Interbloqueo (DeadLock)

Si un conjunto de procesos esta en estado de espera por recursos y nunca cambia de estado porque los recursos por los que espera están siendo utilizados por otros procesos en estado de espera tenemos un DEADLOCK. Los recursos son la memoria, dispositivos de E/S, semáforos, archivos, etc. La forma en que un proceso hace uso de un recurso es:

Solicitud: si el recurso esta disponible se le otorga, si no el proceso pasa a estado de espera.
Uso: El proceso utiliza el recurso
Liberación: el proceso debe liberar el recurso.

Condiciones Necesarias para que se produzca DEADLOCK

Si se presentan simultáneamente las cuatro siguientes condiciones el sistema esta en DEADLOCK.

1. EXCLUSION MUTUA: Por lo menos un proceso que tenga otorgado un recurso en forma exclusiva.
2. USO Y ESPERA: Debe existir al menos un proceso que este haciendo uso de un recurso y que este esperando por otros recursos asignados a otros procesos.
3. NO INTERRUPCION: Los recursos no pueden ser retirados al proceso. Si un proceso hace uso de un recurso no le podrá ser retirado hasta que voluntariamente el proceso lo libere.
4. ESPERA CIRCULAR: Debe existir un conjunto de procesos {P1.....Pn} tal que P1 espera por un recurso utilizado por P2,.....,Pn espera por un recurso utilizado por P1.

Resourse Allocation Graph(Grafo de utilizacion de recursos)
Conjunto de Vértices: U =P U R
P={P1,P2,....,Pn}
R={R1,R2,...,Rn}

Conjunto de Aristas

Una arista de un proceso Pi a un Recurso Rj significa que el proceso i esta haciendo una solicitud por el recurso Rj.
Una arista del recurso Rj al proceso Pi, significa que el recurso esta asignado al proceso.
Si un RAG no tiene ciclos el sistema no esta en DEADLOCK, si los tiene no se puede afirmar nada.

Mecanismos para tratar con Deadlocks

Evasion de Deadlocks

Si se tiene cuidado al en la forma de asignar los recursos se pueden evitar situaciones de Deadlock. Supongamos un ambiente en el que todos los procesos declaren a priori la cantidad máxima de recursos que habá de usar.

Estado Seguro

Un estado es seguro si se pueden asignar recursos a cada proceso (hasta su máximo) en algún orden sin que se genere Deadlock. El estado es seguro si existe un ordenamiento de un conjunto de procesos {P1...Pn} tal que para cada Pi los recursos que Pi podrá utilizar pueden ser otorgados por los recursos disponibles mas los recursos utilizados por los procesos Pj,j<i. Si los recursos solicitados por Pi no pueden ser otorgados, Pi espera a que todos los procesos Pj hayan terminado.

Algoritmo del banquero de Dijkstra

Asigna peticiones de recursos siempre que las mismas den como resultado estados seguros. Solicitudes que den como resultado estados inseguros serán negadas hasta que puedan ser satisfechas. Este algoritmos evita situaciones de Deadlock asignando los recursos en forma correcta.
Las debilidades del algoritmo son: que requiere que la cantidad de recursos del sistema sea constante, requiere una cantidad de procesos constante y requiere que los procesos garanticen que los recursos van a ser devueltos en un intervalo finito de tiempo.

En el área de la informática, el problema del deadlock ha provocado y producido una serie de estudios y técnicas muy útiles, ya que éste puede surgir en una sola máquina o como consecuencia de compartir recursos en una red.
En el área de las bases de datos y sistemas distribuidos han surgido técnicas como el 'two phase locking' y el 'two phase commit' que van más allá de este trabajo. Sin embargo, el interés principal sobre este problema se centra en generar técnicas para detectar, prevenir o corregir el deadlock.

Las técnicas para prevenir el deadlock consisten en proveer mecanismos para evitar que se presente una o varias de las cuatro condiciones necesarias del deadlock. Algunas de ellas son:

- **Asignar recursos en orden lineal:** Esto significa que todos los recursos están etiquetados con un valor diferente y los procesos solo pueden hacer peticiones de recursos 'hacia adelante'. Esto es, que si un proceso tiene el recurso con etiqueta '5' no puede pedir recursos cuya etiqueta sea menor que '5'. Con esto se evita la condición de ocupar y esperar un recurso.

- **Asignar todo o nada:** Este mecanismo consiste en que el proceso pida todos los recursos que va a necesitar de una vez y el sistema se los da solamente si puede dárselos todos, si no, no le da nada y lo bloquea.

- **Algoritmo del banquero:** Este algoritmo usa una tabla de recursos para saber cuántos recursos tiene de todo tipo. También requiere que los procesos informen del máximo de

recursos que va a usar de cada tipo. Cuando un proceso pide un recurso, el algoritmo verifica si asignándole ese recurso todavía le quedan otros del mismo tipo para que alguno de los procesos en el sistema todavía se le pueda dar hasta su máximo. Si la respuesta es afirmativa, el sistema se dice que está en 'estado seguro' y se otorga el recurso. Si la respuesta es negativa, se dice que el sistema está en estado inseguro y se hace esperar a ese proceso.

Para detectar un deadlock, se puede usar el mismo algoritmo del banquero, que aunque no dice que hay un deadlock, sí dice cuándo se está en estado inseguro que es la antesala del deadlock. Sin embargo, para detectar el deadlock se pueden usar las 'gráficas de recursos'. En ellas se pueden usar cuadrados para indicar procesos y círculos para los recursos, y flechas para indicar si un recurso ya está asignado a un proceso o si un proceso está esperando un recurso. El deadlock es detectado cuando se puede hacer un viaje de ida y vuelta desde un proceso o recurso. Por ejemplo, suponga los siguientes eventos:

evento 1: Proceso A pide recurso 1 y se le asigna.
evento 2: Proceso A termina su time slice.
evento 3: Proceso B pide recurso 2 y se le asigna.
evento 4: Proceso B termina su time slice.
evento 5: Proceso C pide recurso 3 y se le asigna.
evento 6: Proceso C pide recurso 1 y como lo está ocupando el proceso A, espera.
evento 7: Proceso B pide recurso 3 y se bloquea porque lo ocupa el proceso C.
evento 8: Proceso A pide recurso 2 y se bloquea porque lo ocupa el proceso B.

