Computer Network Project 2

2015147531_서기원

1. Introduction (Software Environment)

1-1. Linux Version

```
giwon@giwon-virtual-machine:~/Desktop/CSNetwork/pj2$ cat /proc/version
Linux version 5.4.0-48-generic (buildd@lcy01-amd64-010) (gcc version 9.3.0 (Ubuntu 9.3.0-10ubuntu2)
) #52-Ubuntu SMP Thu Sep 10 10:58:49 UTC 2020
giwon@giwon-virtual-machine:~/Desktop/CSNetwork/pj2$ uname -a
Linux giwon-virtual-machine 5.4.0-48-generic #52-Ubuntu SMP Thu Sep 10 10:58:49 UTC 2020 x86_64 x86
_64 x86_64 GNU/Linux
```

1-2. Ubuntu Version

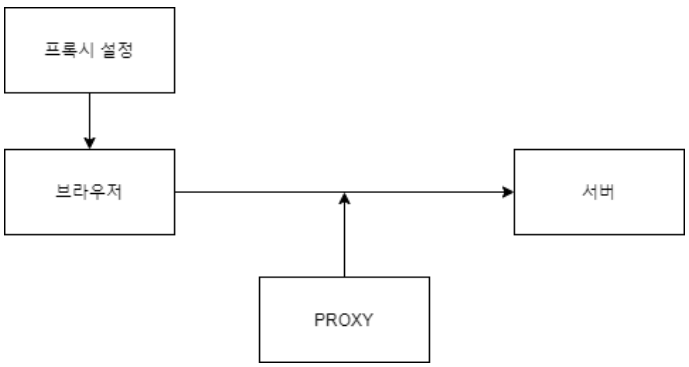
```
giwon@giwon-virtual-machine:~/Desktop/CSNetwork/pj2$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04.1 LTS"
```

1-3. Programming Language : PYTHON 3.8.5

```
giwon@giwon-virtual-machine:~/Desktop/CSNetwork/pj2$ python3 --version
Python 3.8.5
```

