Jamaluddin
5025221157
Pemrograman Jaringan C

Tugas 3 Pemrograman Jaringan C

Link Github : https://github.com/feersdilaa/progjar_task3

1) Pada file server protocol (https://github.com/rm77/progjar/tree/master/progjar4a ),

   Tambahkan kemampuan

   - Upload file
     - Content file yang diupload harus diencode dulu dengan format base64
   - Hapus file

   Jawab :

   @file_protocol.py

```python
import json
import logging

from file_interface import FileInterface

"""
* Class FileProtocol bertugas untuk memproses
  data string dari client dan menerjemahkannya sesuai
  protokol/aturan yang dibuat (dalam format JSON).

* Data yang diterima dari client berbentuk bytes,
  lalu dikonversi menjadi string JSON, kemudian diproses.

* FileProtocol akan memanggil metode dari FileInterface
  sesuai perintah ('command') dan parameter ('params')
  yang diberikan dalam string JSON.
"""

class FileProtocol:
    def __init__(self):
        # Inisialisasi objek untuk menangani operasi file
        self.file_handler = FileInterface()

    def proses_string(self, input_string=''):
        # Log string yang diterima untuk diproses
        logging.warning(f"String diproses: {input_string}")
        try:
            # Parse string menjadi dictionary dari JSON
            request_data = json.loads(input_string)

            # Ambil perintah (command) dan ubah ke huruf kecil
            command = request_data.get('command', '').lower()
            logging.warning(f"Memproses command: {command}")

            # Ambil parameter (jika ada), default ke list kosong
            parameters = request_data.get('params', [])
            logging.warning(f"Parameter: {parameters}")

            # Panggil method dari file_handler sesuai nama command
            method = getattr(self.file_handler, command)
            result = method(parameters)

            # Kembalikan hasil dalam format JSON string
            return json.dumps(result)

        except Exception as error:
            # Log kesalahan jika terjadi error saat parsing atau eksekusi command
            logging.warning(f"Exception saat memproses perintah: {error}")
            return json.dumps({
                'status': 'ERROR',
                'data': str(error)
            })


if __name__ == '__main__':
    # Contoh penggunaan FileProtocol
    fp = FileProtocol()

    # Contoh input valid dalam format JSON
    print(fp.proses_string(json.dumps({"command": "list", "params": []})))
    print(fp.proses_string(json.dumps({"command": "get", "params": ["pokijan.jpg"]})))
```

