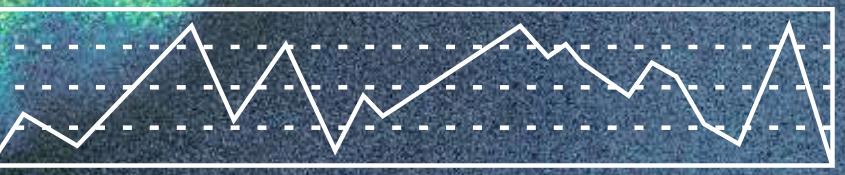
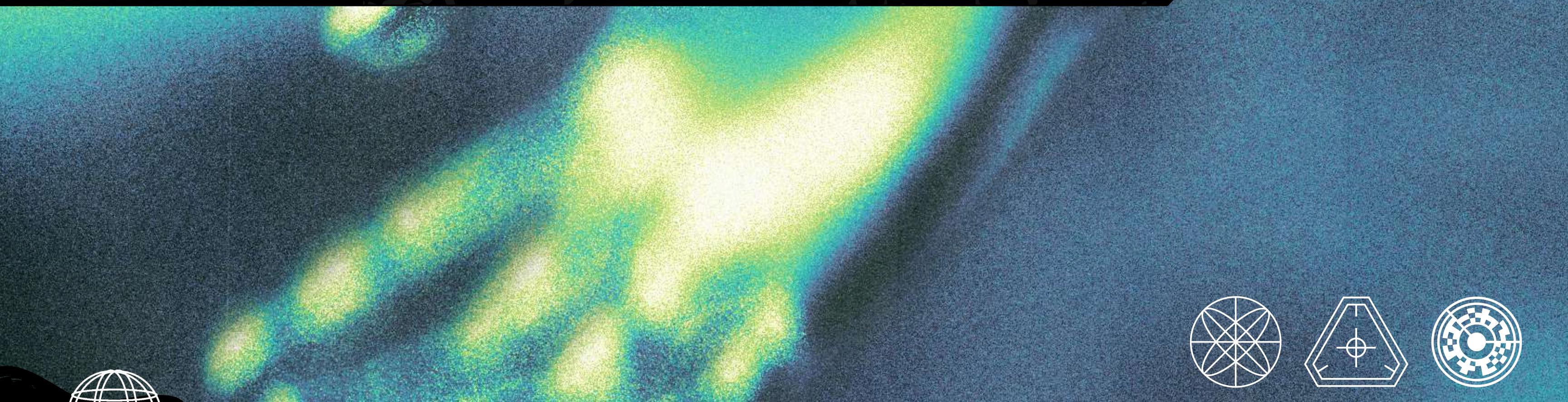


# JARINGAN KOMPUTER



IT4705



Individu



KELOMPOK 1

# SOPA JELOW'S MEMBERS



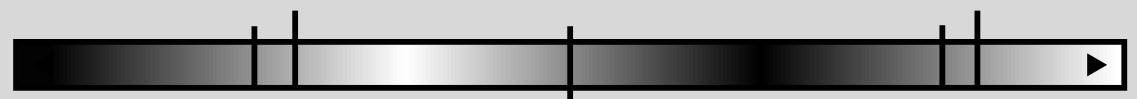
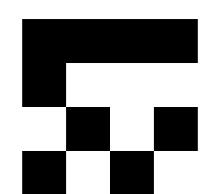
FLORENSIUS H.  
103032330113



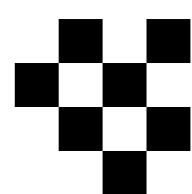
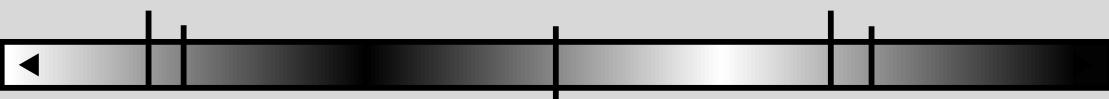
SOFWAN R.  
103032330045



G. AJENG  
103032300087

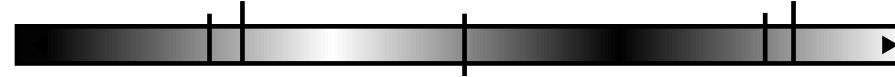


LOGGING IN



# AGENDA

TOPIK BAHASAN



---

Latar Belakang & Tujuan

---

Perancangan dan Implementasi

---

Pengujian dan Hasil (+Kode Program)

