<컴퓨터 네트워크 Project #3>

과 목	컴퓨터네트워크
담당 교수	한승재 교수님
제출 일자	2020.12.12
전공	산업공학과
학번	2016147008
이름	조성현

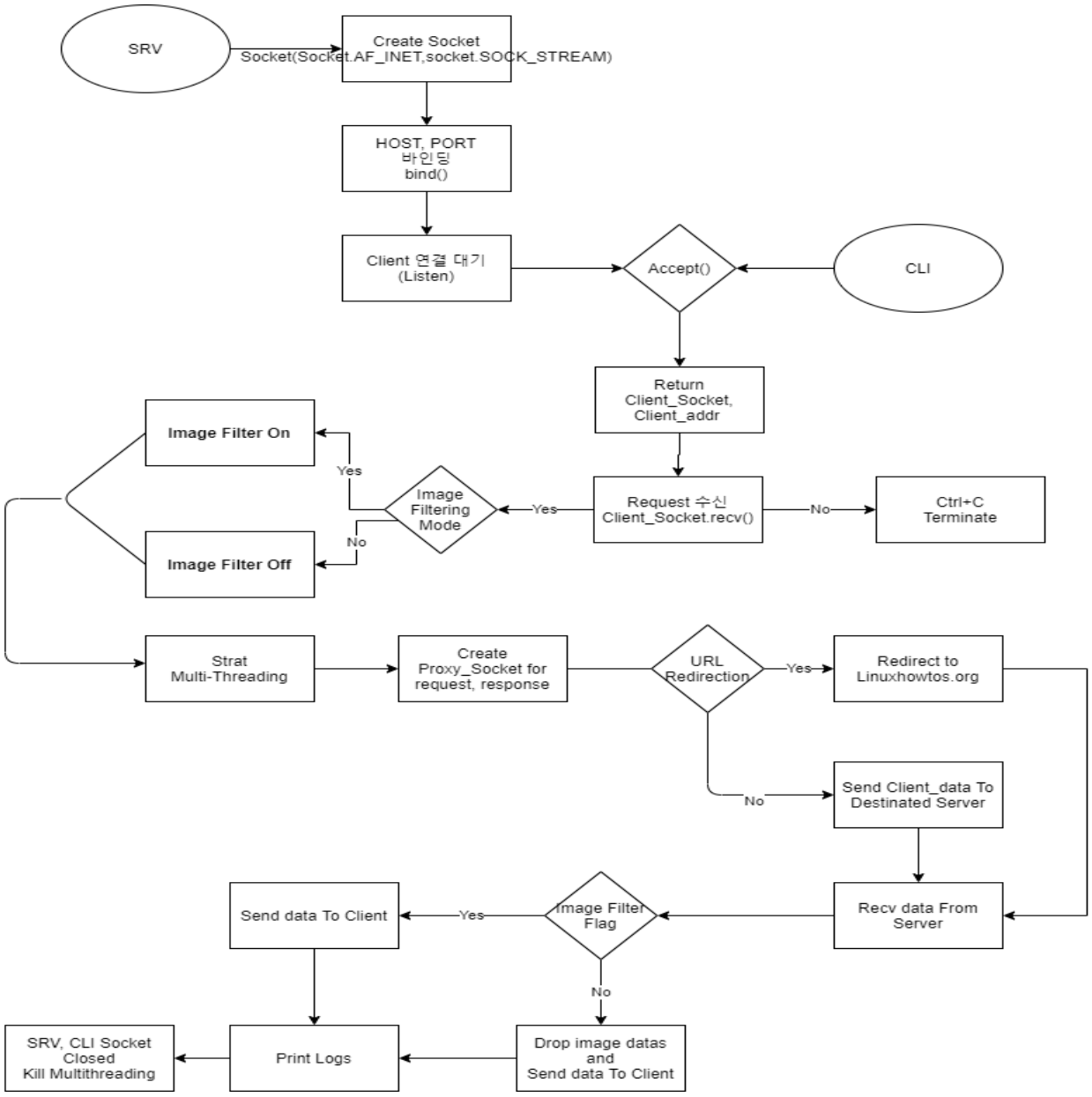
1. Introduction/Reference

- 본 프로젝트에서는 Python(Version: 3.8) 언어를 사용하였다.

- 본 프로젝트의 실행환경은 Ubuntu(Version: 18.04)를 기반으로 한 Linux OS이다.

- 프로젝트에서는 Socket, threading, time, sys 표준 라이브러리를 사용하였다.

2. Flowchart



3. Logical Explanation block by block in detail

```python
import threading
import socket
import sys
import time

buffer_size = 4096
lock = threading.Lock()

# 유저가 입력하는 서버 포트 번호
proxy_port = int(sys.argv[1])


# filtering, redirection을 위한 전역변수
filtered_url = "yonsei"
redirected_url = "http://linuxhowtos.org/"
redirected_server = "linuxhowtos.org"
redirected_host = "linuxhowtos.org"

# filtering, redirection에 대한 flag 변수
image_filtering = {"flag": False}
redirection_flag = {"flag": False}
print_number = 0
```

본 프로젝트에서 사용할 Module을 Import합니다. 사용할 모듈은 Python 표준 라이브러리인 threading, socket, sys, time입니다. 이 후, Socket이 데이터를 받을 때 사용할 Buffer의 크기를 4096으로 설정하고, Threading 과정에서 필요한 lock 변수를 선언했습니다. Port 번호는 sys() 함수를 통해 사용자가 지정할 수 있도록 하였으며, 프로젝트에서 필요한 Image Filtering, URL Redirection을 위한 flag변수 및 Redirection Server에 대한 정보를 생성했습니다. 본 프로젝트에서 URL Redirection은 "Yonsei"라는 단어가 URL에 포함되어 있으면 "linuxhowtos.org"로 redirection을 진행합니다.

Image Filtering, URL Redirection의 flag변수는 초기에 False로 설정하고, 클라이언트의 요청을 판단하여 True로 바꾸거나 다시 False로 바꾸도록 했습니다. 이를 바탕으로 Socket의 recv(), send() 함수를 제어했고, 로그를 출력할 때 Indicator가 flag를 참조하도록 했습니다.

```python
if __name__ == "__main__":
    # 소켓 객체 생성
    # 주소 체계(address family)로 IPv4, 소켓 타입: TCP
    proxy_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print("Starting proxy server on port {}".format(proxy_port))
    print("----------------------------------------")
    # WinError 10048 해결하기 위한 코드
    # Address already in use Error solution
    # future socket available
    proxy_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    # bind 함수는 소켓을 특정 네트워크 인터페이스와 포트 번호에 연결하는데 사용
    # HOST는 hostname, ip address, 빈 문자열 ""이 될 수 있음
    # 빈 문자열이면 모든 네트워크 인터페이스로부터의 접속을 허용
    # port는 1-65536 사이의 숫자를 사용 가능
    proxy_socket.bind(('', proxy_port))
    # accept 함수에서 대기하다가 클라이언트가 접속하면
    # 클라이언트의 소켓과 클라이언트의 주소를 반환해서 저장
    proxy_socket.listen()

    # Main server loop
    while True:
        try:
            client_socket, client_addr = proxy_socket.accept()
            # When a connection arrives from the client, accept it, receive the clients d
            client_data = client_socket.recv(buffer_size)

            # client data가 존재하고
            if client_data:
                # client_data에서 parsing하여 Request Method가 "GET"일 때만 사용
                method = client_data.splitlines()[0].decode(
                    'utf-8').split(' ')[0]
                if method == "GET":
                    client_data = image_filter(client_data)

                    t = threading.Thread(target=p_thread, args=(
                        client_socket, client_addr, client_data))
                    t.daemon = True
                    t.start()

            else:
                pass

        # Ctrl+C -> Terminate
        except KeyboardInterrupt:
            proxy_socket.close()
            sys.exit(1)
    proxy_socket.close()
```

