Assignment 1 Report

Yi Ren 20304391

I used python for my assignment of web proxy server.

In order to respond to HTTP & HTTPS requests, I implemented the socket package in python library. I wrote a *ClientRequest* class to deal with the HTTP/HTTPS requests, which interprets the request messages and gets the target host and port number. For the HTTP requests, the proxy server simply forwards the request to the web server, receives the reply and transfers the reply back to the client. For the HTTPS requests, the proxy server connects with the destination web server first, then send a confirmation message back to the client in order to receive the data to be sent out from the client. After receiving the data from the client, the proxy transfers the data to the web server, gets back from the web server and sends the reply to the client.

I imported the threading package in python library in order to implement a multithreading proxy server. Every time the proxy socket accepts a request from a client, the program generates a new thread to deal with the request. Thus, it is able to handle multiple requests simultaneously.

At the beginning of the program, some user inputs are required from the management console. The user is asked to type in any website URL he/she wants to block or release. The blocked URL will not be connected using the proxy server.

I also wrote a class named *WebCache* which stores the previous HTTP requests. Every time a HTTP request is handled, the data sent back from the web server will be updated into the cache, which is a dictionary written in my python file. The program scans the new HTTP requests to see if there was a same request before. When the dictionary is full, the oldest request saved in the cache will be popped out so that there will always be space for new requests to store.

Below is the source code for the server.py file of my program.

```
import datetime

import socket, sys, threading

blacklist = []
HOST = '127.0.0.1'
PORT = 4000
```

```
LISTEN = 10
BUFLEN = 1024
MAX_CACHE = 15

'''The class that deal with HTTP/HTTPS requests sent from the client'''


class ClientRequest:
    def __init__(self, data, s_client):
        self.__parse(data, s_client)

    def __parse(self, data, s_client):
        url_line = ''
        try:
            first_line = data.split("\r\n")[0]    # Get the first line of the request
            url_line = first_line.split(" ")[1]    # Get the url from the first line
            self.req_type = first_line.split()[2]    # Get the HTTP version from the first line
        except IndexError:
            print("INDEX ERROR")
            s_client.close()

        url_pos = url_line.find("://")    # Remove the http prefix from the url
        if url_pos == -1:
            tmp_url = url_line
        else:
            tmp_url = url_line[(url_pos + 3):]
        port_pos = tmp_url.find(":")    # Get the position of the port number
        if port_pos == -1:    # If not specified, send the request to port 80 of the web server
            self.port = 80
            self.host = tmp_url
        else:
            self.port = int(tmp_url[port_pos + 1:])
            self.host = tmp_url[:port_pos]

        if self.host in blacklist:    # If the url is blocked, close the client socket
            print("The website you are requesting is blocked.")
            s_client.close()
            return


'''The class of the web cache that stores the previous HTTP requests'''


class WebCache(object):
    def __init__(self):
        self.cache = {}
        self.max_size = MAX_CACHE

    def __contains__(self, key):    # Check if the URL requested is in the cache
        return key in self.cache

    def update(self, key, value):    # Store new HTTP requests into the cache
        if not __contains__(self, key) and len(self.cache) >= self.max_size:
            self.remove_oldest()
        self.cache[key] = {'time': datetime.datetime.now(),
                           'value': value}

    def remove_oldest(self):    # If the cache is full, throw away the oldest requests
        oldest = None
        for key in self.cache:
            if oldest is None:
                oldest = key
            elif self.cache[key]['time'] < self.cache[oldest]['time']:
                oldest = key
        self.cache.pop(oldest)

    def get(self, key):
        return self.cache[key]['value']


'''The main class for the proxy server'''


class ProxyServer:

    def __init__(self):
        self.s_proxy = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.s_proxy.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.s_proxy.bind((HOST, PORT))
        self.s_proxy.listen(LISTEN)
        print("Proxy Server starts.")

    '''Update the blocked URL list'''

    def __blockWeb(self):
        while True:
            blocked = input("Please input the website you want to block or release\nTo
block website please enter \'b^\'"
```

```
                                    "before the complete URL, to release please enter \'r^\'\n"
                                    "To finish website blocking commands, please enter \'fff\':
\n")
            if blocked == "exit":
                print("Proxy server closed.")
                self.s_proxy.close()
                sys.exit(0)
            elif blocked.startswith("b^"):
                index = blocked.find("b^")
                blacklist.append(blocked[index+2:])
            elif blocked.startswith("r^"):
                index = blocked.find("r^")
                blacklist.remove(blocked[index+2:])
            elif blocked == "fff":
                return

    def __handle_request(self, s_client, addr):
        print("Accept a request from client: ", addr)
        req_data = b''
        webcache = WebCache()

        while True:   # Collect the request sent by the client until all data has arrived
            try:
                request = s_client.recv(BUFLEN)
                req_data += request
            except BlockingIOError as e:
                break
        # print(req_data.decode())
        client_req = ClientRequest(req_data.decode(), s_client)
        try:
            server_host = socket.gethostbyname(client_req.host)   # Get the IP address
from the URL
        except socket.gaierror:
            print("ERROR: IP address not found.")   # Close if cannot find the IP address
            s_client.close()
            return
        server_port = client_req.port
        print("----------\nTarget host:", client_req.host)    # Print out the request
information to the console
        print("Destination IP address:", server_host)
        print("Server port:", server_port, "\n----------")

        s_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)   # Initialize the
socket for the web server
```

```
        s_server.connect((client_req.host, server_port))
        s_server.settimeout(10)

        if server_port == 443:    # If target port number is 443, then it is an HTTPS request
            success_msg = ('%s %s OK\r\n\r\n' % (client_req.req_type, '200')).encode()
            s_client.send(success_msg)    # Send HTTPS response back to client
            s_server.setblocking(0)
            while True:
                try:
                    cli_data = s_client.recv(BUFLEN)    # Receive the data from the client
                    if not cli_data:
                        break
                    s_server.send(cli_data)    # Forward the request to the web server
                except Exception as e:
                    pass
                try:
                    rep_data = s_server.recv(BUFLEN)    # Receive the reply from the web
server
                    if not rep_data:
                        break
                    s_client.send(rep_data)    # Forward the reply back to the client
                except Exception as e:
                    pass

            s_server.shutdown(socket.SHUT_RDWR)
            s_server.close()
            s_client.close()
            return

        else:    # If port number is not 443, then it is an HTTP request
            if webcache.__contains__(client_req.host):    # If the requested URL is in cache
                print("URL found in cache.")
                rep_data = webcache.get(client_req.host)
                s_client.send(rep_data)    # Send the data in cache to the client
                s_client.close()
                return
            else:
                s_server.send(req_data)    # Forward the request to the web server if not in
cache
                while True:
                    try:
                        rep_data = s_server.recv(BUFLEN)    # Receive reply from web
server
                        if not rep_data:
```

```
                        break
                    s_client.send(rep_data)    # Forward to the client
                except Exception as e:
                    pass
            webcache.update(client_req.host, rep_data)    # Store the new request to
the cache

    def startProxy(self):
        self.__blockWeb()
        while True:
            s_client, addr = self.s_proxy.accept()
            s_client.setblocking(0)
            t = threading.Thread(target=self.__handle_request, args=(s_client, addr))    #
Allow multithreading
            t.setDaemon(True)
            t.start()

if __name__ == '__main__':
    ProxyServer().startProxy()
```