Tema 5. Gestión de Interbloqueo

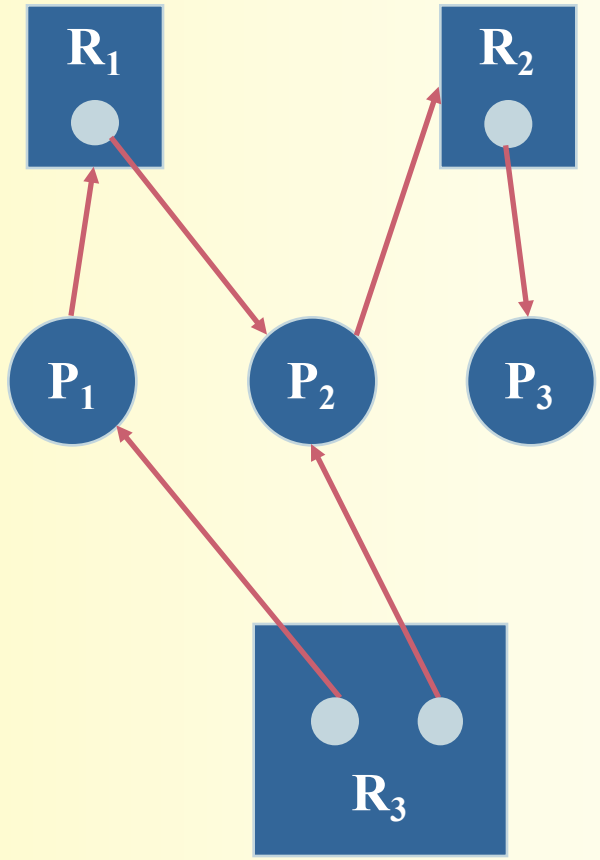
Introducción (I)

- Protocolo de acceso a recursos compartidos:
  - *Solicitud*.
  - Utilización.
  - *Liberación*.
- El sistema operativo suspende a los procesos cuyas *solicitudes* no pueden ser atendidas.
- Un conjunto de procesos está en *interbloqueo* si cada proceso está suspendido esperando por recursos retenidos por otros procesos del conjunto:
  - Los procesos no avanzan.
  - Los recursos no se utilizan.

Introducción (II)

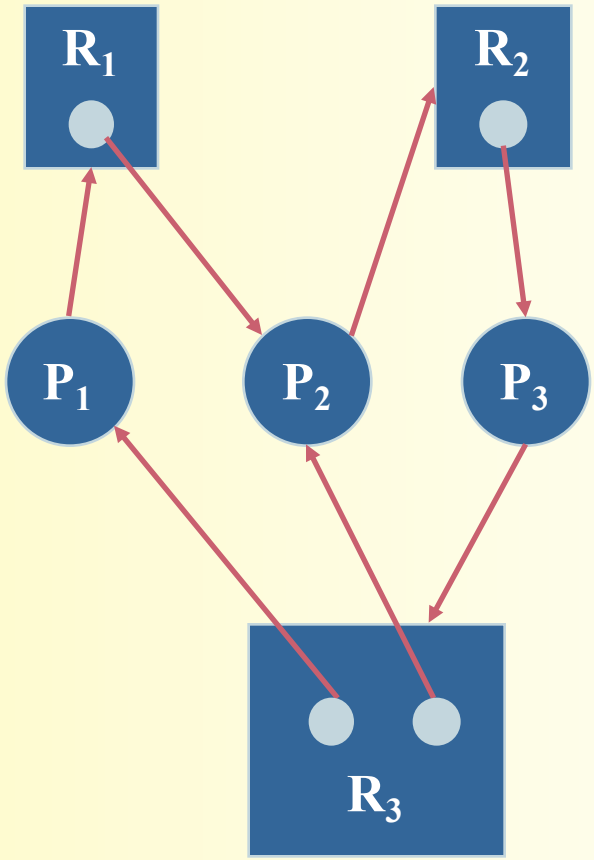
- El sistema operativo debe almacenar el *estado de asignación de recursos.*
- Representación mediante el *Grafo de Asignación de Recursos:*
  - $\{P_1, P_2, \ldots, P_n\}$ denota el conjunto de procesos (círculos).
  - $\{R_1, R_2, \ldots, R_m\}$ denota el conjunto de recursos (rectángulos), con uno o varios ejemplares (puntos).
  - *Asignación*: Pi ← Rj.
  - *Solicitud*: Pi → Rj.