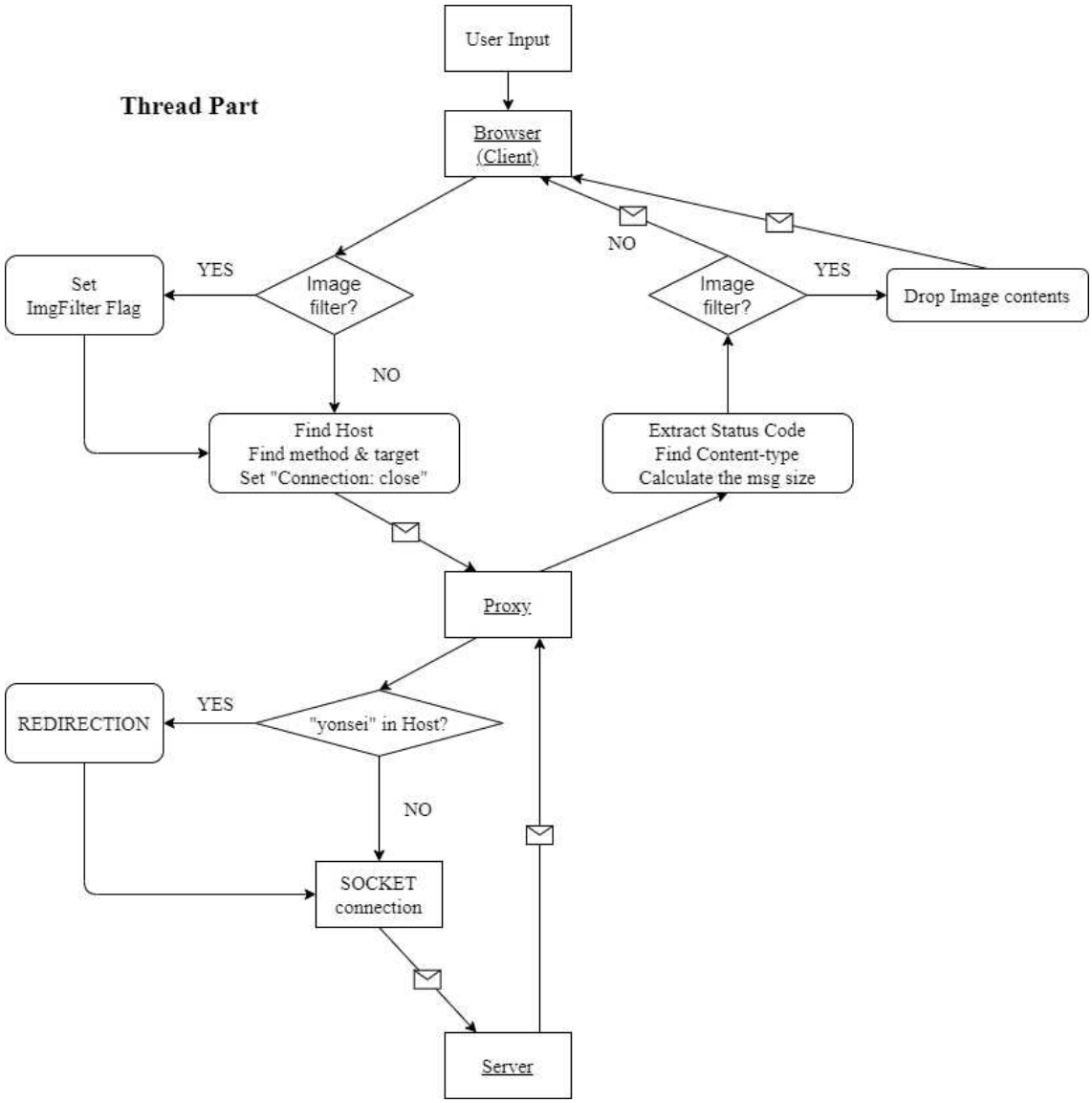
1-4. FireFox Proxy 설정

2. 과제 수행 Flow Chart



클라이언트 역할을 하는 브라우저에서 서버로 통신을 하는 것을 프록시 서버를 생성하여 중간 매개체 역할을 할 수 있게끔 하는 것이 이번 과제의 내용이었다.

3. 프로그램 Flow Chart



4. Detail explanation

```
import socket
import sys
import threading
```

우선 파이썬 기본 라이브러리에 속한 socket, sys, threading 라이브러리를 사용한다.

```
# gloabl variable
global threadNum, taskNum, imagE_filter, lock
```

Thread 번호, 시행 중인 Task 번호, 이미지 필터링 플래그와 락 변수를 전역변수로 사용한다.

```python
def main(Port):
    startString = "Starting proxy server on port %d."%(Port)
    print(startString)

    # Create Socket
    proxySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    proxySocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    proxySocket.bind(("", Port))
    proxySocket.listen()

    while True:
        try:
            # if client access, return socket
            client_socket, addr = proxySocket.accept()
            receive_thread = threading.Thread(target=connect_client, args=(client_socket, addr))
            receive_thread.daemon = True
            receive_thread.start()
        # handle keyboard interrupt
        except KeyboardInterrupt:
            proxySocket.close()
            print("\nexit")
            return
```

해당 파일의 메인 함수로 이번 과제의 클라이언트 역할을 하는 브라우저와 통신하는 데 필요한 clientSocket을 생성하고 송수신을 스레드를 생성하여 각각에 역할을 배정한다. “ctrl + C”인 KeyboardInterrupt를 try except 구문을 통해 예외처리의 형태로 프로그램의 종료를 설계하였다.

```python
# thread function
def connect_client(client_socket, addr):
    global taskNum, image_filter
    url_filter = False

    # set thread id
    lock.acquire()
    t_id = checkThreadNum()
    lock.release()
```

Thread 시작 시 Thread Number를 부여하는 부분이다. 다른 스레드들과 id가 중복되면 안 되기 때문에 lock을 적용하여 고유 id를 부여한다.

```python
threadNum = [0 for i in range(300)]
```

```python
def checkThreadNum():
    global threadNum
    for i in range(300):
        if threadNum[i] == 0:
            threadNum[i] = 1
            return i + 1
    return 302
```

