Project 1 :

Web Server / Web Client

Software Engineering
2016024902 윤세령

Contents

I. Project Preview

- Developing a Multi-threaded Web server that is capable of processing multiple simultaneous service request in parallel.
- The server code will be developed in two stages:
  1. Write a multi-threaded server that simply displays the contents of the HTTP request message that is receives.
  2. Add the code required to generate an appropriate response.
- Demonstrate that the web server is capable of delivering your home page to a web browser.

II. Code Explanation

1. webServer.py

```
# import to use Server
import BaseHTTPServer
from SocketServer import ThreadingMixIn
host = '0.0.0.0'

# Response handler
class MyHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    # action for request 'get'
    def do_GET(self):
        print self.path
        # First connect(/) or connect to '/index.html'
        if (self.path.endswith('/index.html')):
            self.protocol_version = 'HTTP/1.1'
            # Connect to 'index.html' in directory
            f = open('./index.html')
            # Add log
            print('test1')
            # Response(200) means you got request well
            self.send_response(200)
            # Add header
            self.send_header('Content-Type', 'text/html')
            print('test2')
            #self.send_header('content-length', 500000)
            self.end_headers()
            # Setting text that set on the server
            self.wfile.write(f.read())
            f. close()
            return

        # connect to '/image.jpg'
        elif self.path.endswith('/image.jpg'):
            self.protocol_version = 'HTTP/1.1'
            f = open('./image.jpg')
            self.send_response(200)
            print('testImage')
            self.send_header('Content-Type', 'image/jpg')
            #self.send_header('Content-length', 500000)
            self.end_headers()
            self.wfile.write(f.read())
            f.close()
            return


        # HTTP/1.0
        elif self.request_version == 'HTTP/1.0':
            # 400 error
            self.send_error(400, 'BAD REQUEST')
            print('test400')
            self.wfile.write(f.read())


        # Connect another path or exception
        else:
            # 404 error
            self.send_error(404, 'NOT FOUND')
            print('test404')
            self.wfile.write(f.read())

```

- do_GET(self):

  When the first connect or connect to '/text.html', connect to text.html in directory. The content-type is 'text/html' and the content-length is 1024 to send a message. Response message sets 200 so response(200) that means you got request well.

  When connect to '/image.png', send in condition that content-type is 'image/png' and length is 1024. As the first case, response message is 200 and length is 1024.

  When the request-version is 'HTTP/1.0', make 400 error(Bad request) and send it.

  About exception or connect to other paths, send 404 error(Not found).

```
60      # action for request 'put'
61      def do_PUT(self):
62          print('===== PUT =====')
63          print(self.headers)
64          length = int(self.headers['content-length'])
65          content = self.rfile.read(length)
66          # Response(200) means you got request well
67          self.send_response(200)
68          print(content)
69
70      # action for request 'post'
71      def do_POST(self):
72          print('\n===== Request Start =====\n')
73          request_path = self.path
74          print(request_path)
75
76          request_headers = self.headers
77          content_length = request_headers.getheaders('content_length')
78          length = int(content_length[0]) if content_length else 0
79
80          print(request_headers)
81          print(self, rfile.read(length))
82          print('\n===== Request End =====\n')
83
84          # Response(200) means you got request well
85          self.send_response(200)
86
87  # Inherit ThreadHTTPServer to handle requests in a separate thread.
88  class ThreadHTTPServer(ThreadingMixIn, BaseHTTPServer.HTTPServer):
89      pass
90
91  # Main part
92  if __name__ == '__main__':
93      # Declare Server -> host & port with the class server 'MyHandler'
94      server = BaseHTTPServer.HTTPServer(('', 9090), MyHandler)
95      print('Started WebServer on port 9090')
96      print('Press Ctrl + C to quit webserver')
97      # makes work server
98      server.serve_forever()
```

- do_PUT(self) & do_POST(self) :

  These are extra functions, so I didn't do error handling.

- ThreadHTTPServer(ThreadingMixIn, BaseHTTPServer.HTTPServer) :

  Inherit 'ThreadHTTPServer' to make an environment that handle requests in multi-threading.

2. webClient.py

```python
import httplib

host = '127.0.0.1'
port = 9090
conn = httplib.HTTPConnection(host,port)

# request 'index.html'
conn.request("GET", "./index.html")
response = conn.getresponse()

# request 'image.jpg'
conn.request("GET", "./image.jpg")
respose = conn.getresponse()

data1 = response.read()
print data1
```

- request 'text. Html' or 'image.jpg'.

III. Instructions

Put files in directory that the program is running. Approach *http://52.79.241.196:1919/index.html* through my web browser(In my case, I used Chrome). Move to "Computer Network" and click "Go Test". Fill the blanks and see ERROR page or RESULT page.

IV. Program Operating & Results

```
➜  Project1 python webServer.py
Started WebServer on port 9090
Press Ctrl + C to quit webserver
./index.html
test1
127.0.0.1 - - [23/Oct/2019 18:07:30] "GET ./index.html HTTP/1.1" 200 -
test2
./image.jpg
127.0.0.1 - - [23/Oct/2019 18:07:30] "GET ./image.jpg HTTP/1.1" 200 -
testImage
```

```
→  Project1 python webClient.py
<head>
        <title>Dummy HTML</title>
</head>
<body>
        <h1>Computer Network test</h1>
        <h1>Well Done</h1>
        <mata http-equiv="refresh" content="0;URL=./image/cat.jpeg">
</body>
→  Project1
```

Your Test Result

Student INFO

Student Number	Student Name	WebServer IP	WebServer PORT	ACCESS TIME	SCORE
2016024902	SERYOUNG YOON	166.104.141.14	9090	2019-49-23 05:49:19	90

List of Test Items

WEB SERVER SOCKET :	TRUE	
Multi Thread :	TRUE	
STATUS CODE : 200 OK	TRUE	
STATUS CODE : 404 NOT FOUND(EXCEPTION HANDLING)	TRUE	
STATUS CODE : 400 BAD REQUEST(HTTP PROTOCOL VERSION)	False	When Client request as HTTP/1.0 version, You should send 400 Bad Request message to Client
CONTENT LENGTH	TRUE	
CONTENT TYPE TEXT/HTML	TRUE	
CONTENT TYPE IMAGE/JPEG	TRUE	

V. My Opinion

Programming the socket is very interesting. I learned how the server and clients interact through this project. I used 8080 for port number that actually in use, so I've suffered from that happening. The problem was solved by changing port number to 9090.

One regret is that there is not enough time to solve last one problem. I wrote the exception 400 but when I checked my code by log, it didn't work properly. Next time I'll try to implement it to work well.