JARINGAN KOMPUTER –MODUL 8
WEB SERVER

Nama : Muhammad Hamzah Haifan Ma'ruf

NIM : 2311102091

Kelas : S1IF-11-07

Kode Dosen : AIZ

JAWABAN JURNAL

A. Teori

1. Apa yang dimaksud dengan web server? Jelaskan bagaimana sebuah web server berkomunikasi dengan klien menggunakan protokol HTTP! Apa peran socket dalam proses ini? dan mengapa TCP menjadi pilihan utama untuk komunikasi antara web server dan klien?

   Web server adalah sebuah perangkat lunak yang berfungsi untuk melayani permintaan dari klien, seperti browser, dalam mengakses halaman atau sumber daya web. Komunikasi antara web server dan klien dilakukan menggunakan protokol HTTP (HyperText Transfer Protocol), di mana klien mengirimkan permintaan (request) seperti GET atau POST, lalu server merespon dengan data yang diminta, seperti file HTML atau gambar. Dalam proses ini, socket berperan sebagai jalur komunikasi dua arah antara klien dan server, memungkinkan data dikirim dan diterima melalui jaringan. Socket ini dibuka oleh web server untuk menerima koneksi dari klien. Protokol TCP digunakan karena menawarkan komunikasi yang andal—TCP memastikan bahwa data yang dikirim sampai dengan lengkap, urut, dan tidak rusak. Ini sangat penting dalam konteks web, di mana integritas data harus terjamin.

2. Dalam konteks web server, apa yang dimaksud dengan kode status HTTP "404 Not Found"? Bagaimana cara web server menangani situasi ketika file yang diminta oleh klien tidak ditemukan? Jelaskan langkah-langkah yang harus dilakukan oleh server untuk memberikan respons yang sesuai!

   Kode status HTTP "404 Not Found" adalah respon dari web server yang menunjukkan bahwa file atau halaman yang diminta oleh klien tidak tersedia di server. Ketika klien mengirim permintaan, misalnya ingin mengakses halaman.html, server akan mencari file tersebut di dalam direktori penyimpanan. Jika file tidak ditemukan, server akan

mengembalikan respon HTTP dengan kode status 404, yang biasanya disertai dengan halaman error yang menjelaskan bahwa konten yang diminta tidak tersedia. Hal ini dilakukan agar pengguna mengetahui bahwa permintaan mereka tidak dapat dipenuhi, bukan karena masalah jaringan atau server, tetapi karena file memang tidak ada.

3. Mengapa multithreading penting dalam pengembangan web server modern? Jelaskan bagaimana pendekatan multithreading dapat meningkatkan performa web server dalam menangani beberapa permintaan klien secara bersamaan!Apa tantangan yang mungkin muncul saat mengimplementasikan multithreading pada web server?

   Multithreading sangat penting dalam pengembangan web server modern karena memungkinkan server menangani banyak permintaan klien secara bersamaan. Dalam satu waktu, bisa jadi puluhan atau bahkan ratusan pengguna mengakses sebuah website, dan dengan menggunakan multithreading, server dapat membuat satu thread untuk menangani setiap koneksi klien. Hal ini membuat server menjadi lebih efisien dan responsif. Namun, dalam implementasinya, multithreading juga memiliki tantangan, seperti sinkronisasi data antar thread yang bisa menyebabkan konflik (race condition), kemungkinan terjadinya deadlock ketika dua thread saling menunggu, serta penggunaan memori dan sumber daya yang lebih besar jika jumlah thread tidak dikontrol dengan baik.

4. Bagaimana web server menangani file statik seperti HTML, CSS, dan gambar? Jelaskan proses yang terjadi mulai dari permintaan klien hingga file dikirimkan ke klien. Apakah ada perbedaan dalam penanganan file statik dibandingkan dengan file dinamis?