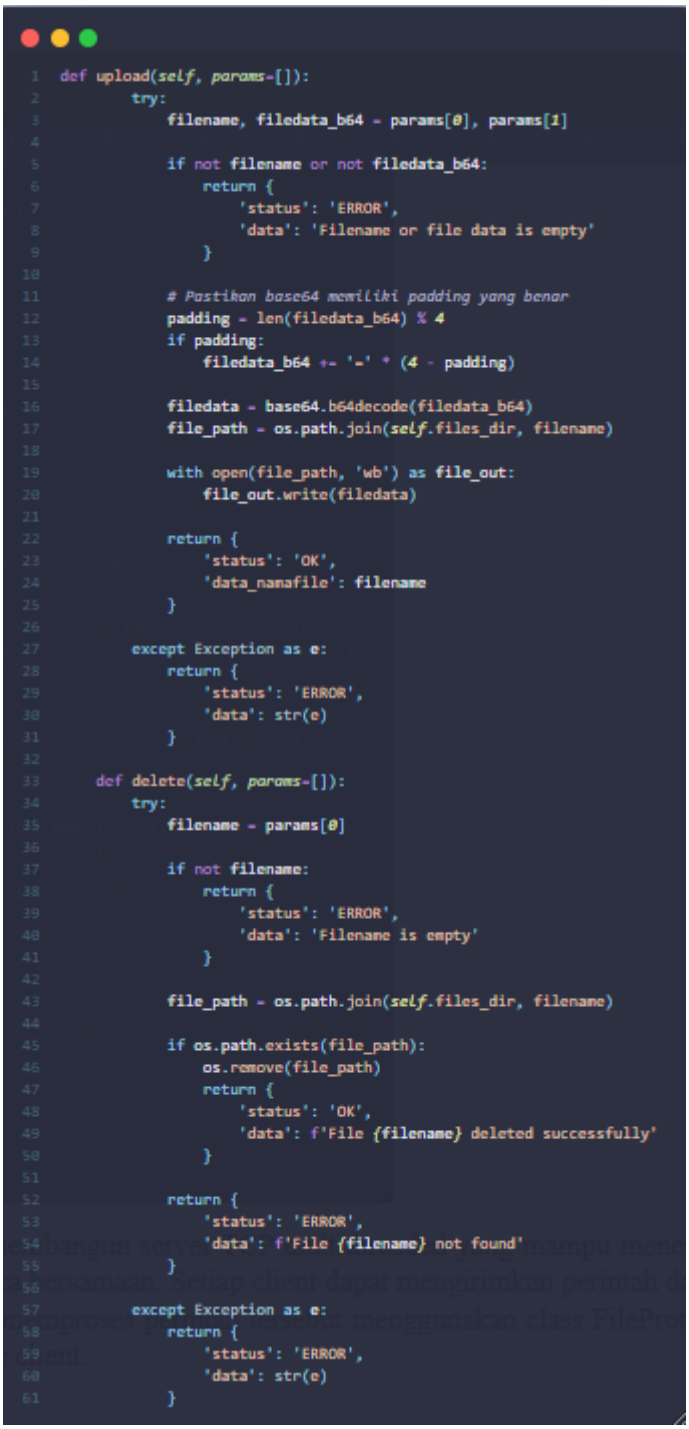
Diatas bagian dari sistem komunikasi antara client dan server yang menggunakan protokol berbasis JSON string untuk melakukan operasi file. Tugas utamanya adalah menerima perintah dari client yang dikirim dalam bentuk string JSON, kemudian mengurai (parse) data tersebut, dan akhirnya mengeksekusi perintah yang diminta dengan bantuan objek dari kelas FileInterface.

@file_server.py

```python
from socket import *
import socket
import threading
import logging
import time
import sys


from file_protocol import  FileProtocol
fp = FileProtocol()


class ProcessTheClient(threading.Thread):
    def __init__(self, connection, address):
        self.connection = connection
        self.address = address
        threading.Thread.__init__(self)

    def run(self):
        while True:
            data = self.connection.recv(32)
            if data:
                d = data.decode()
                hasil = fp.proses_string(d)
                hasil=hasil+"\r\n\r\n"
                self.connection.sendall(hasil.encode())
            else:
                break
        self.connection.close()


class Server(threading.Thread):
    def __init__(self,ipaddress='0.0.0.0',port=8889):
        self.ipinfo=(ipaddress,port)
        self.the_clients = []
        self.my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.my_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        threading.Thread.__init__(self)

    def run(self):
        logging.warning(f"server berjalan di ip address {self.ipinfo}")
        self.my_socket.bind(self.ipinfo)
        self.my_socket.listen(1)
        while True:
            self.connection, self.client_address = self.my_socket.accept()
            logging.warning(f"connection from {self.client_address}")

            clt = ProcessTheClient(self.connection, self.client_address)
            clt.start()
            self.the_clients.append(clt)


def main():
    svr = Server(ipaddress='0.0.0.0',port=6666)
    svr.start()


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

Diatas merupakan code yang membangun server yang mampu menerima koneksi dari banyak client secara bersamaan. Setiap client dapat mengirimkan perintah dalam format JSON, dan server akan memproses perintah tersebut menggunakan class FileProtocol lalu mengembalikan hasilnya ke client.

@file_interface.py



```
def upload(self, params=[]):
        try:
            filename, filedata_b64 = params[0], params[1]

            if not filename or not filedata_b64:
                return {
                    'status': 'ERROR',
                    'data': 'Filename or file data is empty'
                }

