# T E M A 2

# El Interbloqueo

## Contenido

## 2.1. Introducción

La administración de los recursos es una de las principales tareas del sistema operativo. Muchos de los recursos únicamente pueden ser usados por un solo proceso a la vez, como es el caso de las impresoras, dispositivos de cinta, etc. Como consecuencia, todos los sistemas operativos tienen que ofrecer mecanismos que permitan a los procesos acceder de forma exclusiva a determinados recuros. Como consecuencia, cuando un proceso solicita recursos y éstos no están disponibles en ese momento, entra en un estado de espera.

En muchas aplicaciones, un proceso puede requir acceso exclusivo no sólo a un recurso, sino a varios. En sistemas operativos en los que únicamente se ejecuta un proceso a la vez, el proceso simplemente toma todos los recursos que necesita. Sin embargo, en un sistema con multiprogramación, puede suceder que procesos que están en espera de recursos ocupados por otros procesos no cambien nunca de estado, porque los recursos que solicitan están en poder de otros procesos también en espera. A **interbloqueo** (***deadlock***. También **abrazo mortal** o **bloqueo mutuo**).

El interbloqueo se puede definir como sigue:

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

## 2.2. Recursos

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, de cada uno de los cuales pueden existir varios ejemplares idénticos:

- **físicos**: ciclos de CPU, espacio en memoria, dispositivos de E/S (impresoras, unidades de cinta, etc.).
- **lógicos**: ficheros, tablas del sistema, semáforos.

Un proceso debe solicitar un recurso antes de usarlo y liberarlo al terminar su uso. Además, un proceso puede solicitar cuantos recursos necesite.

Existen dos clases de recursos: expropiables y no expropiables. Un recurso expropiable (*preemptable resource*) es aquél que se le puede quitar a un proceso sin que se produzca ningún tipo de malfuncionamiento. Ejemplos son la memoria y la CPU. Un recurso no expropiable (*non preemptable resource*), por el contrario, no puede ser quitado a un proceso sin provocar un fallo en el mismo. Un ejemplo es la impresora. Los interbloqueos se suelen producir, en parte, debido al uso de recursos no expropiables. Sin embargo, existen otros sucesos que pueden originar un interbloqueo, como pueden ser los mecanismos de comunicación entre procesos.

En el modo de operación normal, un proceso utiliza un recurso de acuerdo con la siguiente secuencia:

- **Solicitud**. Si la solicitud no puede atenderse de inmediato, entonces el proceso solicitante debe esperar hasta que pueda adquirir el recurso.
- **Utilización**. El proceso puede trabajar con el recurso.
- **Liberación**. El proceso libera el recurso.

La solicitud y liberación de recursos son llamadas al sistema. Algunos ejemplos son las parejas de llamadas Abrir/Cerrar archivo y Asignar/Liberar memoria. De igual forma, la utilización de recursos sólo puede conseguirse mediante llamadas al sistema, de modo que para cada utilización el sistema operativo comprueba que el proceso que usa el recurso lo había solicitado previamente y ya lo tiene 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 ya se encuentra asignado a otro, puede añadirse a la cola de procesos que esperan tal recurso.

## 2.3. Condiciones para que se produzca Interbloqueo

Un interbloqueo puede surgir si y solo si en un sistema se presentan simultáneamente las siguientes cuatro condiciones:

1. **Exclusión mutua**. Los recursos compartidos son adquiridos y utilizados de modo mútuamente exclusivo, es decir, por un proceso como máximo en cada momento.
2. **Retención y espera**. Cada proceso retiene los recursos que ya le han sido asignados mientras espera para adquirir el resto de recursos.
3. **No expropiación**. Los recursos no se pueden quitar a los procesos. Un recurso sólo puede ser liberado voluntariamente por el proceso que lo retiene.
4. **Espera circular**. Debe existir una cadena circular de dos o más procesos, cada uno de los cuales espera por uno o más recursos en poder del siguiente miembro de la cadena.

La condición de espera circular implica la condición de retención y espera, por lo que las cuatro condiciones no son completamente independientes.

## 2.4. Modelado del Interbloqueo

Los interbloqueos pueden describirse utilizando un grafo dirigido y bipartito $G(N,A)$ llamado **grafo de asignación de recursos**, que consta en un conjunto de $N$ nodos (vértices) y $E$ arcos.

El conjunto de nodos $N$ se divide en dos tipos:

- $P = \{P_1, P_2, \ldots, P_n\}$, conjunto formado por todos los procesos del sistema.
- $R = \{R1, R_2, \ldots, R_n\}$, conjunto formado por todos los tipos de recursos.

Los arcos también son de dos tipos:

- **Arco de solicitud**. Es un arco que parte de un proceso $P_i$ hacia un tipo de recurso $R_j$ y se representa por $P_i \rightarrow R_j$ (o $(P_i, R_j)$). Significa que el proceso $P_i$ solicitó una instancia del recurso $R_j$ y se encuentra esperándolo.
- **Arco de asignación**. Es un arco que sale de un tipo de recurso $R_j$ y se dirige a un proceso $P_i$ (representado por $R_j \rightarrow P_i$ o $(R_j, P_i)$). Significa que se ha asignado un ejemplar del tipo de recurso $R_j$ al proceso $P_i$.

