Sistemas Operativos

Deadlocks – II
(Interbloqueos)

Sistemas Operativos

- ☑ Versión: Mayo 2018
- ☑ Palabras Claves: Deadlock, Bloqueo, Procesos, Recursos, Inanición

Algunas diapositivas han sido extraídas de las ofrecidas para docentes desde el libro de Stallings (Sistemas Operativos) , el de Silberschatz (Operating Systems Concepts)

4 Métodos para el tratamiento del Deadlock

- **Asegurar que el Sistema NUNCA se entrará en Deadlock**
  1. **Prevenir que ocurra (atacar las 4 condiciones, clase pasada)**
  2. **Evitar que ocurra**
- **Entrará en estado de Deadlock**

3. **Permitir y Recuperarse luego que ocurra**

- **No pasa nada**

4. **Ignorar el problema, esperar que nunca ocurra (los SO actuales incluido UNIX). Algoritmo del Avestruz.**

Método 2 Evitar bloqueos mutuos

- Los **bloqueos mutuos** pueden ser **evitados** si se **sabe** cierta **información** sobre los **procesos antes** de la **asignación** de **recursos**.
- Para **cada petición** de **recursos**, el **sistema controla** si **satisfaciendo** el **pedido** entra en un **estado inseguro**, **donde puede producirse** un **bloqueo mutuo**.
- El **sistema satisface** los **pedidos** de **recursos solamente si** se **asegura** que quedará en un **estado seguro**.
- Para que el **sistema** sea **capaz** de **decidir** si el siguiente **estado será seguro** o **inseguro**, debe **saber** por **adelantado** y en **cualquier momento** el **número** y **tipo** de **todos** los **recursos** en **existencia**, **disponibles** y **requeridos**.

Existen **varios algoritmos** para **evitar bloqueos mutuos**:

- **Algoritmo del banquero**, introducido por Dijkstra.
- **Algoritmo de grafo de asignación de recursos.**
- **Algoritmo de Seguridad.**
- **Algoritmo de solicitud de recursos.**

Método 2: Algoritmos para EVITAR el deadlock

1. **Instancia única** de **un tipo** de **recurso**
   - Utilizar **Grafo** de **Asignación** de **Recursos.**
   - **Algoritmo** que **determina** el **estado seguro** de un sistema.
   - **Encuentro secuencia segura**

2. **Múltiples instancias** de **un tipo** de **recurso**
   - Utilice el **Algoritmo del Banquero**
   - **Algoritmo teórico**

Evitar: Algoritmo del Banquero

- Dijkstra ´65: una **cartera** de **clientes** con una línea de **créditos máxima** por **cliente**.
- A **cada pedido** de **dinero** el **banquero** tiene que **decidir si** le **otorga** o **no teniendo** en **cuenta** que si de **golpe** todos **piden** el **máximo**,
- **existe** algún **orden** en el cual **puede satisfacerse** (teniendo en cuenta las **reservas** del banco)

dinero->?, clientes-> ?,

máximo crédito-> ?, reservas->?

Evitar: Algoritmo del Banquero

- Se aplica para sistemas con **múltiples instancias** de **cada recurso**.
- Los procesos **declaran** el número **máximo** de **instancias** de **cada recurso** que **necesitarán**
- Ese **número no puede exceder total** de **instancias** de **recursos** de ese **tipo** en el **sistema**.
- El **SO decidirá** en qué momento **asignarlos**, **garantizando** un **estado seguro**.

Evitar: Algoritmo del Banquero (cont)

- **Cuando** un **proceso solicita** un **recurso puede tener** que **esperar**
- **Cuando** un **proceso obtiene TODOS** sus **recursos**, **debe devolverlos** en una **cantidad** de **tiempo finita**
- Es **difícil** para el **SO saber todo** esto **antes**. Debería realizar simulaciones