0) 최초 시행되는 Main부분입니다. 메인 Proxy_Socket을 생성하였는데, IPv4, TCP 타입의 소켓을 생성하였고 Reuse를 위해 REUSEADDER를 적용했습니다. 이 소켓은 모든 네트워크에 대해 bind를 할 수 있고 사용자가 입력한 port번호에 연결합니다. 이 후, 클라이언트의 요청을 listen()하는 상태가 되고, 클라이언트의 요청이 들어오면 accept()를 하여 Client_socket, Client_addr(IP, PORT)를 return하고 클라이언트로부터 요청 데이터를 수신합니다. 이 요청이 "GET"인 부분에 대하여 Threading을 진행하고, Ctrl+C를 누르면 전체 종료되도록 했습니다. Threading을 진행하는 p_thread는 아래에 설명되어 있으며, 최초 클라이언트에게 요청받을 때 image_filter()함수를 호출하여 이 요청이 Image에 대한 요청이 있는지의 여부를 판단하도록 했습니다. 자세한 내용은 아래에 각각 명기했습니다.

```python
# header에서 request, user_agent 부분을 parsing하는 함수
def request_parser(header_lines):
    request = ""
    user_agent = ""

    for i in range(len(header_lines)):
        if b'HTTP' in header_lines[i]:
            request = header_lines[i]
        elif b'User-Agent' in header_lines[i]:
            user_agent = header_lines[i]
    data = {
        "request": request.decode('utf-8'),
        "user_agent": user_agent.decode('utf-8')[12:],
    }
    return data


# header에서 server_url과 full_url에 대한 정보를 parsing하는 함수 / domain을 얻기 위함
def domain_parser(header_lines):
    first_line_tokens = header_lines[0].split()
    url = first_line_tokens[1].decode('utf-8')
    http_pos = url.find("://")
    server_url = url[(http_pos+3):]
    full_url = url[(http_pos+3):]
    webserver_pos = server_url.find("/")
    server_url = server_url[:webserver_pos]

    data = {
        "server_url": server_url,
        "full_url": full_url,
    }
    return data
```

프로젝트에서 사용할 함수들에 대해서 설명하도록 하겠습니다.

1) request_parser()함수는 client로부터 요청을 받은 데이터를 line by line으로 분할한 header_lines를 매개변수로 입력받습니다. 이 후, 헤더 정보에서 Request부분과 User_agent정보를 헤더 정보에서 Parsing하여 dictionary에 담아 return하도록 했습니다.

2) domain_parser()함수 또한 header_lines를 매개변수로 입력받습니다. 이 후에 헤더 정보를 decode하여 URL을 추출하고, '://', '/' 부분의 위치를 파악하여 클라이언트가 요청한 서버의 도메인 정보를 Parsing했습니다. 참고로 full_url은 이후에 Image Filtering이나 URL Redirection에 필요할 수 있기 때문에 이에 대한 정보 또한 dictionary에 담아서 return하도록 했습니다.

```python
# image filtering을 위한 함수
def image_filter(client_data):
    global image_filtering
    data = client_data
    lines = data.splitlines()
    full_url = domain_parser(lines)["full_url"]

    # ?image_off라는 query가 있을 때
    if("?image_off" in full_url):
        image_filtering.update(flag=True)

    # ?image_on라는 query가 있을 때
    elif("?image_on" in full_url):
        image_filtering.update(flag=False)

    return data
```

3) Image Filtering을 위한 image_filter()함수입니다. 이 함수는 clinet_data를 매개변수로 입력받습니다. Clinet_data를 line by line으로 분할하고, 위에서 정의한 domain_parser()함수를 통해 full_url을 전달받습니다. full_url에서 '?image_off', '?image_on' query문이 존재한다면 최초에 정의한 image_filtering flag변수를 True, False로 바꾸도록 했습니다.

```python
# URL redirection을 위한 함수 // client의 request에서 request, host 부분을 parsing / change
def redirection(c_data):
    global filtered_url, redirected_url, redirected_host
    client_data = c_data
    lines = client_data.splitlines()

    lock.acquire()

    redirection_flag.update(flag=True)
    server_url = redirected_url
    first_line_tokens = lines[0].split()
    first_line_tokens[1] = server_url.encode()
    lines[0] = b' '.join(first_line_tokens)

    second_line_tokens = lines[1].split()
    second_line_tokens[1] = redirected_host.encode()
    lines[1] = b' '.join(second_line_tokens)

    change = client_data.split(b'\r\n')
    change[0] = lines[0]
    change[1] = lines[1]
    client_data = b'\r\n'.join(change)
    lines = client_data.splitlines()
    request = request_parser(lines)["request"]
    user_agent = request_parser(lines)["user_agent"]

    lock.release()

    data = {
        "client_data": client_data,
        "request": request,
        "user_agent": user_agent,
    }

    return data
```

4) URL Redirection을 위한 redirection()함수입니다. 이 함수는 각 Thread에서 filtered_url이 full_url에 존재할 때 호출되는 함수입니다. 이 때, 이 함수는 client_data를 매개변수로 입력받습니다. 그리고 최초에 정의한 filtered_url, redirected_url, redirected_host를 global변수로 설정합니다. Client_data를 line by line으로 분할하고, redirection을 진행하기 때문에 이에 대한 flag변수를 True로 변경합니다. 그리고 클라이언트가 보낸 요청에 대한 정보를 Redirection을 위한 서버에 보내기 위해 header정보를 수정하도록 했습니다. 원래 header정보의 Request부분과 Host을 redirection을 위한 정보로 수정하고, 이를 다시 합치는 방식으로 진행했습니다. 이 과정 또한 lock을 이용하였습니다. Header정보에 대한 수정이 끝나면 수정된 client_data, request, user_agent정보를 dictionary에 담아 return하도록 했습니다. Request정보와 User_agent정보는 Log를 남길 때 필요하기 때문에 이 함수에서 미리 처리해두기로 했습니다.

```python
# http header에서 status, content_type, content_length 정보 parsing하고 return해주는 함수
def http_response_parser(replies):
    status = ""
    content_type = "Not Specified"
    content_length = "0"
    for i in range(len(replies)):
        if b'HTTP' in replies[i]:
            status = replies[i]
        if b'Content-Type' in replies[i]:
            content_type = replies[i].decode(
                encoding="utf-8", errors="ignore").split()[1]
        if b'Content-Length' in replies[i]:
            content_length = replies[i].decode(
                encoding="utf-8", errors="ignore").split()[1]

    data = {
        "status": status.decode(encoding="utf-8", errors="ignore"),
        "content_type": content_type,
        "content_length": content_length,
    }

    return data
```

5) Proxy Socket이 Server로부터 정보를 받아왔을 때 Response_Header를 Parsing하기 위한 함수입니다. Status, Content_type, Content_length는 각각 “”, “Not Specipied”, “0”으로 초기화했는데, 이는 Content_type, Content_Length가 존재하지 않는 응답헤더가 존재하기 때문입니다. 이 후에 line by line으로 전달받은 응답 헤더에서 HTTP, Content-Type, Content-Length의 위치를 탐색하고, 이 부분을 decode하여 dictionary에 담아 return하도록 했습니다. decode방식은 ‘utf-8’로 통일했습니다.

```python
# thread function : client_socket, addr, data를 parameter로 함
# client_data에서 server에 대한 정보 등을 parsing하고, proxy socket에게 넘겨주는 역할을 함
def p_thread(client_socket, client_addr, client_data):
    global image_filtering, redirected_url, redirected_server, redirection_flag
    storage = []

    storage.append("[CLI connected to {}:{}]".format(
        client_addr[0], client_addr[1]))
    # Client Browser Requests appears here
    try:
        # lines: client_data를 line 단위로 split 후 parsing하기 위함
        lines = client_data.splitlines()
        request = request_parser(lines)["request"]
        user_agent = request_parser(lines)["user_agent"]

        storage.append("[CLI ==> PRX --- SRV]")
        storage.append(f"> {request}")
        storage.append(f"> {user_agent}")

        ################################################################
        server_url = domain_parser(lines)["server_url"]
        server_port = 80
        ################## Redirection ##############################
        # 만약 filtered_url이 현재 server_url에 포함되어 있다면 Redirection 진행
        if server_url.find(filtered_url) != -1:
            client_data = redirection(client_data)["client_data"]
            request = redirection(client_data)["request"]
            user_agent = redirection(client_data)["user_agent"]

            proxy_server(redirected_server, server_port, client_socket,
                         client_addr, client_data, request, user_agent, storage)
        else:
            lock.acquire()
            redirection_flag.update(flag=False)
            lock.release()

            proxy_server(server_url, server_port, client_socket,
                         client_addr, client_data, request, user_agent, storage)
    except Exception as e:
        pass
```

6) Client Request가 들어오면 Threading을 시작합니다. p_thread함수는 threading을 위한 함수입니다. Image_filtering, redirected_url, redirected_server, redirection_flag를 global변수로 선언합니다.