Gráficamente se representa cada proceso con un círculo y cada tipo de recurso con un rectángulo. Si de algún tipo de recurso existe más de un ejemplar, se representa cada ejemplar con un punto dentro del rectángulo. Un arco de solicitud parte entonces de un círculo y apunta a un rectángulo; en cambio un arco de asignación parte desde un punto dentro del rectángulo y señala hacia un círculo.

El efecto de las operaciones sobre recursos (solicitud, asignación y liberación) en el grafo de

| | Efecto en el grafo |
|---|---|
| $P_i$ **solicita** un ejemplar del tipo $R_j$ | Se inserta un arco de solicitud $P_i \rightarrow R_j$ |
| Una instancia del recurso $R_j$ se **asigna** al proceso $P_i$ | El arco de solicitud $P_i \rightarrow R_j$ se transforma instantáneamente en arco de asignación $R_j \rightarrow P_i$ |
| El proceso $P_i$ **libera** el recurso $R_j$ | Se elimina el arco de asignación $R_j \rightarrow P_i$ |

Un ejemplo de grafo de asignación de recursos se muestra en la Figura 1.

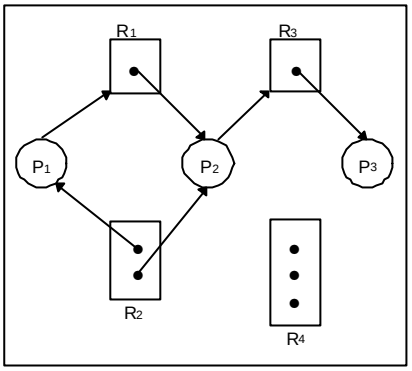


**Figura 1.-** Gráfico de asignación de recursos.

Este grafo de asignación de recursos muestra la siguiente situación:

- Los conjuntos P, R y E son:
  - $P = \{P_1, P_2, P_3\}$
  - $R = \{R_1, R_2, R_3, R_4\}$
  - $E = \{P_1 \to R_1, P_2 \to R_3, R_1 \to P_2, R_2 \to P_2, R_2 \to P_1, R_3 \to P_3\}$
- Los recursos existentes son:
  - Un recurso del tipo $R_1$.
  - Dos recursos del tipo $R_2$.
  - Un recurso del tipo $R_3$.
  - Tres recursos del tipo $R_4$.
- Estados de los procesos:
  - El proceso $P_1$ tiene asignado un recurso de tipo $R_2$, y espera un recurso de tipo $R_1$.
  - El proceso $P_2$ tiene asignado un recurso de tipo $R_1$ y otro de tipo $R_2$, y espera un recurso de tipo $R_3$.
  - El proceso $P_3$ tiene asignado un recurso de tipo $R_3$.

Si el grafo de asignación de recursos no contiene ciclos, entonce
encuentra en interbloqueo. Por otra parte, si existe un ciclo, puede haber interbloqueo. Es decir, la presencia de un ciclo es condición necesaria para la existencia de interbloqueo.

Si de cada tipo de recurso existe un único ejemplar, entonces la presencia de un ciclo determina que existe interbloqueo. En este caso la existencia de un ciclo es condición necesaria y suficiente para la existencia de interbloqueo.

En la Figura 2 se muestran un ejemplo de grafo de asignación con interbloqueo.

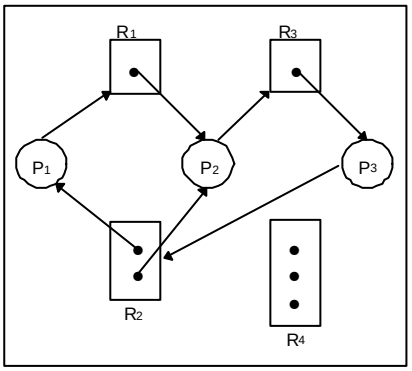


**Figura 2.-** Gráfico de asignación de recursos con un interbloqueo.

## 2.5. Métodos para el tratamiento del interbloqueo

Existen varias estrategias para enfrentar el problema de los interbloqueos:

- Ignorar el problema
- Detección y Recuperación
- Evitación
- Prevención

La estrategia más sencilla es simplemente ignorar el problema. A esta estrategia se la denomina **algoritmo del avestruz** (*Ostrich*): esconder la cabeza en la tierra y pretender que no existe problema alguno. La justificación de este método es que si el interbloqueo se presenta con una frecuencia baja en comparación con los fallos del sistema por otras razones (errores en el sistema operativo, fallos en el hardware, etc.), no tiene sentido tomar medidas para evitar el problema a costa de reducir el rendimiento del sistema. Un ejemplo es el sistema operativo Unix.

Cada tabla del sistema en Unix (tabla de procesos, tabla de nodos-i, etc.) es un recurso finito que puede ser fuente de un posible interbloqueo. El planteamiento de UNIX es ignorar el problema, bajo la hipótesis de que la mayoría de los usuarios preferiría un bloqueo ocasional en vez de una regla que restringiera el uso de los recursos. Esta decisión tiene su motivación en el hecho de que la eliminación del interbloqueo es muy costosa.

