Nama : Muhammad Adam Jhansi

Kelas : 4IA13

NPM : 51421657

Socket Programing

Dokumentasi dan Alur Kerja Program

tcp_server.py

- **import socket, import threading, import tkinter as tk, from tkinter.scrolledtext import ScrolledText**: Mengimpor library yang dibutuhkan. socket untuk operasi jaringan, threading untuk menangani banyak koneksi secara bersamaan (meskipun dalam kode ini hanya menangani satu client), tkinter untuk membuat GUI, dan ScrolledText untuk area teks yang bisa di-scroll.
- **start_server() function**: Fungsi utama untuk memulai server.
    - server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM): Membuat objek socket TCP/IP. socket.AF_INET menunjukkan penggunaan alamat IPv4, dan socket.SOCK_STREAM menunjukkan penggunaan protokol TCP.
    - server_socket.bind(('127.0.0.1', 9000)): Mengikat socket ke alamat IP dan port tertentu. '127.0.0.1' adalah alamat loopback (localhost), yang berarti server hanya akan menerima koneksi dari komputer yang sama. 9000 adalah nomor port yang digunakan untuk komunikasi.
    - server_socket.listen(1): Mulai mendengarkan koneksi yang masuk. Argumen 1 menunjukkan bahwa server akan menerima hingga 1 koneksi yang tertunda.
    - log.insert(tk.END, "Menunggu koneksi client...\n"): Menampilkan pesan di area log server bahwa server sedang menunggu koneksi.

- conn, addr = server_socket.accept(): Menerima koneksi yang masuk. accept() akan memblokir eksekusi hingga ada client yang mencoba terhubung. Setelah koneksi berhasil, ia mengembalikan objek koneksi (conn) yang mewakili koneksi dengan client, dan alamat client (addr).
- log.insert(tk.END, f"Terhubung dengan {addr}\n"): Menampilkan alamat client yang berhasil terhubung.
- **receive() function (nested)**: Fungsi yang berjalan di thread terpisah untuk terus-menerus menerima data dari client.
    - while True:: Loop tak terbatas untuk terus menerima data.
    - try...except: Blok try-except untuk menangani kemungkinan kesalahan koneksi terputus.
    - data = conn.recv(1024).decode(): Menerima hingga 1024 byte data dari client dan mendekodenya menjadi string menggunakan encoding default (biasanya UTF-8).
    - if not data:: Jika tidak ada data yang diterima (koneksi ditutup oleh client), loop akan berhenti menggunakan break.
    - log.insert(tk.END, f"Client: {data}\n"): Menampilkan pesan yang diterima dari client di area log server.
- threading.Thread(target=receive, daemon=True).start(): Membuat dan memulai thread baru yang menjalankan fungsi receive. daemon=True berarti thread akan berhenti ketika thread utama (GUI) ditutup.
- **send_message() function (nested)**: Fungsi untuk mengirim pesan dari server ke client.
    - msg = entry.get(): Mendapatkan teks dari entry field GUI.
    - conn.sendall(msg.encode()): Mengenkode pesan menjadi byte dan mengirimkannya ke client menggunakan koneksi conn. sendall() memastikan semua data terkirim.

        - log.insert(tk.END, f"Server: {msg}\n"): Menampilkan pesan yang dikirim oleh server di area log server.
        - entry.delete(0, tk.END): Menghapus teks di entry field setelah pesan dikirim.
    - send_btn.config(command=send_message): Mengkonfigurasi tombol "Kirim" untuk memanggil fungsi send_message() ketika diklik.
- **GUI Server**: Bagian kode yang membuat antarmuka pengguna server menggunakan tkinter.
    - window = tk.Tk(): Membuat window utama.
    - window.title("TCP Server"): Menetapkan judul window.
    - log = ScrolledText(...): Membuat area teks yang bisa di-scroll untuk menampilkan log komunikasi.
    - entry = tk.Entry(...): Membuat input field untuk menulis pesan yang akan dikirim.
    - send_btn = tk.Button(...): Membuat tombol untuk mengirim pesan.
    - threading.Thread(target=start_server, daemon=True).start(): Membuat dan memulai thread untuk menjalankan fungsi start_server() agar server tidak membekukan GUI saat menunggu koneksi.
    - window.mainloop(): Memulai loop utama GUI yang akan terus berjalan hingga window ditutup.

tcp_client.py

- **import socket, import threading, import tkinter as tk, from tkinter.scrolledtext import ScrolledText**: Mengimpor library yang dibutuhkan, sama seperti server.
- **client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)**: Membuat objek socket TCP/IP.

