Sistemas Operativos

Unidad 5: Fundamentos del interbloqueo.

"La programación secuencial es realmente difícil, y la programación paralela está un paso más allá de eso".

~Andrew S. Tanenbaum

Fundamentos del interbloqueo (deadlock)

Definición	**Deadlock**: Es el bloqueo permanente de un conjunto de procesos que o bien compiten por recursos del sistema o bien se comunican entre sí.

Un conjunto de procesos está interbloqueado cuando cada proceso se encuentra bloqueado a la espera de un evento (por lo general, la liberación de un recurso) que sólo puede generar otro proceso bloqueado del mismo conjunto.

Todos los interbloqueos involucran necesidades conflictivas que afectan a los recursos de 2 o más procesos.

Tener en cuenta que la aparición de un interbloqueo depende tanto de la dinámica de ejecución como de los detalles de implementación de los programas.

El sistema operativo y las aplicaciones interactúan / consumen recursos que podemos clasificar en recursos reutilizables y consumibles

Recurso Reutilizable

Un recurso reutilizable es aquél que sólo lo puede utilizar de forma segura un proceso en cada momento y que no se destruye con su uso, como por ejemplo el procesador, disco, memoria, archivos, etc.

Por ejemplo, supongamos que el espacio de memoria disponible para los procesos es de 200 Kbytes y tenemos los siguientes procesos concurrentes:

	Proceso A	Proceso B
1	solicita 80 Kbytes	solicita 70 Kbytes
2	...	...
3	solicita 60 Kybtes	solicita 80 Kbytes

Si cualquiera de los procesos llega hasta la línea 3 se producirá un interbloqueo.

Recurso Consumible

Un recurso consumible es aquél que puede crearse (producirse) y eliminarse (consumirse) indefinida cantidad de veces. Por ejemplo las interrupciones, señales, mensajes, etc.

Aunque los recursos puedan ser inagotables, pueden producirse también situaciones de interbloqueo. Por ejemplo, si consideramos la siguiente situación:

	Proceso A	Proceso B

1
2
3
	`receive(mensaje_de_B)`
	`...`
	`send(B, mensaje)`
		`receive(mensaje_de_A)`
		...
		`send(A, mensaje)`

Si en el ejemplo anterior, el receive es bloqueante, puede darse una secuencia de ejecución donde A se bloquee esperando un mensaje de B y a su vez B se bloquee esperando un mensaje de A.

La causa del interbloqueo es por lo general un problema de diseño de las aplicaciones, aunque son muy difíciles de detectar. La dificultad en la detección reside en que la aparición de un interbloqueo muchas veces depende de la secuencia de ejecución de las instrucciones.

Condiciones para el interbloqueo

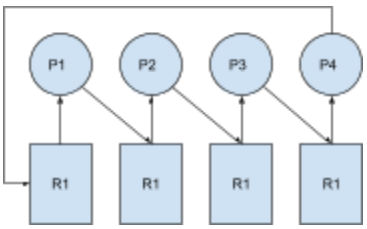
Como mencionamos anteriormente, deben cumplirse ciertas condiciones en el sistema para llegar a la condición de interbloqueo, a saber:

1. **Exclusión mutua**: sólo un proceso puede acceder a un recurso y realizar una acción/función sobre el mismo en un momento determinado.
2. **Retención y espera**: un proceso puede mantener los recursos que le asignaron mientras espera por la asignación de otros recursos.
3. **Sin expropiación**: el sistema operativo no puede forzar la expropiación de un recurso a un proceso que ya lo tiene asignado.

El hecho de que se cumplan estas 3 condiciones (al mismo tiempo), no asegura que exista interbloqueo entre procesos, es decir, estas 3 condiciones son condición necesaria pero no suficiente.

Para que realmente exista interbloqueo entre procesos, hace falta una cuarta condición que llamaremos espera circular.

4. **Espera circular**: existe una lista "cerrada" de procesos, de forma tal que cada proceso posee al menos un recurso necesitado por el siguiente proceso de la lista.



