

**ABRAZO MORTAL -
DEADLOCK -
INTERBLOQUEO**

DEFINICION DE DEADLOCK

Un conjunto de procesos está en estado de "DEADLOCK" cuando cada proceso del conjunto está **esperando** por un evento que solo puede ser causado por otro proceso que está dentro de ese conjunto.

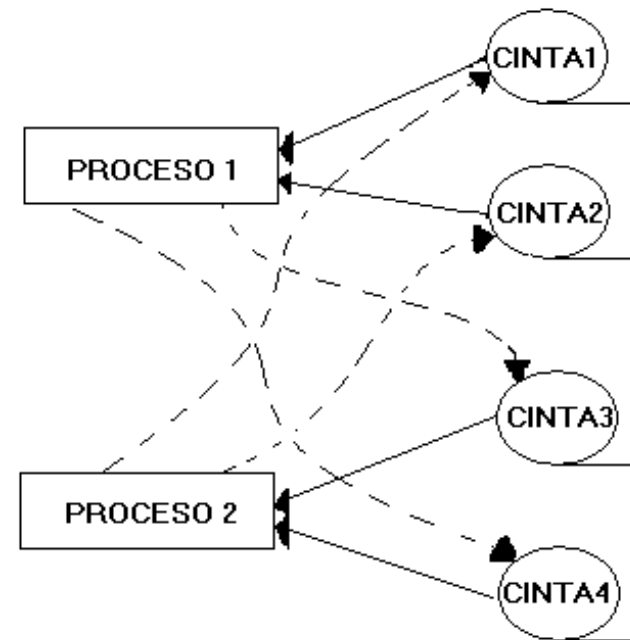
PROBLEMA

Dos objetos (Procesos) desean hacer uso de un único recurso no compartible o un número finito de ellos, y cada uno de ellos posee la porción que el otro necesita.

Ejemplo clásico:

Dos procesos P(1) y P(2), cada uno de ellos tiene asignadas 2 unidades de cinta y cada uno de ellos necesita dos más, pero en el sistema existen sólo 4 unidades en total.

Obviamente estamos en presencia de un deadlock, ya que P(1) espera recursos de P(2) y P(2) espera recursos de P(1).



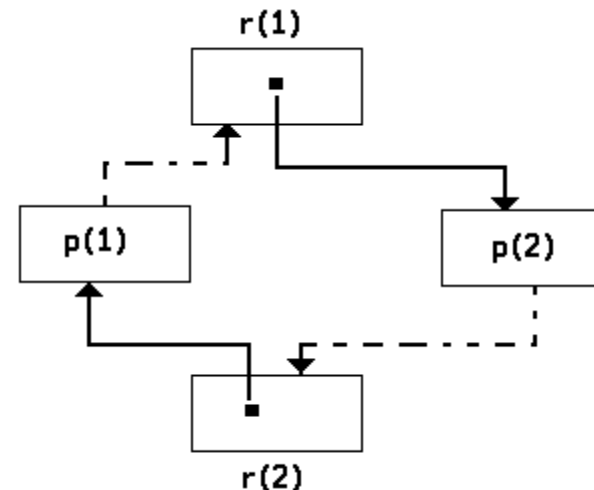
CONDICIONES NECESARIAS PARA EL "DEADLOCK"

Exclusión Mutua: debe haber recursos no compartidos de uso exclusivo.

Espera y Retenido (Hold & Wait): un proceso retiene un recurso y espera por otro

Sin Desalojo: no es posible quitarle el recurso retenido a un proceso. Lo libera voluntariamente.

Espera circular: hay un ciclo de espera entre procesos



El modelo del sistema

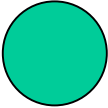

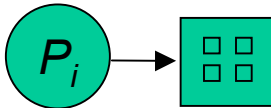
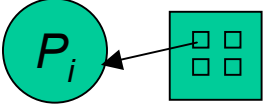
- Tipos de recursos: R_1, R_2, R_3, \dots
- Cada tipo de recurso posee W_i instancias
- Cada proceso utiliza un recurso de esta forma:

Pedirlo : (REQUEST) si no puede ser satisfecho esperar. Será incluido en una cola en espera de ese recurso (Tablas).

Usarlo : (USE) el proceso puede operar el recurso.

