|  |
| --- |
| HTTP Test (with Python) |

20203104 소프트웨어학부 유동현

□ 목적

- TCP 기반의 소켓(Socket) 통신을 활용하여 HTTP의 동작원리를 이해하고, WireShark(패킷 분석 툴)를 활용하여 직접 구현한 HTTP 패킷을 캡쳐 가능하도록 구현하는 데 있다.

□ 목표

1. TCP 기반의 Server, Client 간의 소켓(Socket) 통신 구현

2. Client에서 HTTP Protocol의 GET/PUT/HEAD/POST 형식에 맞는 Request를 Server에 보내고, Response를 받는다

3. Server에서 HTTP Protocol의 Client로부터의 Request에 대한 Response를 보낸다

4. WireShark를 활용하여 Resposne와 Request를 캡쳐한다.

5. 처음에는 Local 환경에서 진행, 이후 2대 이상의 PC를 활용하여 Client와 Server를 분리하여 실행한다.

□ 실행 방법

Server

|  |
| --- |
| $ git clone <https://github.com/DongHyeonYu/ComputerNetwork_test_HTTP.git>  $ cd /ComputerNetwork\_test\_HTTP.git  $ python3 HTTP\_Server.py |

Client

|  |
| --- |
| $ git clone https://github.com/DongHyeonYu/ComputerNetwork\_test\_HTTP.git  $ cd /ComputerNetwork\_test\_HTTP/HTTP/dist/  HTTP\_Client.py 코드 상단의 serverName(IP Address)/serverPort 조정  (Local환경에서 실행 시 조정 불필요)  [방법1]  $ pip install PyQt5  $ pip install pyinstaller  $ cd ../  $ pyinstaller --onefile HTTP\_Test.py  /dist 내부의 HTTP\_Test.exe 실행    [방법2]  $ python3 HTTP\_Client.py |

□ 유의사항

- 반드시 Server Code(HTTP\_Server.py)를 먼저 실행 시킨 후, Client 코드를 실행 시켜야 테스트가 가능하다.

□ 개발환경

Client

|  |
| --- |
| Windows10 / Python 3.12 |

* MacOS에서는 PyQt5 라이브러리 사용불가능으로 [방법2]로 진행

Server

|  |
| --- |
| MacOS / Python 3.12 |

□ Test Case 설계

# CASE 1

|  |
| --- |
| Request :  Method : GET  Path : /  Response : 200 OK |

# CASE 2

|  |
| --- |
| Request :  Method : GET  Path : /NotFoundError  Response : 404 Not Found |

# CASE 3\_4

|  |
| --- |
| Request :  Method : POST  Path : /  Response : 100 Continue & 200 OK |

# CASE 5

|  |
| --- |
| Request :  Method : POST  Path : /  Response : 100 Continue & 400 Bad Request |

# CASE 6

|  |
| --- |
| Request :  Method : POST  Path : /NotFoundError  Response : 404 Not Found |

# CASE 7

|  |
| --- |
| Request :  Method : HEAD  Path : /  Response : 200 OK |

# CASE 8

|  |
| --- |
| Request :  Method : HEAD  Path : /NotFoundError  Response : 404 Not Found |

# CASE 9

|  |
| --- |
| Request :  Method : PUT  Path : /test.jpeg  Response : 100 Continue & 200 OK |

# CASE 10

|  |
| --- |
| Request :  Method : PUT  Path : /400\_Bad\_Request.png  Response : 400 Bad Reqeust |

□ 구현 목표

|  |  |  |
| --- | --- | --- |
| **내용** | **적용 여부** | **비고** |
| TCP기반의 소켓 프로그래밍 작성 | O | HTTP\_Client.py  HTTP\_Server.py |
| GET/POST/HEAD/PUT Request구현 | O | HTTP\_Client.py |
| HTTP Request에 대한 Response구현 | O | HTTP\_Server.py |

□ 추가 구현 사항

|  |  |  |
| --- | --- | --- |
| **내용** | **적용 여부** | **비고** |
| PyQt 활용 간단한 GUI구현 | O | HTTP\_Test.py  HTTP\_Test.exe |
| 간단한 웹페이지 활용 Response, Request 확인 | X |  |

* 기존의 Console에서 Response, Request Message 확인하는데 있어 많은 Test Case로 인하여 가독성이 떨어지는 문제 발생
* 초기 웹페이지를 활용한 Response, Request 확인을 목표로 설정하였으나, Python코드를 HTML코드 내에서 실행(pyscript) 하는데 있어 성능상의 문제(속도, 무한로딩) 발생

