

WEB SERVER BERBASIS TCP
MULTITHREADING DAN SINGLE THREADING



Dosen pembimbing:
Beliau

Disusun oleh:

Farizsyach Razif Januar	1301220439
Imam Wijayanto	1301223117
Muhammad Hilal Abyan	1301223262

**PROGRAM STUDI S1 INFORMATIKA
FAKULTAS INFORMATIKA
(2023/2024)**

DAFTAR ISI

DAFTAR ISI	2
BAB IPENDAHULUAN	3
A. Tujuan Penulisan Laporan	3
B. Landasan Teori.....	3
1. TCP	3
2. Socket Programing	3
BAB II IMPLEMENTASI.....	4
DAFTAR PUSTAKA	11

BAB I

PENDAHULUAN

A. Tujuan Penulisan Laporan

Pada penulisan laporan ini tujuan yang akan dicapai yaitu:

- a) Dapat membuat serta mengimplementasikan TCP socket dan mengaitkannya ke alamat dan port tertentu
- b) Membuat program web server dapat menerima dan memparsing HTTP request yang dikirimkan
- c) Membuat web server dapat mencari dan mengambil file (dari file system) yang diminta oleh client
- d) Membuat web server dapat membuat HTTP response message yang terdiri dari header dan konten file yang diminta
- e) Membuat web server dapat mengirimkan response message yang sudah dibuat ke browser (client) dan dapat ditampilkan dengan benar di sisi client
- f) Membuat web server yang dapat mengirimkan pesan “404 *Not Found*” dan dapat ditampilkan dengan benar di sisi client.
- g) Membuat web server dengan multithreading dan single threading.
- h) Membuat client yang dapat meminta file ke server lalu menampilkan di terminal.

B. Landasan Teori

1. TCP

“Transmission Control Protocol (TCP) adalah salah satu protokol jaringan yang paling umum digunakan untuk mengontrol pengiriman data antar komputer di dalam jaringan. TCP beroperasi di lapisan transport dalam model referensi jaringan OSI (Open Systems Interconnection)” (Adya, 2023).

2. Socket Programing

“Pemrograman socket merupakan pemrograman yang digunakan untuk melakukan komunikasi proses (process-to-process) dalam sebuah jaringan. Selain komunikasi proses, dalam sebuah jaringan komputer juga melakukan komunikasi host (host-to-host). Dalam pengujian pemrograman socket dapat dilakukan dengan menggunakan jenis socket networking sock_stream pada client ke server dengan menggunakan alamat proses (IP dan Port) pada komputer yang sama dengan direktori yang sama dan yang berbeda serta dengan komputer lain yang berbeda.” (Supriyanto, 2005).

BAB II IMPLEMENTASI

1. Buat server_singlethread.py

```
jarkom-tubes > server_singlethread.py > handle_client
1  from socket import *
2  import mimetypes
3
4
5  def handle_client(client_connection):
6      request = client_connection.recv(1024).decode()
7      print(request)
8      # Memisahkan request per baris
9      headers = request.split('\n')
10     # Mengambil file yang diminta yang ada pada header
11     # pada baris pertama setelah request method
12     file_requested = headers[0].split()[1]
13     # index.html sebagai default saat server dijalankan
14     if file_requested == '/':
15         file_requested = '/index.html'
16     try:
17         # Membuka file yang diminta oleh klien
18         with open('./data/'+file_requested, 'rb') as file:
19             content = file.read()
20
21         # Mengambil content-type dari request klien
22         content_type, _ = mimetypes.guess_type(file_requested)
23         # Membuat response header dengan kode 200 OK dan tipe content
24         response_header = f'HTTP/1.0 200 OK\nContent-Type: {content_type}\n\n'.encode()
25         client_connection.send(response_header + content)
26     except FileNotFoundError:
27         # Error Handling jika file yang direquest tidak ditemukan
28         response_header = 'HTTP/1.0 404 NOT FOUND\n\nFile Not Found'.encode()
29         response_content = b''
30         client_connection.send(response_header + response_content)
31     client_connection.close()
32
33 def run_server(server_hostname, server_port):
34     serverSocket = socket(AF_INET, SOCK_STREAM)
35     serverSocket.bind((server_hostname, server_port))
36     serverSocket.listen(1)
37     print(f'[*] Listening on {server_hostname}:{server_port}...')
38     while True:
39         client_connection, client_address = serverSocket.accept()
40         print(f'[*] Accepted connection from {client_address[0]}:{client_address[1]}')
41         handle_client(client_connection)
42
43 def main():
44     server_hostname = 'localhost'
45     server_port = 7070
46
47     # Memulai server dengan host dan port yang ditentukan
48     run_server(server_hostname, server_port)
49
50 # Memanggil fungsi main
51 main()
52
```

