

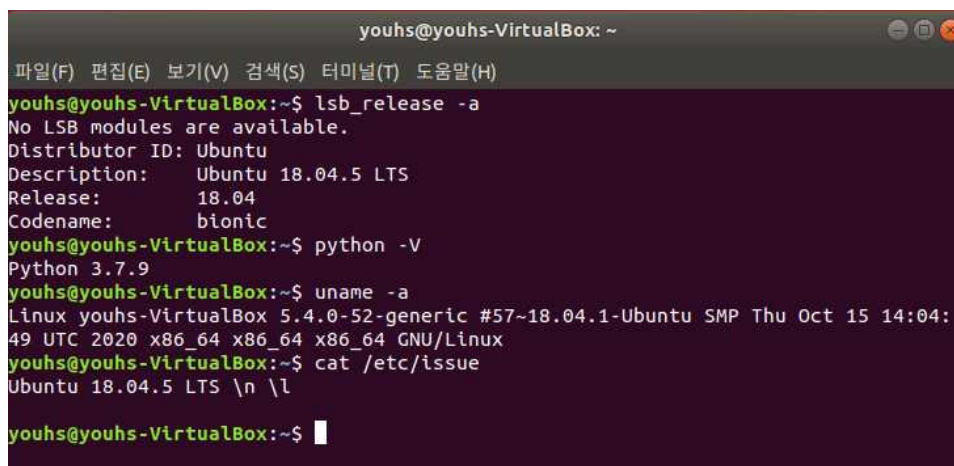
Computer Network Project 3

2015147533 유현석

1. Introduction

Software environment : Ubuntu (Linux)

Programming language(version) : Python 3.7.9



```
youhs@youhs-VirtualBox: ~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
youhs@youhs-VirtualBox:~$ lsb_release -a  
No LSB modules are available.  
Distributor ID: Ubuntu  
Description:   Ubuntu 18.04.5 LTS  
Release:       18.04  
Codename:      bionic  
youhs@youhs-VirtualBox:~$ python -V  
Python 3.7.9  
youhs@youhs-VirtualBox:~$ uname -a  
Linux youhs-VirtualBox 5.4.0-52-generic #57~18.04.1-Ubuntu SMP Thu Oct 15 14:04:  
49 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux  
youhs@youhs-VirtualBox:~$ cat /etc/issue  
Ubuntu 18.04.5 LTS \n \l  
youhs@youhs-VirtualBox:~$
```

2. Reference

1. <https://12bme.tistory.com/325>

-> HTTP 헤더 구조

2. <https://docs.python.org/ko/3/howto/sockets.html>

-> Python socket 함수

3. https://ko.wikipedia.org/wiki/%ED%94%84%EB%A1%9D%EC%8B%9C_%EC%84%9C%EB%B2%84

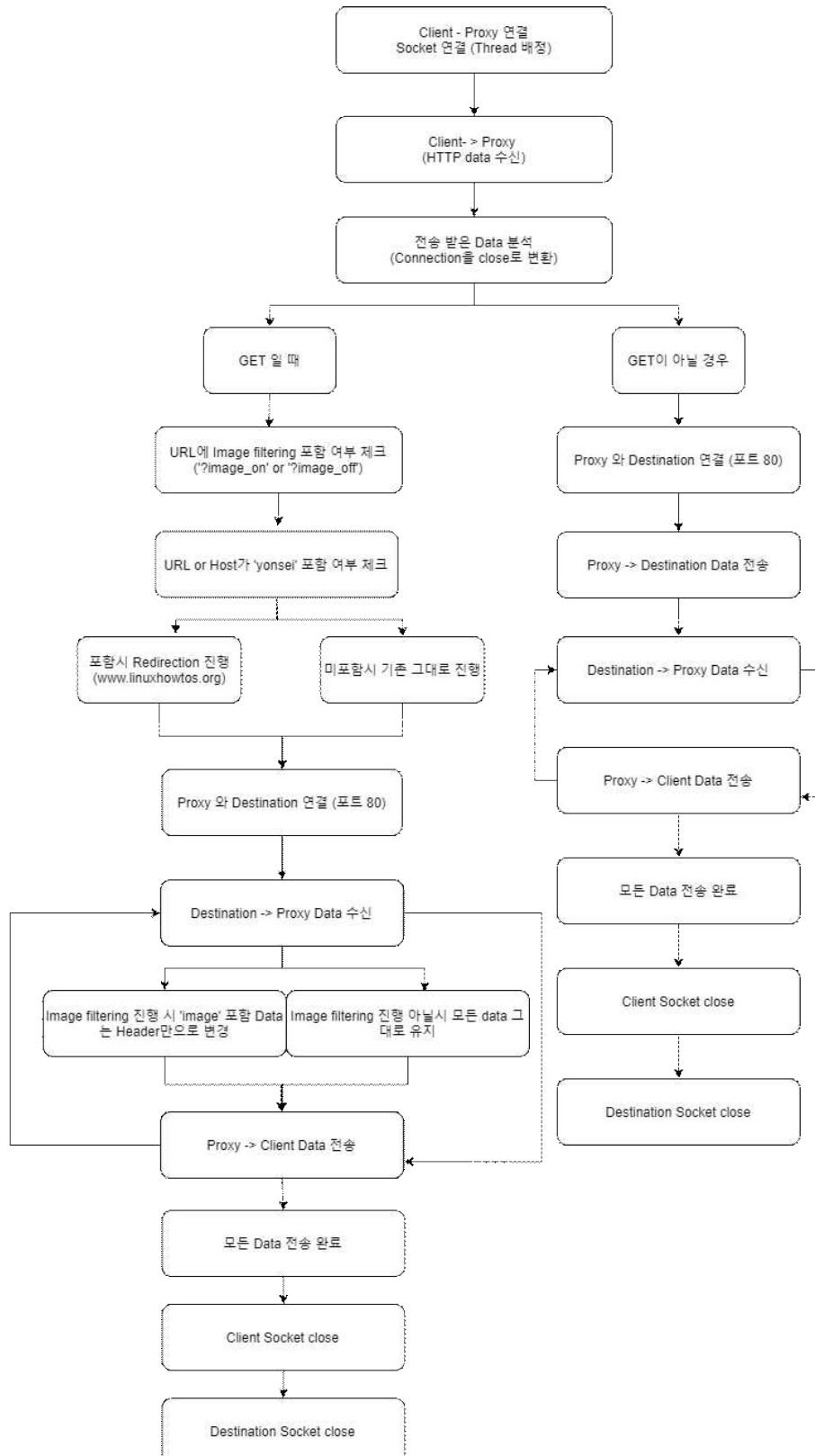
-> Proxy server

4. <https://m.blog.naver.com/PostView.nhn?blogId=sim940228&logNo=220712268894&proxyReferer=https:%2F%2Fwww.google.com%2F>

