دانشگاه صنعتی شریف

دانشکده مهندسی برق

شبکه مخابرات داده ها

گزارش پروژه ی پایانی

استاد: دکتر پاکروان

ارائه کننده: کامیار رجبعلی فردی (97101661)

فهرست مطالب

پیاده سازی

ابتدا توصیحاتی درباره ی روش پیاده سازی و همچنین مختصر توضیحی درباره ی کد های زده شده می‌دهیم.

توابع پیاده سازی شده در Server

1. Stat_HTTP_disconnect

```
def stat_HTTP_disconnect(a_string):
#This function checks the type of the message. The types are written below:
# 1. stat : asks for statistical information of transmissions
# 2. disconnect : in order to stop the session
# 3. HTTP : Messages based on http protocol
    if a_string == 'number of connected clients' or a_string == 'file type stats' or a_string == 'request stats' or a_string =='response stats':
        return 'stat'

    if a_string == 'disconnect':
        return 'disconnect'

    else:
        return 'HTTP'
```

2. string_with_calculated_length

```
def string_with_calculated_length(a_string):
# This function calculates the length of the messages sent by server to client to inform
# the invalidity of the received message.
        location_of_length = 2                              # location of length number
        length_of_string = len(a_string)                    #length of string without length number
        length_of_the_length_of_string = len(str(length_of_string))
        length_of_string += int(length_of_the_length_of_string)     #length of string with length number
        lines_of_string = a_string.split('\n')
        lines_of_string[location_of_length] += str(length_of_string)
        return '\n'.join(lines_of_string)                   #convert list to string
```

در این تابع طول پیام هایی که به عنوان خطا از سرور به کلاینت فرستاده می‌شود محاسبه شده و با همین پیام ارسال می‌شود. در این تابع ابتدا طول رشته را زمانی که فیلد اندازه خالی است محاسبه کرده سپس با تعداد ارقام همین عدد جمع می‌کنیم و در انتها فیلد مربوط به طول رشته را پر می‌کنیم.

3. HTTP_msg

```
def HTTP_msg(a_string, addr):
# This function creates proper response to the received message.
    lines_of_string = a_string.split('\n')                  # spilt string in to lines
    body_loc = -1                                           # location of statring body part of the message
    for i in range(0,len(lines_of_string)):
        if lines_of_string[i] == '':
            break
        body_loc = i                                        # location of body found!

    if body_loc == -1:                                      # if the message has no body part
        b = lines_of_string[0::]
    else:                                                   # if the message consists of body part
        b = lines_of_string[0:body_loc+1]

    first_line = b[0].split(' ')                            # extract first line
    flag = 0

    for i in range(1,len(b)):                               # to check if ': ' is used properly
        temp = b[i].split(': ')
        if len(temp) == 2:
            continue
        flag = 1
        break
```

این تابع رشته ی دریافتی از طرف کلاینت و آدرس آن را به عنوان ورودی می‌گیرد و متناسب با پیام فرستاده شده یک پیام مناسب تولید می‌کند. علاوه بر این اطلاعات آماری پیام دریافتی را در فایل های JSON که در آینده به آنها می‌پردازیم می‌نویسد. این تابع دارای 4 خروجی است که یکی از آنها پیام تولیدی برای ارسال به کلاینت است. یکی دیگر از خروجی ها طول فایل مورد نظر و یکی دیگر از آنها خط اول پیام دریافتی و در انتها خروجی چهارم متدی است که در پیام دریافتی قرار دارد. اگر پیام به گونه ای باشد که هر کدام از این خروجی ها برای آن تعریف نشده باشد، مقدار '-1' برگردانده می‌شود. در قسمت اول تابع سعی شده است تا خط اول، موقعیت بدنه و سرآیند و دیگر پارامتر های مهم پیام استخراج شود.

```
    # 400 bad request
    version = first_line[-1].split('/')[-1]                         # extract version of http protocol
    HTTP_versions = ['1.0', '1.1']
    if flag == 1 or len(first_line) != 3 or HTTP_versions.count(version) == 0:
        request_time = str(datetime.datetime.now())
        msg = 'HTTP/1.0 400 Bad Request\n'                                                                          +\
              'Connetion: close\n'                                                                                  +\
              'Content-Length: ' + '\n'                                                                             +\
              'Content-Type: text/html\n'                                                                           +\
              'Date: ' + request_time + '\n\n'                                                                      +\
              '<html><body><h1>BADREQUEST!</h1></body></html>'
        msg = string_with_calculated_length(msg)                    # insert length of the message

        # Save the statistical data of this message in JSON files
        JSON_LOG(addr, lines_of_string[0].split(' ')[0], '400 Bad Request', request_time)
        JSON_REQUEST_STAT(a_string.split('\n')[0].split(' ')[0])
        JSON_RESPONSE_STAT('400')

        return ('-1', '-1', msg, '-1')
# 501 Not Implemented
service_type = ['GET','HEAD','DELETE','POST','PUT']
if service_type.count(first_line[0]) == 0:
    request_time = str(datetime.datetime.now())                 # calculate date time
    msg = 'HTTP/1.0 501 Not Implemented\n'                                                                          +\
          'Connetion: close\n'                                                                                      +\
          'Content-Length: ' + '\n'                                                                                 +\
          'Content-Type: text/html\n'                                                                               +\
          'Date: ' + request_time + '\n\n'                                                                          +\
          '<html><body><h1>NOTIMPLEMENTED!</h1></body></html>'
    msg = string_with_calculated_length(msg)

    # Save the statistical data of this message in JSON files
    JSON_LOG(addr, lines_of_string[0].split(' ')[0], '501 Not Implemented', request_time)
    JSON_REQUEST_STAT('Improper')
    JSON_RESPONSE_STAT('501')

    #(firstline, length, msg, method)
    return ('-1', '-1', msg, '-1')
# 405 Method Not Allowed
method = ['GET', 'POST']
if method.count(first_line[0]) == 0:
    request_time = str(datetime.datetime.now())
    msg = 'HTTP/1.0 405 Method Not Allowed\n'                                                                       +\
          'Connetion: close\nContent-Length: ' + '\n'                                                               +\
          'Content-Type: text/html\n'                                                                               +\
          'Allow: GET\n'                                                                                            +\
          'Date: ' + request_time + '\n\n'                                                                          +\
          '<html><body><h1>NOTALLOWED!</h1></body></html>'
    msg = string_with_calculated_length(msg)

    # Save the statistical data of this message in JSON files
    JSON_LOG(addr, lines_of_string[0].split(' ')[0], '405 Method Not Allowed', request_time)
    JSON_REQUEST_STAT(lines_of_string[0].split(' ')[0])
    JSON_RESPONSE_STAT('405')

    return ('-1', '-1', msg, '-1')
```

در بخش های بعد که در صفحه ی قبل آورده شده است ارور های 405 ,501 ,400 بررسی می‌شوند و اطلاعات آماری در فایل های جیسون ذخیره می‌شود.

```
# GET 200 OK
if first_line[0] == 'GET':
    request_time = str(datetime.datetime.now())
    if os.path.isfile(first_line[1]):                          # check if the file is existed
        type_of_file = os.path.splitext(first_line[1])[1][1::]  # extract type of file
        if type_of_file == 'jpg' or type_of_file == 'png':
            type_of_file = 'image/' + type_of_file
        else:
            type_of_file = 'text/' + type_of_file

        length_of_file = os.path.getsize(first_line[1])
        msg = 'HTTP/1.0 200 OK\n'                                                                  +\
              'Connetion: close\n'                                                                 +\
              'Content-Length: ' + str(length_of_file) + '\n'                                      +\
              'Content-Type: ' + type_of_file + '\n'                                               +\
              'Date: ' + request_time

        # Save the statistical data of this message in JSON files
        JSON_LOG(addr, lines_of_string[0].split(' ')[0], '200 OK', request_time)
        JSON_FILE_TYPE_STAT(type_of_file)
        JSON_REQUEST_STAT("GET")
        JSON_RESPONSE_STAT('200')
        return (first_line[1], length_of_file, msg, 'GET')
    # 301 Moved Permanently
    else:
        request_time = str(datetime.datetime.now())
        msg = 'HTTP/1.0 301 Moved Permanently\n'                                                   +\
              'Connetion: close\n'                                                                 +\
              'Content-Length: ' + '\n'                                                            +\
              'Content-Type: text/html\n'                                                          +\
              'Date: ' + request_time + '\n\n'                                                     +\
              '<html><body><h1>MOVEDPERMANENTLY!</h1></body></html>'
        msg = string_with_calculated_length(msg)

        # Save the statistical data of this message in JSON files
        JSON_LOG(addr, lines_of_string[0].split(' ')[0], '301 Moved Permanently', request_time)
        JSON_REQUEST_STAT("GET")
        JSON_RESPONSE_STAT('301')

        return ('-1','-1', msg, '-1')
```

در ادامه پیام GET را بررسی می‌کنیم. اگر فایل ارسالی وجود داشته باشد آنگاه URL را استخراج کرده و پس از آن ادرس فایل مورد نظر و نوع فایل استخراج می‌شود و در انتها خط اول پیام، طول فایل، پیام پاسخ و رشته ی 'GET' برگردانده می‌شود. اگر فایل موردنظر وجود نداشته باشد آنگاه پیام 301 ارسال خواهد شد.

```
# HTTP 200 ok
if first_line[0] == 'POST' and len(lines_of_string) == body_loc+1:
    type_of_file = lines_of_string[body_loc-1].split(': ')[1].split('/')[1]
    size_of_file = a_string.split('\n')[-3].split(': ')[1]
    msg = 'HTTP/1.0 200 OK' + '\n'                                                                 +\
          'Connetion: close\nContent-Length: ' + str(len(a_string)) + '\n' +\
          'Content-Type: text/html' + '\n'                                                         +\
          'Date: ' + str(datetime.datetime.now()) + '\n\n'                                         +\
          '<html><body><h1>POST!</h1></body></html>'

    # Save the statistical data of this message in JSON files
    JSON_REQUEST_STAT("POST")
    JSON_RESPONSE_STAT('200')

    return ('Server\\'+first_line[1]+'.'+ type_of_file, size_of_file, msg, 'POST')
```

بعد از آن پیام POST با بدنه ی معتبر مورد بررسی قرار می‌گیرد و سرور سعی می‌کند تا از پیام دریافتی نوع و اندازه ی فایل را بیابد.

