Sistemas Operativos

Deadlocks
(Interbloqueos)

Sistemas Operativos

- ☑ **Versión: Junio 2025**
- ☑ **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)

Definición

- Un conjunto de procesos están en **deadlock** cuando cada uno de ellos esta **esperando por un recurso** que esta siendo **usado por otro proceso** del mismo conjunto
- Un estado de Deadlock puede involucrar recursos de diferentes tipos.

Ejemplos

Con Procesos:

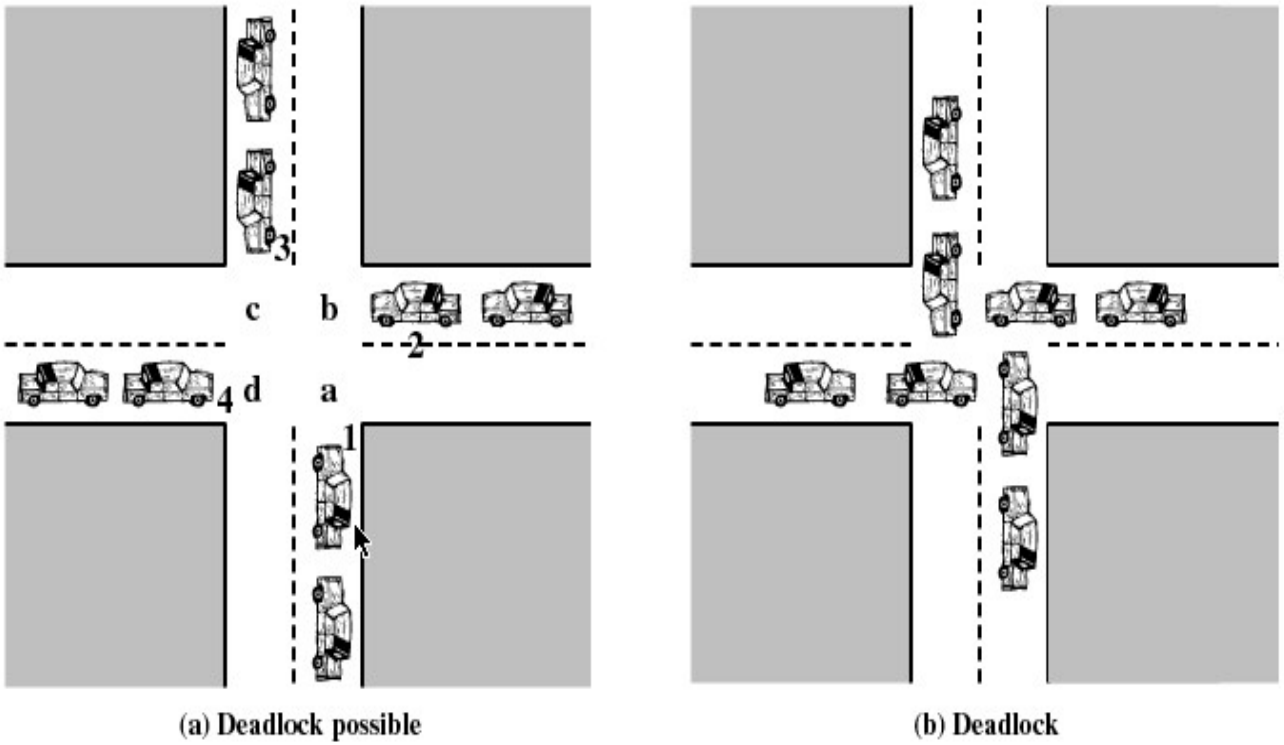
- ☑ Un proceso A pide un scanner.
- ☑ Un proceso B pide una grabadora de CD.
- ☑ El proceso A pide ahora la grabadora de CD
- ☑ El proceso B quiere el scanner.

En una BD:

- ☑ Un proceso A bloquea el registro R1,
- ☑ Un proceso B bloquea el registro R2.
- ☑ Luego cada proceso trata de bloquear el registro que está usando el otro.