            # Pastikan base64 memiliki padding yang benar
            padding = len(filedata_b64) % 4
            if padding:
                filedata_b64 += '=' * (4 - padding)

            filedata = base64.b64decode(filedata_b64)
            file_path = os.path.join(self.files_dir, filename)

            with open(file_path, 'wb') as file_out:
                file_out.write(filedata)

            return {
                'status': 'OK',
                'data_namafile': filename
            }

        except Exception as e:
            return {
                'status': 'ERROR',
                'data': str(e)
            }

    def delete(self, params=[]):
        try:
            filename = params[0]

            if not filename:
                return {
                    'status': 'ERROR',
                    'data': 'Filename is empty'
                }

            file_path = os.path.join(self.files_dir, filename)

            if os.path.exists(file_path):
                os.remove(file_path)
                return {
                    'status': 'OK',
                    'data': f'File {filename} deleted successfully'
                }

            return {
                'status': 'ERROR',
                'data': f'File {filename} not found'
            }

        except Exception as e:
            return {
                'status': 'ERROR',
                'data': str(e)
            }
```

i) Upload File

Metode ini menerima dua parameter melalui list params: nama file (filename) dan isi file yang telah dikodekan dalam format base64 (filedata_b64). Pertama, metode ini memeriksa apakah kedua parameter tersebut tersedia. Jika tidak, akan dikembalikan status ERROR. Selanjutnya, metode memeriksa apakah panjang data base64 sudah benar dan menambahkan padding = jika perlu agar data bisa didekode dengan benar. Setelah itu, data base64 didekode menjadi biner, lalu ditulis ke file di lokasi yang ditentukan. Jika berhasil, metode mengembalikan respon dengan status OK dan nama

file yang telah diunggah. Jika terjadi kesalahan (misalnya decoding gagal atau file tidak bisa ditulis), maka akan dikembalikan status ERROR dengan pesan kesalahan.

ii) Delete File
Metode ini kemudian membuat path lengkap file berdasarkan nama file dan memeriksa apakah file tersebut benar-benar ada di dalam direktori. Jika ditemukan, file dihapus menggunakan os.remove(), dan server mengembalikan status OK dengan pesan bahwa file berhasil dihapus. Namun, jika file tidak ditemukan, akan dikembalikan status ERROR dengan pesan bahwa file tidak ada. Seperti metode lain, jika ada kesalahan saat proses berlangsung, metode ini akan menangkap exception dan mengembalikan status ERROR beserta deskripsi error-nya.

2) Update-lah spesifikasi protokol (PROTOKOL.txt) yang telah ada pada contoh dengan kemampuan yang baru ditambahkan tersebut, berikan penjelasan tambahan dalam satu paragraf

