TCP Socket project

국민대학교 KOOKMIN UNIVERSITY

과목명 | 컴퓨터네트워크

담당교수 | 이창우 교수님

학과 | 소프트웨어학과

학년 | 3학년

학번 | 20181704

이름 | 표상우

제출일 | 2022.05.01

Description

- 소켓 통신을 활용하여 Server, Client 프로그램 작성
- TCP 기반 소켓프로그래밍 작성후 Client에서는 HTTP 프로토콜의 GET/HEAD/POST/PUT Request를 요청하고 Server에서는 Client의 Request에 따라 응답 메시지르 구성하여 Response하도록 구현
- HTTP 명령어 수행 결과 WireShark로 확인- HTTP 명령어 수행 결과 WireShark로
- HTTP 명령어 수행시 Server에서 Client가 요청하는 파일 생성, update, 출력

Environment

- Python 3.8.9
- Apple M1 Mac OS
- port 는 8080 localhost로 진행
- server.py파일을 실행한뒤 client.py 파일을 실행

Code

1. server.py

- 소켓 통신에 필요한 모듈 가져오기

```
from socket import *
import time
```

- HOST, PORT, SIZE 설정

```
HOST = "127.0.0.1"
PORT = 8080
SIZE = 1024
```

- HTTP method를 추출하는 함수

```
def find_method(method):
    method = method.split(' ')
    return method[0]
```

- HTTP url을 추출하는 함수

```
def find_url(url):
    url = url.split(' ')
    return url[1][0:-1]
```

-매개변수로 status, method, body를 받아서 DB를 만들고 관리하는 함수
-status가 CREATED일때 DataBase.txt에 body추가(method == POST)
-status가 OK일때 body가 있으면 DataBase.txt에 key값을 확인하고 value 값 교체(method == PUT)
-status가 OK일때 body가 없으면 DataBase.txt에 있는 key, value값 return(method == GET)
-status가 BAD_REQUEST,NOT_FOUND일때 해당 status return
-status가 CONTINUE일때 return ‘’(method == HEAD)

```
def ManageDB(status, method, body):
    arr = []
    if ':' in body:
        key, value = body.split(':')
    if body != '':
        if status == 'CREATED':
            with open("DataBase.txt", "a") as f:
                f.write(f"{body}\n")
            return 'Created DB'
        elif status == 'OK':
            with open("DataBase.txt", "r") as f:
                lines = f.readlines()
            with open("DataBase.txt", "w") as f:
                for line in lines:
                    if line.rstrip()[0] != key:
                        f.write(line)
                    else:
                        f.write(f"{body}\n")
            return 'Update DataBase'
        elif status == 'BAD_REQUEST':
            return 'BAD_REQUEST'
        elif status == 'NOT_FOUND':
            return 'NOT_FOUND'
        else:
            return 'BAD_REQUEST'
    else:
        if status == 'OK':
            if method == 'GET':
                with open("DataBase.txt", "r") as f:
                    lines = f.readlines()
                    for line in lines:
                        arr.append(line.rstrip())
                    return arr
        elif status == 'CONTINUE':
            return ''
        else:
            return 'BAD_REQUEST'
```

- 매개변수로 status, method, body를 받아서 client에서 보내준 HTTP Request가 맞는지 확인하고 method를 response함수로 보내는 함수

```
def router(url, method, body):
    if '/' in url:
        host, path = url.split('/')
        if host == HOST:
            if method == 'HEAD':
                return response('CONTINUE', method, body)
            if method == 'GET': return response('OK', method, body)
            if method == 'POST':
                if path == 'create':
                    return response('CREATED', method, body)
                else: return response('BAD_REQUEST', method, body)
            elif method == 'PUT':
                if path == 'update': return response('OK', method, body)
                else: return response('BAD_REQUEST', method, body)
            else: return response('NOT_FOUND', method, body)
        else:
            return ''
    else:
        return response('NOT_FOUND', method, body)
```

