Web Server Berbasis TCP dengan menerapkan Socket Programming

Oleh:

- Achmad Rafly Khatami Zain
- Alviko Pradipta Harianto
- Mikail Ardyas Wibowo

Pendahuluan

Topik ini membahas implementasi dan pengujian sebuah server web sederhana serta klien HTTP menggunakan Python. Server web dirancang untuk menangani satu permintaan HTTP pada satu waktu, tetapi dapat melayani beberapa permintaan secara simultan dengan menggunakan threading. Klien HTTP terhubung ke server melalui koneksi TCP, mengirimkan permintaan HTTP, dan menampilkan respons server sebagai output.

Implementasi Server HTTP (Single Thread)

File server satu thread ini hanya bertanggung jawab untuk melayani satu permintaan HTTP dari client. Saat menggunakan satu thread, server dapat menerima koneksi dari klien, menganalisis permintaan HTTP, dan mengirimkan respons kembali ke klien.

Terdapat dua fungsi utama di dalam file server kami yaitu :

1. handle_request(client_connection) : Bertanggung jawab untuk mengelola permintaan yang diterima dari klien.

2. start_server() : Digunakan untuk memulai server web. Pertama, ia membuat socket server menggunakan alamat dan port yang telah ditentukan.

Algoritma Lengkap dari file server (single thread)

```python
import socket
import os

# Menentukan alamat dan port server
serverHost = '127.0.0.16'
serverPort = 12345

def handle_client(client_connection):
    # Menerima permintaan dari klien dan mendekodekannya
    request = client_connection.recv(1024).decode('utf-8')

    # Menganalisis permintaan HTTP
    request_lines = request.split("\r\n")
    request_method, request_path, _ = request_lines[0].split(" ")

    # Mendapatkan path file yang diminta
    if request_path == '/':
        request_path = '/index.html'

    file_path = '.' + request_path

    # Membuat pesan respons HTTP
    if os.path.exists(file_path):
        with open(file_path, 'rb') as file:
            response_body = file.read()
        response_status_line = "HTTP/1.1 200 OK\r\n"
        response_headers = f"Content-Type: text/html\r\nContent-Length: {len(response_body)}\r\n\r\n"
        response = response_status_line.encode('utf-8') + response_headers.encode('utf-8') + response_body
    else:
        response = b"HTTP/1.1 404 Not Found\r\n\r\n<h1>404 Not Found</h1>"

    # Mengirim respons ke klien
    client_connection.sendall(response)

    # Menutup koneksi dengan klien
    client_connection.close()
```

Algoritma Lengkap dari file server (single thread)

```
37
38  def start_server():
39      # Membuat socket server
40      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
41          server_socket.bind((serverHost, serverPort))
42          server_socket.listen(1)
43          print(f"Server berjalan di http://{serverHost}:{serverPort}")
44
45          while True:
46              # Menerima koneksi dari klien
47              client_connection, client_address = server_socket.accept()
48              print(f"Terhubung dengan {client_address}")
49
50              # Menangani permintaan dari klien
51              handle_client(client_connection)
52
53  if __name__ == "__main__":
54      start_server()
55
```

Implementasi Server HTTP (Multi Thread)

Server HTTP dikembangkan menggunakan Python dengan menerapkan Socket Programming. Server menerima permintaan HTTP dari klien, mem-parsing permintaan tersebut, menangani permintaan sesuai dengan spesifikasi, dan mengirimkan respons HTTP ke klien.

Terdapat dua fungsi utama di dalam file server kami yaitu :

1. handle_client(connectionSocket) : Fungsi handle_client bertujuan untuk menangani permintaan HTTP dari klien, memproses permintaan tersebut dengan membaca file yang diminta, dan mengirimkan respons yang sesuai kembali ke klien.

2. run_server() : Fungsi ini merupakan inti dari server.

Algoritma Lengkap dari file server (multi thread)

```python
from socket import *
import sys
import threading

def handle_client(connectionSocket):
    try:
        # Menerima data dari klien hingga 1024 byte dan mendekodenya menjadi string
        message = connectionSocket.recv(1024).decode()

        # Cetak informasi koneksi dan permintaan
        print(f"Terhubung dengan {connectionSocket.getpeername()}")  # Menampilkan alamat IP dan port klien
        print(f"Permintaan diterima: {message.splitlines()[0]}")  # Menampilkan baris pertama dari HTTP request
        host_info = next((line for line in message.splitlines() if line.startswith("Host: ")), "Host: Tidak Diketahui")
        print(host_info)  # Menampilkan informasi Host dari permintaan, atau Host: Tidak Diketahui" jika tidak ada
        print(f"Connection: {connectionSocket.getsockname()}")  # Menampilkan alamat IP dan port server

        # Memeriksa apakah pesan tidak kosong
        if len(message.split()) < 2:
            raise IOError

        # Mendapatkan nama file yang diminta
        filename = message.split()[1]
        # Membuka file yang diminta
        f = open(filename[1:], 'r')
        # Membaca isi file
        outputdata = f.read()

        # Mengirim satu baris header HTTP ke socket
        connectionSocket.send("HTTP/1.1 200 OK\r\n\r\n".encode())

        # Mengirim konten dari file yang diminta ke klien
        connectionSocket.send(outputdata.encode())

    except IOError:
        # Mengirim pesan respons untuk file yang tidak ditemukan
        connectionSocket.send("HTTP/1.1 404 Not Found\r\n\r\n".encode())

    finally:
        # Menutup socket klien
        connectionSocket.close()
```