   Dalam menangani file statik seperti HTML, CSS, JavaScript, atau gambar, web server bekerja dengan cara menerima permintaan dari klien, mencari file yang diminta di direktori server, membaca isinya, dan mengirimkannya kembali ke klien sebagai bagian dari respon HTTP. File statik tidak mengalami proses pengolahan sebelum dikirim, sehingga prosesnya lebih cepat. Berbeda dengan file dinamis seperti skrip PHP atau Python, file dinamis perlu diproses oleh interpreter atau program backend terlebih dahulu sebelum hasilnya dikirim ke klien. Karena itu, penanganan file statik lebih sederhana dibanding file dinamis, dan biasanya dioptimalkan agar proses pengiriman lebih cepat, misalnya dengan caching.

5. Jelaskan perbedaan utama antara web server dan web proxy. Apa fungsi utama dari masing-masing, dan bagaimana mereka bekerja sama dalam arsitektur jaringan?

   Web server dan web proxy memiliki fungsi yang berbeda dalam jaringan. Web server berfungsi untuk menyimpan dan menyajikan konten web kepada klien yang memintanya,

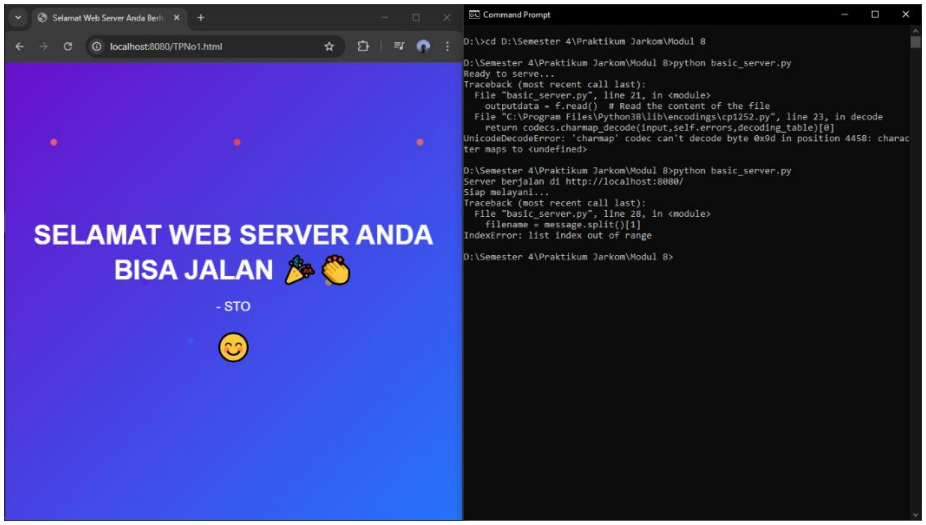
seperti halaman HTML atau file gambar. Sementara itu, web proxy bertindak sebagai perantara antara klien dan server, dengan tujuan mengatur, mengamankan, atau mempercepat akses internet. Web proxy dapat menyimpan cache konten agar permintaan selanjutnya lebih cepat, menyaring akses situs tertentu, atau menyembunyikan alamat IP pengguna. Dalam arsitektur jaringan, proxy sering digunakan di depan web server untuk meningkatkan kinerja dan keamanan. Jadi, web proxy dan web server dapat bekerja sama: klien mengakses proxy terlebih dahulu, lalu proxy meneruskan permintaan ke web server, menerima respon, dan meneruskannya kembali ke klien.