Una vez que se decide afrontar el problema, existen dos estrategias principales para tratar el interbloqueo:

1. Usar un protocolo que garantice que el sistema nunca entre en un estado de interbloqueo.
2. Permitir que el sistema entre en interbloqueo y luego se recupere.

En el primero de los casos existen dos métodos generales, que son conocidos como (*prevention*) y **evitación** (*avoidance*) de interbloqueos. El segundo requiere el empleo técnicas de *detection*) complementadas con técnicas de recuperación (*recovery*).

## 2.6. Prevención del interbloqueo

Para que ocurra un interbloqueo son necesarias cuatro condiciones. La filosofía básica de la prevención de interbloqueos es negar al menos una de ellas.

### 2.6.1. Exclusión mutua

Si ningún recurso se puede asignar de forma exclusiva, no se produciría interbloque. Sin embargo, existen recursos para los que no es posible negar la condición de exclusión mutua, pues la propia naturaleza de los mismos obliga a que sean utilizados en exclusión mutua.

No obstante, es posible eliminar esta condición en algunos recursos. Por ejemplo, una impresora es un recurso no compartible pues si se permite que dos procesos escriban en la impresora al mismo tiempo, la salida resultará caótica. Pero con el *spooling* de salida, varios procesos pueden generar salida al mismo tiempo. En este modelo, el único proceso que solicita en realidad la impresora físicamente es el demonio (*deamon*) de impresión (*spooler*). Puesto que el spooler nunca solicita otros recursos, se elimina el bloqueo originado por la impresora.

El inconveniente es que no todos los recursos pueden usarse de esta forma (por ejemplo, la tabla de procesos no se presta al *spooling* y, además, la implementación de esta técnica puede introducir nuevos motivos de interbloqueo, ya que el spooling emplea una zona de disco finita.

### 2.6.2. Retención y espera

La condición de retención y espera puede ser eliminada exigiendo o forzando a un proceso a que libere todos los recursos que retiene cada vez que solicite un recurso que no esté disponible. Es decir, los procesos que esperan no tienen recursos y los que tienen no esperan.

Hay básicamente dos implementaciones posibles de esta estrategia:

1. El proceso solicita todos los recursos necesarios antes de comenzar su ejecución. Un problema inmediato de este método es que se necesita pre-reclamar todas sus necesidades de recursos. Esto no siempre es posible y, en cualquier caso, requiere un trabajo adicional de estimación. Este método tiende a hacer una pobre utilización de los recursos que origina una reducción del nivel de concurrencia en el sistema.
2. El proceso solicita los recursos de forma incremental a lo largo de su ejecución, pero libera todos los recursos retenidos si se encuentra con una negativa. Es menos exigente que el método anterior, pero tiene el inconveniente de que algunos recursos no pueden ser liberados y readquiridos sin originar problemas. Por ejemplo, los cambios hechos sobre la memoria o sobre ficheros pueden corromper el sistema si no se llevan a término.

Un inconveniente adicional que se presenta en ambos métodos es que pueden conducir a la postergación indefinida (o inanición) de algunos procesos que solicitan recursos muy utilizados.

### 2.6.3. No expropiación

Esta condición puede ser negada, obviamente, permitiendo la expropiación, es decir, permitiendo que el sistema revoque la propiedad de ciertos recursos a los procesos bloqueados. Puesto que la expropiación es involuntaria desde el punto de vista del proceso afectado, el sistema operativo debe encargarse de salvar el estado y restaurarlo cuando el proceso sea posteriormente reanudado. Esto hace que la expropiación de recursos sea aún más difícil de llevar a cabo que la liberación voluntaria de recursos vista en el punto anterior. Sin embargo, este esquema se puede aplicar sobre recursos cuyo estado puede ser salvado y restaurado posteriormente (registros de CPU, memoria).

### 2.6.4. Espera circular

La condición de espera circular se puede romper mediante la ordenación lineal de los diferentes tipos de recursos del sistema. En este método, los recursos del sistema se dividen en clases diferentes $C_j$, donde $j = 1, 2, ..., n$. Los interbloqueos se previenen exigiendo que todos los procesos soliciten y adquieran sus recursos en un orden estrictamente creciente de las clases mencionadas. Además, la adquisición de todos los recursos pertenecientes a una misma clase debe efectuarse con una sola petición, y no incrementalmente.

Con esta estrategia los procesos no pueden quedar interbloqueados sea cual sea la forma en que se entrelacen, ya que no se pueden producir ciclos. Además, el respeto de los procesos a la ordenación prescrita puede ser comprobada en tiempo de compilación.

Una desventaja es que al adquirirse los recursos en el orden prescrito y no solicitarse cuando realmente se necesitan es que se no se aprovechan adecuadamente los recursos.