Las 3 primeras condiciones, como dijimos, son necesarias pero no suficientes para la existencia de interbloqueo. La cuarta condición es una potencial consecuencia de las otras tres. Es decir, si se cumplen las 3 primeras condiciones, se puede producir una secuencia de eventos que conduzcan a una espera circular irresoluble.

Esto último es lo que se considera un interbloqueo propiamente dicho: una espera circular irresoluble.

La cuarta condición, la espera circular es irresoluble debido al cumplimiento de las tres primeras condiciones, en resumen, las cuatro condiciones de forma conjunta constituyen las condiciones necesarias y suficientes para la existencia del interbloqueo.

Cabe destacar que las tres primeras condiciones son cuestiones de diseño, mientras que la cuarta condición es una circunstancia que podría ocurrir dependiendo de la secuencia de peticiones y liberaciones realizadas por los procesos involucrados.

Prevención del interbloqueo

Existen métodos de prevención del interbloqueo que podemos clasificar en 2 categorías: los indirectos y los directos.

Un método indirecto trata de impedir la aparición de una de las 3 condiciones necesarias, mientras que un método directo impide que se produzca una espera circular.

Métodos indirectos

Exclusión mutua

En general no puede eliminarse. Si el acceso a un recurso requiere de exclusión mutua, el sistema operativo debe proporcionarlo. Algunos recursos, por ejemplo una base de datos o un archivo pueden tener un acceso compartido (múltiple) si las operaciones son sólo de lectura, pero debe ser exclusivo para las operaciones de escritura.

Retención y Espera

Esta condición puede eliminarse estableciendo que un proceso deba solicitar todos los recursos al mismo tiempo, bloquéandolo hasta que se le puedan asignar los mismos.

Esta solución tiene 2 serios problemas. El primero es que un proceso puede quedar bloqueado mucho tiempo hasta que se le asignen los recursos, cuando en realidad podría haber sido ejecutado sólo con algunos de los recursos. El segundo es que los recursos pueden permanecer ociosos un período de tiempo considerable, durante el cual podrían haber sido usados por otros procesos.

Otra alternativa para evitar la retención y espera, sería permitir a un proceso solicitar un recurso sólo cuando no tiene otro recurso asignado.

Por ejemplo, si un proceso requiere copiar información desde un DVD al disco, luego ordenar el archivo para finalmente imprimirlo en una impresora, según la primer alternativa deberá solicitar los 3 recursos (DVD, disco e impresora) al principio. Esto es un problema de diseño y se consigue tratando de centralizar lo antes posible las llamadas al sistema. Si se implementara la segunda alternativa, el proceso deberá solicitar el uso del DVD, copiar todos sus datos a memoria, luego liberar el DVD y solicitar el disco, copiar los datos al disco, liberar el disco para finalmente solicitar la impresora.

Sin expropiación (no preemption)

Esta condición se puede impedir de al menos 2 formas. La primera es: si a un proceso que ya tiene asignados N recursos, se le deniega la asignación del recurso N+1, deberá liberar los N recursos que ya le habían sido asignados.

La segunda alternativa sería que si un proceso P1 solicita un recurso que fue asignado a otro proceso P2 y P2 se encuentra bloqueado, el sistema operativo puede expropiar los recursos de P2 para asignarlos a P1.

Tener en cuenta que este segundo esquema evita el interbloqueado si y sólo si no existen 2 procesos con la misma prioridad compitiendo por el mismo recurso.

Esta alternativa sirve sólo cuando se aplica sobre recursos cuyo estado puede ser salvado y restaurado con facilidad (como el caso del procesador), no así con una impresora o disco.

Métodos directos

Espera circular

La espera circular se puede impedir definiendo un orden lineal o jerarquía entre los distintos tipos de recursos. Si un proceso solicita y le asignan un recurso R1, luego sólo podrá solicitar un recurso de tipo R2, asumiendo que los recursos forman parte del conjunto R = {R1, R2, ..., Rn}.