□ TCP 프로그래밍

Server

|  |
| --- |
| from socket import \*  serverPort = 8080  serverSocket = socket(AF\_INET, SOCK\_STREAM)  serverSocket.bind((‘’, serverPort))  serverSocket.listen(1)  …  While True:  connectionSocket, addr = serverSocket.accept()  try:  message = connectionSocket.recv(1024).decode()  …  response = response\_line + headers + response\_body  connectionSocket.send(response.encode())  …  …  connectionSocket.close() |
|  |

* 기존 Python socket 라이브러리 활용
* Port 8080으로 Socket을 생성한 뒤, listen으로 Client의 Message 기다림
* while 무한 루프를 통해 사용자의 Message를 수신한 뒤, response를 생성하여 connectionSocket.send( )를 활용하여 Response 전송

Client

|  |
| --- |
| from socket import \*  serverName = ‘192.168.0.22’  serverPort = 8080  def CASE1():  clientSocekt = socket(AF\_INET, SOCK\_STREAM)  clientSocekt.connect((serverName, serverPort))    …  response = clientSocket.recv(4096).decode()  print(“From Server :”)  print(response)  clientSocekt.close()  return request, response |

* 기존 Python socket 라이브러리 활용
* serverName은 server code가 실행 되고 있는 PC의 내부 IP Address
* 각 CASE를 실행할 때마다 socket( ) 메소드로 소켓을 생성한 뒤, socket.connect( )로 server의 소켓과 연결하고 socket.send( )로 Request 메시지를 보냄
* 메시지를 보내고 socket.recv( ) 메소드로 서버의 Response를 기다리고, Response가 도착하면 화면에 출력

□ HTTP Protocol

Server

|  |
| --- |
| …  lines, request\_line, method, path = request\_parser(message)    if method == “GET”:  GET(path)  elif method == “HEAD”:  HEAD(path)  elif method == “POST”:  POST(connectionSocket, message, path)  elif method == “PUT”:  PUT(connectionSocekt, message, path)  else:  another\_request() |

* TCP 소켓 프로그래밍 코드에서 받은 Request 메시지를 request\_parser( ) 함수로 파싱 한 뒤, 각 method에 맞는 함수로 분기

|  |
| --- |
| def request\_parser(message):  try:  lines = message.split(“\r\n”)  request\_line = lines[0].split()  method = request\_line[0]  path = request\_line[1]  …  return lines, request\_line, method, path |

|  |
| --- |
| def GET(path):  if path == “/”:  response\_line = “HTTP/1.1 200 OK\r\n”  header = “Content-Type: text/html\r\n\r\n”  body = “<html><script>alert(“Hello!”);</script></html>  response = response\_line + header + body  connectionSocket.send(response.encode())  else:  response\_line = “HTTP/1.1 404 Not Found\r\n”  header = “Content-Type: text/html\r\n\r\n”  body = “<html><body><h1>404 Not Found</h1></body></html>  response = response\_line + header + body  connectionSocekt.send (response.encode())  return  def POST(connectionSocket, message, path):  if “Expect: 100-Continue” in message and path == “/”:  response\_line = “HTTP/1.1 100 Continue\r\n\r\n”  connectionSocket.send(response\_line.endcode())    body=connectionSocket.recv(1024).decode().lstrip().rstrip() .upper()  if 0<len(body)<=30:  final\_response\_line = “HTTP/1.1 200 OK\r\n”  header = “Content-Type: text/html\r\n\r\n”  response\_body= f“<html><body><h1>{body}</h1></body></html>”  response = final\_response\_line + header + response\_body  connectionSocket.send(response.encode())  else:  final\_response\_line = “HTTP/1.1 400 Bad Request\r\n”  header = “Content-Type: text/html\r\n\r\n”  response\_body = f“<html><body><h1>400 Bad Request</h1></body></html>”  response = final\_response\_line + header + response\_body  connectionSocket.send(response.encode())  else:  response\_line = “HTTP/1.1 404 Not Found\r\n”  header = “Content-Type: text/html\r\n\r\n”  response\_body = “<html><body><h1>404 Not Found</h1></body></html>”  response = response\_line + header + response\_body  connectionSocket.send(response.encode())  def HEAD(path):  …(GET 동일)  def PUT(connectionSocket, message, path):  …(POST 동일) |