Liberarlo : (RELEASE) el proceso libera el recurso que fue pedido y asignado

GRAFO DE ASIGNACIÓN DE RECURSOS (elementos)

- Procesos 
- Tipo de Recurso con 4 instancias 
- P_i pide instancia of R_j 
- P_i retiene una instancia de R_j 

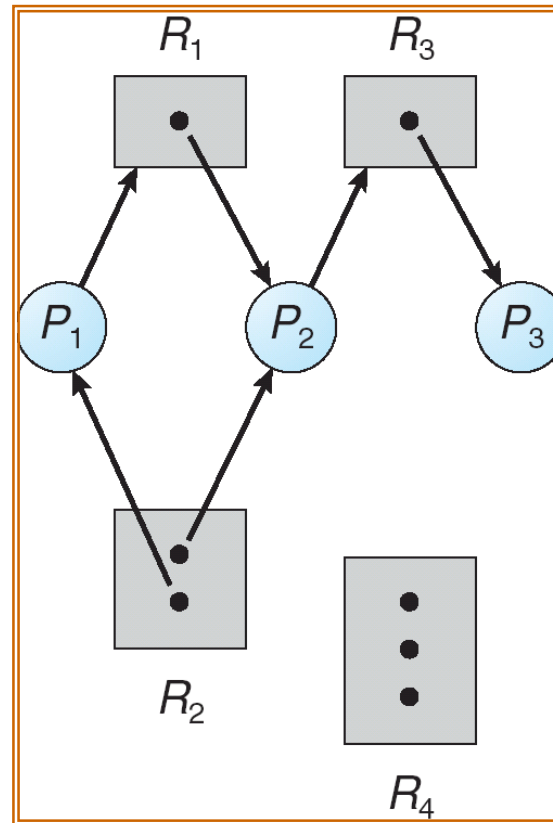
GRAFO DE ASIGNACION DE RECURSOS

- Sea $G = (V, E)$, donde V es el conjunto de vértices y E es el conjunto de flechas (arcos orientados).
- A su vez V está compuesto por 2 tipos:
- $P = \{p(1), p(2), \dots, p(n)\}$ conjunto de Procesos
- $R = \{r(1), r(2), \dots, r(m)\}$ conjunto de Recursos

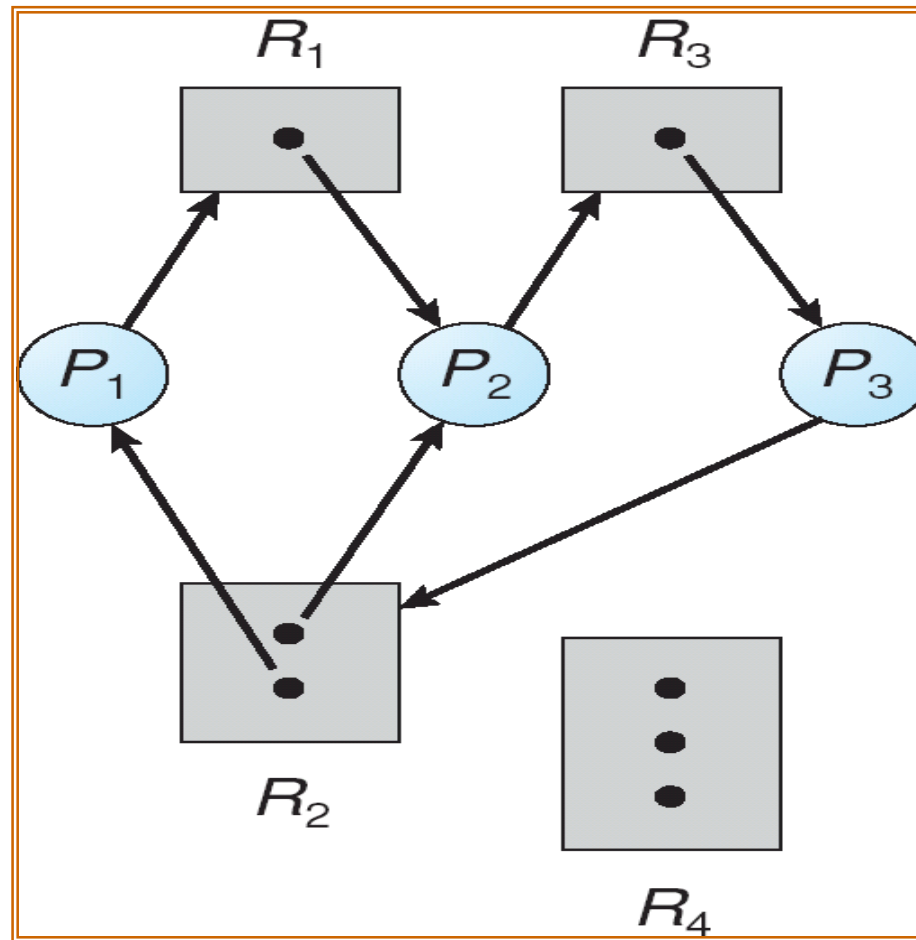
GRAFO DE ASIGNACION DE RECURSOS

- Los elementos de E son:
- $(p(i), r(j)) \implies p(i)$ está esperando una instancia del recurso $r(j)$.
- $(r(j), p(i)) \implies r(j)$ (una instancia de) está asignada al proceso $p(i)$.

