LAPORAN TUGAS BESAR

JARINGAN KOMPUTER

KELAS : IT-47-05
TAHUN AKADEMIK : GENAP 2024/2025

Oleh:

ID KELOMPOK : 1
ANGGOTA MAHASISWA : 1. Florensius Hutagalung (103032330113)
2. Sofwan Rosidi (103032330045)
3. Galuh Ajeng (103032300087)

PRODI S1 TEKNOLOGI INFORMASI

FAKULTAS INFORMATIKA

UNIVERSITAS TELKOM

2024/2025

DAFTAR ISI

DAFTAR ISI i
BAB 1 1
PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Tujuan 1
BAB 2 1
LANDASAN TEORI 2
2.1 Jaringan Komputer dan TCP/IP 2
2.2 Socket Programming 2
2.3 Multithreading 2
BAB 3 2
PERANCANGAN DAN IMPLEMENTASI 2
3.1 Spesifikasi Sistem 3
3.2 Cara Kerja Server Web 3
3.3 Implementasi Multithreading 3
BAB 4 10
4.1 Cara Pengujian 10
4.2 Hasil Pengujian 10
BAB 5 13
KESIMPULAN 13

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Server web adalah program yang melayani permintaan dari klien melalui protokol HTTP. Ketika klien meminta sebuah file, server akan mencari file tersebut di sistem file dan mengirimkannya kembali sebagai respons. Jika file tidak ditemukan, server akan mengirimkan pesan "404 Not Found".

Server yang hanya bisa menangani satu permintaan dalam satu waktu akan menjadi lambat jika banyak klien yang mengakses secara bersamaan. Oleh karena itu, diperlukan server web multithread yang dapat memproses beberapa permintaan secara bersamaan menggunakan thread. Dengan cara ini, setiap permintaan klien akan ditangani oleh thread terpisah sehingga server tetap responsif dan cepat.

1.2 Tujuan

Tujuan dari tugas ini adalah untuk memenuhi tugas besar mata kuliah Jaringan Komputer, dengan membuat server web sederhana yang dapat:

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

BAB 2

LANDASAN TEORI

2.1 Jaringan Komputer dan TCP/IP

Jaringan komputer adalah sekumpulan komputer yang saling terhubung dan dapat saling berkomunikasi untuk bertukar data. Dalam komunikasi jaringan, protokol TCP/IP (Transmission Control Protocol/Internet Protocol) digunakan sebagai standar agar perangkat dapat saling bertukar informasi secara andal. TCP memastikan data dikirim secara utuh dan berurutan, sedangkan IP menangani pengalamatan dan pengiriman data ke tujuan yang tepat. Dalam konteks server web, TCP/IP memungkinkan klien dan server berkomunikasi melalui koneksi yang stabil dan terpercaya.

2.2 Socket Programming

Socket adalah antarmuka pemrograman yang memungkinkan komunikasi antara dua perangkat melalui jaringan. Socket programming digunakan untuk membuat aplikasi yang dapat mengirim dan menerima data melalui jaringan, seperti server dan klien. Pada server web, socket digunakan untuk menerima koneksi dari klien, membaca permintaan HTTP, dan mengirimkan respons. Socket yang kami gunakan dibuat menggunakan bahasa pemrograman Python dan menjadi dasar utama dalam pengembangan aplikasi jaringan kami.

2.3 Multithreading

Multithreading adalah teknik pemrograman yang memungkinkan suatu program menjalankan beberapa thread (aliran eksekusi) secara bersamaan. Dalam konteks server, multithreading digunakan agar server dapat menangani lebih dari satu permintaan klien pada waktu yang sama. Dengan demikian, performa server meningkat dan tidak harus menunggu satu klien selesai sebelum melayani klien lainnya. Setiap thread dapat menangani satu koneksi klien, membuat proses menjadi lebih efisien dan responsif.

BAB 3
PERANCANGAN DAN IMPLEMENTASI

3.1 Spesifikasi Sistem

Server web ini dibangun menggunakan bahasa pemrograman Python dan modul socket untuk komunikasi jaringan serta threading untuk mendukung eksekusi multithreaded. Server dirancang untuk:

a. Menerima koneksi TCP dari klien.
b. Menerima dan mem-parsing request HTTP, khususnya GET.
c. Mengembalikan file yang diminta jika tersedia di sistem file.
d. Mengirimkan pesan HTTP 404 Not Found jika file tidak ditemukan.
e. Menyediakan dua mode operasi: single-threaded (satu klien pada satu waktu) dan multi-threaded (melayani banyak klien secara simultan).