---



# LATAR BELAKANG

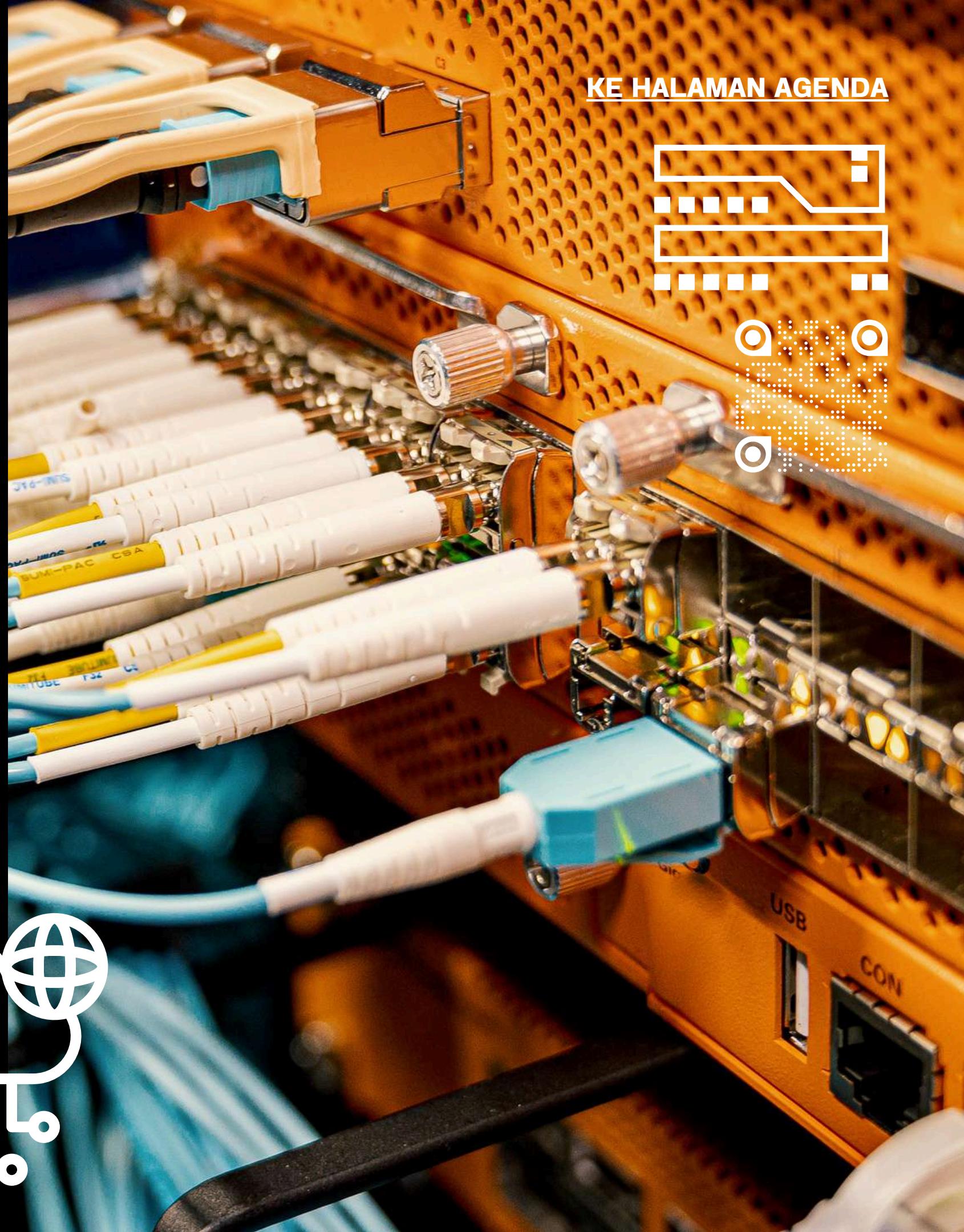
Server web adalah program yang melayani permintaan klien lewat protokol HTTP. Jika file yang diminta ada, server mengirimkannya; jika tidak, dikirimkan pesan "404 Not Found". Server yang hanya memproses satu permintaan dalam satu waktu akan lambat saat banyak klien mengakses. Oleh karena itu, digunakan server web multithread, yang memproses permintaan secara bersamaan dengan thread terpisah agar tetap cepat dan responsif.

