75.74 Sistemas Distribuidos I

Criterios de Corrección

General

- Implica **reentrega**:
    - Si no emplea alguno de los lenguajes canónicos para la materia: golang y Python.
    - Si se accede a secciones críticas sin un mecanismo de sincronización (ej. mutex, locks).
    - Si falta el chequeo de cant. de bytes leídos/escritos en FDs (sockets, archivos).
    - Si falta el cierre de FDs (sockets, archivos).
- Implica **observación o quita de puntos** (dependiendo del contexto, gravedad, cantidad de ocurrencias, etc):
    - Si no se cumplen las buenas prácticas de codificación
    - Si existen gran cantidad de código repetido (ej. no se encapsula la lógica común en paquetes *common* o *utils*).
    - Si no se utilizan archivos de config (o de constantes al menos) y en su lugar se hardcodean valores.
    - Si se abusa de funciones largas (depende del lenguaje de programación, ej. más de 100 líneas es un code smell en Python).
    - Si faltan comentarios en código de alta complejidad.
    - Si hay más de un coding standard en los files que componen el TP.
    - Si no implementación de graceful shutdowns en los procesos.
    - Si no hay un README mínimo que indique cómo levantar y correr el sistema.
    - Si no se proveen todos los archivos para poder correr la entrega.

TP0: Concurrencia y Comunicaciones

- Implica **desaprobación**:
    - **No entregar**
    - Si no se implementa un protocolo de comunicación, cualquiera sea este.

    - Si no se implementa una solución multithreading o multiprocessing para brindar escalabilidad en los ejercicios requeridos.
    - **Si se usan librerías para resolver la comunicación en lugar de sockets.**
    - **Si se usan librerías para serializar los paquetes de comunicación (ej. JSON)**
    - Si no se hace un correcto manejo de envío y recepción de paquetes (caracteres delimitadores o bloques fijos).
- Implica **observación**:
    - Si el protocolo no es claro y ordenado. Puede ser un protocolo complejo pero tiene que mantener coherencia.
    - Si se opta por otros IPCs aparte de Sockets/FileLocks/BlockingQueues para sincronizar y la implementación queda confusa/poco mantenible.
    - Si no se optimiza el almacenamiento de datos para posteriores consultas con algún particionado

TP Escalabilidad: Middleware y Coordinación de Procesos

- Implica **reentrega**:
    - Todas las condiciones de TP anteriores.
    - Si no se utiliza un DAG para justificar la lógica de separación en procesos y coordinación.
    - Si no se despliega el sistema con múltiples containers usando docker-compose.
    - Si no se utilizó un framework MOM (Rabbit, Kafka, ZMQ) para comunicar mensajes entre containers.
    - Si no se utiliza un middleware para ocultar la complejidad de comunicación y grupos de procesos, quedando expuesta la lógica de conectividad en el código del core de negocios.
    - Si no se sincroniza correctamente la recepción de mensajes en los controllers que tienen dependencias (ej: joiners, groupers, sorters, etc).
    - Si no se implementa un graceful quit correctamente.
    - Si no se realiza streaming de los registros del cliente hacia el source del sistema distribuido o no se realiza el streaming de los registros del sink hacia el cliente.
    - Si se utilizan **sleeps** para sincronizar o **spin locks** para poolear.
- Implica **observación o quita de puntos**:
    - Todas las condiciones de TP anteriores.
    - Si se propagan innecesariamente columnas en etapas del DAG cuando ya no son necesarias.
    - Si el DAG es excesivamente complejo en comparación con el requerimiento.
    - Si no hay un diseño organizado para que la lógica de negocio sea simple (orientada a procesar simples registros de string, tuplas o DTOs) o esté desacoplado de la comunicación.
    - Si hay copy-paste donde claramente se podría haber reutilizado código.
    - Si se abusa de la *affinity* con ciertos procesos para implementar la funcionalidad.