## 2.7. Evitación del interbloqueo

La idea básica de la evitación de interbloqueos es conceder únicamente las peticiones de recursos disponibles que no conduzcan a estados propensos al interbloqueo. En general, existe un asignador de recursos que, ante una petición determinada, examina el efecto de conceder dicha petición. Si la concesión puede conducir a un estado potencialmente peligroso, el proceso solicitante queda suspendido hasta el momento en que su petición pueda ser concedida sin problemas, lo cual suele suceder después de que uno o más de los recursos retenidos por otros procesos hayan sido liberados.

Un algoritmo de evitación examina dinámicamente el estado de asignación de recursos para asegurar que no pueda presentarse la condición de espera circular.

El estado de asignación de recursos viene definido por el número de recursos disponibles y asignados, y por la demanda máxima de los procesos.

Los principales algoritmos para evitar interbloqueos se basan en el concepto de **estado seguro**.

## 2.7.1. Trayectorias de recursos

En la Figura 3 se muestra un modelo para dos procesos ($P_1$ y $P_2$) y dos recursos (una impresora y un plotter). El eje horizontal representa el número de instrucciones ejecutadas por el proceso $P_1$ y el eje vertical el número de instrucciones ejecutadas por el proceso $P_2$. En el instante $I_1$, $P_1$ solicita la impresora, en $I_2$ necesita el plotter. La impresora y el plotter se liberan en los instantes $I_3$ e $I_4$, respectivamente. El proceso $P_2$ necesita el plotter en el intervalo de $I_5$ hasta $I_7$ y la impresora desde $I_6$ hasta $I_8$.

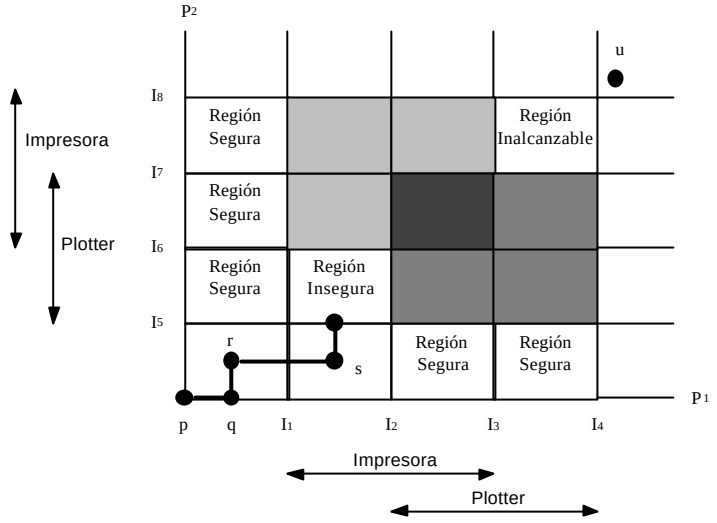


**Figura 3.-** Gráfico de asignación de recursos con un interbloqueo.

Una línea gruesa representa una posible evolución (trayectoria). Cada punto representa un estado conjunto de los dos procesos. Los tramos horizontales indican ejecución de $P_1$, los verticales, ejecución de $P_2$. Con un único procesador, todas las trayectorias serán horizontales o verticales. Además, el movimiento siempre será hacia arriba o hacia la derecha (los procesos no pueden retroceder).

## 2.7.2. Estados seguros e inseguros

Un estado es seguro si el sistema puede asignar recursos a cada proceso hasta alcanzar el máximo de su necesidades siguiendo algún orden arbitrario y aún así evitar el interbloqueo.

Más formalmente, un sistema se encuentra en estado seguro sólo si existe una secuencia segura. $<P_1, P_2, \ldots, P_n>$ es segura para el estado actual de asignación si, para cada proceso $P_i$, los recursos que aún puede solicitar $P_i$ pueden satisfacerse con los recursos actualmente disponibles más los retenidos por todos los $P_j$, tales que $j < i$. En esta situación, si los recursos que necesita $P_i$ no están inmediatamente disponibles, entonces $P_i$ puede esperar a que terminen todos los procesos que le preceden en la secuencia. Una vez terminados éstos, $P_i$ puede obtener todos los recursos necesarios, completar su tarea, devolver los recursos que se le han asignado y terminar. Cuando $P_i$ termina, $P_{i+1}$ puede obtener los recursos que necesita y así sucesivamente. Si no existe esta secuencia, se dice que el estado es inseguro.

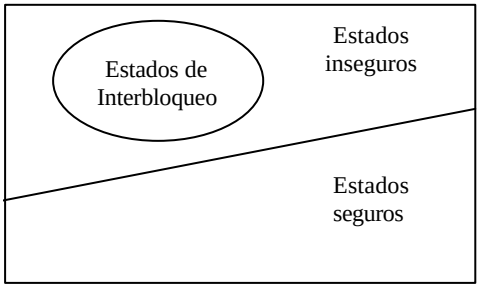


**Figura 4.-** Espacio de estados seguros, inseguros y de intebloqueo.

Un estado inseguro no es un estado de interbloqueo, pero un estado de interbloqueo sí es un estado inseguro. Es decir, no todos los estados inseguros lo son de interbloqueo. En la Figura 4 se puede observar que el conjunto de estados de interbloqueo es un subconjunto del de estados inseguros. Un estado inseguro puede conducir a un interbloqueo. Mientras el estado del sistema sea seguro, el sistema operativo puede evitar estados inseguros y, por tanto, de interbloqueo.