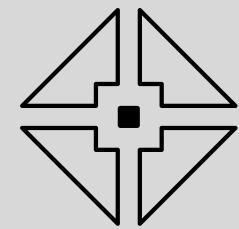
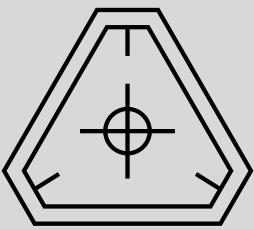
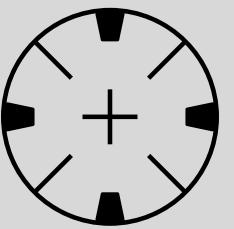


# TUJUAN

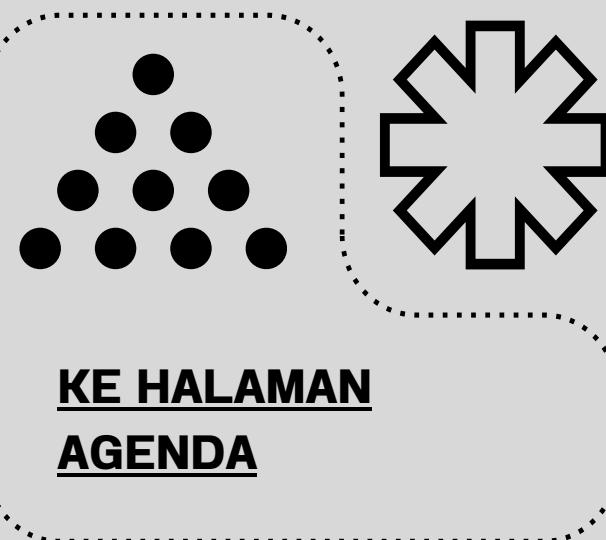
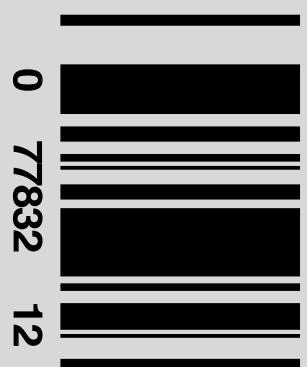
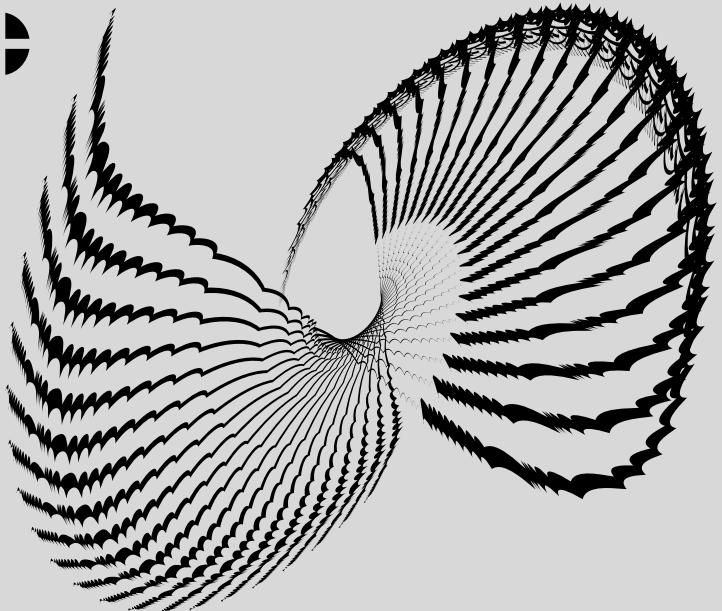
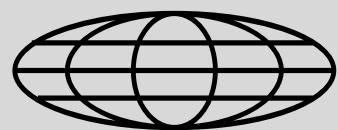
Untuk memenuhi tugas besar mata kuliah Jaringan Komputer, dengan membuat server web sederhana yang dapat:

- Menerima dan memproses request HTTP dari klien.
- Mengirimkan file yang diminta oleh klien.
- Menampilkan pesan "404 Not Found" jika file tidak ditemukan.
- Menggunakan multithreading agar server bisa melayani banyak klien secara bersamaan.





00004512 // 99374128 // 1005934



KE HALAMAN  
AGENDA

\*\*\*\*\*

# PERANCANGAN & IMPLEMENTASI

**CLIENT**

# TCP\_CLIENT.PY

```
import socket          # Untuk membuat koneksi jaringan (menggunakan TCP socket)
import threading       # Untuk membuat thread agar beberapa client bisa berjalan bersamaan
import time            # Untuk memberi jeda (delay) antar client saat mengirim request
import sys             # Untuk membaca argumen dari command-line

JUMLAH_CLIENT = 1      # Jumlah client yang ingin dijalankan (bisa diubah sesuai kebutuhan)

def kirim_request(nomor, host, port, filename):
    # Memberi jeda 5 detik antara masing-masing client berdasarkan urutan nomor
    time.sleep(5 * (nomor - 1))

    # Membuat socket TCP client
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client:
        # Menghubungkan socket client ke server berdasarkan host dan port
        client.connect((host, port))

        # Menyusun HTTP GET request standar sesuai format HTTP/1.1
        request = f"GET /{filename} HTTP/1.1\r\nHost: {host}\r\n\r\n"
        client.sendall(request.encode()) # Mengirim request ke server dalam format byte

    response = b"" # Buffer untuk menyimpan seluruh data respons dari server
    while True:
        data = client.recv(1024) # Menerima data sebanyak 1024 byte per iterasi
        if not data:           # Jika tidak ada data lagi diterima, keluar dari loop
            break
        response += data       # Menambahkan data ke buffer respons

    # Menampilkan hasil respons dari server ke terminal
    print(f"\n--- Response dari koneksi #{nomor} ---")
    print(response.decode(errors='ignore')) # Menampilkan data sebagai teks (mengabaikan karakter error)
```