스레드의 개수가 300을 넘지 않는다고 가정을 하고 앞에서부터 가장 먼저 나오는 0의 index를 id로 부여한다. ID가 1부터 시작하기 때문에 1을 더하여 값을 부여한다.

```
# get request from the client
data = client_socket.recv(8000)
# parsing request
if_get, httpV, cli_req, cli_host, agent, cli_handled_data = handle_request(data)
```

client_socket으로부터 request를 받는 부분이다. request의 길이가 최대 8000을 넘지 않을 것이라는 가정을 하고 버퍼의 크기를 8000으로 설정하였다. 그 이후 다음 과정에서 필요한 변수들을 정의하기 위해 handle_request() 함수를 호출한다.

```
def handle_request(byte_data):
    global image_filter
    str_data = "".join([chr(b) for b in byte_data])
    if_get = False
    # Remove BODY from requestMSG
    withoutBody = str_data[:str_data.find('\r\n\r\n')]
    # Start line : http method, request target, http version
    start_line = withoutBody.split('\r\n')[0]
    httpV = start_line[start_line.find(' HTTP/'):]                    # http version
    req = start_line[:start_line.find(' HTTP/')]                     # method + target
    # check if the method is GET, and whether image filter should be on
    if "GET" in req:
        if_get = True

    if "?image_off" in req:
        image_filter = True
    elif "?image_on" in req:
        image_filter = False

    # request Header
    header = withoutBody[withoutBody.find('\r\n')+2:withoutBody.find('\r\n\r\n')]
    header_split = header.split('\r\n')
    host = "Host field Not Found"
    agent = "Agent field Not Found"
    # find host, agent and make connection close
    for i in range(len(header_split)):
        if "Host:" in header_split[i]:
            host = header_split[i][header_split[i].find(':')+2:]
        elif "User-Agent:" in header_split[i]:
            agent = header_split[i][header_split[i].find(':')+2:]
        elif "Connection:" in header_split[i]:
            header_split[i] = "Connection: close"
        elif "Proxy-Connection:" in header_split[i]:
            header_split[i] = "Connection: close"
    # print(header_split)
    # wrap up the data
    handled_data = start_line + "\r\n"
    for i in range(len(header_split)):
        handled_data += (header_split[i] + "\r\n")
    handled_data += "\r\n"
    return if_get, httpV, req, host, agent, handled_data
```

socket으로 받은 데이터는 바이트 코드이기 때문에 이를 string 타입으로 변환하여 해석하였다.

헤더의 각 부분은 “\r\n”으로 나뉘는 것을 확인할 수 있었고, 바디와 헤더의 구분 또한 “\r\n”으로 이루어지기 때문에 헤더를 추출하기 위해 “\r\n\r\n”을 기준으로 split하였고, request가 “GET” method인지를 파악하여 bool 변수 if_get을 설정하고 host, target 등을 파악하여 저장하고 persistent connection을 방지하기 위해 “Connection: keep-alive”를 “Connection: close”로 설정한다. 이 과정에서 request에 “?image_off” 또는 “?image_on”이 포함되어 있다면 값에 따라 image_filter를 설정한다.

```
# redirection flag
if "yonsei" in cli_host:
    url_filter = True
# if url filter on, change the host and request target
if url_filter and if_get:
    srv_req, srv_host, srv_handled_data = url_handle(cli_handled_data)
else:
    srv_host = cli_host
    srv_req = cli_req
    srv_handled_data = cli_handled_data
```

url_filter의 적용 유무를 확인하는 부분이다. “yonsei”가 Host에 포함되어 있을 경우 url_filter 플래그를 설정한다. “GET” 메소드에 대해서만 url_filter를 적용하면 되기 때문에 bool 변수 if_get의 값에 따라 “www.linuxhowtos.org” 에 해당하는 request로 바꾸기 위해 url_handle() 함수를 호출한다.

```
def url_handle(str_data, http_version):
    start_line = "GET http://www.linuxhowtos.org/" + http_version
    req = "GET http://www.linuxhowtos.org"
    header = str_data[str_data.find('\r\n')+2:str_data.find('\r\n\r\n')]
    header_split = header.split('\r\n')
    for i in range(len(header_split)):
        if "Host:" in header_split[i]:
            host = "www.linuxhowtos.org"
            header_split[i] = "Host: www.linuxhowtos.org"
            break
    handled_data = start_line + "\r\n"
    for i in range(len(header_split)):
        handled_data += (header_split[i] + "\r\n")
    handled_data += "\r\n"
    return req, host, handled_data
```

request의 경우 첫 start line에 해당하는 부분과 host만 다른 것을 확인할 수 있었다. 따라서 “GET http://www.linuxhowtos.org/ (HTTP version)” 부분과 “Host: www.linuxhowtos.org” 부분으로 request를 수정하여 리턴한다. 출력 결과를 위해 req, host를 리턴한다.

```
try:
    serverHost = socket.gethostbyname(srv_host)
except:
    client_socket.close()
    return
```