-> Python으로 구축한 Proxy server 예시

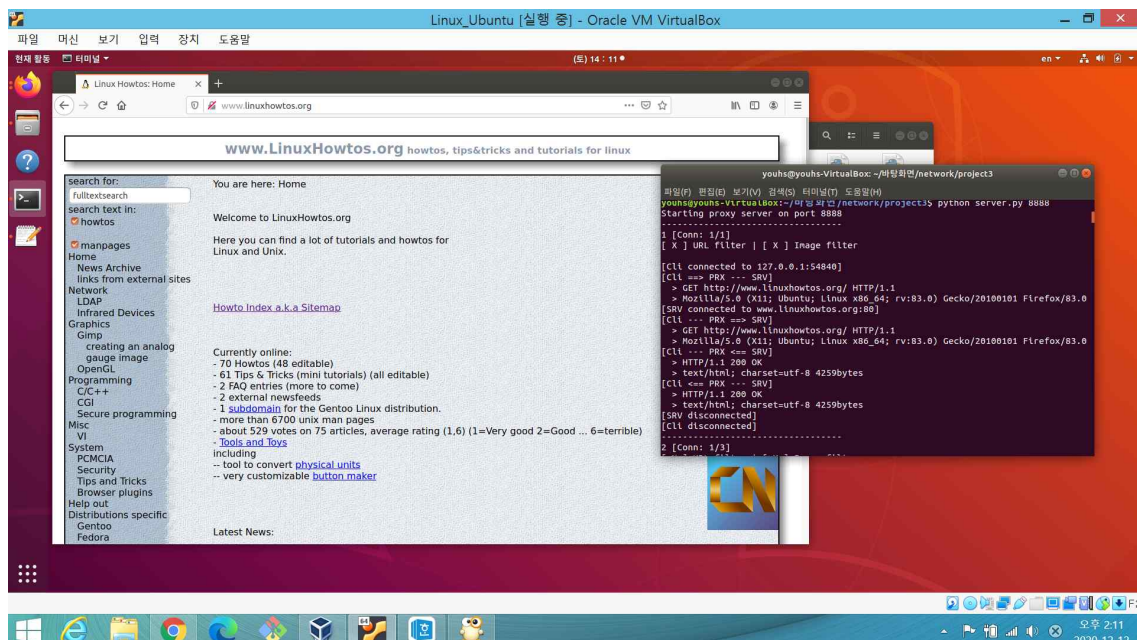
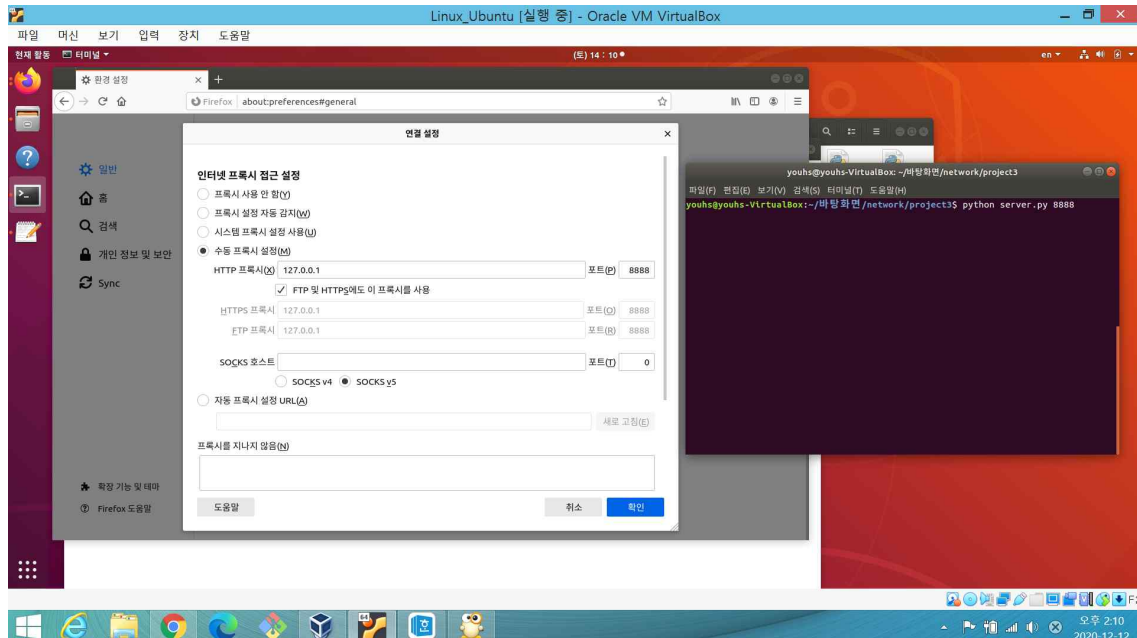
3. Flow chart or Diagram