En la figura 5.1 se observa como el 'resource graph' fue evolucionando hasta que se presentó el deadlock, el cual significa que se puede viajar por las flechas desde un proceso o recurso hasta regresar al punto de partida. En el deadlock están involucrados los procesos A,B y C.

Una vez que un deadlock se detecta, es obvio que el sistema está en problemas y lo único que resta por hacer es una de dos cosas: tener algún mecanismo de suspensión o reanudación que permita copiar todo el contexto de un proceso incluyendo valores de memoria y aspecto de los periféricos que esté usando para reanudarlo otro día, o simplemente eliminar un proceso o arrebatarle el recurso, causando para ese proceso la pérdida de datos y tiempo.

Dead lock en sistemas de spool.

Los sistemas de spool suelen ser los más propensos al dead lock. Varios trabajos parcialmente complejos que generan líneas de impresión a un archivo de spool pueden interbloquearse si el espacio disponible para trabajar se llena antes de completarse alguno de esos trabajos. La solución más común es limitar los spoolers de entrada de modo que no se lean más archivos cuando se llega al límite de su capacidad.

Postergación indefinida.

En los sistemas que mantienen procesos en espera mientras realizan la asignación de recursos y/o procesan la planificación de decisiones, es posible que un proceso sea postergado de manera indefinida mientras otro reciben la atención del sistema. A esta situación se le conoce como postergación indefinida, es diferente del dead lock pero sus consecuencias son igual de negativas.

En algunos sistemas la postergación indefinida se evita aumentando la prioridad de un proceso mientras espera por un recurso, a esto se le llama envejecimiento.

Conceptos sobre recursos.

Un sistema operativo es sobre todo un administrador de recursos, existen básicamente dos tipos de recursos:

* **Recursos no apropiativos.** Un recurso que no se puede liberar antes de completar su actividad sin perder la validez del proceso que lo usa, se dice que un recurso no apropiativo. Una impresora o una unidad de cinta no pueden ser liberados hasta que no termine su trabajo.
* **Recursos apropiativos**. Un recurso que puede ser usado temporalmente por varios procesos sin comprometer el correcto funcionamiento de dichos procesos se dice que es un recurso apropiativo. El **CPU** y la memoria principal (mediante las técnicas de paginación) son recursos que pueden ser asignados temporalmente por varios procesos. La apropiatividad de recursos es extremadamente importante en los sistemas de multiprogramación.

Los datos y los programas son recursos que tienen características especiales. En un sistema multiusuario donde se pueden compartir editores, compiladores y programas en general, es ineficiente cargar una copia de ellos para cada usuario que lo solicite. En lugar de ello se carga una sola vez a la memoria y se hacen varias copias de los datos, una por cada usuario.

El código que no cambia mientras está en uso se llama **código reéntrate**. El código que puede ser cambiado pero que se inicializa cada vez que se usa se llama **reutilizable en serie**. El código reéntrate puede ser compartido simultáneamente por varios procesos mientras que el reutilizable en serie sólo puede ser usado por un proceso a la vez.

Métodos para manejar los Dead Lock,

- ***Prevención***
- ***No permitirlos***

- Evitarlos

- Difícil y caro

- Permitirlos y recuperarlos

- Por perdida de información

Figura # 24. Prevención de Dead Lock.

En principio existen cuatro áreas importantes en la investigación del dead lock, a saber:

1) Prevención:

En las técnicas de prevención el interés se centra en condicionar un sistema para que elimine toda probabilidad de que ocurra un dead lock (normalmente a costa de recursos).

2) Evitación:

En las técnicas para evitar, la idea es poner condiciones menos estrictas que la prevención, para lograr una mejor utilización de los recursos, pero procurando no caer en un dead lock.

3) Detección:

Los métodos de detección se usan en sistemas que permiten la ocurrencia de un dead lock de forma voluntaria o involuntaria.

4) Recuperación:

Los métodos de recuperación se usan para romper los dead lock en un sistema, para poder liberarlo de ellos y los procesos estancados pueden continuar y llegar a su fin

Modelo del sistema.

Un sistema se compone de un número finito de recursos que se distribuyen entre varios procesos que compiten por ellos. Los recursos se dividen en varios tipos, cada uno de los cuales consiste en varios ejemplares idénticos. Los ciclos del CPU, el espacio de memoria, los archivos y los dispositivos de E/S (como impresoras y unidades de cinta) son ejemplos de tipos de recursos.

Un proceso debe solicitar un recurso antes de usarlo, y liberarlo al terminar su uso. Un proceso puede solicitar cuantos recursos quiera para llevar a cabo su tarea. Obviamente, el número no puede exceder del total de recursos disponibles del sistema. En otras palabras, un proceso no puede solicitar tres impresoras si el sistema solo dispone de dos.

En el modo de operación normal, un proceso solo puede utilizar un recurso en la secuencia siguiente:

1. Solicitud. Si la solicitud no puede atenderse de inmediato (por ejemplo, otro proceso está utilizando ese recurso), entonces el proceso solicitante debe esperar hasta que pueda adquirir el recurso.
2. Utilización. El proceso puede trabajar con el recurso (por ejemplo, si el recurso es una impresora, el proceso puede imprimir en ella).
3. Liberación. El proceso libera el recurso.

La solicitud y liberación de recursos son llamadas al sistema. Algunos ejemplos son las llamadas Solicitar y Liberar dispositivos, Abrir y Cerrar archivos, y Asignar y Liberar memoria. La solicitud y liberación de otros recursos puede lograrse atreves de las operaciones espera (P) y señal (V) sobre semáforos. Además la utilización de recursos solo puede conseguirse mediante llamadas al sistema (por ejemplo, para leer o escribir en un archivo o dispositivo de E/S), de modo que para cada utilización, el sistema operativo verifica que el proceso que dispone del recurso ya lo había solicitado y ya se le había asignado. Una tabla del sistema registra si cada uno de los recursos está libre o asignado y, de estar asignado, a qué proceso. Si un proceso solicita un recurso que se encuentra asignado a otro, puede añadirse a la cola de procesos que esperan tal recurso.