Spesifikasi teknis:

a. Bahasa: Python
b. Port Server: 5880
c. Alamat IP: 0.0.0.0 (menerima dari semua alamat)
d. Respons HTTP: 200 OK, 404 Not Found, dan 405 Method Not Allowed
e. Pendukung File: File HTML atau lainnya dari sistem file lokal

3.2 Cara Kerja Server Web

Untuk memahami bagaimana server web ini melayani permintaan dari klien, berikut adalah tahapan-tahapan utama dalam proses kerjanya:

a. Mendengarkan Koneksi:
   Server membuka socket TCP dan menunggu koneksi masuk di port 5880.
b. Menerima Request:
   Ketika ada klien yang terhubung, server membaca data request HTTP dari klien. Umumnya berbentuk:
   - GET /nama_file.html HTTP/1.1
   - Host: localhost
c. Parsing Request:
   Baris pertama request akan dipecah untuk mendapatkan metode HTTP (GET) dan path file yang diminta.
d. Cek dan Kirim File:
   Jika file ditemukan di direktori lokal, server membaca file dan membangun respons HTTP 200 OK, termasuk header Content-Length dan Content-Type. Jika file tidak ditemukan, server membalas dengan respons HTTP 404 Not Found.
e. Response Dikirim ke Klien:
   Header dan konten file dikirim secara langsung ke klien melalui koneksi TCP yang sama.
f. Penanganan Error:
   Jika metode HTTP bukan GET, server merespons dengan 405 Method Not Allowed.
g. Menutup Koneksi:
   Setelah file dikirim (atau error dilaporkan), koneksi TCP ditutup.

3.3 Implementasi Multithreading

Pada mode multithreaded, server menggunakan modul threading untuk melayani beberapa permintaan klien secara paralel:

a. Setelah menerima koneksi dari klien, server membuat thread baru untuk menangani permintaan tersebut.
b. Fungsi handle_request(conn, addr) dipanggil dalam thread tersebut, memungkinkan server kembali mendengarkan koneksi lain tanpa menunggu proses klien sebelumnya selesai.
c. Hal ini memungkinkan server untuk:
   - Melayani banyak klien secara simultan.
   - Mengurangi waktu tunggu klien lainnya, khususnya saat terjadi delay (misalnya saat membaca file atau delay buatan seperti time.sleep(10)).
d. Implementasi multithreaded ditangani dalam fungsi run_multi_threaded(), dengan logika pemisahan thread sebagai berikut:

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

Dengan pendekatan ini, setiap klien akan dilayani oleh thread terpisah, dan tidak saling mengganggu walaupun salah satu proses klien berjalan lama.

3.4 Kode Single-Thread, Multi-thread, dan Client

A. Single-Thread

1. Library

```
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 server.py 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. Komentar di sebelah kanan menjelaskan fungsi dari masing-masing baris secara ringkas.

2. Fungsi Menangani request dari client

```python
# 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)                      # Menampilkan request di terminal

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

        request_line = lines[0]             # Baris pertama adalah request line
        parts = request_line.split()        # Memisahkan method dan path
        if len(parts) < 2:
            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\n")
```

Fungsi handle_request(conn, addr) yang bertugas menangani permintaan dari client pada server. Fungsi ini 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.

3. Fungsi menjalankan server secara single-threaded

```python
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).

4. Entry Point

```
# 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.

B. Multi-Thread

1. Library

```
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
```

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.

2. Fungsi untuk menangani permintaan client secara individual

```python
# Fungsi untuk menangani permintaan client secara individual
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)                      # Menampilkan isi request HTTP dari client

        lines = request.split('\r\n')       # Memisahkan request menjadi baris-baris
        if not lines:
            return                          # Jika request kosong, keluar dari fungsi

        request_line = lines[0]             # Baris pertama dari HTTP request, biasanya seperti: GET /index.html HTTP/1.1
        parts = request_line.split()        # Memisahkan baris menjadi bagian-bagian (method, path, versi)
        if len(parts) < 2:
            return                          # Jika tidak cukup bagian (method dan path), keluar

        method, path = parts[0], parts[1]   # Ekstraksi method (GET) dan path (misalnya /index.html)
        filename = path.lstrip('/')         # Menghapus tanda '/' di awal path untuk mendapatkan nama file

        if method != 'GET':                 # Jika bukan metode GET, tidak dilayani
            time.sleep(10)                  # Simulasi delay pelayanan
            response = "HTTP/1.1 405 Method Not Allowed\r\n\r\n"  # Respons HTTP 405
            conn.sendall(response.encode()) # Kirim respons ke client
            return