اگر بدنه معتبر نباشد بصورت زیر عمل می‌کنیم:

```
# 403 Forbidden
if True:
    request_time = str(datetime.datetime.now())
    msg = 'HTTP/1.0 403 Forbidden\n'                                                                +\
          'Connetion: close\n'                                                                      +\
          'Content-Length: ' + '\n'                                                                 +\
          'Content-Type: text/html\n'                                                               +\
          'Date: ' + request_time + '\n\n'                                                          +\
          '<html><body><h1>FORBIDDEN!</h1></body></html>'
    msg = string_with_calculated_length(msg)

    # Save the statistical data of this message in JSON files
    JSON_LOG(addr, lines_of_string[0].split(' ')[0], '403 Forbidden', request_time)
    JSON_REQUEST_STAT('POST')
    JSON_RESPONSE_STAT('403')

    return ('-1', '-1 ', msg, '-1')
```

4. JSON_LOG

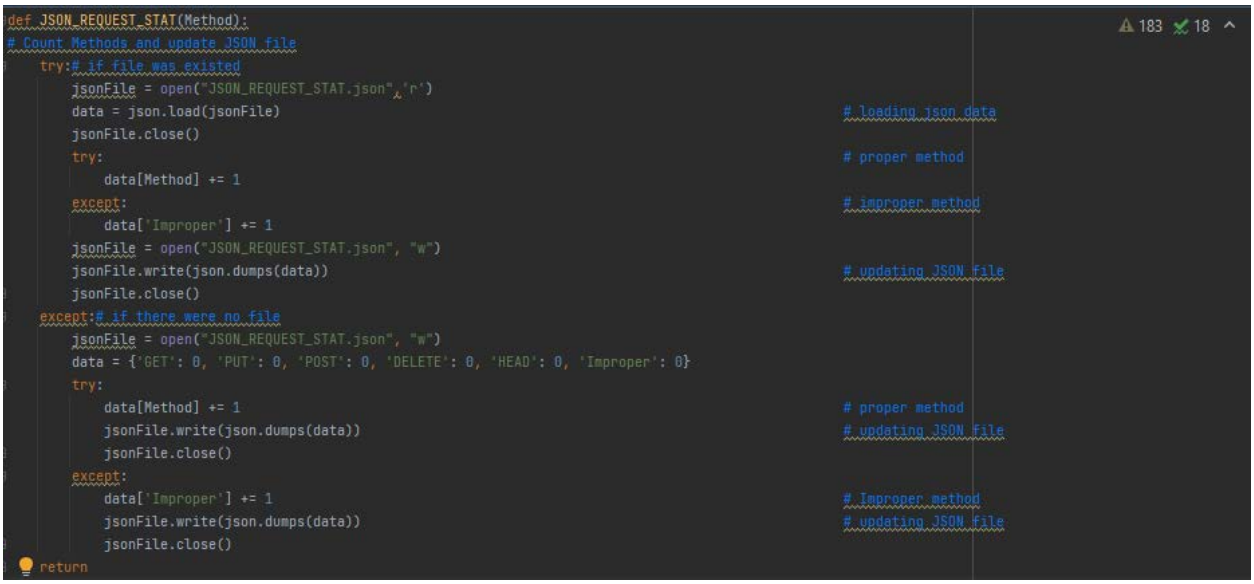
```
def JSON_LOG(addr, Method, status_message, date_time):
# Create and Update log json file.
    try:# if file was existed
        if Method != '' or Method != None:
            jsonFile = open("JSON_LOG.json", "r")
            data = json.load(jsonFile)                                              # loading json data
            jsonFile.close()

            connection_i = str(int(list(data.keys())[-1].split(' ')[-1]) + 1)       # i^th connection of the server
            data.update({'connection ' + connection_i: [str(addr), Method, status_message, date_time]}) #updating JSON file

            jsonFile = open("JSON_LOG.json", "w")
            jsonFile.write(json.dumps(data))
            jsonFile.close()
    except:# if there were no file
            jsonFile = open("JSON_LOG.json", "w")
            data = {'connection 1': [str(addr), Method, status_message, date_time]}
            jsonFile.write(json.dumps(data))
            jsonFile.close()
    return 0
```

در این تابع اطلاعات مربوط به هر پیام دریافتی ثبت و فایل جیسون مربوطه به روز می‌شود. از try, except استفاده شده است زیرا اگر یک فایل جیسون خالی خوانده شود با خطا رو به رو می‌شویم. بقیه توابع مربوط به فایل های جیسون که در ادامه می‌آوریم مشابه هستند.

5. JSON_FILE_TYPE_STAT

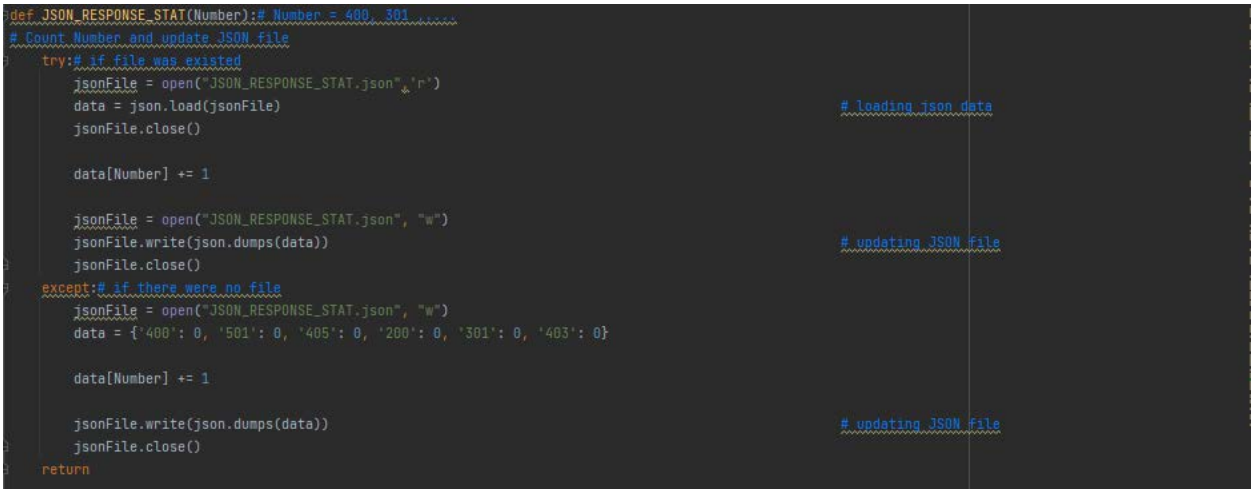
```
def JSON_FILE_TYPE_STAT(type_of_file):
# Count file types
    try:# if file was existed
        jsonFile = open("JSON_FILE_TYPE_STAT.json",'r')
        data = json.load(jsonFile)                                                  # loading json data
        jsonFile.close()

        data[type_of_file] += 1

        jsonFile = open("JSON_FILE_TYPE_STAT.json", "w")                            # i^th connection of the server
        jsonFile.write(json.dumps(data))                                            #updating JSON file
        jsonFile.close()
    except:# if there were no file
        jsonFile = open("JSON_FILE_TYPE_STAT.json", "w")
        data = {'image/jpg': 0, 'text/txt': 0, 'image/png': 0, 'text/html': 0}
        data[type_of_file] += 1
        jsonFile.write(json.dumps(data))
        jsonFile.close()
```

6. JSON_REQUEST_STAT



```
def JSON_REQUEST_STAT(Method):
# Count Methods and update JSON file
    try:# if file was existed
        jsonFile = open("JSON_REQUEST_STAT.json",'r')
        data = json.load(jsonFile)                                  # loading json data
        jsonFile.close()
        try:                                                        # proper method
            data[Method] += 1
        except:                                                     # improper method
            data['Improper'] += 1
        jsonFile = open("JSON_REQUEST_STAT.json", "w")
        jsonFile.write(json.dumps(data))                            # updating JSON file
        jsonFile.close()
    except:# if there were no file
        jsonFile = open("JSON_REQUEST_STAT.json", "w")
        data = {'GET': 0, 'PUT': 0, 'POST': 0, 'DELETE': 0, 'HEAD': 0, 'Improper': 0}
        try:
            data[Method] += 1                                       # proper method
            jsonFile.write(json.dumps(data))                        # updating JSON file
            jsonFile.close()
        except:
            data['Improper'] += 1                                   # Improper method
            jsonFile.write(json.dumps(data))                        # updating JSON file
            jsonFile.close()
    return
```

7. JSON_RESPONSE_STAT



```
def JSON_RESPONSE_STAT(Number):# Number = 400, 301
# Count Number and update JSON file
    try:# if file was existed
        jsonFile = open("JSON_RESPONSE_STAT.json",'r')
        data = json.load(jsonFile)                                  # loading json data
        jsonFile.close()

        data[Number] += 1

        jsonFile = open("JSON_RESPONSE_STAT.json", "w")
        jsonFile.write(json.dumps(data))                            # updating JSON file
        jsonFile.close()
    except:# if there were no file
        jsonFile = open("JSON_RESPONSE_STAT.json", "w")
        data = {'400': 0, '501': 0, '405': 0, '200': 0, '301': 0, '403': 0}

        data[Number] += 1

        jsonFile.write(json.dumps(data))                            # updating JSON file
        jsonFile.close()
    return
```

8. JSON_TO_STRING

```
def JSON_TO_STRING(address):
# Convert Dictionary to string
    response = ''
    jsonFile = open(address,'r')
    data = json.load(jsonFile)
    jsonFile.close()
    for i in list(data.keys()):
        response += i + ' : ' + str(data[i]) + '\n'
    return response
```

تابع بالا آدرس یه فایل جیسون را گرفته و محتویات آن را به رشته تبدیل کرده و برمی‌گرداند. این تابع در بخش telnet کاربرد بسیار خواهد داشت.

