Deadlocks

Hay 3 enfoques para poder llevar adelante el manejo de Deadlocks:

1. **Prevención o evasión**: la idea es no dejar que el sistema entre en un estado de Deadlock.

La estrategia de "**Prevention**" se realiza negando una de las condiciones necesarias para que se de el Deadlock (exclusión mutua, retención y espera, no apropiación y espera circular).

Exclusión mutua:
La exclusión mutua dice que un recurso solo puede ser retenido por un proceso a la vez. Si otro proceso también está demandando el mismo recurso, entonces debe esperar la asignación de ese recurso. Entonces, prácticamente, no podemos violar la exclusión mutua de un proceso porque, en general, un recurso puede realizar el trabajo de un proceso a la vez.

Retención y espera (Hold and Wait):
Retención y espera surge cuando un proceso contiene algunos recursos y está esperando otros recursos que están retenidos por algún otro proceso en espera. Para evitar esto, el proceso puede adquirir todos los recursos que necesite, antes de iniciar su ejecución y posteriormente, inicia su ejecución. De esta forma, el proceso no necesita esperar algunos recursos durante su ejecución. Pero este método no es práctico porque no podemos conocer los recursos que requiere un proceso de antemano, antes de su ejecución. Por lo tanto, otra forma de evitar sujetar y esperar puede ser la técnica de "no sujetar". Por ejemplo, si el proceso necesita 10 recursos R1, R2, R3, ..., R10. En un momento determinado, podemos proporcionar R1, R2, R3 y R4. Después de realizar los trabajos en estos recursos, el proceso necesita liberar estos recursos y luego los otros recursos se proporcionarán al proceso. De esta manera, podemos evitar la condición de retención y espera.

Sin preferencia:
Esta es una técnica en la que un proceso no puede tomar por la fuerza el recurso de otros procesos. Pero si encontramos algún recurso debido al cual se está produciendo un punto muerto en el sistema, entonces podemos apropiarnos por la fuerza de ese recurso del proceso que lo contiene. Al hacerlo, podemos eliminar el punto muerto, pero hay ciertas cosas que deben tenerse en cuenta antes de usar este enfoque contundente. Si el proceso tiene una prioridad muy alta o el proceso es un proceso del sistema, entonces solo el proceso puede apropiarse por la fuerza de los recursos de otros procesos. Además, debe intentar apropiarse de los recursos de aquellos procesos que están en estado de espera.

Espera circular:
La espera circular es una condición en la que el primer proceso está esperando el recurso retenido por el segundo proceso, el segundo proceso está esperando el recurso retenido por el tercer proceso y así sucesivamente. Por fin, el último proceso está esperando el recurso retenido por el primer proceso. Por lo tanto, cada proceso está esperando que el otro libere el recurso. A esto se le llama espera circular. Para evitar esto, lo que podemos

hacer es listar el número de recursos que requiere un proceso y asignar algún número o prioridad a cada recurso (en nuestro caso, estamos usando R1, R2, R3, etc.). Ahora, el proceso tomará los recursos en orden ascendente. Por ejemplo, si el proceso P1 y P2, requiere los recursos R1 y R2, entonces inicialmente, ambos procesos demandarán el recurso R1 y solo uno de ellos obtendrá el recurso R1 en ese momento y el otro proceso tendrá que esperar su turno. Entonces, de esta manera, ambos procesos no se estarán esperando el uno al otro. Uno de ellos estará ejecutando y el otro esperará su turno. Entonces, no hay una espera circular aquí.

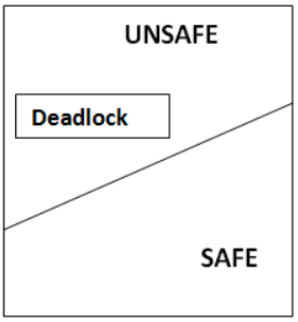
Al utilizar la estrategia de "**Avoidance**", tenemos que hacer una suposición. Necesitamos asegurarnos de que conocemos toda la información sobre los recursos que necesitará el proceso antes de la ejecución del proceso. Utilizamos el algoritmo de Banker (que a su vez es un regalo de Dijkstra) para evitar un punto muerto.

Un algoritmo de evitación de deadlock examina dinámicamente el estado de asignación de recursos para garantizar que nunca exista un caso de condición de espera circular. Donde el estado de asignación de recursos se define por los recursos disponibles y asignados y la demanda máxima del proceso. Hay 3 estados del sistema:

Estado seguro

Cuando un sistema puede asignar los recursos al proceso de tal manera que todavía evitan el punto muerto, el estado se denomina estado seguro. Cuando hay una salida de secuencia segura, podemos decir que el sistema está en estado seguro.