        if os.path.isfile(filename):        # Cek apakah file dengan nama tersebut ada
            with open(filename, 'rb') as f: # Buka file dalam mode biner
                content = f.read()          # Baca seluruh isi file
            time.sleep(10)                  # Simulasi delay pelayanan
            header = "HTTP/1.1 200 OK\r\n"  # Buat header HTTP 200 OK
            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)                  # Simulasi delay pelayanan
            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()                        # Tutup koneksi socket
        end_time = time.time()              # Catat waktu selesai pelayanan
        elapsed = end_time - start_time     # Hitung durasi pelayanan
        print(f"[STATUS] Selesai melayani klien dari {addr}. Waktu pelayanan: {elapsed:.4f} detik\n")
```

Fungsi handle_request(conn, addr) yang menangani request dari client. Fungsinya identik dengan versi single-threaded yang sebelumnya ditampilkan, dan ini menegaskan bahwa logika penanganan client tidak berubah antara mode single dan multi-threaded — hanya cara pemanggilan dan eksekusinya yang berbeda.

Didalam fungsi tersebut terdapat :

- Fungsi menerima koneksi (conn) dan alamat client (addr), lalu mencatat waktu mulai.
- Menerima data dari client, mem-parsing HTTP request (hanya mendukung method GET).
- Jika file diminta ditemukan, dibaca dan dikirimkan ke client disertai header HTTP 200.
- Jika file tidak ditemukan, kirimkan HTTP 404.
- Jika method bukan GET, kirim HTTP 405.
- Setiap langkah utama diberi time.sleep(10) untuk mensimulasikan delay layanan.
- Di akhir, koneksi ditutup dan waktu pelayanan dicetak.

Fungsi ini akan dipanggil dalam thread terpisah di mode multi-threaded, memungkinkan banyak client ditangani bersamaan tanpa harus menunggu client sebelumnya selesai.

3. Fungsi utama untuk menjalankan server dalam mode multi-threaded

```python
# 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.

4. Entry point program

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

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.

C. Client

1. Library

```python
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)
```

Program server ini memanfaatkan beberapa library Python untuk membangun dan mengelola koneksi jaringan secara efisien. Modul socket digunakan sebagai inti untuk membuat server TCP/IP yang dapat menerima dan merespons permintaan client. Modul os membantu memverifikasi keberadaan file yang diminta client sebelum dikirimkan. Untuk memungkinkan server melayani banyak client secara bersamaan, digunakan modul threading yang membuat thread baru untuk setiap koneksi. Sementara itu, modul time dimanfaatkan untuk mencatat durasi pelayanan dan memberikan simulasi delay. Gabungan keempat modul ini membentuk fondasi server sederhana namun fungsional, baik dalam mode single-threaded maupun multi-threaded.

2. Fungsi untuk mengirim request

```python
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.

3. Entry Point

```python
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.

BAB 4

PENGUJIAN DAN HASIL

4.1 Cara Pengujian

Pengujian dilakukan untuk membandingkan kinerja dan perilaku server web dalam dua mode operasi:

a. Single-threaded (file TCP_Server.py)
b. Multi-threaded (file TCP_Multithread.py)

Langkah-langkah pengujian:

a. Menyiapkan File Uji
   File index.html disiapkan pada direktori yang sama dengan server untuk memastikan respons 200 OK jika file diminta.
b. Menjalankan Server.
   - Untuk mode single thread: jalankan TCP_Server.py.
   - Untuk mode multi-thread: jalankan TCP_Multithread.py.
c. Menjalankan Klien
   - Jalankan TCP_Client.py yang sudah dikonfigurasi untuk mengirim 5 klien secara bertahap.
   - Setiap klien mengirim permintaan GET /index.html HTTP/1.1.
   - Terdapat delay 5 detik antar klien, dan server mensimulasikan delay 10 detik per request menggunakan time.sleep(10).
d. Pengamatan
   - Di sisi server: waktu pelayanan setiap klien ditampilkan.
   - Di sisi klien: respons HTTP ditampilkan, baik 200 OK (jika file ada) maupun 404 Not Found.

Uji dilakukan dua kali, masing-masing untuk mode single-thread dan multi-thread.

4.2 Hasil Pengujian

Hasil pengujian menunjukkan perbedaan signifikan antara mode single-thread dan multi-thread:

a. 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.