Si el estado es inseguro, el sistema operativo no puede hacer nada para evitar que los procesos soliciten recursos de tal manera que ocurra un interbloqueo. En otras palabras, el comportamiento de los procesos es el que controla los estados inseguros.

Los algoritmos de evitación se basan en garantizar que el sistema siempre permanecerá en estado seguro. Inicialmente, el sistema está en estado seguro. Cada vez que un proceso solicita un recurso disponible, el sistema debe decidir si se asigna o si el proceso debe esperar. La solicitud se atiende sólo si

**Problema**

Sean tres procesos $P_1$, $P_2$ y $P_3$ y un único tipo de recurso del que existe un total de 10 unidades. En el estado inicial $P_1$ tiene 3 instancias del recurso, pero podría necesitar en algún instante hasta 9; $P_2$ tiene 2 pero podría necesitar 4 más adelante y $P_3$ tiene 2 pero podría necesitar 7. En ese instante quedan libres 3 unidades.

1. Demostrar que el estado inicial es seguro.
2. Demostrar que si $P_1$ obtiene un ejemplar del recurso el estado resultante es inseguro.

## 2.7.3. Algoritmo del banquero

El algoritmo de evitación más famoso se debe a Dijkstra y se conoce con el nombre de **algoritmo del banquero**.

Cuando un nuevo proceso entra al sistema debe declarar el número máximo de ejemplares de cada tipo de recurso que pueda necesitar y que, naturalmente, no podrá exceder del total de recursos del sistema. Cuando un usuario solicita un conjunto de recursos, el sistema debe determinar si la asignación de éstos dejará al sistema en un estado seguro. Si es así, los recursos se asignan; de lo contrario, el proceso tendrá que esperar hasta que otro libere suficientes recursos.

Para implementar el algoritmo se necesitan varias estructuras de datos, que nos permitan representar el estado de asignación de recursos del sistema. Sea $n$ el número de procesos en el sistema y $m$ el número de tipos de recursos. Las estructuras de datos necesarias son:

- *Disponible*: Vector de longitud $m$ que indica el número de recursos de cada tipo que hay disponible.
- *Max*: Matriz de $n \times m$ que define la demanda máxima de cada proceso para cada tipo de recurso.
- *Asignación*: Matriz $n \times m$ que define el número de recursos de cada tipo asignados a cada proceso en ese instante.
- *Necesidad*: Matriz $n \times m$ que indica los recursos que le pueden hacer falta a cada proceso. *Necesidad[i, j] = Max[i, j] - Asignación[i, j]*.

Para simplificar, usaremos la siguiente notación:

- Dados $X$ e $Y$ dos vectores de longitud $n$, se dice que $X \leq Y$ si y sólo si $X[i] \leq Y[i]$ para todo $i = 1, 2, ..., n$. $X < Y$ si $X \leq Y$ y $X \neq Y$.
- Se puede tratar cada fila de las matrices *Necesidad* y *Asignación* como vectores y referirse a ellas como $Necesidad_i$ y $Asignación_i$ que representan, respectivamente, los recursos adicionales que puede solicitar aún el proceso $P_i$ para concluir su tarea y los recursos asignados actualmente al proceso $P_i$.

### 2.7.3.1. Algoritmo del banquero

Sea $Solicitud_i$ el vector de solicitudes para el proceso $P_i$. Cuando el proceso $P_i$ efectúa una solicitud de recursos, el asignador de recursos toma las siguientes acciones:

1. Si $Solicitud_i \leq Necesidad_i$, continuar con el paso 2. De lo contrario, devolver una condición de error, puesto que el proceso se ha excedido de su demanda máxima.
2. Si $Solicitud_i \leq Disponible$, continuar en el paso 3. De los contrario, $P_i$ deberá esperar, pues los recursos no están disponibles.

3. El sistema calcula el nuevo estado de asignación al que llevaría la concesión de los recursos solicitados por $P_i$. Es decir,

   $Disponible := Disponible - Solicitud_i$

   $Asignación_i := Asignación_i + Solicitud_i$

   $Necesidad_i := Necesidad_i - Solicitud_i$

4. Analizar la seguridad del nuevo estado (algoritmo de seguridad). Si el estado resultante es seguro, entonces se confirma la transacción y los recursos son definitivamente asignados a $P_i$. Si el nuevo estado no es seguro, entonces se restablece el estado anterior de asignación de recursos y $P_i$ deberá esperar para obtener su solicitud de recursos.

Cuando un recurso es liberado, el asignador de recursos actualiza la estructura de datos *Disponible* y reconsidera las peticiones pendientes, si es que las hay, de ese tipo de recurso.