Grafo de Asignación de Recursos (I)

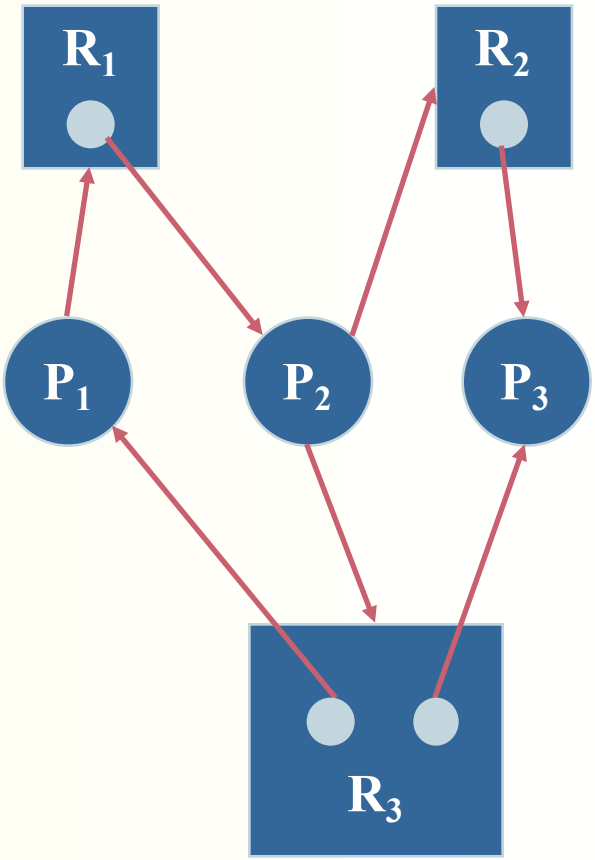


- Si no existe un ciclo en el grafo se puede asegurar que no existe interbloqueo.
- La existencia de ciclo es una condición necesaria pero no suficiente.

Grafo de Asignación de Recursos (II)



Ciclo e Interbloqueo



Ciclo y No Interbloqueo

Condiciones de Interbloqueo

- Cuatro condiciones necesarias, en conjunto suficientes:
  - *Exclusión mutua.*
  - *Retención y espera*.
  - *No expropiación*
  - *Espera circular.*

¿Cuál no se cumplía en el ejemplo anterior?

Estrategias de Gestión de Interbloqueo

1. Garantizar que nunca se llega a situación de interbloqueo.
   - *Prevención*: negación explícita de una de las condiciones necesarias.
   - *Evitación*: denegar solicitudes aún con recursos disponibles (basadas en información adicional).
2. Conceder todas las solicitudes mientras haya recursos suficientes, y tomar medidas si se llega a interbloqueo.
   - *Detección y Recuperación*.
3. No hacer nada.
   - Muchas veces, el interbloqueo afecta a procesos que forman parte de una misma aplicación.
   - La responsabilidad se deja en manos del programador.

Prevención de Interbloqueo (I)

- Se basa en negar explícitamente alguna de las condiciones necesarias.
  - Negar la *exclusión mutua*.
    - Imposible en cierto tipo de recursos (e.g. impresoras).
  - Negar la *retención y espera*.
    - Los procesos solicitan todos los recursos al inicio de su ejecución (no siempre posible).
    - Basta con permitir solicitudes si los procesos no tienen ningún recurso asignado.
    - Desventajas:
      - Infrautilización de recursos.
      - Puede causar inanición.

Prevención de Interbloqueo (II)

- Negar la *no expropiación*.
  - Un proceso que tiene que esperar pierde los recursos que le fueron asignados, y pasa a esperar también por ellos.
  - Basta con demorar la expropiación hasta que algún otro proceso necesite dichos recursos.
  - Desventajas:
    - Sólo para recursos que pueden recuperar su estado (por ejemplo, memoria o CPU).
    - Puede causar inanición.

Prevención de Interbloqueo (III)

- Negar la *espera circular.*
  - Establecer un orden en los recursos y exigir que se soliciten según dicho orden.
  - Basta con exigir que un proceso libere los ejemplares que tenga asignados de R_j antes de solicitar ejemplares de R_i ($i < j$, y orden creciente).
  - Desventajas:
    - Infrautilización de recursos.
    - ¿Puede causar inanición?

