LECTOR-ESCRITOR

By: Jared Isaías Monje Flores

24 DE OCTUBRE DE 2023

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

UdeG

El "problema de los lectores-escritores" es un dilema de programación creado cuando varios lectores y escritores necesitan tener acceso al mismo recurso. Si todos fueran permitido el acceso a la vez, podrían surgir problemas como sobrescrituras, información incompleta, y otros temas. Por lo tanto, los programadores pueden restringir el acceso al control que los hilos de procesamiento de ver el recurso y que, teniendo en cuenta las necesidades del sistema y los usuarios. Hay varias maneras de abordar el problema de los lectores-escritores. Una de las soluciones más comunes implica el uso de semáforos para la bandera del estado de acceso y control.

Desde una perspectiva, cualquier número de lectores podría acceder con seguridad a un recurso porque no están haciendo cambios en el contenido. Una vez que un escritor entra en la ecuación, la situación se vuelve más complicada. Si un hilo está escribiendo mientras que otros flujos leen, los lectores no pueden obtener la información correcta. Podrían recibir sólo una parte del cambio, o pueden ver la información obsoleta y pensar ita € s precisa.

Más de un escritor también podría crear un problema. Los cambios simultáneos a un mismo contenido pueden sobrescribir y crear otros errores. Bajo el problema de los lectores-escritores, programadores deben decidir si los lectores o escritores tienen prioridad, y cómo manejar el acceso. Los lectores o escritores podrían asignar prioridad, o el sistema podrían asignar acceso en un primer llegado, primer servido base. Esta tercera solución puede evitar largas esperas, pero podría venir con sus propios problemas.

En una solución donde los lectores tienen prioridad, el sistema asume que cualquier lector pidiendo acceso se debe permitir en primer lugar, cuando el acceso esté disponible. Esto significa que los escritores que quieren acceder al recurso podrían tener que esperar. Por el contrario, el sistema podría asumir que porque los escritores necesitan hacer cambios que puedan afectar a los lectores, se les debe dar prioridad en el problema de los lectores-escritores. Cuando un lector se hace con un recurso, un escritor podría saltar en hacer un cambio. Esto se aplica no sólo a las acciones del usuario como tratar de guardar un documento, sino a los procesos internos dentro de la computadora que mantener el sistema en funcionamiento.

Otra opción permite que el problema de los lectores-escritores de equilibrar las necesidades de ambas partes, lo que permite el acceso cada subproceso de procesamiento a medida que llega. Esto evita que los escritores de hacer cambios que anulan entre sí o confunden a los lectores, sin salir de los lectores de espera, o forzar a los escritores para sostener mientras que los lectores terminen. Tales prioridades pueden ser construidos en un control de acceso programa de software o memoria en una computadora. Los usuarios pueden ser capaces de hacer cambios si se sienten cómodos con la programación y el sistema lo permite.

Aplicaciones comunes del lector-escritor

Base de datos: En sistemas de gestión de bases de datos, varios usuarios o procesos pueden necesitar leer datos simultáneamente, mientras que solo un proceso debe poder escribir o actualizar los datos. El lector-escritor se utiliza para coordinar el acceso a la base de datos de manera eficiente y segura.

Acceso a archivos compartidos: En sistemas operativos multiusuario, varios usuarios pueden necesitar acceder a un archivo compartido. El lector-escritor se utiliza para permitir múltiples lecturas concurrentes, pero solo una escritura a la vez.

Caché de datos: En aplicaciones que utilizan almacenamiento en caché, los lectores pueden acceder a datos en caché simultáneamente, mientras que un escritor actualiza los datos en caché. Esto mejora el rendimiento de la aplicación.

Problemas comunes del lector-escritor

Condición de carrera: Uno de los problemas más comunes es la condición de carrera, donde múltiples hilos o procesos intentan acceder al recurso compartido al mismo tiempo, lo que puede llevar a resultados inesperados o incorrectos. Para evitar esto, se deben implementar mecanismos de sincronización adecuados.

Inanición: La inanición (starvation) es un problema en el que un lector o escritor puede quedar bloqueado indefinidamente debido a la prioridad incorrecta en el acceso al recurso compartido. Es esencial garantizar que todos los hilos tengan la oportunidad de acceder al recurso.

Prioridades incorrectas: Si no se administran adecuadamente las prioridades entre lectores y escritores, es posible que un grupo de hilos tenga acceso preferencial al recurso compartido, lo que podría ser injusto o ineficiente.

Bloqueo mutuo: El bloqueo mutuo (deadlock) puede ocurrir cuando varios hilos o procesos se bloquean entre sí, lo que impide que el programa avance. Es importante implementar una lógica que evite que esto suceda.

Supongamos que tienes un recurso compartido (por ejemplo, una base de datos o una estructura de datos) que múltiples hilos o procesos pueden querer leer o escribir de manera concurrente.

1. Inicialización de semáforos y variables:

- Declarar dos semáforos: uno para el acceso de escritura y otro para el acceso de lectura.

- Declarar un entero que llevará el registro del número de lectores activos.

```
semaphore mutex = 1;       // Controla el acceso a las variables compartidas.
semaphore wrt = 1;         // Controla el acceso para escritura.
int readCount = 0;         // Contador de lectores activos.
```

2. Acceso de lectura (lector):

- Un lector que desea acceder al recurso debe adquirir el semáforo `mutex` para evitar la condición de carrera en el contador de lectores.

```
P(mutex);              // Bloquea otros lectores.
readCount++;           // Incrementa el contador de lectores activos.
if (readCount == 1)    // Si el primer lector, bloquea el acceso a escritura.
    P(wrt);
V(mutex);              // Libera mutex.
```

- El lector realiza sus operaciones de lectura en el recurso compartido.

- Después de leer, el lector debe liberar el recurso y actualizar el contador.

```
P(mutex);              // Bloquea otros lectores.
readCount--;           // Decrementa el contador de lectores activos.
if (readCount == 0)    // Si es el último lector, permite escritura.
    V(wrt);
V(mutex);              // Libera mutex.
```

3. Acceso de escritura (escritor):

- Un escritor que desea acceder al recurso debe adquirir el semáforo `wrt`, lo que evita que otros escritores y lectores accedan al recurso.

```
17 P(wrt);                // Bloquea otros escritores y lectores.
18 // Realiza operaciones de escritura en el recurso compartido.
19 V(wrt);                // Libera el recurso para otros escritores o lectores.
```

Este algoritmo garantiza que:

- Los lectores pueden leer simultáneamente a menos que haya un escritor activo.

- Solo un escritor puede acceder al recurso a la vez.

- Se evita la inanición y el bloqueo mutuo al garantizar que los lectores no bloqueen a los escritores y viceversa.

La implementación real puede variar según el lenguaje de programación y las API de sincronización utilizadas.

Referencias:

EL PROBLEMA DE LOS LECTORES Y ESCRITORES. (2022). Prezi.com. https://prezi.com/llus9gxf4rsx/el-problema-de-los-lectores-y-escritores/

PROBLEMA DE LOS LECTORES-ESCRITORES. (2023). Infor.uva.es. https://www2.infor.uva.es/~cllamas/concurr/pract98/sisos30/index.html

¿Cuál es el Problema de lectores y escritores? / Prucommercialre.com. (2023). Prucommercialre.com. https://www.prucommercialre.com/cual-es-el-problema-de-lectores-y-escritores/