Tarea de Promoción de Deadlocks

Alumno

- Ulises Jeremias Cornejo Fandos - 13566/7

Consigna

Deberá presentar un informe (extensión no superior a las 2 hojas) que indique cual la frecuencia con la cual se invoca a los algoritmos de detección de Deadlock en Windows y GNU/Linux. Deberá indicar además si este parámetro puede ser modificado.

El informe deberá incluir la fuente de donde extrajo la información

Deseable: *Si lo desea puede agregar información de otro Sistema Operativo.*

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 muestra como debuggear deadlocks a nivel de User-Mode y Kernel-Mode Deadlock. El mismo consiste en una herramienta de debugger que se puede encontrar y activar en el sistema con fines de desarrollo.
- En Deadlock Detection se detalla el funcionamiento de la herramienta que se encarga de hacer la detección de Deadlock en drivers la cual utiliza un grafo de asignación de recursos y busca la existencia de ciclos para identificar posibles deadlocks.
- En !deadlock se detalla como se monitorean los deadlocks detectadas por la herramienta anteriormente mencionada. Muestra información acerca de los deadlocks cada vez que un proceso adquiere un lock y es ahí donde se verifica la existencia de deadlocks.

Cabe destacar que todas las fuentes mencionadas se corresponden con herramientas de desarrollo que no corren durante la ejecución del Sistema Operativo. Esto, sin tener fundamentos solidos al respecto, me permite pensar que Windows no activa el algoritmo de detección de deadlocks cada un cierto tiempo, sino que permite distintas herramientas para hacerlo en caso de ser necesario.

GNU/Linux

En el perfil de github de se puede encontrar el repositorio correspondiente al código del kernel del mismo. En el mismo podemos seleccionar la opción `Find file` buscando el nombre `lock`. La primera opción que aparece es el archivo linux/fs/lock.c

En la linea 960 se encuentra el siguiente comentario:

```
[. . .]

/*
 * Deadlock detection:
 *
 * We attempt to detect deadlocks that are due purely to posix file
 * locks.
 *
 * We assume that a task can be waiting for at most one lock at a time.
 * So for any acquired lock, the process holding that lock may be
 * waiting on at most one other lock.  That lock in turns may be held by
 * someone waiting for at most one other lock.  Given a requested lock
 * caller_fl which is about to wait for a conflicting lock block_fl, we
 * follow this chain of waiters to ensure we are not about to create a
 * cycle.
 *
 * Since we do this before we ever put a process to sleep on a lock, we
 * are ensured that there is never a cycle; that is what guarantees that
 * the while() loop in posix_locks_deadlock() eventually completes.
 *
 * Note: the above assumption may not be true when handling lock
 * requests from a broken NFS client. It may also fail in the presence
 * of tasks (such as posix threads) sharing the same open file table.
 * To handle those cases, we just bail out after a few iterations.
 *
 * For FL_OFDLCK locks, the owner is the filp, not the files_struct.
 * Because the owner is not even nominally tied to a thread of
 * execution, the deadlock detection below can't reasonably work well.
Just
 * skip it for those.
 *
 * In principle, we could do a more limited deadlock detection on
FL_OFDLCK
 * locks that just checks for the case where two tasks are attempting to
 * upgrade from read to write locks on the same inode.
 */

[. . .]
```

En el mismo se indica que el algoritmo de detección de deadlocks se ejecuta cada vez que un proceso intenta bloquear un recurso compartido.