Un ejemplo de un grafo de asignación de recursos

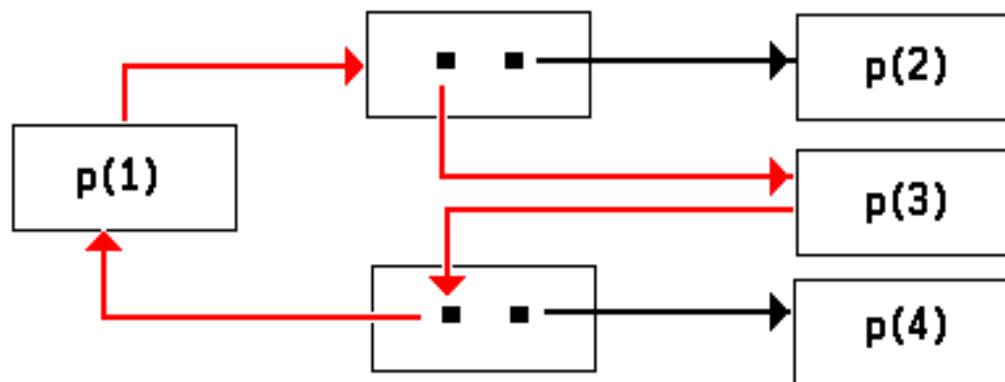


Grafo de asignación de recursos con un Deadlock



GRAFO DE ASIGNACIÓN DE RECURSOS

- En caso de tipos de recursos con más de una instancia, la existencia de un ciclo cerrado es *condición necesaria, pero no suficiente*.
- CONCLUSION: Cuando no existe un ciclo no hay deadlock. Ahora si existe un ciclo puede o no haber deadlock.



Ciclo: p(1) -->
r(1)--> p(3)-->
r(2)--> p(1).

Si p(4) o p(2)
liberan sus
recursos se
rompe el ciclo.

MANEJO DE DEADLOCK

Existen dos maneras de manejar los **Deadlocks**:

- **No permitir que ocurran (Prevención - Evitarlo).**
- **Permitirlo y luego recuperar (es caro y dificultoso).**

PREVENCIÓN

- **EXCLUSIÓN MUTUA**

No hay recursos exclusivos, permiso de acceso en concurrencia (todos leen), pero ...

- **ESPERA Y RETENIDO (Hold & Wait)**

Asignación Total al inicio(dos problemas, asignación sin uso, e inanición)

Si tiene asignado un recurso para pedir otro debe liberar éste.

- **SIN DESALOJO**

Permitirlo. Pérdida de recursos cuando se quiere acceder a uno no disponible.

PREVENCION

- **ESPERA CIRCULAR**

Numeración de recursos

Para asegurar que esta condición no se cumpla consideremos lo siguiente:

A cada "tipo de recurso" le asignamos un número Natural único

$$R = \{r(1), r(2), \dots, r(n)\}$$

$$F: R \rightarrow N$$

$$F(\text{impresora})=1, F(\text{disco})=5, F(\text{cinta})=7$$

PREVENCIÓN

Un proceso puede pedir un $r(j)$ sii $F(r(j)) > F(r(i))$
 $\forall i$ de los recursos que ya posee. Si no
cumple esta condición, debe liberar todos los
 $r(i)$ que cumplen $F(r(i)) \geq F(r(j))$.

Es decir el proceso siempre debe solicitar los
recursos en un orden creciente.

Dadas estas condiciones se puede demostrar
que no tendremos nunca una espera circular,
es decir el sistema está libre de Deadlock

Estado Seguro

- Un estado "seguro" (**SAFE**) es cuando se pueden asignar recursos a cada proceso en algún orden y evitar el deadlock.
- Formalmente: Un sistema está en estado "**seguro**" si existe una "**secuencia segura de procesos**".
- Una secuencia $\langle p(1), p(2), \dots, p(n) \rangle$ es segura si por cada $p(i)$ los recursos que necesitará pueden ser satisfechos por los recursos disponibles más todos los recursos retenidos por todos los $p(j)$ tal que $j < i$

Estado Seguro (un ejemplo)

Tres procesos y un máximo de 12 cintas en la instalación.

Los procesos necesitarían estas cantidades de cintas:

P(0) ----> 10 P(1) ----> 4 P(2) ----> 9

En un instante dado T(0) tienen asignado:

P(0) <--- 5 P(1) <--- 2 P(2) <--- 2 3 Libres

Existe una secuencia segura <P(1),P(0),P(2)>.