<Client + Proxy + Destination Server>



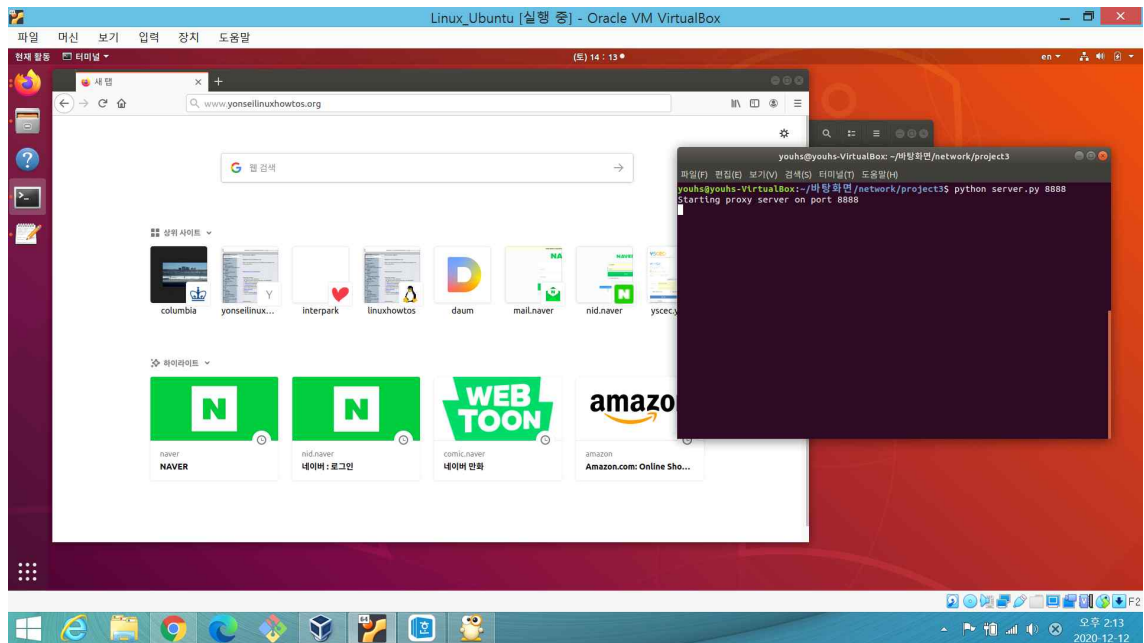
4. Snapshots

<Managing Two Socket + Brower's Proxy setting>

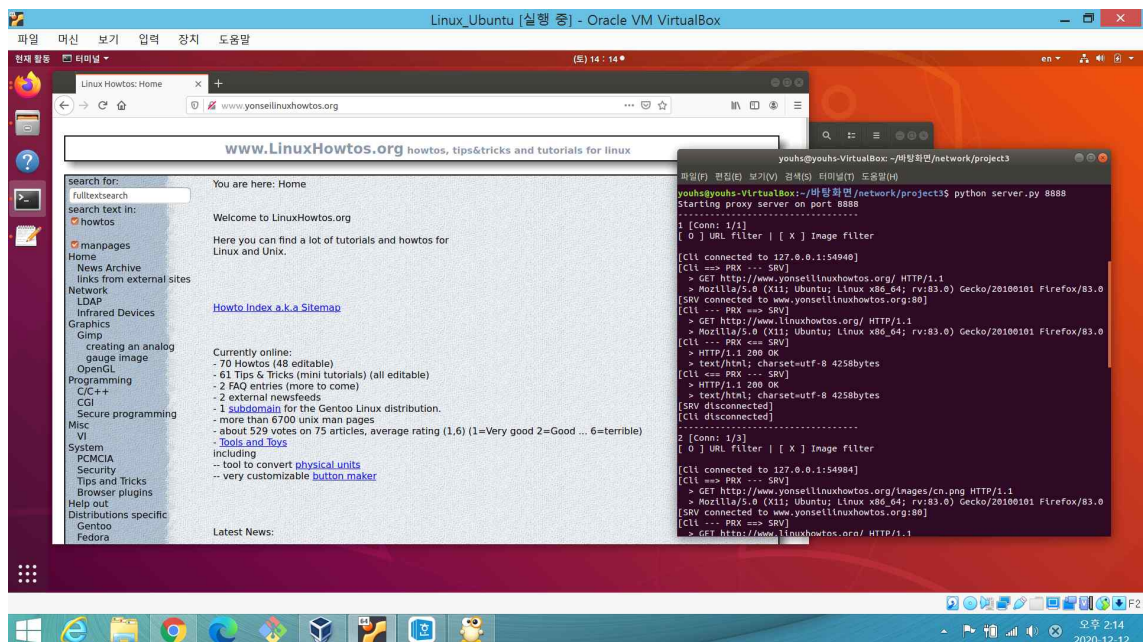


-> 로그를 보면 2가지 socket이 작용하는 것을 볼 수 있다

<URL Filtering and Redirection>

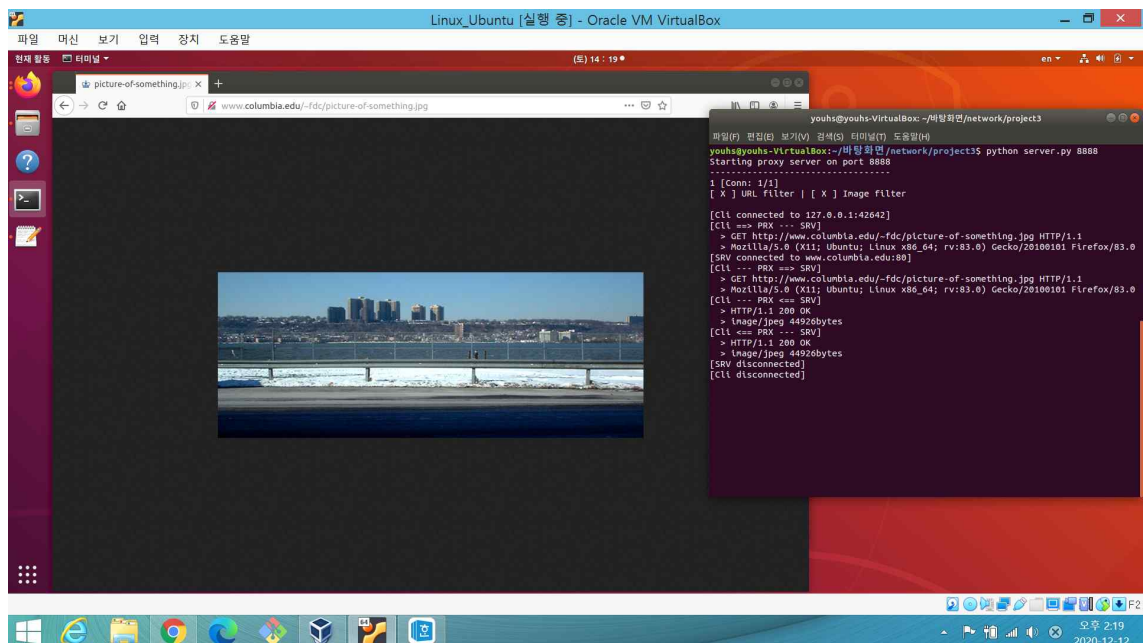


-> 접속하기 전 모습

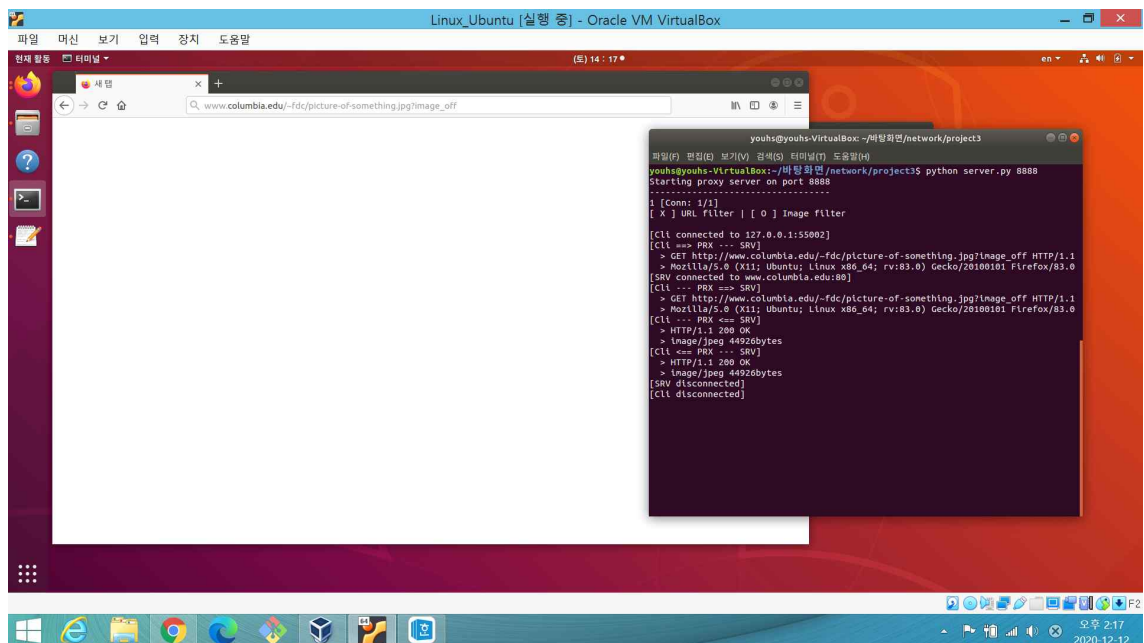


-> URL filtering을 통해 'yonsei' 를 포함 여부를 확인 후 존재 시 redirect 된 것을 확인할 수 있다

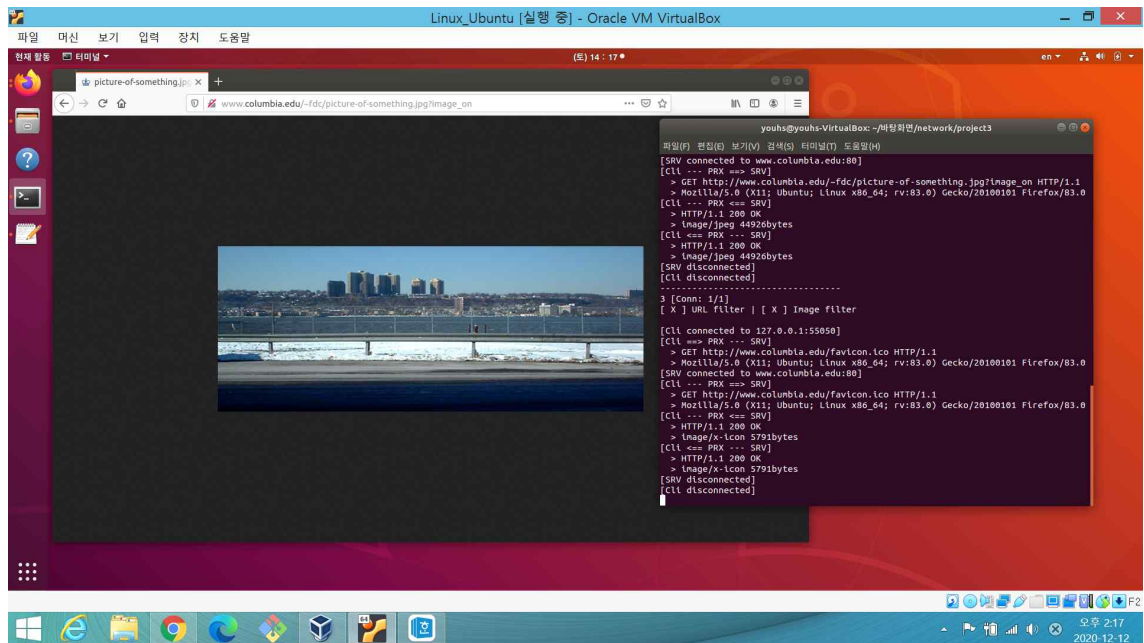
<Image filtering>



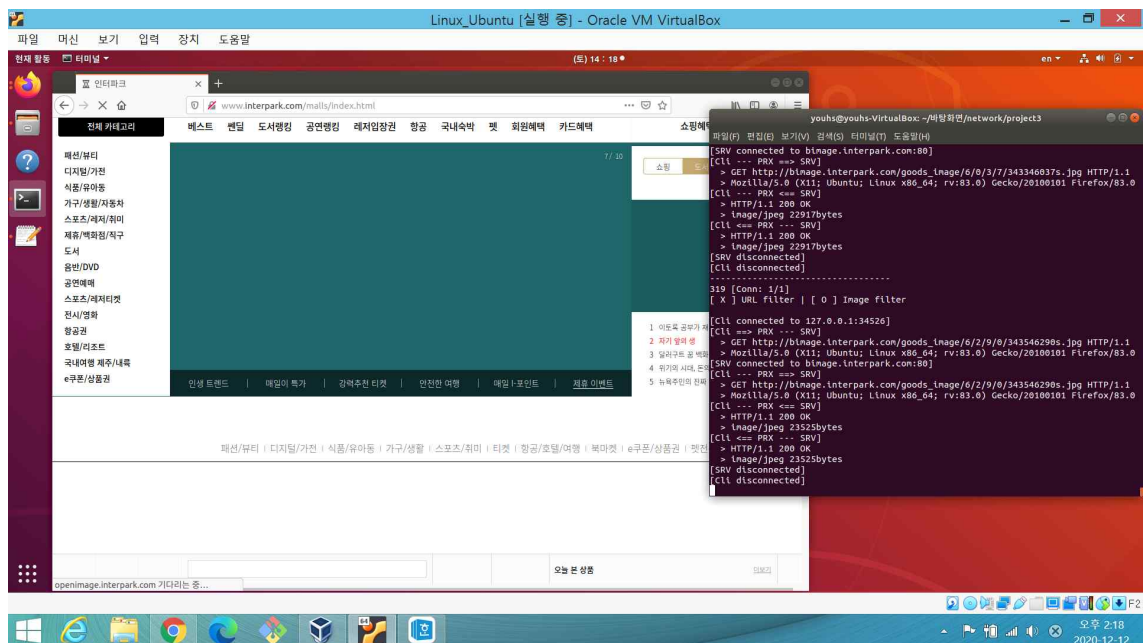
-> 기존 사이트



-> image off를 진행한 상태 이미지가 안 나오는 것을 확인할 수 있다



-> 다시 image on를 진행한 상태 잘 나오는 것을 확인할 수 있다



-> 이미지가 많이 존재하는 interpark 사이트를 들어가서 확인해 본 결과 모든 이미지가 출력되지 않은 것을 확인할 수 있다

4. Logical explanations

1. project.py

① main

```
#start of the main
HOST = '127.0.0.1'
PORT = sys.argv[1]
print("Starting proxy server on port " + PORT)
#Get the input for ip and port

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((HOST, int(PORT)))
server_socket.listen()
#setting the server option and function

while True:
    # when the client comes accept will return new socket and will communicate with socket in new thread
    try:
        client_socket, addr = server_socket.accept()
        th_a = threading.Thread(target = threaded, args =(client_socket, addr))
        th_a.daemon= True
        th_a.start()

    #when the KeyboardInterrupt occur
    except KeyboardInterrupt:
        server_socket.close()
        print("")
        print('exit')
        break

server_socket.close()
```