### 2.7.3.2. Algoritmo de seguridad

El algoritmo para averiguar si un sistema está o no en un estado seguro puede describirse de la siguiente forma:

1. Sean *Trabajo* y *Fin* dos vectores auxiliares de longitud $m$ y $n$, respectivamente. Se inicializan:
   $Trabajo := Disponible$
   $Fin[i] := falso$, para todo $i=1, 2, ..., n$
2. Encontrar un $i$ tal que se cumplan las dos proposiciones siguientes:
   a) $Fin[i] = falso$
   b) $Necesidad_i \pounds Trabajo$
   Si no existe tal $i$, continuar con el paso 4.
3. Hacer
   $Trabajo := Trabajo + Asignación_i$
   $Fin[i] := verdadero$
   Continuar en el paso 2.
4. Si $Fin[i] = verdadero$ para todo $i$, entonces el sistema está en estado seguro. Si no, el estado es inseguro.

**Problema**

Sea un sistema con cinco procesos, $P_1$ a $P_5$, y tres tipos de recursos, A, B y C de los que existen diez ejemplares, cinco y siete, respectivamente. Supongamos que en el instante actual, tenemos la siguiente situación del sistema:

| | A | B | C |
|---|---|---|---|
| $P_1$ | 0 | 1 | 0 |
| $P_2$ | 2 | 0 | 0 |
| $P_3$ | 3 | 0 | 2 |
| $P_4$ | 2 | 1 | 1 |
| $P_5$ | 0 | 0 | 2 |

Max

| A | B | C |
|---|---|---|
| 7 | 5 | 3 |
| 3 | 2 | 2 |
| 9 | 0 | 2 |
| 2 | 2 | 2 |
| 4 | 3 | 3 |

Disponible

| A | B | C |
|---|---|---|
| 3 | 3 | 2 |

Necesidad

| A | B | C |
|---|---|---|
| 7 | 4 | 3 |
| 1 | 2 | 2 |
| 6 | 0 | 0 |
| 0 | 1 | 1 |
| 4 | 3 | 1 |

1. Determinar si este estado es seguro o inseguro.
2. Supongamos ahora que $P_2$ hace una petición $Solicitud_2 = (1, 0, 2)$. ¿Cómo actuaría el algoritmo del banquero para decidir si esta petición puede atenderse?
3. Determinar si las siguientes solicitudes pueden atenderse o no:
   - Solicitud de (3, 3, 1) por parte de $P_5$.
   - Solicitud de (0, 2, 0) por parte de $P_1$.

### 2.7.4. Resumen

La evitación de interbloqueos no requiere la adquisición de todos los recursos de una vez, ni tampoco necesita la expropiación de recursos. Estos se solicitan cuando y como se necesitan. Así, se elimina el problema de la infrautilización de los recursos debido a la adquisición prematura, que aparece con

Como aspecto negativo, la evitación requiere que se haga la pre-reclamación de las necesidades máximas de recursos e impone costes de almacenamiento y computación en tiempo de ejecución para detectar los estados seguros del sistema. Si la pre-reclamación de recursos se realiza de forma conservadora se limita el grado de concurrencia al hacer que el asignador detecte como inseguros un mayor número de estados del sistema.

Debido a que los estados inseguros constituyen un conjunto grande de estados dentro del cual los estados de interbloqueo son sólo un subconjunto, la estrategia de evitación puede posponer innecesariamente la asignación de recursos disponibles a procesos que lo solicitan.

El algoritmo del banquero fue publicado por Dijkstra por primera vez en 1965. Aunque en teoría funciona perfectamente en la práctica no es útil, puesto que los procesos rara vez conocen de antemano sus necesidades máximas de recursos. Además, el número de procesos no es fijo, como precisa el algoritmo, sino que varía dinámicamente al conectarse y desconectarse los usuarios.

## 2.8. Detección del interbloqueo

En lugar de sacrificar el rendimiento previniendo o evitando interbloqueos, algunos sistemas conceden libremente los recursos disponibles a los procesos solicitantes y, ocasionalmente, comprueban el sistema para determinar si se ha producido un interbloqueo e intentar romperlo, si es que lo hay.

Un método práctico es intentar reducir el grafo de asignación de recursos suprimiendo todas las retenciones y peticiones (arcos de asignación y de solicitud) de cada proceso cuyas solicitudes puedan ser concedidas, hasta que se efectúen todas las reducciones posibles. Si el grafo queda completamente reducido después de esta operación, es decir, no quedan arcos, el sistema no está interbloqueado. En caso contrario, el sistema está interbloqueado, y los procesos que quedan son los afectados. La capacidad que tiene el algoritmo para identificar los procesos interbloqueados es también importante, pues facilita la

### 2.8.1. Algoritmo de detección

El algoritmo utiliza la matriz *Asignación* y el vector *Disponible* tal como se hizo en el esquema de evitación.

En vez de la matriz *Necesidad* se utiliza la matriz *Solicitud* del mismo formato para identificar el número de recursos de cada tipo solicitados pero aún no concedidos a los procesos activos. La suma de recursos solicitados y asignados no puede exceder la capacidad máxima del sistema.

El algoritmo funciona contabilizando todas las posibilidades de secuenciar los procesos que quedan por completar.