   Contoh output:

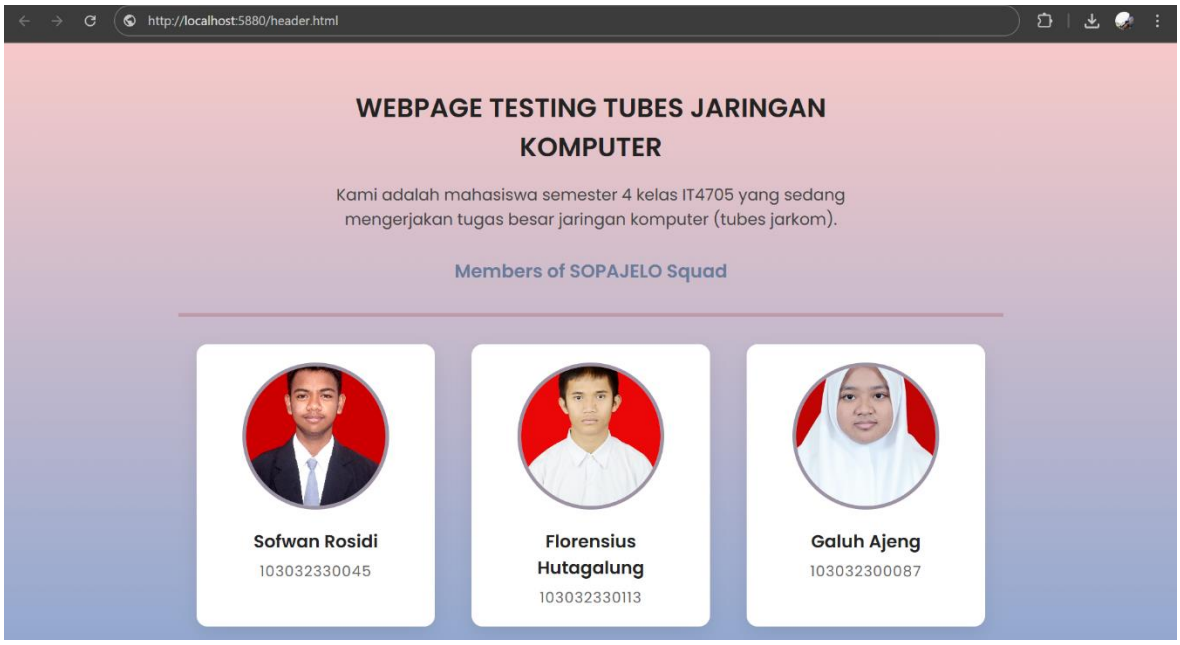
   - Response Client

```
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

<!DOCTYPE html>
<html lang="id">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Perkenalan Mahasiswa IT4705</title>
  <style>
    @import url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;600&display=swap');
    * {
      box-sizing: border-box;
    }
    body {
      margin: 0;
      font-family: 'Poppins', sans-serif;
      background: linear-gradient(180deg, #F7CAC9 0%, #91A8D0 100%);
      color: #333;
      display: flex;
      flex-direction: column;
      min-height: 100vh;
      align-items: center;
      padding: 30px 15px;
    }
    header {
      text-align: center;
      margin-bottom: 20px;
      max-width: 600px;
    }
```

- Response Server

```
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', 54063)
[STATUS] Sedang melayani klien dari ('127.0.0.1', 54063)...
Request:
GET /header.html HTTP/1.1
Host: 127.0.0.1


[STATUS] Selesai melayani klien dari ('127.0.0.1', 54063). Waktu pelayanan: 10.1575 detik

Connected by ('127.0.0.1', 54075)
[STATUS] Sedang melayani klien dari ('127.0.0.1', 54075)...
Request:
GET /header.html HTTP/1.1
Host: localhost:5880
Connection: keep-alive
sec-ch-ua: "Chromium";v="136", "Google Chrome";v="136", "Not.A/Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;
v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: id,id-ID;q=0.9,en-US;q=0.8,en;q=0.7,ja;q=0.6,th;q=0.5,ru;q=0.4,zu;q=0.3


[STATUS] Selesai melayani klien dari ('127.0.0.1', 54075). Waktu pelayanan: 10.0038 detik
```

