



Sistemas Operativos I

Tema 5

Interbloqueos



Equipo de Sistemas Operativos DISCA / DSIC

UPV

Tema 5: Interbloqueos



◦ Contenido

- 1.- Concepto de interbloqueo.
- 2.- Caracterización formal.
 - Modelo de sistema.
 - Representación gráfica.
 - Condiciones de Coffman.
- 3.-Técnicas de tratamiento de interbloqueos
 - Prevención.
 - Evitación: el algoritmo del banquero.
 - Detección y recuperación.

◦ Bibliografía

- A. Silberschatz, P. Galvin.
"Sistemas Operativos.
Conceptos Fundamentales"
(3ª / 5ª Ed.). Addison-Wesley

Tema 5: Interbloqueos



1.- Concepto de interbloqueo.

2.- Caracterización formal.

- Modelo de sistema.
- Representación gráfica.
- Condiciones de Coffman.

3.-Técnicas de tratamiento de interbloqueos

- Prevención.
- Evitación: el algoritmo del banquero.
- Detección y recuperación.

1.- Concepto de interbloqueo.



◦ Concepto

- Datos:
 - un conjunto de *procesos* ejecutándose en un sistema (computador),
 - un conjunto de *recursos* que son utilizados por dichos procesos,

se dice que el conjunto de procesos se encuentra en un estado de **interbloqueo** cuando **todos** sus procesos se encuentran **esperando** un recurso que mantiene retenido otro proceso del grupo.

- En esa situación:
 - Ningún proceso del grupo puede evolucionar (suspendido eternamente).
 - Ningún otro proceso podrá obtener los recursos retenidos, puesto que no pueden ser liberados.
- Los interbloqueos constituyen un grave problema para el que la mayoría de sistemas operativos (como UNIX, por ejemplo) no contemplan ningún tratamiento en absoluto.

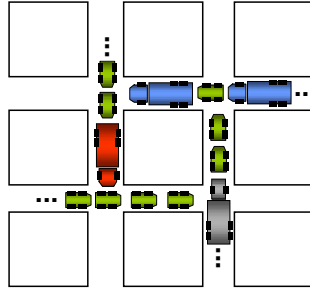
1.- Concepto de interbloqueo.



◦ Ejemplos

- El sistema tiene dos unidades de cinta, P_1 y P_2 tienen cada uno una unidad y necesitan la otra.
- Semáforos $SemA$ y $SemB$, inicializados a 1
- Situación de tráfico en interbloqueo

Proceso1	Proceso2
$P(SemA)$	$P(SemB)$
$P(SemB)$	$P(SemA)$



Tema 5: Interbloqueos



1.- Concepto de interbloqueo.

2.- Caracterización formal.

- Modelo de sistema.
- Representación gráfica.
- Condiciones de Coffman.

3.- Técnicas de tratamiento de interbloqueos

- Prevención.
- Evitación: el algoritmo del banquero.
- Detección y recuperación.

2.- Caracterización formal



◦ Modelo de sistema

- Conjunto de **procesos**, identificados por $P_1, P_2, \dots, P_i, \dots, P_n$.
- Conjunto de **recursos**, identificados por $R_1, R_2, \dots, R_j, \dots, R_m$. Estos recursos pueden ser **físicos** (discos, cintas, impresoras, etc.), o **lógicos** (monitores, semáforos, etc.).

De cada recurso puede haber una o más **instancias**. Dos recursos se consideran en realidad instancias del mismo recurso si un proceso que solicita dicho recurso considera que puede obtener cualquiera de ellas indistintamente.

- El uso que un proceso hace de un recurso sigue este **protocolo**:
 - **Petición** del recurso: Si no está disponible, el proceso queda suspendido hasta que lo esté.
 - **Uso** del recurso.
 - **Liberación** del recurso.