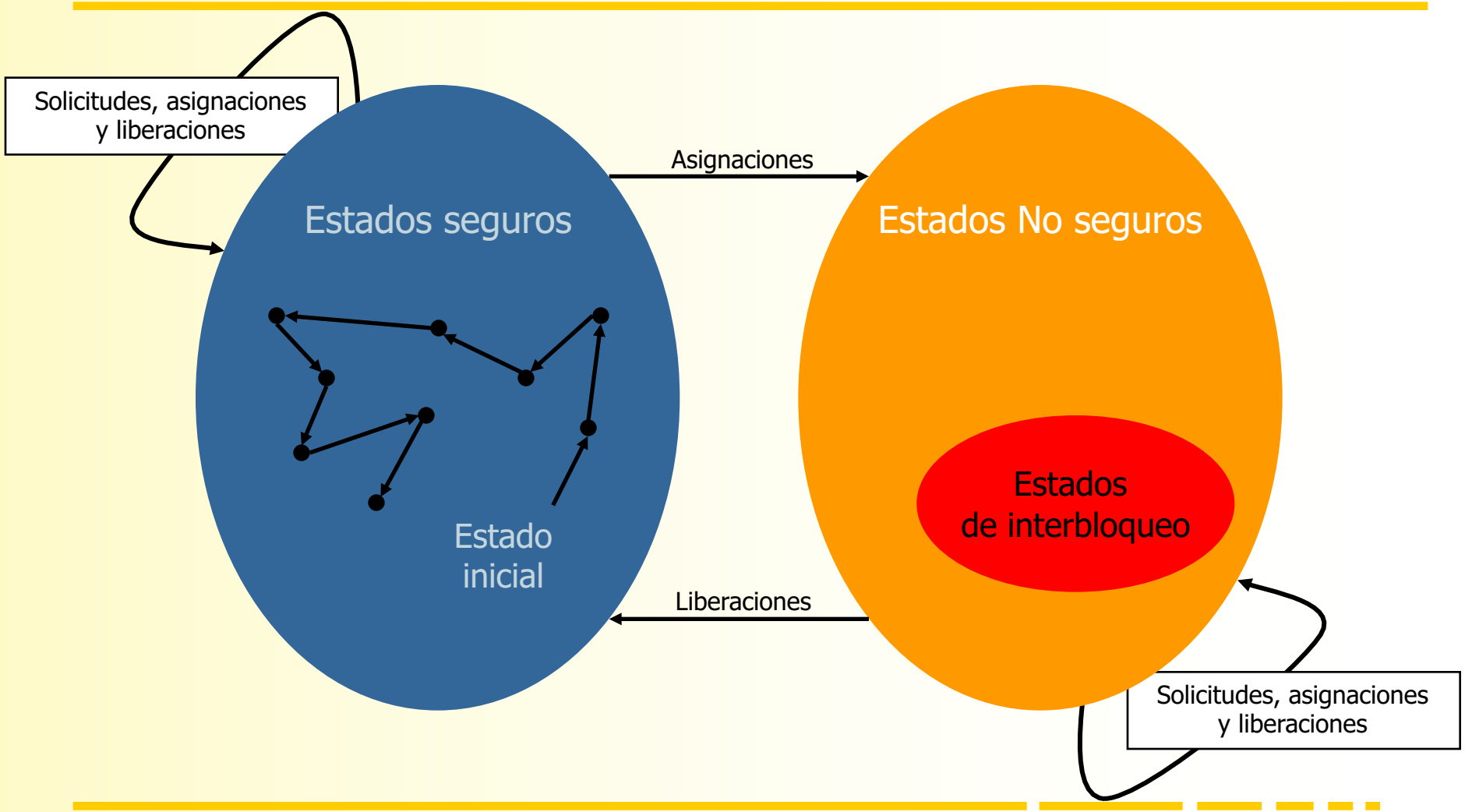
Evitación de Interbloqueo (I)

- Se basa en denegar solicitudes que se podrían conceder, evitando que se llegue a interbloqueo.
- Desventaja:
  - Infrautilización de recursos.
  - Necesitan información adicional para decidir qué solicitudes deben ser denegadas.
- Método basado en el conocimiento de las necesidades máximas de cada proceso:
  - Algoritmo del Banquero.
  - Evita la *espera circular* y/o *retención y espera.*

