

Nama : Christoforus Indra Bagus Pratama
NRP : 5025231124
Kelas : Pemrograman Jaringan – D
Link Github : <https://github.com/itozt/tugas4Progjar/tree/main>

TUGAS 4

❖ Perintah Soal

Dengan berbasis pada program http server pada <https://github.com/rm77/progjar/tree/master/progjar5>

1. Dengan method yang ada pada spesifikasi HTTP server, tambahkan kemampuan http server untuk :
 - Melihat daftar file pada satu direktori (LIST)
 - Mengupload sebuah file (UPLOAD)
 - Menghapus file (DELETE)
2. Jalankan http server dengan mode
 - Thread pool
 - Process pool
3. Buatlah client implementation dari operasi tambahan tersebut. Jalankan operasi client server untuk kemampuan tersebut, berikanlah screenshot seperlunya, dan penjelasan dalam paragraf

❖ Langkah – Langkah Pengerjaan

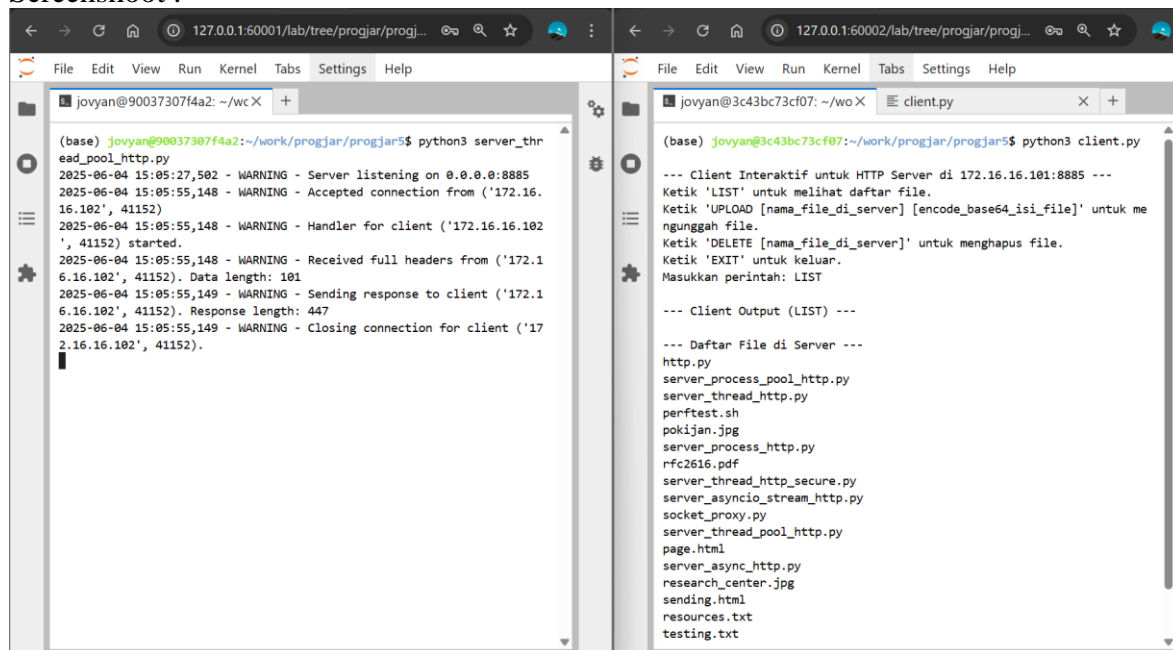
1. Buka Mesin 1 sebagai Server dan Mesin 2 sebagai Client.
Gunakan command di browser :
`http://127.0.0.1:60001 # untuk mesin 1`
`http://127.0.0.1:60002 # untuk mesin 2`
2. Pada mesin 2 (client), instal perintah requests dengan command :
`pip install requests`
3. Pada mesin 1 dan 2, ubah file http.py dengan command :
`vim http.py`
Lalu, ubah isinya menjadi seperti yang ada pada file di sini : [Link Github http.py](#)
4. Pada mesin 1 (server), ubah file server_thread_pool_http.py dengan command :
`vim server_thread_pool_http.py`
Lalu, ubah isinya menjadi seperti yang ada pada file di sini : [Link Github server_thread_pool_http.py](#)
5. Pada mesin 1 (server), ubah file server_process_pool_http.py dengan command :
`vim server_process_pool_http.py`
Lalu, ubah isinya menjadi seperti yang ada pada file di sini : [Link Github server_process_pool_http.py](#)
6. Pada mesin 2 (client), buat file client.py dengan command :
`nano client.py`
Lalu, isinya menjadi seperti yang ada pada file di sini : [Link Github client.py](#)
7. Pada mesin 2 (client), buat file domain.crt pada direktori yang sama dengan file client.py
`nano domain.crt`
Lalu, isinya menjadi seperti yang ada pada file di sini : [Link Github domain.crt](#)
8. Pada mesin 1, nyalakan server dengan command :
`python3 server_thread_pool_http.py` → untuk percobaan thread pool
`python3 server_thread_pool_http.py` → untuk percobaan process pool
9. Pada mesin 2, cek isi file client.py dengan command :
`nano client.py`
Pastikan Server IP dan Server Port sudah benar
`SERVER_IP = '172.16.16.101'` # IP mesin 1 (server)
`SERVER_PORT = 8885` # Port 8885 untuk Thread Pool, 8889 untuk Process Pool
10. Pada mesin 2, jalankan program client.py dengan command :
`python3 client.py`
11. Cek output yang dihasilkan pada mesin 1 (server), pastikan benar-benar sudah terhubung dengan mesin 2 (client).
12. Pada mesin 2, berikan perintah dengan ketentuan sebagai berikut :
 - Ketik 'LIST' untuk melihat daftar file.

