() is a function used to check a change in a designated socket. I was waiting for a change in the socket set. When the socket performs any action, all other sockets except the operated socket are removed and the corresponding socket is proceeded.

4. Implementation Details

```
1  import sys
2  import os
3  from urllib.parse import urlparse
4  from socket import *
5  import select
```

Sys was exported for interpreter control. OS was imported to control the OS and store cache comfortably in multiple paths. For example A function such as getcwd() indicating the current path may be used. In order to conveniently parse url, urlparse was imported. Finally for make my proxy server handle multiple requests simultaneously, i import select module.

```
6
7  def sendRequest(url):
8      path = url.path
9      hostname = url.netloc
10     clientTcpSocket = socket(AF_INET, SOCK_STREAM)
11     clientTcpSocket.connect((hostname,80))
12     requestString = f"GET {path} HTTP/1.0\r\nHost:{hostname}\r\n\r\n"
13     clientTcpSocket.send(requestString.encode())
14     resp = clientTcpSocket.recv(1 << 20)
15     return resp
16
```

Parse the parameter url and get hostname. Then, make clientTcpSocket using socket() function and connect with (hostname, port number 80). Encode the requestString and sent. Then receive and store at response. Finally return response.

```
1  def main():
2      tcpServerSocket = socket(AF_INET,SOCK_STREAM)
3      tcpServerSocket.bind(("",8321))
4      tcpServerSocket.listen(10)
5
6      tcpServerSocket2 = socket(AF_INET,SOCK_STREAM)
7      tcpServerSocket2.bind(("",8322))
8      tcpServerSocket2.listen(10)
9
10     tcpServerSocket3 = socket(AF_INET,SOCK_STREAM)
11     tcpServerSocket3.bind(("",8323))
12     tcpServerSocket3.listen(10)
13
14     read_socket_list = [tcpServerSocket, tcpServerSocket2, tcpServerSocket3]
15     print("Sockets created")
```

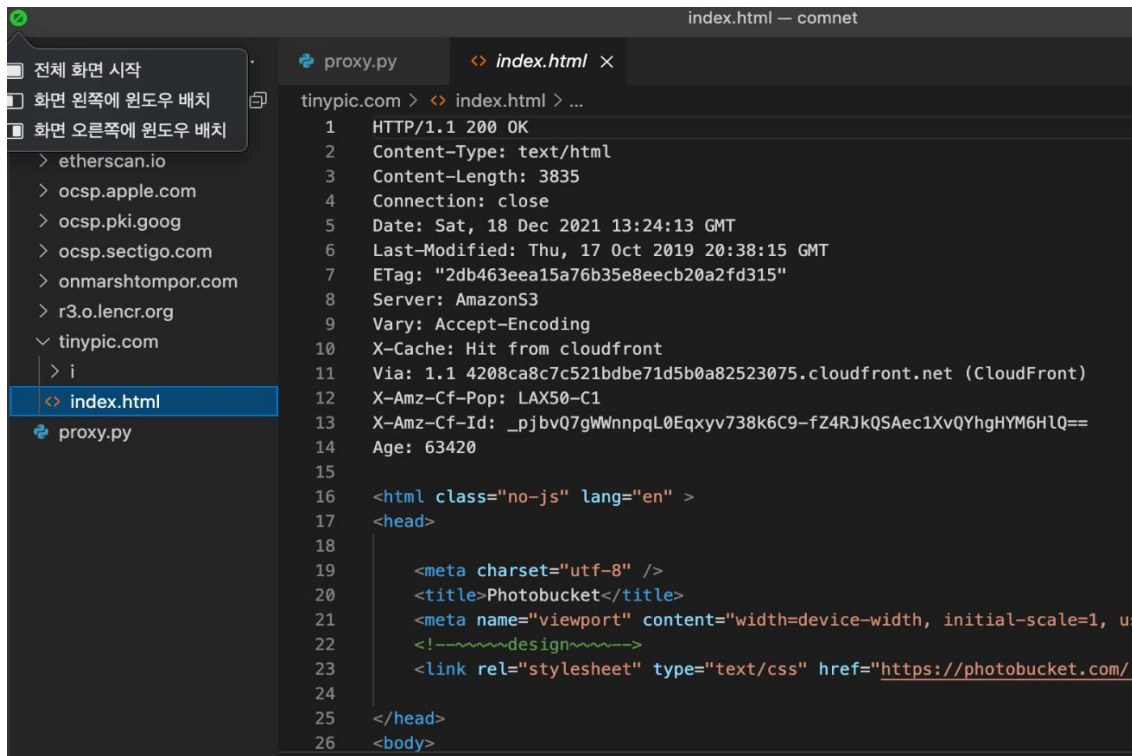
For multiple requests simultaneously. I make 3 tcp server socket. Using socket() function create socket, and bind address to socket. Then, using listen() function ready to get client's request. Making socket list by these socket 1,2,3.

```
while True:
    conn_read_socket_list, conn_write_socket_list, conn_except_socket_list = select.select(read_socket_list, [], [])
    for conn_read_socket in conn_read_socket_list:
        if conn_read_socket == tcpServerSocket:
            connectionSocket, addr = tcpServerSocket.accept() #return access client information(ip,portnum)
            message = connectionSocket.recv(1 << 20).decode()
            cwd = os.getcwd() #current working directory
            print(message)
            urlString = message.split()[1]
            parsedUrl = urlparse(urlString)
            hostname = parsedUrl.netloc
            path = parsedUrl.path
            if path == "/" or path == "" :
                path = "/index.html"
            cachedFilePath = os.path.join(cwd,hostname,path[1:])
```