-> ip는 127.0.0.1인 localhost로 고정하고 인풋으로 들어올 port를 저장하고 그 저장된 값을 바탕으로 서버 socket을 만듭니다. 그 과정에서 사용되는 함수들은 저번에 설명한 함수들과 마찬가지로 사용되었습니다. 서버를 설정한 후 client socket이 들어올 때까지 대기합니다. client socket 즉 사용자가 작업을 하면 threaded 함수가 실행됩니다. 만약 ctrl + c가 발생하면 exit를 실행하고 socket을 종료하고 끝냅니다.

② Threaded (각 thread에서 사용될 함수)

I 도입부 (header 확인 및 URL filtering , redirection)

```
def threaded(client_socket, addr):
    boolget = 0
    global boolimage
    global boolredir
    global threadid

    data = client_socket.recv(4096)
    first= ' '
    second= ' '
    third= ' '
    first1= ' '
    first2= ' '
    second2 = ' '
    size_res = 0

    try:
        if not data:
            client_socket.close()
            #when the data is not exist
        else:
            first, second, third = prx_serv(data) #find out the host, http, user-agent
            data ,first1, second1 = redirection(data,first,second) #After redirection (yonsei is contain) also change the connection keep-alive ->close
```

-> client와 연결된 후 뒤에서 출력시 사용될 정보의 변수를 초기화해주고 만약 빈 data가 들어올 시 socket을 종료하고 data가 존재할 시 정보를 확인할 prx_serv 함수를 불러옵니다. 또한 URL filtering을 진행하기 위해서 'yonsei' 존재 여부를 확인하고 존재할 시 redirection 해줄 redirection 함수를 호출합니다.

I -1 prx_serv 함수

```
def prx_serv(data):
    global boolimage
    try:
        index = data.find(b'\r\n\r\n')
        cli_to_prx = data[:index].decode()
        if(cli_to_prx.find('GET') != -1):
            #check the header and get the info of URL,HOST,AGENT
            index=cli_to_prx.find('GET')
            index2=cli_to_prx.find('\n')
            first_line = cli_to_prx[index:index2-1]
            cli_to_prx = cli_to_prx[index2+1:]

            url = first_line[:9]
            if(url[-10]=='?' and url[-9:].find('image_off')!=-1):
                #Checking the if image filtering off or on
                boolimage = 1
            if(url[-9]!='?' and url[-8:].find('image_on')!=-1):
                boolimage = 0

            index=cli_to_prx.find('Host:')
            index2=cli_to_prx.find('\n')
            second_line = cli_to_prx[index+6:index2-1]
            cli_to_prx = cli_to_prx[index2+1:]

            index=cli_to_prx.find('Agent:')
            index2=cli_to_prx.find('\n')
            third_line = cli_to_prx[index+7:index2-1]
            cli_to_prx = cli_to_prx[index2+1:]

            return first_line,second_line,third_line

        else:
            #If header not contain the 'GET'
            first_line = '-1'

            index=cli_to_prx.find('Host:')
            cli_to_prx = cli_to_prx[index:]
            index2=cli_to_prx.find('\n')
            second_line = cli_to_prx[index+6:index2-1]
            third_line = '-1'

            return first_line,second_line,third_line

    except:
        # -> just get the HOST, other URL,AGENT is -1
        #when error occur
        first_line = '-1'
        second_line = '-1'
```

-> prx_serv 함수는 제일 처음 Client에서 넘어온 header를 분석하고 그 중에서 URL, HOST, USER-AGENT 정보를 각각 저장하는 역할을 합니다. 그 과정에서 URL에서 image filtering 여부를 체크해서 전역 변수인 boolimage에 전달해줍니다.

I -2 redirection 함수

```
def redirection(data,first,second):
    global boolredl
    cli_to_prx =data.decode()

    if (second.find('yonsei') ==-1):
        # Not yonsei -> just change the connection
        index=cli_to_prx.find('GET')
        index2=cli_to_prx.find('\n')
        index=cli_to_prx.find('GET')
        index2=cli_to_prx.find('\n')
        indextmp = index2
        index=cli_to_prx[index2+1:].find('Host')+indextmp
        index2=cli_to_prx[index2+1:].find('\n')+indextmp
        index=cli_to_prx.find('Host:')
        index2=cli_to_prx[index:].find('\n')+indextmp+1
        indextmp = index2
        index2=cli_to_prx[index2+1:].find('\n') +indextmp+1
        indextmp = index2
        index2=cli_to_prx[index2+1:].find('\n') +indextmp+1
        indextmp = index2
        index2=cli_to_prx[index2+1:].find('\n') +indextmp+1
        indextmp = index2
        index2=cli_to_prx[index2+1:].find('\n') +indextmp+1
        indextmp = index2
        index1=cli_to_prx[index2+1:].find('\n') + indextmp+1
        third = 'Connection: close'
        cli_to_prx = cli_to_prx.replace(cli_to_prx[index2+1:index1-1],third,1)
        data = cli_to_prx.encode()
        return data,first,second
    else:
        #return the original URL, HOST, data
        # yonsei contain -> change the URL and HOST also change the connection
        first = 'http://www.linuxhowtos.org/ HTTP/1.1'
        index=cli_to_prx.find('GET')
        index2=cli_to_prx.find('\n')
        cli_to_prx = cli_to_prx.replace((cli_to_prx[index+4:index2-1]),(first),1)
        index=cli_to_prx.find('GET')
        index2=cli_to_prx.find('\n')
        indextmp = index2
        second = 'www.linuxhowtos.org'
        index=cli_to_prx[index2+1:].find('Host')+indextmp
        index2=cli_to_prx[index2+1:].find('\n')+indextmp
        cli_to_prx = cli_to_prx.replace(cli_to_prx[index+7:index2],second,1)
        index=cli_to_prx.find('Host:')
        index2=cli_to_prx[index:].find('\n')+indextmp+1
        indextmp = index2
        index2=cli_to_prx[index2+1:].find('\n') +indextmp+1
        indextmp = index2
```

-> redirection 함수는 제일 처음 Client에서 넘어온 header를 분석하고 넘어온 URL, HOST를 확인한 후 'yonsei'가 포함되어 있으면 'www.linuxhowtos.org'로 redirection 해주기 위해서 URL과 HOST를 바꾸고 connection 역시 close로 변경해줍니다. 그리고 수정된 data를 저장하여 return 해줍니다. 만약 'yonsei'가 포함되어 있지 않으면 connection만 close로 변경해줍니다.