در این تابع نوع دستور دریافت شده (اعم از http و stats و disconnect) بررسی می‌شود و نوع آن بصورت رشته برگردانده می‌شود.

9. stat_msg

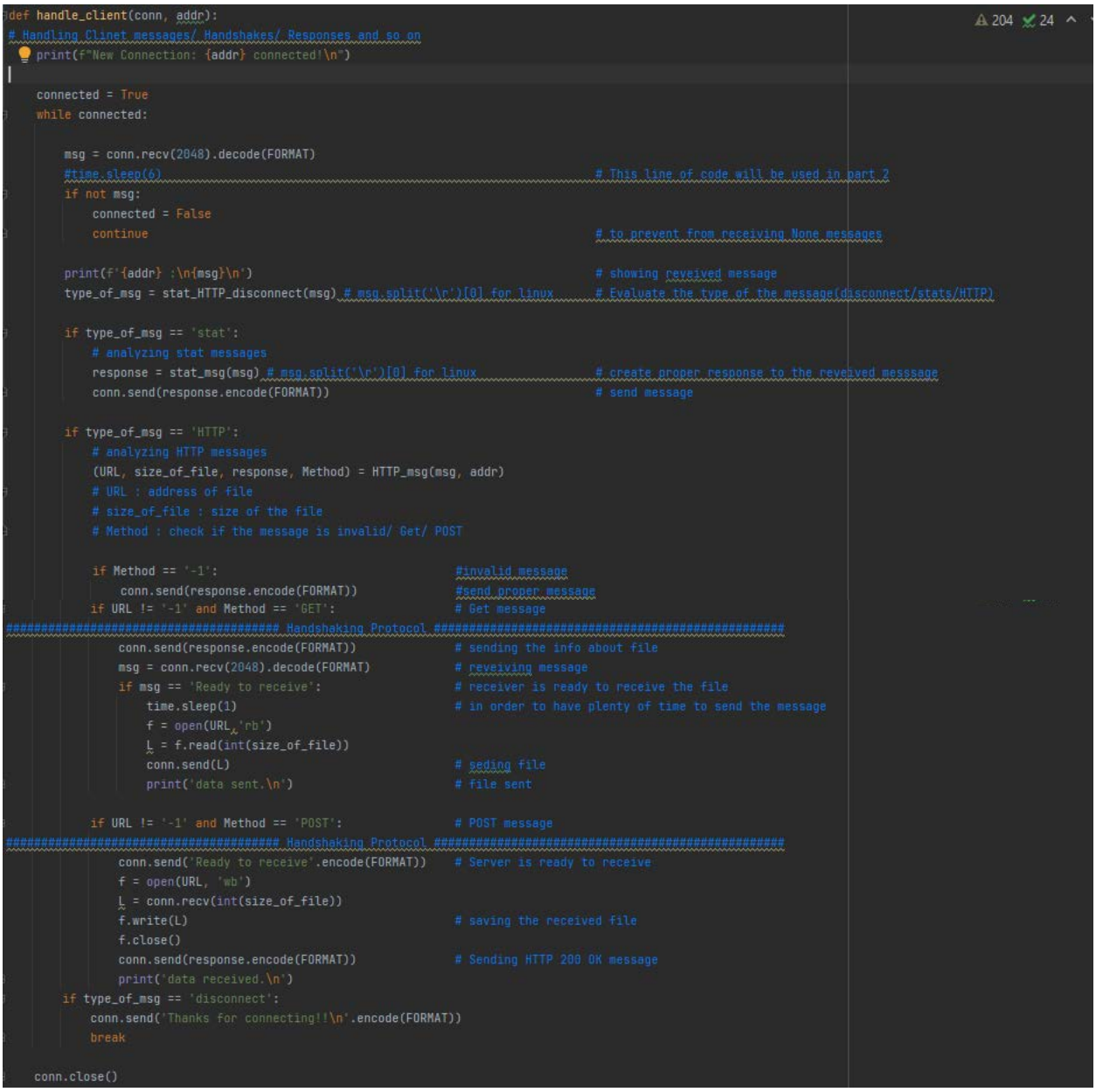
```
def stat_msg(a_string):
# This function executes statistical messages and send json's data files to the client

    if a_string == 'number of connected clients':
        return 'number of connected clients : ' + str(int(threading.activeCount()-1)) + '\n'

    if a_string == 'file type stats':
        try:
            response = JSON_TO_STRING("JSON_FILE_TYPE_STAT.json")
            return response
        except:  #if there was no file existed
            response = 'image/jpg : 0\n' +\
                       'text/txt : 0\n' +\
                      'image/png : 0\n' +\
                      'text/html : 0'
            return response

    if a_string == 'request stats':
        try:
            response = JSON_TO_STRING("JSON_REQUEST_STAT.json")
            return response
        except:  #if there was no file existed
            response = 'GET : 0\n'+\
                       'PUT : 0\n'+\
                       'POST : 0\n'+\
                       'DELETE : 0\n'+\
                       'HEAD : 0\n'+\
                       'Improper : 0'
            return response
    if a_string == 'response stats':
        try:
            response = JSON_TO_STRING("JSON_RESPONSE_STAT.json")
            return response
        except:  #if there was no file existed
            response = '400 : 0\n'+\
                       '501 : 0\n'+\
                       '405 : 0\n'+\
                       '200 : 0\n'+\
                       '301 : 0\n'+\
                       '403 : 0'
            return response
```

در این تابع، دستورات آماری (که در تلنت مورد استفاده قرار می‌گیرند) بررسی می‌شوند. و در این تابع به ازای هر دستور، تابع JSON_TO_STRING فراخوانی می‌شود و اگر فایلی وجود داشته باشد آنرا بصورت رشته برمی‌گرداند. در غیر این صورت یک رشته که تمامی مقادیر مربوطه آن صفر است برگردانده خواهد شد. البته توجه کنید که برای پیاده سازی چنین تابعی از try, except استفاده می‌شود تا از خواندن یک فایل جیسون خالی جلوگیری شود.

در قطعه کد زیر پورت و آدرس IP سرور مشخص شده و بصورت تاپل به سرور متصل خواهد شد.

```
PORT = 5050
SERVER = socket.gethostbyname(socket.gethostname())# Fetching IP address
print(SERVER)

ADDR = (SERVER, PORT)                           # IP - PORT tuple
FORMAT = 'utf-8'


server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(ADDR)                               # binding tuple of address to server
```

10. handle_client



```
def handle_client(conn, addr):
# Handling Clinet messages/ Handshakes/ Responses and so on
    print(f"New Connection: {addr} connected!\n")

    connected = True
    while connected:

        msg = conn.recv(2048).decode(FORMAT)
        #time.sleep(6)                                                  # This line of code will be used in part 2
        if not msg:
            connected = False
            continue                                                   # to prevent from receiving None messages

        print(f'{addr} :\n{msg}\n')                                    # showing reveived message
        type_of_msg = stat_HTTP_disconnect(msg) # msg.split('\r')[0] for linux      # Evaluate the type of the message(disconnect/stats/HTTP)

        if type_of_msg == 'stat':
            # analyzing stat messages
            response = stat_msg(msg) # msg.split('\r')[0] for linux        # create proper response to the reveived messsage
            conn.send(response.encode(FORMAT))                             # send message

        if type_of_msg == 'HTTP':
            # analyzing HTTP messages
            (URL, size_of_file, response, Method) = HTTP_msg(msg, addr)
            # URL : address of file
            # size_of_file : size of the file
            # Method : check if the message is invalid/ Get/ POST

            if Method == '-1':                                 #invalid message
                conn.send(response.encode(FORMAT))             #send proper message
            if URL != '-1' and Method == 'GET':                # Get message
########################################## Handshaking Protocol ##################################################
                conn.send(response.encode(FORMAT))             # sending the info about file
                msg = conn.recv(2048).decode(FORMAT)           # reveiving message
                if msg == 'Ready to receive':                  # receiver is ready to receive the file
                    time.sleep(1)                              # in order to have plenty of time to send the message
                    f = open(URL,'rb')
                    L = f.read(int(size_of_file))
                    conn.send(L)                               # seding file
                    print('data sent.\n')                      # file sent

            if URL != '-1' and Method == 'POST':               # POST message
######################################## Handshaking Protocol ##################################################
                conn.send('Ready to receive'.encode(FORMAT))   # Server is ready to receive
                f = open(URL, 'wb')
                L = conn.recv(int(size_of_file))
                f.write(L)                                     # saving the received file
                f.close()
                conn.send(response.encode(FORMAT))             # Sending HTTP 200 OK message
                print('data received.\n')
        if type_of_msg == 'disconnect':
            conn.send('Thanks for connecting!!\n'.encode(FORMAT))
            break

    conn.close()
```