각 Thread에 대하여 storage 리스트를 선언하는데, 이는 각 쓰레드가 시작부터 종료때까지 남길 log를 저장할 공간입니다. 이후에 위에서 정의한 함수들을 바탕으로 Request, User_agent정보를 입력받습니다. Server_url역시 위에서 정의한 함수를 바탕으로 입력받으며, server_port는 http요청에 대한 '80'으로 고정했습니다. 그리고 URL Redirection의 여부를 확인하기 위한 if condition을 생성했고, Redirection이 진행될 때와 그렇지 않을 때 각각 다른 server로 Threading을 진행합니다. 한편, Redirection이 진행되지 않으면 flag는 False입니다. 최종적으로 client_data에서 Parsing이 끝난 요청 정보를 서버와 직접 통신하게 될 proxy_socket에게 넘겨주도록 했습니다. 이를 통해 Client는 서버 사이에 Proxy_server를 두고 통신할 수 있게 됩니다.

```python
# 메인 proxy_server에 대한 함수
# server와 client의 정보를 가지고 socket을 생성한다.
# client의 요청을 server에게 send하고, server의 응답을 recv하여 client에게 send해준다.
def proxy_server(server_url, server_port, client_socket, client_addr, client_data, reques
    global print_number, image_filtering, redirection_flag
    try:
        proxy_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # print("연결?")
        proxy_socket.connect((server_url, server_port))
        storage.append(f"[SRV connected to {server_url}:{server_port}]")

        proxy_socket.send(client_data)

        storage.append("[CLI --- PRX ==> SRV]")
        storage.append(f"> {request}")
        storage.append(f"> {user_agent}")

        proxy_socket.setblocking(0)
        total_data = []
        data = ""
        timeout = 2
        begin = time.time()
```

7) proxy_server()함수는 각 Thread에서 데이터를 전달받습니다. 이후 서버와 클라이언트에게 각각 요청, 응답을 받을 수 있는 소켓을 생성합니다. 이 소켓은 클라이언트에게 받은 요청을 바탕으로 해당 Server와 Connect를 진행합니다. 이후, 클라이언트의 요청을 서버에게 대신 전달하도록 합니다.

```python
        ########################################################################################
        # http 요청은 TCP를 이용하기 때문에 timeout을 이용하여 손실되는 데이터가 없도록 한다.
        while True:
            # print("while...")
            # if you got some data, then break after timeout
            if total_data and time.time()-begin > timeout:
                # print("??????????/")
                break
            # if you got no data at all, wait a little longer, twice the timeout
            elif time.time()-begin > timeout * 2:
                # print("fffffffffffffff")
                break

            # Read reply or data to from end web server
            try:
                reply = proxy_socket.recv(buffer_size)
                # print(reply)
                if reply:
                    total_data.append(reply)
                    begin = time.time()
                else:
                    time.sleep(0.1)
            except Exception as e:
                pass

        # join all parts to make final string
        data = b''.join(total_data)
```

8) 이 후 서버에서 데이터를 수신합니다. 수신하는 동안에는 TCP 연결을 기반으로 하기 때문에 timeout에 의한 패킷 손실을 고려하여 모든 데이터를 온전히 수신할 수 있도록 timeout에 대한 조건을 넣었습니다. 또한 버퍼의 사이즈 역시 고려하여 손실되는 데이터가 없도록 수신할 때마다 total_data라는 리스트에 이를 담아두고, 수신이 끝나면 이를 최종적으로 결합하도록 했습니다.

```
229 ################################################################################
230         datas = data.splitlines()
231         status = http_response_parser(datas)["status"]
232         ctype = http_response_parser(datas)["content_type"]
233         clength = http_response_parser(datas)["content_length"]
234         storage.append("[CLI --- PRX <== SRV]")
235         storage.append(f'> {status}')
236         storage.append(f'> {ctype} {clength}bytes')
237 
238         # image filtering 기능이 작동되면
239         # Content-type = image인 response의 body를 drop하여 client에게 전송한다.
240         if image_filtering.get('flag') == True and 'image' in ctype:
241             clength = "0"
242             data = data.split(b'\r\n\r\n')[0]
243             # storage.append(data)
244 
245         client_socket.send(data)  # send reply back  to client
246 
247         # send notification to proxy server
248         storage.append("[CLI <== PRX --- SRV]")
249         storage.append(f'> {status}')
250         storage.append(f'> {ctype} {clength}bytes')
251 
252         proxy_socket.close()
253         client_socket.close()
254         storage.append("[CLI disconnected]")
255         storage.append("[SRV disconnected]")
```

9) 8)과정이 끝나면 프록시는 서버에서 전달받은 데이터를 가지고 있습니다. 이 정보를 다시 Parsing 하여 응답 Status, 응답 Content-Type, 응답 Content-Length를 추출합니다. 이는 현재 Image Filtering의 작동 상태에 따라서 클라이언트에게 데이터를 보내줄 때 필요합니다. 만약 Image Filtering 기능이 작동하고 있다면, 현재 수신한 데이터 중에서 Content-Type이 'Image'인 데이터의 로그들을 제거하도록 합니다. 그리고 그 외에 정보에 한해서만 클라이언트에게 전송하도록 했습니다.

이 과정이 끝나면 각 Thread는 CLI – PRX – SRV의 전체적인 과정을 마치게 됩니다. 따라서 최종적으로 해당 Thread를 종료하기 위해 각 소켓을 종료하도록 합니다.

```
256 ############################ 생성된 로그 출력 ###################################
257             lock.acquire()
258             thread_num_pos = str(threading.currentThread()).find('-')
259             thread_num = str(threading.currentThread())[thread_num_pos+1]
260             print_number += 1
261             storage.insert(
262                 0, f"{print_number} [Conn:    {thread_num}/    {threading.activeCount()}]")
263 
264             # image filtering, URL Redirection에 대한 flag를 확인하여 Indicator를 설정함
265             if image_filtering.get('flag') == True:
266                 image_indicator = "O"
267             elif image_filtering.get('flag') == False:
268                 image_indicator = "X"
269             if redirection_flag.get('flag') == True:
270                 url_indicator = "O"
271             elif redirection_flag.get('flag') == False:
272                 url_indicator = "X"
273             storage.insert(
274                 1, f"[ {url_indicator} ] URL filter  |  [ {image_indicator} ] Image filter")
275             storage.append("----------------------------------------")
276             storage.insert(2, " ")
277             # 로그 출력
278             for log in storage:
279                 print(log)
280             lock.release()
281 ###########################################################################
282         except Exception as e:
283             proxy_socket.close()
284             client_socket.close()
285             sys.exit()
286 
```

