Contadores Concurrentes

Los contadores concurrentes se utilizan en programación paralela y multihilo para realizar operaciones de conteo en un entorno donde múltiples hilos o procesos acceden al mismo recurso. La principal preocupación aquí es garantizar que las operaciones de lectura y escritura en el contador sean atómicas, es decir, que no haya interferencia entre los hilos al actualizar el contador. Esto se logra mediante técnicas de sincronización, como el uso de mutexes o semáforos para evitar condiciones de carrera. Un contador sin protección puede dar lugar a resultados erróneos debido a que varios hilos pueden intentar modificar su valor simultáneamente.

Listas Enlazadas Concurrentes

Las listas enlazadas concurrentes son estructuras de datos utilizadas en programación multihilo donde los elementos de la lista están interconectados mediante punteros. Para asegurarse de que varios hilos puedan acceder y modificar la lista sin causar inconsistencias, se debe emplear sincronización adecuada. Generalmente, esto implica el uso de bloqueos (locks) o atomics para garantizar que solo un hilo pueda modificar la lista a la vez, evitando condiciones de carrera y garantizando la integridad de la estructura de datos. En entornos multihilo, también se pueden utilizar técnicas como el bloqueo optimista o el bloqueo de lectura-escritura (para permitir que múltiples hilos lean la lista de manera concurrente mientras que solo uno puede modificarla).

Colas Concurrentes

Las colas concurrentes permiten que múltiples hilos agreguen y eliminen elementos de la cola de manera segura. Al igual que en las listas enlazadas concurrentes, es necesario proteger la estructura de la cola con técnicas de sincronización para evitar condiciones de carrera. Las colas pueden ser implementadas utilizando semáforos, mutexes, o técnicas avanzadas como locks de múltiples lectores o colas de mensajes, para asegurarse de que un solo hilo pueda modificar la cola a la vez, mientras otros hilos puedan consumir o agregar elementos sin causar inconsistencias.

Tablas Hash Concurrentes

Las tablas hash concurrentes son estructuras de datos que permiten almacenar pares de claves y valores y acceder a ellos rápidamente mediante una clave. En un entorno multihilo, el acceso concurrente a una tabla hash puede ser problemático debido a las colisiones en las claves o el acceso simultáneo. Para manejar estos casos, se utilizan bloqueos (locks) o técnicas de atomización para asegurar que las operaciones de lectura y escritura sean seguras. También pueden implementarse hashes con múltiples particiones para permitir un acceso concurrente más eficiente a diferentes partes de la tabla.

Variables de Condición

Las variables de condición son primitivas de sincronización que permiten que los hilos se bloqueen hasta que se cumpla una condición determinada. Son útiles cuando un hilo necesita esperar a que otro hilo realice una acción antes de continuar. Por ejemplo, un hilo productor puede esperar hasta que haya espacio en un buffer, mientras que un hilo consumidor puede esperar hasta que haya datos disponibles. Las variables de condición generalmente se usan junto con un mutex para garantizar que las condiciones se evalúan y modifican de manera atómica.

Implementación del Join

La implementación de join se refiere a cómo un hilo puede esperar a que otro hilo termine su ejecución. Esto es útil cuando un hilo depende de los resultados de otro hilo antes de continuar. En muchas bibliotecas de hilos, como pthread, se ofrece una función pthread_join() que permite que un hilo espere a que otro termine. Esto se logra bloqueando al hilo que invoca join hasta que el hilo al que se espera termine su ejecución.

Reimplementación del Join
Una reimplementación del join consiste en crear una función que simule el comportamiento de join de una manera personalizada, generalmente utilizando variables de condición y semáforos. Al reimplementar join, se debe crear un mecanismo para que un hilo se ponga en espera hasta que otro hilo termine, lo cual puede implicar la implementación de un sistema de sincronización propio.

