Nama : Alvin Zanua Putra

NRP : 5025231064

Kelas : Pemrograman Jaringan – D

**TUGAS 4**

Link Github Folder TUGAS34 :

<https://github.com/alvinzanuaputra/PROGJAR24/tree/main/assets/src/TUGAS4>

1. **Langkah-Langkah Pengerjaan :**

Pertama-tama kita modif dulu dari kode yang ada di keempat file (file\_server.py, file\_client\_cli\_test.py, file\_interface.py, file\_protocol.py) adalah sebagai berikut:

Setelah kita memodifikasi file\_server.py dengan menaikkan ukuran buffer menjadi 512 KB, sistem server menjadi lebih efisien saat menangani file besar, karena data lebih banyak ditampung dalam sekali recv() sehingga mempercepat proses upload dan download file. Server ini bisa dijalankan dalam tiga mode: single thread, thread pool, dan process pool. Ia menerima koneksi dari client, membaca perintah dalam format teks seperti UPLOAD, GET, atau LIST, lalu menjalankan aksi yang diminta. File disimpan dalam folder storage/ dan di-encode dalam base64 saat dikirim.

Untuk menangani file dengan lebih rapi dan modular, disediakan dua modul tambahan: file\_interface.py dan file\_protocol.py. Pada file\_interface.py, seluruh operasi terhadap file disediakan sebagai fungsi Python dalam bentuk metode class, misalnya:

* list() untuk melihat semua file di direktori,
* upload() untuk menyimpan file dari data base64,
* get() untuk membaca file dan mengembalikan isinya dalam base64,
* delete() untuk menghapus file.

Modul ini adalah abstraksi akses file lokal, dan sangat berguna agar server tidak perlu langsung berurusan dengan operasi file secara kasar. Kemudian file\_protocol.py adalah semacam jembatan logika yang mem-parsing perintah dari string mentah menjadi panggilan ke fungsi di FileInterface. Misalnya, ketika perintah "UPLOAD file.txt <data\_base64>" diterima, maka file\_protocol.py akan memecah perintah ini dan memanggil fungsi upload() di FileInterface. Hasil dari setiap perintah kemudian dikemas dalam format JSON agar mudah diproses oleh client. Lalu, untuk client, file\_client\_cli\_test.py menyediakan alat untuk berinteraksi dengan server dari sisi pengguna. File ini bisa dijalankan dari terminal dan mendukung beberapa mode:

* upload: mengirim file ke server.
* download: mengambil file dari server.
* list: melihat file yang ada di server.
* stress & stress\_all: melakukan stress testing dengan banyak worker (baik thread atau process) untuk menguji kecepatan server menangani banyak koneksi sekaligus. Hasilnya disimpan dalam CSV.

Client akan meng-encode file menjadi base64 sebelum mengirimkannya, dan menerima balasan dari server dalam format JSON. Ketika mendownload file, client akan menyimpan file hasil unduhan di lokal dengan prefix unduh\_.

Kesimpulannya, keempat file ini membentuk sistem file server-client lengkap:

* file\_server.py: menangani koneksi dan permintaan client.
* file\_interface.py: menyediakan fungsi dasar manipulasi file.
* file\_protocol.py: menjembatani string perintah ke fungsi logika file.
* file\_client\_cli\_test.py: antarmuka client dan alat uji performa.

Semua ini terhubung lewat protokol berbasis teks sederhana yang dibatasi dengan \r\n\r\n sebagai akhir perintah dan menggunakan encoding base64 untuk aman mengirimkan file biner melalui jaringan. Dapat dilihat dari struktur file dalam source github berikut : <https://github.com/alvinzanuaputra/PROGJAR24/tree/main/assets/src/TUGAS4>

1. **Selanjutnya menjalankan program file\_server.py dan file\_client\_cli\_test.py :**

**Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.**Pengujian ini merupakan bagian dari validasi sistem file server-client yang telah dibangun sebelumnya menggunakan empat buah program utama, yaitu file\_server.py, file\_client\_cli\_test.py, file\_protocol.py, dan file\_interface.py. Dalam pengujian ini, server dijalankan menggunakan perintah python3 file\_server.py --mode thread --workers 1, yang berarti server hanya menggunakan satu thread worker untuk menangani setiap koneksi dari client secara bergiliran. Server ini berperan sebagai penerima perintah seperti UPLOAD, GET, dan LIST, serta menyimpan file pada folder storage/.

Di sisi client, digunakan file\_client\_cli\_test.py yang berfungsi sebagai antarmuka pengguna untuk mengirim perintah ke server. Tiga file PDF berukuran 10MB, 50MB, dan 100MB yang berada di dalam folder files/ diunggah satu per satu ke server menggunakan perintah --mode upload dengan --pool\_mode thread --pool\_size 1. Ini berarti tiap file dikirim menggunakan satu thread client. Perintah ini secara internal memanggil fungsi upload\_remote() yang membaca file, mengubahnya menjadi base64, dan mengirim perintah UPLOAD ke server. Server kemudian memprosesnya menggunakan fungsi handle\_client() di file\_server.py, mendekode base64 dan menyimpan file ke storage/, serta mencatat aktivitas di log.

Setelah proses upload berhasil, dilanjutkan dengan proses download file dari server menggunakan perintah --mode download pada file yang sama. Perintah ini memicu fungsi download\_remote() yang mengirimkan perintah GET ke server. Server membaca file dari storage/, mengubahnya kembali ke base64, lalu mengirimkannya ke client. File hasil download disimpan di direktori utama proyek (bukan di dalam folder files atau storage) dengan awalan nama unduh\_.

**Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.**Seluruh alur ini bekerja berdasarkan pemrosesan perintah teks yang di-handle oleh file\_protocol.py, yang bertugas mem-parsing perintah menjadi panggilan metode logika file yang disediakan oleh file\_interface.py. File interface inilah yang bertanggung jawab langsung dalam melakukan operasi file seperti upload, get, dan list.

Dengan kombinasi 1 server worker dan 1 client thread, pengujian ini membuktikan bahwa sistem sudah mampu menangani file besar secara stabil, dan komunikasi antara komponen server-client bekerja sebagaimana mestinya. Penggunaan buffer recv(524288) dalam file\_server.py juga terbukti efektif mempercepat proses upload/download karena dapat menampung data lebih besar dalam satu kali terima.

Setelah Worker 1 ini selesai masih ada proses manual yang harus dijalankan secara berulang agar saat stress test nanti optimal yaitu melanjutkan kombinasi 54 multithread dan 54 multiprocess untuk command menjalankan worker selanjutnya dapat dilihat pada github :

Sebuah gambar berisi teks, cuplikan layar, Tidak bisa, Font

Konten yang dihasilkan AI mungkin salah.Sebuah gambar berisi teks, cuplikan layar, software

Konten yang dihasilkan AI mungkin salah.