Using select() make my proxy server handle multiple requests simultaneously. At conn_read_socket in conn_read_socket_list, if conn_read_socket == tcpServerSocket then implement code, and elif conn_read_socket == tcpServerSocket2 implement code, ... I use 3 socket for multi I/O. Using accept(), wait for receive access client's socket and address(ip, port number). Receive message and parse by urlparse. And check current working directory and set path for cache information.

```
50  try:
51      print("preopen")
52      cachedFile = open(cachedFilePath, 'rb')
53      print("post open")
54      print("pre cache read")
55      connectionSocket.send(cachedFile.read())
56      cachedFile.close()
57
58      print("post cache read")
59      print('Sent back the cached data')
60
61  except:
62      response = sendRequest(parsedUrl)
63      print(f"response: {response}")
64      if not os.path.exists(os.path.dirname(cachedFilePath)):
65          os.makedirs(os.path.dirname(cachedFilePath))
66      with open(cachedFilePath, 'wb') as cachedFile:
67          cachedFile.write(response)
68      connectionSocket.send(response)
69      connectionSocket.close()
70
```

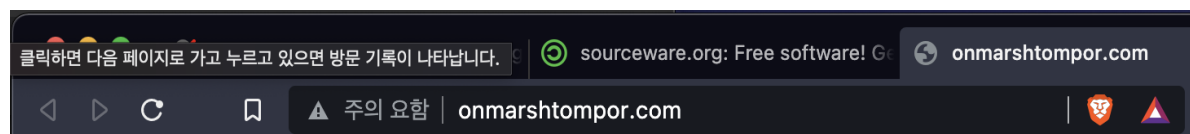




```
index.html — comnet
proxy.py index.html x
tinypic.com > index.html > ...
1 HTTP/1.1 200 OK
2 Content-Type: text/html
3 Content-Length: 3835
4 Connection: close
5 Date: Sat, 18 Dec 2021 13:24:13 GMT
6 Last-Modified: Thu, 17 Oct 2019 20:38:15 GMT
7 ETag: "2db463eea15a76b35e8eeb20a2fd315"
8 Server: AmazonS3
9 Vary: Accept-Encoding
10 X-Cache: Hit from cloudfront
11 Via: 1.1 4208ca8c7c521bdbe71d5b0a82523075.cloudfront.net (CloudFront)
12 X-Amz-Cf-Pop: LAX50-C1
13 X-Amz-Cf-Id: _pjbvQ7gWwnpqL0Eqxyv738k6C9-fZ4RJkQSAec1XvQYhgHYM6H1Q==
14 Age: 63420
15
16 <html class="no-js" lang="en" >
17 <head>
18
19 <meta charset="utf-8" />
20 <title>Photobucket</title>
21 <meta name="viewport" content="width=device-width, initial-scale=1, us
22 <!--~~~~~design~~~~~-->
23 <link rel="stylesheet" type="text/css" href="https://photobucket.com/r
24
25 </head>
26 <body>
```

If you access the http where the cache exists, the above results come out. Because there is already a cache, it does not request the origin server, but immediately request the information in the cache to the client.