Fungsi `handle_client` bertanggung jawab untuk menangani setiap koneksi dari klien. Saat koneksi diterima, fungsi ini pertama-tama membaca pesan HTTP yang dikirim oleh klien menggunakan metode `recv()` dari objek koneksi. Pesan tersebut kemudian dipecah menjadi baris-baris header menggunakan `split('\n')`, dan file yang diminta oleh klien diambil dari baris pertama setelah request method. Jika file yang diminta adalah root (`/`), maka file yang akan dikirim adalah `index.html` sebagai default.



Selanjutnya, fungsi mencoba membuka file yang diminta oleh klien menggunakan `open()`, dengan mode membaca biner (`'rb'`). Jika file tersebut ditemukan, isinya dibaca menggunakan `read()`, dan tipe kontennya ditebak menggunakan `mimetypes.guess_type()`. Setelah itu, fungsi membuat header respons HTTP dengan kode status 200 OK dan tipe konten yang sesuai. Header tersebut dikirimkan ke klien, diikuti dengan konten file yang telah dibaca.

Namun, jika file yang diminta tidak ditemukan, fungsi akan menangani kesalahan tersebut dengan mengirimkan respons HTTP 404 NOT FOUND kepada klien. Setelah menangani permintaan klien, koneksi ditutup menggunakan `close()`.

Fungsi `run_server`, di sisi lain, bertanggung jawab untuk mengelola koneksi masuk. Saat server pertama kali dijalankan, ia menggunakan `bind()` untuk mengikat socket server ke alamat dan port yang ditentukan, kemudian memulai mendengarkan koneksi masuk menggunakan `listen()`. Ketika koneksi diterima, fungsi `accept()` digunakan untuk menerima koneksi tersebut dari antrean koneksi yang masuk. Setelah menerima koneksi, fungsi `handle_client` dipanggil untuk menangani koneksi tersebut. Ini memungkinkan server untuk melayani permintaan dari banyak klien secara efisien. Dengan cara ini, server dapat beroperasi dalam loop tak terbatas, selalu siap untuk menerima koneksi baru dan menangani permintaan dari klien-klien yang terhubung.

Fungsi `main` digunakan untuk inisiasi nilai port server dan nama hostname dan menjalankan fungsi `run_server` untuk mengaktifkan server.