접속을 요청할 ip주소를 알아내기 위해 socket 라이브러리의 gethostbyname()함수를 사용하였다. 함수의 실패를 처리하기 위해 try except문으로 구성하였다.

```
# set socket which connects my proxy and the destination server
srvSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
srvSocket.connect((serverHost, 80))
srvSocket.send(srv_handled_data.encode())
```

서버와 연결하는 소켓을 생성하여 위에서 가공한 request 데이터를 인코딩하여 전달한다.

```
while True:
    loopT += 1
    msg = srvSocket.recv(4096)
    if not msg:
        client_socket.close()
        cli_close = "[CLI disconnected]\n"
        break
    dataSize += len(msg)
    if loopT == 1:
        status, content_type, response_header = handle_msg(httpV, msg)
        srv_prx = "[CLI --- PRX <== SRV]\n"
    if image_filter and ("image/" in content_type):
        client_socket.send(response_header.encode())
        client_socket.close()
        cli_close = "[CLI disconnected]\n"
        break
    client_socket.send(msg)
```

서버가 보내는 response를 받아 client로 다시 보내주는 부분이다. response의 크기는 알 수 없기 때문에 While문으로 response가 끝날 때까지 받을 수 있게 하였다. 더 이상 받을 메시지가 없다면 client로 보낼 메시지 또한 없기 때문에 client 소켓을 종료한다. dataSize 변수에 메시지의 크기를 저장하고 response 헤더 부분에서의 정보를 파악하기 위해 첫 루프에서 handle_msg() 함수를 호출한다. image_filter가 적용되었을 경우, content_type이 “image/”를 포함하는 경우 body를 드랍하여 헤더만 client로 전달하여 client는 해당 요청을 응답받은 것으로 생각하게끔 하고 연결을 종료한다.

```
def handle_msg(http_version, byte_data):
    str_data = "".join([chr(b) for b in byte_data])
    # Remove BODY from responseMSG
    withoutBody = str_data[:str_data.find('\r\n\r\n')]
    res = withoutBody[:withoutBody.find('\r\n')]
    status = res[len(http_version):]
    from_content = withoutBody[withoutBody.find('Content-Type: '):]
    content_type = from_content[14:from_content.find('\r\n')]
    return status, content_type, withoutBody
```

서버로부터 받은 response를 request의 경우와 마찬가지로 string type으로 변환하여 body를 제거하고 status와 content_type을 추출한다. image_filter가 적용된 경우 헤더를 client에게 전달하기 위해 body를 제거한 헤더 부분 또한 리턴한다.

```
lock.acquire()
if if_get:
    if image_filter:
        result = "------------------------------------------------\n" + str(taskNum) + connection1 + print_filter(image_filter, url_filter)
        print(result)
    else:
        result = "------------------------------------------------\n" + str(taskNum) + connection1 + print_filter(image_filter, url_filter) + "\n" + co
        print(result)
threadNum[t_id-1] = 0
taskNum += 1
lock.release()
```

