TECNOLÓGICO DE ESTUDIOS SUPERIORES DEL ORIENTE

DEL ESTADO DE MÉXICO

Sistemas Operativos

PROYECTO:

"Proceso Productor-Consumidor

conectado a Base de Datos en MariaDB"

5S12

Integrantes:

- Bustos Vidal Jonathan Daniel
- Quintero Vázquez Valeria

Ingeniera en Sistemas Computacionales

Índice

Introducción

El proceso productor-consumidor es un problema clásico de sincronización en la programación concurrente que se utiliza para ilustrar cómo los procesos (o hilos) pueden coordinarse al compartir un recurso común. Este problema se presenta comúnmente en sistemas donde un productor genera datos y un consumidor los procesa, usando un búfer como intermediario.

Componentes principales:

1. **Productor**: Es el proceso o hilo que genera datos o productos y los coloca en un búfer compartido.
2. **Consumidor**: Es el proceso o hilo que toma los datos o productos del búfer para procesarlos.
3. **Búfer compartido**: Es una estructura de datos (como un arreglo o cola) que actúa como intermediario entre el productor y el consumidor. Puede ser de tamaño limitado (búfer finito) o ilimitado.

Solución:

Para evitar conflictos, se usan técnicas de sincronización como:

- **Semáforos**: Para controlar el acceso al búfer compartido.
- **Mutex (exclusión mutua)**: Para asegurar que solo un proceso acceda al búfer a la vez.
- **Variables de condición**: Para indicar si el búfer está lleno o vacío.

Este proyecto tiene como objetivo crear un sistema que permita enviar y recibir información sobre estudiantes de manera rápida y sencilla. Para ello, utilizamos el protocolo de comunicación MQTT, que es ideal para transmitir datos entre diferentes aplicaciones. En este caso, un programa (el productor) envía información sobre un estudiante, como su nombre, matrícula, materias y calificaciones, a través de un mensaje en formato JSON.

El otro programa (el consumidor) recibe este mensaje, lo procesa y guarda la información en una base de datos. De esta manera, se centraliza la información de los estudiantes en una base de datos, lo que facilita su consulta y almacenamiento.

El uso de MQTT hace que este sistema sea eficiente y fácil de implementar, permitiendo que los datos se transmitan de forma rápida y segura entre las aplicaciones.

Desarrollo

El desarrollo de este proyecto se dividió en tres partes principales: el productor, el consumidor y la librería que facilita la conexión con la base de datos. Cada uno de estos componentes se diseñó con el propósito de transmitir y almacenar información sobre estudiantes a través de mensajes MQTT, utilizando el formato JSON para estructurar los datos.

1. Productor

El productor es el encargado de recopilar la información de un estudiante y enviarla a través de un mensaje MQTT. Para obtener los datos del estudiante, el programa recibe los valores como parámetros desde la línea de comandos. Estos datos incluyen el nombre, apellido, matrícula, carrera, número de materias y las materias junto con sus respectivas calificaciones.

Una vez que se obtienen estos datos, el productor utiliza la biblioteca json-c para estructurar la información en un formato JSON. Luego, este mensaje JSON es enviado al servidor MQTT (concretamente al tema "estudiantes") utilizando la biblioteca mosquitto.

El código del productor hace lo siguiente:

1. Recibe los datos del estudiante.
2. Crea un objeto JSON que contiene toda la información del estudiante.
3. Publica el mensaje JSON en el servidor MQTT

2. Consumidor

El consumidor se encarga de recibir los mensajes enviados por el productor. Para ello, se suscribe al tema "estudiantes" del servidor MQTT. Cuando un mensaje es recibido, el consumidor procesa los datos en formato JSON y los inserta en una base de datos MySQL/MariaDB.

El consumidor realiza lo siguiente:

1. Se conecta al servidor MQTT y se suscribe al tema "estudiantes".
2. Recibe el mensaje que contiene la información del estudiante en formato JSON.
3. Extrae los datos del mensaje (nombre, matrícula, materias, calificaciones, etc.).
4. Llama a las funciones de la librería para guardar estos datos en la base de datos.

3. Librería (enviolib)

La librería enviolib proporciona las funciones necesarias para interactuar con la base de datos. En este proyecto, la base de datos se utiliza para almacenar la información de los estudiantes recibida a través de los mensajes MQTT.

Las funciones de la librería son:

- **Conectar a la base de datos**: Establece una conexión con la base de datos MySQL/MariaDB utilizando las credenciales y la información de conexión proporcionada.
- **Escribir en la base de datos**: Inserta los datos del estudiante en la tabla semestre de la base de datos. La información de materias y calificaciones se guarda en formato JSON, lo que permite almacenar múltiples valores de manera estructurada.

La tabla semestre en la base de datos tiene los siguientes campos:

- id: Identificador único para cada entrada.
- nombre, apellido_paterno, apellido_materno, matricula, carrera: Información personal del estudiante.
- numero_materias: Número de materias en las que el estudiante está inscrito.
- materias: Lista de materias en formato JSON.
- calificaciones: Lista de calificaciones en formato JSON.