این تابع بدنه ی اصلی کد است و تمامی ارسال و دریافت ها از طریق این تابع انجام می‌شود. کد در یک حلقه بینهایت صورت می‌گیرید بطوری که اگر پیامی نباشد دوباره حلقه تکرار می‌شود تا زمانی که پیام disconnect وارد شود. در اولین خط درون حلقه پیامی از سمت کلاینت فرستاده می‌شود. سپس نوع این پیام با تابع stat_HTTP_disconnect تعیین می‌شود و به فراخور هر دستور، یک قطعه کد جداگانه اجرا خواهد شد. حالت stat , disconnect پیچیدگی خاصی ندارند و پس از دریافت پیام، با توجه به توابع معرفی شده در صفحات قبلی، پاسخ مناسب داده خواهد شد. اگر پیام از نوع HTTP باشد آن را به تابع HTTP_msg پاس می‌دهیم و پس از آن پیام ممکن است به سه دسته ی مختلف تقسیم شود. دسته ی اول پیام هایی هستند که به نوعی خطا دارند و برای آنها، سرور پیام خطای مناسب را ارسال خواهد کرد. در نوع دوم پیام هایی را خواهیم دید که از نوع GET هستند و فایل مورد

نظر در دیتابیس سرور موجود است. در این حالت بافر کلاینت بایستی بنابر اندازه ی فایل موردنظر تعیین شود پس نمی‌توان همراه با ارسال پیام 200 ok محتوای فایل را نیز ارسال کرد. برای رفع این مشکل از یک پرتکل hand shake استفاده می‌کنیم به این صورت که سرور پس از دریافت پیام کلاینت، یک پیام 200 ok که حاوی طول و نوع فایل است برای او ارسال می‌کند. کلاینت با استخراج نوع و اندازه ی فایل، اندازه ی بافر خود را تنظیم می‌کند و پیام Ready to receive را ارسال می‌کند. پس از آن سرور پیام را بصورت بایت بایت برای کلاینت ارسال خواهد کرد و به این شکل نشست به پایان می‌رسد. نوع دیگری از پیام های HTTP، پیام های post هستند که در آنها کلاینت می‌خواهد فایلی را در سرور بارگذاری کند. در این حالت مشابه حالت قبلی از یک پروتکل دستداد برای تنظیم بافر سرور استفاده خواهد شد. به این صورت که سرور پس از دریافت پیام کلاینت، اندازه و طول فایل را از آن استخراج کرده و بافر خود را تنظیم می‌کند و پیام Ready to receive را برای کلاینت ارسال می‌کند و پس از دریافت فایل، پیام HTTP ok 200 را برای کلاینت ارسال خواهد کرد و اینگونه نشست تمام می‌شود.

11. start

```
def start():
    server.listen()                                                 # listening until receive a message
    print(f'Server is Listening on {SERVER}')
    while True:
        conn, addr = server.accept()                                # accept the request
        thread = threading.Thread(target=handle_client, args=(conn, addr))  # multi threading
        thread.start()
        print(f'Active Connections: {threading.activeCount()-1}\n')            # print number of active connections
```

در این تابع به مولتی تردینگ توجه می‌شود و همچنین سرور تا زمانی که پیامی دریافت نکند به شبکه گوش می‌دهد.

توابع پیاده سازی شده در Client

1. msg_checker

```
def msg_checker(a_string):
    if a_string == 'Ready to receive':                  # Used for handshakes
        return ('-1', -1, 'POST')

    b = a_string.split('\n')                            # divide string to lines
    first_line = b[0]
    state = first_line.split(' ')[2]                    # state of message
    if state == 'OK':
        type_of_file = b[-2].split(': ')[1].split('/')[1]
        #size_of_file = int(b[-3].split(': ')[1])
        return (type_of_file , int(b[-3].split(': ')[1]), 'GET')
    return('-1', -1, '-1')
```

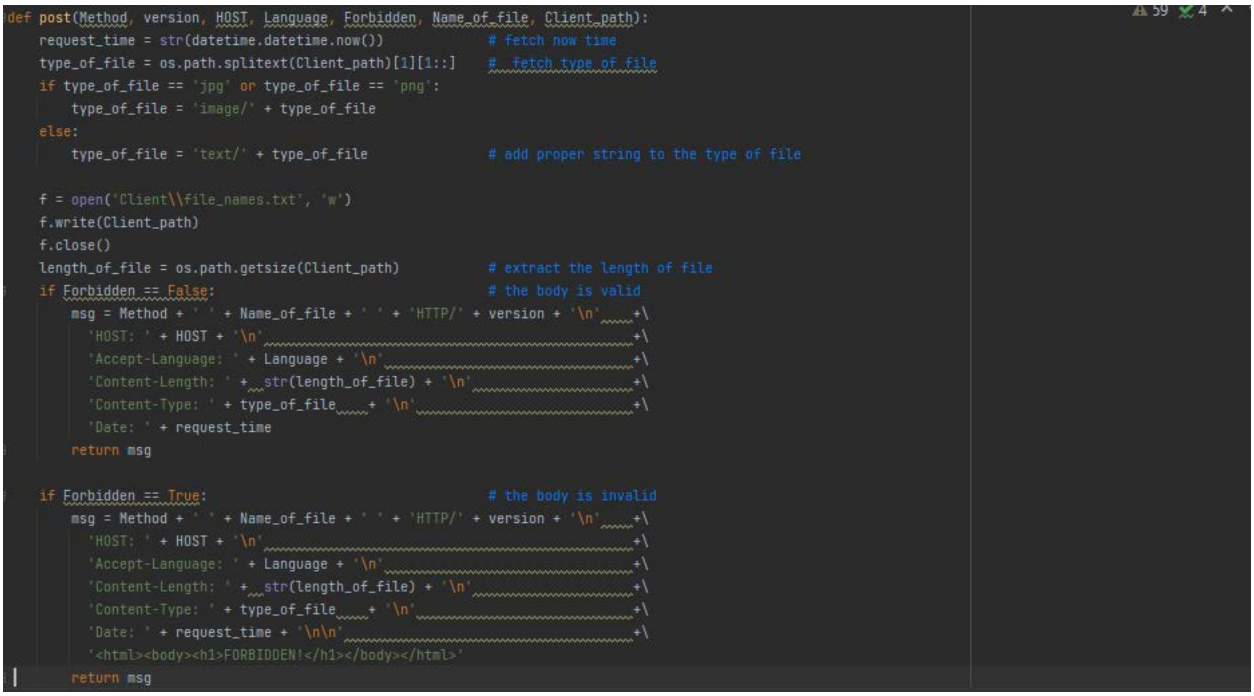
در این تابع پیام های دریافتی از طرف سرور مورد بررسی قرار خواهد گرفت و به فراخور پیام دریافتی اقدامات لازم انجام می‌گیرد.

2. get

```
def get(Method, version, HOST, Language, path):
# sending get messages
    msg = Method + ' ' + path + ' HTTP/' + version + '\n' + 'HOST: ' + HOST + '\n' + 'Accept-Language: ' + Language

    f = open('Client\\file_names.txt', 'w') # open a file to save the name and URL
    Client_path = 'Client\\' + path.split('\\')[-1]
    f.write(Client_path)
    f.close()
```

در این تابع دستور get ساخته شده و ارسال می‌شود. همچنین یک فایل تکست نیز ساخته می‌شود تا نام فایل های دریافتی و ارسالی ذخیره شده و در مواقع مورد نیاز مورد استفاده قرار گیرد.

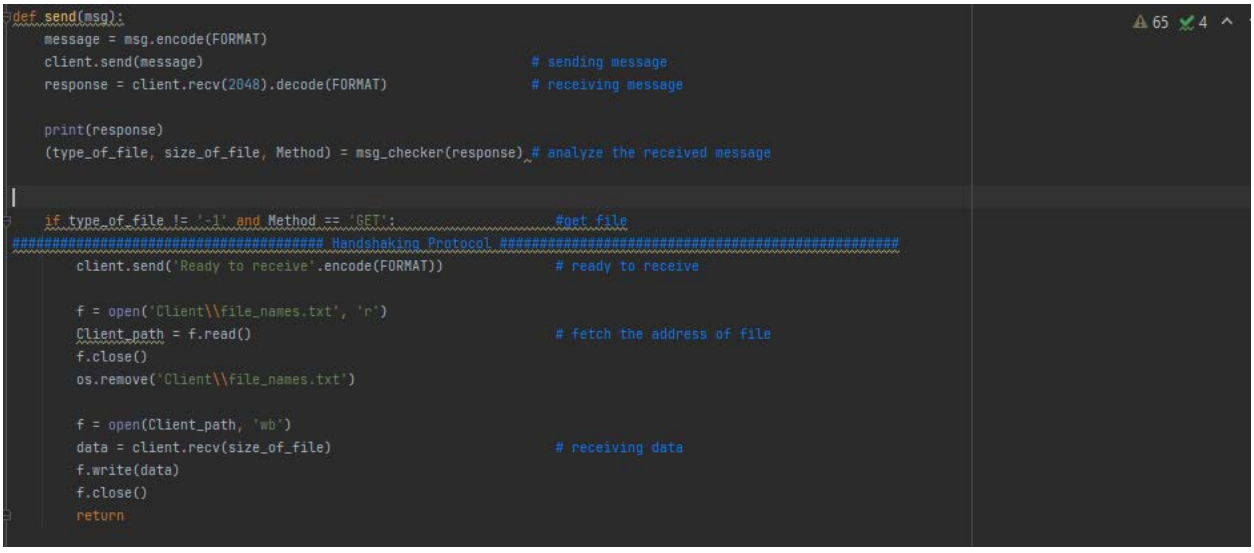
3. post



```
def post(Method, version, HOST, Language, Forbidden, Name_of_file, Client_path):
    request_time = str(datetime.datetime.now())                 # fetch now time
    type_of_file = os.path.splitext(Client_path)[1][1::]        #  fetch type of file
    if type_of_file == 'jpg' or type_of_file == 'png':
        type_of_file = 'image/' + type_of_file
    else:
        type_of_file = 'text/' + type_of_file                   # add proper string to the type of file

    f = open('Client\\file_names.txt', 'w')
    f.write(Client_path)
    f.close()
    length_of_file = os.path.getsize(Client_path)               # extract the length of file
    if Forbidden == False:                                      # the body is valid
        msg = Method + ' ' + Name_of_file + ' ' + 'HTTP/' + version + '\n'+\
          'HOST: ' + HOST + '\n'+\
          'Accept-Language: ' + Language + '\n'+\
          'Content-Length: ' + str(length_of_file) + '\n'+\
          'Content-Type: ' + type_of_file + '\n'+\
          'Date: ' + request_time
        return msg

    if Forbidden == True:                                       # the body is invalid
        msg = Method + ' ' + Name_of_file + ' ' + 'HTTP/' + version + '\n'+\
          'HOST: ' + HOST + '\n'+\
          'Accept-Language: ' + Language + '\n'+\
          'Content-Length: ' + str(length_of_file) + '\n'+\
          'Content-Type: ' + type_of_file + '\n'+\
          'Date: ' + request_time + '\n\n'+\
          '<html><body><h1>FORBIDDEN!</h1></body></html>'
        return msg
```

در این تابع پیام های post ساخته شده و ارسال می‌شوند. همچنین در این تابع با خواندن فایل مربوطه طول و نوع آن استخراج شده و در جایگاه مناسب ارسال می‌شود.