- Ketik 'UPLOAD [nama_file_di_server] [encode_base64_isi_file]' untuk mengunggah file.
 - Ketik 'DELETE [nama_file_di_server]' untuk menghapus file.
 - Ketik 'EXIT' untuk keluar.
13. Cek output respon yang dihasilkan pada mesin 1 (server), dan cek output yang diterima pada mesin 2 (client).

❖ Hasil dan Pembahasan Thread Pool

➤ LIST

Screenshoot :



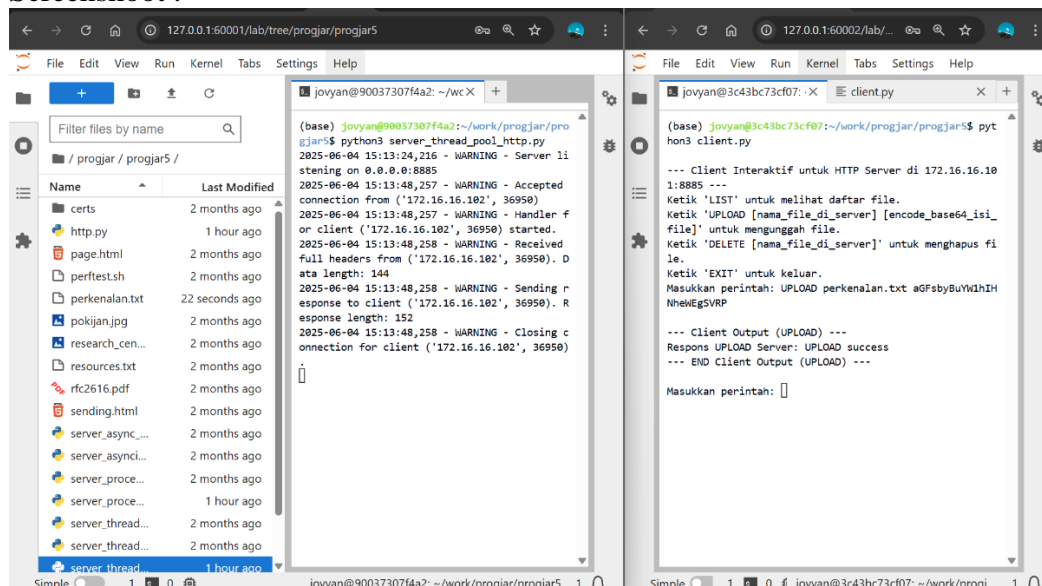
Penjelasan :

Percobaan yang telah dilakukan menunjukkan sistem client-server berfungsi dengan sangat baik dan sesuai harapan. Saat server di Mesin 1 dijalankan dan client di Mesin 2 memulai komunikasi dengan perintah LIST, client berhasil membuat koneksi TCP ke server di 172.16.16.101 pada port 8885. Client kemudian membangun sebuah permintaan HTTP POST yang valid, dengan body berisi perintah LIST, lalu mengirimkannya ke server. Di sisi server, log menunjukkan koneksi diterima dengan sukses dari client. Server lalu mengalokasikan sebuah thread dari thread pool untuk menangani permintaan tersebut. Permintaan LIST diterima dan diurai sepenuhnya oleh server.