Un conjunto de procesos se encuentra en estado de bloqueo mutuo cuando cada uno de ellos espera un suceso que sólo puede originar otro proceso del mismo conjunto.

Caracterización.

Debe ser obvio que los bloqueos mutuos son indeseables, pues cuando se dan, los procesos nunca terminan su ejecución y los recursos del sistema se paralizan, impidiendo que se inicien otros procesos. Antes de analizar los distintos métodos para tratar el problema del bloqueo mutuo se describirán las circunstancias que los caracterizan.

Condiciones necesarias para que ocurra un Dead Lock.

Coffman, Elphick y Shoshani. Establecieron las cuatro condiciones necesarias para que se produzca un dead lock.

1.- Los procesos reclaman control exclusivo de los recursos que solicitan (**exclusión mutua**).

2.- Los procesos mantienen los recursos que se les han asignado mientras esperan por recursos adicionales (**condición de espera**).

3.- No se pueden tomar los recursos que los procesos tienen hasta su completa utilización (**condición de no apropiatividad**).

4.- Existe una cadena circular de procesos en la cual cada uno de ellos mantiene uno o más recursos que son requeridos por el siguiente proceso de la cadena (**condición de espera circular**).

Se deben presentar las cuatro condiciones para que puede aparecer un Dead Lock. La condición de espera circular implica la condición de retención y espera, por lo que las cuatro condiciones no son completamente independientes.

Prevención

En las estrategias de prevención de dead Locks, los recursos son otorgados a los procesos solicitados, de tal manera que la solicitud de un recurso nunca llega a un Dead Lock. Esta estrategia se cerciora de que cuando menos una de cuatro condiciones de Dead Lock nunca llegue a ocurrir.

*** Exclusión mutua.**

La condición de exclusión mutua debe conservarse para recursos no compartibles. Los recursos compartibles, no requieren acceso mutuamente excluyente y, por tanto, no pueden participar en un dead lock. Los archivos de sólo lectura son un buen ejemplo de recursos compartibles. Si varios procesos tratan de abrir al mismo tiempo un archivo de sólo lectura, se les puede otorgar accesos simultáneos al archivo, por lo general no es posible evitar dead

lock`s negando la condición de exclusión mutua. Por su naturaleza algunos recursos no pueden compartirse.

***** Negación de la condición de **"*espera por*"**.

La primera estrategia de **Havender** requiere que todos los recursos que va a necesitar un proceso sean pedidos de una sola vez. El sistema deberá asignarlos según el principio **"todo o nada"**. Si el conjunto de recursos que necesita un proceso está disponible se le asigna (todo) y se le permite entrar en ejecución. Si no es así, el proceso deberá esperar hasta que su conjunto de recursos esté disponible. Mientras un proceso espera. No podrá tener ningún recurso.

Esta estrategia presenta las siguientes desventajas:

- Puede llevar a la postergación indefinida, ya que quizá todos los recursos requeridos estén disponibles al mismo tiempo (costos de operación altos).
- Puede llevar un serio desperdicio de recursos, se requiere tener una gran cantidad de recursos para poder responder a cumplir petición.
- Es ineficiente por que decrementa la concurrencia del sistemas

Una variante consiste en dividir el programa en bloques que puedan ser ejecutados con relativa independencia uno del otro. Esto reduce el desperdicio, pero implica un trabajo extra en el diseño de las aplicaciones.

Negación de la condición de "*no apropiatividad*

Cuando un sistema cuenta con los recursos suficientes para que los procesos puedan funcionar sin problemas (bloqueos). Cuando el sistema permite compartir los recursos y los procesos mantienen los que otros necesitan sucede un dead lock.

La segunda estrategia sugiere que cuando a un proceso que mantiene recursos se le niegue una petición de recursos adicionales deberá liberar sus recursos y, si es necesario, pedirlos de nuevo, junto con los adicionales.

Al retirar los recursos a un proceso que no puede avanzar se niega la condición de **"no apropiatividad"**. Un problema de esta política es la postergación indefinida.

Negación de la condición de "espera circular"

Si se da a los recursos una manera exclusiva y se obliga a los procesos a pedirlos en forma ascendente es imposible que ocurra una espera circular. Esta propuesta considera:

- Los recursos deben ser numerados reflejando el orden en el cual la mayoría de los trabajos los usan**.**
- Los procesos deben pedir los recursos en forma ascendente**.**
- Para los procesos que requieren de los recursos en un orden diferente, los recursos deberán ser tomados y retenidos mucho antes de su utilización. (Implica el desperdicio de los recursos mantenidos pero no usados).

Ordenamiento lineal del recurso.

El ordenamiento lineal elimina la posibilidad de la espera circular, pero, obliga al diseñador a trabajar más con sus códigos. Además, los números de recursos son asignados por la instalación y hay que vivir con ellos durante largos periodos (meses o años). Si los números cambian se tienen que rescribir los programas y los sistemas existentes.

La forma más simple de prevenir un Dead Lock es obteniendo todos los recursos necesarios antes que el proceso continué y así se elimine la **"condición de espera".**

En otro método de prevención de dead lock, un proceso bloqueado devuelve los recursos solicitados por un proceso activo, al igual que elimina la condición de **"no apropiatividad"**

Rosenkrantz et al Ha propuesto la siguiente optimización de este método. Todos los procesos son asignados a prioridades únicas que pueden ser totalmente ordenadas. Un proceso de retención cede el derecho de prioridad a otro proceso que sostiene el recurso necesario solo si el proceso de retención tiene el recurso necesario solo si el proceso de retención tiene mayor prioridad.

Este método reduce los derechos de prioridad al 50% al igual que previene los dead locks. En el ordenamiento de Havender's todos los recursos son ordenados de manera única y todos los procesos solicitan los recursos de manera ascendente.

Únicamente eliminando la condición de "espera circular". Si un proceso ya sostiene algunos recursos, entonces puede pedir solo aquellos acomodos en un rango superior en el orden. Obteniendo recursos, de ésta forma, previene la formación de un ciclo o de un "knot".