II 'GET'이 포함되지 않은 경우

```
if(first1 == '-1'):
    #check get of not if -1 -> not get just send to host
    proxy_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    if (second1 != '' or second1 != '-1'):
        proxy_socket.connect((second1,80))
        proxy_socket.send(data)
        while True:
            req_data = proxy_socket.recv(4096)
            client_socket.send(req_data)
            if not req_data:
                break
        proxy_socket.close()
```

-> header에 get이 포함되지 않은 경우로서 이런 경우에는 앞에서 알아낸 host로 http 연결을 진행한 후 그대로 destination server로 전송하고 수신 후 다시 client로 전달해준다. 모든 data가 수신 및 송신이 완료되면 사용되었던 proxy 및 client socket을 close 해줍니다.

III 'GET'이 포함된 경우

```
else:
    #check when it is get
    boolget=1
    #we can know the boolget by 1 when it is get option
    getnum=0
    for i in range(1,100):
        if (threadid[i] == 0):
            threadid[i] = -1
            getnum=i
            break
    proxy_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    proxy_socket.connect((second1,80))
    proxy_socket.send(data)
    #connect to the destination server
    a=0
    signal=0
    while True:
        if(a==0):
            req_data = proxy_socket.recv(1024)
            total_dat = req_data
            first2, second2 = prx_cli(req_data)
            if(boolimage==1):
                first3, second3, req_data, signal = image_filter(req_data)
                client_socket.send(req_data)
            else:
                req_data = proxy_socket.recv(1024)
                cur1_dat = total_dat
                total_dat = cur1_dat+ req_data
                if(boolimage == 0 or signal == 0 ):
                    client_socket.send(req_data)
            a +=1
            if not req_data:
                size_res =len(total_dat)
                break
        proxy_socket.close()
        client_socket.close()
    if (boolget==1):
        print_log(addr[0],addr[1],first,second,third,first1,size_res,first2,second2,getnum)
        boolredt = 0
        threadid[getnum] = 0
```

-> header에 get이 포함된 경우로서 이 경우 역시 앞에서 알아낸 host로 http 연결을 진행한 후 redirection 여부에 따라 나온 결과를 destination server로 전송하고 data를 수신한다. **prx_cli** 함수로 전달 받은 data를 해석해서 성공 여부를 저장하고 content type을 확인하여 저장합니다. 역시 앞에서 결정된 image filtering 여부에 따라서 **image_filter** 함수를 실행한다. **image_filter** 함수에서 나온 결과에 따라서 data에서 body 부분에 해당하는 정보의 전달 유무가 정해진다. 모든 data가 수신 및 송신이 완료되면 총 사용된 data의 크기를 구하고 사용되었던 proxy 및 client socket을 close 해줍니다. 마지막으로 지금까지의 과정이 담긴 정보들을 **print_log** 함수로 전달하여 log를 출력해줍니다.

III-1 prx_cli 함수

```
def prx_cli(data):                                     #proxy server's response info
    try:
        index = data.find(b'\r\n\r\n')
        cli_to_prx = data[:index].decode()
        index = cli_to_prx.find('\r\n')
        first_line = cli_to_prx[:index]
        if (cli_to_prx.find('Content-Type')== -1):      # check the http signal success or not
            second_line = 'Not specified'              # check the content type
        else:
            index = cli_to_prx.find('Content-Type:')
            if(cli_to_prx[index+14:].find('\r\n')== -1):
                second_line = cli_to_prx[index+14:]
            else:
                index2=cli_to_prx[index+14:].find('\r\n')
                second_line = cli_to_prx[index+14:index+index2+14]
        return first_line,second_line                  #Return the http info and content type
    except:
        first_line = 'ERROR'
        second_line = 'ERROR'
        return first_line,second_line
def prx_send(data):                                     #check the header and get the info of III
```

-> **prx_cli** 함수는 전달 받은 data를 해석해서 성공 여부를 저장하고 content type을 확인하여 저장합니다. 이 과정에서 content type이 정해지지 않은 header에 대해서는 'Not specified'로 저장을 해줍니다. 성공 여부는 header의 첫 번째 줄에 있는 내용으로 대체합니다.

III-2 image_filter 함수

```
def image_filter(data):                                #when the image filtering is happen
    try:
        index = data.find(b'\r\n\r\n')
        cli_to_prx = data[:index].decode()
        if (cli_to_prx.find('image')== -1):             #check the image or not (this case is not image)
            first_line = cli_to_prx[:index]
            if (cli_to_prx.find('Content-Type')== -1):
                second_line = 'Not specified'
        else:
            index = cli_to_prx.find('Content-Type:')
            if(cli_to_prx[index+14:].find('\r\n')== -1):
                second_line = cli_to_prx[index+14:]
            else:
                index2=cli_to_prx[index+14:].find('\r\n')
                second_line = cli_to_prx[index+14:index+index2+14]
        signal = 0
        return first_line,second_line,data,signal      # it mean this header not contain image
    except:
        first_line = 'ERROR'
        second_line = 'ERROR'
        signal = 0
        return first_line,second_line,data,signal      #when it is not image check the http and content type and return with signal

    else:
        first_line = cli_to_prx[:index]
        if (cli_to_prx.find('Content-Type')== -1):
            second_line = 'Not specified'
        else:
            index = cli_to_prx.find('Content-Type:')
            if(cli_to_prx[index+14:].find('\r\n')== -1):
                second_line = cli_to_prx[index+14:]
            else:
                index2=cli_to_prx[index+14:].find('\r\n')
                second_line = cli_to_prx[index+14:index+index2+14]
        data = cli_to_prx.encode()
        signal = 1
        return first_line,second_line,data,signal      # it mean this header contain image
    except:
        first_line = 'ERROR'
        second_line = 'ERROR'
        signal = 0
        return first_line,second_line,data,signal      #when it is image check the http and content type and return with signal
```

-> **image_filter** 함수는 전달 받은 header를 해석해서 image라는 단어를 체크해서 만약 header 안에 image라는 단어가 존재 시 signal을 1로 전달해서 image 여부를 전달해준다. 또한 image가 아닐 시 signal 0과 앞에 **prx_cli** 함수와 마찬가지로 성공 여부를 저장하고 content type을 확인하여 저장합니다. 이 과정에서 content type이 정해지지 않은 header에 대해서는 'Not specified'로 저장을 해줍니다. 성공 여부는 header의 첫 번째 줄에 있는 내용으로 대체합니다.