Setelah itu, server memproses perintah LIST, mengambil daftar file yang ada di direktorinya, dan membangun respons HTTP 200 OK yang berisi daftar file tersebut di bagian body-nya. Respons ini kemudian dikirim kembali ke client, dan server menutup koneksi. Di sisi client, respons HTTP dari server berhasil diterima. Client secara otomatis memisahkan header dari body respons, lalu mencetak body yang berisi daftar nama-nama file. Inilah mengapa deretan nama file seperti `http.py`, `server_process_pool_http.py`, dan lainnya muncul di terminal client. Seluruh proses komunikasi dari pengiriman perintah hingga penerimaan dan tampilan hasil berjalan dengan lancar, menegaskan bahwa implementasi fungsionalitas LIST telah berhasil divalidasi dengan baik pada server mode Thread Pool.

➤ UPLOAD

Screenshoot :



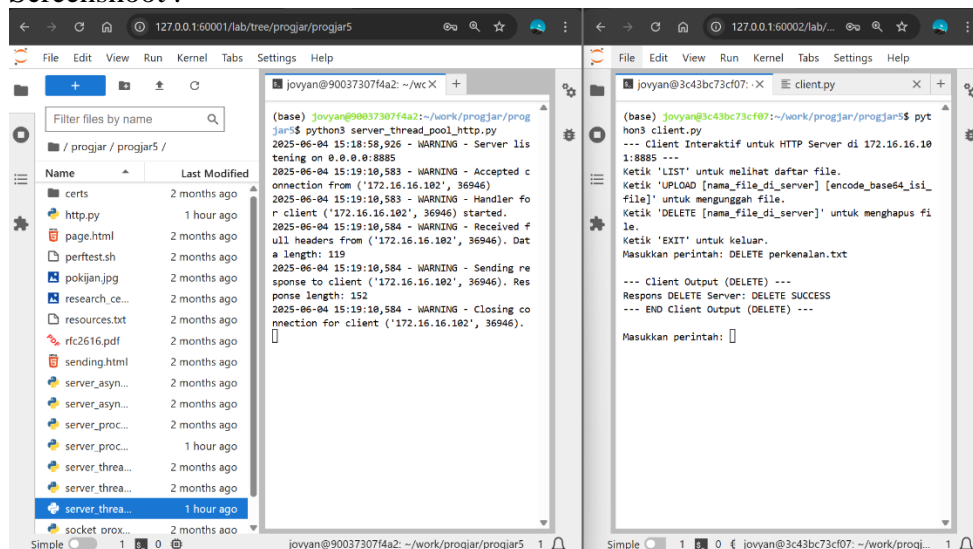
Penjelasan :

Pada percobaan ini, client berhasil melakukan operasi UPLOAD pada server Thread Pool. Saat client di Mesin 2 mengirim perintah `UPLOAD perkenalan.txt aGFsbyBuYW1hIHNeWEgSVRP`, client membangun permintaan HTTP POST yang valid. Permintaan ini menyertakan perkenalan.txt sebagai nama file dan aGFsbyBuYW1hIHNeWEgSVRP (konten Base64 dari "Halo nama saya ITO") sebagai isi file di body permintaan, dengan total panjang data 144 byte.

Server di Mesin 1, yang beroperasi dalam mode Thread Pool, menerima koneksi dan permintaan UPLOAD tersebut. Sebuah thread dari pool memproses permintaan, mendekode konten Base64, dan menyimpan file `perkenalan.txt` di direktori server. Setelah berhasil mengunggah file, server mengirimkan respons HTTP 200 OK kembali ke client dengan pesan "UPLOAD success" di body respons, berukuran 152 byte. Client kemudian menerima respons ini dan menampilkan pesan sukses tersebut di terminalnya.