	Necesita	Tiene
P(0)	10	5
P(1)	4	2
P(2)	9	2

Estado Seguro (un ejemplo)

P(1) toma 2 más.

Cuando termina libera 4, más la instancia que quedaba libre son 5.

Luego P(0) toma 5. Cuando termina libera las 10 retenidas y luego P(2) toma 7 y luego termina.

Luego estamos en un estado seguro, con lo cual no podemos pasar a un estado de "deadlock" si se asignan los recursos en el orden dado por la secuencia dada.

Estado Seguro

Una secuencia segura es una **simulación** partiendo del estado de un sistema en un momento dado.

NO significa que los pedidos y/o liberaciones vayan a realizarse de esa forma en el mundo real.

Estado Seguro

De un estado "seguro" podemos pasar a uno "inseguro" (UNSAFE).

Supongamos que en el instante $T(1)$, se da un recurso más a $P(2)$:

$P(0) \leftarrow 5$ $P(1) \leftarrow 2$ $P(2) \leftarrow 3$ 2 Libres

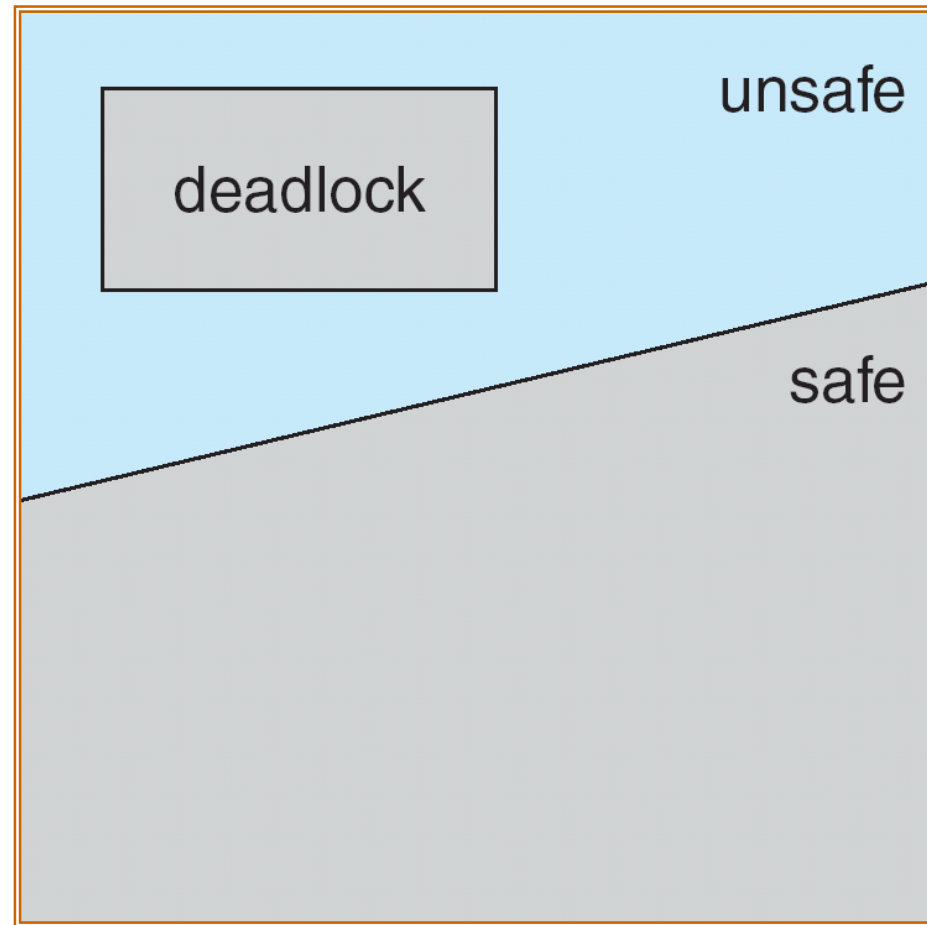
Solo $P(1)$ puede satisfacer sus requerimientos, aquí no existe una secuencia segura, pues ni $P(0)$, ni $P(2)$ pueden obtener la totalidad de recursos que requieren, entonces podemos desembocar en un estado de "deadlock". Si se hubiera respetado la secuencia "segura", esto no se habría producido.

Estado Seguro

CONCLUSIONES:

- *Un estado de "deadlock" **es** un estado "inseguro".*
- *Un estado "inseguro" puede desembocar en un "deadlock", pero **no necesariamente** lo es.*
- *Si aún pidiendo un recurso, y estando éste libre se debe esperar, esto reduce la utilización del recurso*

Safe, Unsafe , Deadlock



ALGORITMOS PARA EVITAR EL DEADLOCK

El Algoritmo del Banquero. (The Banker's Algorithm) Dijkstra (1965) Habermann (1969)

Se basa en determinar si los recursos que están libres pueden ser adjudicados a procesos sin abandonar el estado "seguro".

