Nama : Alpian Roymundus Siringo-ringo

NIM : 11211009

Sistem Terdistribusi A

Tugas 2

```
import tkinter as tk
from tkinter import messagebox
import time

class SimulasiSistemTerdistribusi:
    def __init__(self, root):
        self.root = root
        self.root.title("Simulasi Model Komunikasi Terdistribusi")

        # Kanvas untuk representasi visual
        self.kanvas = tk.Canvas(self.root, width=600, height=400)
        self.kanvas.pack()

        # Buat komponen visual
        self.buat_komponen()

        # Tombol untuk memicu simulasi
        self.buat_tombol_simulasi()

    def buat_komponen(self):
        # Membuat representasi server, klien, dan broker
        self.server = self.kanvas.create_rectangle(250, 50, 350, 100,
fill="lightblue", tags="server")
        self.kanvas.create_text(300, 75, text="Server")

        self.klien1 = self.kanvas.create_rectangle(50, 300, 150, 350,
fill="lightgreen", tags="klien1")
        self.kanvas.create_text(100, 325, text="Klien 1")

        self.klien2 = self.kanvas.create_rectangle(450, 300, 550, 350,
fill="lightgreen", tags="klien2")
        self.kanvas.create_text(500, 325, text="Klien 2")

        # Broker untuk publish-subscribe
        self.broker = self.kanvas.create_rectangle(250, 200, 350, 250,
fill="lightyellow", tags="broker")
        self.kanvas.create_text(300, 225, text="Broker")
```

```
    def buat_tombol_simulasi(self):
        # Membuat tombol simulasi
        tombol_frame = tk.Frame(self.root)
        tombol_frame.pack(pady=10)

        # Tombol-tombol simulasi
        tk.Button(tombol_frame, text="Simulasi Request-Response",
command=self.simulasi_request_response).pack(side=tk.LEFT, padx=10)
        tk.Button(tombol_frame, text="Simulasi Publish-Subscribe",
command=self.simulasi_publish_subscribe).pack(side=tk.LEFT, padx=10)
        tk.Button(tombol_frame, text="Simulasi Message Passing",
command=self.simulasi_message_passing).pack(side=tk.LEFT, padx=10)
        tk.Button(tombol_frame, text="Simulasi RPC",
command=self.simulasi_rpc).pack(side=tk.LEFT, padx=10)

    def simulasi_request_response(self):
        self.perbarui_status("Klien 1 mengirimkan Request ke Server...")
        self.animasi_pesan("klien1", "server", "Request")
        self.tunda_pemrosesan()
        self.perbarui_status("Server mengirimkan Response ke Klien 1...")
        self.animasi_pesan("server", "klien1", "Response")

    def simulasi_publish_subscribe(self):
        self.perbarui_status("Klien 1 (Publisher) mengirimkan Publish ke
Broker...")
        self.animasi_pesan("klien1", "broker", "Publish")
        self.tunda_pemrosesan()
        self.perbarui_status("Broker memberi notifikasi ke Klien 2
(Subscriber)...")
        self.animasi_pesan("broker", "klien2", "Notify")

    def simulasi_message_passing(self):
        self.perbarui_status("Klien 1 mengirimkan Pesan ke Klien 2...")
        self.animasi_pesan("klien1", "klien2", "Pesan")
        self.tunda_pemrosesan()
        self.perbarui_status("Klien 2 menerima Pesan dari Klien 1.")

    def simulasi_rpc(self):
        self.perbarui_status("Klien 1 memanggil RPC di Server...")
        self.animasi_pesan("klien1", "server", "RPC: sum(5, 3)")
        self.tunda_pemrosesan()
        self.perbarui_status("Server mengeksekusi RPC dan mengembalikan
hasil...")
```

```
        self.animasi_pesan("server", "klien1", "Hasil: 8")
        self.perbarui_status("RPC selesai. Hasil diterima oleh Klien 1.")

    def animasi_pesan(self, pengirim, penerima, pesan):
        koordinat_pengirim = self.kanvas.bbox(pengirim)
        koordinat_penerima = self.kanvas.bbox(penerima)

        # Membuat teks pesan yang bergerak dari pengirim ke penerima
        teks_pesan = self.kanvas.create_text(koordinat_pengirim[0] + 50,
koordinat_pengirim[1] - 20, text=pesan, fill="red")

        # Menghitung langkah-langkah untuk animasi
        jarak = ((koordinat_penerima[0] - koordinat_pengirim[0]) ** 2 +
(koordinat_penerima[1] - koordinat_pengirim[1]) ** 2) ** 0.5
        langkah = int(jarak / 5)
        x_diff = (koordinat_penerima[0] - koordinat_pengirim[0]) / langkah
        y_diff = (koordinat_penerima[1] - koordinat_pengirim[1]) / langkah

        for _ in range(langkah):
            self.kanvas.move(teks_pesan, x_diff, y_diff)
            self.root.update()
            time.sleep(0.02)

        # Setelah mencapai penerima, hapus pesan
        self.kanvas.delete(teks_pesan)

    def perbarui_status(self, status):
        self.kanvas.delete("status")
        self.kanvas.create_text(300, 20, text=status, fill="black",
tags="status")

    def tunda_pemrosesan(self):
        self.root.update()
        time.sleep(1)

# Menjalankan aplikasi
if __name__ == "__main__":
    root = tk.Tk()
    app = SimulasiSistemTerdistribusi(root)
    root.mainloop()
```