➤ DELETE

Screenshoot :



Penjelasan :

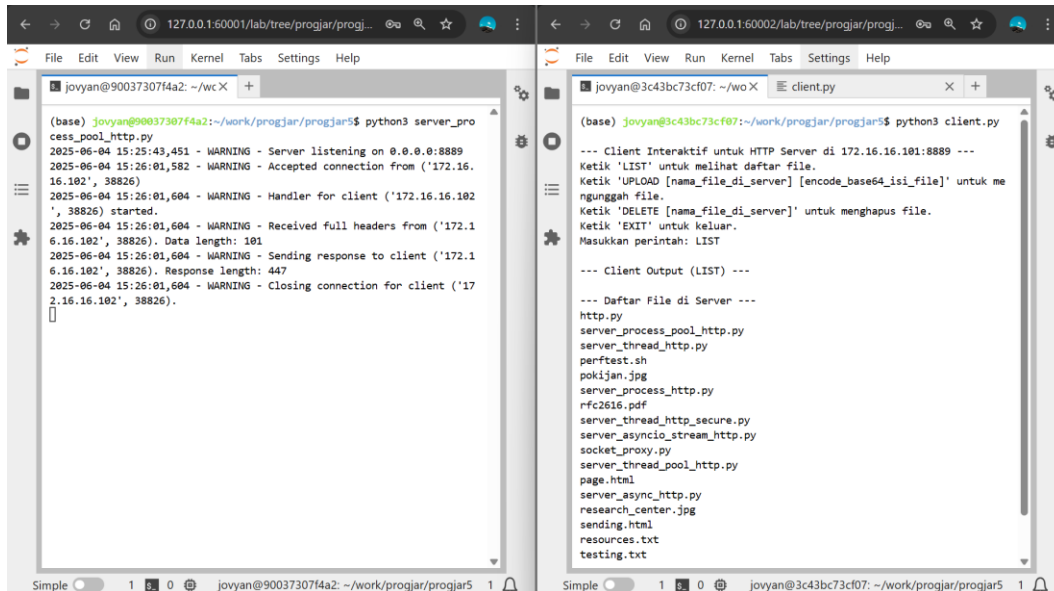
Dalam percobaan ini, client berhasil melakukan operasi DELETE pada server Thread Pool. Saat client di Mesin 2 mengirim perintah `DELETE perkenalan.txt`, ia membuat permintaan HTTP POST yang valid. Permintaan ini menyertakan perkenalan.txt sebagai nama file yang akan dihapus di body permintaan, dengan total panjang data 119 byte.

Server di Mesin 1, yang beroperasi dalam mode Thread Pool, menerima koneksi dan permintaan DELETE tersebut. Sebuah thread dari pool memproses permintaan, menemukan file perkenalan.txt, dan menghapusnya dari direktori server. Setelah berhasil menghapus file, server mengirimkan respons HTTP 200 OK kembali ke client dengan pesan "DELETE SUCCESS" di body respons, berukuran 152 byte. Client kemudian menerima respons ini dan menampilkan pesan sukses tersebut di terminalnya.