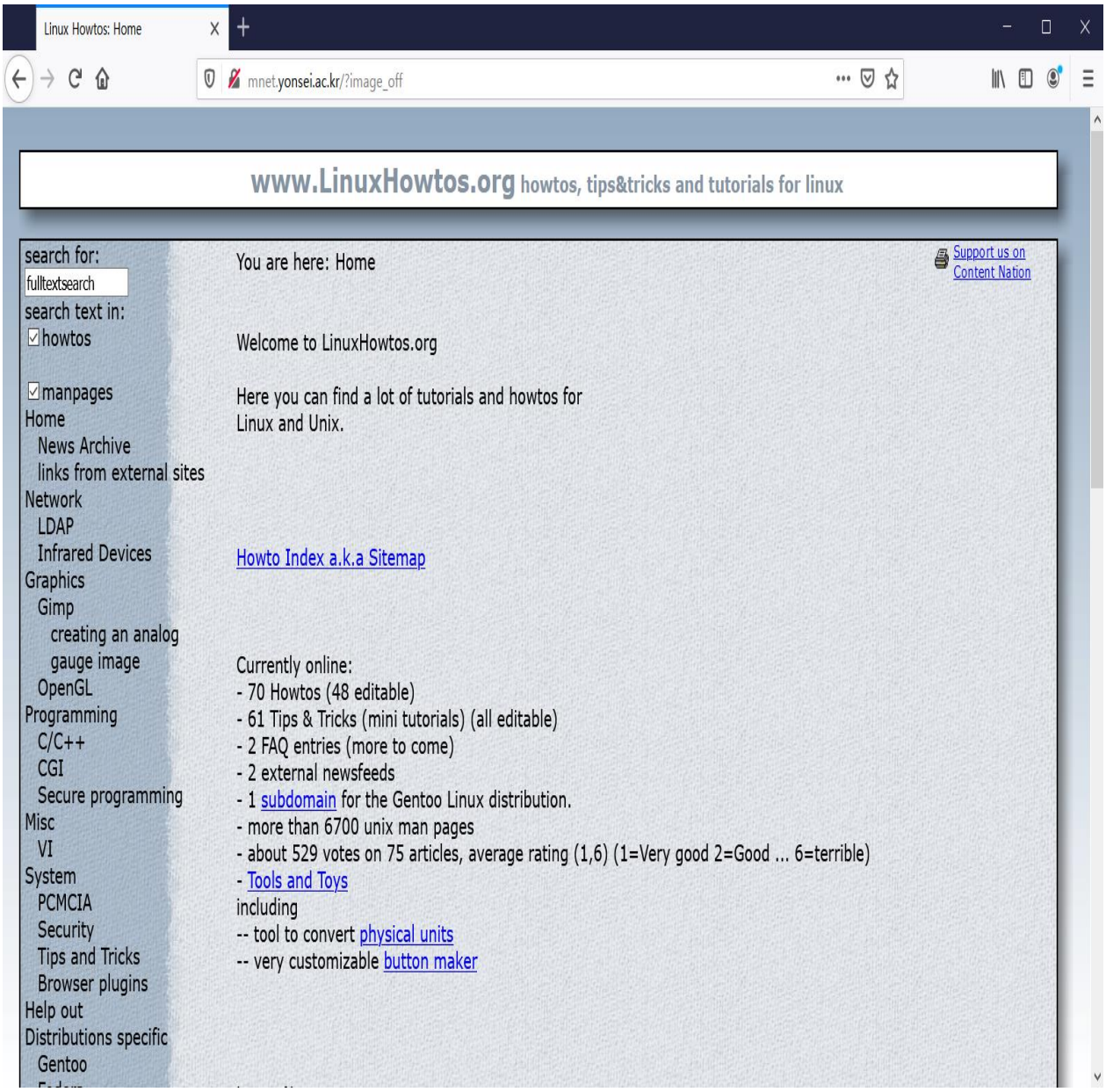
10) 마지막으로 지금까지 얻은 정보를 출력하는 부분입니다. Multi-Threading이 진행되고 있기 때문에 출력 로그가 혼재될 가능성이 있습니다. 이를 해결하기 위하여 lock을 이용하였고, 현재 작동되고 있는 Thread의 수는 threading.currentThread()함수를 통해 얻어냈고, 현재 Thread의 Number 역시 threading.currentThread()의 정보에서 번호 정보만을 Parsing하였습니다. 그리고 각 Thread 로그를 출력할 때 순서번호는 print_number 변수를 통해 매 출력마다 1씩 증가하도록 했습니다.

마지막으로, Image Filtering과 URL Redirection의 flag를 확인하여 출력 시 적절한 “O”,”X” indicator를 사용하도록 했으며, 최종적으로 Storage에 담긴 log를 출력하였습니다.

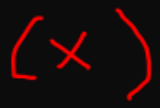
4. Snapshots of Each Function

① http://mnet.yonsei.ac.kr?image_off에 요청을 보냈습니다.

이 때, 'yonsei'가 URL에 포함되어 있고 '?imgae_off'라는 query가 있기 때문에 Redirection과 Image Filtering이 작동합니다. 따라서 해당 요청은 'linuxhowtos.org'로 보내지게 되고, URL Redirection, Image Filtering에 대한 정보는 모두 "O"로 표시됩니다. 첫번째 로그는 ahnlab에 컴퓨터 내에서 자동으로 보낸 요청인데, 이 부분은 무시해주시면 감사하겠습니다. 또한, Multi-threading에 대한 log가 많아서 일부를 캡쳐하였습니다.