2.- Caracterización formal



◦ Condiciones de Coffman:

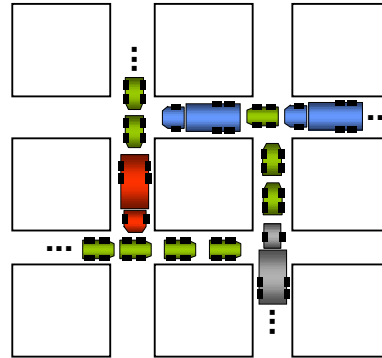
- Se ha demostrado que las siguientes cuatro condiciones son **necesarias** (aunque no suficientes) para que se produzca un interbloqueo:
 - 1) **Exclusión mutua**: Al menos un recurso debe ser utilizado en exclusión mutua, es decir, de modo no compartido.
 - 2) **Retener y esperar**: Debe haber al menos un proceso que retenga un recurso y que haya pedido algún otro recurso que posea otro proceso, por lo que estará esperando.
 - 3) **No expulsión**: El sistema no puede arrebatarse los recursos que ha asignado previamente a los procesos. En otras palabras, un proceso mantiene retenido un recurso hasta que deja de utilizarlo y lo libera voluntariamente.
 - 4) **Espera circular**: Debe existir un conjunto de procesos $\{P_1, P_2, \dots, P_n\}$ tal que P_1 se encuentra esperando un recurso que retiene P_2 , P_2 espera un recurso que retiene P_3 , ..., P_{n-1} espera un recurso que mantiene P_n , y P_n espera un recurso que mantiene P_1 .
- Si todas ellas se cumplen simultáneamente, el sistema se encuentra en situación de **riesgo** de sufrir un interbloqueo.

2.- Caracterización formal



◦ Ejemplo

- Situación de tráfico en interbloqueo:
 - Cada sección de la calle se considera un recurso.
 - **Exclusión mutua:** sólo un vehículo puede ocupar una sección de la calle.
 - **Retener y esperar:** cada vehículo ocupa una sección de la calle y está esperando para moverse a la siguiente sección.
 - **No expulsión:** no se puede quitar una sección de la calle ocupada por un vehículo. El vehículo la liberará cuando se mueva a la siguiente sección.
 - **Espera circular:** cada vehículo está esperando a que mueva el vehículo de enfrente.



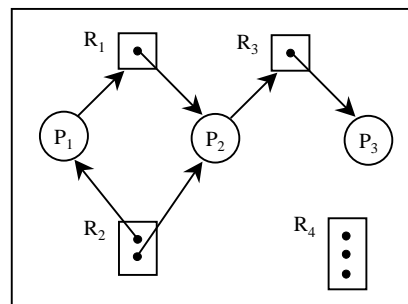
2.- Caracterización formal



◦ Representación gráfica: Grafo de asignación de recursos.

- Una asignación concreta de recursos a procesos puede ser representada de forma gráfica mediante un grafo en el que:
 - Existen dos tipos de **nodos**: círculos (procesos) y cuadrados (recursos). Los cuadrados poseen tantos puntos en su interior como instancias haya de dicho recurso.
 - Los **arcos** son dirigidos. Si van de un proceso a un recurso, indican **petición**, y si van de un recurso a un proceso, indican **asignación**. En este último caso, el arco va en realidad de una instancia concreta (punto) al proceso.

- Ejemplo:

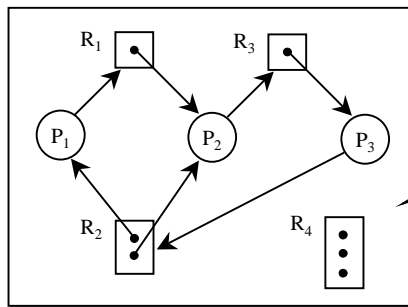


2.- Caracterización formal



o Grafo de asignación de recursos (2)

- Interpretación del grafo:
 - Si no existen ciclos (dirigidos), entonces **no hay** interbloqueo.
 - Si existe algún ciclo dirigido, entonces:
 - Si hay sólo una instancia de cada recurso, entonces **hay** un interbloqueo.
 - Si hay más de una instancia, entonces **puede haber** un interbloqueo.
- Ejemplo anterior: Ahora P_3 solicita R_2 .



P_1 está esperando un recurso que posee P_2 , que espera un recurso que posee P_3 , que espera un recurso que poseen P_1 y P_2 .



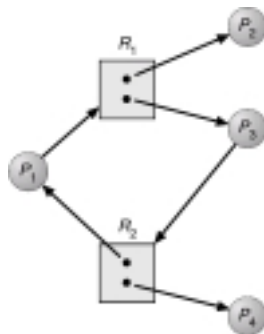
¡ INTERBLOQUEO !