Prevenir

Se puede prevenir el bloqueo siempre y cuando se consiga que alguna de las condiciones necesarias para la existencia de un bloqueo no se produzca.

1. los procesos tienen que compartir recursos con exclusión mutua:
    - No se de a un proceso directamente acceso exclusivo al recurso, si no se usa otro proceso que realiza el trabajo de todos los demás manejando una cola de tareas (por ejemplo, un demonio para imprimir con su cola de documentos por imprimir).
2. los procesos quieren acceder a un recurso más mientras ya tienen acceso exclusivo a otro:
    - Se exige que un proceso pida todos los recursos que va a utilizar al comienzo de su trabajo
3. los recursos no permiten ser usados por más de un proceso al mismo tiempo:
    - Se permite que un proceso aborte a otro proceso con el fin de obtener acceso exclusivo al recurso. Hay que tener cuidado de no caer en *livelock*
4. Existe una cadena circular entre peticiones de procesos y locación de recursos:
    - Se ordenan los recursos linealmente y se fuerza a los procesos que accedan a los recursos en el orden impuesto. Así es imposible que se produzca un ciclo.

En las prácticas aplicamos dicho método para prevenir bloqueos en la lista concurrente.

Prevención de Deadlocks

La estrategia consiste en anular alguna de las cuatro condiciones necesarias para que se produzca un Deadlock.

1. No puede ser anulada porque existen recursos que deben ser usados en modalidad exclusiva.
2. La alternativa sería hacer que todos los procesos solicitaran todos los recursos que habrán de utilizar antes de utilizarlos al momento de su ejecución lo cual sería muy ineficiente.
3. Para anular esta condición cuando un proceso solicita un recurso y este es negado el proceso deberá liberar sus recursos y solicitarlos nuevamente con los recursos adicionales. El problema es que hay recursos que no pueden ser interrumpidos.
4. Espera Circular: esta estrategia consiste en que el sistema operativo numere en forma exclusiva los recursos y obligue a los procesos a solicitar recursos en forma ascendente. El problema de esto es que quita posibilidades a la aplicación.

Deadlock no puede ocurrir a menos que tenemos todas las cuatro condiciones. Si aseguramos que no puede ocurrir por lo menos una de las condiciones, no podemos tener deadlock.

- **Exclusión mutua.** En general, no podemos eliminar esta condición. Hay recursos como impresoras que no son compatibles.

- **Retención y espera.** Para no ocurrir esta condición, cuando un proceso solicita recursos, no puede retener otros. Protocolos:
    - Un proceso puede solicitar recursos solamente si no tiene ningunos.
    - Un proceso tiene que solicitar todos los recursos antes de la ejecución.

Problemas:

    - La utilización de recursos puede ser baja.
    - Starvation (bloqueo indefinido) si se necesitan algunos recursos populares.
- **No apropiación.** Si un proceso retiene varios recursos y solicita otro que no está disponible, se le expropian todos los recursos que retiene. El proceso tiene que recuperar todos los recursos antes de ejecutar otra vez.

Pero en general no podemos expropiar recursos como impresoras y grabadores.

- **Espera circular.** Hacemos una ordenación de los tipos de recursos en el sistema (R_1, R_2, ...). Un proceso tiene que solicitar sus recursos en orden (y todos los ejemplares de un tipo al mismo tiempo). Si necesita un tipo de recurso más baja en la ordenación, tiene que liberar los otros que retiene.
- Problemas con la prevención de deadlock: Utilización baja de recursos y reducción de la productividad del sistema.

Evitar

El sistema no da permiso de acceso a recursos si es posible que el proceso se bloquee en el futuro. Un método es el algoritmo del banquero (Dijkstra) que es un algoritmo centralizado y por eso posiblemente no muy practicable en muchas situaciones.
Se garantiza que todos los procesos actúan de la siguiente manera en dos fases:

1. Primero se obtiene todos los cerrojos necesarios para realizar una tarea, eso se realiza solamente si se puede obtener todos a la vez,
2. Después se realiza la tarea durante la cual posiblemente se liberan recursos que no son necesarias.

Ejemplo:
Asumimos que tengamos 3 procesos que actúan con varios recursos. El sistema dispone de 12 recursos.

proceso	recursos pedidos	recursos reservados
A	4	1
B	6	4
C	8	5
suma	18	10

es decir, de los 12 recursos disponibles ya 10 están ocupados. La única forma que se puede proceder es dar el acceso a los restantes 2 recursos al proceso B. Cuando B haya terminado va a liberar sus 6 recursos que incluso pueden estar distribuidos entre A y C, así que ambos también pueden realizar su trabajo.

Con un argumento de inducción se verifica fácilmente que nunca se llega a ningún bloqueo.

Evitación

Un método para evitar los Dead Lock`s consiste en requerir información adicional sobre cómo se solicitarán los recursos. Por ejemplo en un sistema con una unidad de cinta y una impresora, podríamos saber que el proceso P solicitará primero la unidad de cinta y luego la impresora, antes de liberar ambos recursos. El proceso Q, por otra parte, solicitará primero la impresora y después la unidad de cinta. Con este conocimiento de la secuencia completa de la solicitud y liberación para cada proceso para cada solicitud requiere que el sistema considera los recursos disponibles en ese momento, los actualmente asignados a cada proceso y las futuras solicitudes y liberaciones de cada proceso para decidir si puede satisfacer la solicitud presente o debe esperar para evitar un posible dead lock futuro.
Los diversos algoritmos difieren en la cantidad y tipo de información que requieren.

El modelo más sencillo y útil requiere que cada proceso declare el *número máximo* de recursos de cada tipo que puede necesitar. Con información a priori para cada proceso es posible construir un algoritmo que asegure que el sistema nunca entrará en estado de dead lock. Este algoritmo define la estrategia de evitación de dead lock`s.

El estado de asignación de recursos viene definido por el número de recursos disponibles y asignados, y por la demanda máxima de los procesos. Un estado es seguro si el sistema puede asignar recursos a cada proceso (hasta el máximo) siguiendo algún orden u aun así evitar el dead lock.

