Nama : Alvin Zanua Putra

NRP : 5025231064

Kelas : Pemrograman Jaringan – D

**TUGAS 2**

**Link Github File server\_thread.py** :

<https://github.com/alvinzanuaputra/PROGJAR24/blob/main/assets/src/TUGAS2/server_thread.py>

1. **Port 45000 dengan transport TCP**

Sebuah gambar berisi teks, cuplikan layar, Font

Konten yang dihasilkan AI mungkin salah.Jalankan mesin 1, kemudian ubah port yang ada di dalam file server\_thread.py dari ‘8889’ menjadi ‘45000’ dalam self.my\_socket.bind dalam class Server di bagian function def run (self) menjadi self.my\_socket.bind((‘0.0.0.0’, 45000)). Angka ‘0.0.0.0’ yang artinya server akan bind di semua interface server dan bisa diakses dari manapun tanpa mencantumkan port spesifik

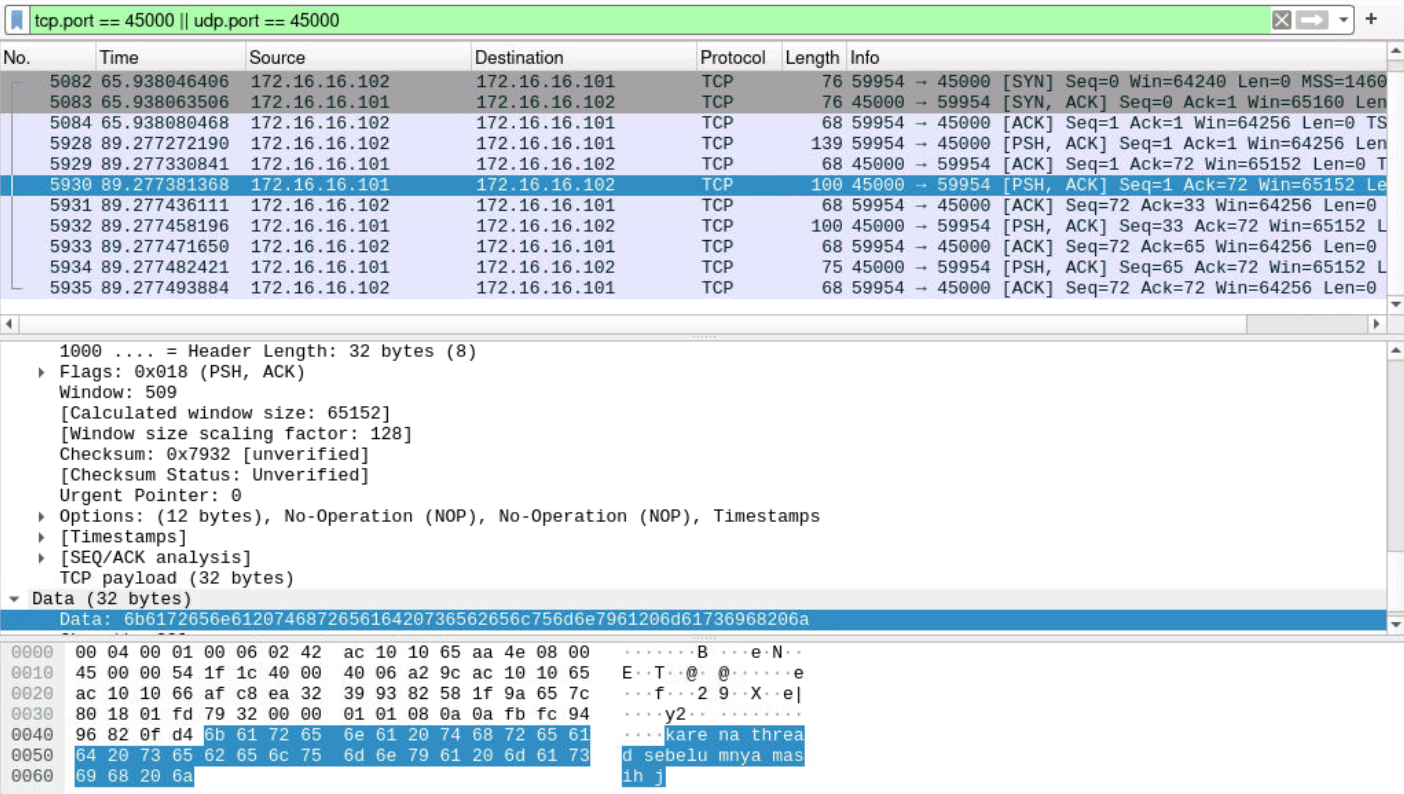
1. **Server harus dapat melayani request yang concurrent, gunakan contoh multithreading pada file server\_thread.py**