2.- Caracterización formal



o Grafo de asignación de recursos (3)

- Ejemplo de grafo de asignación de recursos con un ciclo pero sin interbloqueo



Tema 5: Interbloqueos



1.- Concepto de interbloqueo.

2.- Caracterización formal.

- Modelo de sistema.
- Representación gráfica.
- Condiciones de Coffman.



3.-Técnicas de tratamiento de interbloqueos

- Prevención.
- Evitación: el algoritmo del banquero.
- Detección y recuperación.

3.-Técnicas de tratamiento de interbloqueos



◉ Alternativas en el tratamiento de interbloqueos:

- Ignorar el problema, asumiendo que dicha situación nunca se dará en el sistema. Es la aproximación que mantienen muchos sistemas operativos, incluido UNIX.
- Emplear algún algoritmo o protocolo que asegure que **nunca** se va a poder producir un interbloqueo. Esta solución puede adoptar dos formas alternativas:
 - **Prevención.** Consiste en conseguir que no puedan darse simultáneamente las cuatro condiciones de Coffman. De esta forma, el interbloqueo no puede llegar a producirse.
 - **Evitación.** Consiste en llevar la cuenta de los recursos disponibles en el sistema, los recursos que poseen los procesos y los que pueden llegar a solicitar. Cada vez que un proceso hace una petición de un recurso, el sistema analiza toda esa información para conceder (o denegar) dicho recurso.
- Utilizar un algoritmo que pueda detectar una situación de interbloqueo (**detección**) y seguir alguna técnica que permita deshacer dicha situación (**recuperación**).

Prevencción de interbloqueos



◦ Veamos cuáles de las condiciones de Coffman son evitables, y cómo:

- **Exclusión mutua:**

No es posible eliminar esta condición, pues la mayoría de recursos son inherentemente no compartibles (i.e., *reutilizables serie*).

- **Retener y esperar:**

Para deshacer esta condición, se debe obligar a los procesos a:

- Solicitar todos sus recursos de una vez, al principio de su ejecución, o bien...
- Utilizar los recursos de uno en uno, liberando cada recurso antes de solicitar el siguiente. En cualquier caso, si necesita más de uno a la vez, debe entonces solicitar todos ellos al principio.

Estas aproximaciones tienen dos **problemas**:

- **Baja utilización** de los recursos, puesto que estarán retenidos desde el principio de la ejecución de los procesos, pero evidentemente no se estarán utilizando en todo momento.
- **Inanición** de los procesos que necesiten muchos recursos solicitados muy frecuentemente por los demás procesos.

Prevencción de interbloqueos (2)



- **No expulsión:**

Para eliminar esta restricción, se puede aplicar el siguiente algoritmo: Cuando un proceso P solicita recursos que no están libres, el sistema examina si los poseen procesos que a su vez están esperando. En ese caso, el sistema **se apropia** de dichos recursos y se los ofrece a P.

Los procesos a los que se les ha arrebatado recursos sólo podrán continuar cuando obtengan de nuevo dichos recursos, más los que estaban esperando.

- **Espera circular:**

Esta condición se puede romper si imponemos un **orden total** a los recursos, y se obliga a que los procesos soliciten siempre los recursos siguiendo dicho orden. Es decir:

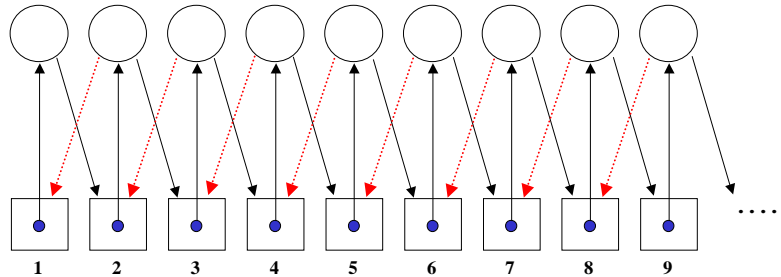
- Definimos la función $F: R \rightarrow N$, que asocia a cada recurso un número natural.
- Un proceso que posee un recurso R_i puede solicitar otro recurso R_j si y sólo si $F(R_i) < F(R_j)$ (o bien ha liberado los recursos que no cumplen con la condición).

