Primer tarea de Promoción: Deadlocks

Alumno

- Amoroso Lihuel Pablo - 13497/2

Consigna

Investigar cuál de los 3 enfoques para el manejo de Deadlocks utilizan Windows y GNU/Linux:

- Se deberá presentar un informe que explique brevemente los 3 enfoques (asegurar que no se entrara en estado de deadlock -prevenir o evitar-, permitir y recuperar, ignorar).
- Deberá explicar el/los adoptados por cada SO, si son o no configurables e indicar las fuentes de donde obtuvo la información.
- El informe no podrá superar las 5 carillas y deberá subirse a la plataforma antes del 3/5.

Enfoques para afrontar el deadlock

Existen distintos enfoques con el fin de llevar adelante el manejo de Deadlocks:

1. Usar un protocolo que asegure que NUNCA se entrará en estado de deadlock

    1. Prevenir: es decir, que no se cumplan alguna de las 4 condiciones. Para lograr esto es necesario que al menos una de las condiciones no pueda mantenerse, esto es, imponer algún tipo de restricción en la forma en que los procesos requieren los recursos, por lo que no habrá forma de que los pedidos que realicen estos provoquen deadlock.
    2. Evitar: en otras palabras, tomar decisiones de asignación en base al estado del sistema asignando cuidadosamente los recursos, manteniendo información actualizada sobre requerimiento y uso de recursos. Es preciso tener en cuenta que esta solución solo puede funcionar si el sistema sabe que recursos pedirá un proceso en el futuro y ello es una suposición poco realista. Si bien es posible utilizar el algoritmo del banquero, lo cierto es que es casi imposible de implementar a causa de la suposición anterior.

    Un detalle no tan menor es que la diferencia entre **prevenir** y **evitar** es muy sutil. Yo uso la siguiente estrategia:

    - Prevenir: cambiar las reglas del juego.
    - Evitar: chequear si un recurso puede ser permitido y si, de permitirlo, este podría causar deadlock, entonces no permitirlo.

2. Detección y Recuperación: esto es, utilizar diferentes técnicas que permitan determinar si ocurrió un deadlock, y en tal caso, poder recuperar el sistema del mismo. Es posible construir un grafo de pedidos de recurso y chequear por ciclos. Con recursos de una sola instancia se podrá usar un grafo de asignación; en cambio, si los recursos tuvieran varias instancias, será conveniente el algoritmo del banquero.

3. Ignorar el problema y esperar que nunca ocurra un deadlock: es lo que comunmente se conoce como algoritmo del avestruz(meter la cabeza en la arena y pretender que no existe problema alguno). Simplemente esperar que el deadlock no ocurra. En general, es una estrategia razonable. El deadlock es algo que ocurre rara vez; de hecho, un sistema puede estar años sin que ocurra un deadlock en él. Si el sistema operativo tiene un sistema de prevención o detección de deadlock ejecutándose, esto tendrá un impacto negativo en la performance de éste(lo va a ralentizar) porque siempre que un proceso o hilo pida un recurso, el sistema tendrá que chequear si, permitiendo el acceso al recurso, tal petición podría causar un deadlock. Lógicamente, si ocurriera un deadlock sería necesario reiniciar el sistema o, al menos, matar manualmente un par de procesos, pero incluso eso no se considera una situación extrema en muchas casos.

Windows

En la documentación de drivers de Microsoft para Windows se encuentran distintos mecanismos, alternativas y herramientas para afrontar deadlocks en Windows.

- En el artículo Debugging a Deadlock se explica cómo debuggear deadlocks a nivel de User-Mode y Kernel-Mode. El mismo detalla que Windows dispone de una herramienta de *debugging* nativa, la cual, entre muchas opciones, permite activar la detección de deadlock sobre uno o varios drivers.

- En Deadlock Detection se indica el funcionamiento de la herramienta, el cual consiste en un grafo de asignación de recursos y busca la existencia de ciclos para identificar posibles deadlocks.

- En !deadlock se muestra cómo se monitorean los deadlocks detectados por la herramienta anteriormente mencionada. La información acerca de los deadlocks es mostrada cada vez que un proceso adquiere un lock y es ahí en donde se verifica la existencia de deadlocks.