III-3 print_log 함수

```
def print_log(tp,port,first,second,third,first1,size_res,first2,second2,getnum): #print the whole log info
    global count
    global boolredl
    global boolimage
    count +=1
    print("-----")
    write = str(count) + ' [Conn: ' + str(getnum) + '/' + str(threading.activeCount()-1) + ']'
    print(write)
    urlf='X'
    imgf='X'
    if(boolredl==1):
        urlf='O'
    else:
        urlf='X'
    if(boolimage==1):
        imgf='O'
    else:
        imgf='X'
    print('[' + str(urlf) + ' ] ' + 'URL filter' [ ' + str(imgf) + ' ] Image filter')
    print()
    newInfo = "[cli connected to " + tp + ":" + str(port) + "]"
    print(newInfo)
    print("[cli ==> PRX --- SRV]")
    print(' > ' + first)
    print(' > ' + third)
    write = "[SRV connected to " + second + ":" + str(80) + "]"
    print(write)
    print("[cli --- PRX ==> SRV]")
    print(' > ' + first1)
    print(' > ' + third)
    print("[cli --- PRX <== SRV]")
    print(' > ' + first2)
    print(' > ' + str(second2) + ' ' + str(size_res) + 'bytes')
    print("[cli <== PRX --- SRV]")
    print(' > ' + first2)
    if(boolimage==1):
        print(" > Image filtering")
    else:
        print(' > ' + str(second2) + ' ' + str(size_res) + 'bytes')
    print("[SRV disconnected]")
    print("[cli disconnected]")
```

-> print_log 함수는 전달 받은 모든 정보들을 저장해서 출력해주는 역할을 합니다. 그 과정에서 Image filtering 및 URL filtering 적용 여부를 체크하여 log를 출력할 때 표시해줍니다. 또한 Image filtering이 적용 됐을 때는 Cli로 전달하는 size는 구하지 않는다.

IV 에러가 발생했을 때

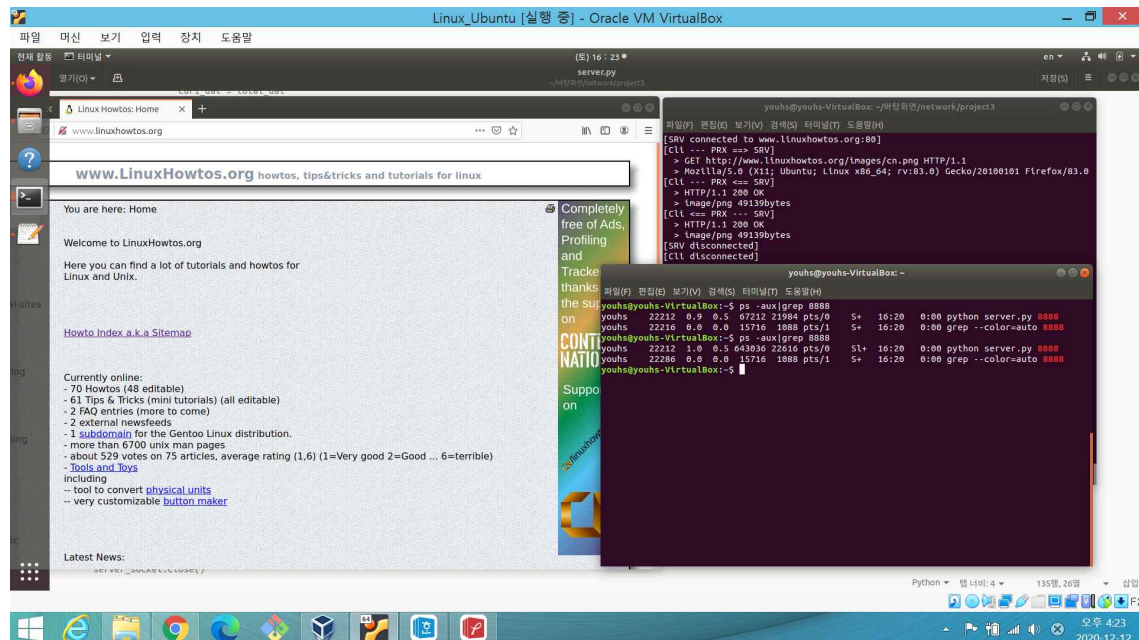
```
except:
    proxy_socket.close()
    client_socket.close()
```

-> 에러가 발생 시 기존에 쓰던 모든 socket 즉 proxy_socket 및 client_socket을 닫아준다.