Estructuras asociadas del Banquero

- ***n*: cantidad de procesos**
- ***m*: cantidad de tipos de recursos**
- **k: instancias del recurso**
- ***Disponible/available*: VECTOR de *m* componentes, con la cantidad de recursos disponibles para cada tipo, tal que si disponible(j)=*k*, indica que hay *k* instancias del recurso Rj . (el TOTAL de recursos disponibles)**
- ***Max*: MATRIZ de *n* x *m*. Max(i,j)=*k*, indica que Pi necesitará en total *k* instancias del recurso Rj (lo max que puede pedir SOLICITUDES)**
- ***Asignación/allocation*: MATRIZ de *n* x *m*. Asignacion(i,j)=*k*, indica que hay *k* instancias del recurso Rj asignadas a Pi (Lo que me dieron ASIGNADO)**
- ***Need/Necesidad*: MATRIZ de *n* x *m*. Need(i,j)=*k*, indica que Pi necesitará *k* instancias mas de las que ya tiene, del recurso Rj (lo que me falta NECESIDADES)**

Need [i,j] = Max[i,j] – Allocation [i,j]

Tener en cuenta:

- Si **X** e **Y** son **vectores** de n componentes,
  - decimos que **X≤Y** si y solo si **X(i)≤Y(i)**, para **todo i**=1,..,n.
  - decimos que **X<Y** si y solo **si** **X(i)≤Y(i)**, y **X≠Y** para **todo i**=1,..,n.
- Este **algoritmo** toma las **filas** de las **matrices** como **vectores**.
- **Recursos asignados** a **Pi** representados por el vector $Asig_i$ que es la **fila i** de la **matriz Asig(i,x)**.

Algoritmo del Banquero

Algoritmo **requerir un recurso** :

1.Si **Requerimiento(i) ≤ Necesidad(i**) **seguir**, **sino ERROR**

•pide más que lo declarado en MAX(i)

2.Si **Requerimiento(i) ≤ Disponible** **seguir**, **sino debe esperar,**

• **pues el recurso no está disponible**

3.Luego el sistema pretende **adjudicar** los **recursos** a **p(i**), **modificando** los **estados** de la siguiente forma:

•**Disponible = Disponible - Requerimiento(i)**

•**Asignación(i) = Asignación(i) + Requerimiento(i)**

•**Necesidad(i) = Necesidad(i) - Requerimiento(i)**

4.Luego aplicamos el algoritmo de estado seguro. Si esta nueva situación mantiene al sistema en **estado seguro**, **los recursos son adjudicados.** Si el nuevo **estado es inseguro, p(i) debe esperar** y, además, **se restaura el estado anterior de asignación total 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

Algoritmo del Estado Seguro (de Shoshani y Coffman)

1. Sean **Trabajo** y ***Fin*** dos **vectores auxiliares** de longitud ***m*** y ***n***, respectivamente. Se inicializan:
   - ***Trabajo := Disponible*** **(Trabajo() acumula los recursos libres que quedan)**
- ***Fin[i] := falso***, para **todo** ***i=1, 2, ..., n*** **(Fin() para procesos ya controlados)**

2. **Encontrar un *i*** tal que se **cumplan** las **2 proposiciones** siguientes:
- **a) *Fin[i] = falso***
- **b) *Necesidadi ≤ Trabajo***

Si no existe tal *i*, continuar con el paso 4.

3. Hacer
- ***Trabajo := Trabajo + Asignacióni***
- ***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**.

Cabe aclarar que no se están ejecutando los procesos: se está analizando si existe una secuencia segura de ejecución.

Ejemplo:

Existe la siguiente matriz en donde A, B y C son 3 tipos diferentes de recursos y Asignados, Requeridos y Disponibles las estructuras de control

Asignados			*Requeridos*			*Disponibles*		
A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
0	1	0	0	0	0	0	0	0
2	0	0	2	0	2			
3	0	3	0	0	0			
2	1	1	1	0	0			
0	0	2	0	0	0			

Para iniciar con el proceso primero verificamos so *Requeridos <= Disponibles*, vemos que para los recursos de tipo A son iguales, tambien los de tipo B y los de tipo C entre las dos estructuras, por tanto si cumple entonces se marca en rojo la fila de evaluada para denotar que ya ha terminado el proceso y luego se suman los *Requeridos* de esa fila con los *Disponibles* de esa fila y el resultado se pone en la siguiente fila de *Disponibles* quedando de la siguiente forma:

Asignados			*Requeridos*			*Disponibles*		
A	B	C	A	B	C	A	B	C
0	1	0	0	0	0	0	0	0
2	0	0	2	0	2	0	1	0
3	0	3	0	0	0			
2	1	1	1	0	0			
0	0	2	0	0	0			

Nota: Recuerde llevar el orden en que los procesos se van terminando con éxito

Ahora se repite el proceso con la siguiente fila, se evalua si *Requeridos <= Disponibles* pero como no es asi entonces pasamos a la siguiente fila y repetimos el proceso y nos quedaria de la siguiente manera

Asignados			*Requeridos*			*Disponibles*		
A	B	C	A	B	C	A	B	C
0	1	0	0	0	0	0	0	0
2	0	0	2	0	2	0	1	0
3	0	3	0	0	0	3	1	3
2	1	1	1	0	0			
0	0	2	0	0	0			

Asi lo hacemos hasta terminar con el ultimo proceso y nos queda lo sigueinte:

Asignados			*Requeridos*			*Disponibles*		
A	B	C	A	B	C	A	B	C
0	1	0	0	0	0	0	0	0
2	0	0	2	0	2	0	1	0
3	0	3	0	0	0	3	1	3
2	1	1	1	0	0	5	2	4
0	0	2	0	0	0	5	2	6

Ahora vamos a repetir el proceso con los que no pudimos terminar desde el principio tomando en cuenta el ultimo registro de la estructura de *Disponibles*

y hacemos la comparacion *Requeridos* <= *Disponibles* (2 0 2) <= (5 2 6) y finalmente podemos terminar el proceso pendiente quedandonos la tabla de la siguiente manera:

Asignados			*Requeridos*			*Disponibles*		
A	B	C	A	B	C	A	B	C
0	1	0	0	0	0	0	0	0
2	0	0	2	0	2	0	1	0
3	0	3	0	0	0	3	1	3
2	1	1	1	0	0	5	2	4
0	0	2	0	0	0	5	2	6

Notese que al hacer el ultimo cal**** quedo en la lista de disponibles la siguiente convinacion de recursos:
(7 2 6)

Finalmente el orden en que los procesos se terminaron es el siguiente:

1, P_3, P_4, P_5, P_2>

Ejercicio

La siguiente tabla representa un sistema en estado seguro. Si el proceso P3 solicita (1, 0, 0, 0) es posible asignárselo sin alterar el estado del sistema

Proceso	Asignado				Máximo				Disponible			
P1	1	0	0	1	2	0	2	1	2	1	1	0
P2	0	0	1	1	0	1	2	3				
P3	0	0	2	0	2	0	2	1				
P4	0	1	0	1	1	1	0	1				
P5	2	0	1	1	3	2	3	1				

Algoritmo del Banquero

- **no** se **usa** en la práctica
- **difícil** de **establecer Requerimientos** a priori
- los **procesos varían dinámicamente** (se **crean** y se **terminan**) lo que **complica** el **cálculo** de **estado** seguro
- **dinámicamente no sirve**, qué hacemos?

Por ejemplo: Prevenir estáticamente atacando los 4 criterios de Deadlock

3° Método: Detección y Recuperación

Si no puedo asegurar que el DL no ocurrirá necesito usar este esquema (el NUNCA)

- ☑ **Permitir** que el sistema **entre** en **estado** de **deadlock**. (Ceder libremente los recursos).
- ☑ **Mantener información de recursos asignados y pedidos**
- ☑ **Algoritmo** que examine si **ocurrió** un **deadlock** (lo **detecte)**
- ☑ **Algoritmo** para **recuperación** del **deadlock**. (Intentar romper alguna condición)

Detección

- Con **recursos** con **1** sóla **instancia**
  - **análisis del grafo de asignación**
- Con **recursos** de **varias instancias**
  - **Algoritmo de Shoshani y Coffman**
  - **Algoritmo del Banquero**