Prevencción de interbloques (3)



• Espera circular. Orden total:

- En una hipotética cadena circular, cada proceso P_i (que posee R_i) espera el recurso R_{i+1} (que posee P_{i+1}). Con eso tendríamos que $\forall i F(R_i) < F(R_{i+1})$.
- Pero entonces se cumpliría que: $F(R_0) < F(R_1) < \dots < F(R_n) < F(R_0)$, lo que es imposible por definición del orden total.



No está permitido, no se cumple $F(R_i) < F(R_j)$

Tema 5: Interbloques



1.- Concepto de interbloqueo.

2.- Caracterización formal.

- Modelo de sistema.
- Representación gráfica.
- Condiciones de Coffman.

3.-Técnicas de tratamiento de interbloques



- Prevención.
- Evitación: el algoritmo del banquero.
- Detección y recuperación.

Evitación de interbloques. Generalidades



En este tipo de técnicas, el sistema necesita conocer:

- La necesidad máxima de recursos de los procesos que se están ejecutando.
- La asignación actual de recursos a procesos.
- La cantidad actual de instancias libres de cada recurso en el sistema.

A partir de dicha información, se determina:

- **Estado seguro:** un estado (de asignación de recursos) se considera seguro si en él no hay posibilidad de interbloqueo.

Para que un estado sea seguro, es necesario que los procesos formen una **secuencia segura**. Una secuencia segura es una cierta ordenación de los procesos que cumple que los recursos que aún puede pedir cualquier P_i pueden ser satisfechos con los recursos libres más los recursos retenidos por los $P_j, j < i$.

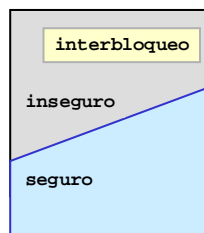
Si dicha secuencia existe, como mucho cada proceso tendrá que esperar a que liberen sus recursos los procesos que van antes que él en la secuencia.

En base a ello, cuando un proceso realice una **petición** de recursos, el sistema se los **concederá sólo** en el caso de que la petición mantenga al sistema en un **estado seguro**.

Evitación de interbloques. Generalidades



Espacios de estado seguro, inseguro e interbloqueo

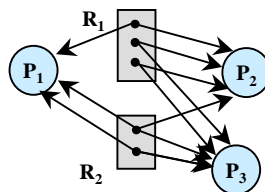


Secuencia segura

- es una cierta ordenación de los procesos que cumple que los recursos que aún puede pedir cualquier P_i pueden ser satisfechos con los recursos libres más los recursos retenidos por los $P_j, j < i$.

Aún puede solicitar

P_1	1	2
P_2	3	1
P_3	2	1
	R_1	R_2



Secuencias seguras:

- P_2, P_3 y P_1
- P_3, P_2 y P_1



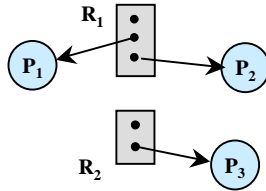
Estado SEGURO

Evitación de interbloques. Generalidades

○ Situación 2

Aún puede solicitar

P ₁	0	2
P ₂	2	1
P ₃	2	1
	R ₁	R ₂



No existe una Secuencia segura

↓
Estado INSEGURO

↓
Dependiendo de cuándo se soliciten los recursos y cuando se liberen, podrá darse un interbloqueo o no

Evitación. El algoritmo del banquero

○ Suponemos:

- n procesos
- m recursos

○ Estructuras de datos necesarias:

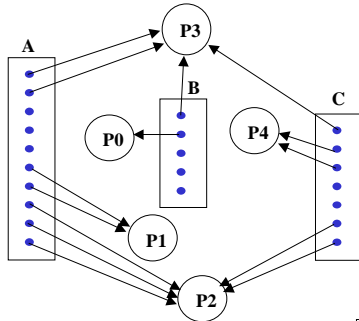
Disponibles[m]	Cantidad de instancias disponibles de cada recurso.
Max[n, m]	Máximo nº de instancias de cada recurso que cada proceso puede pedir
Asignado[n, m]	Nº de instancias de cada recurso actualmente asignadas a cada proceso.
Necesito[n, m]	Nº de peticiones de instancias de cada recurso que cada proceso aún no ha hecho.

NOTACION: En las matrices como Asignado denotaremos por Asignado[i] a la fila i-ésima de dicha matriz.