4. send



```
def send(msg):
    message = msg.encode(FORMAT)
    client.send(message)                                    # sending message
    response = client.recv(2048).decode(FORMAT)             # receiving message

    print(response)
    (type_of_file, size_of_file, Method) = msg_checker(response) # analyze the received message


    if type_of_file != '-1' and Method == 'GET':                #get file
######################################## Handshaking Protocol ##################################################
        client.send('Ready to receive'.encode(FORMAT))          # ready to receive

        f = open('Client\\file_names.txt', 'r')
        Client_path = f.read()                                  # fetch the address of file
        f.close()
        os.remove('Client\\file_names.txt')

        f = open(Client_path, 'wb')
        data = client.recv(size_of_file)                        # receiving data
        f.write(data)
        f.close()
        return
```

```
    if type_of_file == '-1' and Method == 'POST':                # post file
##################################### Handshaking Protocol ##################################################
        length_of_file = msg.split('\n')[-3].split(': ')[1]      # fetch length of file

        f = open('Client\\file_names.txt', 'r')
        Client_path = f.read()
        f.close()
        os.remove('Client\\file_names.txt')

        f = open(Client_path, 'rb')
        data = f.read(int(length_of_file))
        client.send(data)                                        # sending data
        f.close()
        response = client.recv(2048).decode(FORMAT)              # receive 200 ok message
        print(response)
```

این تابع بدنه ی اصلی کد کلاینت را تشکیل می‌دهد و تمامی توضیحات آن مشابه تابع handle_client در سرور است که از توضیحات دوباره ی آنها صرف نظر می‌کنیم.

بخش اول: پیاده سازی HTTP Server

در این بخش هر فرمت پیام را از طریق کلاینت به سرور ارسال کرده و پیام دریافتی را بررسی می‌کنیم.

بررسی خطای 400 :

در پیام زیر بین بخش Method و path دو space وجود داشته و نسخه ی HTTP 1.6 می‌باشد که اشتباه است.

پیام ارسالی کلاینت:

```
send(get('GET', '1.6', 'developer.mozilla.org', 'fr','Server\\myhtml.html'))
```

پیام دریافتی در سرور:

```
('192.168.56.1', 59996) :
GET Server\myhtml.html HTTP/1.6
HOST: developer.mozilla.org
Accept-Language: fr
```

پیام دریافتی در کلاینت:

```
HTTP/1.0 400 Bad Request
Connetion: close
Content-Length: 166
Content-Type: text/html
Date: 2021-07-16 15:13:31.205255

<html><body><h1>BADREQUEST!</h1></body></html>
```

بررسی خطای 501 :

این خطا بررسی می‌کند تا دستوری غیر از GET, POST, PUT, DELETE, HEAD مورد بررسی قرار نگیرد و در صورت عدم تطابق یک پیام خطا برای کلاینت ارسال می‌کند.

پیام ارسالی کلاینت:

```
send(get('GETT', '1.1', 'developer.mozilla.org', 'fr','Server\\myhtml.html'))
```

پیام دریافتی در سرور:

```
('192.168.56.1', 61065) :
GETT Server\myhtml.html HTTP/1.1
HOST: developer.mozilla.org
Accept-Language: fr
```

پیام دریافتی در کلاینت:

```
HTTP/1.0 501 Not Implemented
Connetion: close
Content-Length: 174
Content-Type: text/html
Date: 2021-07-16 15:23:04.219268

<html><body><h1>NOTIMPLEMENTED!</h1></body></html>
```

بررسی خطای 405 :

این پیام بررسی می‌کند تا از میان دستورات GET, POST, PUT, DELETE, HEAD تنها دستورات GET, POST مورد بررسی قرار گیرند.

پیام ارسالی کلاینت:

```
send(get('DELETE', '1.1', 'developer.mozilla.org', 'fr', 'Server\\myhtml.html'))
```

پیام دریافتی در سرور:

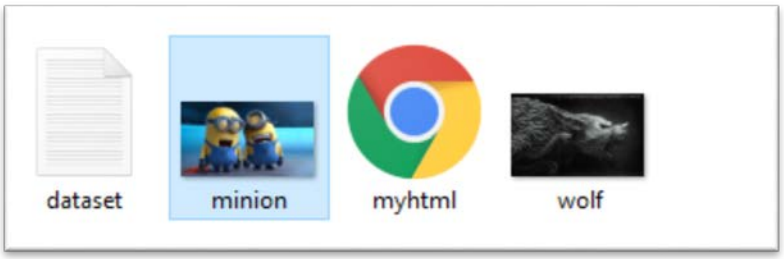
```
('192.168.56.1', 61602) :
DELETE Server\myhtml.html HTTP/1.1
HOST: developer.mozilla.org
Accept-Language: fr
```

پیام دریافتی در کلاینت:

```
HTTP/1.0 405 Method Not Allowed
Connetion: close
Content-Length: 184
Content-Type: text/html
Allow: GET
Date: 2021-07-16 15:31:52.862955

<html><body><h1>NOTALLOWED!</h1></body></html>
```

بررسی دستور GET زمانی که فایل مورد نظر در سرور وجود دارد :

فرض کنید محتویات پوشه ی سرور و کلاینت به ترتیب بصورت زیر باشد:



و قصد داریم تا فایل مشخص شده در سرور را برای کلاینت با استفاده از دستور زیر ارسال کنیم:

```
send(get('GET', '1.1', 'developer.mozilla.org', 'fr', 'Server\\minion.jpg'))
```

پیام دریافتی در سرور:

```
('192.168.56.1', 62879) :
GET Server\minion.jpg HTTP/1.1
HOST: developer.mozilla.org
Accept-Language: fr
```

پیام دریافتی در کلاینت:

```
HTTP/1.0 200 OK
Connetion: close
Content-Length: 52808
Content-Type: image/jpg
Date: 2021-07-16 15:39:47.052162
```

حال پس از دریافت این پیام از طرف سرور پوشه ی کلاینت را بررسی می‌کنیم که بصورت زیر است:

همانطور که مشاهده می‌شود فایل مطلوب به درستی ارسال شده است.

بررسی دستور GET زمانی که فایل مورد نظر در سرور وجود ندارد :

پیام ارسالی کلاینت:

```
send(get('GET', '1.1', 'developer.mozilla.org', 'fr', 'Server\\Not_in_server.png'))
```

پیام دریافتی در سرور:

```
('192.168.56.1', 49455) :
GET Server\Not_in_server.png HTTP/1.1
HOST: developer.mozilla.org
Accept-Language: fr
```

پیام دریافتی در کلاینت:

```
HTTP/1.0 301 Moved Permanently
Connetion: close
Content-Length: 178
Content-Type: text/html
Date: 2021-07-16 16:07:09.024021

<html><body><h1>MOVEDPERMANENTLY!</h1></body></html>
```

بررسی دستور POST با بدنه ی معتبر :

به وسیله ی این دستور کلاینت می‌تواند فایلی را برای سرور ارسال کرده و سرور آن را در پوشه ی خود ذخیره کند. حالتی که بدنه‌ی دستور معتبر نباشد در بخش بعدی مورد بررسی قرار خواهد گرفت. فرض کنید کلاینت بخواهد تصویر زیر را در سرور ذخیره کند:

پیام ارسالی کلاینت:

```
send(post('POST', '1.1', 'developer.mozilla.org', 'fr', False, 'jungle', 'Client\\jungle.png'))
```

پیام دریافتی در سرور:

```
POST jungle HTTP/1.1
HOST: developer.mozilla.org
Accept-Language: fr
Content-Length: 13578
Content-Type: image/png
Date: 2021-07-16 16:18:11.476357
```

پیام دریافتی در کلاینت:

```
HTTP/1.0 200 OK
Connetion: close
Content-Length: 147
Content-Type: text/html
Date: 2021-07-16 16:18:11.477358

<html><body><h1>POST!</h1></body></html>
```

حال پس از دریافت این پیام از طرف سرور پوشه ی سرور را بررسی می‌کنیم که بصورت زیر است:

همانطور که مشاهده می‌شود فایل مطلوب به درستی ارسال شده است.

