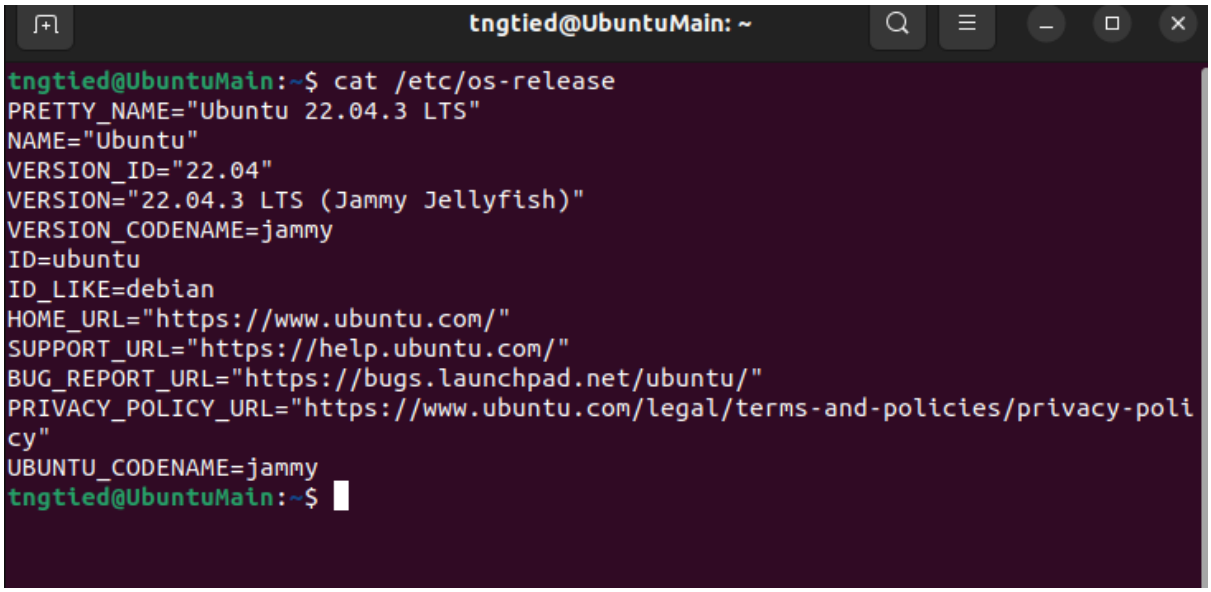
Computer Network

Assginment 3

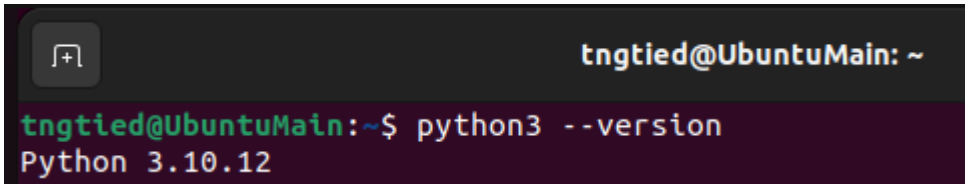
2020114026

1. Introduction/Reference (2pts)

a. Software environment



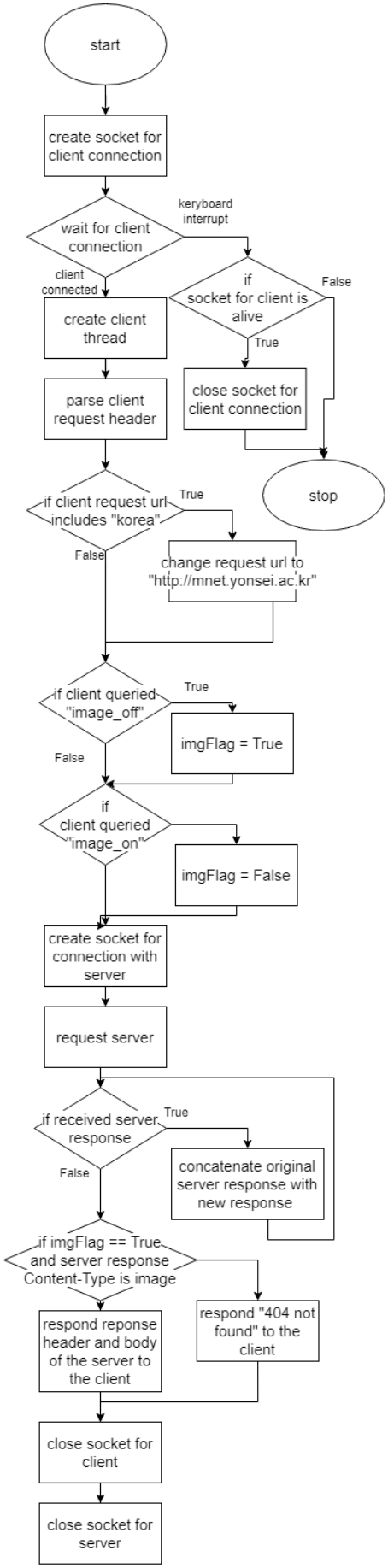
b. programming language



c. references

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers
- https://docs.python.org/3/library/urllib.parse.html
- https://stackoverflow.com/questions/36065792/binary-response-content-requests-lib
- https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest_API/Sending_and_Receiving_Binary_Data
- https://stackoverflow.com/questions/34252273/what-is-the-difference-between-socket-send-and-socket-sendall