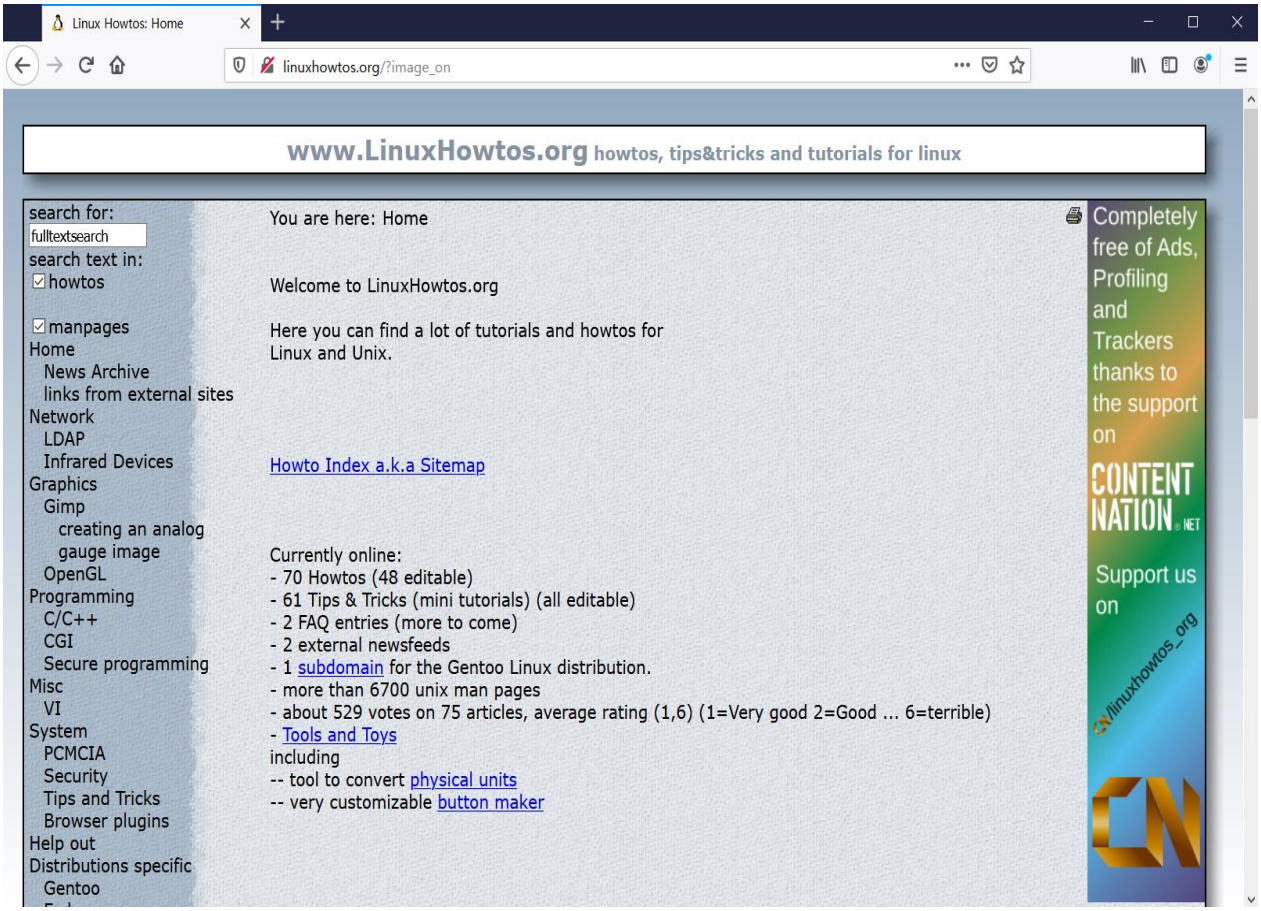


```
josunghyeon@DESKTOP-8TJH6BQ:/mnt/c/Users/josunghyeon/Desktop/computerNetwork$ python3 project.py 9001
Starting proxy server on port 9001
----------------------------------------
1 [Conn:    1/    2]
[ X ] URL filter  |  [ X ] Image filter

[CLI connected to 127.0.0.1:44714]
[CLI ==> PRX --- SRV]
> GET http://gms.ahnlab.com/live.html HTTP/1.0
> MeDCore
[SRV connected to gms.ahnlab.com:80]
[CLI --- PRX ==> SRV]
> GET http://gms.ahnlab.com/live.html HTTP/1.0
> MeDCore
[CLI --- PRX <== SRV]
> HTTP/1.1 200 OK
> -Options: nosniff 3bytes
[CLI <== PRX --- SRV]
> HTTP/1.1 200 OK
> -Options: nosniff 3bytes
[CLI disconnected]
[SRV disconnected]
----------------------------------------
2 [Conn:    2/    2]
[ O ] URL filter  |  [ O ] Image filter

[CLI connected to 127.0.0.1:44724]
[CLI ==> PRX --- SRV]
> GET http://mnet.yonsei.ac.kr/?image_off HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
[SRV connected to linuxhowtos.org:80]
[CLI --- PRX ==> SRV]
> GET http://linuxhowtos.org/ HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
[CLI --- PRX <== SRV]
> HTTP/1.1 200 OK
> text/html; charset=utf-8 3912bytes
[CLI <== PRX --- SRV]
> HTTP/1.1 200 OK
> text/html; charset=utf-8 3912bytes
[CLI disconnected]
[SRV disconnected]
----------------------------------------
3 [Conn:    5/    6]
[ O ] URL filter  |  [ O ] Image filter

[CLI connected to 127.0.0.1:44742]
[CLI ==> PRX --- SRV]
> GET http://static.linuxhowtos.org/data/bluefade.jpg HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
[SRV connected to static.linuxhowtos.org:80]
[CLI --- PRX ==> SRV]
> GET http://static.linuxhowtos.org/data/bluefade.jpg HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
[CLI --- PRX <== SRV]
> HTTP/1.1 200 OK
> image/jpeg 310bytes
[CLI <== PRX --- SRV]
> HTTP/1.1 200 OK
> image/jpeg 0bytes
[CLI disconnected]
[SRV disconnected]
----------------------------------------
```



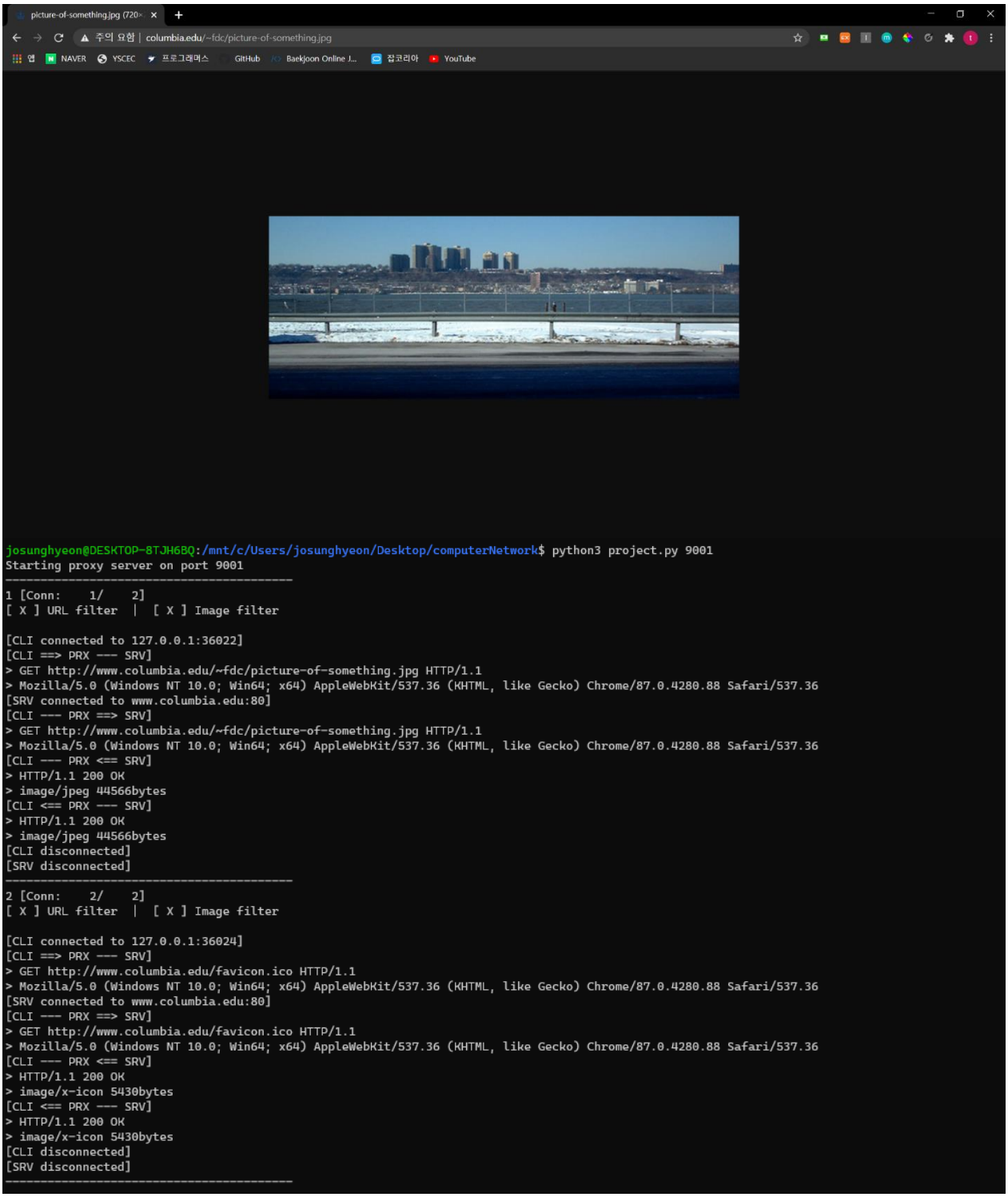
② http://linuxhowtos.org?image_on에 요청을 보냈습니다.

Image_on이기 때문에 이전에 없었던 오른쪽의 배너 이미지가 삽입된 것을 확인할 수 있습니다. 전송받은 브라우저와 로그를 결과로 제시하였습니다. URL, Image Filter는 모두 "X"로 표시되었음을 확인할 수 있으며, 전체 로그 중 일부를 캡쳐했습니다.