Algoritma Lengkap dari file server (multi thread)

```python
def start_server():
    # Membuat socket server TCP
    serverSocket = socket(AF_INET, SOCK_STREAM)

    # Mempersiapkan socket server
    serverPort = 1234
    serverHost = '127.0.0.15'

    serverSocket.bind((serverHost, serverPort))  # Mengikat socket ke alamat dan port tertentu
    serverSocket.listen(5)  # Mendengarkan hingga 5 koneksi

    print(f"Server berjalan di http://{serverHost}:{serverPort}")

    while True:
        # Menerima koneksi baru
        connectionSocket, addr = serverSocket.accept()
        print(f"Terhubung dengan {addr}")  # Menampilkan alamat IP dan port klien yang terhubung

        # Membuat thread baru untuk menangani permintaan klien
        client_thread = threading.Thread(target=handle_client, args=(connectionSocket,))
        client_thread.start()

    # Menutup socket server (kode ini tidak akan pernah tercapai karena loop while True)
    serverSocket.close()
    sys.exit()  # Mengakhiri program setelah mengirim data yang sesuai

if __name__ == "__main__":
    start_server()
```

Implementasi Klien HTTP

Klien HTTP dibuat menggunakan Python untuk terhubung ke server HTTP, mengirimkan permintaan HTTP, dan menampilkan respons dari server sebagai output.

Terdapat satu fungsi utama di dalam file klien kami yaitu :

- **run_client(server_host, server_port, filename) : Dari parameternya fungsi ini menerima tiga parameter yaitu “server_host”, “server_port”, dan “filename”. Fungsi ini juga memiliki berbagai macam fungsi seperti :**

1. **Membuat Socket Klien**
2. **Membuat Koneksi dengan Server**
3. **Mengirim Permintaan HTTP GET**
4. **Menerima Respons dari Server**
5. **Menampilkan Respons dari Server**
6. **Menangani Kesalahan**

Algoritma lengkap dari file client

```
You, 2 hours ago | 1 author (You)
from socket import *      You, 2 hours ago • Uncommitted changes
import sys

def run_client(server_host, server_port, filename):
    # Membuat socket klien TCP
    clientSocket = socket(AF_INET, SOCK_STREAM)

    try:
        # Membuat koneksi dengan server
        clientSocket.connect((server_host, server_port))

        # Mengirim permintaan HTTP GET
        request = f"GET /{filename} HTTP/1.1\r\nHost: {server_host}\r\n\r\n"
        clientSocket.sendall(request.encode())

        # Menerima respons dari server
        response = b""
        while True:
            data = clientSocket.recv(1024)
            if not data:
                break
            response += data

        # Menampilkan respons dari server
        print(response.decode())

    except Exception as e:
        print(f"Error: {e}")

    finally:
        # Menutup socket klien
        clientSocket.close()
```

Algoritma lengkap dari file client

```
34  if __name__ == "__main__":
35      # Memeriksa argumen baris perintah
36      if len(sys.argv) != 4:
37          print("Usage: client.py server_host server_port filename")
38          sys.exit(1)
39
40      # Mendapatkan argumen dari baris perintah
41      server_host = sys.argv[1]
42      server_port = int(sys.argv[2])
43      filename = sys.argv[3]
44
45      # Menjalankan fungsi klien
46      run_client(server_host, server_port, filename)
47
```

Pengujian (Single Thread)

Berikut adalah cara - cara testing yang kami lakukan :

1. **Kita perlu menjalankan file server di dalam terminal yang sesuai dengan file direktori.**

```
C:\Users\rafly\OneDrive\Desktop\File Telkom\Semester 4\Jaringan Komputer\tubes>python server_single.py
Server berjalan di http://127.0.0.16:12345
```

2. **Kita akan menjalankan file client dengan perintah sebagai berikut “python client.py server_host server_port file_name”.**

Pengujian (Single Thread)

3. Berikut adalah keluaran dari file client dan server:

```
C:\Users\rafly\OneDrive\Desktop\File Telkom\Semester 4\Jaringan Komputer\tubes>python client.py 127.0.0.16 12345 Test.html
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 262

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=, initial-scale=1.0">
    <title>Tugas Besar Jaringan Komputer</title>
</head>
<body>
    <h1>TUGAS BESAR JARINGAN KOMPUTER</h1>
</body>
</html>
```

```
C:\Users\rafly\OneDrive\Desktop\File Telkom\Semester 4\Jaringan Komputer\tubes>python server_single.py
Server berjalan di http://127.0.0.16:12345
Terhubung dengan ('127.0.0.1', 58889)
```