- 매개변수로 status, method, body를 받아서 해당 status에 맞는 response를 만들고 return해주는 함수

```
def response(status, method, body):
```

```
    date = time.strftime('%a, %d %b %Y %H:%M:%S GMT',
time.localtime(time.time()))
    if status == 'CONTINUE':
        DBbody = ManageDB(status, method, body)
        return f"HTTP/1.1 100 CONTINUE\r\nDate: {date}\r\nContent-Type: text/
html\r\nConnection: keep-alive\r\nContent-Length: {len(body)}\r\n\n{DBbody}"
    if status == 'OK':
        DBbody = ManageDB(status, method, body)
        return f"HTTP/1.1 200 OK\r\nDate: {date}\r\nContent-Type: text/
html\r\nConnection: keep-alive\r\nContent-Length: {len(body)}\r\n\n{DBbody}"
    if status == 'CREATED':
        DBbody = ManageDB(status, method, body)
        return f"HTTP/1.1 201 CREATED\r\nDate: {date}\r\nContent-Type: text/
html\r\nConnection: keep-alive\r\nContent-Length: {len(body)}\r\n\n{DBbody}"
    if status == 'BAD_REQUEST':
        DBbody = ManageDB(status, method, body)
        return f"HTTP/1.1 400 BAD_REQUEST\r\nDate: {date}\r\nContent-Type: text/
html\r\nConnection: keep-alive\r\nContent-Length: {len(body)}\r\n\n{DBbody}"
    if status == 'NOT_FOUND':
        DBbody = ManageDB(status, method, body)
        return f"HTTP/1.1 404 NOT_FOUND\r\nDate: {date}\r\nContent-Type: text/
html\r\nConnection: keep-alive\r\nContent-Length: {len(body)}\r\n\n{DBbody}"

with socket(AF_INET, SOCK_STREAM) as server_socket: # 소켓 객체 생성
    server_socket.bind((HOST, PORT))  # 생성한 소켓에 HOST와 PORT 바인딩
    server_socket.listen(1)  # 서버가 클라이언트의 접속을 허용
```

- client의 socket, address 저장
- recv함수로 client가 보낸 HTTP Reqeust 받기
- Server에서 보낼 response 만들기 위해 router함수에 추출한 url, method, body를 넣고 response를 만들어서 res_message 변수에 저장
- send를 통해 client에게 response 전송

```
    while True:
        connectionsocket, client_addr = server_socket.accept()
# accept 함수에서 대기하다가 클라이언트가 접속하면 새로운 소켓을 리턴
        data = connectionsocket.recv(SIZE).decode('utf-8')
# client에서 보내는 데이터 받기
        print(data)
        print('\n-------------------------------------\n')
        data = data.split('\n')
        method = find_method(data[0])
        url = find_url(data[1])
        body = data[-1]
        res_message = router(url, method, body)
        connectionsocket.send(res_message.encode('utf-8'))  # 데이터 인코딩하여 보내기
```

2. client.py

- 소켓 통신에 필요한 모듈 가져오기

```
from socket import *
```

- IP, PORT, SIZE 설정

```
IP = "127.0.0.1"
PORT = 8080
SIZE = 1024
```

- client에서 보낼 HTTP Request들의 요소들을 test_case라는 딕셔너리 형태로 생성

```
test_case = [
    {'url': '127.0.0.1/',
        'method': 'HEAD',
```

```
        'data': ''
    },
    {'url': '127.0.0.1/create',
        'method': 'POST',
        'data': '1:test1'
    },
    {'url': '127.0.0.1/create',
        'method': 'POST',
        'data': '2:test2'
    },
    {'url': '127.0.0.1/create',
        'method': 'PUT',
        'data': '3:test3'
    },
    {'url': '127.0.0.1/',
        'method': 'GET',
        'data': ''
    },
    {'url': '127.0.0.1/update',
        'method': 'PUT',
        'data': '1:test5'
    },
    {'url': '127.0.0.1/ComputerNetwork',
        'method': 'POST',
        'data': '6:test6'
    },
    {'url': '127.0.0.1/',
        'method': 'GET',
        'data': ''
    }
]
```