Más formalmente, un sistema se encuentra en estado seguro sólo si existe una secuencia segura. Si no existe esta secuencia, se dice que el estado del sistema es inseguro.

Un estado seguro no es un estado de dead lock, y un estado de dead lock es un estado inseguro; pero no todos los estados inseguros son dead lock`s.

Detección

Se implementa un proceso adicional que vigila si los demás forman una cadena circular.
Más preciso, se define el grafo de asignación de recursos:

- Los procesos y los recursos representan los nodos de un grafo.
- Se añade cada vez una arista entre un nodo tipo recurso y un nodo tipo proceso cuando el proceso ha obtenido acceso exclusivo al recurso.
- Se añade cada vez una arista entre un nodo tipo recurso y un nodo tipo proceso cuando el proceso está pidiendo acceso exclusivo al recurso.
- Se eliminan las aristas entre proceso y recurso y al revés cuando el proceso ya no usa el recurso.

Cuando se detecta en el grafo resultante un ciclo, es decir, cuando ya no forma un grafo acíclico, se ha producido una posible situación de un bloqueo.
Se puede reaccionar de dos maneras si se ha encontrado un ciclo:

- No se da permiso al último proceso de obtener el recurso.
- Sí se da permiso, pero una vez detectado el ciclo se aborta todos o algunos de los procesos involucrados.

Sin embargo, las técnicas pueden dar como resultado que el programa no avance, incluso, el programa se puede quedar atrapado haciendo trabajo inútil: crear situaciones de bloqueo y abortar procesos continuamente.

Detección y Recuperación de Deadlocks

1. Cuando hay una única ocurrencia de cada recurso. (variante del grafo de "*wait for graph*"). Consiste en reducir el grafo, retirando las aristas que van de recursos a procesos y de procesos a recursos, colocando nuevas aristas que reflejan la situación de espera entre procesos. Si el grafo reducido tiene ciclos el sistema esta en Deadlock. Orden de operaciones = n^2 , n = cantidad de aristas.
2. Cuando hay múltiples ocurrencias de cada recurso

```
Procedure detecta_deadlock
var Disponible:array[1..n] of integer //# de recursos
Usados:array[1..n] of integer
Solicitados:array[1..n] of integer
Finalizado:array[1..n] of boolean
Begin
Work:=Disponibles;
For i:=1..n do if(usados[i]≠0) then finish[i]:=false
Else finish[i]:=true;
While(encontrar_indice_i = true) do {
Work:=work + usados[i];
```

Finish[i]:=true; }
If ((finish[i] = false) para algun i), 1≤i≤n then à El sistema esta en Deadlock.
End
Function encontrar_indice_i : Boolean
Begin
If (existe i tal que (Finish[i]=false && solicitados[i] ≤ work)
Then -> true
Else -> false
End

La evitacíon de deadlock tiene un costo porque todos los estados inseguros no son estados de deadlock. Esto implica que hay tiempos cuando algunos procesos tienen que esperar y recursos están desocupados sin que es necesario.

El sistema operativo puede chequear de vez en cuando si hay un deadlock. ¿Cuán frecuentemente debieramos chequear si hay deadlock?

- El costo de algoritmo vs. el número de procesos en deadlock.
- Saber quien creó el deadlock.
- Cuando la utilización de la CPU es baja.

Tenemos que eliminar los deadlocks que encontramos. Posibilidades:

- Abortar todos los procesos en deadlock. ¡Esto es un método común!
- Abortar procesos uno a la vez hasta que no haya deadlock.
- Retroceder los procesos a algún punto y reiniciar. El deadlock puede reocurrir.
- Expropiar recursos de procesos hasta que no haya deadlock. Retrocedemos los procesos expropiados.

Tenemos que seleccionar un proceso para abortar o retroceder.
Factores:

- La prioridad
- El tiempo que el proceso ha corrido
- El número y tipo de los recursos adquiridos
- La clase de proceso (batch o interactiva)

La starvation es un problema.

Detección y Recuperación

Es el hecho de determinar si existe o no un Dead Lock, e identificar cuales son los procesos y recursos implicados en él. El uso de algoritmos de detección de interbloqueo implica una sobrecarga en el tiempo de ejecución. Para ser correcto, un algoritmo de detección de interbloqueo debe de cumplir dos criterios:

1) Debe detectar todos los dead lock´s existentes en un tiempo finito.

2) No debe reportar "falsos" Dead Lock's.

Grafo de asignación de recursos.

Los Dead Lock pueden describirse con mayor precisión en función de un grafo dirigido llamado grafo de asignación de recursos, que consiste en un conjunto de vértices V y aristas A. El conjunto de vértices V se divide en dos tipos, P = {P1,P2, ... , Pn}, el conjunto formado por todos los procesos del sistema, y R ={R1,R2, ... ,Rn}, el conjunto integrado por todos los tipos de recursos del sistema.

Representación mediante grafos del estado del sistema.

El estado de un sistema es dinámico; esto es, los procesos del sistema toman y liberan recursos continuamente. La representación del dead lock requiere la representación del estado de la interacción procesos - recursos, la representación se hace mediante un grafo dirigido que se conoce como gráfica de asignación de recursos .

En los sistemas de bases de datos distribuidos (**DDBS**) está representación se conoce como gráfica de espera de transacción (**Transaction Wait-For TWF**).

Los dead lock`s pueden describirse con mayor precisión en función de un grafo dirigido llamado grafo de asignación de recursos.
La simbologia es la siguiente **. a, b, c y d.**

GRÁFICA DE PETICIÓN Y ASIGNACIÓN DE RECURSOS

La técnica para la reducción de gráficas implica las consideraciones siguientes:

- Si las peticiones de recursos de un proceso piden ser concedidas, entonces la gráfica puede ser reducida.
- La reducción de una gráfica consiste en retirar las flechas que van de los recursos a los procesos y retirar las flechas que van del proceso al recurso.
- Si una gráfica puede ser reducida para todos sus procesos entonces no hay dead lock.
- Si una gráfica no puede ser reducida para todos sus procesos, entonces los procesos irreducibles constituyen la serie de procesos interbloqueados de la gráfica.
- Detección de interbloqueo mediante grafos.

