# LAPORAN
# JARINGAN KOMPUTER

# MEMBUAT PROGRAM WEB SERVER SEDERHANA BERBASIS TCP SOCKET PROGRAMMING


**Disusun Oleh**
**IHAB HASANAIN AKMAL - 103032330054**
**FAISAL IHSAN SANTOSO - 103032330012**
**ARIE FARCHAN FYRZATULLAH - 103032330094**
**TELKOM UNIVERSITY**

**2024**

## A. Latar Belakang

Jika Anda ingin mempelajari cara membuat server web sederhana, maka pembuatan program web server berbasis pemrograman socket TCP bisa menjadi langkah awal yang baik. Pemrograman socket TCP digunakan untuk membuat aplikasi web yang mengoperasikan protokol TCP/IP, protokol standar untuk komunikasi Internet. Dengan memanfaatkan pemrograman socket TCP, Anda bisa membangun server web yang menerima permintaan dari klien dan mengirimkan respons menggunakan protokol HTTP. Melalui praktik ini, Anda dapat memahami konsep dan protokol web dasar, serta meningkatkan keterampilan pemrograman Anda. Selain itu, membuat server web sederhana juga dapat membantu memahami cara kerja server web yang lebih kompleks di balik layar. Laporan ini cocok untuk mereka yang ingin belajar membuat server web sederhana menggunakan bahasa pemrograman dan antarmuka TCP, atau yang ingin memperdalam pemahaman mereka tentang konsep dan protokol jaringan dasar.

## B. Batasan Masalah

1. Laporan hanya memfokuskan pada pembuatan server web sederhana menggunakan satu bahasa pemrograman saja, dan tidak membahas implementasi di berbagai bahasa pemrograman.
2. Hanya membahas implementasi protokol HTTP versi 1.1 untuk permintaan dan respon web yang diterima dan dikirim oleh server.
3. Hanya membahas implementasi server web pada lingkungan pengembangan lokal dan tidak membahas implementasi pada lingkungan produksi, seperti hosting web server di internet atau cloud.
4. Tidak membahas implementasi fitur-fitur yang lebih kompleks seperti cache, load balancing, dan clustering.
5. Fokus pada implementasi server web yang sederhana dan mudah dipahami, sehingga pembaca dapat memahami dasar-dasar cara membuat server web dan TCP socket programming.

## C. Sistem Yang Dibangun

Berikut adalah sistem atau codingan yang sudah dibangun.

1. Untuk server yang menerapkan single thread :

```
import socket
import os
import mimetypes
import time

def handle_client(conn, addr):
    try:
        req = conn.recv(1024).decode()
        print(f"[REQUEST from {addr}]:\n{req}")

        if not req:
            return
        time.sleep(5)

        path = req.split()[1].lstrip('/') or 'home.html'

        if os.path.isfile(path):
            with open(path, 'rb') as f:
```

```
                content = f.read()
            ctype = mimetypes.guess_type(path)[0] or
'application/octet-stream'
            headers = (
                f"HTTP/1.1 200 OK\r\n"
                f"Content-Type: {ctype}\r\n"
                f"Content-Length: {len(content)}\r\n"
                f"Connection: close\r\n\r\n"
            ).encode()
            conn.sendall(headers + content)
        else:
            resp = (
                "HTTP/1.1 404 Not Found\r\n"
                "Content-Type: text/html\r\n"
                "Connection: close\r\n\r\n"
                "<html><body><h1>404 Not Found</h1></body></html>"
            ).encode()
            conn.sendall(resp)
    except Exception as e:
        print(f"[ERROR]: {e}")
    finally:
        conn.close()

def start_server(port=2025):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
        server.bind(('', port))
        server.listen(10)
        print(f"[STARTING] Web Server running on port {port}...")
        try:
            while True:
                conn, addr = server.accept()
                handle_client(conn, addr)
        except KeyboardInterrupt:
            print("\n[SHUTTING DOWN] Server stopped.")

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

2. Untuk Server yang menerapkan multithread