menunjukkan proses komunikasi antara server dan client menggunakan socket di Python. Pada koneksi kedua, server mengirimkan pesan ke client berupa **"karena thread sebelumnya masih jalan yaitu 2 thread ini multithreading"**, sebagai contoh output program untuk menunjukkan bahwa server sedang menangani lebih dari satu koneksi secara bersamaan dengan multithreading.

**Sebuah gambar berisi software, Software multimedia, teks, Perangkat lunak grafia

Konten yang dihasilkan AI mungkin salah.**

Capture Wireshark :



Pada hasil capture Wireshark, terlihat adanya komunikasi TCP antara dua mesin, yaitu dari IP **172.16.16.102** (client) menuju **172.16.16.101** (server) pada port **45000**. Proses TCP handshake berjalan sesuai prosedur, diawali dengan **SYN**, dilanjutkan **SYN-ACK**, lalu **ACK**, yang menandakan koneksi berhasil dibangun. Setelah itu, terdapat paket data yang dikirim dari server ke client dengan flag **PSH, ACK**, yang artinya server sedang mengirimkan data payload ke client sambil tetap menjaga koneksi terbuka. Hasil request terlihat Sebagian yaitu **“karena thread sebelumnya masih j”** menandakan bahwa multithreading berhasil dilakukan antar server.

1. **Ketentuan request yang dilayani : Diawali dengan string “TIME dan diakhiri dengan karakter 13 dan karakter 10”. Setiap request dapat diakhiri dengan string “QUIT” yang diakhiri dengan karakter 13 dan 10**

Disini saya menambahkan pada bagian def run (self) : logging agar server mencetak isi request **dalam bentuk byte** (untuk membuktikan \r\n = \x0d\x0a atau 13 10).

  def run(self):

                while True:

                        data = *self*.connection.recv(1024)

                        if not data:

                                break

                        logging.warning(f"Raw data dari client: {data} ({list(data)})")  # log byte

                        message = data.decode()

                        # Respons hanya jika diakhiri \r\n

                        if message.endswith("\r\n"):

*self*.connection.sendall(b"DITERIMA\r\n")

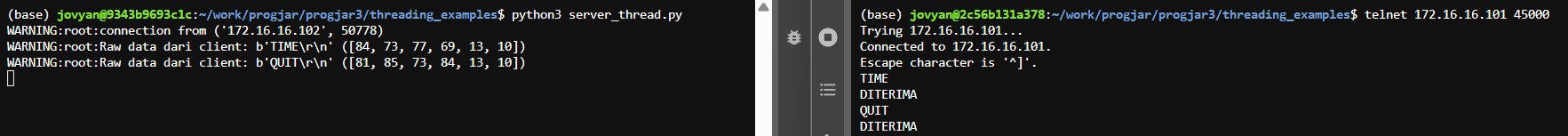
                        else:

*self*.connection.sendall(b"DITOLAK: Tidak diakhiri CRLF\r\n")

                        if message.strip() == "QUIT":

                                break

**Sebuah gambar berisi teks, elektronik, cuplikan layar, software

Konten yang dihasilkan AI mungkin salah.**Apa pun isi pesannya, selama diakhiri \r\n, server membalas "DITERIMA\r\n". Jika tidak, balas "DITOLAK: Tidak diakhiri CRLF\r\n". Jika isi "QUIT", server keluar.