❖ Hasil dan Pembahasan Process Pool

➤ LIST

Screenshoot :



```
(base) jovyan@90037307f4a2: ~/work/progjar/progjar5$ python3 server_process_pool_http.py
2025-06-04 15:25:43,451 - WARNING - Server listening on 0.0.0.0:8889
2025-06-04 15:26:01,582 - WARNING - Accepted connection from ('172.16.16.102', 38826)
2025-06-04 15:26:01,604 - WARNING - Handler for client ('172.16.102', 38826) started.
2025-06-04 15:26:01,604 - WARNING - Received full headers from ('172.16.16.102', 38826). Data length: 101
2025-06-04 15:26:01,604 - WARNING - Sending response to client ('172.16.16.102', 38826). Response length: 447
2025-06-04 15:26:01,604 - WARNING - Closing connection for client ('172.16.16.102', 38826).

--- Client Interaktif untuk HTTP Server di 172.16.16.101:8889 ---
Ketik 'LIST' untuk melihat daftar file.
Ketik 'UPLOAD [nama_file_di_server] [encode_base64_isi_file]' untuk mengunggah file.
Ketik 'DELETE [nama_file_di_server]' untuk menghapus file.
Ketik 'EXIT' untuk keluar.
Masukkan perintah: LIST

--- Client Output (LIST) ---

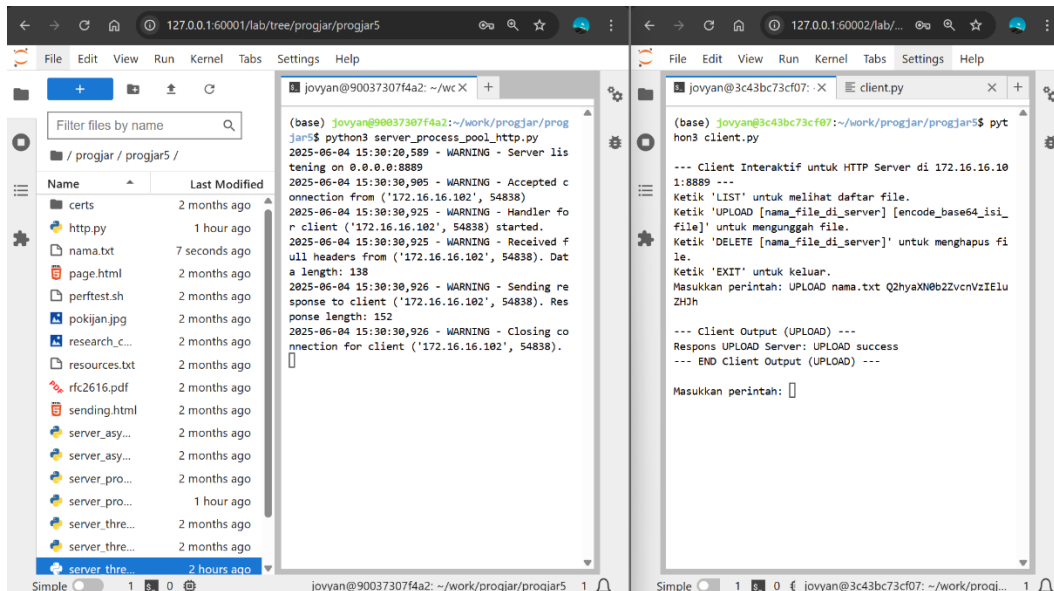
--- Daftar File di Server ---
http.py
server_process_pool_http.py
server_thread_http.py
perftest.sh
pokijan.jpg
server_process_http.py
rfc2616.pdf
server_thread_http_secure.py
server_asyncio_stream_http.py
socket_proxy.py
server_thread_pool_http.py
page.html
server_async_http.py
research_center.jpg
sending.html
resources.txt
testing.txt
```

Penjelasan :

Dalam percobaan ini, client berhasil menguji fungsionalitas LIST pada server yang berjalan dalam mode Process Pool. Saat client di Mesin 2 mengirim perintah LIST, ia membuat permintaan HTTP POST yang valid ke server di 172.16.16.101 pada port 8889. Server menerima permintaan ini, dan sebuah proses dari process pool ditugaskan untuk memprosesnya. Server berhasil membaca seluruh permintaan (101 byte), memproses perintah LIST, dan mengirimkan respons HTTP 200 OK yang berisi daftar file sebagai body (panjang 447 byte) kembali ke client. Client kemudian menerima, mem-parsing, dan menampilkan daftar file tersebut di terminalnya, menegaskan bahwa operasi LIST berfungsi dengan benar pada server mode Process Pool.

➤ UPLOAD

Screenshoot :



```
(base) jovyan@90037307f4a2: ~/work/progjar/progjar5$ python3 server_process_pool_http.py
2025-06-04 15:30:20,589 - WARNING - Server listening on 0.0.0.0:8889
2025-06-04 15:30:30,905 - WARNING - Accepted connection from ('172.16.16.102', 54838)
2025-06-04 15:30:30,925 - WARNING - Handler for client ('172.16.16.102', 54838) started.
2025-06-04 15:30:30,925 - WARNING - Received full headers from ('172.16.16.102', 54838). Data length: 138
2025-06-04 15:30:30,926 - WARNING - Sending response to client ('172.16.16.102', 54838). Response length: 152
2025-06-04 15:30:30,926 - WARNING - Closing connection for client ('172.16.16.102', 54838).

--- Client Interaktif untuk HTTP Server di 172.16.16.101:8889 ---
Ketik 'LIST' untuk melihat daftar file.
Ketik 'UPLOAD [nama_file_di_server] [encode_base64_isi_file]' untuk mengunggah file.
Ketik 'DELETE [nama_file_di_server]' untuk menghapus file.
Ketik 'EXIT' untuk keluar.
Masukkan perintah: UPLOAD nama.txt Q2hyaXN0b2ZvcnVzIEluZHIh

--- Client Output (UPLOAD) ---
Respons UPLOAD Server: UPLOAD success
--- END Client Output (UPLOAD) ---

Masukkan perintah: 
```