- method, data, url을 매개변수로 받아 client에서 보낼 HTTP Request를 생성하는 함수

```
def request_formating(method, data, url):
    return f"{method} / HTTP/1.1\r\nHost: {url}\r\nConnection: keep-
alive\r\nContent-Type: text/html\r\nContent-Length: {len(data)}\r\nUser-Agent:
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit\r\n\n{data}"
```

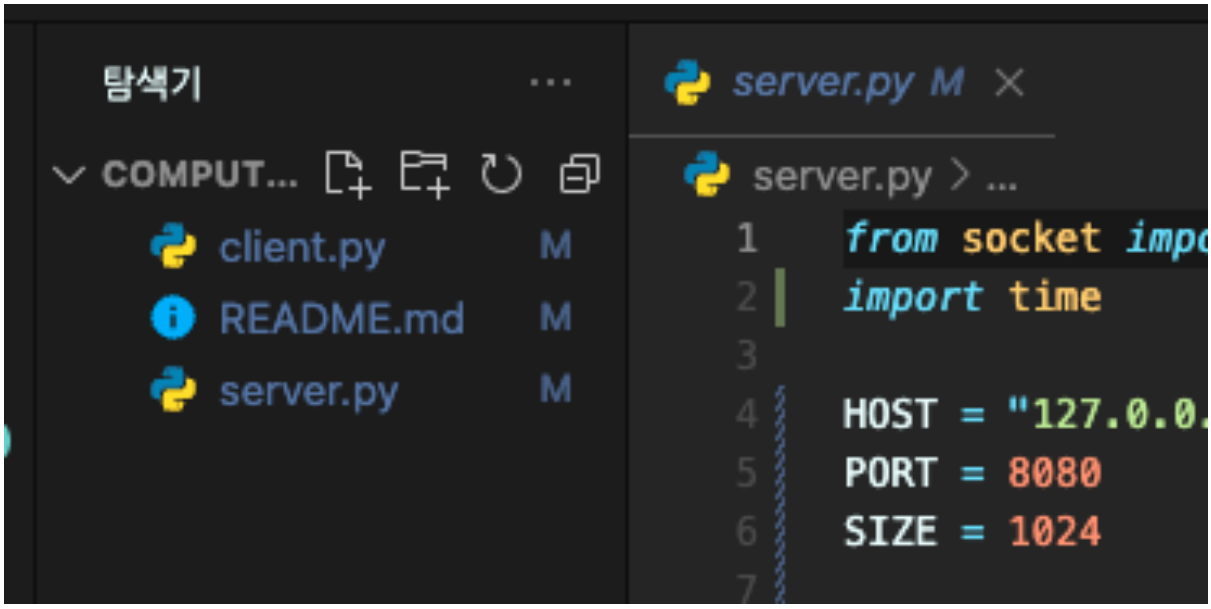
- socket생성후 연결
- test_case에서 url, method, data받아서 request_formating함수에 넣고 clinet가 보낼 HTTP Request생성
- send함수로 server에게 request 전송
- recv함수로 server가 보낼 response를 수신 대기
- response가 오면 출력후 socket 종료

```
for test in test_case:
    with socket(AF_INET, SOCK_STREAM) as client_socket: # 소켓 객체 생성
        client_socket.connect((IP, PORT))  # 생성한 소켓에 HOST와 PORT 연결
        method = test['method']
        url = test['url']
        data = test['data']
        request = request_formating(method, data, url)
        client_socket.send(request.encode('utf-8')) # 메세지 전송
        response = client_socket.recv(SIZE).decode('utf-8')
# 클라이언트가 보낸 메세지 수신 대기
        print(response) # 수신 받은 문자 출력
        print('\n---------------------------------------\n')
        client_socket.close()
```