*Un grafo **G** consiste de un conjunto finito no vacío.*

V = C(G) de: **P** puntos (vértices) conjunto **X** de **q** parejas desordenadas de puntos de **V(aristas).**

cada par **X = {U,V}** de puntos en **X** y una línea de **G** por lo tanto **X = UV.**
Un grafo de **p** puntos y **q** líneas se denomina un grafo **(p,q)**, el grafo **(1,0)** es trivial.

Petición = Proceso - Recurso
Pi

Rx **Ry** El proceso **Pi** tiene el recurso **Rx** y solicita el recurso **Ry**.

Para determinar si existe un ciclo en un grafo se puede usar la representación matricial del grafo dirigido. Dado que se tienen parejas ordenadas **{Rx, Ry}**, la matriz se construye colocando un 1 en la diagonal en **(x,x)** y **(y,y)** y un 1 en la posición **(x,y)** de la matriz. **.**

Reducción de la matriz del grafo.
Reducción de la matriz de un grafo correspondiente. No existen vértices terminales; los vértices 1 y 2 son iniciales y pueden ser eliminados; existe un Inter bloqueo.

Representación vectorial

Un vértice terminal sólo aparece en la columna requiere y un vértice inicial sólo aparece en la columna Tiene. Para reducir esta representación se recorren de arriba a abajo los vectores y se buscan los vértices terminales e iniciales y se elimina su renglón.

El proceso se repite hasta:

1) No existen renglones o

2) No se pueden eliminar renglones.

Si no se pueden eliminar renglones las transiciones producen un Dead Lock.

Para el grafo de la en el primer paso se eliminan los procesos P1 (vértice inicial), P2 y P3 (vértice terminal). En el segundo paso se elimina el proceso P4 (vértice terminal), inicial.

Para el grafo de la el primero se eliminan los procesos P1,P2,P3 (vértices iniciales), P4(vértice inicial al eliminar el proceso P1), Los procesos restantes no pueden ser eliminados, por lo tanto existe un dead lock. El resultado de la reducción

Recuperación

Recuperación ante Deadlocks

1. Cancelación de procesos
    1. Cancelación de todos los procesos involucrados. Esto resuelve la situación de Deadlock pero tiene un costo muy alto de reprocesamiento.
    2. Cancelacion de un proceso por vez hasta resolver la situación de Deadlock. La ventaja de esto es que el costo de reprosesamiento de la información es menor pero cada vez que se cancela un proceso debe ejecutarse el algoritmo de detección de deadlock. Los criterios para elegir el candidato a ser cancelado son: por prioridad, por tiempo de uso de CPU, por cantidad de recursos utilizados y por cuantos recursos adicionales habrá de utilizar.
2. Obtención de recursos (resourse Preemption). El sistema operativo selecciona un proceso y le quita los recursos otorgados. Los criterios para seleccionar el candidato son los mismos que para la cancelación. El proceso seleccionado se le quitan los recursos y se le lleva a un estado consistente (Rollback).

Recuperación de un Dead Lock.

La única forma en que el sistema operativo puede recuperarse de un interbloqueo es retirando uno o más procesos y reclamar sus recursos para que otros procesos puedan terminar. Normalmente varios procesos perderán una parte o la totalidad del trabajo realizado hasta ese momento. Este puede parecer un precio pequeño comparado con dejar que el interbloqueo se complique por varios factores.

- En primer lugar, puede no estar claro que el sistema este bloqueado o no.
- Muchos sistemas tienen medios pobres para suspender un proceso por tiempo indefinido y reanudarlo más tarde.

- Aún cuando existan medios efectivos de suspensión /reanudación, con toda seguridad, estos implicarán una sobrecarga, considerable y pueden requerir la atención de un operador altamente calificado. No siempre se dispone de tales operadores.
- Recuperarse de un interbloqueo de proporciones modestas puede suponer una cantidad razonable de trabajo.

Una forma de elegir los procesos que pueden ser retirados es de acuerdo a las prioridades de los procesos. Pero esto también tiene sus dificultades.

- Las prioridades de los procesos bloqueados pueden no existir, así que el operador deberá tomar una decisión arbitraria.
- Las prioridades pueden ser incorrectas, o un poco confusas debido a consideraciones especiales.
- La decisión óptima de cuáles procesos retirar pueden requerir de un esfuerzo considerable para determinarla.

El enfoque más deseable a la recuperación del Dead Lock están un mecanismo efectivo de
Suspensión / reanudación. Esto permitirá detener temporalmente los procesos y después reanudarlos sin pérdida del trabajo.

MECANISMOS PARA EVITARLOS.

Havender llegó a la conclusión de que si no se cumple una de las cuatro condiciones necesarias para el interbloqueo es posible que éste ocurra.

Para evitarlo sugirió:

- Cada proceso deberá pedir todos los recursos requeridos de una sola vez y no podrá proceder hasta que le hayan sido asignados todos.
- Si un proceso que mantiene ciertos recursos se le niega una nueva petición, este proceso deberá liberar sus recursos originales y en caso necesario pedirlos de nuevo con los recursos adicionales.
- Se impondrá la ordenación lineal de los tipos de recursos en todos los procesos, es decir, si a un proceso le han sido asignados recursos de un tipo dado, en lo sucesivo sólo podrá pedir aquellos recursos de los tipos que siguen en el ordenamiento.

Otra alternativa, para manejar los dead lock, es tener información acerca de como los recursos se van a requerir, el modelo más simple y más útil requiere que en cada proceso declare el máximo número de recursos que van a requerir con lo cual es posible construir un algoritmo que asegure que el sistema no entrara en dead lock.

Un estado es seguro si el sistema puede asignar recursos a cada proceso en algún orden evitando el dead lock.

Formalmente un sistema esta en estado seguro solamente si existe una secuencia segura. Una secuencia de procesos < P1, P2, ... Pn> esta en secuencia segura si para cada Pi, los recursos que Pi pueda requerir pueden ser satisfechos por los recursos disponibles más los recursos que tuvieron los Pj donde j < i. Si no se puede satisfacer Pi entonces Pi espera hasta que los Pj terminen.