Problema del Productor-Consumidor
El problema del productor-consumidor es un clásico de la programación concurrente que implica dos tipos de hilos: un productor, que produce datos, y un consumidor, que consume esos datos. El desafío es coordinar la sincronización de los hilos para evitar que los hilos intenten acceder a la misma área de memoria de manera concurrente, lo que podría causar inconsistencias o corrupción de datos.

Productor/Consumidor: Única CV y Sentencia if
Cuando se utiliza una sola variable de condición (CV) y una sentencia if, el productor y el consumidor tienen que verificar una condición antes de actuar. Sin embargo, este enfoque puede ser menos eficiente si no se controla correctamente la condición en que el hilo productor o consumidor debe esperar, lo que podría provocar bloqueos innecesarios.

Productor/Consumidor: Única CV y Sentencia while
Este enfoque también utiliza una sola variable de condición (CV), pero en este caso se emplea un ciclo while para verificar repetidamente la condición antes de que el hilo productor o consumidor continúe. El uso de while es importante porque permite volver a comprobar la condición después de que el hilo haya sido despertado, lo que ayuda a prevenir condiciones de carrera.

Productor/Consumidor - Single Buffer: Usar 2 Variables de Condición y un While
En la versión de buffer único, se utilizan dos variables de condición: una para el productor (cuando hay espacio en el buffer) y otra para el consumidor (cuando hay datos en el buffer). Ambas variables de condición están sincronizadas con un ciclo while para asegurar que los hilos esperen hasta que sea seguro actuar, evitando así conflictos y condiciones de carrera.

Semáforos
Los semáforos son una de las primitivas más utilizadas para la sincronización en programación concurrente. Son variables de control que permiten que los hilos adquieran y liberen recursos de manera controlada.

Semáforos Binarios (Locks)
Los semáforos binarios, también conocidos como locks, son semáforos que pueden tener solo dos valores posibles: 0 o 1. Un valor de 1 indica que el recurso está disponible, y un valor de 0 indica que el recurso

está bloqueado. Se utilizan para implementar mecanismos de exclusión mutua (mutexes) en la programación multihilo.

Semáforos como Variables de Condición
Un semáforo también puede funcionar como una variable de condición en ciertos casos, permitiendo que un hilo se bloquee hasta que un semáforo llegue a un valor específico. Se utiliza para coordinar la ejecución entre hilos en problemas como el productor-consumidor o en la implementación de la sincronización de eventos.

Problemas Clásicos
Problema del Productor-Consumidor
Como se mencionó, este problema trata sobre cómo gestionar la producción y el consumo de datos entre dos hilos, sincronizando sus operaciones para evitar condiciones de carrera y asegurar la integridad de los datos.

Problema de los Lectores y Escritores
El problema de los lectores y escritores se refiere a la situación en la que múltiples hilos intentan leer y escribir sobre un recurso compartido. Los lectores pueden acceder al recurso de manera concurrente, pero los escritores requieren acceso exclusivo. Las técnicas de sincronización, como los locks de lectura-escritura, se utilizan para garantizar que los escritores no interfieran con los lectores y viceversa.

Problema de los Filósofos
El problema de los filósofos comensales es un clásico de la programación concurrente que ilustra cómo los hilos (filósofos) pueden quedar bloqueados esperando recursos (tenedores) de manera que crean un ciclo de dependencia. La solución a este problema implica el uso de técnicas de sincronización para evitar el interbloqueo y la inanición de los filósofos.

Implementación de Semáforos
La implementación de semáforos involucra el uso de primitivas de bajo nivel para garantizar que los semáforos se comporten correctamente, es decir, que puedan bloquear y desbloquear hilos de manera eficiente.

Conclusiones
Los temas cubiertos en este examen son esenciales para entender la programación concurrente y cómo manejar la sincronización entre hilos en entornos multihilo. El uso adecuado de semáforos, variables de condición, y otros mecanismos de sincronización es fundamental para evitar condiciones de carrera, interbloqueos y problemas de inconsistencia de datos. Tener una comprensión sólida de estos conceptos permitirá diseñar sistemas eficientes y seguros en aplicaciones multihilo.