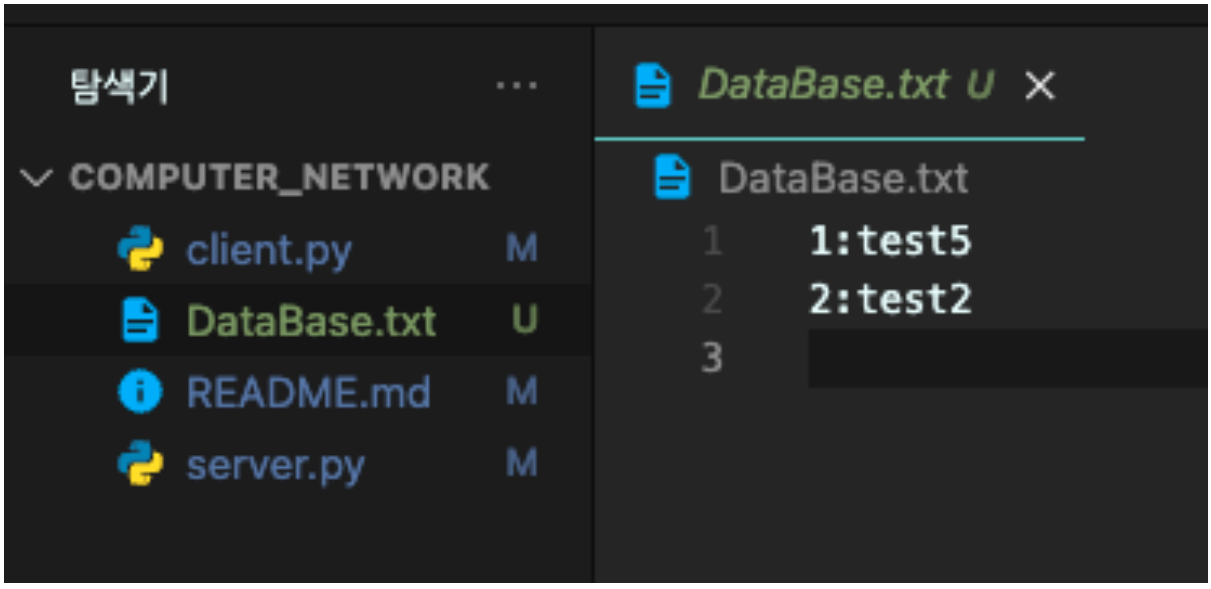
Result

1. 코드 실행전, 실행 후

- 실행 전에는 DataBase.txt 파일이 없었는데



- 실행후 DataBase.txt 파일에 client가 보낸 data가 들어있는 상태로 생성되었다.



2. case별 코드 실행 결과

* 우측은 server.py 출력 결과, 좌측은 client.py 출력 결과

- case 1 : url : 127.0.0.1/, method : HEAD, data : ‘’
    - server가 열려있는지 확인

- case 2 : url : 127.0.0.1/create, method : POST, data : ‘1:test1’
    - method로 POST를 요청하면 server는 client가 준 data를 DataBase에 추가
    - data가 잘 저장 되었다면 ‘Created DB’라는 메세지와 201, CREATED라는 status를 response

- case 3 : url : 127.0.0.1/create, method : POST, data : ‘2:test2’
    - method로 POST를 요청하면 server는 client가 준 data를 DataBase에 추가
    - data가 잘 저장 되었다면 ‘Created DB’라는 메세지와 201, CREATED라는 status를 response

- case 4 : url : 127.0.0.1/create, method : PUT, data : ‘3:test3’
    - method가 PUT을 요청하면 Database를 업데이트 하기 위해 url의 path가 update가 와야하는데 해당 case에서는 create가 왔기 때문에 server는 400, BAD_REQUEST를 response

```
PUT / HTTP/1.1
Host: 127.0.0.1/create
Connection: keep-alive
Content-Type: text/html
Content-Length: 7
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit

3:test3

____________________________________
```

```
HTTP/1.1 400 BAD_REQUEST
Date: Sun, 01 May 2022 17:50:16 GMT
Content-Type: text/html
Connection: keep-alive
Content-Length: 7

BAD_REQUEST

____________________________________
```