Instancia única de cada tipo de recurso

Mantener un **grafo wait-for (Grafo de espera)**

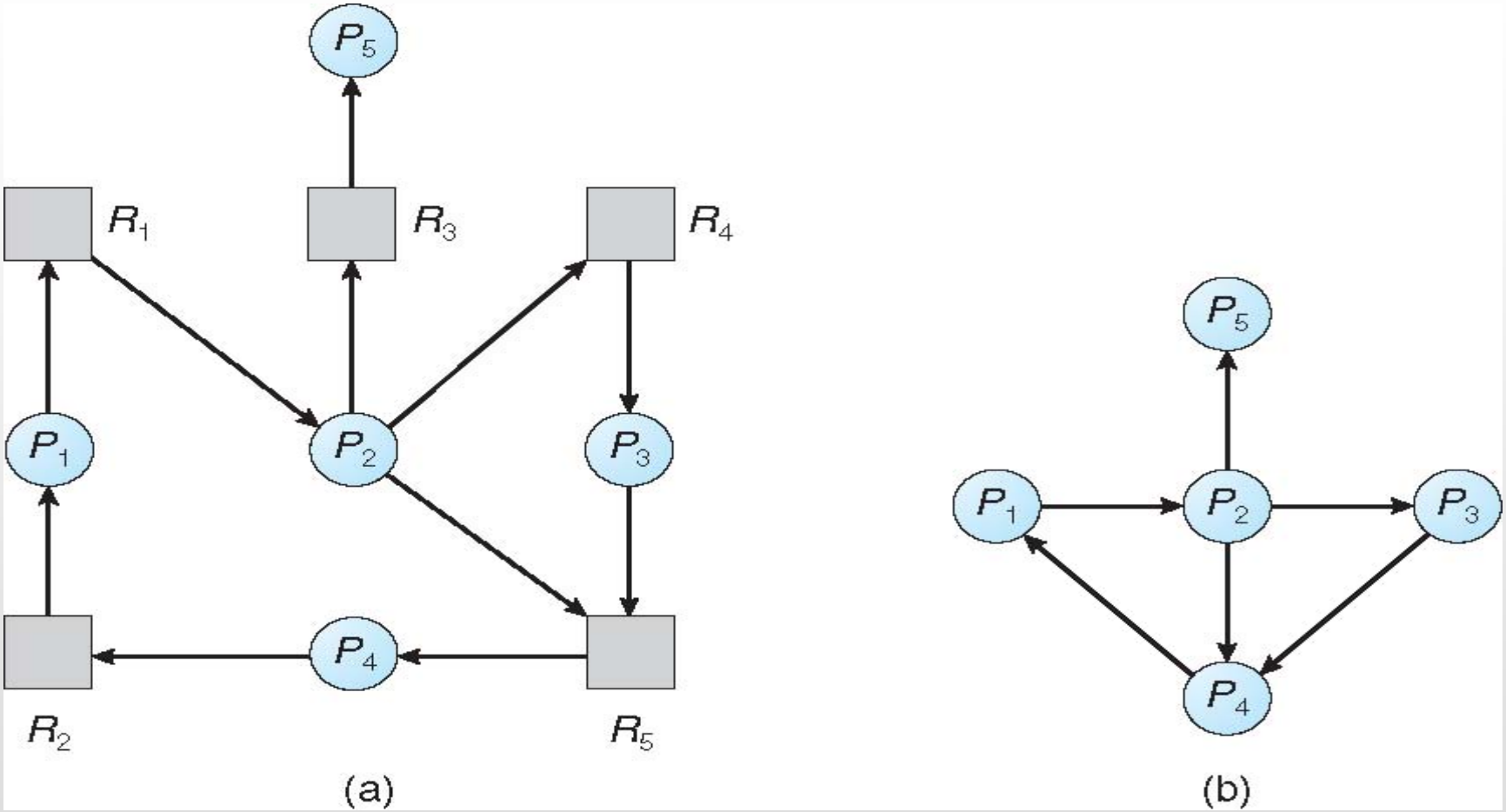
- **Nodes = Procesos (quita los recursos)**
- $P_i \rightarrow P_j$ if ***P_i espera que P_j*** **libere recurso Rq**

Invoca periódicamente un **algoritmo** que **busca** un **ciclo** en el gráfico.