B. Web Server

1. Lengkapi skeleton code diatas! Lakukan ekseskusi untuk menjalankan code yang sudah anda lengkapi dengan menggunakan browser! (hint: pada address bar ketik: http://localhost:{nomor_port}/{nama_file})

```
from socket import *
import sys

# Membuat server socket
serverSocket = socket(AF_INET, SOCK_STREAM)

# Menentukan port server
serverPort = 8080  # kamu bisa ganti port ini jika perlu

# Bind socket ke port
serverSocket.bind(('', serverPort))

# Mendengarkan koneksi
serverSocket.listen(1)

print(f"Server berjalan di http://localhost:{serverPort}/")
print("Siap melayani...")

while True:
    # Menerima koneksi dari client
    connectionSocket, addr = serverSocket.accept()

    try:
        # Menerima request dari client
        message = connectionSocket.recv(1024).decode()

        # Ambil nama file yang diminta
        filename = message.split()[1]
        f = open(filename[1:], 'r', encoding='utf-8')  # Gunakan UTF-8!
```

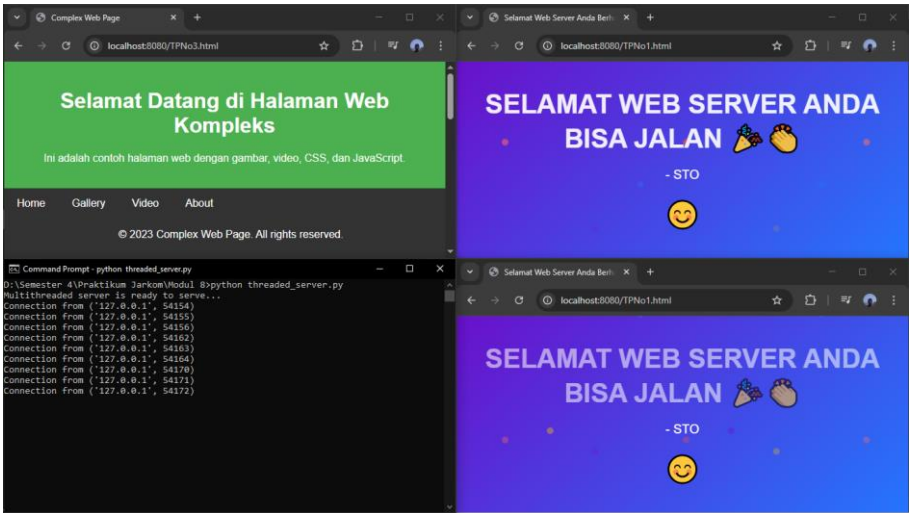
```
        # Baca isi file
        outputdata = f.read()

        # Kirim header HTTP 200 OK
        connectionSocket.send("HTTP/1.1 200 OK\r\n".encode())
        connectionSocket.send("Content-Type:
text/html\r\n\r\n".encode())

        # Kirim isi file ke client
        connectionSocket.send(outputdata.encode())

        # Tutup koneksi
        connectionSocket.close()

    except IOError:
        # Jika file tidak ditemukan, kirim 404
        error_message = """\
HTTP/1.1 404 Not Found\r\n
Content-Type: text/html\r\n\r\n
<html><head></head><body><h1>404 Not Found</h1></body></html>
"""
        connectionSocket.send(error_message.encode())
        connectionSocket.close()

# Program tidak akan sampai sini kecuali dimatikan secara manual
serverSocket.close()
sys.exit()
```



2. Selanjutnya jika web server yang sudah anda lengkapi tersebut dapat di-run, maka selanjutnya akan dikembangkan lagi, karena server itu seharusnya tidak mungkin hanya melakukan satu kali request HTTP. Kembangkan web server tersebut agar dapat melayani beberapa request sekaligus. Manfaatkan threading! Setelah anda menambahkan dan menyesuaikan threading pada kode tersebut, jalankan web server dan lakukan penerapan threading dengan mengakses 3 file yang sudah disediakan

```
from socket import *
import threading

# Fungsi untuk menangani client secara terpisah (multithreading)
def handle_client(connectionSocket):
    try:
        message = connectionSocket.recv(1024).decode()

        # Cek jika message kosong atau tidak sesuai format
        if not message or len(message.split()) < 2:
            connectionSocket.close()
            return

        filename = message.split()[1]

        # Buka file HTML yang diminta dengan encoding UTF-8
        with open(filename[1:], 'r', encoding='utf-8') as f:
            outputdata = f.read()

        # Kirim header HTTP 200 OK
        connectionSocket.send("HTTP/1.1 200 OK\r\n".encode())
        connectionSocket.send("Content-Type:
text/html\r\n\r\n".encode())

        # Kirim isi file ke browser
        connectionSocket.send(outputdata.encode())

    except FileNotFoundError:
        # Kirim respon 404 jika file tidak ditemukan
        error_message = """\
HTTP/1.1 404 Not Found\r\n
Content-Type: text/html\r\n\r\n
<html><head></head><body><h1>404 Not Found</h1></body></html>
"""
        connectionSocket.send(error_message.encode())

    except Exception as e:
        print(f"[ERROR] {e}")
```