Jawab:

FILE SERVER
TUJUAN: melayani client dalam request file server

ATURAN PROTOKOL:
- client harus mengirimkan request dalam bentuk string
- string harus dalam format
  REQUEST spasi PARAMETER
- PARAMETER dapat berkembang menjadi PARAMETER1 spasi PARAMETER2 dan seterusnya

REQUEST YANG DILAYANI:
- informasi umum:
  * Jika request tidak dikenali akan menghasilkan pesan
    - status: ERROR
    - data: request tidak dikenali
  * Semua result akan diberikan dalam bentuk JSON dan diakhiri
    dengan character ascii code #13#10#13#10 atau "\r\n\r\n"

LIST
* TUJUAN: untuk mendapatkan daftar seluruh file yang dilayani oleh file server
* PARAMETER: tidak ada
* RESULT:
- BERHASIL:
  - status: OK
  - data: list file
- GAGAL:
  - status: ERROR
  - data: pesan kesalahan

GET
* TUJUAN: untuk mendapatkan isi file dengan menyebutkan nama file dalam parameter
* PARAMETER:
  - PARAMETER1 : nama file
* RESULT:

- BERHASIL:
  - status: OK
  - data_namafile : nama file yang diminta
  - data_file : isi file yang diminta (dalam bentuk base64)
- GAGAL:
  - status: ERROR
  - data: pesan kesalahan

UPLOAD
* TUJUAN: mengunggah file ke server
* PARAMETER:
  - PARAMETER1 : nama file
  - PARAMETER2 : melakukan encode terhadap isi file
* RESULT:
- BERHASIL:
  - status: OK
  - data_namafile : nama file yang berhasil diunggah ke server
- GAGAL:
  - status: ERROR
  - data: pesan berisi kesalahan

Klien memakai perintah UPLOAD untuk mengirimkan file ke server agar disimpan di folder "files". Permintaan ini memerlukan nama file dan isi file yang sudah dikodekan dalam format base64. Proses encoding base64 dilakukan di sisi klien sebelum pengiriman. Jika berhasil, server akan merespons dengan status OK dan nama file yang disimpan. Namun, jika ada masalah seperti parameter tidak lengkap, format base64 tidak valid, atau kegagalan penyimpanan, server akan mengembalikan status ERROR beserta pesan kesalahan yang sesuai.

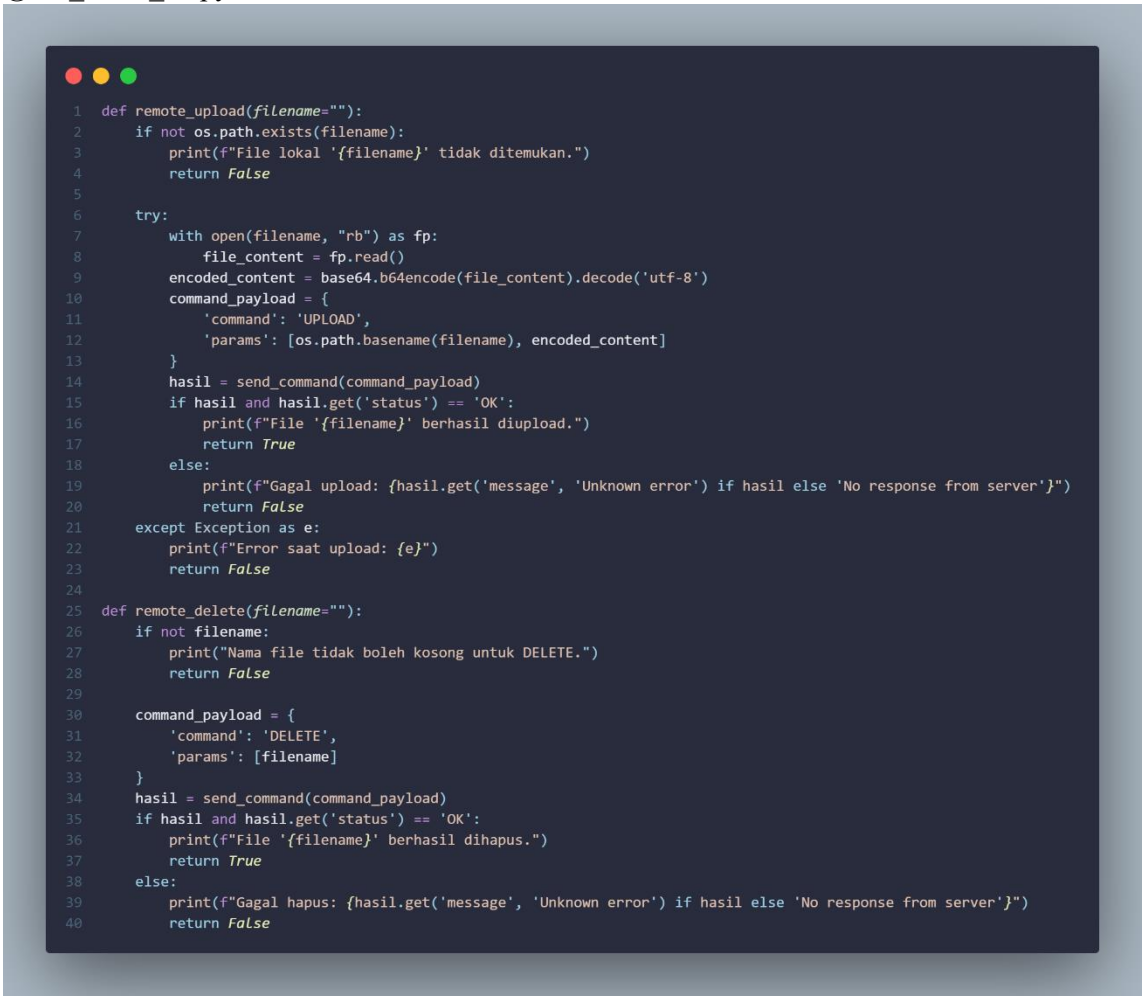
DELETE
* TUJUAN: untuk menghapus file dengan menyebutkan nama file dalam parameter
* PARAMETER:
  - PARAMETER1 : nama file
* RESULT:
- BERHASIL:
  - status: OK
  - data: pesan berisi sukses menghapus file
- GAGAL:
  - status: ERROR
  - data: pesan berisi kesalahan

Klien menggunakan perintah DELETE untuk menghapus file di folder "files" pada server. Cukup berikan nama file yang ingin dihapus. Server akan mencari file tersebut, dan jika ditemukan, akan menghapusnya. Jika berhasil, server merespons dengan status OK dan konfirmasi. Jika ada masalah, seperti file tidak ditemukan, server akan mengembalikan status ERROR dan pesan kesalahan.

3) Buatlah client implementation dari operasi tambahan tersebut. Jalankan operasi client server untuk kemampuan tersebut, berikanlah screenshot seperlunya, dan penjelasan dalam paragraph

Jamaluddin
5025221157
Pemrograman Jaringan C

Jawab :
@file_client_cli.py