Si hay un **ciclo**, **existe** un **bloqueo**

Un algoritmo para detectar un ciclo en una gráfica requiere un orden de n^2 **operaciones**, donde **n** es el **número** de **vértices** de **Procesos** en el grafico.

Instancia única de cada tipo de recurso



Multiples Instancias de cada tipo de recurso

- **Disponible**: Un **vector** de **longitud *m* indica** el **número** de **recursos disponibles** de **cada tipo**
- **Asignación**: Una **matriz *n* x *m* define** el **número** de **recursos** de **cada tipo asignado** actualmente a **cada proceso**
- **Solicitud**: Una **matriz *n* x *m* indica** la **petición actual** de cada **proceso**. Si **Request [i][j] = *k*** , **entonces** el **proceso Pi** está **solicitando *k* más instancias** del **tipo** de **recurso Rj**. (Proceso se encuentra en **estado de Espera**)

Example of Detection Algorithm

5 procesos P(0)....P(4), 3 recursos A, B, C. A tiene 7 instancias, B tiene 2 y C tiene 6.

	Asignación			Requerimiento			Disponible		
	A	B	C	A	B	C	A	B	C
P(0)	0	1	0	0	0	0	0	0	1
P(1)	2	0	0	2	0	2			
P(2)	3	0	2	0	0	0			
P(3)	2	1	1	1	0	0			
P(4)	0	0	2	0	0	2			

Decimos que el sistema no está en estado de deadlock. Si ejecutamos nuestro algoritmo encontramos la secuencia segura p(0), p(2), p(3), p(1), p(4) que nos da Finish(i) = Verdadero para todo i.

Example of Detection Algorithm

5 procesos P(0)....P(4), 3 recursos A, B, C. A tiene 7 instancias, B tiene 2 y C tiene 6.

	Asignación			Requerimiento			Disponible		
	A	B	C	A	B	C	A	B	C
P(0)	0	1	0	0	0	0	0	0	1
P(1)	2	0	0	2	0	2			
P(2)	3	0	2	0	0	**2**			
P(3)	2	1	1	1	0	0			
P(4)	0	0	2	0	0	2			

Si ahora **modificamos** el **requerimiento de p(2**) para el **recurso** pase a 4 (es decir que p(2) pida 2instancias más de C) el **sistema está en deadlock.**

A pesar de poder utilizar los recursos asignados a p(0) **no alcanza** para **satisfacer** los **requerimientos** de los **otros procesos**. Luego **existe un deadlock** por los **procesos p(1), p(2), p(3) y p(4).**

Cuando Uso del Algoritmo de Detección

Cuando, y **con qué frecuencia**, invocar **depende** de:

- ¿Con qué **frecuencia** es **probable** que **ocurra** un deadlock?
- ¿**Cuántos procesos tendrán** que **ser revertidos** (rolled back)?

Un aspecto **importante** es la **frecuencia** con la que se **debe ejecutar** el **algoritmo** de **detección** de **interbloqueos**, que **suele** ser un **parámetro** del **sistema**.

Cuando Uso del Algoritmo de Detección

- Una **posibilidad extrema** es **comprobar** el est**a**do **cada vez** que se soli**c**ita un recurso y éste **no puede ser asignado**. **Consume mucha CPU**
- Si el **algoritmo** de **detección** se **invoca arbitrariamente**, puede **haber muchos ciclos** en el **gráfico** de **recursos** y, por lo tanto, **no podríamos** decir cuál de los **muchos procesos bloqueados** "**causó**" el **bloqueo**.

Cuando Uso del Algoritmo de Detección

- Otra **alternativa** es **activar** el **algoritmo ocasionalmente**, a **intervalos regulares** o cuando **uno** o **más procesos** queden **bloqueados** durante un **tiempo** sospechosamente **largo**.
- Por ejemplo **1 hora**, o **cuando** la **utilización** del **Procesador desciende** su actividad por **debajo** de un determinado **porcentaje** de **utilización**, pues un '**deadlock**' eventualmente **disminuye** el **uso** de **procesos** en actividad.