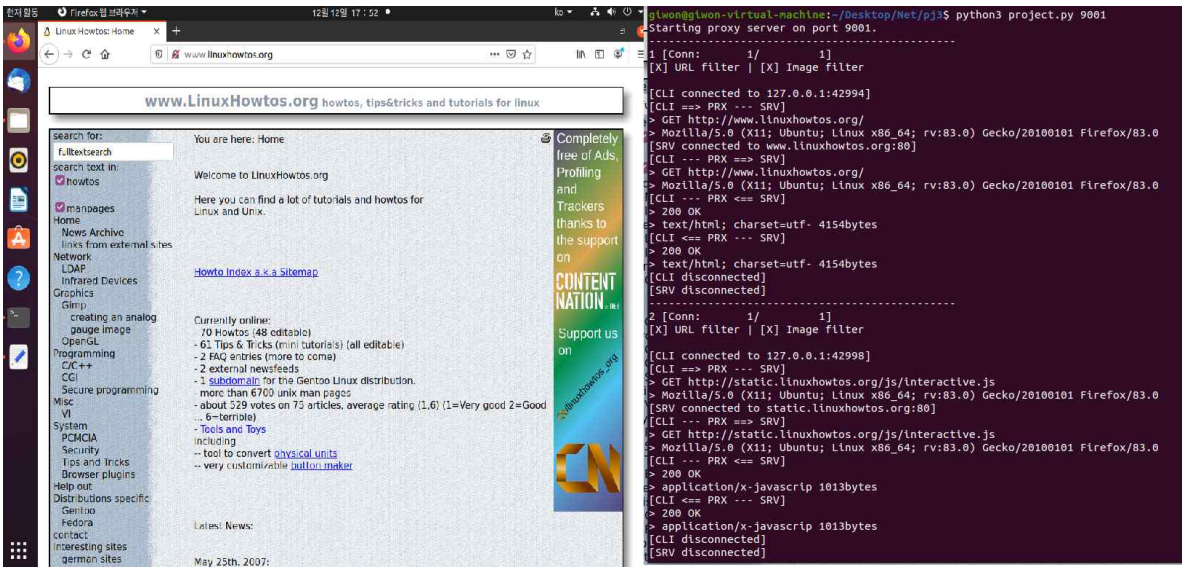
결과 출력 부분이다. 각 실행 과정에서 저장된 string들을 thread 내 모든 작업이 끝난 이후 마지막에 모두 모아 출력한다. task 넘버가 중복되지 않도록 lock을 적용한다.