```
import socket
import threading
import os
import mimetypes
import time

def handle_client(conn, addr):
    try:
        req = conn.recv(1024).decode()
        print(f"[REQUEST from {addr}]:\n{req}")
        if not req:
            return
        time.sleep(5)
        path = req.split()[1].lstrip('/') or 'home.html'

        if os.path.isfile(path):
            with open(path, 'rb') as f:
                content = f.read()
            ctype = mimetypes.guess_type(path)[0] or
'application/octet-stream'
            headers = (
                f"HTTP/1.1 200 OK\r\n"
                f"Content-Type: {ctype}\r\n"
                f"Content-Length: {len(content)}\r\n"
                f"Connection: close\r\n\r\n"
            ).encode()
            conn.sendall(headers + content)
        else:
            resp = (
                "HTTP/1.1 404 Not Found\r\n"
                "Content-Type: text/html\r\n"
                "Connection: close\r\n\r\n"
                "<html><body><h1>404 Not Found</h1></body></html>"
            ).encode()
            conn.sendall(resp)
    except Exception as e:
        print(f"[ERROR]: {e}")
    finally:
        conn.close()

def start_server(port=2026):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
        server.bind(('', port))
        server.listen(10)
        print(f"[STARTING] Web Server running on port {port}...")
```

```
        try:
            while True:
                conn, addr = server.accept()
                threading.Thread(target=handle_client, args=(conn,
addr), daemon=True).start()
                print(f"[ACTIVE CONNECTIONS]:
{threading.active_count() - 1}")
        except KeyboardInterrupt:
            print("\n[SHUTTING DOWN] Server stopped.")

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

3. Code untuk Client :

```
import threading
import socket
import sys

def request_page(i, host, port, file):
    try:
        s = socket.create_connection((host, int(port)))
        request = f"GET /{file} HTTP/1.1\r\nHost:
{host}:{port}\r\nConnection: close\r\n\r\n"
        s.sendall(request.encode())

        response = b""
        while True:
            data = s.recv(4096)
            if not data:
                break
            response += data

        s.close()
        print(f"[Thread-{i} RESPONSE FROM SERVER]:")
        print(response.decode('utf-8', errors='replace'))
        print()

    except Exception as e:
        print(f"[Thread-{i} ERROR]: {e}")

if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("Usage: python script.py <server_host> <server_port>
<filename>")
        sys.exit(1)
```