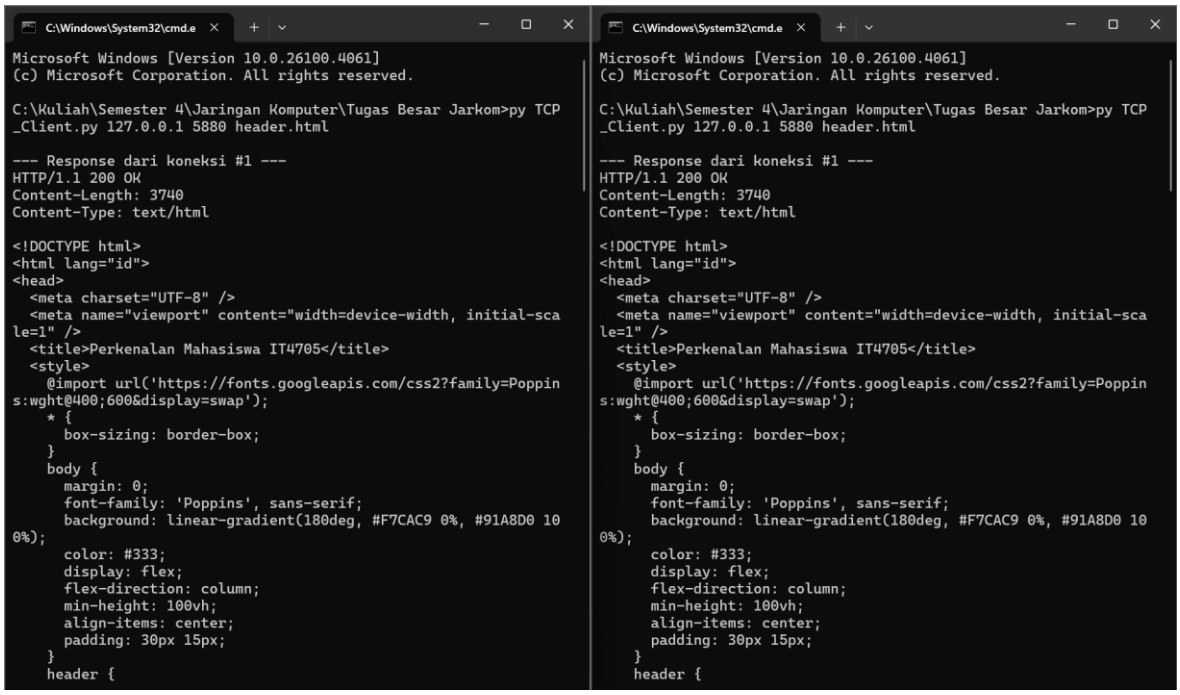
- Response Web



b. 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.

Contoh Output:

- Response Klien(Menggunakan 2 CMD yang berbeda) :



- Response Server :

```
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', 55140)...
Request:
GET /header.html HTTP/1.1
Host: 127.0.0.1


[STATUS] Sedang melayani klien dari ('127.0.0.1', 55141)...
Request:
GET /header.html HTTP/1.1
Host: 127.0.0.1


[STATUS] Selesai melayani klien dari ('127.0.0.1', 55140). Waktu
 pelayanan: 10.0018 detik

[STATUS] Selesai melayani klien dari ('127.0.0.1', 55141). Waktu
 pelayanan: 10.0021 detik
```

Pengujian ini membuktikan bahwa implementasi multithreading pada server web efektif dalam meningkatkan performa dan efisiensi dalam menangani banyak permintaan secara simultan.

BAB 5
KESIMPULAN

Berdasarkan hasil perancangan dan implementasi, server web sederhana yang dibangun menggunakan bahasa pemrograman Python berhasil menjalankan fungsi utamanya, yaitu menerima dan memproses request HTTP dari klien, mengirimkan file yang diminta, serta memberikan pesan error seperti "404 Not Found" dan "405 Method Not Allowed" bila terjadi kesalahan. Server ini juga mendukung dua mode operasi, yaitu single-threaded dan multi-threaded, yang memungkinkan pengujian performa dalam dua skenario berbeda.

Penggunaan multithreading dalam pengembangan server terbukti memberikan peningkatan performa yang signifikan. Pada mode single-threaded, server hanya mampu menangani satu permintaan dalam satu waktu, sehingga klien lainnya harus menunggu hingga proses sebelumnya selesai. Hal ini menyebabkan waktu total yang dibutuhkan untuk melayani lima klien mencapai sekitar 50 detik.

Sebaliknya, mode multi-threaded memungkinkan server untuk melayani beberapa klien secara bersamaan dengan menggunakan thread terpisah untuk setiap koneksi. Hasil pengujian menunjukkan bahwa waktu total yang dibutuhkan hanya sedikit lebih dari 10 detik, karena tidak ada penundaan antar klien yang disebabkan oleh proses blocking.

Dengan demikian, dapat disimpulkan bahwa implementasi multithreading merupakan pendekatan yang sangat efektif dalam meningkatkan efisiensi dan responsivitas server web, terutama dalam kondisi dengan banyak permintaan secara bersamaan. Pendekatan ini memungkinkan server tetap cepat dan stabil dalam melayani klien secara paralel.