2. Buat `server_multithread.py`

jarkom-tubes >  server_multithread.py >  run_server

```
1  from socket import *
2  import threading
3  import mimetypes
4
5  def handle_client(client_connection):
6      request = client_connection.recv(1024).decode()
7      print(request)
8      #Memisahkan header per baris
9      headers = request.split('\n')
10     #Mengambil file yang diminta yang ada pada header
11     #pada baris pertama setelah request method
12     file_requested = headers[0].split()[1]
13     #index.html sebagai default saat server dijalankan
14     if file_requested == '/':
15         file_requested = '/index.html'
16     try:
17         #Membuka file yang diminta oleh klien
18         with open('./data/'+file_requested, 'rb') as file:
19             content = file.read()
20
21         #Mengambil content-type dari request klien
22         content_type, _ = mimetypes.guess_type(file_requested)
23         #Membuat response header dengan kode 200 OK dan tipe content
24         response_header = f'HTTP/1.0 200 OK\nContent-Type: {content_type}\n\n'.encode()
25         client_connection.send(response_header+content)
26     except FileNotFoundError:
27         #Error Hnadling jika file yang direquest tidak ditemukan
28         response_header = 'HTTP/1.0 404 NOT FOUND\n\nFile Not Found'.encode()
29         response_content = b''
30         client_connection.send(response_header+response_content)
31     client_connection.close()
32
33 def run_server(server_hostname, server_port):
34     serverSocket = socket(AF_INET, SOCK_STREAM)
35     serverSocket.bind((server_hostname, server_port))
36     serverSocket.listen(1)
37     print(f'[*] Listening on {server_hostname}:{server_port}...')
38     while True:
39         client_connection, client_address = serverSocket.accept()
40         print(f"[*] Accepted connection from {client_address[0]}:{client_address[1]}")
41         client_handler = threading.Thread(target=handle_client, args=(client_connection,))
42         client_handler.start()
43
44 def main():
45     server_hostname = 'localhost'
46     server_port = 6969
47     run_server(server_hostname, server_port)
48
49 # Memanggil fungsi main
50 main()
51
```

Fungsi `handle_client` merupakan inti dari server HTTP ini. Ketika server menerima koneksi dari klien, fungsi ini bertanggung jawab untuk menangani permintaan klien. Pertama, ia membaca pesan HTTP yang dikirim oleh klien dan memisahkan bagian header dari konten. Kemudian, dari header, fungsi ini mengidentifikasi file yang diminta oleh klien. Jika file yang diminta adalah '/' (root), fungsi akan mengarahkannya ke file 'index.html' sebagai default. Selanjutnya, fungsi mencoba membuka file yang diminta oleh klien. Jika file tersebut ditemukan, isinya dibaca dan dikirimkan sebagai respons HTTP 200 OK bersama dengan tipe kontennya. Namun, jika file tidak ditemukan, server akan mengirimkan respons HTTP 404 NOT FOUND, memberi tahu klien bahwa file yang diminta tidak tersedia. Setelah menangani permintaan klien, koneksi ditutup.

Fungsi `run_server` digunakan untuk mengelola koneksi masuk. Ketika server pertama kali dijalankan, fungsi ini mengikat socket server ke alamat dan port yang ditentukan. Kemudian, dengan menggunakan fungsi `listen`, server mulai mendengarkan koneksi masuk. Ketika koneksi diterima, fungsi `accept()` digunakan untuk menerima koneksi tersebut dari antrian koneksi yang masuk. Setelah menerima koneksi, fungsi membuat thread baru yang memanggil fungsi `handle_client` untuk menangani koneksi tersebut. Ini memungkinkan server untuk menjalankan beberapa koneksi secara bersamaan melalui penggunaan threading. Dengan cara ini, server dapat melayani permintaan dari banyak klien secara efisien. Selanjutnya, server terus berjalan dalam loop tak terbatas, selalu siap untuk menerima koneksi baru dan menangani permintaan dari klien-klien tersebut.