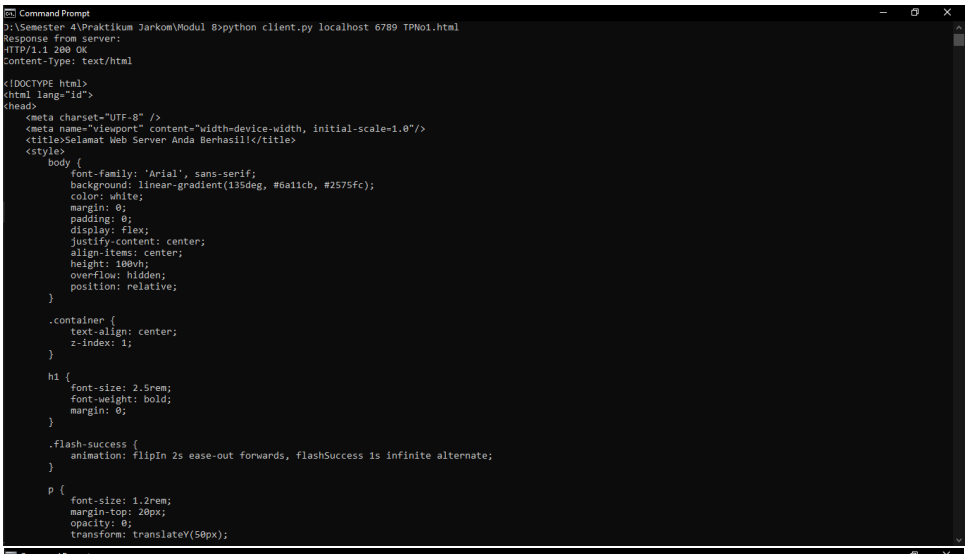
```
    # Tutup koneksi
    connectionSocket.close()

# Setup socket
serverSocket = socket(AF_INET, SOCK_STREAM)
serverPort = 8080
serverSocket.bind(('', serverPort))
serverSocket.listen(5)

print("Multithreaded server is ready to serve...")

while True:
    connectionSocket, addr = serverSocket.accept()
    print(f"Connection from {addr}")
    client_thread        =        threading.Thread(target=handle_client,
args=(connectionSocket,))
    client_thread.start()
```