Evitar el interbloqueo

Las estrategias analizadas en la *prevención del interbloqueo*, tienden a establecer reglas o restricciones sobre el diseño del sistema operativo y el manejo de los procesos de forma tal que se pueda impedir al menos una de las cuatro condiciones que posibilitan el interbloqueo. La desventaja de estas alternativas es que conlleva a un uso ineficiente de los recursos y una ejecución ineficiente de los procesos.

La estrategia de *evitar el interbloqueo*, por su parte, decide dinámicamente si la petición actual de reserva de un recurso, en caso de que se conceda, puede potencialmente causar un interbloqueo.

Definición	Este tipo de estrategias, requiere conocer de antemano las futuras solicitudes de recursos de cada proceso.

Existen 2 técnicas dentro de la *predicción del interbloqueo*, la primera apunta a no ejecutar el proceso si los recursos que necesita podrían conducir a un interbloqueo y la segunda directamente no asigna los recursos a un proceso si dicha asignación podría provocar un interbloqueo.

Denegación de inicio de un proceso

A partir de la existencia de N procesos y M recursos en el sistema, se definen los siguientes vectores y matrices:

R = {R1, R2, ..., Rm}	cantidad total de cada recurso en el sistema
D = {D1, D2, ..., Dm}	recursos disponibles en el sistema
$N = \begin{matrix} N_{11} & N_{12} & .... & N_{1m} \\ N_{21} & N_{22} & .... & N_{2m} \\ ... & ... & .... & ... \\ N_{n1} & N_{n2} & .... & N_{nm} \end{matrix}$	N_{ij} = necesidades del proceso i con respecto al recurso j

$A = A_{11}\ A_{12}\ ....\ A_{1m}$ $A_{21}\ A_{22}\ ....\ A_{2m}$ $...\ \ ...\ \ ....\ ...$ $A_{n1}\ A_{n2}\ ....\ A_{nm}$	A_{ij} = asignación actual al proceso i con respecto al recurso j

Cada fila de la matriz de necesidad indica los requisitos máximos de cada proceso con respecto a cada recurso.

Se cumplen las siguientes relaciones:

$Ri = Dj + \sum_{i=1}^{n} Aij,\ \forall j$ todos los recursos están o bien disponibles o bien asignados

$Nij \leq Rj,\ \forall\ i,\ j$ ningún proceso puede necesitar más recursos de los existentes en el sistema

$Aij \leq Nij,\ \forall\ i, j$ ningún proceso puede tener asignados más recursos que los necesitados.

A partir de estas relaciones se establece la política de denegación de inicio de un proceso. Un proceso sólo se inicia si se cumple que:

$$Rij \geq N(n+1)j + \sum_{i=1}^{n} Nij,\ \forall\ j$$

Esta relación especifica que sólo podrá iniciarse un proceso si se pueden satisfacer las necesidades máximas de todos los procesos actuales más las del nuevo proceso. El problema de esta política es que asume el peor caso, es decir, que todos los procesos van a solicitar sus necesidades máximas simultáneamente..

Denegación de asignación de un recurso (algoritmo del banquero)

Esta estrategia, también conocida como **algoritmo del banquero**, propone que el sistema puede encontrarse, a partir de la asignación de recursos realizada, en 2 estados posibles: seguro y no seguro.

El estado seguro es aquél en el que hay al menos una secuencia de asignación de recursos a los procesos que no implica un interbloqueo, o lo que es lo mismo decir que todos los procesos pueden ejecutarse por completo.

Un estado no seguro es justamente lo opuesto a un estado seguro.

La idea detŕas de el algoritmo del banquero es que se ejecute cada vez que un proceso realiza una solicitud de recursos. En ese momento el algoritmo determinará en caso de llevar a cabo la

asignación de los recursos solicitados, si el estado del sistema será seguro o no. En caso de que el resultado de la asignación sea un estado no seguro, se denegará la asignación de los recursos al proceso y el mismo se bloqueará hasta que los recursos estén disponibles.

En términos matemáticos, deberá cumplirse la siguiente relación:

$$Nij \ - \ Aij \ \leq \ Dj, \ \forall j$$

Es decir, para que el sistema trabaje en un estado seguro, los recursos necesitados por un proceso, menos los ya asignados deberán ser menores o iguales que los disponibles por el sistema.