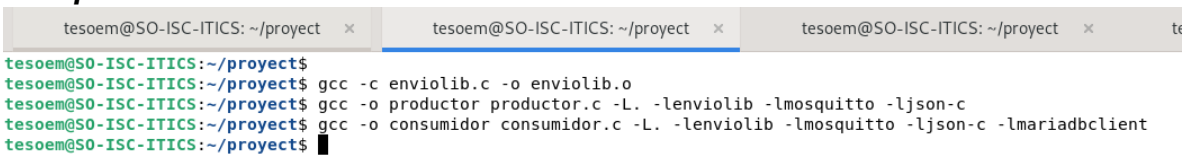
Por lo tanto, todo junto funciona de la siguiente forma:

El productor se ejecuta y, utilizando los parámetros de entrada (como nombre, matrícula, materias y calificaciones), crea un mensaje JSON con los datos del estudiante, después el productor publica este mensaje en el tema estudiantes del servidor MQTT. Luego el consumidor, que está suscrito al tema estudiantes, recibe el mensaje. Por lo tanto, el consumidor procesa los datos, extrae la información del JSON y utiliza la librería enviolib para guardar estos datos en la base de datos.

Y finalmente, la base de datos almacena la información de los estudiantes, incluyendo las materias y calificaciones.

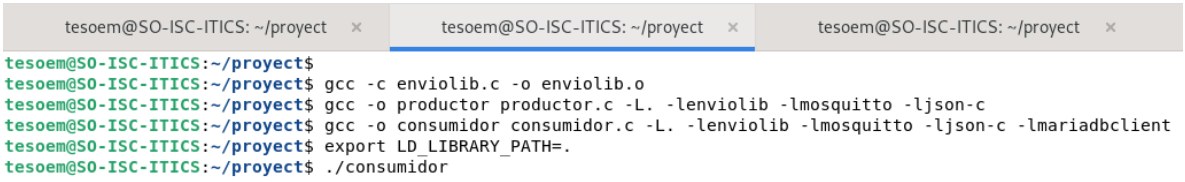
Evidencia del Funcionamiento:

Compilación:

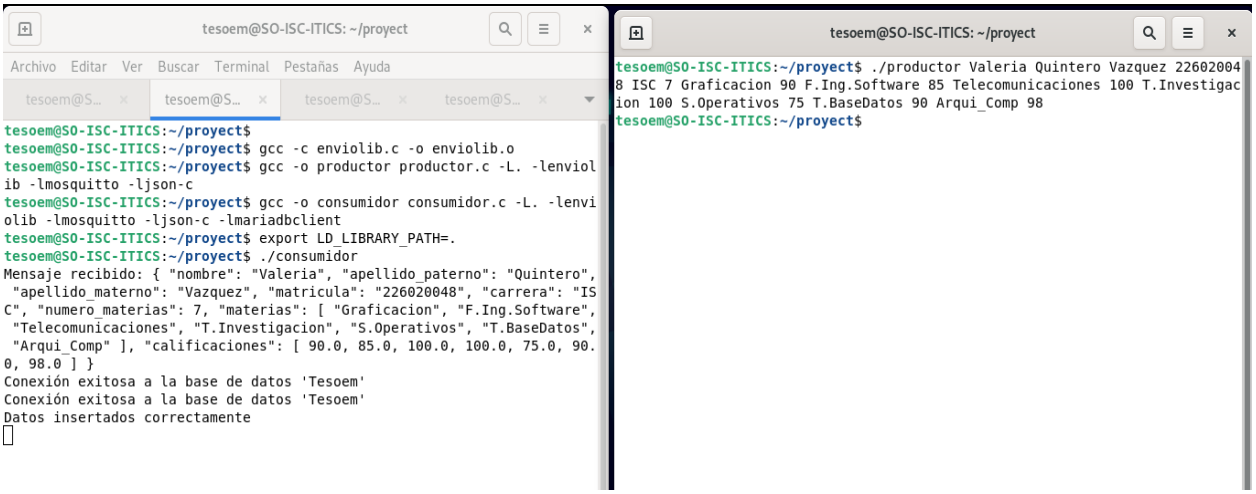


Ejecución:

Iniciar el consumidor para que reciba el mensaje del productor



En otra terminal iniciar el productor y poner los datos a mandar al consumidor para que este los reciba.