5. Compare the performance with applying Muti thread and before in chart

1. using Threading

<memory>



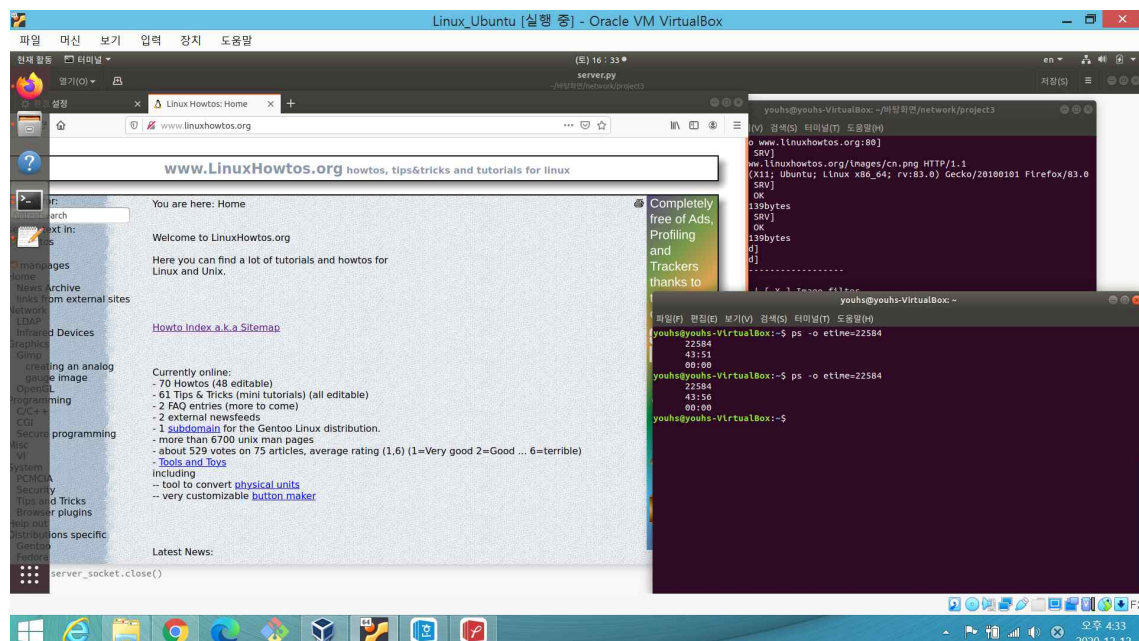
Cpu : 67212 -> 643036 변화

차이 = 575824

Memory : 21984 -> 22616 변화

차이 = 632

<time check>

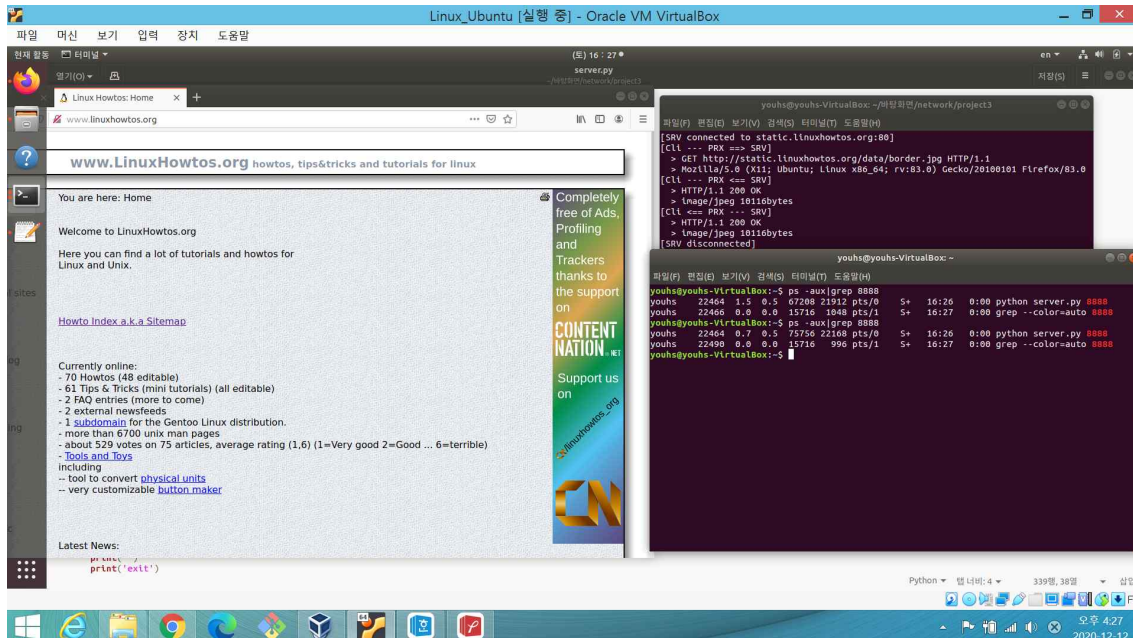


time : 43:51 -> 43:56

차이 = 5sec

2. not using threading

<memory>



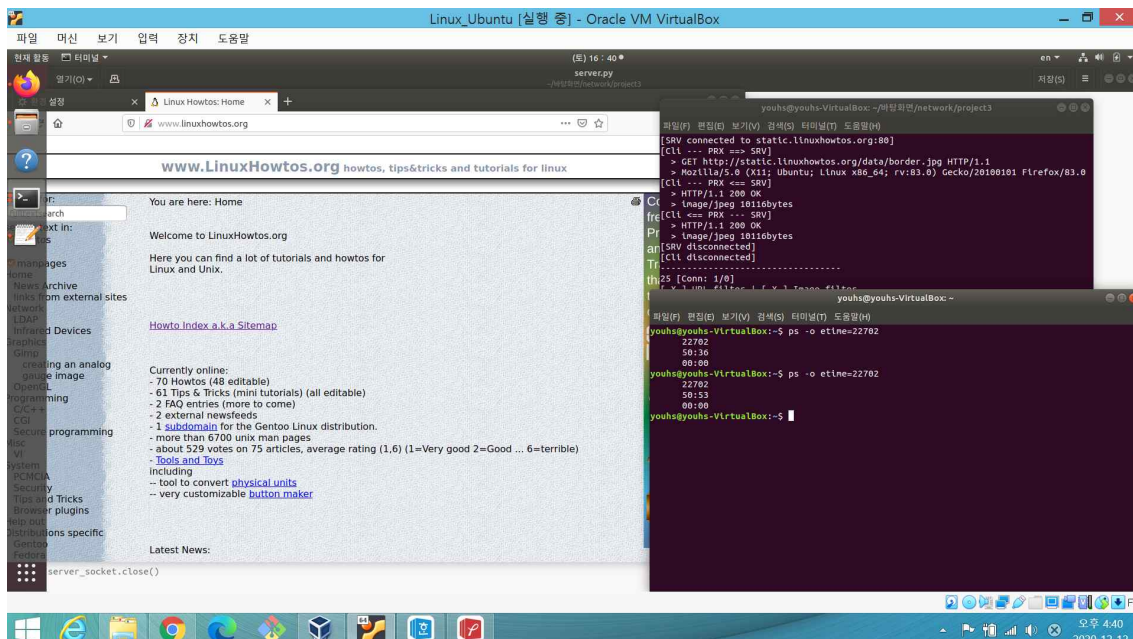
Cpu : 67208 -> 75756 변화

차이 = 69036

Memory : 21912 -> 22168 변화

차이 = 256

<time check>



time : 50:36 -> 50:53

차이 = 17sec

3. 비교

3-1 threading 이용 O

Cpu : 67212 -> 643036 변화	차이 = 575824
Memory : 21984 -> 22616 변화	차이 = 632
time : 43:51 -> 43:56	차이 = 5sec

3-2 threading 이용 X

Cpu : 67208 -> 75756 변화	차이 = 69036
Memory : 21912 -> 22168 변화	차이 = 256
time : 50:36 -> 50:53	차이 = 17sec

-> Cpu 나 Memory 부분에서 봤을 때는 threading을 이용하지 않는 것이 훨씬 효과적이었지만 시간적인 측면에서 봤을 때 threading을 이용하는 것이 훨씬 효과적이었습니다.