Supondremos N procesos y M clases de recursos y ciertas estructuras de datos

Banquero y Seguridad

- **Disponible** : Vector de longitud M. $\text{Disponible}(j) = k$, indica que existen k instancias disponibles del recurso $r(j)$.
- **MAX** : Matriz de $N \times M$. Define la máxima demanda de cada proceso.
 $\text{MAX}(i,j) = k$, dice que $p(i)$ requiere a lo sumo k instancias del recurso $r(j)$.
- **Asignación** : Matriz de $N \times M$. Define el número de recursos de cada tipo actualmente tomados por cada proceso.
 $\text{Asignación}(i,j) = k$, dice que el proceso $p(i)$ tiene k instancias del recurso $r(j)$.
- **Necesidad** : Matriz de $N \times M$. Define el resto de necesidad de recursos de cada proceso.
 $\text{Necesidad}(i,j) = k$ significa que el proceso $p(i)$ necesita k más del recurso $r(j)$.
- **Requerimiento** : Es el vector de requerimientos de $p(i)$.
 $\text{Requerimiento}(i)(j) = k$, indica que $p(i)$ requiere k instancias del recurso $r(j)$.

Banquero y Seguridad

De lo cual se desprende que:

$$\text{Necesidad}(i,j) = \text{MAX}(i,j) - \text{Asignación}(i,j)$$

Para simplificar utilizaremos la notación siguiente:

Si X e Y son dos vectores:

$$\begin{array}{ll} X \leq Y & \text{sii} \quad X(i) \leq Y(i) \quad \forall i \\ X < Y & \text{sii} \quad X \leq Y, \text{ y } X \neq Y. \end{array} \quad y$$

Además trataremos aparte las matrices por sus filas, por ejemplo:

$\text{Asignación}(i)$ define todos los recursos asignados por el proceso $p(i)$.

$\text{Requerimiento}(i)$ es el vector de requerimientos de $p(i)$.

Banquero y Seguridad - ALGORITMO

Cuando se requiere un recurso se realizan primero estas verificaciones:

1. Si $\text{Requerimiento}(i) \leq \text{Necesidad}(i)$ seguir, sino ERROR (se pide más que lo declarado en $\text{MAX}(i)$).
2. Si $\text{Requerimiento}(i) \leq \text{Disponible}$ seguir, si no debe esperar, pues el recurso no está disponible
3. Luego el sistema pretende adjudicar los recursos a $p(i)$, para lo cual realiza una simulación modificando los estados de la siguiente forma:

Disponible	= Disponible - Requerimiento(i)
Asignación(i)	= Asignación(i) + Requerimiento(i)
Necesidad(i)	= Necesidad(i) - Requerimiento(i)

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 anterior estado de asignación total de recursos. Para verificarlo usa el **ALGORITMO DE SEGURIDAD**

ALGORITMO DE SEGURIDAD

Paso 1. Definimos **Work = Disponible** (de M elementos) y **Finish = F** (de Falso) para todo i con $i=1,2,\dots,n$ (Procesos). **Finish(i)** va marcando cuáles son los procesos ya controlados (si están en V) y con **Work** vamos acumulando a los recursos libres que quedan a medida que terminan.

Paso 2. Buscamos la **punta de la secuencia** de 'SAFE' y los subsiguientes. Encontrar el i tal que:

- a) $\text{Finish}(i) = F$ y
 - b) $\text{Necesidad}(i) \leq \text{Work}$
- si no existe ese i , **ir a Paso 4.**

Paso 3. $\text{Work} = \text{Work} + \text{asignación}(i)$
 $\text{Finish}(i) = V$ (por Verdadero);
ir a Paso 2.

Paso 4. Si $\text{Finish}(i) = V$ para todo i, el sistema está en estado "SAFE", o sea, encontramos la secuencia de procesos.

Requiere $M \times N \times N$ operaciones.

ALGORITMO DEL BANQUERO (ejemplo)

Sea el conjunto de procesos $\{p(0), p(1), p(2), p(3), p(4)\}$ y 3 recursos $\{A, B, C\}$. A tiene 10 instancias, B tiene 5 instancias y C tiene 7 instancias.