A. Pemilihan Model Komunikasi

Pada simulasi, terdapat 4 model simulasi yang disimulasikan, yaitu:

1. **Request-Response**: Di mana satu klien mengirimkan request ke server, dan server memberikan response. Ini adalah pola komunikasi sinkron yang umum digunakan dalam aplikasi web.
2. **Publish-Subscribe**: Di mana satu entitas (Publisher) mengirimkan pesan ke broker, dan broker mendistribusikan pesan tersebut ke entitas lain yang berlangganan (Subscriber).
3. **Message Passing**: Di mana dua klien berkomunikasi langsung tanpa perantara (broker atau server), menunjukkan pola peer-to-peer (P2P) dalam komunikasi terdistribusi.
4. **Remote Procedure Call (RPC)**: Di mana satu klien meminta eksekusi fungsi atau prosedur di server, dan server mengembalikan hasil setelah eksekusi selesai.

   Model-model komunikasi ini dipilih untuk menunjukkan berbagai pola interaksi antara komponen sistem terdistribusi, mulai dari pola sinkron (Request-Response dan RPC) hingga pola asinkron (Publish-Subscribe dan Message Passing).

B. Komponen Sistem

Sistem terdiri dari beberapa komponen utama yang merepresentasikan entitas dalam sistem terdistribusi, yaitu:

1. **Server**: Bertanggung jawab untuk memproses permintaan dari klien dalam model Request-Response dan RPC.
2. **Klien**: Ada dua klien dalam simulasi ini, Klien 1 dan Klien 2. Masing-masing klien berinteraksi dengan server atau dengan broker dalam berbagai model komunikasi.
3. **Broker**: Berfungsi sebagai perantara dalam model Publish-Subscribe, di mana ia menerima pesan dari publisher dan mendistribusikannya ke subscriber.

   Komponen-komponen ini berfungsi sebagai simpul dalam sistem terdistribusi yang mengirim dan menerima pesan satu sama lain sesuai dengan model komunikasi yang dipilih.

C. Implementasi Logika Interaksi

Simulasi ini meniru cara kerja model komunikasi dalam sistem terdistribusi melalui interaksi antara komponen yang telah ditentukan: server, klien, dan broker. Setiap model komunikasi (Request-Response, Publish-Subscribe, Message Passing, dan Remote Procedure Call - RPC) diimplementasikan dengan logika yang berbeda, dan setiap model menunjukkan bagaimana pesan dikirim, diterima, dan diproses.

1. Simulasi Request-Response

Pada model ini, logika komunikasi mengikuti pola klasik client-server, di mana klien mengirim permintaan (request) dan server memberikan balasan (response):

- Langkah Pertama (Request): Klien 1 mengirimkan request ke server. Dalam kode, ini divisualisasikan oleh fungsi animasi_pesan(*"klien1", "server", "Request"*), yang menampilkan teks bergerak dari posisi Klien 1 ke posisi Server.
- Tunda Pemrosesan: Simulasi memberi jeda sejenak untuk meniru pemrosesan di server dengan fungsi *tunda_pemrosesan()*, yang menyebabkan sistem menunggu selama satu detik, seolah-olah server sedang memproses permintaan.
- Langkah Kedua (Response): Setelah pemrosesan selesai, server mengirimkan respons kembali ke Klien 1, divisualisasikan oleh *animasi_pesan("server", "klien1", "Response").* Pesan "Response" bergerak dari server ke klien.

2. Simulasi Publish-Subscribe

Model ini menunjukkan komunikasi asinkron, di mana publisher (Klien 1) mengirimkan pesan ke broker, dan broker kemudian mendistribusikan pesan tersebut kepada subscriber (Klien 2).

- **Langkah Pertama (Publish):** Klien 1 bertindak sebagai publisher yang mengirimkan pesan ke broker. Ini divisualisasikan dengan animasi_pesan("klien1", "broker", "Publish"), di mana pesan bergerak dari Klien 1 ke Broker.
- **Tunda Pemrosesan**: Setelah pesan diterima oleh broker, ada jeda sejenak untuk meniru pemrosesan, mirip dengan model request-response.
- **Langkah Kedua (Notify):** Broker memberi notifikasi ke subscriber (Klien 2) tentang pesan yang diterima. Ini dilakukan dengan animasi_pesan("broker", "klien2", "Notify"), menunjukkan pesan bergerak dari broker ke Klien 2.