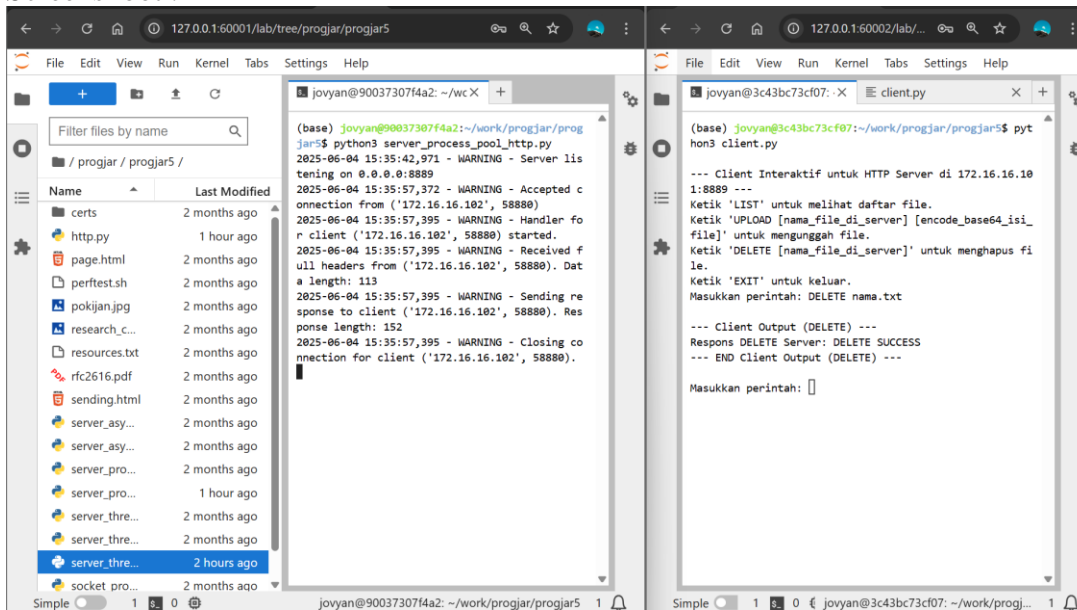
Penjelasan :

Dalam percobaan ini, client berhasil mengunggah file ke server yang beroperasi dalam mode Process Pool. Saat client di Mesin 2 mengirim perintah UPLOAD nama.txt Q2hyaXN0b2ZvcnVzIEluZHIh, client membuat permintaan HTTP POST yang valid. Permintaan ini berisi nama file nama.txt dan konten Base64 Q2hyaXN0b2ZvcnVzIEluZHIh di bagian body, dengan total panjang data 138 byte. Server di Mesin 1, yang mendengarkan pada port 8889 menggunakan process pool, menerima dan memproses permintaan UPLOAD tersebut. Sebuah proses dari pool menangani permintaan, mendekode konten Base64, dan berhasil menyimpan file nama.txt di direktori server. Sebagai konfirmasi, server mengirimkan respons

HTTP 200 OK dengan pesan "UPLOAD success" kembali ke client, yang kemudian ditampilkan di terminal client.

➤ DELETE

Screenshoot :



Penjelasan :

Pada percobaan ini, client berhasil melakukan operasi DELETE pada server yang berjalan dalam mode Process Pool. Ketika client di Mesin 2 mengirim perintah DELETE nama.txt, client membuat permintaan HTTP POST yang valid. Permintaan ini menyertakan nama.txt sebagai nama file yang akan dihapus di bagian body, dengan total panjang data 113 byte. Server di Mesin 1, yang mendengarkan pada port 8889 menggunakan process pool, menerima dan memproses permintaan DELETE tersebut. Sebuah proses dari pool menangani permintaan, berhasil menghapus file nama.txt dari direktori server. Sebagai konfirmasi, server mengirimkan respons HTTP 200 OK dengan pesan "DELETE SUCCESS" kembali ke client, yang kemudian ditampilkan di terminal client.

❖ Penjelasan Program yang Dimodifikasi

➤ http.py

Link : [http.py](#)