2. Flow chart or Diagram (8pts)


start
create socket for client connection
wait for client connection
keryboard interrupt
client connected
if socket for client is alive
False
True
create client thread
close socket for client connection
parse client request header
stop
if client request url includes "korea"
True
False
change request url to "http://mnet.yonsei.ac.kr"
if client queried "image_off"
True
imgFlag = True
False
if client queried "image_on"
imgFlag = False
create socket for connection with server
request server
if received server response
True
concatenate original server response with new response
False
if imgFlag == True and server response Content-Type is image
respond reponse header and body of the server to the client
respond "404 not found" to the client
close socket for client
close socket for server

3. Logical explanations block by block in detail. (10pts)

a. imports and global declaration

import sys import socket from urllib.parse import urlparse import threading	Import sys to get port number argument when running the prx.py file Import socket to open and run and close sockets for client and server connection Import urlip.parse.urlpase to parse the url and extract domain, path, query from url Import threading for client threading
port = sys.argv[1] request_count = 0 imgFlag = [False]	Get port number from system argument variable Initialize request count variable for printing Initialize image filter flag variable by false

b. function parse_header

def parse_header(req_lines):	Get list of lines of string as an argument
header = {}	Initialize python dictionary variable to contain headers

for line in req_lines:
header_parts = line.split(": ")
if (len(header_parts) > 1):
header[header_parts[0]] = header_parts[1]
	For list variable string in req_lines, split by ": ". if the splitted items are more than 1 then it means the string was header line. Then add the key and value of header into the dictionary

return header	Return list of headers in a form of dictionary

c. function handle_client

def handle_client(CLI_conn, CLI_addr): global imgFlag, request_count	Get client connection and address as an argument. Make global variable image flag(imgFlag) and request_count usuable in function.
try: CLI_req_headerlines, CLI_req_, CLI_req_body = CLI_conn.recv(4096).partition(b'₩r₩n₩r₩n') except (OSError, KeyboardInterrupt) as e: return CLI_req_headerlines = CLI_req_headerlines.decode("utf-8").split("₩r₩n")	Try to receive client request and split it by '₩r₩n₩r₩n' to get header and body. If keyboard interrupt happens, then terminate the function. Decode header by 'utf-8' and split it line by line.
request_count += 1 print("----------") korFlag = False CLI_req_path = CLI_req_headerlines[0].split(' ')[1] parsed_url = urlparse(CLI_req_path)	Cummulate request count, and initialize korFlag which indicates if the client request has to be redirected or not. Get client request from header and parse it by urlip.parse.urlpase.
if ("korea" in CLI_req_path): korFlag = True parsed_url = urlparse("http://mnet.yonsei.ac.kr/")	If 'korea' is included in url then set korFlag True and again set parsed_url with http://mnet.yonsei.ac.kr/ to redirect

if (parsed_url.query == "image_off"):
imgFlag[0] = True
if (parsed_url.query == "image_on"):
imgFlag[0] = False
	If extracted query of the parsed url is "image_off" then set image flag True.
	If extracted query of the parsed url is "image_on" then set image flag True.

print("%d [%c] Redirected [%c] Image filter" % (request_count, ("O" if korFlag else "X"), ("O" if imgFlag[0] else "X")))	Print current status of the proxy server.

client_ip, client_port = CLI_addr print(f"[CLI connected to {client_ip}:{client_port}]") CLI_req_header = parse_header(CLI_req_headerlines) print("[CLI ==> PRX --- SRV]") print(" > %s" % (CLI_req_headerlines[0])) print(" > %s" % (CLI_req_header['User-Agent'].splitlines()[0]))	
SRV_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) SRV_socket.connect((parsed_url.hostname, 80)) SRV_addr = SRV_socket.getpeername() print(f"[SRV connected to {SRV_addr[0]}:{SRV_addr[1]}]")	Create socket to connect with the server(requested url host). Set socket TCP and connect by port 80 which is used for HTTP connection. Get actual address used to connect again, and print it.

print("[CLI --- PRX ==> SRV]")
SRV_req_headers = {
"Host" : CLI_req_header["Host"],
'GET' : parsed_url.path,
"Connection" : "close",
"User-Agent" : CLI_req_header['User-Agent'],
'Accept': CLI_req_header['Accept']
}
if ('Accept-Language' in SRV_req_headers.keys()):
SRV_req_headers['Accept-Language']: CLI_req_header['Accept-Language']
	Write headers to request the server based on the request header of the client.
	Set connection "close" to prevent persistent connection.