Una secuencia está en estado seguro solo si existe una secuencia segura. Una secuencia de proceso P1, P2, Pn es una secuencia segura para el estado de asignación actual si para cada Pi la solicitud de recursos que Pi todavía puede hacer, puede ser satisfecha por los recursos actualmente disponibles extrae los recursos retenidos por todos los Pj con j <i.

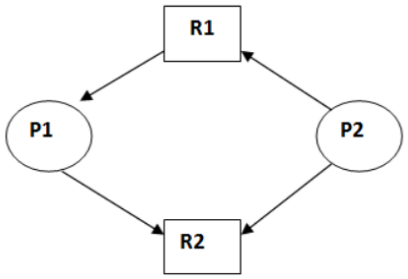


Métodos para evitar el estancamiento

1) Gráfico de asignación de recursos

Este gráfico también es una especie de algoritmo gráfico de los banqueros donde un proceso se denota con un círculo Pi y los recursos se denotan con un rectángulo RJ (.dots) dentro de los recursos que representan copias.

La presencia de un ciclo en el gráfico de asignación de recursos es una condición necesaria pero no suficiente para la detección de un deadlock. Si el tipo de cada recurso tiene exactamente una copia, entonces la presencia de ciclo es necesaria, así como una condición suficiente para la detección de deadlock.



Esto está en un estado inseguro (existe un ciclo) si P1 solicita P2 y P2 solicitan R1, se producirá un deadlock.

2) **Algoritmo bancario**

Los algoritmos del gráfico de asignación de recursos no son aplicables al sistema con múltiples instancias del tipo de cada recurso. Entonces, para este sistema se utiliza el algoritmo de Banker.

Aquí, cada vez que un proceso ingresa al sistema, debe declarar la máxima demanda posible.

En tiempo de ejecución, mantenemos alguna estructura de datos como la asignación actual, la necesidad actual, la disponibilidad actual, etc. Siempre que un proceso solicita algunos recursos, primero verificamos si el sistema está en un estado seguro o no, es decir, si cada proceso requiere el máximo de recursos, entonces hay una secuencia de hormigas en el cual la solicitud puede ser atendida, si la respuesta es afirmativa, la solicitud se asigna; de lo contrario, se rechaza.

2. **Detección y recuperación**: deje que se produzca un Deadlock y, a continuación, realice la preferencia para manejarlo una vez que se produzca.

Terminación del proceso:
El deadlock se puede eliminar abortando un proceso o más. El aborto se procesa a la vez hasta que se elimina el ciclo de deadlock. Esto puede ayudar a recuperar el sistema del deadlock de archivos.

Prevención de recursos:
Para eliminar el deadlock utilizando la preferencia de recursos, solicitamos los mismos recursos que pasen los procesos y le damos estos recursos a otro proceso hasta que se rompa el ciclo del deadlock .
Aquí, un proceso retrocede parcialmente hasta que se ejecuta el último punto de control o hasta que se ejecuta el algoritmo de detección.

3. **Ignore el problema por completo**: si el Deadlock es muy raro, deje que suceda y reinicie el sistema. Este es el enfoque que adoptan tanto Windows como UNIX.

Si deseamos rendimiento, el sistema debe ignorar el deadlock; de lo contrario, si el sistema está tratando con algunos datos muy importantes y no puede perderlos, entonces definitivamente debería optar por la prevención del deadlock.

En cuanto a si alguno de los SO es configurable o no, no lo son, como explique en el tipo de manejo de deadlock “ignore el problema por completo”, Windows y Unix utilizan solo ese enfoque, únicamente detectan el deadlock al momento de ejecución y se “quejan”. Esto lo hacen con la tecnología lockdep, la cual es un validador de corrección de bloqueo en tiempo de ejecución

Fuente de información:
https://www.geeksforgeeks.org/introduction-of-deadlock-in-operating-system/
https://afteracademy.com/blog/what-are-deadlock-handling-techniques-in-operating-system
https://www.includehelp.com/operating-systems/deadlock-and-method-for-handling-deadlock.aspx
https://docs.oracle.com/javadb/10.10.1.2/devguide/cdevconcepts28436.html
https://stackoverflow.com/questions/13658645/kernel-dealing-with-deadlocks-in-unix
https://www.javatpoint.com/os-strategies-for-handling-deadlock