```python
def remote_upload(filename=""):
    if not os.path.exists(filename):
        print(f"File lokal '{filename}' tidak ditemukan.")
        return False

    try:
        with open(filename, "rb") as fp:
            file_content = fp.read()
        encoded_content = base64.b64encode(file_content).decode('utf-8')
        command_payload = {
            'command': 'UPLOAD',
            'params': [os.path.basename(filename), encoded_content]
        }
        hasil = send_command(command_payload)
        if hasil and hasil.get('status') == 'OK':
            print(f"File '{filename}' berhasil diupload.")
            return True
        else:
            print(f"Gagal upload: {hasil.get('message', 'Unknown error') if hasil else 'No response from server'}")
            return False
    except Exception as e:
        print(f"Error saat upload: {e}")
        return False

def remote_delete(filename=""):
    if not filename:
        print("Nama file tidak boleh kosong untuk DELETE.")
        return False

    command_payload = {
        'command': 'DELETE',
        'params': [filename]
    }
    hasil = send_command(command_payload)
    if hasil and hasil.get('status') == 'OK':
        print(f"File '{filename}' berhasil dihapus.")
        return True
    else:
        print(f"Gagal hapus: {hasil.get('message', 'Unknown error') if hasil else 'No response from server'}")
        return False
```

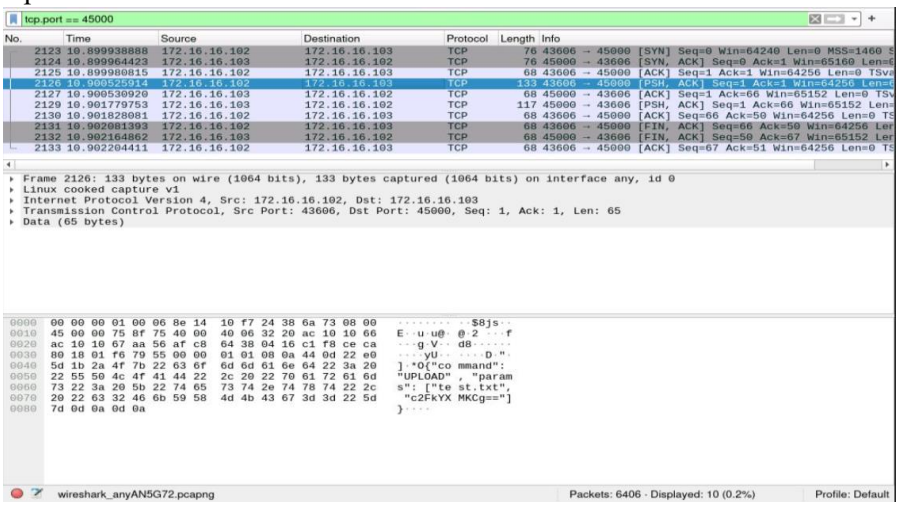
- remote_upload(filename): Fungsi ini membaca konten file lokal yang ditentukan, mengenkodenya ke Base64, dan mengirimkan perintah UPLOAD beserta nama file dan kontennya ke server. Ini memungkinkan pengguna untuk mengunggah file dari mesin lokal mereka ke server.
- remote_delete(filename): Fungsi ini mengirimkan perintah DELETE ke server untuk menghapus file tertentu berdasarkan nama yang diberikan.

```python
if __name__ == '__main__':
    command_handlers = {
        'LIST': remote_list,
        'GET': remote_get,
        'UPLOAD': remote_upload,
        'DELETE': remote_delete
    }

    while True:
        try:
            perintah = input("$ ").strip()
            if not perintah:
                continue

            tokens = shlex.split(perintah)
            cmd = tokens[0].upper() # Ambil perintah utama dan ubah ke huruf besar