SRV_req_str = f"GET {parsed_url.path} HTTP/1.1₩r₩n" for key, value in SRV_req_headers.items(): SRV_req_str += f"{key}: {value}₩r₩n" SRV_req_str += "₩r₩n" SRV_socket.sendall(SRV_req_str.encode('utf-8') + CLI_req_ + CLI_req_body)	Write header in string object first based on the header written above, then encode it with 'utf-8', and add '₩r₩n₩r₩n' and request body. Send the whole request to the server.
print(" > %s" % (parsed_url.hostname + parsed_url.path)) print(" > %s" % (CLI_req_header['User-Agent'].splitlines()[0])) print("[CLI --- PRX <== SRV]")	Print current status of the proxy server.

SRV_res = b''
while True:
try:
chunk = SRV_socket.recv(4069)
if not chunk:
break
SRV_res += chunk
 	
except socket.error:
break
	While the server is sending the response, receive response and add it to the previously received responses. If server has not sent anything, then break out of the while loop.

SRV_res_headerlines, SRV_res_, SRV_recv_body = SRV_res.partition(b'₩r₩n₩r₩n') SRV_res_headerlines = SRV_res_headerlines.decode('utf-8').split("₩r₩n") SRV_res_headers = parse_header(SRV_res_headerlines) SRV_res_status = SRV_res_headerlines[0]	Split the response of the server into header and body. Convert header object(which is in utf-8 encoded form) into list of header strings.
print(" > %s" % (SRV_res_status)) if ('Content-Length' in SRV_req_headers.keys()):	Print response status and content length.

Content_Length = SRV_res_headers['Content-Length'] else: Content_Length = len(SRV_recv_body) print(" > %s %sbytes" % (SRV_res_headers['Content-Type'], Content_Length)) print("[CLI <== PRX --- SRV]")	
notFoundFlag = False if (imgFlag[0] and SRV_res_headers["Content-Type"][0:5] == "image"): CLI_res_status = "404 Not Found" CLI_res_str = "HTTP/1.1 404 Not Found₩r₩nConnection: close₩r₩n₩r₩n".encode("utf-8") + SRV_res_ notFoundFlag = True	Initiate notFoundFlag which indicates if the response to the client is 404 or not by false. If imgFlag is True and the "Content-Type" of the server response is image, then write response to the client "404 Not Found" and set the notFoundFlag True.

else:
CLI_res_status = SRV_res_status
CLI_res_str = SRV_res_headerlines[0]
CLI_res_header = SRV_res_headers
if (imgFlag[0]):
CLI_res_header['Content-Security-Policy'] = "default-src 'self'; img-src ;"
for key, value in CLI_res_header.items():
CLI_res_str += f"{key}: {value}₩r₩n"
CLI_res_str += "₩r₩n"
CLI_res_str = CLI_res_str.encode("utf-8")
CLI_res_str += SRV_recv_body
	If not then write header of the response for the client with the server response header dictionary.
	If imageflag is true then write header 'Content-Security-Policy' to make source of images none.
	Endcode header with 'utf-8' and response body of the server behind it, with '₩r₩n₩r₩n' to indicate header and body separation between them.

CLI_conn.sendall(CLI_res_str) print(" > %s" % (CLI_res_status)) if (not notFoundFlag): print(" > %s %sbytes" % (SRV_res_headers['Content-Type'], Content_Length))	Send the response written above to the client and print the current status of the proxy server. If the sent response is 404 not found then don't print the type and bytes of the content.
CLI_conn.close() print("[CLI disconnected]") SRV_socket.close() print("[SRV disconnected]")	Close the client connection and print it is closed. Close the server socket and print it is closed.

d. function run_proxy_server

def run_proxy_server(): server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) server.bind(('127.0.0.1', port)) server.listen(5) print("Starting proxy server on port %d" % port)	create http tcp socket to get client request. Bind the socket with localhost and port number received from the system argument. Print that the proxy server has started running.

while True:
try:
client_conn, addr = server.accept()
client_handler = threading.Thread(target=handle_client, args=(client_conn, addr))
client_handler.start()
	Start while loop which accepts client connection and start proxy thread.

except KeyboardInterrupt:	If keyboard interrupt happens, then catch it. If

try: client_conn.close() except UnboundLocalError: pass break server.close()	client socket is alive then close it and break out of the loop. If loop has closed then close the socket.

e. main

if __name__ == '__main__': run_proxy_server()	If the file has started as main then run proxy server by calling the function.