				Necesidad = MAX - Asig			
Asignación	A	B	C	MAX	A	B	C
p(0)	0	1	0		7	5	3
p(1)	2	0	0		3	2	2
p(2)	3	0	2		9	0	2
p(3)	2	1	1		2	2	2
p(4)	0	0	2		4	3	3
Disponibles	A	B	C				
	3	3	2				

Requerimiento de p(1)= $\langle 1, 0, 2 \rangle$

ALGORITMO DEL BANQUERO (ejemplo)

Para decidir que puedo garantizar la satisfacción de este requerimiento:

a) Veo que $\text{Req}(i) \leq \text{Disponible}$ $((1,0,2) \leq (3,2,2))$

b) Veo que $\text{Req}(i) \leq \text{Necesidad}$ $((1\ 0\ 2) \leq (1\ 2\ 2))$

c) Cumplimos el requerimiento, con lo cual se altera la información reflejando el siguiente estado:

	ASIG	NEC	NUEVO DISP.
p(0)	0 1 0	7 4 3	2 3 0
p(1)	3 0 2	0 2 0	
p(2)	3 0 2	6 0 0	
p(3)	2 1 1	0 1 1	
p(4)	0 0 2	4 3 1	

Veamos ahora el algoritmo de Seguridad

ALGORITMO DEL BANQUERO

(ejemplo del algoritmo de seguridad)

Paso 1 $WORK = \langle 2\ 3\ 0 \rangle$ y $Finish(i) = F$ para todo i

Paso 2 Existe i ? Tomemos **P1** porque $\langle 0\ 2\ 0 \rangle \leq \langle 2\ 3\ 0 \rangle$ y $Finish(1)=F$

Paso 3 $WORK = WORK + ASIG\ P(1) = \langle 2\ 3\ 0 \rangle + \langle 3\ 0\ 2 \rangle = \langle 5\ 3\ 2 \rangle$
 $Finish(1) = V$

Paso 2 Existe i ? Tomemos **P3** porque $\langle 0\ 1\ 1 \rangle \leq \langle 5\ 3\ 2 \rangle$ y $Finish(3)=F$

Paso 3 $WORK = WORK + ASIG\ P(3) = \langle 5\ 3\ 2 \rangle + \langle 2\ 1\ 1 \rangle = \langle 7\ 4\ 3 \rangle$
 $Finish(3) = V$

Paso 2 Existe i ? Tomemos **P4** porque $\langle 4\ 3\ 1 \rangle \leq \langle 7\ 4\ 3 \rangle$ y $Finish(4)=F$

Paso 3 $WORK = WORK + ASIG\ P(4) = \langle 7\ 4\ 3 \rangle + \langle 0\ 0\ 2 \rangle = \langle 7\ 4\ 5 \rangle$
 $Finish(4) = V$

.... Y así siguiendo obtenemos la secuencia segura $\{ p1, p3, p4, p0, p2 \}$

ALGORITMO DEL BANQUERO -

Algunas consideraciones

El algoritmo del Banquero impone fuertes restricciones al sistema ya que solo permite asignar recursos si el estado resultante es seguro.

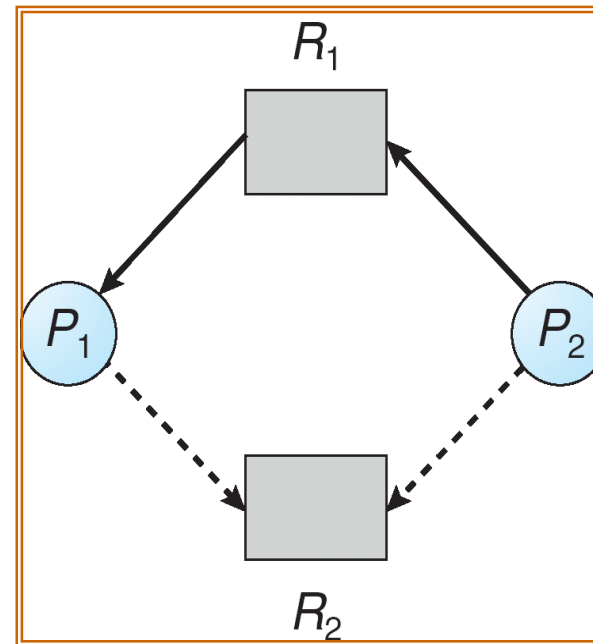
Podrían asignarse recursos pasando por estados inseguros y finalmente no llegar a entrar en deadlock pero el Banquero no lo permite.

Además el algoritmo requiere mucha información previa que puede no estar disponible. Cómo se puede estar seguro que el proceso efectivamente requerirá lo declarado en MAX?