- case 5 : url : 127.0.0.1/, method : GET, data : ‘’
    - method로 GET를 요청하면 server는 DataBase의 데이터를 읽어서 client에게 response
    - client는 server에게 받은 현재 DataBase에 있는 Data들을 출력
    - server는 client에게 200, OK를 response

```
GET / HTTP/1.1
Host: 127.0.0.1/
Connection: keep-alive
Content-Type: text/html
Content-Length: 0
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit



____________________________________
```

```
HTTP/1.1 200 OK
Date: Sun, 01 May 2022 17:50:16 GMT
Content-Type: text/html
Connection: keep-alive
Content-Length: 0

['1:test1', '2:test2']

____________________________________
```

- case 6 : url : 127.0.0.1/update, method : PUT, data : ‘1:test5’
    - method가 PUT을 요청하면 Database를 업데이트 하기 위해 url의 path가 update가 와야하고 해당 case에서는 path가 update이기 때문에 key가 1인 data의 value를 바꿈
    - data를 update했다면 server는 client에게 200, OK와 ‘Update DataBase’ 라는 문장을 response

```
PUT / HTTP/1.1
Host: 127.0.0.1/update
Connection: keep-alive
Content-Type: text/html
Content-Length: 7
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit

1:test5

____________________________________
```

```
HTTP/1.1 200 OK
Date: Sun, 01 May 2022 17:50:16 GMT
Content-Type: text/html
Connection: keep-alive
Content-Length: 7

Update DataBase

____________________________________
```

- case 7 : url : 127.0.0.1/ComputerNetwork, method : POST, data : ‘6:test6’
    - method가 POST을 요청하면 Database에 data를 추가 하기 위해 url의 path가 create가 와야하는데 해당 case에서는 ComputerNetwork가 왔기 때문에 server는 400, BAD_REQUEST를 response

```
POST / HTTP/1.1
Host: 127.0.0.1/ComputerNetwork
Connection: keep-alive
Content-Type: text/html
Content-Length: 7
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit

6:test6

------------------------------------
```

```
HTTP/1.1 400 BAD_REQUEST
Date: Sun, 01 May 2022 17:50:16 GMT
Content-Type: text/html
Connection: keep-alive
Content-Length: 7

BAD_REQUEST

------------------------------------
```

- case 8 : url : 127.0.0.1/, method : GET, data : ‘’
    - method로 GET를 요청하면 server는 DataBase의 데이터를 읽어서 client에게 response
    - client는 server에게 받은 현재 DataBase에 있는 Data들을 출력
    - server는 client에게 200, OK를 response
    - case5의 GET 요청 결과와 비교하면 중간에 PUT 요청으로 data가 업데이트 되어서 key가 1인 data가 변경된 것을 확인 가능