Recuperación frente al deadlock

Cuando un algoritmo de detección determina que existe un interbloqueo, existen varias alternativas para tratar de eliminarlo:

☑**Caso 1:** El **Operador** lo puede **resolver manualmente**. (Informar al operador del SO)

☑**Caso 2:** Esperar que el **sistema** se **recupere automáticamente** del **deadlock**. **El Sistema rompe** el deadlock:

- **Abortar** 1 o más **Procesos** para **romper ciclo**
- **Expropiar recursos** a 1 o más **Procesos del ciclo**

Cómo Recuperarse frente al deadlock

- **Violar** la **exclusión mutua** y **asignar** el **recurso** a **varios procesos**
- **Cancelar** los **procesos suficientes** para romper la **espera circular**
- **Desalojar recursos** de los **procesos** en **deadlock**.

Recuperación frente al deadlock

Para **eliminar** el deadlock **matando procesos** pueden **usarse 2 métodos**:

☑ **Matar todos** los procesos en estado de deadlock. **Simple** pero a un **muy alto costo**.

☑ **Matar de a un proceso por vez hasta eliminar el ciclo.** Este **método** requiere considerable **overhead** ya que por **cada proceso** que vamos **eliminando** se debe **re-ejecutar** el **Algoritmo** de **Detección** para **verificar** si el **deadlock** efectivamente **desapareció** o **no**.

Recuperación frente al deadlock

TERMINACIÓN DE PROCESOS

- Puede **no** ser **fácil terminar** un **proceso**
- (p.e. si se encuentra **actualizando** un **archivo**).
- Cuando se utiliza el método incremental, se presenta un nuevo **problema** de política o **decisión**: **selección de la víctima**
- Se debe **seleccionar** aquel **proceso** cuya **terminación represente** el **coste mínimo para el sistema**.

Recuperación: Criterios para elegir proceso "víctima"

¿En qué orden debemos optar por abortar/terminar?

- ☑ **Prioridad más baja**.
- ☑ **Menor cantidad** de **tiempo** de **CPU** hasta el momento.
- ☑ **Mayor tiempo restante estimado para terminar**.
- ☑ **Menor cantidad** de **recursos asignados** hasta ahora.
- ☑ Hay **recursos** del **proceso** que **deben completarse?**
- ☑ ¿**Cuántos procesos tendrán** que ser **terminados**?
- ☑ ¿El **proceso** es **interactivo** o **batch**? Puede **volver atrás**?

Ideal: **elegir** un **proceso** que se **pueda volver** a **ejecutar sin problemas** (ej. una compilación)

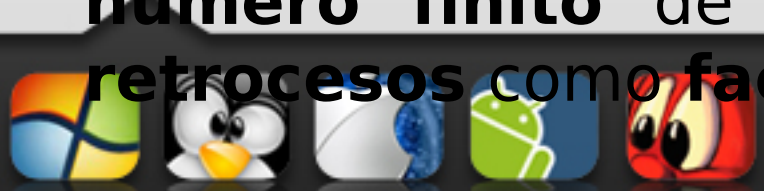
Recuperación frente al deadlock

EXPROPIACIÓN DE RECURSOS

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:

- **Apropiación/Selección de Víctima - minimizar el costo**
- **Rollback - regresar** a algún **estado seguro**, **reiniciar** el **proceso** para **ese estado**
- **Inanición -** el **mismo proceso siempre** se puede **escoger** como **víctima**. **Asegurar** que se **elige** un **numero finito** de **veces**, **Incluir** el **número** de **retrocesos** como **factor** de **coste**



Recuperación Por apropiación

Selección de Víctima

- ☑ **A qué proceso** le **saco** los **recursos**?
- ☑ A **cuáles recursos**?
- ☑ A **qué proceso** se lo **asigno**?
- Elegir la de **mínimo costo**

Cómo?