بررسی دستور POST **با بدنه ی غیر معتبر (** FORBIDDEN **) :**

پیام ارسالی کلاینت:

```
send(post('POST', '1.1', 'developer.mozilla.org', 'fr', True, 'jungle', 'Client\\jungle.png'))
```

پیام دریافتی در سرور:

```
('192.168.56.1', 52203) :
POST jungle HTTP/1.1
HOST: developer.mozilla.org
Accept-Language: fr
Content-Length: 13578
Content-Type: image/png
Date: 2021-07-16 16:32:16.419291

<html><body><h1>FORBIDDEN!</h1></body></html>
```

پیام دریافتی در کلاینت:

```
HTTP/1.0 403 Forbidden
Connetion: close
Content-Length: 163
Content-Type: text/html
Date: 2021-07-16 16:32:16.420284

<html><body><h1>FORBIDDEN!</h1></body></html>
```

بخش دوم: پیاده سازی HTTP Client

در این بخش فرض می‌کنیم 4 کلاینت می‌خواهند به یک سرور وصل شوند و پیام هایی را ارسال کنند. از آنجایی که نمی‌توانیم همزمان همه ی کلاینت ها را به سرور متصل کنیم(به اندازه ی کامپیوتر سریع نیستیم☺) در کد سرور برای هر تقاضا یک وقفه ی 6 ثانیه ای ایجاد می‌کنیم.

```
def handle_client(conn, addr):
    print(f"New Connection: {addr} connected!\n")

    connected = True
    while connected:

        msg = conn.recv(2048).decode(FORMAT)
        time.sleep(6)
        if not msg:
            connected = False
            continue

        print(f'{addr} :\n{msg}\n')
        type_of_msg = stat_HTTP_disconnect(msg)
```

حال فرض کنید کلاینت ها بخواهند از دستورات زیر استفاده کنند:

کلاینت1 :

```
send(post('POST', '1.1', 'developer.mozilla.org', 'fr', False, 'jungle', 'Client\\jungle.png'))
send(get('HEAD', '1.0', 'developer.mozilla.org', 'fr','Server\\postfile.txt'))
send(get('METHOD ', '1.6', 'developer.mozilla.org', 'fr','Server\\postfile.txt'))
```

کلاینت 2 :

```
send(get('GET', '1.0', 'developer.mozilla.org', 'fr','Server\\minion.jpg'))
send(post('POST', '1.1', 'developer.mozilla.org', 'fr', False, 'Ezio', 'Client2\\Ezio.jpg'))
```

کلاینت 3 :

```
send(get('GET', '1.0', 'developer.mozilla.org', 'fr','Server\\dataset.txt'))
send(get('GET', '1.0', 'developer.mozilla.org', 'fr','Server\\myhtml.html'))
```

کلاینت 4 :

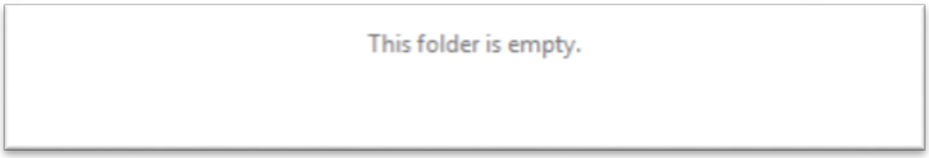
```
send(get('POST', '1.1', 'developer.mozilla.org', 'fr', 'Server\\wolf.jpg'))
send(get('HEAD', '1.6', 'developer.mozilla.org', 'fr','Server\\myhtml.html'))
send(post('POST', '1.1', 'developer.mozilla.org', 'fr', True, 'Ezio', 'Client2\\Ezio.jpg'))
```

پوشه ی کلاینت ها و سرور بصورت زیر است:

کلاینت 1 :

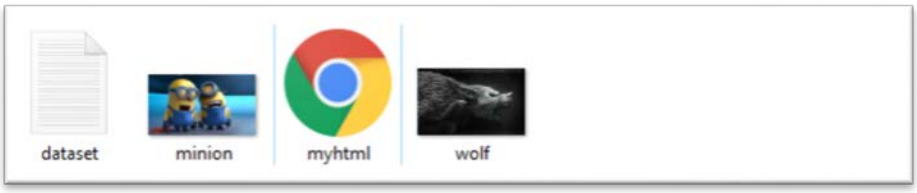
کلاینت 2 :

کلاینت 3 :



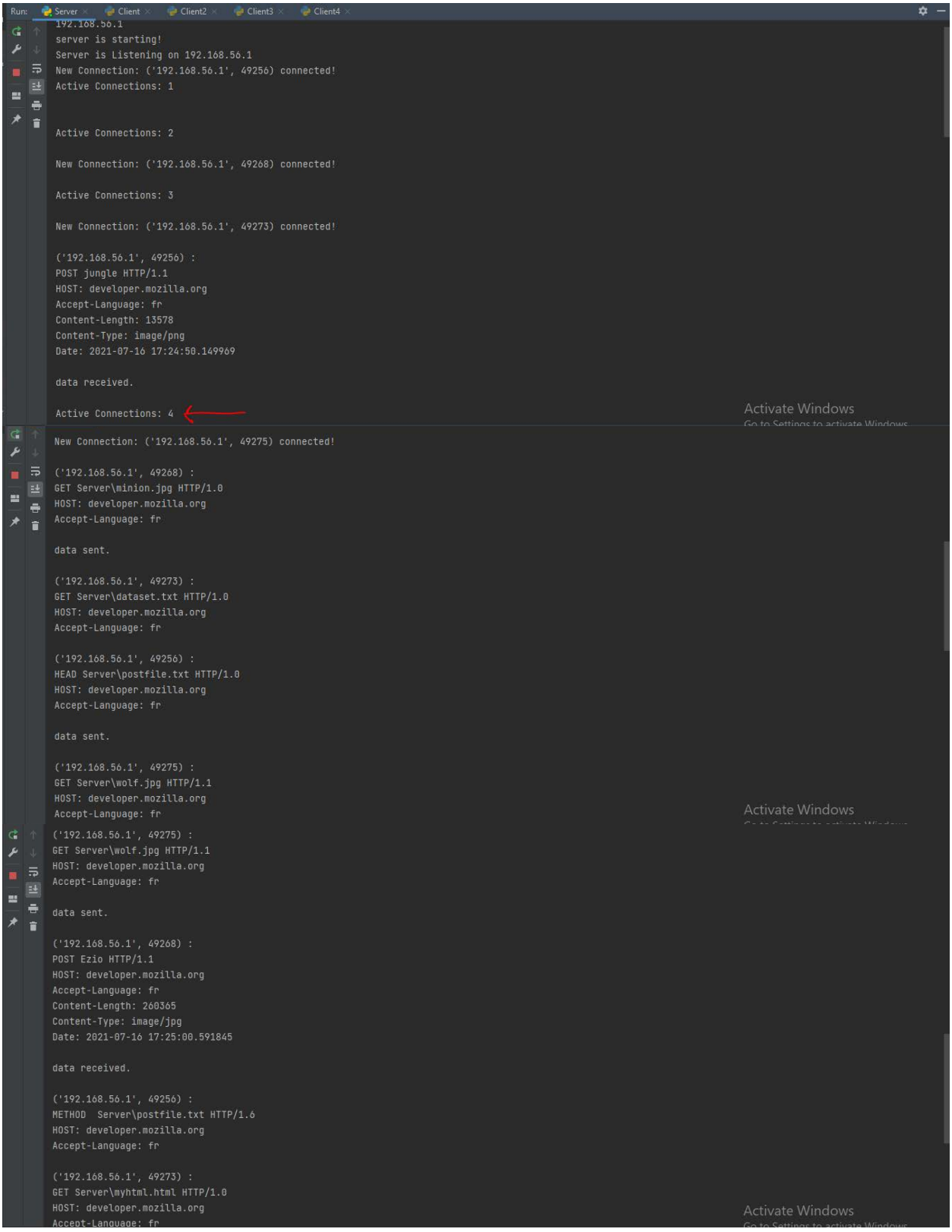
کلاینت 4 :

سرور :



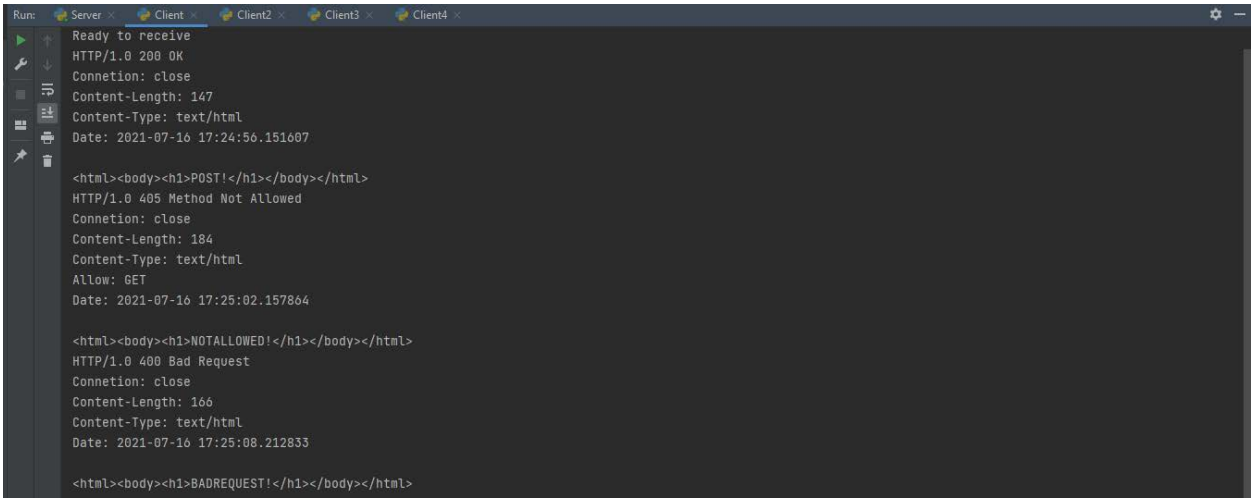
پس از اجرای برنامه، کنسول های هر کد پایتون بصورت زیر بدست آمد:

سرور :