Si existe una secuencia de terminación, el sistema no está en estado de interbloqueo. En caso contrario, el sistema esta interbloqueado, ya que no hay forma de que todos los procesos activos terminen.

En los métodos de detección, el asignador de recursos simplemente con
recursos disponibles Cuando es invocado para determinar si se ha producido interbloqueo, funciona de la siguiente forma:

1. Sean *Trabajo* y *Fin* dos vectores de trabajo de longitud *m* y *n*, respectivamente. Se inicializan:
   *Trabajo := Disponible*
   *Fin[i] := falso*, para todo *i=1, 2, ..., n* si $Asignación_i \neq 0$
   *Fin[i] = verdadero*, en caso contrario

2. Encontrar un valor de índice, *i*, tal que se cumplan las dos proposiciones siguientes:
   a) *Fin[i] = falso*
   b) $Solicitud_i \leq Trabajo$

   Si no existe tal *i*, continuar con el paso 4.
3. Hacer
   $Trabajo := Trabajo + Asignación_i$

   *Fin[i] := verdadero*

   Continuar en el paso 2.
4. Si *Fin[i] = verdadero* para todo *i*, entonces el sistema no está en estado interbloqueado. Si no, el estado es de interbloqueo. Además, si *Fin[i] = falso*, entonces el proceso *Pi* está implicado en el interbloqueo detectado.

**Problema**

Sean cinco procesos, $P_1$ a $P_5$, y tres tipos de recursos, A, B y C con siete, dos y seis unidades respectivamente. En el instante considerado, tenemos el siguiente estado de asignación de recursos:

| | A | B | C |
|---|---|---|---|
| P1 | 0 | 1 | 0 |
| P2 | 2 | 0 | 0 |
| P3 | 3 | 0 | 3 |
| P4 | 2 | 1 | 1 |
| P5 | 0 | 0 | 2 |

Solicitud

| A | B | C |
|---|---|---|
| 0 | 0 | 0 |
| 2 | 0 | 2 |
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 2 |

Disponible

| A | B | C |
|---|---|---|
| 0 | 0 | 0 |

1. Determinar si este estado es o no de interbloqueo.
2. Si que el proceso $P_3$ solicita una unidad adicional de C, determinar si el estado resultante es de interbloqueo.

### 2.8.2. Utilización de los algoritmos de detección

Un aspecto importante es la frecuencia de detección de interbloqueos, que suele ser un parámetro del sistema. Una posibilidad extrema es comprobar el estado cada vez que se solicita un recurso y éste no puede ser asignado. El inconveniente es el tiempo de CPU que se utiliza. Otra alternativa es activar el algoritmo ocasionalmente, a intervalos regulares o cuando uno o más procesos queden bloqueados durante un tiempo sospechosamente largo.

Una frecuencia alta de comprobación tiene la ventaja de descubrir con prontitud los interbloqueos, capacitando al sistema a actuar rápidamente. Con una frecuencia baja, se reduce el gasto de tiempo de ejecución para la detección, a expensas de dejar interbloqueos sin detectar durante más largos períodos de tiempo. Esta estrategia puede redundar en un empobrecimiento en la utilización de recursos, puesto que los procesos interbloqueados continúan reteniendo los recursos ya concedidos sin realizar trabajo alguno con ellos.

## 2.9. Recuperación del interbloqueo

Cuando un algoritmo de detección determina que existe un interbloqueo, existen varias alternativas. La más simple es informar al operador del sistema para que éste se ocupe de él manualmente. La otra posibilidad es hacer que el sistema rompa el interbloqueo y se recupere automáticamente, para lo cual existen dos opciones: una consiste en abortar uno o más procesos para romper la espera circular y la segunda es expropiar algunos recursos de uno o más de los procesos implicados.

### 2.9.1. Terminación de procesos

Existen dos métodos, en los cuales el sistema recupera todos los recursos de los procesos terminados:

- **Abortar todos los procesos interbloqueados**. El ciclo se rompe, pero a un coste muy elevado, ya que posiblemente estos procesos efectuaron operaciones durante mucho tiempo.
- **Abortar procesos de uno en uno hasta eliminar el ciclo**. Puede suponer mucho tiempo de proceso adicional puesto que, después de abortar cada proceso, hay que invocar al algoritmo de detección

En cualquiera de los casos, puede que no sea fácil terminar un proceso. Si éste se encuentra actualizando un fichero, abortarlo puede ocasionar que el fichero quede en un estado incorrecto. Cuando se utiliza el método incremental, se presenta un nuevo problema de "política", que es la selección de la víctima. Desde un punto de vista económico se debe seleccionar aquel proceso cuya terminación represente el costo mínimo. Sin embargo, el término costo mínimo no es muy preciso.