- ☑ Puede haber **intervención** "**manual**" del **operador**.
- ☑ Se puede **aplicar** según la **naturaleza** del **recurso**

Retroceso (Rollback)

- **Volver** hacia una **instancia segura** del **proceso** que le sacamos los recursos.
- Luego **relanzar** el **proceso**
- **Establecer puntos** de **comprobación** (check points)
- **Información asociada** a esos **puntos**:
  - **Imagen** de la **memoria**
  - **Recursos asignados** en **ese momento**
- **No sobreescribir check points anteriores**

Pasos en recuperación con rollback

1. **Detectar** el **interbloqueo**
2. **Detectar recursos** que se **solicitan**
3. **Detectar** qué **procesos** tienen esos **recursos**
4. **Volver atrás antes** de la **adquisición** del **recurso** (check points anteriores)
5. **Asignación** del **recurso** a **otro proceso**

Inanición

1. el **mismo proceso siempre** se puede **escoger** como **víctima**. (ej. Si se asignan prioridades)

Solución:

1. **Asegurar** que se **elige** un **numero finito** de **veces**
2. **Incluir** el **número** de **retrocesos** como **factor** de **coste**

Resumen de Detección y Recuperación

- La **Detección y Recuperación** de interbloqueos **proporciona** un **mayor grado** potencial de **concurrencia que** las técnicas de **Prevención** o de **Evitación**.
- Además, hay **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 **rollback**.
- 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**.

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**.
- 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 apropiaciones** o permitir **predicciones** precisas.

Estrategia combinada para el manejo de interbloqueos

- **Determinar que** se va a **usar** para **evitar** el **deadlock depende** de la **clase de recursos**.
- La **ordenación** de **recursos**, vista en P**revención**, puede **servir** para **prevenir interbloqueos entre clases**.
- **Un sistema** que **emplee** esta **estrategia** de **“clases de recursos” no estará sujeto a deadlocks**.
- **Incluso produciéndose** un **deadlock**, **no involucrará más** de **una** “**clase**”, ya que se **utiliza** la **técnica** de **ordenamiento de recursos**.

Estrategia combinada para el manejo de interbloqueos

Ejemplo:

Consideremos un **sistema con 4 clases de recursos ordenados** de esta forma:

1. **Recursos internos**: utilizados por el sistema, por ejemplo **BCP/PCB (tabla de procesos)**
2. **Memoria Central**: **memoria** usada por el proceso del **usuario**
3. **Recursos del Proceso**: **dispositivos asignables. Ejemplo impresoras o unidades extraíbles (cintas, discos, floppies) y archivos.**
4. **Espacio de swapping**: espacio del proceso del usuario en **almacenamiento secundario**

Estrategia combinada para el manejo de interbloqueos

Solución ideal: para **prevenir** el **Deadlocks** es con la **ordenación** las **clases** y la **estrategia** a **cada clase**:

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 **difícilmente** ser **tolerados**
- La **Prevención** mediante ordenación de recursos es probablemente la **mejor alternativa**.
- **Numeración** de **recursos** demostró que **nunca entra** en **Deadlock**

Estrategia combinada para el manejo de interbloqueos

1. **Memoria Central**

- La **existencia** de **almacenamiento de intercambio** hace que la **Prevención** mediante a**propiación/desalojo** sea una **elección razonable**. (Memoria Virtual, Swapping)
- 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**.

Estrategia combinada para el manejo de interbloqueos

1. **Recursos del Proceso**

- La **Evitación** se ve **facilitada** por **pre-reclamar** las **necesidades** de **recursos** que habitualmente se efectúa mediante **instrucciones de control**. **Solicitado y asignado en forma total**
- La **Prevención** también es **posible mediante** la **ordenación** de **recursos**
- La **Detección y Recuperación** **no** es **deseable** por la **posibilidad** de **modificación de archivos** que **pertenecen** a esta **clase** de recursos

Estrategia combinada para el manejo de interbloqueos

1. **Espacio de Swapping**

- Una posibilidad es la **adquisición anticipada** de **todo** el **espacio necesario** en **disco** (**Prevención**), ya que **se conocen** los **requisitos previos** de **almacenamiento**.
- **Preasignación** del **espacio TOTAL NECESARIO**