5. Results

1) Normal

url_filter : x / image_filter : x

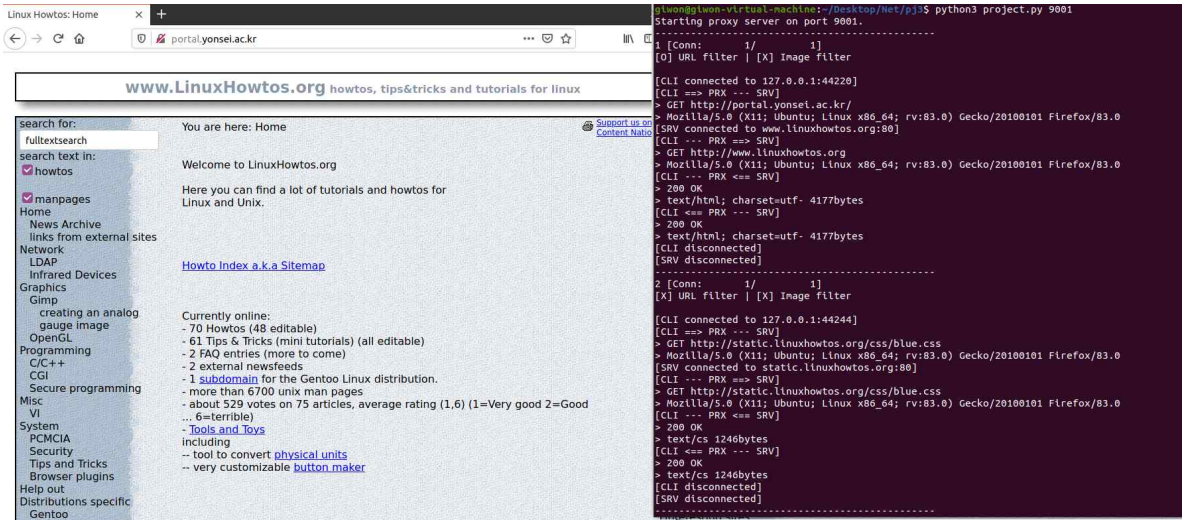
input : http://www.linuxhowtos.org



2) URL 필터

url_filter : O / image_filter : x

input : http://portal.yonsei.ac.kr

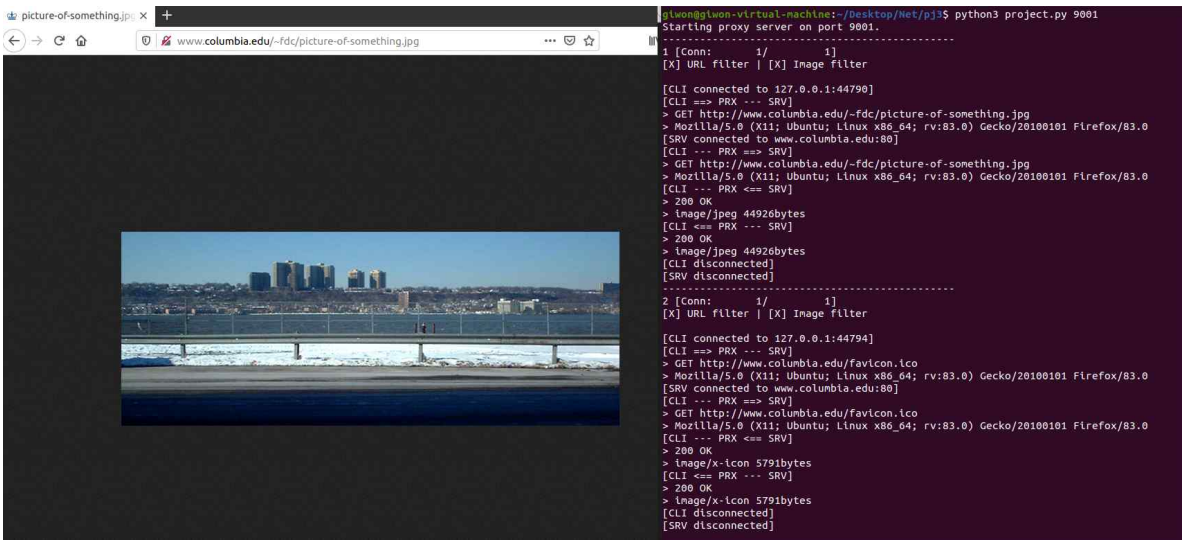


3) IMAGE 필터

- 필터 적용 전후로 비교하겠습니다.

url_filter : x / image_filter : x

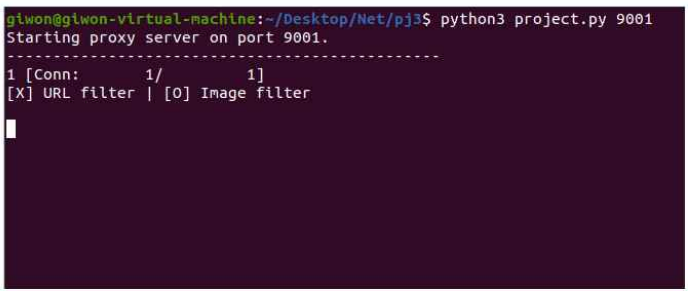
input : http://www.columbia.edu/~fdc/picture-of-something.jpg



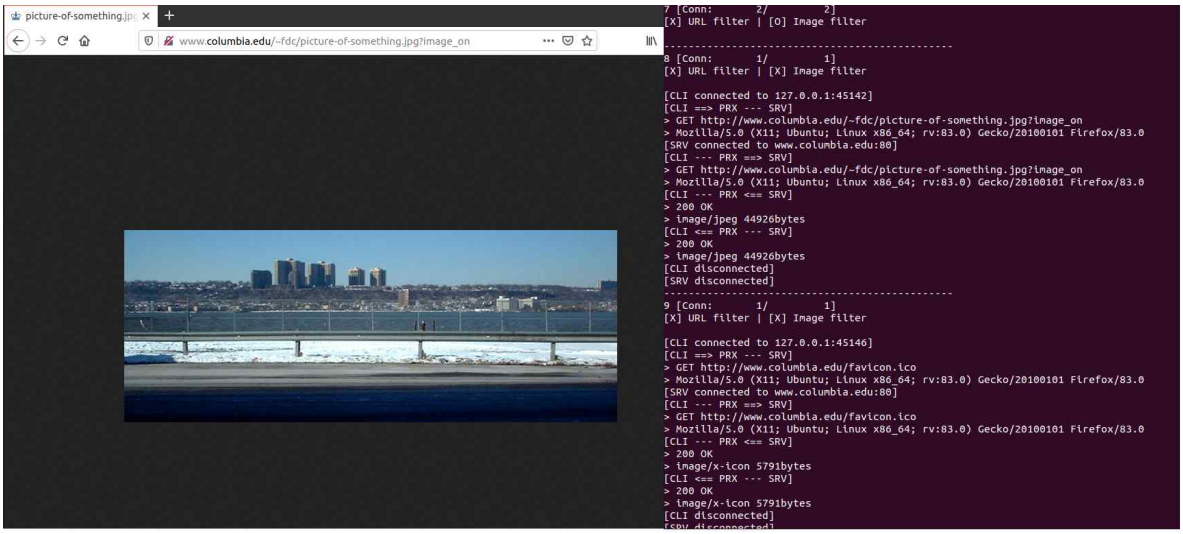
※ image를 출력하였다가 image_off를 실행하게 되면 캐시로 인해 올바른 결과가 나오지 않았습니다. 따라서 올바른 결과를 도출하기 위해 실행 전 Firefox의 모든 캐시 및 기록을 제거하였습니다.