Evitación. El algoritmo del banquero



Un ejemplo:



Disponible		
3	3	2
A	B	C

	Asignado		
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2
	A	B	C

	Necesito		
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1
	A	B	C

	MAX		
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3
	A	B	C

Evitación. El algoritmo del banquero



ALGORITMO 1: Algoritmo de seguridad.

Establece si el sistema se encuentra actualmente en un estado seguro.

- Estructuras de datos auxiliares:
 - Trabajo[m]: Acumula los recursos de los procesos que pueden evolucionar
 - Acabado[n]: Booleano que indica cuando un proceso ha acabado

Algoritmo:

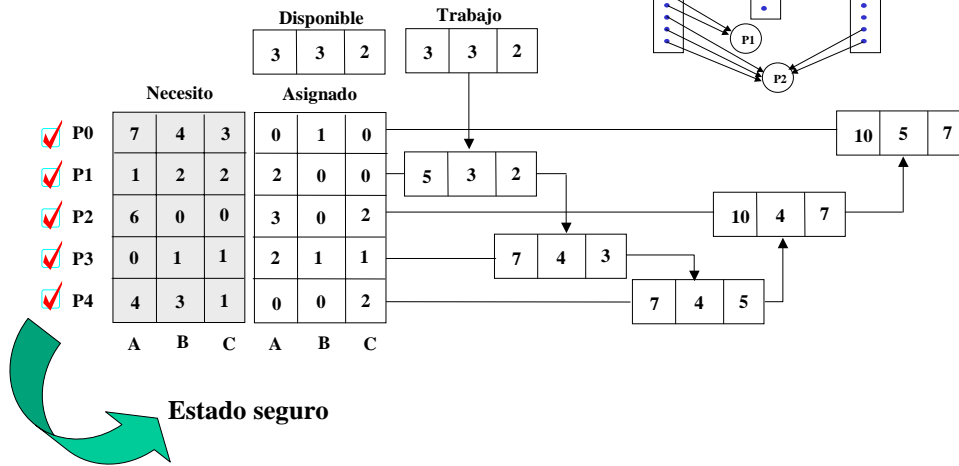
```

Funcion Seguridad retorna Boolean
Trabajo := Disponible
Para todo i
    Acabado[i] := false
FinPara
Mientras  $\exists$  i tal que Acabado[i]=False AND Necesito[i]<=Trabajo
    Trabajo := Trabajo + Asignado[i]
    Acabado[i] := True
FinMientras
Si  $\forall$  i Acabado[i]=True Entonces
    Seguridad := True
Sino
    Seguridad := False
FinSi
Fin Seguridad
    
```

Evitación. El algoritmo del banquero



Un ejemplo:



Evitación. El algoritmo del banquero



ALGORITMO 2: Algoritmo de petición de recursos.

Lo utiliza el sistema para averiguar si puede satisfacer una petición de recursos.

- Estructura de datos auxiliar:
 - Peticion[i]: Vector de tamaño m que refleja los recursos que pide el proceso i.

Algoritmo:

Invocación al algoritmo anterior

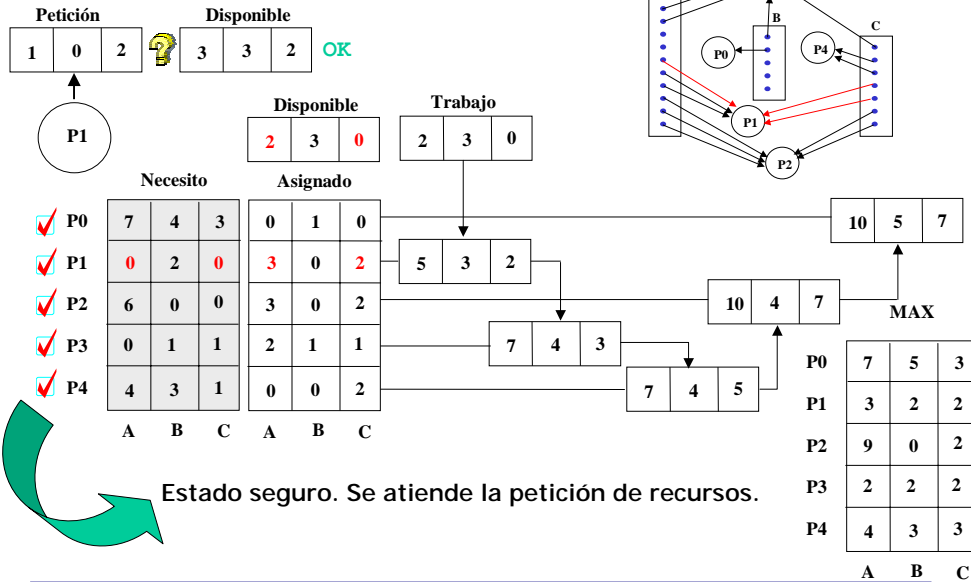
```

Procedimiento Peticion_Recursos( Peticion[i] )
Si Peticion[i] > Necesito[i]
Entonces ERROR
Finsí
Si Peticion[i] > Disponible
Entonces Suspender_Proceso(i)
Sino
Disponible := Disponible - Peticion[i]
Asignado[i] := Asignado[i] + Peticion[i]
Necesito[i] := Necesito[i] - Peticion[i]
Si Seguridad
Entonces Dar_Recursos_A(i)
Sino
Recuperar_estado_previo()
Suspender_Proceso(i)
Finsí
Finsí
Fin Peticion_Recursos
    
```