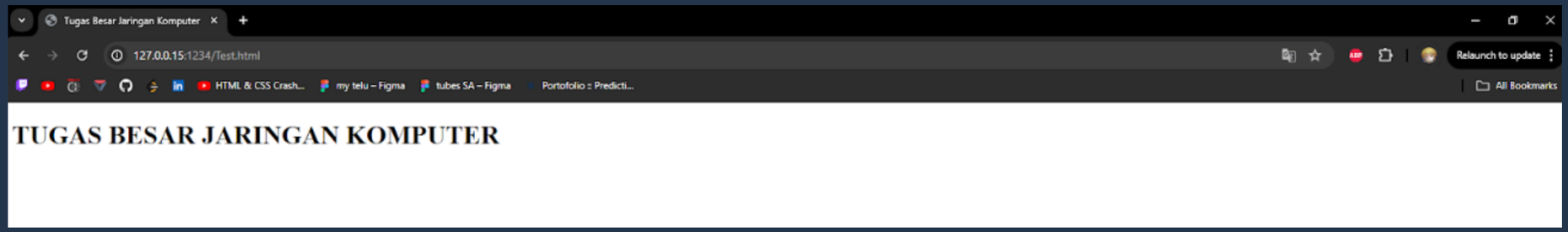
Pengujian (Multi Thread)

Terdapat suatu perbedaan pada multi thread kita bisa membuka berbagai macam file di dalam browser dan di cmd. Sedangkan pada single thread kita hanya bisa membuka satu di cmd atau di browser. Berikut adalah cara - cara testing yang kami lakukan :

1. Kita perlu menjalankan file server di dalam terminal yang sesuai dengan file direktori.

```
C:\Users\rafly\OneDrive\Desktop\File Telkom\Semester 4\Jaringan Komputer\tubes>python server_multi.py
Server berjalan di http://127.0.0.15:1234
```

2. Kita akan membuka file HTML dengan link yang sesuai dengan nomor host dan nomor port yaitu “http://127.0.0.15:8080/test1.html”



Pengujian (Multi Thread)

3. Kita akan menjalankan file client dengan perintah sebagai berikut “python client.py server_host server_port file_name”.

4. Berikut adalah keluaran dari client dan server setelah kita membuka semua file dan menjalankan client sesuai dengan perintahnya

```
C:\Users\rafly\OneDrive\Desktop\File Telkom\Semester 4\Jaringan Komputer\tubes>python client.py 127.0.0.15 1234 test1.html
HTTP/1.1 200 OK

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tugas Besar Jaringan Komputer</title>
</head>
<body>
    <h1>TUBES JARINGAN KOMPUTER</h1>
</body>
</html>
```

Pengujian (Multi Thread)

```
C:\Users\rafly\OneDrive\Desktop\File Telkom\Semester 4\Jaringan Komputer\tubes>python client.py 127.0.0.15 1234 Test.html
HTTP/1.1 200 OK

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=, initial-scale=1.0">
    <title>Tugas Besar Jaringan Komputer</title>
</head>
<body>
    <h1>TUGAS BESAR JARINGAN KOMPUTER</h1>
</body>
</html>
```

Pengujian (Multi Thread)

```
C:\Users\rafly\OneDrive\Desktop\File Telkom\Semester 4\Jaringan Komputer\tubes>python server_multi.py
Server berjalan di http://127.0.0.15:1234
Terhubung dengan ('127.0.0.1', 58810)
Terhubung dengan ('127.0.0.1', 58811)
Terhubung dengan ('127.0.0.1', 58810)
Permintaan diterima: GET /Test.html HTTP/1.1
Host: 127.0.0.15:1234
Connection: ('127.0.0.15', 1234)
Terhubung dengan ('127.0.0.1', 58811)
Permintaan diterima: GET /favicon.ico HTTP/1.1
Host: 127.0.0.15:1234
Connection: ('127.0.0.15', 1234)
Terhubung dengan ('127.0.0.1', 58818)
Terhubung dengan ('127.0.0.1', 58818)
Permintaan diterima: GET /test1.html HTTP/1.1
Host: 127.0.0.15
Connection: ('127.0.0.15', 1234)
Terhubung dengan ('127.0.0.1', 58826)
Terhubung dengan ('127.0.0.1', 58826)
Permintaan diterima: GET /test2.html HTTP/1.1
Host: 127.0.0.15
Connection: ('127.0.0.15', 1234)
Terhubung dengan ('127.0.0.1', 58827)
Terhubung dengan ('127.0.0.1', 58827)
Permintaan diterima: GET /Test.html HTTP/1.1
Host: 127.0.0.15
Connection: ('127.0.0.15', 1234)
```

Kesimpulan

File server single thread ini hanya dapat mengatasi satu permintaan HTTP pada satu waktu. Ini berarti server akan memproses satu permintaan dari klien secara berurutan, sehingga klien lain harus menunggu hingga permintaan sebelumnya selesai diproses. Sebaliknya, server dengan multithreading dapat menangani banyak permintaan HTTP secara bersamaan, memungkinkan server untuk melayani beberapa klien secara paralel dan lebih efisien dalam lingkungan dengan banyak permintaan.

Terima Kasih