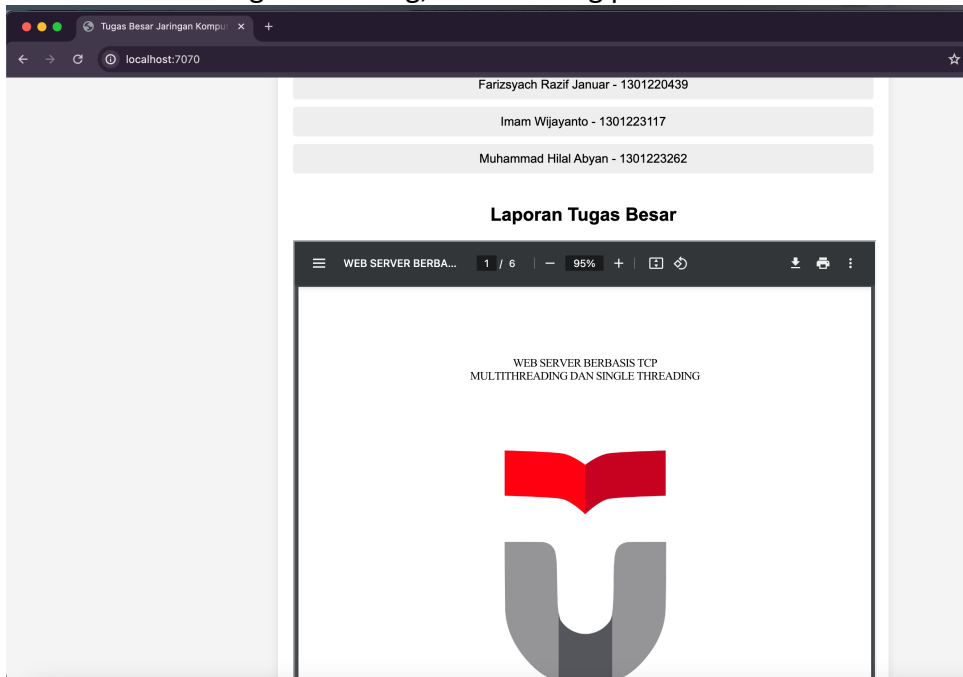
Fungsi `main` digunakan untuk inisiasi nilai port server dan nama hostname dan menjalankan fungsi `run_server` untuk mengaktifkan server.

3. Buat client.py

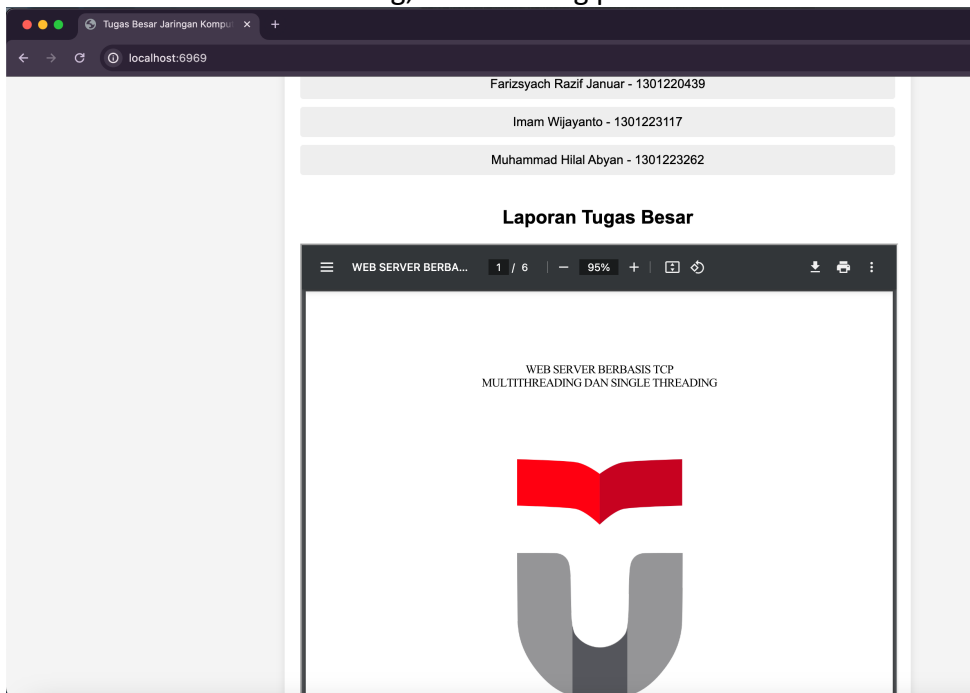
jarkom-tubes > client.py > ...

```
1  from socket import *
2  import sys
3
4  def request_file(server_hostname, server_port, file_requested):
5      # Create a TCP/IP socket
6      client_socket = socket(AF_INET, SOCK_STREAM)
7      client_socket.connect((server_hostname, server_port))
8
9      # Formulate the HTTP GET request
10     request = f'GET {file_requested} HTTP/1.0\r\nHost: {server_hostname}\r\n\r\n'
11     client_socket.send(request.encode())
12
13     # Receive the response from the server
14     response = b''
15     while True:
16         chunk = client_socket.recv(1024)
17         if not chunk:
18             break
19         response += chunk
20
21     # Close the connection
22     client_socket.close()
23
24     # Split the response into headers and body
25     header_end = response.find(b'\r\n\r\n')
26     if header_end != -1:
27         header = response[:header_end].decode()
28         body = response[header_end + 4:]
29     else:
30         header = response.decode()
31         body = b''
32
33     # Print the headers and the body
34     print(header)
35     if body:
36         print(body.decode())
37
38 if len(sys.argv) != 4:
39     print(f"Usage: {sys.argv[0]} <serverhost> <serverport> <filerequested>")
40     sys.exit(1)
41 server_hostname = sys.argv[1] # Server hostname
42 server_port = int(sys.argv[2]) # Server port
43 file_requested = sys.argv[3] # File requested
44
45 # Request the file from the server and print its contents to the terminal
46 request_file(server_hostname, server_port, file_requested)
47
```