Kode client ini berfungsi untuk membuat koneksi TCP ke server dan mengirim permintaan HTTP GET untuk sebuah file tertentu. Informasi host, port, dan nama file diambil dari argumen command-line. Fungsi kirim\_request dijalankan dalam thread dan memberi jeda antar client berdasarkan urutan agar tidak mengirim secara bersamaan. Setelah koneksi berhasil, client membangun request HTTP, mengirimkannya, lalu membaca dan menampilkan respons dari server. Respons diterima dalam blok 1024 byte hingga selesai, kemudian ditampilkan di terminal. Program ini mendukung eksekusi multiclient secara paralel menggunakan threading.

# TCP\_CLIENT.PY

```
if __name__ == '__main__':
    # Memastikan jumlah argumen yang dimasukkan dari command-line adalah 3 (host, port, filename)
    if len(sys.argv) != 4:
        print("Penggunaan: python TCP_Client.py <host> <port> <filename>")
        sys.exit(1) # Keluar dari program jika argumen tidak sesuai

    # Mengambil argumen dari command-line
    host_input = sys.argv[1] # Argumen ke-1: alamat host (contoh: "127.0.0.1")
    port_input = int(sys.argv[2]) # Argumen ke-2: port server (contoh: 5880)
    filename_input = sys.argv[3] # Argumen ke-3: nama file yang ingin diminta (contoh: "header.html")

    threads = [] # List untuk menyimpan semua thread client

    # Membuat dan menjalankan thread sebanyak JUMLAH_CLIENT
    for i in range(JUMLAH_CLIENT):
        # Membuat thread untuk menjalankan fungsi kirim_request dengan argumen yang sesuai
        t = threading.Thread(target=kirim_request, args=(i+1, host_input, port_input, filename_input))
        threads.append(t) # Menyimpan thread ke list
        t.start() # Menjalankan thread (client mulai mengirim request)

    # Menunggu semua thread selesai (supaya program tidak keluar sebelum semua client selesai)
    for t in threads:
        t.join()
```

Program memastikan bahwa pengguna memberikan tiga argumen command-line dengan format <host> <port> <filename>. Jika tidak, program akan berhenti dan menampilkan cara penggunaan yang benar. Setelah itu, program mengambil argumen yang diberikan dan menyimpannya ke dalam variabel host\_input, port\_input, dan filename\_input. Selanjutnya, program membuat thread sebanyak JUMLAH\_CLIENT. Setiap thread menjalankan fungsi kirim\_request() secara paralel agar beberapa client dapat mengirim request HTTP ke server secara bersamaan. Semua thread disimpan ke dalam list dan dijalankan satu per satu. Terakhir, join() digunakan agar program menunggu semua client selesai sebelum keluar. Ini memastikan bahwa semua respons dari server diproses dengan benar.

**SINGLE-THREAD**

# TCP\_SERVER.PY

```
import socket          # Mengimpor modul socket untuk membuat koneksi TCP
import os              # Mengimpor modul os untuk memeriksa dan membaca file
import time            # Mengimpor modul time untuk memberi delay dan mengukur waktu

HOST = '0.0.0.0'        # Server akan menerima koneksi dari semua IP
PORT = 5880             # Port yang digunakan server
```

Kode ini mengimpor tiga modul: socket untuk membuat koneksi jaringan (TCP atau UDP), os untuk berinteraksi dengan sistem file seperti membaca atau memeriksa keberadaan file, dan time untuk memberikan jeda (delay) atau mengukur waktu eksekusi. Variabel HOST diset ke '0.0.0.0', yang berarti server akan menerima koneksi dari semua alamat IP yang mencoba terhubung. Variabel PORT diset ke 5880, yang merupakan port yang digunakan server untuk mendengarkan koneksi masuk dari client.