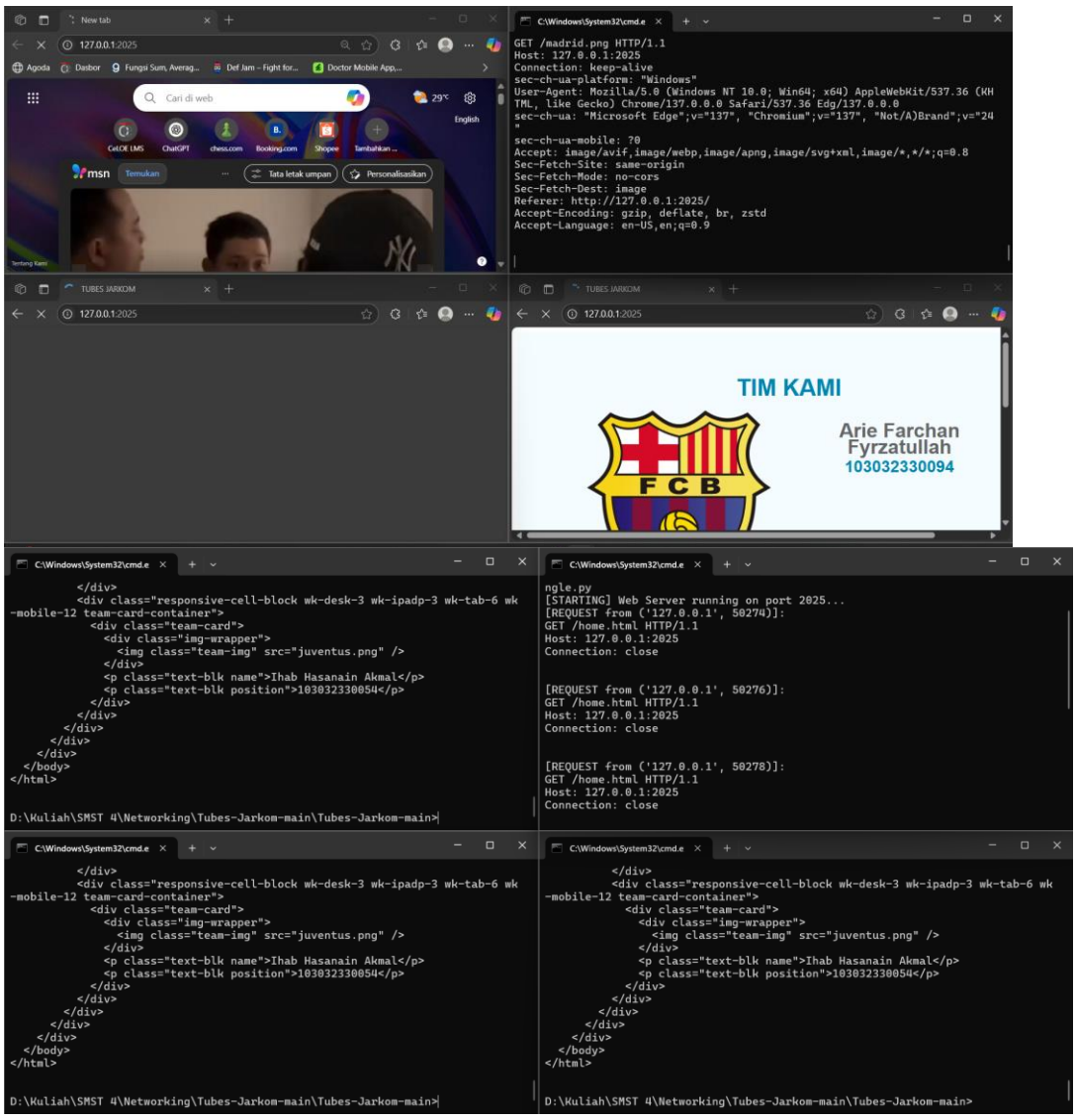
```
    _, host, port, file = sys.argv

    threads = [threading.Thread(target=request_page, args=(i, host,
port, file)) for i in range(10)]
    for t in threads:
        t.start()
    for t in threads:
        t.join()
```

## D. Hasil Program

Berikut hasil dari program tersebut:

1. Hasil dari single threading :




Pada server single-threading, setiap request dari client dilayani secara bergantian. Terlihat di terminal server, request dari client diproses satu per satu, meskipun file yang diminta sama.

Server tidak bisa melayani beberapa koneksi secara bersamaan, sehingga jika ada banyak client, koneksi berikutnya harus menunggu hingga koneksi sebelumnya selesai.

2. Hasil dari Multithreading :


Setelah server multi-threading dijalankan, terlihat bahwa setiap kali client melakukan request, jumlah Active Connections pada terminal server bertambah sesuai jumlah koneksi aktif. Misalnya saat ada tiga client yang terhubung bersamaan, jumlah koneksi aktif bertambah hingga ACTIVE CONNECTIONS: 3.

Setiap client berhasil menerima file home.html berisi informasi tim secara bersamaan tanpa harus menunggu client lain selesai. Hal ini menunjukkan bahwa server dapat melayani banyak koneksi secara paralel berkat multi-threading, sehingga lebih responsif dibanding single-threading.

## E. Kesimpulan

Dari hasil pengujian ini dapat disimpulkan bahwa penggunaan metode multi-threading pada server jauh lebih efektif dan efisien dibandingkan single-threading, terutama saat menangani banyak koneksi client secara bersamaan. Multi-threading memungkinkan server memproses beberapa request secara paralel, sehingga meningkatkan kecepatan respons dan mengurangi waktu tunggu antar client. Sebaliknya, single-threading memiliki keterbatasan karena hanya dapat memproses satu request dalam satu waktu, yang dapat menyebabkan antrean saat beban koneksi meningkat.