- **connect() function**: Fungsi untuk mencoba terhubung ke server.
    - try...except: Blok try-except untuk menangani kemungkinan kesalahan saat mencoba terhubung.
    - client_socket.connect(('127.0.0.1', 9000)): Mencoba membuat koneksi ke server dengan alamat IP '127.0.0.1' dan port 9000.
    - log.insert(tk.END, "Terhubung ke server...\n"): Menampilkan pesan bahwa client berhasil terhubung ke server.
    - **receive() function (nested)**: Fungsi yang berjalan di thread terpisah untuk terus-menerus menerima data dari server. Logikanya mirip dengan fungsi receive() di server.
    - threading.Thread(target=receive, daemon=True).start(): Membuat dan memulai thread untuk menjalankan fungsi receive.
    - except Exception as e:: Menangkap berbagai kemungkinan kesalahan selama proses koneksi dan menampilkan pesan error di log client.
- **send_message() function**: Fungsi untuk mengirim pesan dari client ke server. Logikanya mirip dengan fungsi send_message() di server.
- **GUI Client**: Bagian kode yang membuat antarmuka pengguna client menggunakan tkinter. Struktur dan elemen GUI mirip dengan server.
- connect(): Memanggil fungsi connect() segera setelah GUI client dibuat untuk mencoba terhubung ke server.
- window.mainloop(): Memulai loop utama GUI client.

Logika Program

1. **Server Initialization:**
    - Server membuat socket dan mengikatnya ke alamat IP dan port tertentu.
    - Server mulai mendengarkan koneksi yang masuk.

- Server menampilkan pesan "Menunggu koneksi client...".

2. **Client Connection:**
    - Client membuat socket.
    - Client mencoba terhubung ke alamat IP dan port server.
    - Jika koneksi berhasil, client menampilkan pesan "Terhubung ke server...".
    - Server menerima koneksi dari client dan menampilkan alamat client yang terhubung.
    - Sebuah objek koneksi (conn di server, client_socket setelah terhubung di client) dibuat untuk komunikasi dua arah antara server dan client.
3. **Two-way Communication:**
    - **Menerima Pesan (Server):** Sebuah thread di server terus-menerus menunggu data dari client menggunakan conn.recv(). Ketika data diterima, ia mendekodenya dan menampilkannya di log server. Jika koneksi terputus, thread akan berhenti.
    - **Mengirim Pesan (Server):** Ketika pengguna mengetik pesan di entry field server dan menekan tombol "Kirim", fungsi send_message() dipanggil. Pesan dienkode dan dikirim ke client menggunakan conn.sendall(), dan pesan tersebut juga ditampilkan di log server.
    - **Menerima Pesan (Client):** Sebuah thread di client terus-menerus menunggu data dari server menggunakan client_socket.recv(). Ketika data diterima, ia mendekodenya dan menampilkannya di log client. Jika koneksi terputus, thread akan berhenti.
    - **Mengirim Pesan (Client):** Ketika pengguna mengetik pesan di entry field client dan menekan tombol "Kirim", fungsi send_message() dipanggil. Pesan dienkode dan dikirim ke server menggunakan client_socket.sendall(), dan pesan tersebut juga ditampilkan di log client.
4. **GUI Interaction:**

- Kedua program menggunakan tkinter untuk membuat jendela dengan area log (ScrolledText), input field (Entry), dan tombol kirim (Button).
- Log area digunakan untuk menampilkan informasi status koneksi dan pesan yang diterima/dikirim.
- Input field digunakan untuk menulis pesan yang akan dikirim.
- Tombol kirim memicu fungsi pengiriman pesan.
- Penggunaan thread (threading.Thread) memastikan bahwa operasi jaringan (menunggu koneksi, menerima data) tidak membekukan antarmuka pengguna, sehingga GUI tetap responsif.

Alur Kerja Program TCP Client dan Server

1. Inisialisasi Socket Server:

- Server membuat socket TCP (`AF_INET`, `SOCK_STREAM`)
- Binding ke alamat IP `127.0.0.1` dan port `9000`
- Mulai mendengarkan koneksi menggunakan `listen()`

2. Menunggu dan Menerima Koneksi Client:

- Server menggunakan `accept()` untuk menerima satu client
- Jika client terhubung, alamat client dicetak ke log dan console

3. Thread untuk Menerima Pesan dari Client:

- Thread baru dibuat di server untuk menerima pesan dari client secara terus-menerus
- Menggunakan `recv()` untuk membaca data dari socket

4. Pengiriman Pesan dari Server:

- Pesan diketik di GUI `Entry` server dan dikirim melalui tombol "Kirim"
- Pesan dikirim ke client menggunakan `sendall()`

5. Inisialisasi Socket Client:

- Client membuat socket TCP
- Client menghubungkan diri ke server `127.0.0.1:9000` menggunakan `connect()`

6. Thread untuk Menerima Pesan dari Server:

- Setelah terkoneksi, client membuat thread untuk menerima pesan dari server
- Pesan ditampilkan di GUI dan juga di console

7. Pengiriman Pesan dari Client:

- Client mengetikkan pesan dan mengirimkannya menggunakan tombol "Kirim"
- Pesan dikirim ke server menggunakan `sendall()`

8. Antarmuka GUI:

- Menggunakan `tkinter` untuk membangun tampilan

- Log pesan menggunakan `ScrolledText`
- Input pesan menggunakan `Entry`
- Tombol kirim mengirimkan pesan yang diketik

9. Output ke Console:

- Setiap pesan masuk dan keluar juga dicetak menggunakan `print()`
- Berguna untuk debugging dan log tambahan di terminal

10. Penanganan Error:

- Semua operasi socket dilindungi oleh blok `try-except`
- Thread daemon menjaga agar aplikasi tetap berjalan tanpa crash

Catatan: Program ini hanya mendukung satu client yang terhubung ke server