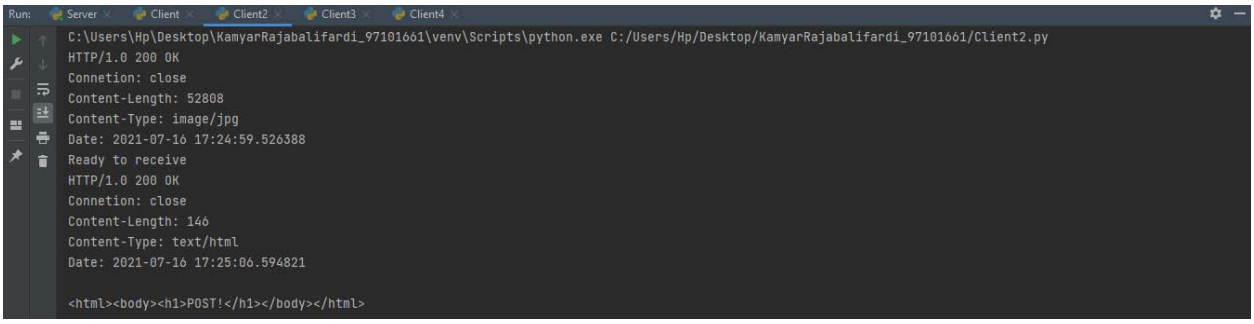

Run: Server Client Client2 Client3 Client4
192.168.56.1
server is starting!
Server is Listening on 192.168.56.1
New Connection: ('192.168.56.1', 49256) connected!
Active Connections: 1
Active Connections: 2
New Connection: ('192.168.56.1', 49268) connected!
Active Connections: 3
New Connection: ('192.168.56.1', 49273) connected!
('192.168.56.1', 49256) :
POST jungle HTTP/1.1
HOST: developer.mozilla.org
Accept-Language: fr
Content-Length: 13578
Content-Type: image/png
Date: 2021-07-16 17:24:50.149969
data received.
Active Connections: 4
New Connection: ('192.168.56.1', 49275) connected!
('192.168.56.1', 49268) :
GET Server\minion.jpg HTTP/1.0
HOST: developer.mozilla.org
Accept-Language: fr
data sent.
('192.168.56.1', 49273) :
GET Server\dataset.txt HTTP/1.0
HOST: developer.mozilla.org
Accept-Language: fr
('192.168.56.1', 49256) :
HEAD Server\postfile.txt HTTP/1.0
HOST: developer.mozilla.org
Accept-Language: fr
data sent.
('192.168.56.1', 49275) :
GET Server\wolf.jpg HTTP/1.1
HOST: developer.mozilla.org
Accept-Language: fr
('192.168.56.1', 49275) :
GET Server\wolf.jpg HTTP/1.1
HOST: developer.mozilla.org
Accept-Language: fr
data sent.
('192.168.56.1', 49268) :
POST Ezio HTTP/1.1
HOST: developer.mozilla.org
Accept-Language: fr
Content-Length: 260365
Content-Type: image/jpg
Date: 2021-07-16 17:25:00.591845
data received.
('192.168.56.1', 49256) :
METHOD Server\postfile.txt HTTP/1.0
HOST: developer.mozilla.org
Accept-Language: fr
('192.168.56.1', 49273) :
GET Server\myhtml.html HTTP/1.0
HOST: developer.mozilla.org
Accept-Language: fr
Activate Windows
Activate Windows
Activate Windows

```
('192.168.56.1', 49275) :
HEAD Server\myhtml.html HTTP/1.6
HOST: developer.mozilla.org
Accept-Language: fr

('192.168.56.1', 49275) :
POST Ezio HTTP/1.1
HOST: developer.mozilla.org
Accept-Language: fr
Content-Length: 260365
Content-Type: image/jpg
Date: 2021-07-16 17:25:10.715212

<html><body><h1>FORBIDDEN!</h1></body></html>
```

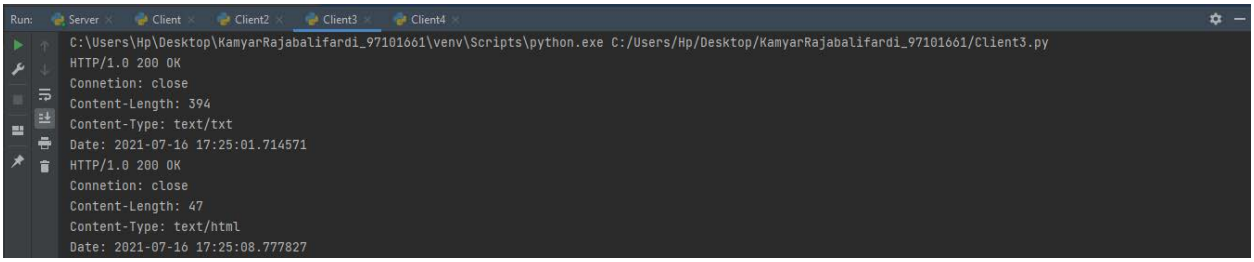
کلاینت 1:



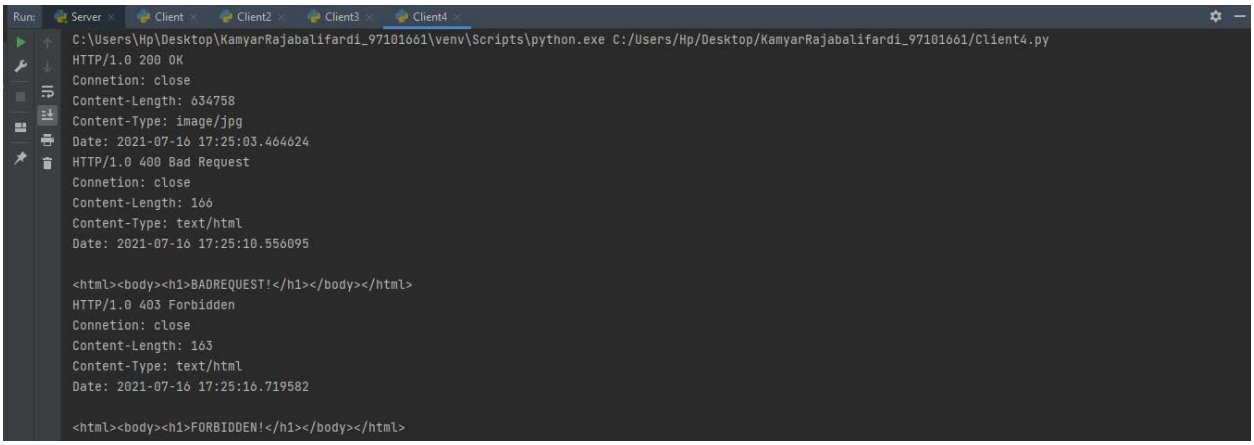
کلاینت 2 :



کلاینت 3 :



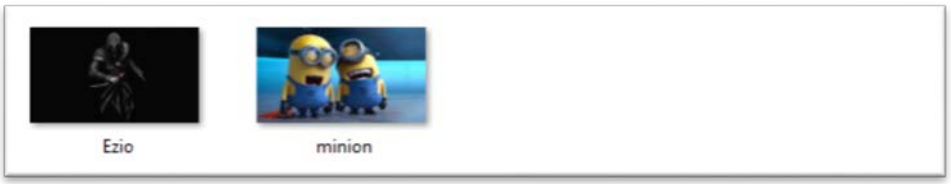
کلاینت 4 :



پوشه ی کلاینت ها و سرور بصورت زیر است:

کلاینت 1 :

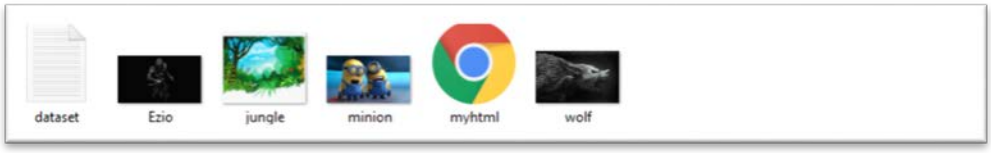
کلاینت 2 :



کلاینت 3 :

کلاینت 4 :

سرور :



بخش سوم : Logging

در این قسمت از تابع JSON_LOG که خودمان پیاده سازی کردیم برای ثبت اطلاعات هر اتصال استفاده می‌کنیم. ساختار فایل حیسون ایجاد شده به این صورت است که هر کلید(key) نشان دهنده ی چندمین اتصال، و مقدار(values) متناظر با آن کلید نشان دهنده ی اطلاعات مربوط به این اتصال است که در قالب یک لیست نوشته می‌شود که می‌توان آنرا بصورت زیر درنظر گرفت:

$$\{i^{th}\ connection\colon\ [address\ of\ client\ ,request\ type\ ,response\ type\ ,the\ start\ of\ the\ connection]\}$$