url_filter : x / image_filter : x

input : http://www.columbia.edu/~fdc/picture-of-something.jpg?image_off



이후 다시 ?image_on을 적용시킨 결과



6. Multi-thread vs. Single

- 시간 측정을 위해 파이썬 표준 datetime라이브러리를 사용하였습니다.
- www.linuxhowtos.org에 접속하는 기준으로 측정하였습니다.
- 25번째 통신이 끝날 때 통신이 완료되는 것을 확인하여 25번째 실행시간을 기준으로 비교하였습니다.
- ps –ef로 PID를 조회하였습니다.
- top –p [PID] -b –d [TIME] > [FILE].txt 로 메모리 사용량을 조회하였습니다.

1) Single

```
-----------------------------------------------
25 [Conn:       1/         0]
[X] URL filter | [X] Image filter

[CLI connected to 127.0.0.1:49648]
[CLI ==> PRX --- SRV]
> GET http://www.linuxhowtos.org/favicon.ico
> Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
[SRV connected to www.linuxhowtos.org:80]
[CLI --- PRX ==> SRV]
> GET http://www.linuxhowtos.org/favicon.ico
> Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
[CLI --- PRX <== SRV]
> 200 OK
> image/x-ico 2108bytes
[CLI <== PRX --- SRV]
> 200 OK
> image/x-ico 2108bytes
[CLI disconnected]
[SRV disconnected]
exectution time : 0:00:24.875776
```

```
giwon@giwon-virtual-machine: ~/Desktop/Net/pj3
root       14105       2  0 18:54 ?        00:00:00 [kworker/0:1-events]
root       14106       2  0 18:54 ?        00:00:00 [kworker/0:2-cgroup_destroy]
root       14280       2  0 18:57 ?        00:00:00 [kworker/2:1-events]
root       14281       2  0 18:57 ?        00:00:00 [kworker/3:6]
giwon      14283     959 10 18:57 ?        00:00:02 /usr/lib/firefox/firefox -new-window
giwon      14338   14283  4 18:57 ?        00:00:01 /usr/lib/firefox/firefox -contentproc -childID 1 -isForBr
giwon      14387   14283  2 18:57 ?        00:00:00 /usr/lib/firefox/firefox -contentproc -childID 2 -isForBr
giwon      14432   14283  1 18:57 ?        00:00:00 /usr/lib/firefox/firefox -contentproc -childID 3 -isForBr
giwon      14460     959  0 18:57 ?        00:00:00 /usr/libexec/tracker-store
giwon      14470    1650  1 18:57 pts/0    00:00:00 python3 project.py 9001
giwon      14471   13304  0 18:57 pts/1    00:00:00 ps -ef
giwon@giwon-virtual-machine:~/Desktop/Net/pj3$ top -p 14470 -b -d 3 > single.txt
```

```
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  14470 giwon     20   0   20852  10956   6528 S   0.0   0.3   0:00.02 python3
```