# TCP\_SERVER.PY

```
# Fungsi untuk menangani request dari client
def handle_request(conn, addr):
    print(f"[STATUS] Sedang melayani klien dari {addr}...")
    start_time = time.time() # Mencatat waktu mulai pelayanan

    try:
        request = conn.recv(1024).decode() # Menerima dan mendekode request dari client
        print("Request:")
        print(request)

        lines = request.split('\r\n') # Memecah request menjadi baris-baris
        if not lines:
            return

        request_line = lines[0] # Jika kosong, keluar
        parts = request_line.split() # Baris pertama adalah request line
        if len(parts) < 2: # Memisahkan method dan path
            return # Jika format tidak valid, keluar

        method, path = parts[0], parts[1] # Mendapatkan metode dan path
        filename = path.lstrip('/') # Menghapus '/' di awal path

        if method != 'GET': # Hanya mendukung GET
            time.sleep(10) # Simulasi delay
            response = "HTTP/1.1 405 Method Not Allowed\r\n\r\n"
            conn.sendall(response.encode()) # Kirim response 405
            return

        if os.path.isfile(filename): # Jika file ada
            with open(filename, 'rb') as f:
                content = f.read() # Baca konten file
            time.sleep(10) # Delay sebelum kirim
            header = "HTTP/1.1 200 OK\r\n"
            header += f"Content-Length: {len(content)}\r\n"
            header += "Content-Type: text/html\r\n\r\n"
            conn.sendall(header.encode() + content) # Kirim header + konten
        else:
            time.sleep(10) # Delay sebelum response
            response = "HTTP/1.1 404 Not Found\r\n\r\nFile not found"
            conn.sendall(response.encode()) # Kirim response 404

    finally:
        conn.close() # Tutup koneksi
        end_time = time.time() # Catat waktu selesai
        elapsed = end_time - start_time # Hitung durasi
        print(f"[STATUS] Selesai melayani klien dari {addr}. Waktu pelayanan: {elapsed:.4f} detik\r\n")
```

Fungsi `handle_request(conn, addr)` mencatat waktu mulai layanan, menerima request dari client melalui `conn.recv()`, lalu memecah request menjadi baris-baris untuk mendapatkan method (seperti GET) dan path file yang diminta. Jika method bukan GET, server merespons dengan kode HTTP 405 (Method Not Allowed) setelah delay 10 detik. Jika method adalah GET, server memeriksa apakah file yang diminta ada di direktori. Jika file ditemukan, server membaca isinya, menyiapkan response 200 (OK), dan mengirimkan konten file tersebut ke client. Jika file tidak ditemukan, server mengirim response 404 (Not Found). Delay 10 detik disisipkan sebelum mengirimkan response, yang kemungkinan dimaksudkan untuk simulasi atau pengujian kestabilan client-server. Setelah semua proses selesai, koneksi ditutup dan waktu pelayanan dicetak ke terminal.

# TCP\_SERVER.PY

```
# Fungsi menjalankan server secara single-threaded
def run_single_threaded():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
        server.bind((HOST, PORT))                  # Bind socket ke host dan port
        server.listen(1)                          # Menunggu maksimal 1 koneksi
        print(f"[MODE: SINGLE THREAD] Server listening on port {PORT}...\n")

    while True:
        conn, addr = server.accept()          # Terima koneksi dari client
        print(f"Connected by {addr}")
        handle_request(conn, addr)           # Tangani permintaan secara langsung (blocking)
```

Fungsi `run_single_threaded()` yang menjalankan server dalam mode single-threaded, artinya server hanya dapat menangani satu client pada satu waktu. Di dalam blok `with`, socket dibuat menggunakan `socket.AF_INET` (untuk IPv4) dan `socket.SOCK_STREAM` (untuk koneksi TCP). Socket tersebut kemudian di-bind ke alamat HOST dan PORT, dan mendengarkan koneksi masuk dengan antrean maksimal satu koneksi (`listen(1)`). Dalam loop `while True`, server akan terus-menerus menerima koneksi client menggunakan `accept()`, menampilkan alamat client yang terhubung, dan langsung memproses permintaan dengan memanggil fungsi `handle_request(conn, addr)`. Karena mode ini single-threaded, koneksi berikutnya harus menunggu sampai permintaan sebelumnya selesai ditangani (bersifat blocking)

# TCP\_SERVER.PY

```
# Entry point program
if __name__ == '__main__':
    run_single_threaded()                                # Jalankan server
```

Blok `if __name__ == '__main__':` berfungsi untuk memastikan bahwa fungsi `run_single_threaded()` hanya akan dipanggil jika file Python ini dijalankan secara langsung (bukan diimpor sebagai modul oleh file lain). Ketika skrip dijalankan, server akan dimulai dalam mode single-threaded dengan memanggil `run_single_threaded()`, dan server pun siap menerima serta menangani koneksi dari client satu per satu secara berurutan. Baris ini menandai titik awal eksekusi program.

