Assignment: Design of a Microkernel System

This assignment is designed to assess your ability to design a robust, secure, and highly flexible architecture based on the **Microkernel Pattern**. The goal is to achieve maximum isolation and easy extension without altering the core system.

1. Problem Description (Context and Focus)

The organization is developing a central platform, referred to as the **Microkernel**, which must achieve near-perfect fault isolation.

The Microkernel's primary responsibility must be strictly limited to managing the essential resources required for the system's operation, primarily **Inter-Process Communication (IPC)** and basic process management.

All specific business logic, data transformations, and even core services (like user authentication or database access) must be implemented as separate, independent **Plugin Modules (Servers)** running in their own isolated memory spaces (processes). This ensures that a failure in any single service cannot crash the core or affect other services.

2. Microkernel System Design Requirements

You are to design the architecture for the **Microkernel** with the following key functionalities:

Core Responsibilities (Minimalist Core)

1. **Process Management:** The Microkernel must be responsible for launching, monitoring, and terminating Plugin Processes.
2. **IPC Mechanism:** The core function is to facilitate secure communication between the Microkernel and the Plugin Processes, and indirectly between Plugins themselves.
3. **Plugin Loading and Initialization:** The Microkernel must be able to launch and initialize plugin processes upon startup.

Plugin Contract and Communication

4. **Plugin Contract Enforcement:** The Kernel must ensure that every plugin process adheres to a specific, well-defined communication contract (e.g., a message format).
5. **Event Distribution/Orchestration (via IPC):** The Microkernel must have a reliable mechanism for sending Events (Messages) to interested Plugin Processes using IPC (e.g., sockets, message queues). Example: When a new

user logs in, a UserLoggedInEvent message is sent, which all interested plugins receive and process in their own isolated environment.

6. **Isolation and Error Handling:** A failure in one Plugin Process (e.g., a crash) **must** be contained within that process and **not** take down the Microkernel or other Plugin Processes.

Technical Guidelines

- **Language/Framework:** Choose the language/framework best suited for demonstrating process management and IPC (e.g., C#, C++ or Python). Provide justification for your choice.
- **Dependencies:** Plugin Processes should not have direct knowledge of, or dependency on, one another's memory or code. All communication must pass through the Microkernel's IPC mechanism.

3. Plugin Design Requirements

You are required to describe and include an example implementation of one plugin:

- **Plugin (Server):** A simple "Logging and Metric Plugin" (MetricsLoggerProcess) that runs as a **separate process** and connects to the Microkernel via the defined IPC channel.
    - It must listen for UserLoggedInEvent and DataProcessedEvent.
    - Upon receiving the event via IPC, it should log the deserialized event data to a mock database/console and increment an internal counter.

4. Deliverables and Documentation (The Architecture Package)

Your submission must consist of two parts:

A. Code Deliverable (Minimal but Demonstrative)

- A functional (though simplified) implementation of the **Microkernel** demonstrating process launching and IPC setup.
- The Interface(s)/Contract(s) defining the message structure for IPC.
- A functional implementation of the example plugin process (MetricsLoggerProcess).

B. Architecture Documentation (Essential) PDF

This is the most crucial part of the deliverable and must be presented in a structured format.

1. Architectural Viewpoint

- **Architecture Diagram (C4/UML/Component):** A high-level diagram illustrating the main components (**Microkernel**, **Plugin Processes**, IPC Channels, Message Contracts, Data Sources) and the flow of messages/events.
- **Decisions and Justifications (ASR – Architectural Significant Requirements):** Document the most important architectural choices you have made and why.
    - *Example:* Choice of IPC mechanism (e.g., Named Pipes vs. TCP Sockets vs. Shared Memory).
    - *Example:* Justification for the chosen language based on its support for concurrency and processes.

2. Quality Attributes

- **Modifiability/Extensibility:** Explain *how* the architecture ensures that a new plugin process can be added without modifying the Microkernel or other existing plugin processes.
- **Robustness/Fault Tolerance:** Describe the mechanisms you have implemented (or designed) to ensure that a crash (failure) in one Plugin Process does not affect the others, specifically focusing on **process isolation and IPC failure handling**.

3. Technology and Implementation Details

- **IPC Protocol Definition:** Detailed description of the message format (e.g., JSON, Protocol Buffers) used over the IPC channel.
- **Orchestration:** Describe the step-by-step process of how the Microkernel handles an incoming event, focusing on the cross-process communication:
    1. Microkernel receives Event X (e.g., from a web request).
    2. Microkernel serializes Event X into the IPC protocol.
    3. Microkernel transmits the message to all registered Plugin Processes via the IPC channel.
    4. Plugin Processes deserialize the message and execute the processing logic.

Assessment: The architect will be assessed based on the clarity of the documentation, the design's adherence to Microkernel principles, and how effectively the architecture utilizes IPC to achieve superior isolation and security.