Evitación. El algoritmo del banquero



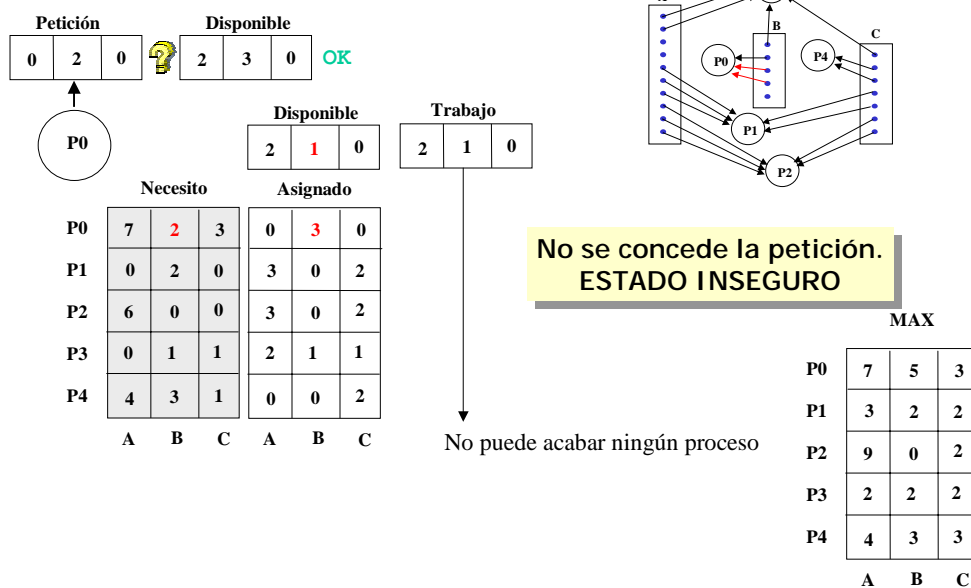
Un ejemplo:



Evitación. El algoritmo del banquero



Un ejemplo:



Tema 5: Interbloqueos



1.- Concepto de interbloqueo.

2.- Caracterización formal.

- Modelo de sistema.
- Representación gráfica.
- Condiciones de Coffman.

3.-Técnicas de tratamiento de interbloqueos

- Prevención.
- Evitación: el algoritmo del banquero.
- Detección y recuperación.



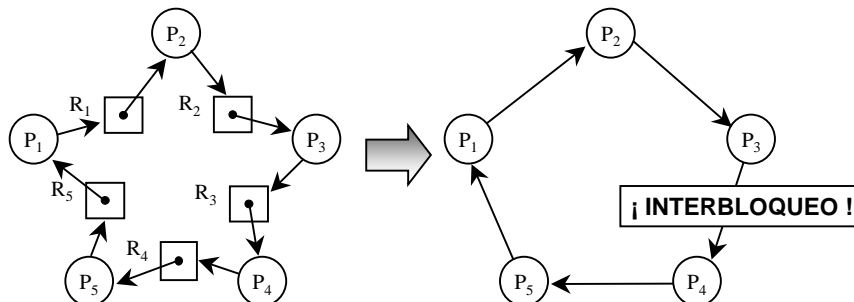
Detección de interbloqueos



◦ Algoritmos de detección. Casos:

- 1) Si sólo hay recursos de instancia única \Rightarrow **Grafo de "espera-a"** (*wait-for*)
 - Se basa en el grafo de asignación de recursos.
 - Se eliminan los nodos correspondientes a recursos, y se ajustan los arcos de forma que **habrá un arco del proceso P_i al proceso P_j si P_j posee un recurso que P_i ha solicitado.**
 - Existirá un interbloqueo **si y sólo si** hay un ciclo en el grafo.