            if cmd == 'EXIT':
                print("Keluar dari aplikasi.")
                break
            elif cmd in command_handlers:
                if cmd == 'LIST':
                    command_handlers[cmd]() # LIST tidak butuh parameter dari tokens
                elif len(tokens) >= 2: # GET, UPLOAD, DELETE butuh minimal 1 parameter (namafile)
                    command_handlers[cmd](tokens[1])
                else:
                    print(f"Format: {cmd} <namafile>")
            else:
                print("Perintah tidak dikenal.")
        except KeyboardInterrupt:
            print("\nKeluar dari aplikasi.")
            break
        except Exception as e:
            print(f"Terjadi kesalahan tak terduga: {e}")
```

Melalui CLI diatas, dapat langsung mengetikkan dan menjalankan berbagai perintah seperti LIST untuk menampilkan daftar file, GET <nama_file> untuk mengunduh file, UPLOAD <nama_file_lokal> untuk mengunggah file dari perangkat lokal, atau DELETE <nama_file> untuk menghapus file di server. Input diurai menggunakan modul shlex, yang memungkinkan penanganan string yang lebih robust, termasuk nama file yang mengandung spasi. Klien kemudian memanggil fungsi remote_ yang sesuai dengan perintah yang dimasukkan.

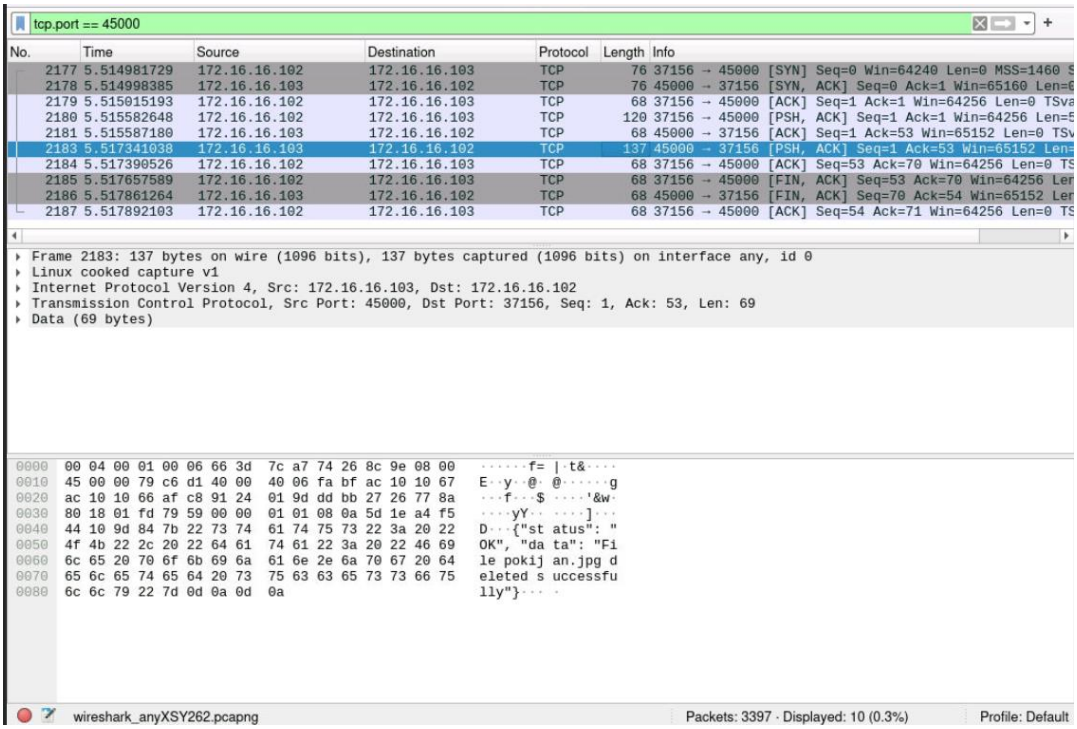
4) Capture Request di Wireshark

- Upload File