**MULTI-THREAD**

# TCP\_MULTITHREAD.PY

```
import socket          # Mengimpor modul socket untuk membuat dan mengelola koneksi TCP/IP
import os              # Mengimpor modul os untuk memeriksa dan membaca file di sistem
import threading       # Mengimpor modul threading untuk memungkinkan multiple client ditangani secara bersamaan
import time            # Mengimpor modul time untuk mencatat waktu dan memberikan delay simulasi

HOST = '0.0.0.0'        # Server akan menerima koneksi dari semua IP (bind ke semua interface jaringan)
PORT = 5880             # Port yang digunakan server untuk menerima koneksi
```

Implementasi server multi-threaded ditandai dengan tambahan impor modul threading. Tujuan penggunaan threading adalah untuk memungkinkan server menangani beberapa client secara bersamaan dengan membuat thread terpisah untuk setiap koneksi yang masuk. Modul lain seperti socket, os, dan time tetap digunakan sebagaimana pada versi single-threaded sebelumnya – untuk membuat koneksi jaringan, membaca file dari sistem, dan melakukan delay simulasi atau pencatatan waktu.

Variabel HOST dan PORT tetap didefinisikan sama seperti sebelumnya, yaitu HOST = '0.0.0.0' agar menerima koneksi dari semua alamat IP, dan PORT = 5880 sebagai port tempat server mendengarkan koneksi masuk.

# TCP\_MULTITHREAD.PY

```
def handle_request(conn, addr):
    print(f"[STATUS] Sedang melayani klien dari {addr}...") # Log saat mulai menangani klien
    start_time = time.time() # Mencatat waktu mulai pelayanan

    try:
        request = conn.recv(1024).decode() # Menerima hingga 1024 byte data dari client, kemudian decode dari byte ke string
        print("Request:")
        print(request)

        lines = request.split('\r\n')
        if not lines:
            return

        request_line = lines[0]
        parts = request_line.split()
        if len(parts) < 2:
            return

        method, path = parts[0], parts[1]
        filename = path.strip('/')

        if method != 'GET':
            time.sleep(10)
            response = "HTTP/1.1 405 Method Not Allowed\r\n\r\n" # Respons HTTP 405
            conn.sendall(response.encode())
            return

        if os.path.isfile(filename):
            with open(filename, 'rb') as f:
                content = f.read()
            time.sleep(10)
            header = "HTTP/1.1 200 OK\r\n"
            header += f"Content-Length: {len(content)}\r\n" # Tambahkan panjang konten
            header += "Content-Type: text/html\r\n\r\n" # Tambahkan jenis konten
            conn.sendall(header.encode() + content) # Kirim header + isi file ke client
        else:
            time.sleep(10)
            response = "HTTP/1.1 404 Not Found\r\n\r\nFile not found" # Respons jika file tidak ditemukan
            conn.sendall(response.encode()) # Kirim respons 404 ke client

    finally:
        conn.close()
        end_time = time.time()
        elapsed = end_time - start_time
        print(f"[STATUS] Selesai melayani klien dari {addr}. Waktu pelayanan: {elapsed:.4f} detik\r\n")
```

Fungsi `handle_request(conn, addr)` pada kode di atas bertugas menangani setiap permintaan dari client dengan mencatat waktu mulai pelayanan, membaca data request, mem-parsing HTTP request, dan merespons sesuai kondisi. Jika metode bukan GET, server merespons dengan kode HTTP 405. Jika file yang diminta ada, konten file dibaca dan dikirim bersama header HTTP 200, sedangkan jika file tidak ditemukan, dikirimkan respons HTTP 404. Fungsi ini juga menyimulasikan delay pada beberapa tahap menggunakan `time.sleep(10)` untuk menggambarkan waktu proses. Setelah selesai, koneksi ditutup dan durasi pelayanan dicetak ke terminal. Fungsi ini dapat digunakan baik pada server single-threaded maupun multi-threaded.

# TCP\_MULTITHREAD.PY

```
# Fungsi utama untuk menjalankan server dalam mode multi-threaded
def run_multi_threaded():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server: # Membuat socket TCP
        server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) # Mengizinkan reuse port (hindari error bind saat restart)
        server.bind((HOST, PORT)) # Bind socket ke host dan port yang ditentukan
        server.listen() # Mulai mendengarkan koneksi masuk
        print(f"[MODE: MULTI THREAD] Server listening on port {PORT}...\n")

    while True:
        conn, addr = server.accept() # Menerima koneksi baru dari client
        thread = threading.Thread(target=handle_request, args=(conn, addr)) # Buat thread baru untuk menangani client
        thread.start() # Jalankan thread
```