در عکس زیر، نمونه ای از این فایل را مشاهده می‌کنید.


```
{"connection 1": ["('192.168.56.1', 59996)", "GET", "400 Bad Request", "2021-07-16 15:13:31.205255"],
 "connection 2": ["('192.168.56.1', 61023)", "HEAD", "405 Method Not Allowed", "2021-07-16 15:22:40.521284"],
 "connection 3": ["('192.168.56.1', 61065)", "GETT", "501 Not Implemented", "2021-07-16 15:23:04.219268"],
 "connection 4": ["('192.168.56.1', 61602)", "DELETE", "501 Not Implemented", "2021-07-16 15:27:59.624987"],
 "connection 5": ["('192.168.56.1', 61777)", "GETT", "501 Not Implemented", "2021-07-16 15:29:34.684389"],
 "connection 6": ["('192.168.56.1', 61810)", "PUT", "501 Not Implemented", "2021-07-16 15:29:57.132884"],
 "connection 7": ["('192.168.56.1', 61896)", "PUT", "501 Not Implemented", "2021-07-16 15:30:41.248760"],
 "connection 8": ["('192.168.56.1', 62027)", "DELETE", "405 Method Not Allowed", "2021-07-16 15:31:52.862955"],
 "connection 9": ["('192.168.56.1', 62879)", "GET", "200 OK", "2021-07-16 15:39:47.052162"],
 "connection 10": ["('192.168.56.1', 49455)", "GET", "301 Moved Permanently", "2021-07-16 16:07:09.024021"],
 "connection 11": ["('192.168.56.1', 50137)", "GET", "200 OK", "2021-07-16 16:13:22.669128"],
 "connection 12": ["('192.168.56.1', 52147)", "POST", "403 Forbidden", "2021-07-16 16:31:46.165648"],
 "connection 13": ["('192.168.56.1', 52203)", "POST", "403 Forbidden", "2021-07-16 16:32:16.420284"],
 "connection 14": ["('192.168.56.1', 63320)", "GET", "200 OK", "2021-07-16 17:11:53.343207"],
 "connection 15": ["('192.168.56.1', 63321)", "GET", "200 OK", "2021-07-16 17:11:55.210652"],
 "connection 16": ["('192.168.56.1', 63319)", "HEAD", "405 Method Not Allowed", "2021-07-16 17:11:57.414109"],
 "connection 17": ["('192.168.56.1', 63321)", "GET", "200 OK", "2021-07-16 17:12:02.267094"],
 "connection 18": ["('192.168.56.1', 63319)", "METHOD", "400 Bad Request", "2021-07-16 17:12:03.435954"],
 "connection 19": ["('192.168.56.1', 49268)", "GET", "200 OK", "2021-07-16 17:24:59.526388"],
 "connection 20": ["('192.168.56.1', 49273)", "GET", "200 OK", "2021-07-16 17:25:01.714571"],
 "connection 21": ["('192.168.56.1', 49256)", "HEAD", "405 Method Not Allowed", "2021-07-16 17:25:02.157864"],
 "connection 22": ["('192.168.56.1', 49275)", "GET", "200 OK", "2021-07-16 17:25:03.464624"],
 "connection 23": ["('192.168.56.1', 49256)", "METHOD", "400 Bad Request", "2021-07-16 17:25:08.212833"],
 "connection 24": ["('192.168.56.1', 49273)", "GET", "200 OK", "2021-07-16 17:25:08.777827"],
 "connection 25": ["('192.168.56.1', 49275)", "HEAD", "400 Bad Request", "2021-07-16 17:25:10.556095"],
```

بخش چهارم: تلنت

در این بخش ابتدا فایل Server.py و تمامی فایل های جیسون متناظر با آن را به لینوکس منتقل کرده و در ترمینال لینوکس برنامه را اجرا می‌کنیم. اما توجه کنید که در لینوکس هنگام ورودی دادن از طریق ترمینال کاراکتر اضافه ی "\r" نیز با کاراکترهای وارد شده توسط کاربر به سرور داده می‌شود که موجب اختلال در کار برنامه خواهد شد. برای رفع این مشکل قبل از پاس کردن رشته ی ورودی به توابع موجود در کد سرور، از دستور split برای جدا کردن این کاراکتر از دیگر کاراکترها بصورت زیر استفاده خواهیم کرد:

```
def handle_client(conn, addr):
    print(f"New Connection: {addr} connected!\n")

    connected = True
    while connected:

        msg = conn.recv(2048).decode(FORMAT)
        #time.sleep(6)
        if not msg:
            connected = False
            continue

        print(f'{addr} :\n{msg}\n')
        type_of_msg = stat_HTTP_disconnect(msg.split('\r')[0])

        if type_of_msg == 'stat':
            response = stat_msg(msg.split('\r')[0])
            conn.send(response.encode(FORMAT))
```

حال در ترمینال تمامی 5 دستور ذکر شده در صورت پروژه را اجرا کرده و با داده های موجود در ویندوز مقایسه می‌کنیم.

```
kamyar@kamyar-VirtualBox:~$ telnet 127.0.1.1 5050
Trying 127.0.1.1...
Connected to 127.0.1.1.
Escape character is '^]'.
number of connected clients
number of connected clients : 1
request stats
GET : 11
PUT : 0
POST : 10
DELETE : 1
HEAD : 4
Improper : 12
response stats
400 : 9
501 : 5
405 : 4
200 : 16
301 : 1
403 : 3
file type stats
image/jpg : 5
text/txt : 2
image/png : 0
text/html : 2
disconnect
Thanks for connecting!!
Connection closed by foreign host.
```

فایل های جیسون در ویندوز بصورت زیر هستند:

request stats:

```
{"GET": 11, "PUT": 0, "POST": 10, "DELETE": 1, "HEAD": 4, "Improper": 7}
```

response stats:

```
{"400": 4, "501": 5, "405": 4, "200": 16, "301": 1, "403": 3}
```

file type stats:

```
{"image/jpg": 5, "text/txt": 2, "image/png": 0, "text/html": 2}
```

همانطور که مشاهده می‌شود خروجی برنامه از لینوکس با ویندوز مطابقت دارد.

بخش پنجم: پیاده سازی Web Server

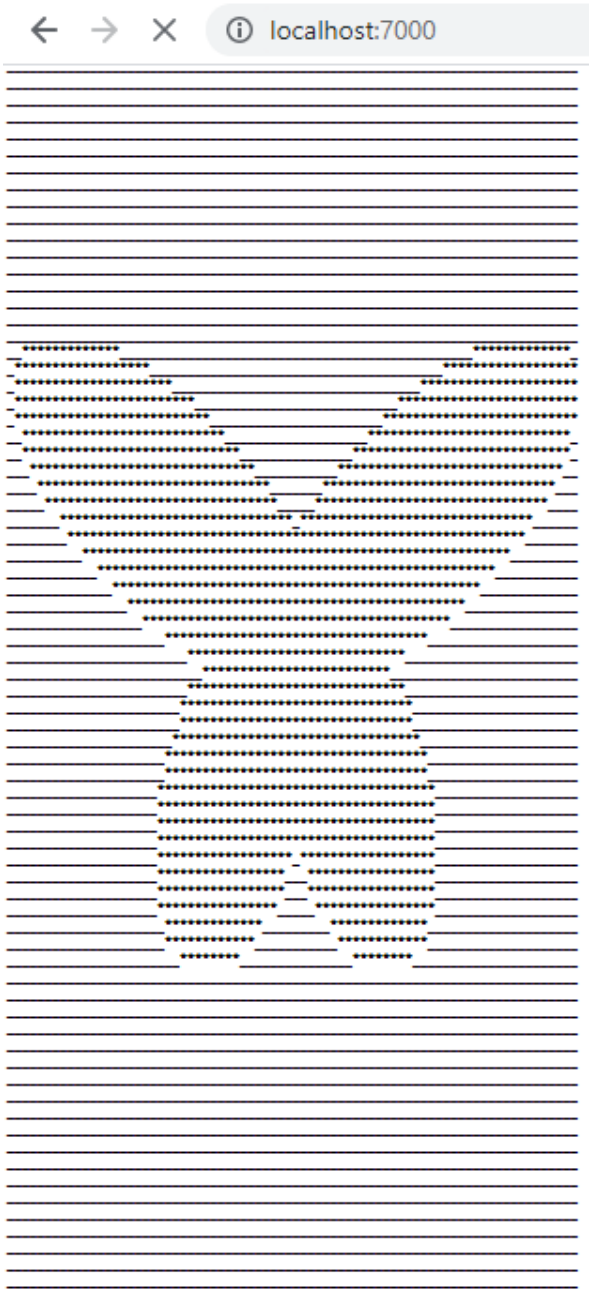
از آنجاییکه دستورات این قسمت همگی از نوع GET هستند بنابراین یک سرور دیگر به نام ServerWEB.py ایجاد می‌کنیم که تنها این دسته از دستورات را بررسی می‌کند.

```
def handle_client(conn, addr):
    print(f"New Connection: {addr} connected!")

    connected = True
    while connected:

        msg = conn.recv(2048).decode(FORMAT)
        if not msg:
            connected = False
            continue
        print(msg)
        butterfly = butter_fly(17)
        response = 'HTTP/1.1 200 OK' + '\n' +\
                   'Content-Type: ' + 'text/html; charset=utf-8' + '\n\n' +\
                   '<html><body><h1>'+ butterfly +'</body></html>'
        conn.send(response.encode(FORMAT))

    conn.close()
```

همچنین متن نوشته شده به زبان html را در تابع butterfly نوشتیم که یک پروانه با ابعادی که ورودی این تابع تعیین می‌کند رسم می‌کند.

```python
def butter_fly(n):
    l = math.ceil(2.2*n)
    butterfly = ''
    for j in range(l-l-1):
        for i in range(-l,l,1):
            if math.pow(abs(i)+2*j-2*n, 2) + 5*math.pow(abs(i)-2*j, 2) <= 5*math.pow(n, 2) or math.pow(abs(i)-j-n, 2)
                butterfly += '*'
            else:
                butterfly += '_'
        butterfly += '<br/>'
    return butterfly
```

پس از وارد کردن ادرس http://localhost:7000 را در مرورگر وارد کردیم با متن زیر مواجه شدیم



همچنین پیام دریافتی از سمت مرورگر که در کنسول چاپ کردیم بصورت زیر است:

```
GET / HTTP/1.1
Host: localhost:7000
Connection: keep-alive
Cache-Control: max-age=0
sec-ch-ua: " Not;A Brand";v="99", "Google Chrome";v="91", "Chromium";v="91"
sec-ch-ua-mobile: ?0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
```

همانطور که مشاهده می‌شود مرورگر هم از پروتکل HTTP برای درخواست های خود استفاده می‌کند.