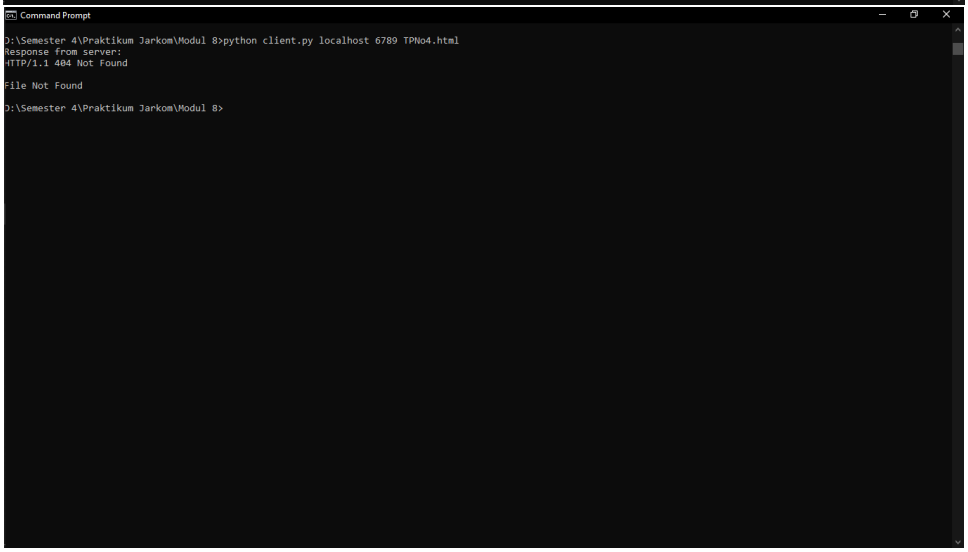


3. Selanjutnya dari pada menguji server menggunakan browser, kenapa tidak menggunakan client saja! sehingga bagaimana cara membuat klien HTTP sederhana dalam Python yang menguji server dengan mengirim permintaan GET menggunakan koneksi TCP? Klien harus menerima argumen baris perintah berupa alamat IP atau nama host server, nomor port server, dan jalur file yang diminta. Setelah terhubung ke server, klien harus mengirim permintaan HTTP GET, menerima respons dari server, dan menampilkannya sebagai output.

```
import socket
import sys
```

```
def main():
    if len(sys.argv) != 4:
        print("Usage: python client.py <host> <port> <filename>")
        return

    host = sys.argv[1]
    port = int(sys.argv[2])
    filename = sys.argv[3]

    try:
        client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        client_socket.connect((host, port))

        # Kirim permintaan HTTP GET
        request = f"GET /{filename} HTTP/1.1\r\nHost: {host}\r\n\r\n"
        client_socket.send(request.encode())

        # Terima dan tampilkan response
        response = client_socket.recv(4096)
        print("Response from server:")
        print(response.decode())

        client_socket.close()

    except Exception as e:
        print(f"Error: Unable to connect to the server at
{host}:{port}.")
        print(f"Details: {e}")

if __name__ == '__main__':
    main()
```