Evitación de Interbloqueo (II). Estados Seguros

- *Estado seguro:*
  - Estado de asignación de recursos donde *solicitudes* no pueden hacer que se pase a un estado con interbloqueo (análisis del caso peor);
  - o, estado donde es posible encontrar un orden para satisfacer a todos los procesos hasta sus necesidades máximas;
  - o, estado donde existe una *secuencia segura:*
    - Una secuencia de procesos (P_1, …, P_n) es una *secuencia segura* si las necesidades máximas de P_j en ese momento pueden satisfacerse con los recursos libres más los que tienen asignados los procesos p_i, con i<j.
- *Estado No seguro:*
  - No conlleva Interbloqueo.
  - Si no lo hay, sólo depende de los procesos que se llegue a Interbloqueo

Evitación de Interbloqueo (III). Estados Seguros



Evitación de Interbloqueo (IV). Estados Seguros

- Para cada *solicitud*:
  - ¿Existen recursos libres para atenderla?
    - NO:
      - El proceso espera.
    - SÍ:
      - Simular que se atiende (*asignación* de recursos).
      - ¿El estado resultante es *seguro*?
        - NO: el proceso espera.
        - SÍ: se atiende la *solicitud*.
- Se necesita un algoritmo para determinar si un estado es *seguro* (*algoritmo de seguridad*).

Evitación de Interbloqueo (V). Estados Seguros

- Un único recurso con 12 ejemplares.
- 3 procesos:

	Necesidad máxima	Asignación actual
P_1	10	5
P_2	4	2
P_3	9	2

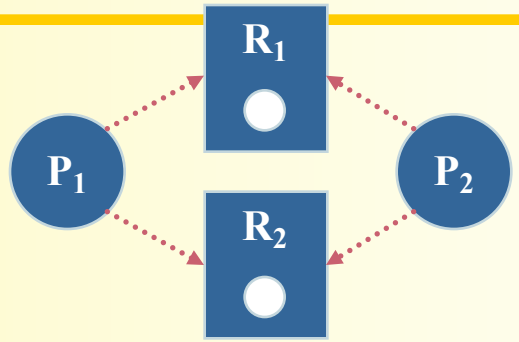
- El sistema se encuentra en un estado seguro.
  - Secuencia segura: (P_2,P_1,P_3).

¿y si P_3 solicita un nuevo ejemplar?

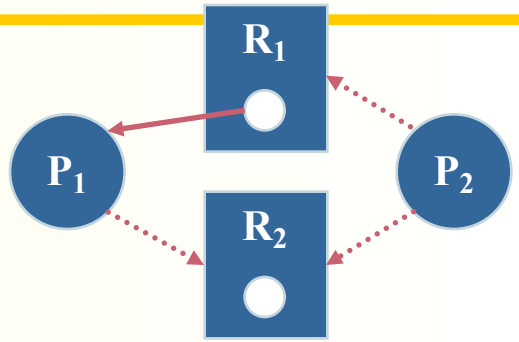
Evitación de Interbloqueo (VI). Algoritmos de Seguridad

- Si sólo existe un ejemplar de cada recurso ($\{P_1, \ldots, P_n\}, R_1$) :
  - Se añaden *flechas de reserva* al grafo de asignación de recursos.
  - Simulación de *solicitud* atendida: se cambian las *flechas de reserva* por *flechas de asignación.*
  - Algoritmo de seguridad: algoritmo de detección de ciclos
  - Complejidad (n^2).
- En caso contrario ($\{P_1, \ldots, P_n\}, \{R_1, \ldots, R_m\}$) :
  - *Algoritmo del Banquero.*
  - El algoritmo busca una *secuencia segura* sobre una representación matricial del estado de asignación de recursos y las necesidades máximas de cada proceso.
  - Complejidad ($m \cdot n^2$).

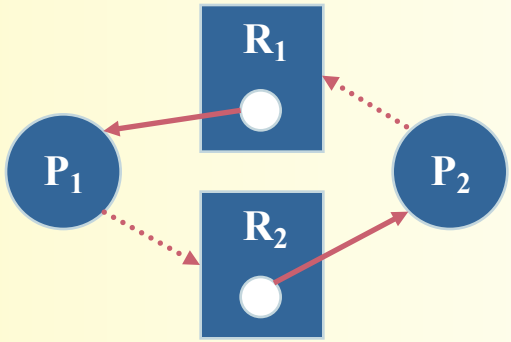
Evitación de Interbloqueo (VII). Algoritmos de Seguridad



