MicroLab6

Hector Sandoval

Septiembre 2025

Práctica 1: Ciclo de vida de hilos y condiciones de carrera

1. Interleaving mínimo que explica la pérdida

Un interleaving mínimo que puede causar una condición de carrera es el siguiente: el Hilo 1 lee el valor de `global`, luego el Hilo 2 también lee el mismo valor de `global`. Posteriormente, el Hilo 1 incrementa el valor y lo escribe de nuevo, seguido por el Hilo 2 que también incrementa el valor (basado en el valor que leyó antes) y lo escribe de nuevo. Esto puede resultar en que ambos hilos incrementen el contador, pero solo uno de los incrementos se guarda, causando una pérdida.

2. Cambio en el throughput al pasar de mutex a sharded

Al usar mutex, el throughput puede ser limitado debido a la contención, ya que solo un hilo puede acceder al recurso compartido a la vez. En cambio, al usar contadores particionados (sharded), cada hilo puede trabajar en su propia sección del contador, lo que reduce la contención y mejora el throughput, especialmente en sistemas con múltiples núcleos.

3. Coste del reduce que anula la ganancia

El coste del reduce puede anular la ganancia cuando el número de hilos es bajo o cuando el trabajo por hilo es pequeño. Si el tiempo que se tarda en combinar los resultados de los hilos es mayor que el tiempo ahorrado al ejecutar los hilos en paralelo, entonces la ganancia se anula.

Práctica 2: Búfer circular productor/consumidor

1. Uso de `while` y no `if` al esperar

Se usa `while` en lugar de `if` porque la condición de espera puede cambiar mientras el hilo está bloqueado. Usar `while` asegura que el hilo vuelva a verificar

la condición después de ser despertado, evitando que se produzcan errores si la condición no se cumple.

2. Diseño de un shutdown limpio sin pérdidas

Un shutdown limpio se puede lograr estableciendo una bandera de parada (como `stop`) que indique a los productores y consumidores que deben dejar de operar. Al finalizar, el productor debe notificar a los consumidores que no habrá más datos, utilizando `pthread_cond_broadcast` para despertar a todos los consumidores.

3. Política de signaling que reduce la latencia

Usar `pthread_cond_signal` es más eficiente cuando solo un hilo necesita ser despertado, ya que reduce la sobrecarga de despertar a todos los hilos. Sin embargo, si hay múltiples consumidores esperando, `pthread_cond_broadcast` puede ser necesario para asegurarse de que al menos uno de ellos pueda continuar.

Práctica 3: Lectores/Escritores y equidad

1. Cuándo conviene `rwlock` frente a `mutex`

`rwlock` es más conveniente cuando hay una mayoría de operaciones de lectura en comparación con las operaciones de escritura. Permite que múltiples lectores accedan simultáneamente a los datos, mientras que un `mutex` solo permite un acceso exclusivo, lo que puede ser ineficiente en escenarios con muchas lecturas.

2. Cómo evitar starvation del escritor

Para evitar la inanición del escritor, se pueden implementar políticas de prioridad que aseguren que los escritores tengan acceso eventual a los recursos. Por ejemplo, se puede limitar el número de lectores que pueden acceder al recurso al mismo tiempo o implementar un sistema de turnos.

3. Impacto del tamaño de bucket en la contención

Un tamaño de bucket más pequeño puede aumentar la contención, ya que más hilos pueden intentar acceder al mismo bucket simultáneamente. Un tamaño de bucket más grande puede reducir la contención, pero también puede aumentar el uso de memoria. Es importante encontrar un equilibrio adecuado.

Práctica 4: Deadlock intencional, diagnóstico y corrección

1. Condiciones de Coffman que se cumplen

Las condiciones de Coffman que se cumplen en este deadlock son:

- **Exclusión mutua**: Ambos hilos necesitan acceso exclusivo a los mutex.
- **Retención y espera**: Cada hilo mantiene un mutex mientras espera por otro.
- **No preempción**: Los mutex no pueden ser forzados a liberarse.
- **Espera circular**: Hilo 1 espera por el mutex B que está siendo sostenido por Hilo 2, y Hilo 2 espera por el mutex A que está siendo sostenido por Hilo 1.

2. Cómo probar el deadlock con GDB/Helgrind

Para probar el deadlock con GDB, puedes ejecutar el programa y usar el comando `thread apply all bt` para obtener un backtrace de todos los hilos. Esto te mostrará en qué parte del código están bloqueados los hilos. Con Helgrind, puedes ejecutar el programa con `valgrind --tool=helgrind ./tu_programa` para detectar condiciones de carrera y deadlocks. Helgrind analizará el acceso a los mutex y te informará si hay interbloqueos.

3. Estrategia de ordenación global adoptada en un sistema real

Una estrategia efectiva para evitar deadlocks es establecer un orden global para adquirir los mutex. Por ejemplo, siempre adquirir el mutex A antes del mutex B en todos los hilos. Esto elimina la posibilidad de que se forme un ciclo de espera, ya que todos los hilos seguirán el mismo orden al adquirir los recursos.

Práctica 5: Pipeline por etapas con `pthread_barrier_t` y `pthread_once`

1. Dónde conviene barrera frente a colas

Las barreras son convenientes cuando se necesita sincronizar múltiples hilos en un punto específico de la ejecución, asegurando que todos los hilos lleguen a ese punto antes de continuar. En contraste, las colas son más adecuadas para la comunicación entre hilos, donde un hilo produce datos y otro los consume. Las barreras son útiles en pipelines donde las etapas deben completarse antes de pasar a la siguiente.

2. Cómo medir el throughput por etapa

Para medir el throughput por etapa, puedes contar el número de operaciones completadas en cada etapa durante un período de tiempo determinado. Registra el tiempo de inicio y fin de cada etapa y calcula el número de operaciones por segundo. Esto te permitirá evaluar el rendimiento de cada etapa del pipeline.

3. Cómo diseñar un graceful shutdown sin deadlocks

Un shutdown limpio se puede lograr implementando una señal de parada que indique a los hilos que deben finalizar su trabajo. Puedes usar una variable compartida que los hilos verifiquen periódicamente. Al recibir la señal de parada, los hilos deben completar su trabajo actual y salir de manera ordenada. Además, asegúrate de que todos los hilos estén sincronizados antes de finalizar el programa, utilizando barreras o mutex para evitar condiciones de carrera.