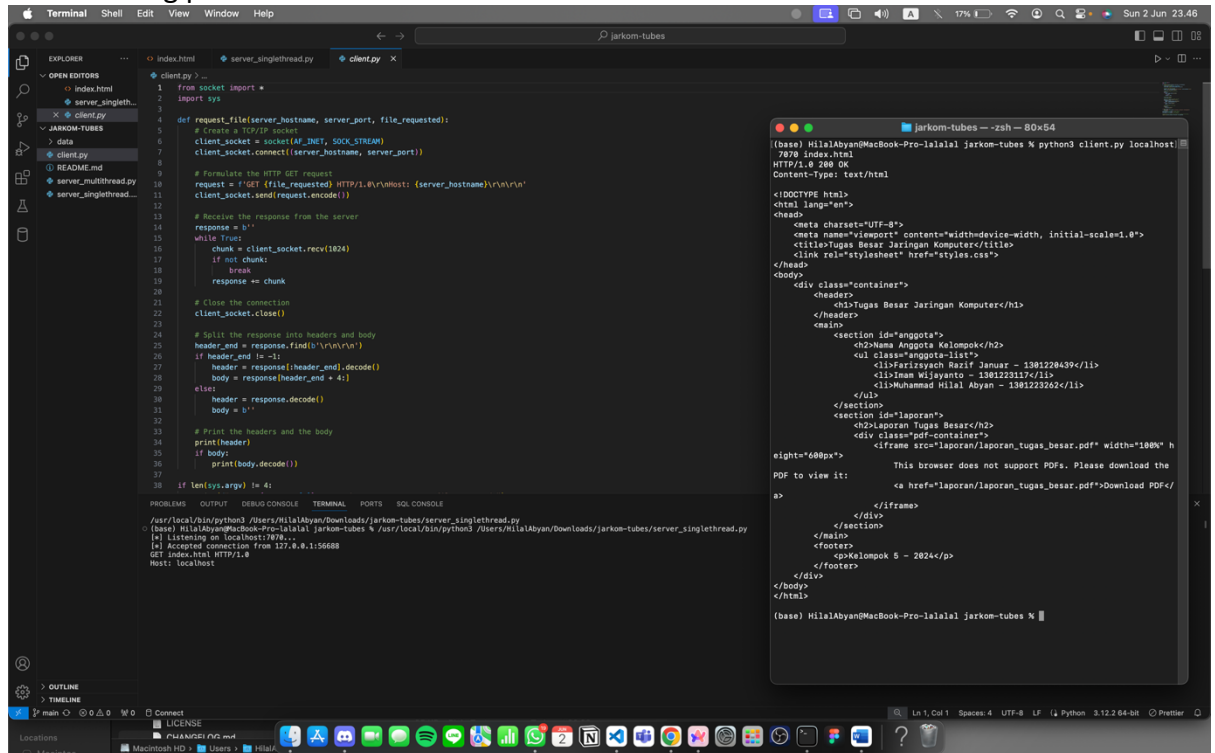

4. Jalankan server single threading, hasil running pada localhost:7070



5. Jalankan server multithreading, hasil running pada localhost:6969



6. Hasil running pada client



The image shows a VS Code editor with a Python client script and its output in a terminal window.

client.py

```
1 from socket import *
2 import sys
3
4 def request_file(server_hostname, server_port, file_requested):
5     # Create a TCP/IP socket
6     client_socket = socket(AF_INET, SOCK_STREAM)
7     client_socket.connect((server_hostname, server_port))
8
9     # Formulate the HTTP GET request
10    request = f'GET {file_requested} HTTP/1.0\r\nHost: {server_hostname}\r\n\r\n'
11    client_socket.send(request.encode())
12
13    # Receive the response from the server
14    response = b''
15    while True:
16        chunk = client_socket.recv(1024)
17        if not chunk:
18            break
19        response += chunk
20
21    # Close the connection
22    client_socket.close()
23
24    # Split the response into headers and body
25    header_end = response.find(b'\r\n\r\n')
26    if header_end != -1:
27        header = response[:header_end].decode()
28        body = response[header_end + 4:].decode()
29    else:
30        header = response.decode()
31        body = b''
32
33    # Print the headers and the body
34    print(header)
35    if body:
36        print(body.decode())
37
38 if len(sys.argv) != 4:
```

Terminal Output:

```
(base) HilaAbyan@MacBook-Pro-lalalal jarkon-tubes % python3 client.py localhost/7070 index.html