Fungsi `run_multi_threaded()` adalah fungsi utama yang menjalankan server dalam mode multi-threaded, memungkinkan server untuk menangani banyak client secara bersamaan. Fungsi ini membuat socket TCP, mengatur opsi agar port bisa digunakan ulang (SO\_REUSEADDR), lalu bind ke alamat host dan port yang telah ditentukan, dan mulai mendengarkan koneksi masuk. Dalam loop utama, setiap koneksi baru yang diterima akan langsung ditangani oleh thread baru menggunakan modul `threading`, dengan `handle_request` sebagai fungsi target. Dengan demikian, server tetap responsif dan dapat melayani beberapa klien secara paralel tanpa menunggu permintaan sebelumnya selesai.

# TCP\_MULTITHREAD.PY

```
# Entry point program (dijalankan saat file ini dieksekusi langsung)
if __name__ == '__main__':
    run_multi_threaded()                      # Jalankan fungsi server multi-threaded
```

Bagian yang akan dijalankan ketika file Python dieksekusi secara langsung (bukan diimpor sebagai modul). Pada bagian ini, fungsi `run_multi_threaded()` dipanggil, sehingga server akan dijalankan dalam mode multi-threaded. Dengan pendekatan ini, setiap koneksi klien akan ditangani oleh thread terpisah, memungkinkan server melayani beberapa permintaan secara paralel dan meningkatkan efisiensi serta responsivitas dibandingkan mode single-threaded.

HTML

```
        }
    </style>
</head>
<body>
<header>
    <h1>WEBPAGE TESTING TUBES JARINGAN KOMPUTER</h1>
    <p>Kami adalah mahasiswa semester 4 kelas IT4705 yang sedang mengerjakan tuga
</header>
<div class="welcome-text">Members of SOPAJELO Squad</div>
<div class="top-border"></div>
<section class="members">
    <article class="member">
        
        <h2>Sofwan Rosidi</h2>
        <p>103032330045</p>
    </article>
    <article class="member">
        
        <h2>Florensius Hutagalung</h2>
        <p>103032330113</p>
    </article>
    <article class="member">
        
        <h2>Galuh Ajeng</h2>
        <p>103032300087</p>
    </article>
</section>
</body>
</html>
```

**WEBPAGE TESTING TUBES JARINGAN  
KOMPUTER**

Kami adalah mahasiswa semester 4 kelas IT4705 yang sedang mengerjakan tugas besar jaringan komputer (tubes jarkom).

Members of SOPAJELO Squad

---



**Sofwan Rosidi**  
103032330045



**Florensius  
Hutagalung**  
103032330113



**Galuh Ajeng**  
103032300087

# PENGUJIAN DAN HASIL



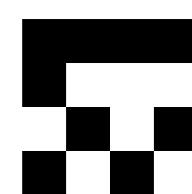
TCP\_CLIENT.PY



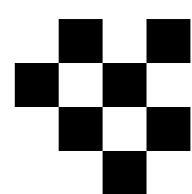
TCP\_SERVER.PY



TCP\_MULTITHREAD.  
PY



[KE HALAMAN AGENDA](#)



# Single Thread

## Response Server

```
C:\Windows\System32\cmd.e x + v
Microsoft Windows [Version 10.0.26100.4061]
(c) Microsoft Corporation. All rights reserved.

C:\Kuliah\Semester 4\Jaringan Komputer\Tugas Besar Jarkom>py TCP_Server.py
[MODE: SINGLE THREAD] Server listening on port 5880...

Connected by ('127.0.0.1', 52389)
[STATUS] Sedang melayani klien dari ('127.0.0.1', 52389)...
Request:
GET /header.html HTTP/1.1
Host: 127.0.0.1
```

## Response Client

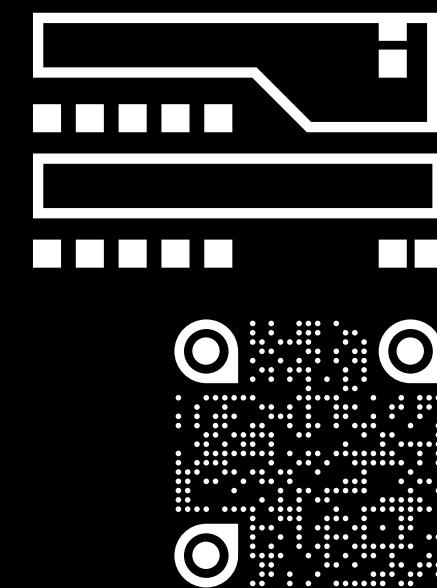
```
C:\Kuliah\Semester 4\Jaringan Komputer\Tugas Besar Jarkom>py TCP_Client.py 127.0.0.1 5880
Masukkan nama file yang ingin diminta: header.html

--- Response dari koneksi #1 ---
HTTP/1.1 200 OK
Content-Length: 3740
Content-Type: text/html
```

# HASIL SINGLE-THREAD

Mode Single-threaded (TCP\_Server.py):

- Klien dilayani satu per satu secara berurutan.
- Setiap klien harus menunggu klien sebelumnya selesai sepenuhnya (~10 detik per klien).
- Total waktu untuk 5 klien: ~50 detik.
- Tidak ada paralelisme, semua proses blocking.