Resumiendo, existe una herramienta de *debugging*(el driver verifier) que, entre muchas opciones, permite monitorear el deadlock de uno o varios drivers. Por defecto, esta herramienta viene desactivada.

GNU/Linux

En el perfil de github de Linus Torvalds se puede encontrar el repositorio correspondiente al código del kernel de GNU/Linux. Buscando 'deadlock' en varios archivos dentro de ese repositorio, di con un comentario en linux/fs/lock.c que dice lo siguiente(el comentario estaba en inglés, pero por las dudas y porque es más cómodo de analizar, lo traduje):

```
/*
/*
 * Detección de deadlock:
 *
 * Intentamos detectar deadlocks que sean puramente por bloqueos relacionados a
 * archivos posix.
 *
 * Asumimos que una tarea podría estar esperando por, a lo sumo, un bloqueo a la
 * vez. Entonces para cada bloqueo adquirido, el proceso manteniendo el bloqueo
 * podría estar esperando, como máximo, otro bloqueo. Tal bloqueo, en turnos,
 * podría ser mantenido por alguien esperando, como mucho, otro bloqueo.
 * Dada una petición de bloqueo, caller_fl, el cual está a punto de esperar un
 * bloqueo conflictivo como podría ser block_fl, seguimos esta cadena de espera
```

```
 * para asegurar que no estemos a punto de crear un ciclo.
 *
 * Dado que hacemos esto antes de poner a dormir un proceso con un bloqueo, nos
 * aseguramos que nunca hay un ciclo; eso es lo que garantiza que el bucle while()
 * en posix_locks_deadlock() eventualmente se completa.
 *
 * Nota: la suposición de arriba podría no ser verdad cuando se manejasen pedidos
 * de bloqueo desde un cliente NFS roto. Podría también fallar en presencia de
 * tareas(como hilos posix) que compartiesen la misma tabla de archivos abierta.
 * Para manejar tales casos, simplemente abortamos la operación luego de algunas *
iteraciones.
 *
 * Para los bloqueos FL_OFDLCK, el dueño es el archivo, no la struct de archivos.
 * Porque el dueño no está ni nominalmente atado a un hilo de ejecución, es que la
 * detección de deadlock más abajo no podría funcionar razonablemente bien.
 * Simplemente conviene omitir tal detección para esos casos.
 *
 * En principio, podríamos hacer una detección de deadlock más limitada sobre
 * los bloqueos de FL_OFDLCK que simplemente chequee por el caso en el que dos
 * tareas estén intentando alternar bloqueos lectura a escritura sobre el mismo
 * inodo.
 */
```

En resumen: la idea es detectar los posibles deadlocks que se den en archivos posix. Para la mayoría de esos casos, no hay forma de que surja un deadlock. Para el resto, simplemente se aborta la operación después de algunas iteraciones.

Conclusión de lo que entendí(aunque no se pida)

A partir de las posiciones que toman los sistemas operativos actuales(Windows y GNU/Linux) y de lo que estuve leyendo, me parece una buena idea ignorar el problema del deadlock dado que este parece ser un problema poco frecuente(al menos en los sistemas operativos domésticos). En Windows, la herramienta que se encarga de monitorearlo por defecto viene desactivada; y en GNU/Linux, la resolución de deadlocks(que a mi parecer es **preventiva**, dado que lo que se utiliza es una opción en la que no hay forma de que ocurra deadlock) parece estar más enfocada a los bloqueos de archivos posix. Intenté buscar documentación acerca del manejo de deadlock en otros sistemas operativos(SunOS y Solaris) pero no encontré nada, así es que tampoco puedo concluir si en otros sistemas operativos el problema es más frecuente o bien de consecuencias más graves.

Referencias

- *Diapositivas de deadlock de la cátedra(Tema 3, transparencias 1 y 2)*.
- *CSCI.4210 Operating Systems Deadlock* - http://www.cs.rpi.edu/academics/courses/fall04/os/c10/
- *Deadlock Detection - Windows drivers | Microsoft Docs* - https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/deadlock-detection
- *Debugging a Deadlock - Windows drivers | Microsoft Docs* - https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugging-a-deadlock
- *deadlock - Windows drivers | Microsoft Docs* - https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/-deadlock

- *El repositorio del kernel de GNU/Linux(y más específicamente el archivo locks.c) - https://github.com/torvalds/linux/blob/master/fs/locks.c*