HTTP/1.0 200 OK
Content-Type: text/html
<html lang=en>
<head>
  <meta charset=UTF-8>
  <meta name=viewport content=device-width, initial-scale=1.0>
  <title>Tugas Besar Jaringan Komputer</title>
  <link rel=stylesheet href=styles.css>
</head>
<body>
  <div class=container>
    <header>
      <h2>Tugas Besar Jaringan Komputer</h2>
    </header>
    <main>
      <section id=anggota>
        <h3>Nama Anggota</h3>
        <ul class=anggota-list>
          <li>Farizyach Bariz Januar - 1301220439</li>
          <li>Iman Wijayanto - 1301223117</li>
          <li>Muhammad Hilal Abyan - 1301223262</li>
        </ul>
      </section>
      <section id=laporan>
        <h3>Laporan Tugas Besar</h3>
        <div class=pdf-container>
          <iframe src=laporan/laporan_tugas_besar.pdf width=100% height=600px>
            This browser does not support PDFs. Please download the PDF to view it:
            <a href=laporan/laporan_tugas_besar.pdf>Download PDF</a>
          </iframe>
        </div>
      </section>
    </main>
    <footer>
      <p>Kelompok 5 - 2024</p>
    </footer>
  </div>
</body>
</html>
```

DAFTAR PUSTAKA

Adya. (2023, September 22). Apa itu Transmission Control Protocol (TCP)? *Tutorial Digital Marketing, Website, & Bisnis Online - Exabytes*.

<https://www.exabytes.co.id/blog/transmission-control-protocol/>

Supriyanto, A. (2005). *Model Pengujian Komunikasi Socket dengan Protokol TCP/IP*. Neliti.

<https://www.neliti.com/publications/244266/model-pengujian-komunikasi-socket-dengan-protokol-tcpip>