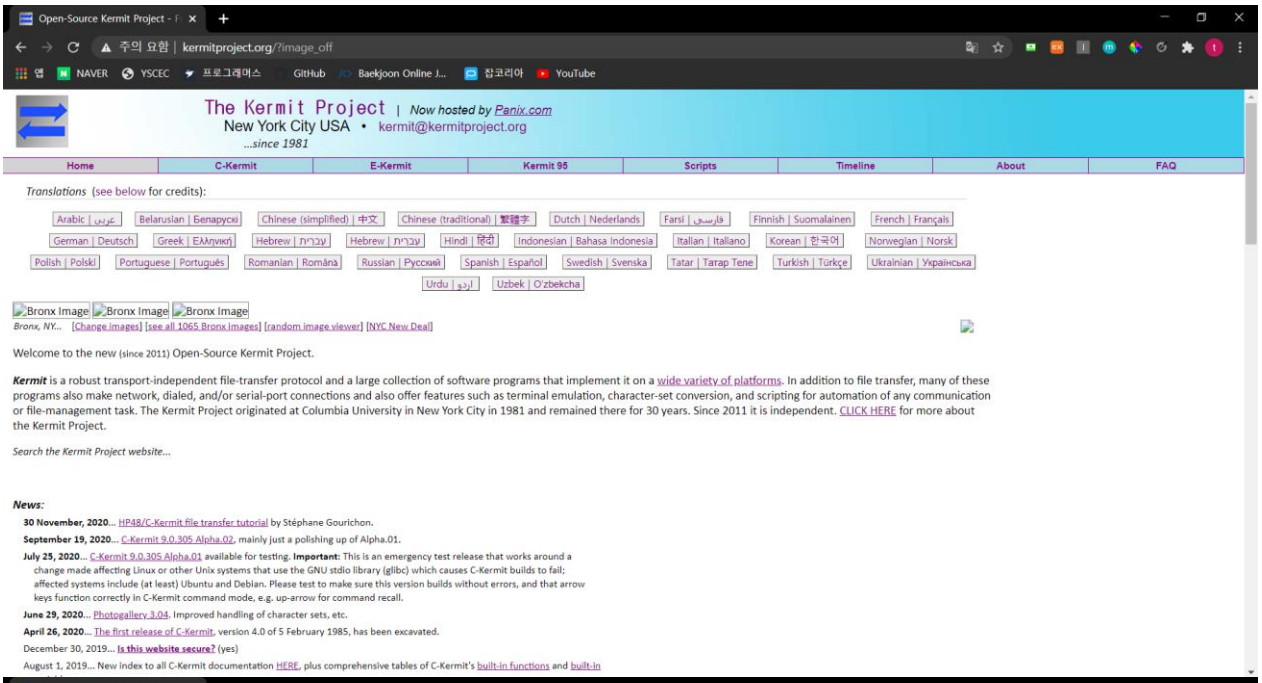


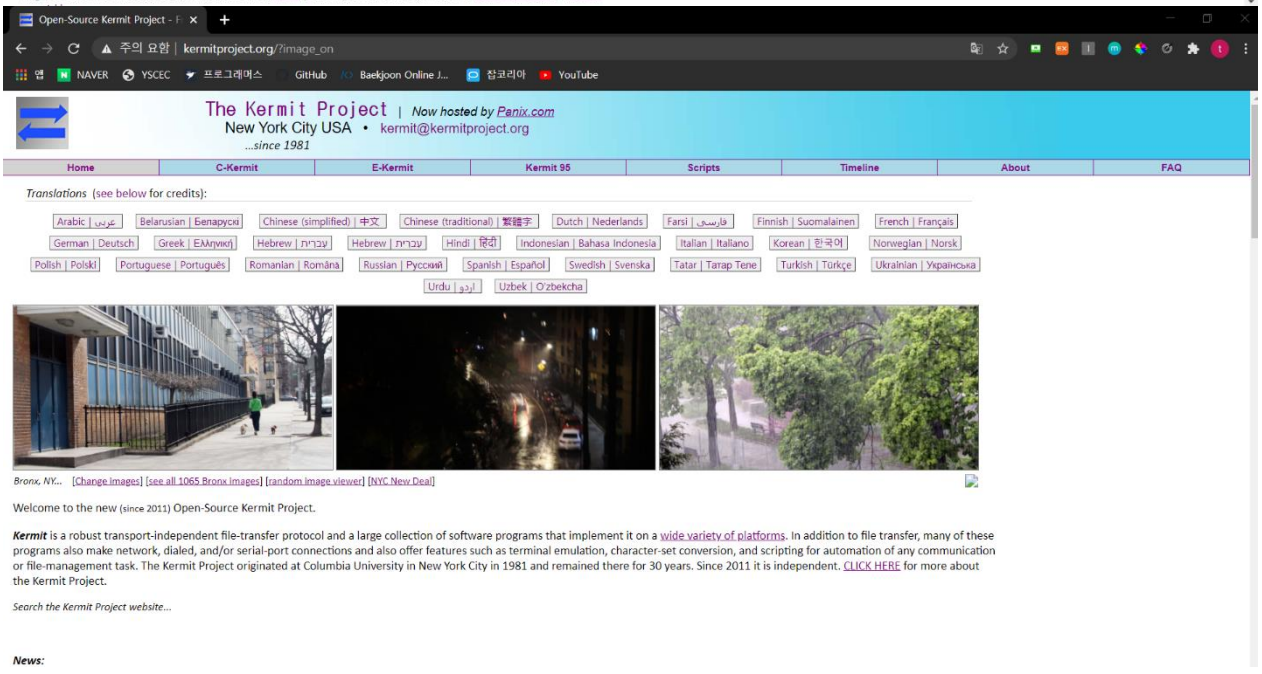
```
josunghyeon@DESKTOP-8TJH6BQ:/mnt/c/Users/josunghyeon/Desktop/computerNetwork$ python3 project.py 9001
Starting proxy server on port 9001
--------------------------------------------
1 [Conn:    1/    2]
[ X ] URL filter  |  [ X ] Image filter

[CLI connected to 127.0.0.1:45122]
[CLI ==> PRX --- SRV]
> GET http://linuxhowtos.org/?image_on HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
[SRV connected to linuxhowtos.org:80]
[CLI --- PRX ==> SRV]
> GET http://linuxhowtos.org/?image_on HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
[CLI --- PRX <== SRV]
> HTTP/1.1 200 OK
> text/html; charset=utf-8 3846bytes
[CLI <== PRX --- SRV]
> HTTP/1.1 200 OK
> text/html; charset=utf-8 3846bytes
[CLI disconnected]
[SRV disconnected]
--------------------------------------------
2 [Conn:    2/    2]
[ X ] URL filter  |  [ X ] Image filter

[CLI connected to 127.0.0.1:45128]
[CLI ==> PRX --- SRV]
> GET http://linuxhowtos.org/data/blank.gif?w=1285&h=702 HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
[SRV connected to linuxhowtos.org:80]
[CLI --- PRX ==> SRV]
> GET http://linuxhowtos.org/data/blank.gif?w=1285&h=702 HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
[CLI --- PRX <== SRV]
> HTTP/1.1 200 OK
> image/gif 55bytes
[CLI <== PRX --- SRV]
> HTTP/1.1 200 OK
> image/gif 55bytes
[CLI disconnected]
[SRV disconnected]
--------------------------------------------
3 [Conn:    3/    2]
[ X ] URL filter  |  [ X ] Image filter

[CLI connected to 127.0.0.1:45164]
[CLI ==> PRX --- SRV]
> GET http://linuxhowtos.org/?image_on HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
[SRV connected to linuxhowtos.org:80]
[CLI --- PRX ==> SRV]
> GET http://linuxhowtos.org/?image_on HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
[CLI --- PRX <== SRV]
> HTTP/1.1 200 OK
> text/html; charset=utf-8 3846bytes
[CLI <== PRX --- SRV]
> HTTP/1.1 200 OK
> text/html; charset=utf-8 3846bytes
[CLI disconnected]
[SRV disconnected]
--------------------------------------------
4 [Conn:    4/    4]
[ X ] URL filter  |  [ X ] Image filter

[CLI connected to 127.0.0.1:45170]
[CLI ==> PRX --- SRV]
> GET http://static.linuxhowtos.org/css/blue.css HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
[SRV connected to static.linuxhowtos.org:80]
[CLI --- PRX ==> SRV]
> GET http://static.linuxhowtos.org/css/blue.css HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
[CLI --- PRX <== SRV]
> HTTP/1.1 200 OK
> text/css 876bytes
[CLI <== PRX --- SRV]
> HTTP/1.1 200 OK
> text/css 876bytes
[CLI disconnected]
[SRV disconnected]
--------------------------------------------
5 [Conn:    5/    3]
[ X ] URL filter  |  [ X ] Image filter

[CLI connected to 127.0.0.1:45176]
[CLI ==> PRX --- SRV]
> GET http://static.linuxhowtos.org/css/nonie.css HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
[SRV connected to static.linuxhowtos.org:80]
[CLI --- PRX ==> SRV]
> GET http://static.linuxhowtos.org/css/nonie.css HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
[CLI --- PRX <== SRV]
> HTTP/1.1 200 OK
> text/css 419bytes
[CLI <== PRX --- SRV]
> HTTP/1.1 200 OK
> text/css 419bytes
[CLI disconnected]
[SRV disconnected]
--------------------------------------------
```