Client

|  |
| --- |
| def CASE1():  …  request\_line = f“GET / HTTP/1.1\r\n”  headers = (f“Host: {serverName}:{serverPort}\r\n”  f“User-Agent: Custom/1.0\r\n”  f“Connection: close\r\n\r\n”)  request = request\_line + headers  clientSocekt.send(request.endcode())  …  def CASE2():  …  request\_line = f“GET /**NotFoundError** HTTP/1.1\r\n”  headers = (f“Host: {serverName}:{serverPort}\r\n”  f“User-Agent: Custom/1.0\r\n”  f“Connection: close\r\n\r\n”)  …  def CASE3\_4():  …  message = “Hello World”  request\_line = f“POST / HTTP/1.1\r\n”  headers = (f“Host: {serverName}:{serverPort}\r\n”  f“Expect: 100-Continue\r\n”  f“User-Agent: Custom/1.0\r\n”  f“Content-Length: {len(message)}\r\n”  f“Content-Type: text/plain\r\n\r\n”)  …  response = clientSocket.recv(1024).decode()    if “100 Continue” in response:  clientSocket.send((message + “\r\n”).endcode())  final\_response = clientSocekt.recv(1024).decode()  …  def CASE5():  …  message = “ ”  request\_line = f“POST / HTTP/1.1\r\n”  headers = (f“Host: {serverName}:{serverPort}\r\n”  f“Expect: 100-Continue\r\n”  f“User-Agent: Custom/1.0\r\n”  f“Content-Length: {len(message)}\r\n”  f“Content-Type: text/plain\r\n\r\n”)  …  response = clientSocket.recv(1024).decode()    if “100 Continue” in response:  clientSocket.send((message + “\r\n”).endcode())  final\_response = clientSocekt.recv(1024).decode()  …  def CASE6():  …  message = “Hello World”  request\_line = f“POST /**NotFoundError** HTTP/1.1\r\n”  headers = (f“Host: {serverName}:{serverPort}\r\n”  f“Expect: 100-Continue\r\n”  f“User-Agent: Custom/1.0\r\n”  f“Content-Length: {len(message)}\r\n”  f“Content-Type: text/plain\r\n\r\n”)  …  response = clientSocket.recv(1024).decode()    if “100 Continue” in response:  clientSocket.send((message + “\r\n”).endcode())  final\_response = clientSocekt.recv(1024).decode()  …  def CASE7():  …  request\_line = f“HEAD / HTTP/1.1\r\n”  headers = (f“Host: {serverName}:{serverPort}\r\n”  f“User-Agent: Custom/1.0\r\n”  f“Connection: close\r\n\r\n”)  …  def CASE8():  …  request\_line = f“HEAD /NotFoundError HTTP/1.1\r\n”  headers = (f“Host: {serverName}:{serverPort}\r\n”  f“User-Agent: Custom/1.0\r\n”  f“Connection: close\r\n\r\n”)  …  def CASE9():  …  filename = “test.jpeg”  content\_type = “image/jpeg”  …  request\_line = (f“PUT /{filename} HTTP/1.1\r\n”  f“Host: {serverName}:{serverPort}\r\n”  f“Expect: 100-Continue\r\n”  f“Content-Type: {content\_type}\r\n\r\n”  f“Content-Length: {len(file\_content)\r\n}”  …  response = clientSocekt.recv(1024).decode()  if “100 Continue in response:  clientSocket.send(file\_content)  final\_response = clientSocket.recv(1024).decode()  …  def CASE10():  …  filename = “400\_Bad\_Request.png”  content\_type = “image/png”  …  request\_line = (f“PUT /{filename} HTTP/1.1\r\n”  f“Host: {serverName}:{serverPort}\r\n”  f“Expect: 100-Continue\r\n”  f“Content-Type: {content\_type}\r\n\r\n”  f“Content-Length: {len(file\_content)\r\n}”  …  response = clientSocekt.recv(1024).decode()  if “100 Continue in response:  clientSocket.send(file\_content)  final\_response = clientSocket.recv(1024).decode()  … |

□ CASE 설명

# CASE1

* 기본적인 GET Method, Route path 로 요청
* Server Code에서 200 OK 응답

# CASE2

* GET Method, 의도적으로 존재하지 않는path(/NotFoundError)로 Request를 보냄,
* Server Code에서 else문을 통해, Route path가 아닌 모든 path에 대하여 404 Not Found Error를 발생하도록 처리