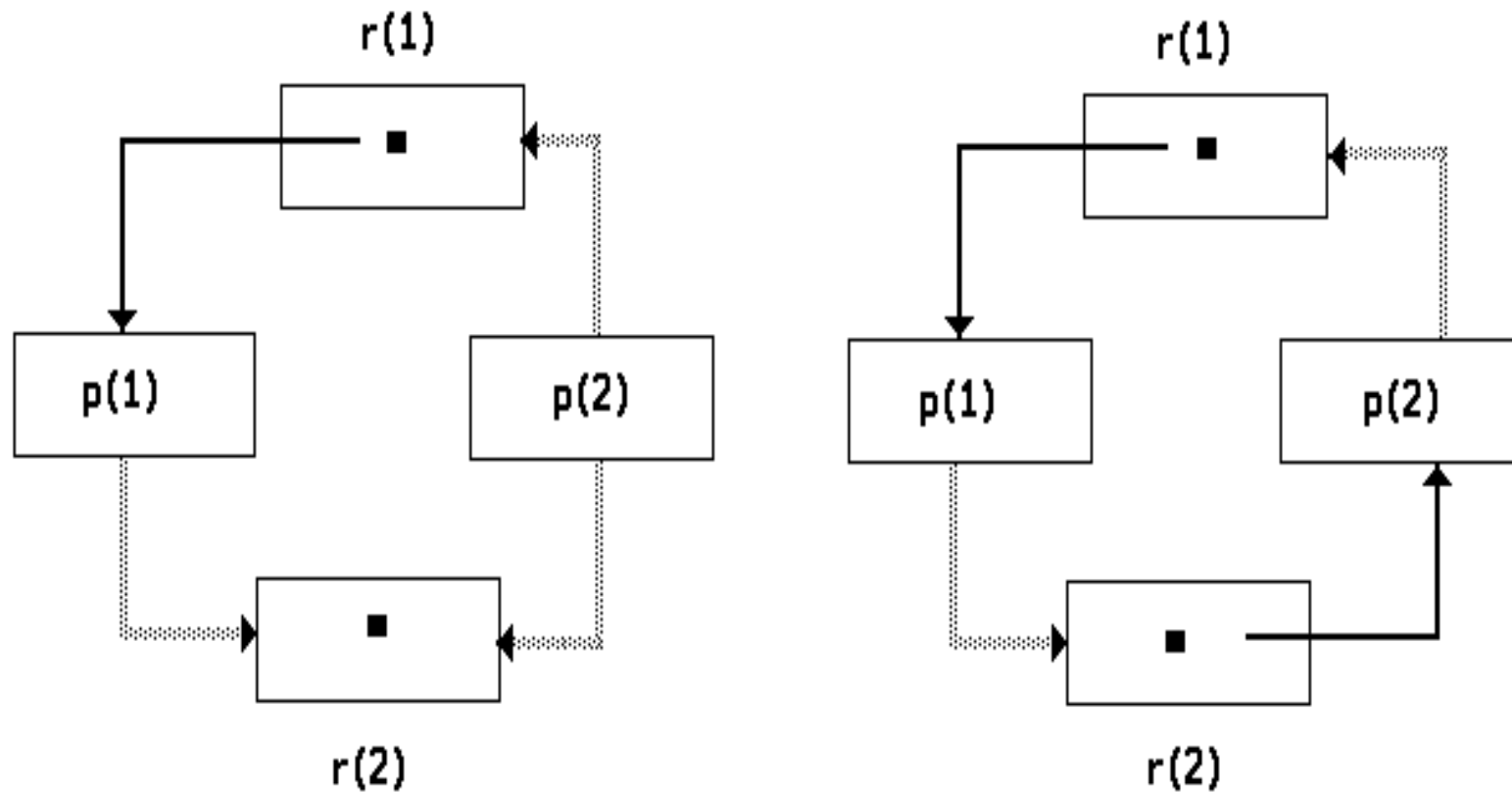
Algoritmo para recursos de una sola instancia

Cuando se tienen recursos de una sola instancia basta con utilizar un grafo de asignación de recursos para determinar si el sistema se encuentra o no en estado seguro.

En estos grafos se denota con una línea punteada el posible requerimiento de un recurso por parte de un proceso.