File http.py berfungsi sebagai inti server, menangani pemrosesan permintaan dan pembuatan respons. Perubahan utama meliputi penambahan fungsionalitas LIST, UPLOAD, dan DELETE sebagai perintah kustom yang dikirim melalui permintaan HTTP POST. Untuk mendukung ini, modul os dan base64 diimpor, dan metode proses diadaptasi untuk menerima raw bytes dari socket. Ini mengatasi masalah TypeError sebelumnya. Metode proses kini secara spesifik mengidentifikasi permintaan POST ke path /command, kemudian mendekode body permintaan tersebut dan meneruskannya ke metode handle_custom_command yang baru. Selain itu, penanganan permintaan GET untuk file statis ditingkatkan dengan validasi keberadaan file dan penentuan tipe konten yang lebih akurat.

```
def proses(self, raw_data_bytes):
    # ...
    headers_part_str = raw_data_bytes[:header_end_index].decode('latin-1')
    body_bytes = raw_data_bytes[header_end_index + 4:]
    # ...
    if (method == 'POST'):
        if object_address == '/command':
            try:
                body_string = body_bytes.decode('utf-8')
                return self.handle_custom_command(body_string)
            except UnicodeDecodeError:
                return self.response(400, 'Bad Request', 'Invalid body encoding for custom command.'.en
    # ...
```

Fungsionalitas inti LIST, UPLOAD, dan DELETE diimplementasikan dalam metode handle_custom_command yang baru. Perintah LIST menggunakan os.listdir untuk menampilkan file. Perintah UPLOAD memisahkan nama file dan konten Base64 dari body permintaan, mendekode konten Base64, dan menulisnya ke file di server. Perintah DELETE menghapus file berdasarkan nama yang diberikan. Setiap operasi ini juga dilengkapi dengan validasi dasar untuk nama file demi keamanan, serta

memberikan respons HTTP yang spesifik (misalnya, 200 OK, 400 Bad Request, 404 Not Found, atau 500 Internal Server Error) kembali ke client.

```
def handle_custom_command(self, command_string):
    command_string = command_string.strip()
    thedir = './'

    if command_string.upper() == 'LIST':
        # ... logika LIST ...
    elif command_string.upper().startswith('UPLOAD '):
        parts = command_string.split(' ', 2)
        if len(parts) < 3: # Validasi format UPLOAD
            return self.response(400, 'Bad Request', 'UPLOAD command format: UPLOAD [filename] [base64_
            filename = parts[1].strip()
            base64_content = parts[2].strip()
            # ... logika dekode dan simpan file ...
    elif command_string.upper().startswith('DELETE '):
        parts = command_string.split(' ', 1)
        if len(parts) < 2: # Validasi format DELETE
            return self.response(400, 'Bad Request', 'DELETE command format: DELETE [filename]'.encode(
            filename = parts[1].strip()
            # ... logika hapus file ...
    else:
        return self.response(400, 'Bad Request', 'Unknown custom command.'.encode('utf-8'), {'Content-t
```

➤ *client.py*

Link : [client.py](#)

File client.py kini berfungsi sebagai antarmuka interaktif yang ditingkatkan untuk berkomunikasi dengan server HTTP. Penyesuaian utama meliputi konfigurasi SERVER_IP dan SERVER_PORT yang jelas di awal file untuk kemudahan penyesuaian alamat server.

```
# Konfigurasi Server Tujuan
SERVER_IP = '172.16.16.101'
SERVER_PORT = 8885 # Sesuaikan port 8885 untuk Thread Pool, 8889 untuk Proc

# Konfigurasi Logging
logging.basicConfig(level=logging.ERROR, format='%(levelname)s:%(message)s'
```

Fungsi send_command telah diperkuat untuk membaca respons HTTP dengan lebih andal. Ia kini secara aktif mem-parsing header untuk menemukan Content-Length, memastikan seluruh body respons diterima sebelum melanjutkan, dan menggunakan timeout untuk mencegah program macet. Client juga dilengkapi dengan loop interaktif yang memungkinkan pengguna mengetikkan perintah LIST, UPLOAD, atau DELETE. Logika UPLOAD telah diperbaiki agar pengguna dapat memasukkan nama file tujuan di server dan konten Base64 secara langsung, tanpa ketergantungan pada file lokal.

```
def send_command(command_data_bytes):
    # ...
    sock.settimeout(10.0)
    # Fase 1: Baca header sampai '\r\n\r\n'
    # ...
    # Fase 2: Baca body berdasarkan Content-Length
    content_length_match = re.search(r'Content-Length:\s*(\d+)', headers_part, re.IGNORECASE)
    if content_length_match:
        # ... logika baca body ...
    # ...

    elif command_type == 'UPLOAD':
        parts = user_input.split(' ', 2) # Pisahkan 3 bagian
        if len(parts) < 3: # Validasi format
            # ... pesan error ...
        server_filename = parts[1].strip()
        encoded_content = parts[2].strip()
        # ... bentuk command_body dan kirim ...
```