③ 과제 스펙에 제시된 Colombia.edu~ 에 요청을 보냈습니다. URL Redirection, Image filtering은 없으며 정상적으로 출력되는 것을 확인할 수 있습니다.



③ 다른 http website에 Image_on을 한 상태로 보낸 결과와 Image_off를 한 상태로 보낸 결과를 보면 브라우저의 이미지의 존재 여부가 달라집니다. 위의 사진이 이미지 필터링을 요청한 것이며, 아래는 이미지 필터링을 해제한 화면입니다. 또한 Interface Log의 위 사진이 Image_Filter = "O"인 사진이며 아래 사진이 Image_Filter = "X"인 사진입니다.

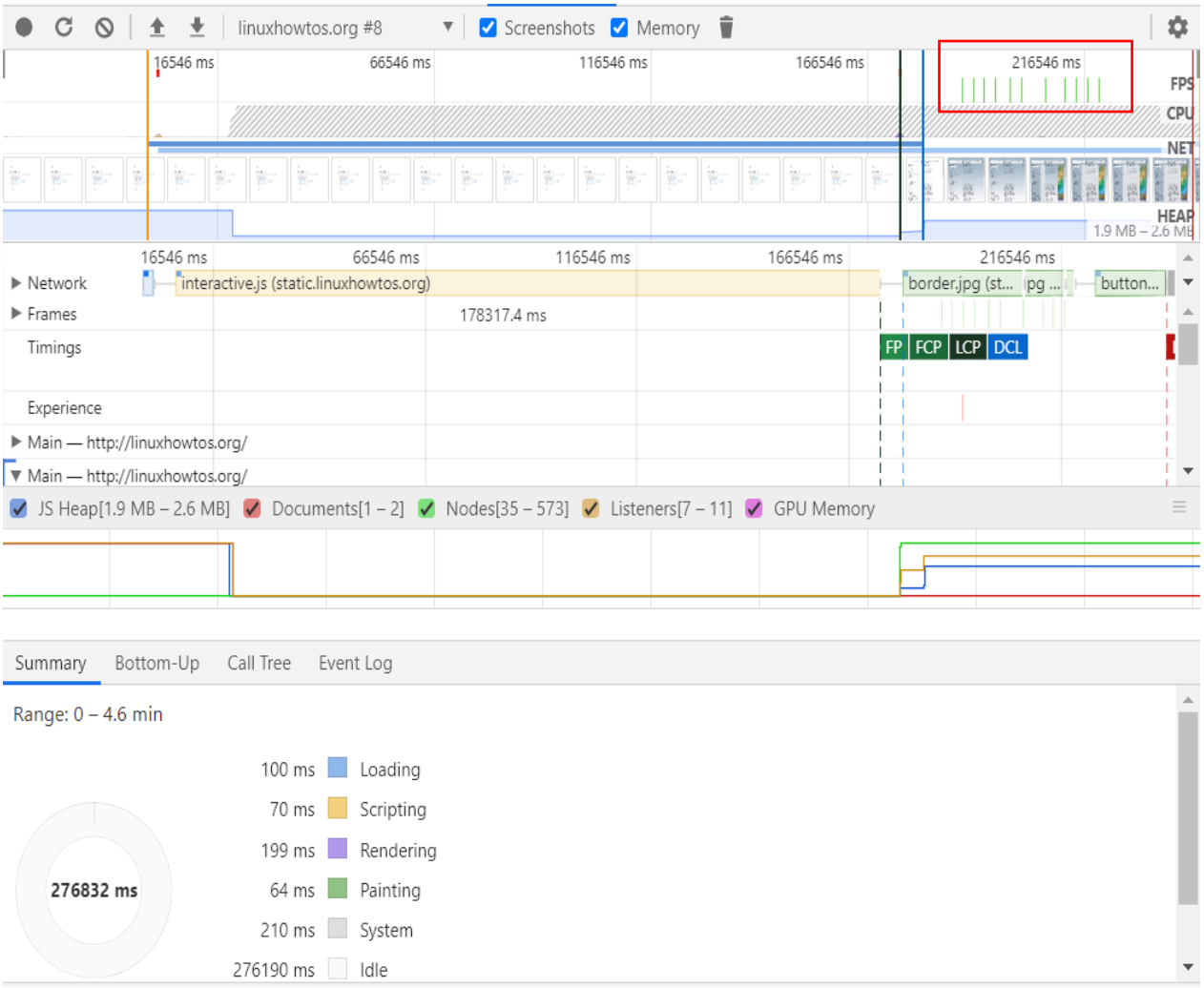




```
josunghyeon@DESKTOP-8TJH6BQ:/mnt/c/Users/josunghyeon/Desktop/computerNetwork$ python3 project.py 9001
Starting proxy server on port 9001
----------------------------------------
1 [Conn:    1/    2]
[ X ] URL filter  |  [ O ] Image filter

[CLI connected to 127.0.0.1:36068]
[CLI ==> PRX --- SRV]
> GET http://www.kermitproject.org/?image_off HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
[SRV connected to www.kermitproject.org:80]
[CLI --- PRX ==> SRV]
> GET http://www.kermitproject.org/?image_off HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
[CLI --- PRX <== SRV]
> HTTP/1.1 304 Not Modified
> Not Specified 0bytes
[CLI <== PRX --- SRV]
> HTTP/1.1 304 Not Modified
> Not Specified 0bytes
[CLI disconnected]
[SRV disconnected]
----------------------------------------
2 [Conn:    3/    4]
[ X ] URL filter  |  [ O ] Image filter

[CLI connected to 127.0.0.1:36080]
[CLI ==> PRX --- SRV]
> GET http://www.kermitproject.org/bronx564.jpg HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
[SRV connected to www.kermitproject.org:80]
[CLI --- PRX ==> SRV]
> GET http://www.kermitproject.org/bronx564.jpg HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
[CLI --- PRX <== SRV]
> HTTP/1.1 200 OK
> image/jpeg 88961bytes
[CLI <== PRX --- SRV]
> HTTP/1.1 200 OK
> image/jpeg 0bytes
[CLI disconnected]
[SRV disconnected]
----------------------------------------
3 [Conn:    2/    3]
[ X ] URL filter  |  [ O ] Image filter

[CLI connected to 127.0.0.1:36070]
[CLI ==> PRX --- SRV]
> GET http://www.kermitproject.org/bronx358.jpg HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
[SRV connected to www.kermitproject.org:80]
[CLI --- PRX ==> SRV]
> GET http://www.kermitproject.org/bronx358.jpg HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
[CLI --- PRX <== SRV]
> HTTP/1.1 200 OK
> image/jpeg 238773bytes
[CLI <== PRX --- SRV]
> HTTP/1.1 200 OK
> image/jpeg 0bytes
[CLI disconnected]
[SRV disconnected]
----------------------------------------
4 [Conn:    4/    2]
[ X ] URL filter  |  [ O ] Image filter
```