Cuando Pi termine Pi+1 puede obtener sus recursos y así sucesivamente.

Se tienen **12 unidades de cinta** se tiene **3 procesos** ¿Ese sistema esta en estado seguro o no?.

Requerimiento procesos - recursos.

Requerimiento máximo - Necesidad actual = Necesidad más (Disponible).

Se tiene 12 recursos por lo tanto 12-9 = 3, entonces la secuencia es segura.

La secuencia segura es **< P1,P0,P2>**

Haga una situación en la cual el sistema estaría es estado inseguro.

Un estado seguro esta libre de dead lock y un estado de dead lock, es un estado inseguro pero no todos los estados inseguros producen dead lock.

Si se esta en un estado seguro se puede pasar a un estado inseguro.

Estrategias para evitarlos

Evitación del interbloqueo y el algoritmo de **Dijkstra**. Si las condiciones necesarias para que tenga lugar un interbloqueo están en su lugar, aún es posible evitar el interbloqueo teniendo cuidado al asignar los recursos.

El algoritmo de planificación que pueda evitar los interbloqueos fue ideado por **Dijkstra** (1965) y se le conoce como algoritmo del banquero. En ese algoritmo se modela la forma en que un banquero puede tratar a un grupo de clientes a quienes les ha otorgado líneas de crédito. en la (***figura # 36 (a)***) se observan cuatro clientes, a cada uno de los cuales se le ha

otorgado cierto número de unidades de crédito. El banquero sabe que los clientes no necesitan su límite de crédito máximo de inmediato, de manera que sólo ha reservado 10 unidades en lugar de 22 para darles servicio.

Los clientes emprenden sus respectivos negocios, haciendo solicitudes de préstamo de cuando en cuando. En cierto momento***)***. A una lista de clientes que muestra el dinero que ya se presentó y el máximo del crédito disponible se le llama estado del sistema.

Tres estados de asignación de recursos

(a) Seguro. (b) Seguro. (c) Inseguro.

Un estado es seguro si existe una secuencia de estados que lleva a todos los clientes que obtienen préstamos hasta sus límites de crédito. es seguro por que con las dos restantes, el banquero puede demorar cualquier solicitud salvo la de Carlos, con lo que permite que termine y devuelva sus cuatro recursos. Con cuatro unidades, el banquero puede permitir que David o Bárbara tengan las unidades que necesitan para terminar y así sucesivamente.

Si estando en el estado de la se le otorga una unidad más a Bárbara, el banquero no podrá completar ninguna de la línea de crédito de su clientes. Un estado inseguro no tiene que conducir a un interbloqueo, ya que un cliente podría no necesitar su línea de crédito disponible, pero el banquero no puede confiar en ese comportamiento.

Haciendo una analogía con un Sistema Operativo tenemos: El estado actual del sistema se denomina seguro, Si el Sistema Operativo puede permitir a los usuarios actuales completar sus trabajos en un intervalo de tiempo finito. De lo contrario se le denomina inseguro. En otras palabras: Un estado seguro es aquel en el cual la asignación de recursos es tal que los usuarios pueden llegar a terminar. Un estado seguro es aquel que puede llegar a terminar. Un estado inseguro es aquel que puede llegar a un dead lock, (nunca termina).

La asignación de recursos por el algoritmo del banquero es la siguiente:

- Se permite todas las condiciones de exclusión mutua, espera por y no apropiatividad.
- Se permite a los procesos mantener sus recursos mientras esperan por otros.
- El sistema sólo concede peticiones que den como resultado estados seguros.

El algoritmo del banquero para n recursos (de Dijkstra).

Sea n un número de proceso y m un número de recurso y sea disponible un vector de longitud m que indica el número de recursos disponibles y sea máxima una matriz de (n x m) que define la máxima demanda de cada proceso así por ejemplo si tuviéramos máxima (i,j) = k define los recursos asignados a cada proceso, por ejemplo la asignación (i , j) = k quiere decir que el proceso i tiene asignado K instancias del recurso j necesidades (n * m), que indica los recursos que requieren los procesos. Necesidades (i , j) = Máxima (i , j) - Asignación (i , j).

Se puede considera cada renglón de las matrices asignación y necesidades como vectores y referirnos como asignación de i y necesidades de i.

Algoritmo:

Sea requerimiento de i un vector de requerimientos de proceso i.

Cuando un requerimiento de un recurso es hecho por el proceso i se realizaran las siguientes acciones:

1).- Si Requerimiento de i <= Necesidades de i ir al segundo paso. Sino error.
2).- Si requerimiento de i <= disponible ir al tercer paso. Si no recurso no disponible Pi debe esperar.
3).-
Hacer Disponible = Disponible - Requerimiento,
Asignación = Asignación + Requerimiento,
Necesidades = Necesidades - Requerimiento.

Si el estado es seguro se completa la transacción de lo contrario el proceso debe esperar y el estado anterior es restablecido.

Algoritmo para ver si el estado es seguro.

Sea trabajo (m) y final (n) vectores de longitud m y n respectivamente, hacer Trabajo = Disponible y final (i) = falso para i = 1,...n

1).- Encuentre una i tal que final (i) de *i* = *falso* y *Necesidades (i)* <= *Trabajo* si no existe ir al punto tres.
2).- *Trabajo* = *Trabajo* + *Asignación (i)*, *Final (i)* = *verdad* ir al paso uno.

3).- Si *final (i)* = *verdad* para toda i entonces el sistema esta en estado seguro de lo contrario esta en estado inseguro.

Secuenciabilidad

Los archivos secuenciales son un tipo de archivo en los que la información puede leerse y escribirse empezando desde el principio del archivo.
Debemos tomar en consideración algunas características que deben tener los archivos secuenciales:

1. La escritura de nuevos datos siempre se hace al final del archivo.

2. Para leer una zona concreta del archivo hay que avanzar siempre, si la zona está antes de la zona actual de lectura, será necesario "rebobinar" el archivo.