실행 시간 : 24.875776초

메모리 사용 : 10956

2) Multi-thread

```
25 [Conn:       1/         2]
[X] URL filter | [X] Image filter

[CLI connected to 127.0.0.1:50304]
[CLI ==> PRX --- SRV]
> GET http://www.linuxhowtos.org/favicon.ico
> Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
[SRV connected to www.linuxhowtos.org:80]
[CLI --- PRX ==> SRV]
> GET http://www.linuxhowtos.org/favicon.ico
> Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
[CLI --- PRX <== SRV]
> 200 OK
> image/x-ico 2108bytes
[CLI <== PRX --- SRV]
> 200 OK
> image/x-ico 2108bytes
[CLI disconnected]
[SRV disconnected]
exectution time : 0:00:18.960259
```

```
giwon@giwon-virtual-machine: ~/Desktop/Net/pj3
root       14657       2  0 19:01 ?        00:00:00 [kworker/1:1-events]
root       14658       2  0 19:01 ?        00:00:00 [kworker/1:2-events]
root       14659       2  0 19:01 ?        00:00:00 [kworker/1:3-events]
root       14660       2  0 19:01 ?        00:00:00 [kworker/1:4-events]
root       14661       2  0 19:01 ?        00:00:00 [kworker/1:5-events]
giwon      14663     959 18 19:01 ?        00:00:03 /usr/lib/firefox/firefox -new-window
giwon      14716   14663  7 19:01 ?        00:00:01 /usr/lib/firefox/firefox -contentproc -childID 1 -isForBr
giwon      14762   14663  5 19:01 ?        00:00:00 /usr/lib/firefox/firefox -contentproc -childID 2 -isForBr
giwon      14803   14663  2 19:01 ?        00:00:00 /usr/lib/firefox/firefox -contentproc -childID 3 -isForBr
giwon      14828    1650  0 19:01 pts/0    00:00:00 python3 project.py 9001
giwon      14830   13304  0 19:01 pts/1    00:00:00 ps -ef
giwon@giwon-virtual-machine:~/Desktop/Net/pj3$ top -p 14828 -b -d 3 > multi.txt
```

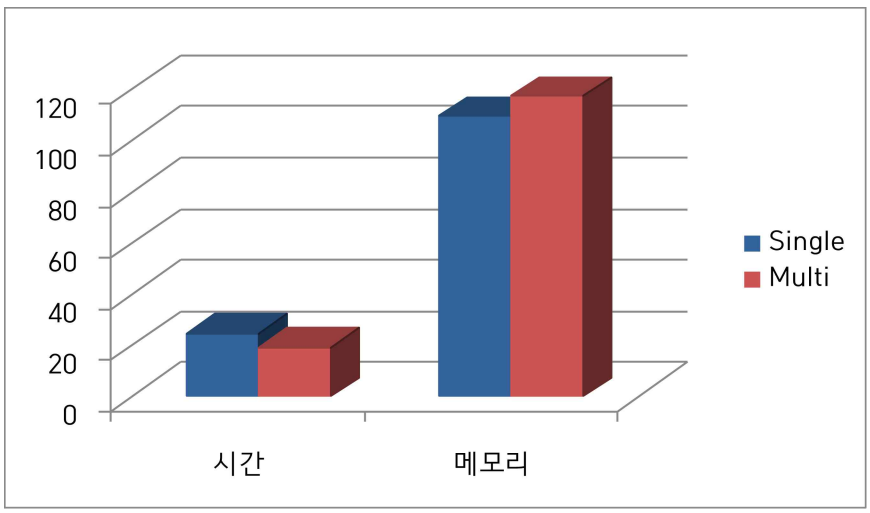
```
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  14828 giwon     20   0 1364324  11736   6848 S   1.3   0.3   0:00.06 python3
```

실행 시간 : 18.960259초

메모리 사용 : 11736

3) 비교

※ 가독성을 위해 단위를 맞추었습니다.



실행 시간 측면에서는 멀티 스레드 방식이 효율적이고 메모리 사용 측면에서는 싱글 프로세스 방식이 효율적인 것을 확인할 수 있었습니다.

7. Reference

Computer Network 2020-2 project 2

https://docs.python.org/ko/3/library/index.html

https://docs.python.org/ko/3/library/socket.html

https://velog.io/@teddybearjung/HTTP-%EA%B5%AC%EC%A1%B0-%EB%B0%8F-%ED%95%B5%EC%8B%AC-%EC%9A%94%EC%86%8C