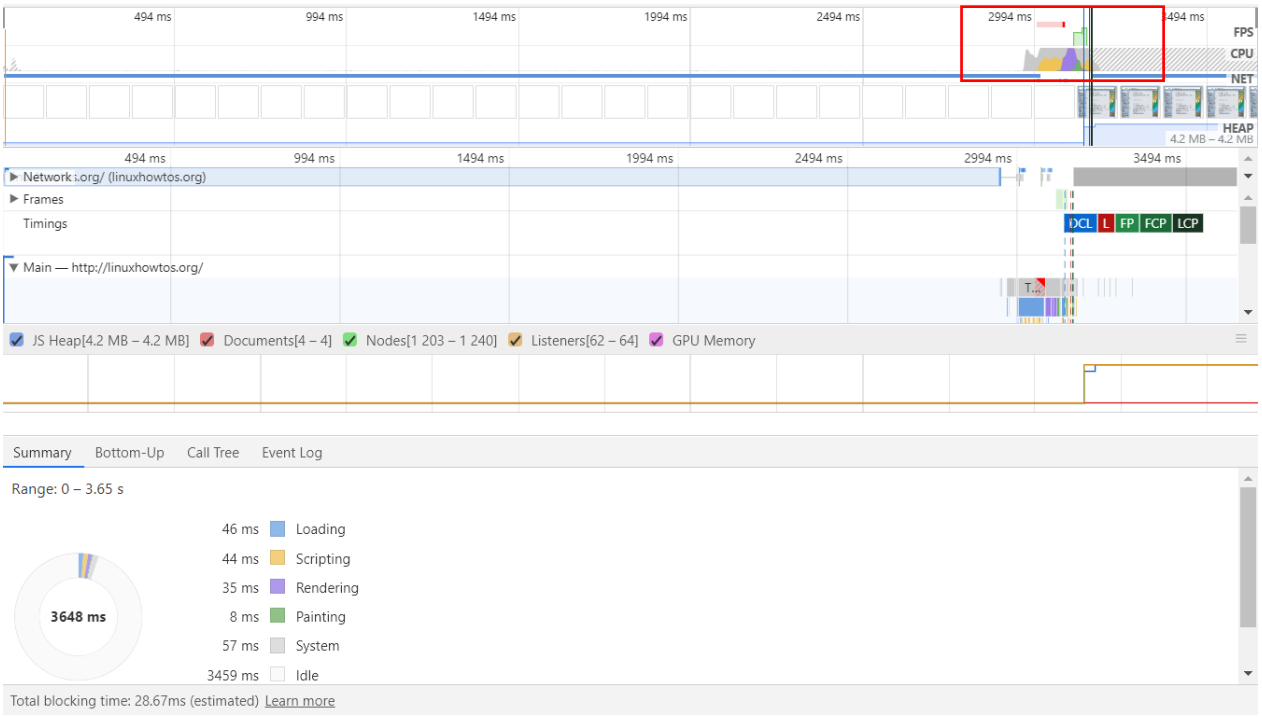
```
----------------------------------------
19 [Conn:    1/    2]
[ X ] URL filter  |  [ X ] Image filter

[CLI connected to 127.0.0.1:36158]
[CLI ==> PRX --- SRV]
> GET http://www.kermitproject.org/?image_on HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
[SRV connected to www.kermitproject.org:80]
[CLI --- PRX ==> SRV]
> GET http://www.kermitproject.org/?image_on HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
[CLI --- PRX <== SRV]
> HTTP/1.1 304 Not Modified
> Not Specified 0bytes
[CLI <== PRX --- SRV]
> HTTP/1.1 304 Not Modified
> Not Specified 0bytes
[CLI disconnected]
[SRV disconnected]
----------------------------------------
20 [Conn:    2/    4]
[ X ] URL filter  |  [ X ] Image filter

[CLI connected to 127.0.0.1:36160]
[CLI ==> PRX --- SRV]
> GET http://www.kermitproject.org/bronx307.jpg HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
[SRV connected to www.kermitproject.org:80]
[CLI --- PRX ==> SRV]
> GET http://www.kermitproject.org/bronx307.jpg HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
[CLI --- PRX <== SRV]
> HTTP/1.1 200 OK
> image/jpeg 183966bytes
[CLI <== PRX --- SRV]
> HTTP/1.1 200 OK
> image/jpeg 183966bytes
[CLI disconnected]
[SRV disconnected]
----------------------------------------
21 [Conn:    2/    3]
[ X ] URL filter  |  [ X ] Image filter

[CLI connected to 127.0.0.1:36170]
[CLI ==> PRX --- SRV]
> GET http://www.kermitproject.org/bronx926.jpg HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
[SRV connected to www.kermitproject.org:80]
[CLI --- PRX ==> SRV]
> GET http://www.kermitproject.org/bronx926.jpg HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
[CLI --- PRX <== SRV]
> HTTP/1.1 200 OK
> image/jpeg 188530bytes
[CLI <== PRX --- SRV]
> HTTP/1.1 200 OK
> image/jpeg 188530bytes
[CLI disconnected]
[SRV disconnected]
----------------------------------------
22 [Conn:    2/    2]
[ X ] URL filter  |  [ X ] Image filter

[CLI connected to 127.0.0.1:36172]
[CLI ==> PRX --- SRV]
> GET http://www.kermitproject.org/bronx79.jpg HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
[SRV connected to www.kermitproject.org:80]
[CLI --- PRX ==> SRV]
> GET http://www.kermitproject.org/bronx79.jpg HTTP/1.1
> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
[CLI --- PRX <== SRV]
> HTTP/1.1 200 OK
> image/jpeg 244353bytes
[CLI <== PRX --- SRV]
> HTTP/1.1 200 OK
> image/jpeg 244353bytes
[CLI disconnected]
[SRV disconnected]
----------------------------------------
```

5. Comparison the performance with applying Multi thread and before in Chart



Chrome의 성능 분석 차트를 이용하여 Multi-Thread의 사용 여부에 따른 성능을 비교해보았다. 정확한 성능을 비교하기 위하여 캐시를 비운 상태에서 실행하도록 하였다.

위 그림은 Multi-Thread를 사용하지 않고 linuxhowtos.org에 요청을 보낸 것이다. 차트에서 살펴볼 수 있듯이 약 20000ms 이후에 화면이 로딩되고, Elapsed time이 상당히 긴 것을 확인할 수 있다. 화면이 로딩된 순간으로부터 하나의 프로세스 안에서 로딩이 이루어지고 있다. 또한, CPU의 사용을 보았을 때에도 전체를 하나의 프로세스가 점유하고 있음을 알 수 있다.



위 그림은 Multi-Thread를 사용하여 linuxhowtos.org에 요청을 보낸 것이다. 차트에서 살펴볼 수 있듯이 약 2~3000ms 이후에 화면이 로딩되고, Elapsed time이 쓰레드를 사용하기 전보다 상당히 짧아진 것을 확인할 수 있다. 또한, CPU의 사용을 보았을 때에도 여러 쓰레드가 CPU를 나누어서 할당받고 있음을 알 수 있다.

6. Reference

James F.Kurose, Keith W. Ross, “COMPUTER NETWORKING A TOP-DOWN-APPROACH”