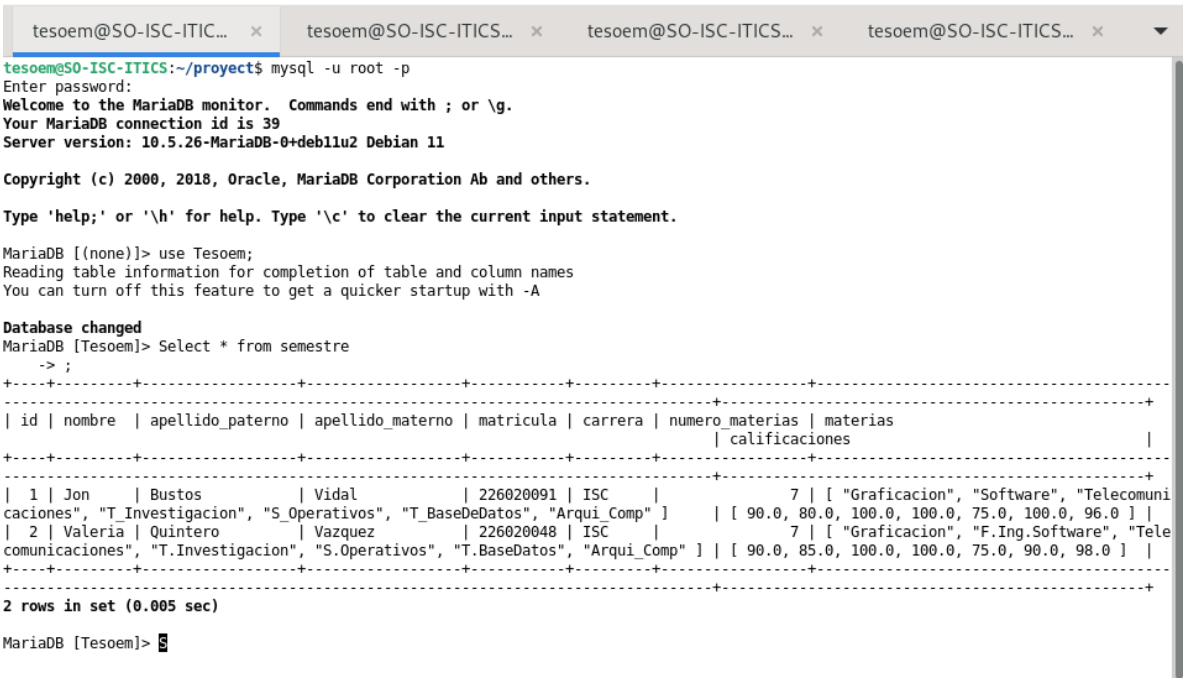


Se puede apreciar que el consumidor recibió el mensaje del productor con éxito, y siendo almacenado en la base de datos de MariaDB

Consulta de la BD

Ingresar a MariaDB con el siguiente comando:

Verificación de la inserción en la Base de Datos Tesoem, tabla semestre.



Se puede ver que el consumidor envió los datos correctamente a la base de datos de MariaDB

Conclusión

Este proyecto demostró cómo integrar un sistema distribuido utilizando el protocolo MQTT para transmitir datos entre aplicaciones. Al emplear un productor que envía información sobre estudiantes en formato JSON y un consumidor que recibe estos mensajes y los almacena en una base de datos MySQL/MariaDB, se logró crear un flujo eficiente de datos entre diferentes partes del sistema.

El sistema funciona correctamente, permitiendo que la información sobre el estudiante, como su nombre, matrícula, materias y calificaciones, se guarde de forma ordenada y pueda ser consultada más tarde. Esto se logra de manera rápida y sin mucha complejidad gracias al uso de MQTT y el formato JSON.

En general, el proyecto cumplió su objetivo de transmitir y almacenar la información de manera efectiva, y tiene la posibilidad de ser mejorado o ampliado en el futuro si es necesario.

Bibliografía

Biblioteca Mosquitto (para MQTT) Mosquitto. (s.f.). *Mosquitto - Eclipse*. Eclipse Foundation. Recuperado de https://mosquitto.org

Biblioteca JSON-C (s.f.). *JSON-C: A JSON library for C*. Recuperado de https://github.com/json-c/json-c

GeeksforGeeks. (2024, 11 octubre). *Producer Consumer Problem in C*. GeeksforGeeks. https://www.geeksforgeeks.org/producer-consumer-problem-in-c/

IBM / AIX 7.3. (s. f.). https://www.ibm.com/docs/es/aix/7.3?topic=models-producerconsumer

IBM Business Automation Workflow 22.x. (2024, enero 25). Ibm.com. https://www.ibm.com/docs/es/baw/22.x?topic=formats-javascript-object-notation-json-format

MQTT broker: How it works, popular options, and quickstart. (s/f). Www.emqx.com. Recuperado el 7 de enero de 2025, de https://www.emqx.com/en/blog/the-ultimate-guide-to-mqtt-broker-comparison

Propuesta de mejora

Aunque el sistema desarrollado funciona correctamente, no se valida la entrada de datos. Sería beneficioso implementar una validación más rigurosa de los datos antes de enviarlos al servidor MQTT, asegurando que no se publiquen datos incompletos o incorrectos. Esto también incluye verificar que el formato del JSON sea correcto antes de su publicación.

Y también otra mejora posible seria optimizar la base de datos, se podría mejorar la estructura de la base de datos para que las materias y calificaciones se guarden de manera más eficiente. Actualmente, se almacenan en formato JSON, pero separar estas columnas en tablas relacionadas podría facilitar consultas más rápidas y análisis más complejos en el futuro.