3. Simulasi Message Passing

Model ini menggambarkan komunikasi langsung antar-node, di mana pesan dikirimkan antara dua klien tanpa perantara.

- Langkah Pertama (Send Message): Klien 1 mengirim pesan langsung ke Klien 2, tanpa melalui server atau broker. Ini divisualisasikan dengan animasi_pesan("klien1", "klien2", "Pesan"), yang menunjukkan pergerakan pesan langsung dari Klien 1 ke Klien 2..
- Tunda Pemrosesan: Setelah pesan sampai, sistem menunggu sejenak untuk meniru penerimaan dan pemrosesan pesan di Klien 2.
- Langkah Kedua (Return Result): Klien 2 menerima pesan dari Klien 1. Simulasi ini

menyederhanakan proses pengiriman dan penerimaan pesan tanpa melibatkan entitas pihak ketiga.

Model ini menggambarkan komunikasi peer-to-peer yang sangat efisien, di mana pesan dikirim langsung dari pengirim ke penerima.

4. Simulasi Remote Procedure Call (RPC)

Model RPC memungkinkan klien untuk memanggil fungsi di server, dengan server mengeksekusi fungsi tersebut dan mengembalikan hasil ke klien. Model ini sering digunakan dalam sistem mikroservis.

- Langkah Pertama (RPC Call): Klien 1 memanggil fungsi di server melalui RPC. Dalam simulasi, fungsi yang dipanggil adalah *sum(5, 3)*. Ini divisualisasikan oleh *animasi_pesan("klien1", "server", "RPC: sum(5, 3)")*, yang menunjukkan bahwa Klien 1 mengirimkan permintaan untuk menjalankan fungsi tersebut di server.
- Tunda Pemrosesan: Server memproses permintaan dan menjalankan fungsi, yang diwakili oleh jeda pemrosesan dengan tunda_pemrosesan().
- Langkah Kedua (Return Result): Setelah fungsi selesai dieksekusi, server mengembalikan hasil (dalam kasus ini, hasil *sum(5, 3)* adalah 8) ke Klien 1. Pesan ini divisualisasikan oleh *animasi_pesan("server", "klien1", "Hasil: 8")*, yang menunjukkan hasil dikirimkan kembali ke Klien 1.

RPC menggambarkan model sinkron yang efisien untuk menjalankan proses di server dengan cara yang mirip dengan pemanggilan fungsi lokal di klien.

D. Representasi Visual

Representasi visual menggunakan Tkinter Canvas untuk membuat dan menampilkan elemen-elemen berikut:

1. **Server, Klien, dan Broker**: Ditampilkan sebagai kotak dengan warna berbeda di kanvas, yang memudahkan pengguna untuk memahami peran masing-masing entitas.
2. **Pesan**: Ketika simulasi dimulai, pesan digambarkan sebagai teks yang bergerak dari satu entitas ke entitas lainnya. Sebagai contoh, dalam model Request-Response, teks pesan "Request" akan bergerak dari Klien 1 ke Server, dan "Response" akan bergerak kembali dari Server ke Klien 1.
3. **Status**: Di atas kanvas, status simulasi juga ditampilkan dalam bentuk teks, menjelaskan apa yang sedang terjadi (misal: "Klien 1 mengirimkan Request ke Server...").

E. Desain Interaksi Pengguna

Pada simulasi ini, pengguna bisa memilih model komunikasi ingin mereka simulasikan melalui beberapa tombol yang disediakan di bawah kanvas:

- Tombol-tombol tersebut memicu fungsi yang sesuai untuk memulai simulasi dari masing-masing model komunikasi (Request-Response, Publish-Subscribe, Message Passing, dan RPC).
- Pengguna juga dapat melihat status proses di atas kanvas, yang diperbarui sesuai dengan tahap simulasi yang sedang berlangsung.

F. Mekanisme Perbandingan

Dalam simulasi ini, kita bisa membandingkan beberapa aspek dari model komunikasi:

1. **Kecepatan Komunikasi**: Setiap model memiliki waktu tunggu (time.sleep) yang dapat disesuaikan untuk mensimulasikan perbedaan dalam latensi antara model-model komunikasi.
2. **Urutan Pesan**: Kita bisa melihat bagaimana urutan pengiriman pesan dan respons berbeda antara model sinkron (Request-Response dan RPC) dan model asinkron (Publish-Subscribe dan Message Passing).
3. Jumlah Pesan: Dalam model seperti Publish-Subscribe, kita bisa mengamati bagaimana satu pesan dari publisher dapat menghasilkan beberapa notifikasi (berlangganan).