Ejemplo:



Detección de interbloqueos (2)



2) Si hay recursos de múltiples instancias → **Algoritmo de detección**

▫ **Estructuras de datos:**

- Disponible[m]: Número de instancias disponibles de cada recurso.
- Asignado[n,m]: Cantidad de instancias asignadas a los procesos.
- Peticion[n,m]: Cantidad de peticiones de instancias de recursos actualmente.

▫ **Algoritmo:**

Aquellos procesos para los que Acabado[i] sea falso, forman parte de un interbloqueo.

```

Funcion Deteccion retorna Boolean
Trabajo := Disponible
Para todo i
    Si Asignado[i] <> 0
        Entonces Acabado[i] := False
        Sino Acabado[i] := True
    FinSi
FinPara
Mientras ∃ i tal que Acabado[i]=False AND
                    Peticion[i]<=Trabajo
    Trabajo := Trabajo + Asignado[i]
    Acabado[i] := True
FinMientras
Si ∀ i Acabado[i]=True
    Entonces Deteccion := False
    Sino Deteccion := True
FinSi
Fin Deteccion
    
```

Detección de interbloqueos (3)



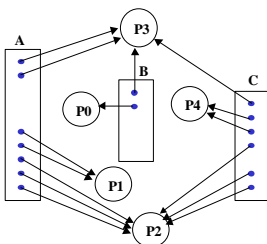
◦ **¿Cuándo se invoca este algoritmo? Alternativas:**

- Cada vez que una petición de recursos no puede ser concedida inmediatamente.
- Cada cierto intervalo de tiempo.

◦ **Comparación:**

- La primera alternativa produce más sobrecarga, pero es más fácil detectar qué procesos provocan el interbloqueo (pues como mucho habrá un interbloqueo).
- Con la segunda alternativa podemos medir la sobrecarga, pero si existen varios ciclos será difícil determinar qué procesos están involucrados en cada uno de los interbloqueos.

◦ **Ejemplo:**



	Petición		
	A	B	C
P0	0	0	0
P1	2	0	2
P2	0	0	0
P3	1	0	0
P4	0	0	2

- 1 La secuencia <P0, P2, P3, P1, P4> finaliza con Acabado[i] = true para todo i.
- 2 P2 pide una instancia adicional del recurso C :
Peticion[P2] = 0 0 1
Cuál es el **estado** del sistema?
▫ Existe un **interbloqueo**, en el que intervienen los procesos P1, P2, P3, y P4.

Recuperación de interbloqueos



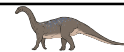
Alternativas:

- Terminación de procesos.
 - Terminación de **todos** los procesos interbloqueados.
 - Terminación **iterativa** de procesos, hasta que el interbloqueo desaparezca.
- Apropiación de recursos. El sistema se apropia de recursos de los procesos interbloqueados, hasta que desaparece el interbloqueo.

Problemas a resolver:

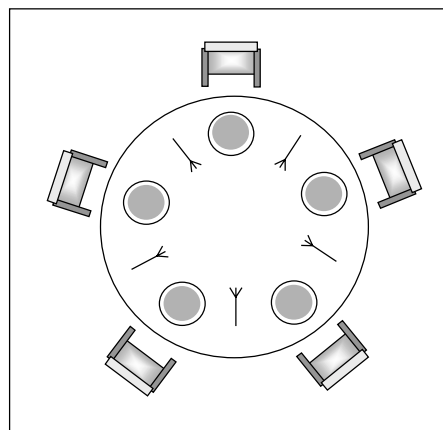
- **Selección de una víctima:** A quién se elige para apropiarse de sus recursos.
- **Vuelta atrás:** El proceso a quien se le quitan los recursos debe ser vuelto a un estado seguro. La solución trivial es abortar dicho proceso.
- **Inanición.** Hay que considerar que no se debería quitar siempre los recursos al mismo proceso, sobre todo si la vuelta atrás supone abortarlo y obligarle a empezar desde el principio.