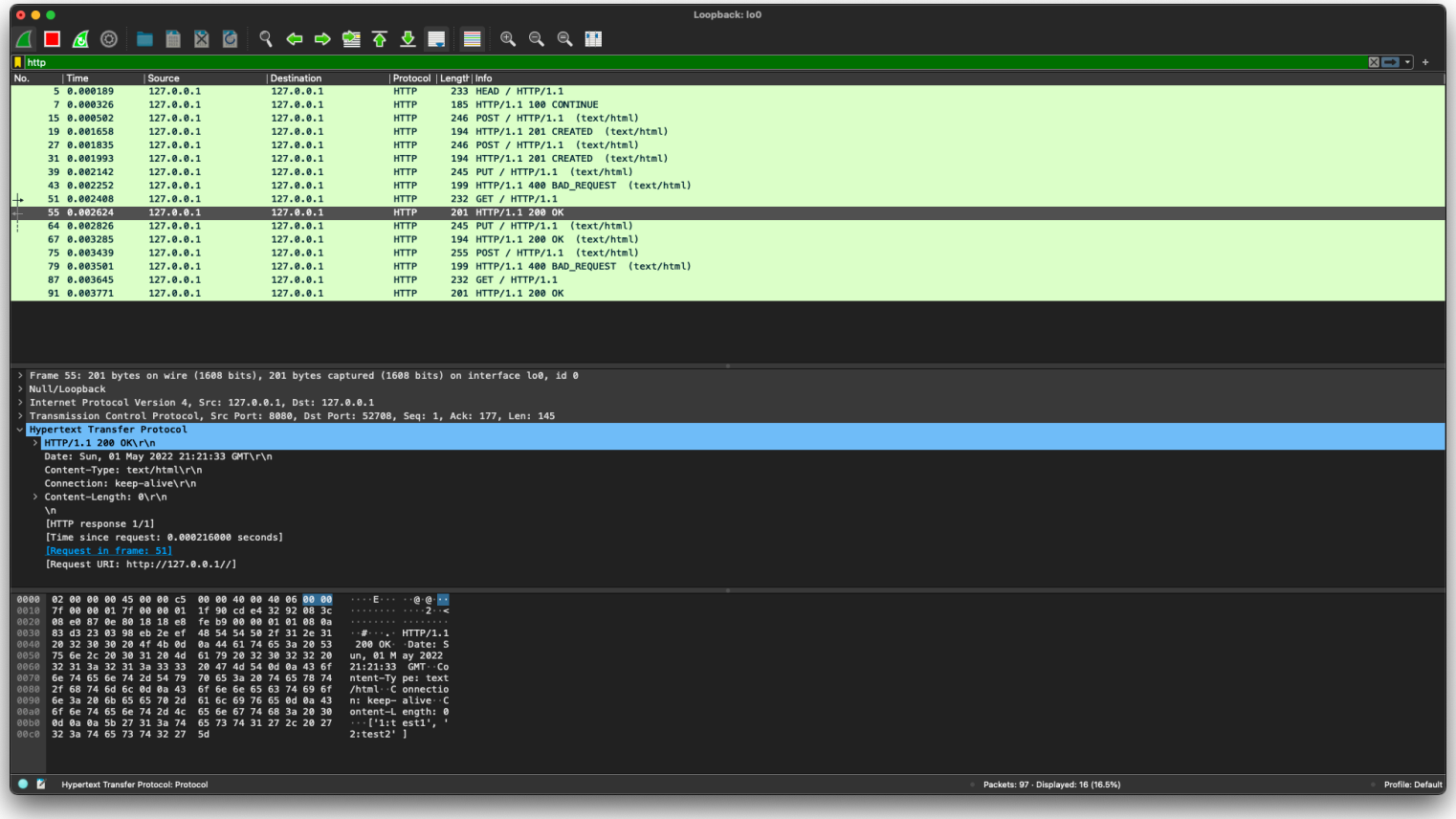
```
GET / HTTP/1.1
Host: 127.0.0.1/
Connection: keep-alive
Content-Type: text/html
Content-Length: 0
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit
```

```
------------------------------------

HTTP/1.1 200 OK
Date: Sun, 01 May 2022 17:50:16 GMT
Content-Type: text/html
Connection: keep-alive
Content-Length: 0

['1:test5', '2:test2']
```