Komunikasi diawali dengan TCP 3-way handshake standar (paket SYN, SYN-ACK, ACK) yang berhasil membangun koneksi di port 45000. Setelah koneksi terbentuk, klien mengirimkan segmen TCP yang berisi data aplikasi. Analisis hex dump pada paket ini (khususnya Paket 2126) mengungkapkan bahwa klien mengirimkan perintah dalam format JSON. Isi JSON tersebut adalah {"command":"UPLOAD","params":["te st.txt","c2FkYXMKCg=="]}. Ini berarti klien sedang meminta server untuk mengunggah sebuah file. Nama file yang diunggah adalah "te st.txt" (perhatikan adanya spasi dalam nama file), dan konten file tersebut, setelah didekode dari Base64 ("c2FkYXMKCg=="), adalah teks "sada" diikuti oleh dua karakter baris baru. Server kemudian mengirimkan paket ACK sebagai tanda bahwa ia telah menerima data dari klien. Sesi komunikasi ditutup dengan klien memulai proses FIN-ACK.

- Delete File




Komunikasi ini dimulai dengan TCP 3-way handshake standar (paket SYN, SYN-ACK, ACK) yang berhasil membangun koneksi. Setelah koneksi terbentuk, server mengirimkan sebuah segmen TCP yang berisi data aplikasi (Paket 2183), ditandai dengan flag PSH (Push) dan ACK. Paket ini memiliki panjang data aplikasi (Len) sebesar 69 byte. Bagian hex dump dari Paket 2183 mengungkapkan bahwa server merespons dalam format JSON. Isi JSON tersebut adalah {"status": "OK", "data": "File pokijan.jpg deleted successfully"}. Ini menunjukkan bahwa server telah berhasil memproses suatu perintah, kemungkinan besar perintah DELETE untuk file "pokijan.jpg", dan mengonfirmasi keberhasilan operasi tersebut kepada klien. Klien kemudian mengakui penerimaan data ini dengan mengirimkan paket ACK (Paket 2184, 2186). Selanjutnya, server memulai proses penutupan koneksi dengan mengirimkan FIN flag (Paket 2185), yang diakui oleh klien (Paket 2187).