Ejemplo: El problema de los 5 filósofos



Enunciado :

- En una mesa hay 5 filósofos.
- Cada uno puede comer y pensar.
- Protocolo para comer:
 - **Coger tenedor derecho.**
 - **Coger tenedor izquierdo.**
 - **Comer.**
 - **Dejar tenedor derecho.**
 - **Dejar tenedor izquierdo.**
- Cada tenedor es un recurso **reutilizable serie**. Hay sólo 5 tenedores para los 5 filósofos.



Ejemplo: El problema de los 5 filósofos



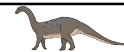
◦ Solución con semáforos:

- Cada tenedor es un semáforo, inicializado a uno.
- Para que un filósofo pueda comer tiene que cerrar el semáforo de su derecha y el de su izquierda.
- Cuando deja de comer, abre ambos semáforos.

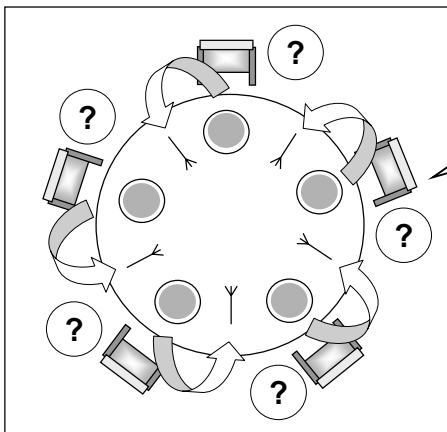
◦ Implementación:

```
var tenedor : array[0..4] of Semaforo := 1;  
  
task Filosofo( i:0..4 )  
  repeat  
    P( tenedor[i] );  
    P( tenedor[(i+1) mod 5] );  
    comer();  
    V( tenedor[i] );  
    V( tenedor[(i+1) mod 5] );  
    pensar();  
  until false;  
end Filosofo;
```

Ejemplo: El problema de los 5 filósofos



◦ Problema: ¿Qué ocurre si cada uno coge el tenedor derecho (i) e intenta coger el izquierdo ($i+1$)?



Los 5 procesos han ejecutado
 $P(tenedor[i]);$
y se quedan esperando en
 $P(tenedor[(i+1) \bmod 5]);$
↓
¡ INTERBLOQUEO !

Soluciones triviales:

- Permitir que sólo 4 filósofos se sienten a la mesa.
- Cada filósofo coge ambos tenedores a la vez, o ninguno.
- Los filósofos pares cogen primero el tenedor derecho y los impares el izquierdo.

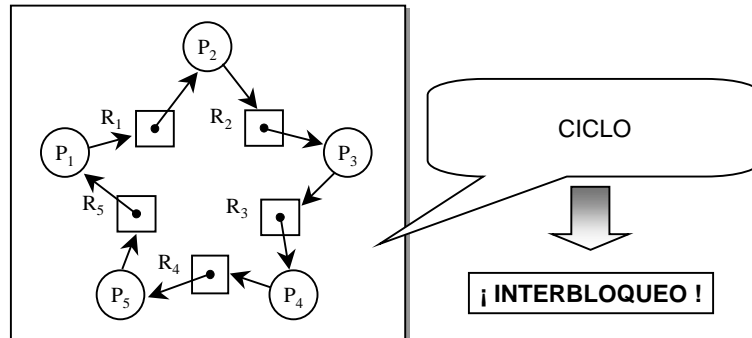
Ejemplo: El problema de los 5 filósofos



Caracterización formal: Grafo de asignación de recursos

- 5 procesos (filósofos).
- 5 recursos (tenedor) de instancia única.

NOTA: Los tenedores no son todos iguales, puesto que cada proceso solicita un tenedor *en concreto* (el de su izquierda o el de su derecha).



Detección de interbloqueos



Ejemplo: 5 filósofos.

- Supongamos que se ha producido el interbloqueo:

Cuando se invoque Deteccion():

Estado inicial:

Trabajo Acabado
0 0 0 0 0 F F F F F

No hay iteraciones posibles.



$\forall i$ Acabado[i] = False
Los cinco procesos están interbloqueados