Veamos el siguiente ejemplo:

Estado inicial:

	Necesidad			Asignación			Necesidad - Asignación		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	2	2	2
P2	6	1	3	6	1	2	0	0	1
P3	3	1	4	2	1	1	1	0	3
P4	4	2	2	0	0	2	4	2	0

El vector de R (recursos) es R = {9, 3, 6} y el vector D (disponibles) es D = {0, 1, 1}

Con la asignación actual y los recursos disponibles ¿Podemos afirmar que es un estado seguro?

Para responder a esa pregunta debemos averiguar si alguno de los procesos puede ejecutarse por completo con los recursos disponibles. ¿Por qué? Porque si alguno de los procesos puede ejecutarse con los recursos disponibles, una vez que finalice podrá liberar los recursos posibilitando que el resto se ejecute también.

Del estado inicial, vemos que al proceso P2 sólo le hace falta 1 unidad de R3 para poder ejecutarse (eso lo vemos en la 3er matriz). Por lo tanto el sistema operativo puede realizar dicha asignación para que P2 ejecute. Al finalizar la ejecución, P2 liberará los recursos solicitados y pasará a estado finalizado. El estado del sistema resultante será:

	Necesidad			Asignación			Necesidad - Asignación		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	2	2	2
P2	0	0	0	0	0	0	0	0	0
P3	3	1	4	2	1	1	1	0	3
P4	4	2	2	0	0	2	4	2	0

El vector de R (recursos) es R = {9, 3, 6} y el vector D (disponibles) es D = {6, 2, 3}

Note que como P2 finalizó, toda la fila de la matriz N y de la matriz A están en 0 (cero) y el vector D se incrementa.

Ahora tendríamos que repetir las mismas preguntas ¿Podemos afirmar que es un estado seguro? Al igual que antes, si se asignan los recursos al proceso P1, el mismo podrá ejecutar hasta finalizar, dejando el sistema en el siguiente estado:

	Necesidad			Asignación			Necesidad - Asignación		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	0	0	0
P2	0	0	0	0	0	0	0	0	0
P3	3	1	4	2	1	1	1	0	3
P4	4	2	2	0	0	2	4	2	0

El vector de R (recursos) es R = {9, 3, 6} y el vector D (disponibles) es D = {7, 2, 3}.

El proceso se repite con P3 y P4, determinando entonces que el *estado inicial* planteado era un *estado seguro*.

Ahora bien, suponga el siguiente escenario inicial:

	Necesidad			Asignación			Necesidad - Asignación		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	2	0	1	1	2	1
P2	6	1	3	5	1	1	1	0	2
P3	3	1	4	2	1	1	1	0	3
P4	4	2	2	0	0	2	4	2	0

El vector de R (recursos) es R = {9, 3, 6} y el vector D (disponibles) es D = {0, 1, 1}

Con la asignación actual y los recursos disponibles ¿Podemos afirmar que es un estado seguro?

En este caso está claro que no es un estado seguro ya que no hay ninguna asignación posible que permita satisfacer las necesidades de recursos de un proceso. En este caso todos los procesos requieren al menos 1 unidad de R1.

Detección y recuperación del interbloqueo

Las estrategias para prevenir o evitar un interbloqueo, son muy restrictivas en el sentido de que resuelven el problema limitando el acceso a los recursos o bloqueando procesos. La detección del interbloqueo por su parte es más permisiva ya que los recursos se conceden siempre que sea posible y periódicamente se verifica que no exista la condición de espera circular.

La comprobación de si existe interbloqueo se puede hacer cada vez que un proceso solicita un recurso, o con menor frecuencia aún. Realizar la comprobación por cada solicitud conlleva a una detección temprana del interbloqueo, por otra parte, la desventaja es que se consume un tiempo de procesador considerable.

En caso de que el algoritmo detecte un interbloqueo aplicará alguna de las estrategias de recuperación del interbloqueo que mencionaremos más adelante.