If you look at the tinypic.com folder, the index.html file is stored. It stores well the html format as we learned.



```
empty OK

GET http://onmarshtomp.com/favicon.ico HTTP/1.1
Host: onmarshtomp.com
Proxy-Connection: keep-alive
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/604.4664.45 Safari/537.36
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8,en;q=0.7
Sec-GPC: 1
Referer: http://onmarshtomp.com/
Accept-Encoding: gzip, deflate
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7

preopen
response: b'HTTP/1.1 204 No Content\r\nServer: nginx\r\nDate: Sun, 18 Dec 2021 13:24:13 GMT\r\nExpires: Thu, 31 Dec 2037 23:55:55 GMT\r\nCache-Control: public, must-revalidate, proxy-revalidate\r\n\r\n'
```

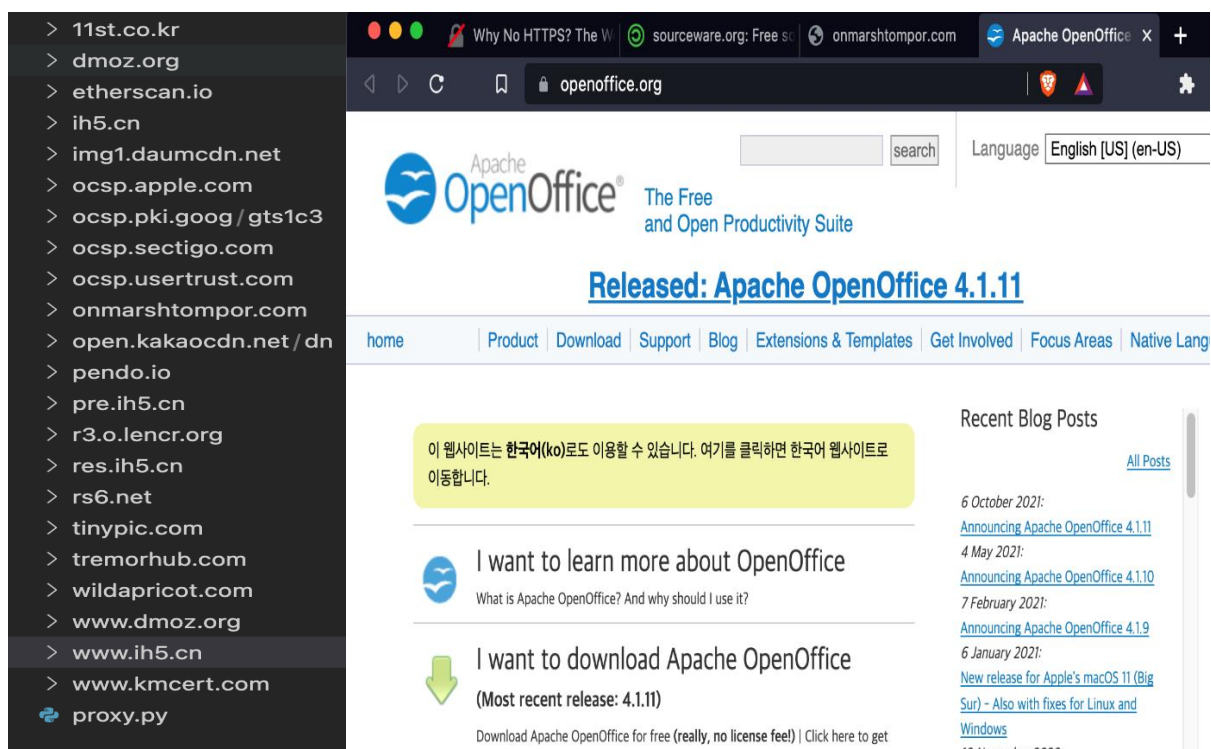
```

GET http://onmarshtomp.com/ HTTP/1.1
Host: onmarshtomp.com
Proxy-Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 6.0.4664.45 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml,application/signed-exchange;v=b3;q=0.9
Sec-GPC: 1
Accept-Encoding: gzip, deflate
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7

preopen
post open
pre cache read
post cache read
Sent back the cached data

```

The http that the assignment pdf offer site “whynohttps.com site”, there was also a homepage that was state as above “empty ok”. This case was also well stored in the cache and reacted well when cache was hit when it was miss.



The above picture shows that cavities are stored in several current working directories. It was also confirmed that it was possible to open multiple tabs at the same time.

I will finish the result part by attaching the results of some tests below.