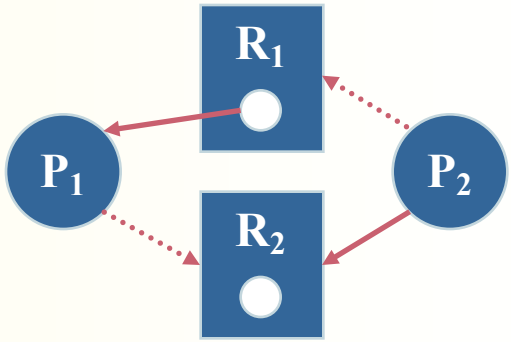
P_1 y P_2 pueden solicitar R_1 y R_2



P_1 solicita R_1
No aparecen ciclos → R_1 se asigna a P_1



P_2 solicita R_2
Aparece un ciclo → P_2 espera



P_2 espera
Se le asignará R_2 cuando P_1 libere R_1

Evitación de Interbloqueo (VIII). Algoritmo del Banquero

- Nunca conceder efectivo disponible si hay riesgo de no poder satisfacer las necesidades de todos los clientes.
- Notación ($\{P_1, \ldots, P_n\}$, $\{R_1, \ldots, R_m\}$) :
  - *Disponible[j]* = ejemplares no asignados del recurso R_j.
  - *Asignación^k[j]* = ejemplares del recurso R_j que tiene asignados el proceso P_k.
  - *Máxima^k[j]* = necesidad máxima del recurso R_j que puede necesitar simultáneamente el proceso P_k.
  - *Necesidad^k[j]* = *Máxima^k[j]* - *Asignación^k[j]*
  - *Fin[k]* = indica si el proceso P_k se ha incluido en la secuencia segura. Inicialmente *Fin* = *Falso*.
  - *Mi_Disponible[j]* = acumula *Disponible[j]* + *Asignación^k[j]* de todos los procesos P_k encontrados de la secuencia segura. Inicialmente *Mi_Disponible* = *Disponible.*
  - *Solicitud^k[j]* = ejemplares del recurso R_j que solicita el proceso P_k.

Evitación de Interbloqueo (IX). Algoritmo del Banquero

- Simulación de *solicitud* atendida ($Solicitud^k$):
  - $Disponible = Disponible - Solicitud^k$
  - $Asignación^k = Asignación^k + Solicitud^k$
  - $Necesidad^k = Necesidad^k - Solicitud^k$
- Algoritmo del Banquero (algoritmo de seguridad):
  - Repeat:
    - Buscar k tal que $Fin[k] = falso$ y $Necesidad^k \leq Mi\_Disponible$
    - Si existe:
      - $Fin[k] = verdadero$
      - $Mi\_Disponible = Mi\_Disponible + Asignación^k$
    - En caso contrario, salir del bucle.
  - Si existe k tal que $Fin[k] = falso$, el estado es inseguro (deshacer la simulación).
  - En caso contrario, es seguro.

Detección de Interbloqueo

- Conceder todas las *solicitudes* mientras haya recursos para atenderlas, y tomar medidas si se detecta interbloqueo.
  - No se restringen escenarios sin riesgo de interbloqueo.
  - No hay infrautilización de recursos.
- Para la detección se usan los mismos algoritmos de seguridad que en la evitación, teniendo en cuenta las *solicitudes* reales (no el caso peor).
- ¿Cuándo ejecutar el algoritmo de detección?
  - Con cada *solicitud* que no se puede satisfacer inmediatamente.
    - La detección más rápida.
    - Permite identificar al proceso que causa el interbloqueo.
    - Puede suponer un coste excesivo.
  - Periódicamente.
    - Dificultad de elegir el periodo más adecuado.
  - Cuando se detecte una degradación de las prestaciones del sistema (factor de utilización de la CPU bajo).

Recuperación de Interbloqueo

- Abortar procesos, dejando libres los recursos que retienen.
  - Todos los procesos que participan en el interbloqueo (drástico).
  - Un proceso tras otro, comprobando si se resuelve el problema.
    - Costoso.
    - Selección en base a prioridades, número y tipo de recursos consumidos, etc.
- Expropiar recursos:
  - Selección de procesos víctima (como arriba).
  - Selección de recursos (no todos se pueden expropiar).
  - Retroceso: devolver los procesos a un punto anterior de su ejecución (lo más drástico sería reiniciar el proceso).
  - Hay que considerar los procesos afectados en el pasado para no causar inanición.