Al igual que en las estrategias anteriores, para posibilitar la detección del interbloqueo se dispone de un vector con la cantidad total de Recursos (R), uno con la cantidad de recursos

Disponibles (D), la matriz de Asignación (A) y una matriz de Solicitudes (S). El algoritmo que posibilita tal detección es el siguiente:

1. Se inicia un vector temporal T asignándole el vector de D.
2. En la matriz A se marcan todas las filas que se encuentren completamente en 0 (cero).
3. De las filas sin marcar en A, se busca el primer proceso cuyas *Solicitudes* de recursos se puedan satisfacer. Es decir, se busca la 1er fila de S que sea menor o igual que D. Si no se encuentran procesos que cumplan tal condición, el algoritmo finaliza.
4. Si se encuentra 1 fila que cumpla la condición del punto 3, se marca el proceso (en la matriz A) y se suma la fila correspondiente al proceso en la matriz A al vector T

Si al final del proceso existen procesos sin marcar, los mismos se encuentran en interbloqueo.

La lógica de este algoritmo es ver qué procesos pueden ejecutarse a partir de las peticiones y recursos disponibles. Esto se realiza en el paso 3, Si se detecta un proceso que puede ser ejecutado, se asume que ejecuta hasta finalizar y libera los recursos asignados. Esto es lo que se hace en el paso 4 al sumar los recursos asignados al vector de recursos disponibles.

Tener en cuenta que a diferencia del algoritmo del banquero, este algoritmo no garantiza la prevención del interbloqueo, sino que determina la existencia actual del interbloqueo.

Ejemplo:

Solicitud

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

Asignación

R1	R2	R3	R4	R5
1	0	1	1	0
1	1	0	0	0
0	0	0	1	0
0	0	0	0	0

El vector de R (recursos) es R = {2, 1, 1, 2, 1} y el vector D (disponibles) es D = {0, 0, 0, 0, 1}

El algoritmo realiza lo siguiente:

1. Establece T = {0, 0, 0, 0, 1}
2. Marca en la matriz A, a P4 porque no tiene recursos asignados
3. Recorre S buscando una fila que sea menor o igual a T. El proceso P3 satisface dicha condición.

4. Actualiza T, sumando el valor de la fila de A correspondiente al índice de la fila encontrada en S. En este caso D = {0, 0, 0, 0, 1} + {0, 0, 0, 1, 0} => D = {0, 0, 0, 1, 1}

El proceso termina porque ni P1 ni P2, es decir ni S[0] ni S[1] son menores que D. Por lo tanto ambos procesos se encuentran en interbloqueo.

Recuperación del interbloqueo

Una vez que se detectó un interbloqueo se pueden aplicar algunas estrategias para recuperar el sistema de dicha situación. En orden de complejidad creciente son:

1. Finalizar todos los procesos que se encuentren participando del interbloqueo
2. Retroceder todos los procesos en interbloqueo a un punto de control previo al interbloqueo. Esta estrategia implica que se implementen mecanismos de retroceso y rearranque. Esta técnica posee el riesgo de que se repita el interbloqueo
3. Finalizar sucesivamente todos los procesos involucrados en el interbloqueo hasta que el mismo desaparezca. Por cada proceso finalizado, se debe invocar al algoritmo de detección.
4. Expropiar los recursos a los procesos involucrados en el interbloqueo hasta que el mismo desaparezca.

Para los puntos 3 y 4 se deben definir políticas de selección que impliquen un costo mínimo. Algunas de estas políticas pueden ser:

a. menor cantidad de tiempo de procesador consumida.
b. menor cantidad de salida producida
c. mayor tiempo restante estimado
d. menor cantidad de recursos asignados
e. menor prioridad

Bibliografía utilizada

● William Stallings. Sistemas operativos. Pearson Education. S.A., Madrid, 2005. ISBN-84-205-4462-0.

● AndrewS.Tanenbaum. Modern Operating System. Pearson Education Inc., 2009. ISBN-Q-lB-filBMST-L

● Multilevel Feedback queue. Wikipedia, La enciclopedia libre, 2019 [consulta: 21 de marzo del 2019]. Disponible en https://en.wikipedia.org/wiki/Multilevel_feedback_queue