3. Los ficheros sólo se pueden abrir para lectura o para escritura, nunca de los dos modos a la vez.

Archivos Secuenciales

Se refiere al procesamiento de los registros, no importa el orden en que se haga, para eso los registros están organizados en forma de una lista y recuperarlos y procesarlos uno por uno de principio a fin.

Rendimientos de los archivos Secuenciales; dependiendo del dispositivo de almacenamiento utilizado el archivo se puede mostrar el usuario como si fuera un sistema secuencial.

Al finalizar un archivo secuencial se denota con una marca de fin de archivo. (End end-of-file)

Seriabilidad: Cuando hablamos de seriabilidad nos referimos a la capacidad de reproducir un producto x en número limitado de veces.

TEORÍA DE SERIABILIDAD

Una forma de evitar los problemas de interferencia es no permitir que las transacciones se intercalen. Una ejecución en la cual ninguna de dos transacciones se intercala se conoce como *serial*. Para ser más precisos,

una ejecución se dice que es serial si, para todo par de transacciones, todas las operaciones de una transacción se ejecutan antes de cualquier operación de la otra transacción. Desde el punto de vista del usuario, en una ejecución serial se ve como si las transacciones fueran operaciones que el DBS procesa atómicamente. Las transacciones seriales son correctas dado que cada transacción es correcta individualmente, y las transacciones que se ejecutan serialmente no pueden interferir con otra.

Si el DBS procesara transacciones serialmente, significaría que no podría ejecutar transacciones concurrentemente, si entendemos concurrencia como ejecuciones intercaladas. Sin dicha concurrencia, el sistema usaría sus recursos en un nivel relativamente pobre y podría ser sumamente ineficiente.

Una solución puede ser ampliar el rango de ejecuciones permisibles para incluir aquellas que tienen los mismos efectos que las seriales. Dichas ejecuciones se conocen como *serializables*. Entonces, una ejecución es serializable si produce el mismo resultado y tiene el mismo efecto sobre la base de datos que alguna ejecución serial de las mismas transacciones. Puesto que las transacciones seriales son correctas, y dado que cada ejecución serializable tiene el mismo efecto que una ejecución serial, las ejecuciones serializables también son correctas.

Las ejecuciones que ilustran la actualización perdida y el análisis inconsistente no son serializables. Por ejemplo, ejecutando las dos transacciones de Depositar serialmente, en cualquier orden, da un resultado diferente al de la ejecución intercalada que pierde una actualización, por lo tanto, la ejecución intercalada no es serializable. Similarmente, la ejecución intercalada de Transferir e Imprimir Suma tiene un efecto diferente a los de cada ejecución serial de las dos transacciones, y por ello no es serializable.

Aunque éstas dos ejecuciones intercaladas no son serializables, muchas otras sí lo son. Por ejemplo, consideremos la siguiente ejecución intercalada de Transferir e Imprimir Suma. Esta ejecución tiene el mismo efecto que la ejecución serial de Transferir seguida de Imprimir Suma por lo tanto es serializable.

$Leer_4$ (Cuentas [8]) devuelve el valor de US**$200**
$Escribir_4$ (Cuentas [8], US**$100**)
$Leer_3$ (Cuentas [8]) devuelve el valor de US**$100**
$Leer_4$ (Cuentas [9]) devuelve el valor de US**200**
$Escribir_4$ (Cuentas [9], US**$300**)
$Commit_4$
$Leer_3$ (Cuentas [9]) devuelve el valor de US**$300**
$Commit_3$

La teoría de seriabilidad es una herramienta matemática que permite probar si un sincronizador trabaja o no correctamente. Desde el punto de vista de la teoría de seriabilidad, una transacción es una representación de una ejecución de operaciones de lectura y escritura y que indica el orden en el que se deben ejecutar estas operaciones. Además, la transacción contiene un Commit o un Abort como la última operación para indicar si la ejecución que representa terminó con éxito o no. Por ejemplo, la ejecución del siguiente programa.

Procedure P

begin

Start;
temp := Leer(**x**);
temp := temp **+** 1;
Escribir(**x**, temp);
Commit;

end

Puede ser presentado como $\mathbf{r_1[x]}$ -› $\mathbf{w_1[x]}$ -› $\mathbf{c_1}$. Los subíndices identifican esta transacción particular y la distinguen de cualquier otra transacción que acceda al mismo dato. En general, usaremos $\mathbf{r_1[x]}$ (o $\mathbf{w_1[x]}$ para denotar la ejecución de Leer (o Escribir) ejecutado por la transacción $\mathbf{T_1}$ sobre el dato **x**.

Cuando se ejecuta concurrentemente un conjunto de transacciones, sus operaciones deben estar intercaladas. La manera de modelar esto es usando una estructura llamada *historia*. Una historia indica el orden en el que se deben ejecutar las operaciones de las transacciones en relación a otras. Si una transacción $\mathbf{T_i}$ especifica el orden de dos de sus operaciones, estas dos operaciones deben aparecer en ese mismo orden en cualquier historia que incluya a $\mathbf{T_i}$. Además, es necesario que una historia especifique el orden de todas las *operaciones conflictivas* que aparezcan en ella.

Se dice que dos operaciones están en conflicto si ambas operan sobre el mismo dato y al menos una de ellas es una operación de escritura (Escribir). Por lo tanto, Leer(**x**) está en conflicto con Escribir(**x**), mientras que Escribir(**x**) está en conflicto tanto con Leer(**x**) como con Escribir(**x**). Si dos operaciones están en conflicto, es muy importante su orden de ejecución. Consideremos las siguientes transacciones

$T_1 = r_1[x]$ -› $w_1[x]$ -› c_1
$T_3 = r_3[x]$ -› $w_3[y]$ -› $w_3[x]$ -› c_3
$T_4 = r_4[y]$ -› $w_4[x]$ -› $w_4[y]$ -› $w_4[z]$ -› c_4

Una historia completa sobre $\mathbf{T_1}$, $\mathbf{T_3}$, $\mathbf{T_4}$ es

$H_1 = r_4[y]\ r_1[x]\ w_1[x]\ c_1\ w_4[x]\ r_3[x]\ w_4[y]\ w_3[y]\ w_4[z]\ w_3[x]\ c_4\ c_3$