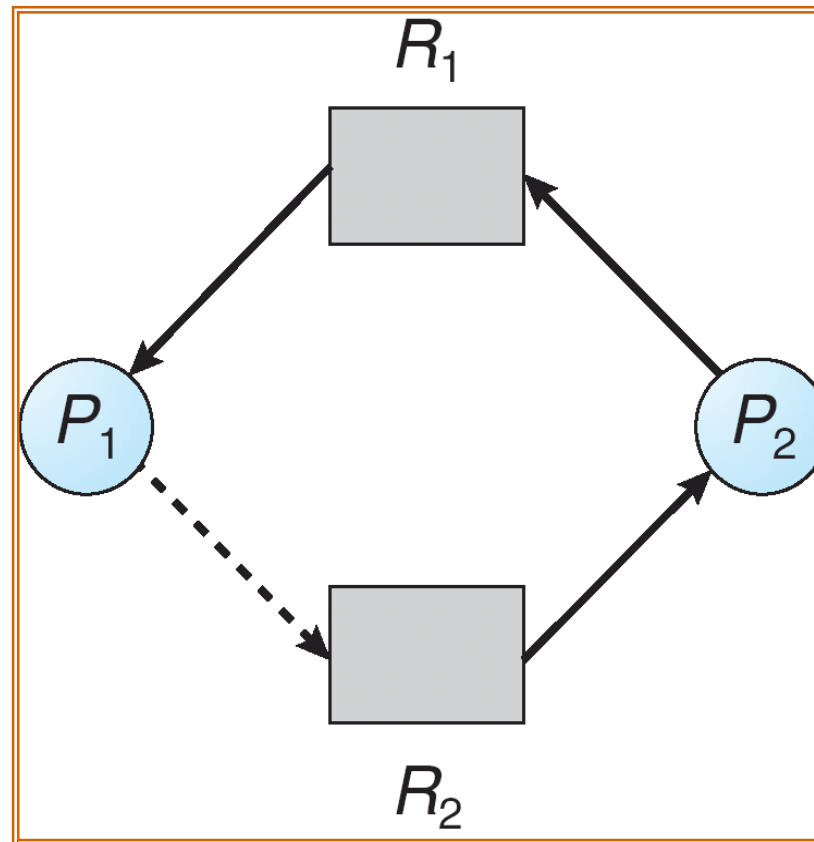


Algoritmo para recursos de una sola instancia



Algoritmo para recursos de una sola instancia

Ejemplo de un estado Inseguro en un grafo de asignación de recursos.



DETECCION DE DEADLOCK

ALGORITMO DE SHOSHANI Y COFFMAN (1970)

- **Disponible** : Vector de longitud M. $\text{Disponible}(j) = k$, indica que existen k instancias disponibles del recurso $r(j)$.
- **Asignación** : Matriz de $N \times M$. Define el número de recursos de cada tipo actualmente tomados por cada proceso.
 $\text{Asignación}(i,j) = k$, dice que el proceso $p(i)$ tiene k instancias del recurso $r(j)$.
- **Requerimiento o Espera**: Matiz de $N \times M$. Define los recursos requeridos por cada proceso en un instante, es decir, aquellos recursos por los cuales el proceso se encuentra en espera de disponibilidad.

DETECCION DE DEADLOCK

ALGORITMO DE SHOSHANI Y COFFMAN (1970)

- P1.** Work = Disponible
Si Asignación(i) $\neq 0 \Rightarrow$ Finish(i) = Falso sino
Finish(i) = Verdadero
- P2.** Encontrar un i tal que
a) Finish(i) = F
b) Requerimiento(i) \leq Work sino ir a **Paso P4.**
- P3.** Work = Work + Asignación(i)
Finish(i) = V ir a **Paso P2.**
- P4.** Si Finish(i) = F entonces el sistema está en DEADLOCK.

Más aún, los p(j) cuyo Finish(j) = F son los procesos involucrados en el deadlock

RECUPERACION FRENTE 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.

Para eliminar el deadlock matando procesos pueden usarse dos métodos:

- Matar todos los procesos en estado de deadlock. Elimina el deadlock 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 reejecutar el algoritmo de detección para verificar si el deadlock efectivamente desapareció o no.

RECUPERACION FRENTE DEADLOCK

1) Selección de Víctimas

Habría que seleccionar aquellos procesos que rompen el ciclo y tienen mínimo costo. Para ello debemos tener en cuenta:

- Prioridad.
- Tiempo de proceso efectuado, faltante.
- Recursos a ser liberados (cantidad y calidad).
- Cuántos procesos quedan involucrados.
- Si por el tipo de dispositivo es posible volver atrás.

RECUPERACION FRENTE DEADLOCK

2) Rollback

- Volver todo el proceso hacia atrás (cancelarlo).
- Volver hacia atrás hasta un punto en el cual se haya guardado toda la información necesaria (CHECKPOINT)

RECUPERACION FRENTE DEADLOCK

3) Inanición (starvation)

Es un tipo especial de Deadlock en el cual los recursos no llegan nunca a adquirirse.

Ejemplo típico: Procesos largos en una administración del procesador de tipo Mas Corto Primero.

Si el sistema trabajase con prioridades, podría ser que las víctimas fuesen siempre las mismas, luego habría que llevar una cuenta de las veces que se le hizo **Rollback** y usarlo como factor de decisión.