# CASE3\_4

* POST Method, Route path로 요청, Expect: 100-Continue 를 통해 서버에서 100응답을 받은 이후 100 Continue(CASE3)이면 body인 “Hello World” 메시지를 서버에 전송
* 서버에서 200 OK(CASE4) 응답과 함께 body의 모든 문자가 대문자로 변경된 응답을 받음

# CASE5

* POST Method, 100 응답을 받은 이후, body를 전송할 때 의도적으로 빈 문자열을 전송
* Server Code에서 빈 문자열을 전송받을 경우, 400 Bad Request가 발생하도록 처리

# CASE6

* POST Method, “hello world” 메시지를 의도적으로 존재하지 않는 path(/NotFoundError)로 전송
* Server Code에서 Routh path(/)가 아닌 모든 경로에 대해서 404 Not Found Error 가 발생하도록 처리

# CASE7

* HEAD Method, Route path로 요청
* Server Code에서 200 OK응답

# CASE8

* HAED Method, 의도적으로 존재하지 않는 경로(/NotFoundError)로 요청
* Server Code에서 Route path(/)가 아닌 모든 경로에 대하여 404 Not Found Error가 발생하도록 처리

# CASE9

* PUT Method, test.jpeg(Image)전송
* Expect: 100-Continue를 통해 100 응답을 받은 이후 Image File을 open함수를 통해 bytes code로 변환한 후 서버에 전송
* Server Code에서 파일을 저장이후 200 OK 응답

# CASE10

* PUT Method, 400\_Bad\_Request.png(Image)전송
* Server Code에서 받을 수 있는 파일의 최대크기를 8192(byte)로 제한, 이를 초과하는 파일의 경우 400 Bad Request가 발생하도록 처리

□ 실행 환경

Server (MacOS, Terminal)

|  |
| --- |
| 텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진  자동 생성된 설명 |

Client (Windows10, PyQt GUI)

|  |
| --- |
| 텍스트, 스크린샷, 디스플레이, 번호이(가) 표시된 사진  자동 생성된 설명 |

□ WireShark 캡쳐 내용

|  |
| --- |
| 텍스트, 전자제품, 스크린샷, 번호이(가) 표시된 사진  자동 생성된 설명 |

* 캡쳐 필터 ip.addr == 192.168.0.22(Server PC 내부 IP Address)
* http 프로토콜 캡쳐

□ 실행 화면

# CASE1

Server

|  |
| --- |
| 텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진  자동 생성된 설명 |

Client

|  |
| --- |
|  |

# CASE2

Server

|  |
| --- |
| 텍스트, 스크린샷, 폰트이(가) 표시된 사진  자동 생성된 설명 |

Client

|  |
| --- |
|  |

# CASE3\_4

Server

|  |
| --- |
| 텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진  자동 생성된 설명 |

Client

|  |
| --- |
|  |

# CASE5

Server

|  |
| --- |
| 텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진  자동 생성된 설명 |

Client

|  |
| --- |
|  |

# CASE6

Server

|  |
| --- |
| 텍스트, 스크린샷, 폰트이(가) 표시된 사진  자동 생성된 설명 |

Client

|  |
| --- |
|  |

# CASE7

Server

|  |
| --- |
| 텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진  자동 생성된 설명 |

Client

|  |
| --- |
|  |

# CASE8

Server

|  |
| --- |
| 텍스트, 스크린샷, 폰트이(가) 표시된 사진  자동 생성된 설명 |

Client

|  |
| --- |
|  |

# CASE9

Server

|  |
| --- |
| 텍스트, 스크린샷, 폰트이(가) 표시된 사진  자동 생성된 설명 |
| 텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진  자동 생성된 설명 |

Client

|  |
| --- |
|  |

# CASE10

Server

|  |
| --- |
| 텍스트, 스크린샷, 폰트이(가) 표시된 사진  자동 생성된 설명 |

Client

|  |
| --- |
|  |

□ 참고문헌

HTTP RFC7231

<https://datatracker.ietf.org/doc/html/rfc7231>

Python Socket 프로그래밍

<https://mcc96.tistory.com/58>

Python PyQt5 라이브러리 사용

<https://dev-guardy.tistory.com/41>

HTTP Status Code

<https://dev-cho.tistory.com/78>