Muchos factores determinan el proceso a eliminar, incluyendo:

1. La prioridad del proceso
2. Tiempo de ejecución ya consumido y tiempo de ejecución previsto
3. Cuántos recursos y de qué tipo está usando el proceso
4. Cuántos recursos más necesita el proceso para concluir
5. Cuántos procesos es preciso terminar
6. Tipo del proceso (por lotes o interactivo)

### 2.9.2. Expropiación de recursos

Consiste en quitar sucesivamente los recursos de los procesos que los tienen y asignarlos a otros que los solicitan hasta conseguir romper el interbloqueo. Hay que considerar tres aspectos:

1. **Selección de la víctima**. ¿A qué procesos y qué recursos se expropiarán? Igual que antes, se debe determinar el orden de expropiación para minimizar el costo.
2. **Retroceso** (*rollback*). ¿Qué pasa con el proceso al que se expropia un recurso? Obviamente, no puede continuar con su ejecución normal, pues le falta un recurso necesario. Se debe hacer retroceder al proceso hasta llegar a un estado anterior seguro, para más adelante reiniciarlo a partir ese estado.
3. **Postergación indefinida**. En un sistema donde la selección de la víctima se basa en factores de costo, puede suceder que siempre se elija como víctima al mismo proceso, lo cual puede conducir a una situación de bloqueo efectivo del mismo (inanición). Se debe asegurar que un mismo proceso sólo es elegido un número finito (y pequeño) de veces. La solución habitual es incluir el número de retrocesos como factor de costo.

### 2.9.3. Resumen

La detección y recuperación de interbloqueos proporciona un mayor grado potencial de concurrencia que las técnicas de prevención o de evitación. Además, la sobrecarga en tiempo de ejecución de la detección

El precio a pagar es la sobrecarga debida a la recuperación una vez que se han detectado los interbloqueos y también una reducción en el aprovechamiento de los recursos del sistema debido a aquellos procesos que son reiniciados o vueltos hacia atrás.

La recuperación de interbloqueos puede ser atractiva en sistemas con una baja probabilidad de interbloqueos. En cambio, en sistemas con elevada carga, se sabe que la concesión sin restricciones de peticiones de recursos puede conducir a frecuentes interbloqueos.

## 2.10. Estrategia combinada para el manejo de interbloqueos

Ninguno de los métodos presentados es adecuado para ser utilizado como estrategia exclusiva de manejo de interbloqueos en un sistema complejo. En vez de ello, la prevención, evitación y detección pueden combinarse para obtener una máxima efectividad. Esto puede conseguirse dividiendo los recursos del sistema en una colección de clases disjuntas y aplicando el método más adecuado de manejo de interbloqueo a los recursos de cada clase particular. La división puede efectuarse según la jerarquía de diseño de un sistema operativo dado, o de acuerdo con las características dominantes de ciertos tipos de recursos, tales como tolerar expropiaciones o permitir predicciones precisas.

La ordenación de recursos, descrita en el apartado de prevención, puede servir para prevenir interbloqueos entre clases. Puede emplearse entonces el tratamiento particular de cada clase.

Considérese un sistema que posee las siguientes clases de recursos, ordenados en la siguiente forma:

1. **Recursos internos**: recursos del sistema, tales como la tabla de procesos.
2. **Memoria principal**: memoria usada por los procesos de usuario.
3. **Recursos usados por los procesos**: dispositivos asignables, tales como impresoras o unidades con soporte extraible (cintas, discos, floppies) y ficheros.
4. **Espacio de intercambio**, espacio para cada proceso de usuario en almacenamiento secundario.

La idea básica es proporcionar prevención de interbloqueo entre las cuatro clases de recursos mediante la ordenación lineal de las peticiones en el orden presentado. Sin embargo, dentro de cada clase se selecciona como estrategia local la más adecuada para las características específicas de los recursos que la componen.

Pueden aplicarse las siguientes estrategias:

1. **Recursos internos del sistema**. Debido a las frecuentes peticiones y liberaciones de recursos a este nivel y a los frecuentes cambios de estado resultantes, el recargo en tiempo de ejecución de la evitación e incluso de la detección pueden difílcimente ser tolerados. La prevención mediante ordenación de recursos es probablemente la mejor alternativa.
2. **Memoria principal**. La existencia de almacenamiento de intercambio hace que la prevención mediante expropiación sea una elección razonable. La evitación no es deseable debido a su recargo en tiempo de ejecución y a su tendencia a infrautilizar los recursos. La detección es posible, pero no deseable debido al recargo en tiempo de ejecución en caso de detección frecuente o a la memoria no utilizada retenida por los procesos interbloqueados.
3. **Recursos usados los procesos**. La evitación se ve facilitada por la pre-reclamación de las necesidades de recursos que habitualmente se efectúa mediante instrucciones de control. La prevención también es posible mediante la ordenación de recursos, pero la estrategia de detección y recuperación no es deseable por la posibilidad de modificación de ficheros que pertenecen a esta clase de recursos.
4. **Espacio de intercambio**. Una posibilidad es la adquisición anticipada de todo el espacio necesario (prevención), ya que se conocen los requisitos previos de almacenamiento.

## 2.11. Referencias


- “Modern Operating Systems”. A.S. Tanenbaum. Prentice-Hall. 1992.
- "Operating System Concepts". A. Silberschatz, P. B. Galvin. Addison-Wesley. 1994.
- "Sistemas Operativos, Segunda Edición". Deitel, H. M., Addison-Wesley, 1992.

I.