3. Wireshark


Loopback: lo0
http
No. Time Source Destination Protocol Length Info
5 0.000189 127.0.0.1 127.0.0.1 HTTP 233 HEAD / HTTP/1.1
7 0.000326 127.0.0.1 127.0.0.1 HTTP 185 HTTP/1.1 100 CONTINUE
15 0.000502 127.0.0.1 127.0.0.1 HTTP 246 POST / HTTP/1.1 (text/html)
19 0.001658 127.0.0.1 127.0.0.1 HTTP 194 HTTP/1.1 201 CREATED (text/html)
27 0.001835 127.0.0.1 127.0.0.1 HTTP 246 POST / HTTP/1.1 (text/html)
31 0.001993 127.0.0.1 127.0.0.1 HTTP 194 HTTP/1.1 201 CREATED (text/html)
39 0.002142 127.0.0.1 127.0.0.1 HTTP 245 PUT / HTTP/1.1 (text/html)
43 0.002252 127.0.0.1 127.0.0.1 HTTP 199 HTTP/1.1 400 BAD_REQUEST (text/html)
51 0.002408 127.0.0.1 127.0.0.1 HTTP 232 GET / HTTP/1.1
55 0.002624 127.0.0.1 127.0.0.1 HTTP 201 HTTP/1.1 200 OK
64 0.002826 127.0.0.1 127.0.0.1 HTTP 245 PUT / HTTP/1.1 (text/html)
67 0.003285 127.0.0.1 127.0.0.1 HTTP 194 HTTP/1.1 200 OK (text/html)
75 0.003439 127.0.0.1 127.0.0.1 HTTP 255 POST / HTTP/1.1 (text/html)
79 0.003501 127.0.0.1 127.0.0.1 HTTP 199 HTTP/1.1 400 BAD_REQUEST (text/html)
87 0.003645 127.0.0.1 127.0.0.1 HTTP 232 GET / HTTP/1.1
91 0.003771 127.0.0.1 127.0.0.1 HTTP 201 HTTP/1.1 200 OK
Frame 55: 201 bytes on wire (1608 bits), 201 bytes captured (1608 bits) on interface lo0, id 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 8080, Dst Port: 52708, Seq: 1, Ack: 177, Len: 145
Hypertext Transfer Protocol
HTTP/1.1 200 OK\r\n
Date: Sun, 01 May 2022 21:21:33 GMT\r\n
Content-Type: text/html\r\n
Connection: keep-alive\r\n
Content-Length: 0\r\n
\n
[HTTP response 1/1]
[Time since request: 0.000216000 seconds]
[Request in frame: 51]
[Request URI: http://127.0.0.1//]
0000 02 00 00 00 45 00 00 c5 00 00 40 00 40 06 00 00 ····E··· ··@·@···
0010 7f 00 00 01 7f 00 00 01 1f 90 cd e4 32 92 08 3c ···· ···· ····2··<
0020 08 e0 87 0e 80 18 18 e8 fe b9 00 00 01 01 08 0a ···· ···· ····
0030 83 d3 23 03 98 eb 2e ef 48 54 54 50 2f 31 2e 31 ··#· ···· HTTP/1.1
0040 20 32 30 30 20 4f 4b 0d 0a 44 61 74 65 3a 20 53 200 OK· ·Date: S
0050 75 6e 2c 20 30 31 20 4d 61 79 20 32 30 32 32 20 un, 01 M ay 2022
0060 32 31 3a 32 31 3a 33 33 20 47 4d 54 0d 0a 43 6f 21:21:33 GMT ·Co
0070 6e 74 65 6e 74 2d 54 79 70 65 3a 20 74 65 78 74 ntent-Ty pe: text
0080 2f 68 74 6d 6c 0d 0a 43 6f 6e 6e 65 63 74 69 6f /html·· C onnectio
0090 6e 3a 20 6b 65 65 70 2d 61 6c 69 76 65 0d 0a 43 n: keep- alive··C
00a0 6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 68 3a 20 30 ontent-L ength: 0
00b0 0d 0a 0a 5b 27 31 3a 74 65 73 74 31 27 2c 20 27 ···['1:t est1', '
00c0 32 3a 74 65 73 74 32 27 5d 2:test2']
Hypertext Transfer Protocol: Protocol
Packets: 97 · Displayed: 16 (16.5%)
Profile: Default