C. Web Proxy

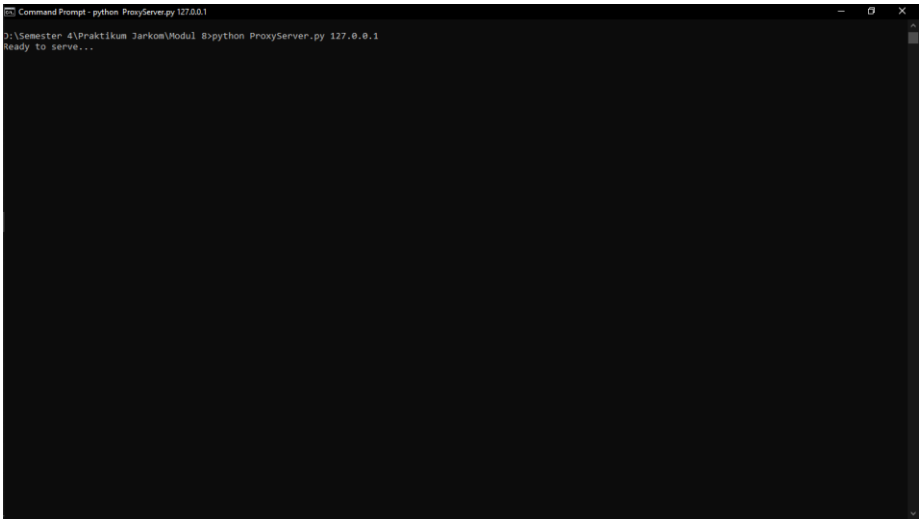
1. Lengkapi skeleton code diatas! Lakukan ekseskusi untuk menjalankan code yang sudah anda lengkapi!

```
from socket import *
import sys

if len(sys.argv) <= 1:
```

```
    print('Usage: "python ProxyServer.py server_ip"\n[server_ip: It
is the IP Address Of Proxy Server]')
    sys.exit(2)

# Create a server socket, bind it to a port and start listening
tcpSerSock = socket(AF_INET, SOCK_STREAM)
# Fill in start.
tcpSerSock.bind(('', 8888))  # Bind to port 8888
tcpSerSock.listen(1)         # Listen for incoming connections
# Fill in end.

while 1:
    # Start receiving data from the client
    print('Ready to serve...')
    tcpCliSock, addr = tcpSerSock.accept()
    print('Received a connection from:', addr)

    # Fill in start.
    message = tcpCliSock.recv(1024).decode()  # Receive the HTTP
request message from the client
    # Fill in end.
    print(message)

    # Extract the filename from the given message
    print(message.split()[1])
    filename = message.split()[1].partition("/")[2]
    print(filename)
    fileExist = "false"
    filetouse = "/" + filename
    print(filetouse)

    try:
        # Check whether the file exists in the cache
        f = open(filetouse[1:], "r")
        outputdata = f.readlines()
        fileExist = "true"

        # ProxyServer finds a cache hit and generates a response
message
        tcpCliSock.send("HTTP/1.0 200 OK\r\n".encode())
        tcpCliSock.send("Content-Type:text/html\r\n".encode())
        # Fill in start
        for line in outputdata:  # Iterate through each line of the
cached content
            tcpCliSock.send(line.encode())  # Send the cached
content to the client
        tcpCliSock.send("\r\n".encode())  # End of response
```

```
        # Fill in end.
        print('Read from cache')

    except IOError:
        if fileExist == "false":
            # Fill in start.
            c = socket(AF_INET, SOCK_STREAM)  # Create a socket on
the proxy server
            # Fill in end.
            hostn = filename.replace("www.", "", 1)
            print(hostn)

            try:
                # Connect to the socket to port 80
                # Fill in start.
                c.connect((hostn, 80))  # Connect to the actual web
server on port 80
                # Fill in end.

                # Create a temporary file on this socket and ask
port 80 for the file requested by the client
                fileobj = c.makefile('r', 0)
                fileobj.write("GET " + "http://" + filename + "
HTTP/1.0\n\n")

                # Read the response into buffer
                # Fill in start.
                response = fileobj.read()  # Read the entire
response from the web server
                # Fill in end.

                # Create a new file in the cache for the requested
file.
                # Also send the response in the buffer to client
socket and the corresponding file in the cache
                tmpFile = open("./" + filename, "wb")
                # Fill in start.
                tmpFile.write(response.encode())  # Write the
response to the cache file
                tcpCliSock.send(response.encode())  # Send the
response to the client
                tmpFile.close()  # Close the cache file
                # Fill in end.
            except:
                print("Illegal request")
        else:
            # HTTP response message for file not found
```

```
            # Fill in start.
            tcpCliSock.send("HTTP/1.0 404 Not Found\r\n".encode())
# Send 404 error response to the client
            tcpCliSock.send("Content-Type:text/html\r\n".encode())
# Specify the content type as HTML
            tcpCliSock.send("\r\n".encode())  # End of HTTP headers
(blank line)
            tcpCliSock.send("<html><body><h1>404 Not
Found</h1></body></html>\r\n".encode())  # Send HTML content
            # Fill in end.

    # Close the client socket
    tcpCliSock.close()

# Fill in start.
tcpSerSock.close()  # Close the server socket
# Fill in end.
```



2. Apa yang membedakan kode diatas (Web Proxy) dengan kode sebelumnya (Web Server)? Apakah mungkin sebuah web server menjadi web proxy juga?