Server ini hanya menerima permintaan (request) dari client jika pesan dimulai dengan kata **"TIME"** dan diakhiri dengan dua karakter khusus: **\r\n** yang artinya "carriage return" dan "line feed" (kode ASCII 13 dan 10). Ini adalah standar akhir baris dalam banyak protokol jaringan. Jika format tidak persis seperti itu, server akan menolak dengan pesan error. Client juga bisa mengakhiri sesi dengan mengirim pesan **"QUIT\r\n"**, yang memberitahu server untuk mengirim "Goodbye" lalu menutup koneksi. Dibuktikan dengan adanya format request “TIME” diikuti . . yang berarti ada respons 2 digit yaitu \r\n.

1. **Server akan merespon dengan jam dengan ketentuan, Dalam bentuk string (UTF-8) Diawali dengan “JAM<spasi><jam>”<jam> berisikan info jam dalam format “hh:mm:ss” dan diakhiri dengan karakter 13 dan karakter 10**

if not data:

  break

    logging.warning(f"Raw data dari client: {data} ({list(data)})")  # Tambahan ini

                        message = data.decode()

                        if message.startswith("TIME") and message.endswith("\r\n"):

                                now = datetime.now()

                                response = now.strftime("JAM %d %m %y %H:%M:%S\r\n")

*self*.connection.sendall(response.encode())

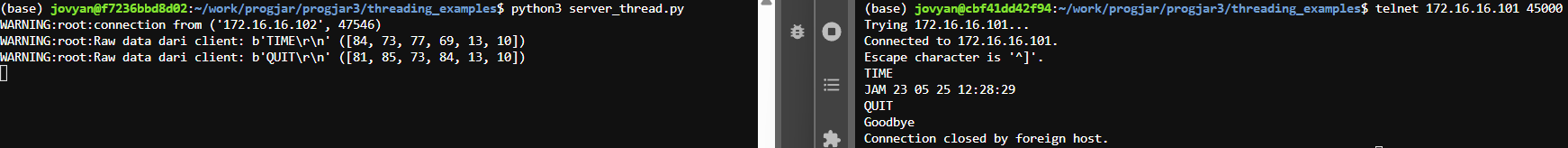
                        elif message.startswith("QUIT") and message.endswith("\r\n"):

*self*.connection.sendall(b"Goodbye\r\n")

                                breaK

                        else:

*self*.connection.sendall(b"ERROR: Format tidak dikenali\r\n")

Server hanya menerima request dengan format \r\n (kode 13 dan 10). Salah satu karakter saja (\n atau \r saja) akan ditolak.

**Sebuah gambar berisi teks, cuplikan layar, software, nomor

Konten yang dihasilkan AI mungkin salah.Pada Capture Wireshark :**

Server ini hanya menerima permintaan (request) dari client jika pesan dimulai dengan kata **"TIME"** dan diakhiri dengan dua karakter khusus: **\r\n** yang berarti *carriage return* dan *line feed* (kode ASCII 13 dan 10). Ini adalah standar penanda akhir baris dalam banyak protokol jaringan. Jika format tidak sesuai (misalnya hanya \n), server akan menolak permintaan dengan mengirimkan pesan error. Selain itu, client dapat mengakhiri sesi dengan mengirimkan **"QUIT\r\n"**, yang akan dibalas oleh server dengan **"Goodbye"** dan menutup koneksi.

Sebagai respon terhadap permintaan **"TIME\r\n"**, server mengirimkan waktu saat ini dalam format UTF-8, dimulai dengan string **"JAM "** diikuti oleh waktu dalam format **"hh:mm:ss"**, dan diakhiri dengan **\r\n**. Format waktu ini dihasilkan menggunakan modul datetime, khususnya dengan datetime.now().strftime("%H:%M:%S"). Contoh balasan dari server adalah: **JAM 12:28:29\r\n**