➤ *server_thread_pool_http.py*

Link : [server_thread_pool_http.py](#)

File server_thread_pool_http.py menjalankan server HTTP menggunakan Thread Pool untuk menangani klien secara bersamaan. Modifikasi utamanya adalah pada fungsi ProcessTheClient, yang kini lebih andal dalam menerima data. Fungsi ini secara iteratif membaca byte dari socket hingga header HTTP lengkap (b"\r\n\r\n") terdeteksi, memastikan seluruh permintaan klien diterima sebelum diproses oleh httpserver.proses(). Peningkatan logging dengan logging.basicConfig dan penanganan error yang lebih luas (termasuk pencetakan traceback penuh) juga diterapkan untuk debugging yang lebih baik, serta memastikan koneksi klien selalu ditutup dengan benar.

```
def ProcessTheClient(connection,address):
    # ...
    rcv_bytes = b""
    while True: # Baca data sampai header lengkap atau koneksi ditutup
        data = connection.recv(2048)
        if data:
            rcv_bytes += data
            if b"\r\n\r\n" in rcv_bytes:
                break
        else:
            break
```

Fungsi utama Server() juga disesuaikan untuk menggunakan ThreadPoolExecutor, secara efisien menyerahkan setiap koneksi klien yang diterima ke thread yang tersedia di pool. Konfigurasi ini memungkinkan server menangani banyak klien secara bersamaan tanpa overhead pembuatan thread baru untuk setiap permintaan. Server juga kini memiliki penanganan yang bersih untuk interupsi dari pengguna (KeyboardInterrupt), memungkinkan penutupan server yang rapi saat Ctrl+C ditekan

```
def Server():
    # ... inisialisasi socket dan bind ke port 8885 ...
    with ThreadPoolExecutor(20) as executor: # Penggunaan ThreadPoolExecutor
        while True:
            try:
                connection, client_address = my_socket.accept()
                executor.submit(ProcessTheClient, connection, client_address) # Serahkan ke thread pool
            except KeyboardInterrupt: # Tangani Ctrl+C
                break
    # ... penanganan error accept lainnya ...
```

➤ *server_process_pool_http.py*

Link : [server_process_pool_http.py](#)

File server_process_pool_http.py menjalankan HTTP server menggunakan Process Pool untuk menangani klien secara bersamaan. Modifikasi utamanya berfokus pada peningkatan robustnes penerimaan data dari klien dan logging yang lebih informatif. Sebuah konfigurasi logging.basicConfig ditambahkan di awal file untuk mengontrol level dan format pesan log. Ini membantu dalam debugging dan pemantauan aktivitas server.

```
logging.basicConfig(level=logging.WARNING, format='%(asctime)s - %(levelname)s - %(message)s')

def ProcessTheClient(connection,address):
    # ...
    rcv_bytes = b""
    while True: # Baca data hingga header HTTP lengkap atau koneksi ditutup
        data = connection.recv(2048)
        if data:
            rcv_bytes += data
            if b"\r\n\r\n" in rcv_bytes:
                break
        else:
            break
```

Fungsi ProcessTheClient kini lebih canggih dalam menerima data, secara iteratif membaca byte dari socket hingga header HTTP lengkap (b"\r\n\r\n") terdeteksi. Ini memastikan server menerima seluruh permintaan sebelum menyerahkannya ke httpserver.proses(). Penanganan error juga ditingkatkan dengan blok try-except yang lebih luas dan logging.error(exc_info=True) untuk mencetak traceback penuh jika terjadi pengecualian, serta memastikan socket klien selalu ditutup dengan benar di blok finally. Fungsi utama Server() menggunakan ProcessPoolExecutor dengan 20 proses, secara efisien menyerahkan setiap koneksi klien yang diterima ke proses yang tersedia di pool, dan memiliki penanganan yang bersih untuk interupsi pengguna (KeyboardInterrupt).

```
def Server():
    # ... inisialisasi socket dan bind ke port 8889 ...
    my_socket.bind(('0.0.0.0', 8889)) # Port spesifik untuk process pool
    # ...
    with ProcessPoolExecutor(20) as executor: # Penggunaan ProcessPoolExecutor
        while True:
            try:
                connection, client_address = my_socket.accept()
                executor.submit(ProcessTheClient, connection, client_address) # Serahkan ke process pool
            except KeyboardInterrupt: # Tangani Ctrl+C
                break
    # ... penanganan error accept lainnya ...
```