   Fungsi Utama

   - Web Server: Menyimpan dan melayani konten asli (HTML, gambar, dll)
   - Web Proxy: Bertindak sebagai perantara antara client dan server, bisa menyimpan cache

Operasi

- Web Server merespons langsung dengan konten yang diminta
- Web Proxy pertama memeriksa cache lokal, jika tidak ada baru meminta ke server asli

Kemungkinan Web Server menjadi Proxy

- Secara teknis mungkin, tetapi tidak disarankan karena:
  - Konflik fungsi (server vs perantara)
  - Masalah performa
  - Masalah keamanan
- Lebih baik menggunakan komponen terpisah yang khusus untuk masing-masing fungsi

3. Pada kode diatas belum ada error handling, sehingga jika diberikan kode error handling berikut:

```
if "404 Not Found" in response:
print("File not found on the web server.")
tcpCliSock.send("HTTP/1.0 404 Not Found\r\n\r\n".encode())
```

akan disimpan dimana dan dibaris nomor berapa?

Kode error handling untuk "404 Not Found" sebaiknya ditempatkan setelah membaca response dari server asli (setelah line 70), sekitar line 71-72:

```
# Read the response into buffer
response = fileobj.read()  # Read the entire response from the web
server

# Error handling for 404
if "404 Not Found" in response:
    print("File not found on the web server.")
    tcpCliSock.send("HTTP/1.0 404 Not Found\r\n\r\n".encode())
    continue  # Skip caching and move to next request
```

4. Berikan pendapat anda apa yang terjadi pada web server jika ditambahkan post method?

Perubahan yang Diperlukan

- Proxy perlu meneruskan body request dari client ke server
- Proxy harus menangani berbagai header HTTP tambahan
- Cache menjadi lebih kompleks karena POST biasanya tidak di-cache

Dampak pada Web Server

- Beban lebih berat karena harus memproses request yang lebih kompleks
- Masalah keamanan meningkat (harus menangani data sensitif)
- Kompleksitas sistem bertambah (penanganan session, cookies, dll)
- Potensi masalah dengan stateful operations

Implementasi

- Proxy perlu memodifikasi kode untuk:
  - Menerima dan meneruskan body request
  - Menangani berbagai HTTP methods
  - Memproses header dengan benar
- Cache policy perlu diperbarui untuk mengecualikan POST requests

Contoh modifikasi untuk menangani POST:

```
# Di bagian penerimaan request
if message.startswith("POST"):
    # Get content length
    headers = message.split('\r\n')
    length = 0
    for header in headers:
        if header.startswith("Content-Length:"):
            length = int(header.split(':')[1].strip())

    # Receive the body
    body = tcpCliSock.recv(length).decode()

    # Forward to destination server
    c.sendall((message + body).encode())
```

D. Analisa Code

1. Bagaimana cara server menangani masalah encoding dalam konteks membaca file dan mengirim respons ke klien? Apa yang akan terjadi jika file yang diminta oleh klien berisi karakter non-ASCII (misalnya, karakter Unicode seperti ä, ß, atau 日本)? Jelaskan

langkah-langkah yang dilakukan oleh server untuk memastikan kompatibilitas dengan karakter tersebut.

Cara

- Baca file dengan encoding='utf-8'.
- Encode ulang konten ke UTF-8 (outputdata.encode('utf-8')).
- Sertakan charset=utf-8 di header HTTP.

Karakter Non-ASCII dengan UTF-8 menangani karakter Unicode (ä, ß, 日本語) dengan benar.

2. Mengapa server menggunakan metode sendall() alih-alih send() untuk mengirim respons ke klien? Apa yang akan terjadi jika send() digunakan dan data yang dikirim lebih besar dari buffer klien

- sendall():
  Mengirim data hingga habis, cocok untuk data besar (misal file HTML 10 KB).

- send():
  Hanya kirim sebagian data jika buffer penuh, berisiko pengiriman tidak lengkap.

- Alasan Pakai sendall():
  Memastikan seluruh respons terkirim tanpa perlu loop manual.