# Multi Thread

## Response Server

```
C:\Windows\System32\cmd.e X + - □  
Microsoft Windows [Version 10.0.26100.4061]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Kuliah\Semester 4\Jaringan Komputer\Tugas Besar Jarkom>py TCP  
_Multithread.py  
[MODE: MULTI THREAD] Server listening on port 5880...  
  
[STATUS] Sedang melayani klien dari ('127.0.0.1', 52719)...  
Request:  
GET /header.html HTTP/1.1  
Host: 127.0.0.1  
  
[STATUS] Sedang melayani klien dari ('127.0.0.1', 52720)...  
Request:  
GET /header.html HTTP/1.1  
Host: 127.0.0.1
```

## Response Client1

```
C:\Kuliah\Semester 4\Jaringan Komputer\Tugas Besar Jarkom>py TCP  
_Client.py 127.0.0.1 5880 header.html  
  
--- Response dari koneksi #1 ---  
HTTP/1.1 200 OK  
Content-Length: 3740  
Content-Type: text/html
```

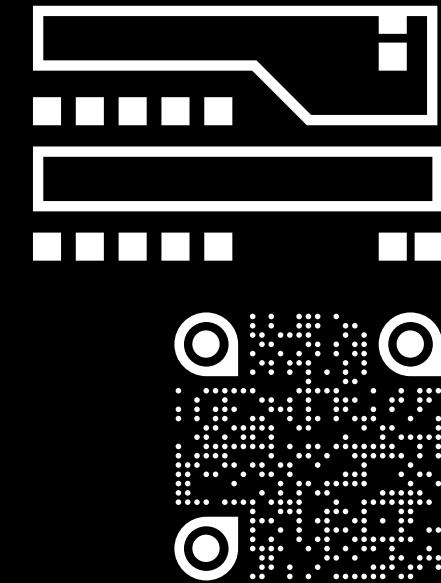
## Response Client2

```
C:\Kuliah\Semester 4\Jaringan Komputer\Tugas Besar Jarkom>py TCP_Client.py 127  
  
--- Response dari koneksi #1 ---  
HTTP/1.1 200 OK  
Content-Length: 3740  
Content-Type: text/html
```

# HASIL MULTI-THREAD

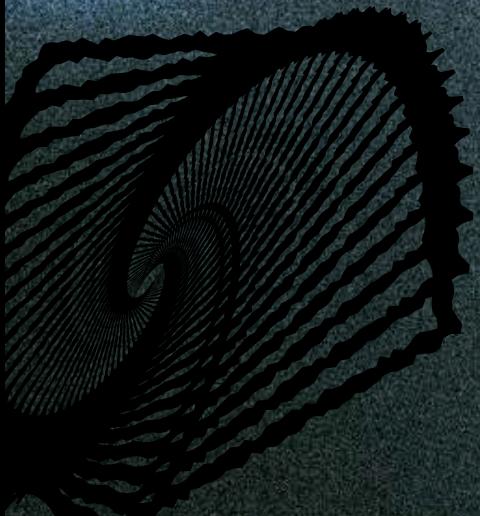
Mode Multi-threaded (TCP\_Multithread.py):

- Klien dilayani secara paralel, masing-masing dalam thread terpisah.
- Semua klien dapat mengakses file dalam waktu hampir bersamaan.
- Total waktu untuk 5 klien: hanya sedikit lebih dari 10 detik.
- Thread bekerja efisien menangani delay dan respons tanpa blocking klien lain.

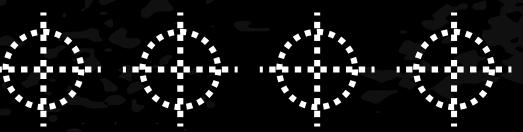




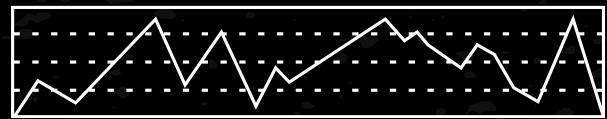
# TERIMA HASIH!



http://www.sopajelo.com



SOPAJELO -



0628199501202016