Ejemplos

Otro Ejemplo:



(a) Deadlock possible

(b) Deadlock

Figure 6.1 Illustration of Deadlock

Ejemplos

Con recursos:

Process P		Process Q	
Step	**Action**	**Step**	**Action**
p_0	Request (D)	q_0	Request (T)
p_1	Lock (D)	q_1	Lock (T)
p_2	Request (T)	q_2	Request (D)
p_3	Lock (T)	q_3	Lock (D)
p_4	Perform function	q_4	Perform function
p_5	Unlock (D)	q_5	Unlock (T)
p_6	Unlock (T)	q_6	Unlock (D)

Recursos

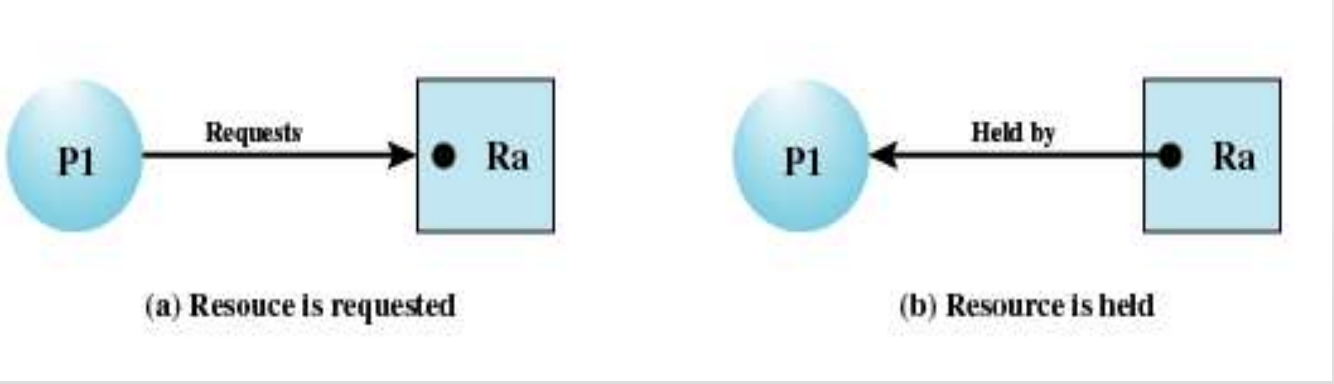
- Todo Sistema contiene recursos
- **Recursos físicos**
  - CPU, memoria, dispositivos.
- **Recursos lógicos**
  - archivos, registros, semáforos, etc.
- **Recursos apropiativos**: se le pueden quitar al proceso sin efectos dañinos (ej: memoria, CPU).
- **Recursos no apropiativos**: si se le saca al proceso, éste falla (interrumpir una escritura a CD, impresora).
- Cada recurso R_j puede tener W_i **instancias idénticas** (puede haber 2 impresoras del mismo tipo)
  - Si son idénticas, se puede asignar cualquier instancia del recurso
- **Clase de Recursos**: es el conjunto de instancias de un recurso

Recursos (cont.)

- Siguiendo un modo de operación normal, un proceso emplea un recurso siguiendo la siguiente secuencia:
  1. **Solicitud**: Si no puede ser concedida inmediatamente, el proceso deberá esperar a que el recurso sea liberado
  2. **Uso**: El proceso puede operar sobre el recurso
  3. **Liberación**: Se libera el recurso para que pueda ser utilizado por otro proceso
- Si el recurso que se quiere utilizar está ocupado, se sigue un **"Ciclo corto de solicitud" → 1. Solicitud fallida, 2. Espera inactiva, 3. Nuevo intento de solicitud**.

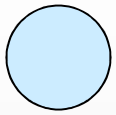
Representación de procesos y recursos

- ☑ Para poder representar la asignación de recursos, se utiliza un **Grafo de Asignación de recursos**.
- ☑ El grafo **permite visualizar el estado** de los recursos del sistema y procesos en un momento determinado.
- ☑ Cada **Proceso o recurso es representado por un nodo**. Un recurso con varias **instancias** posee mas de un "Punto" dentro de nodo.
- ☑ Una **arista representa una relación entre un Proceso y un Recurso**. Notar que las aristas **son dirigidas** y dependiendo de la dirección indican distintos estados

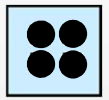


Representación de procesos y recursos

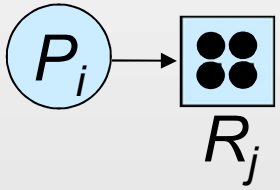
Proceso



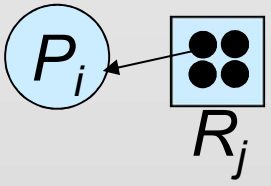
Tipo de Recurso con 4 instancias



P_i solicita una instancia de R_j



P_i tiene asignada una instancia de R_j



Ejemplo – Varias Instancias

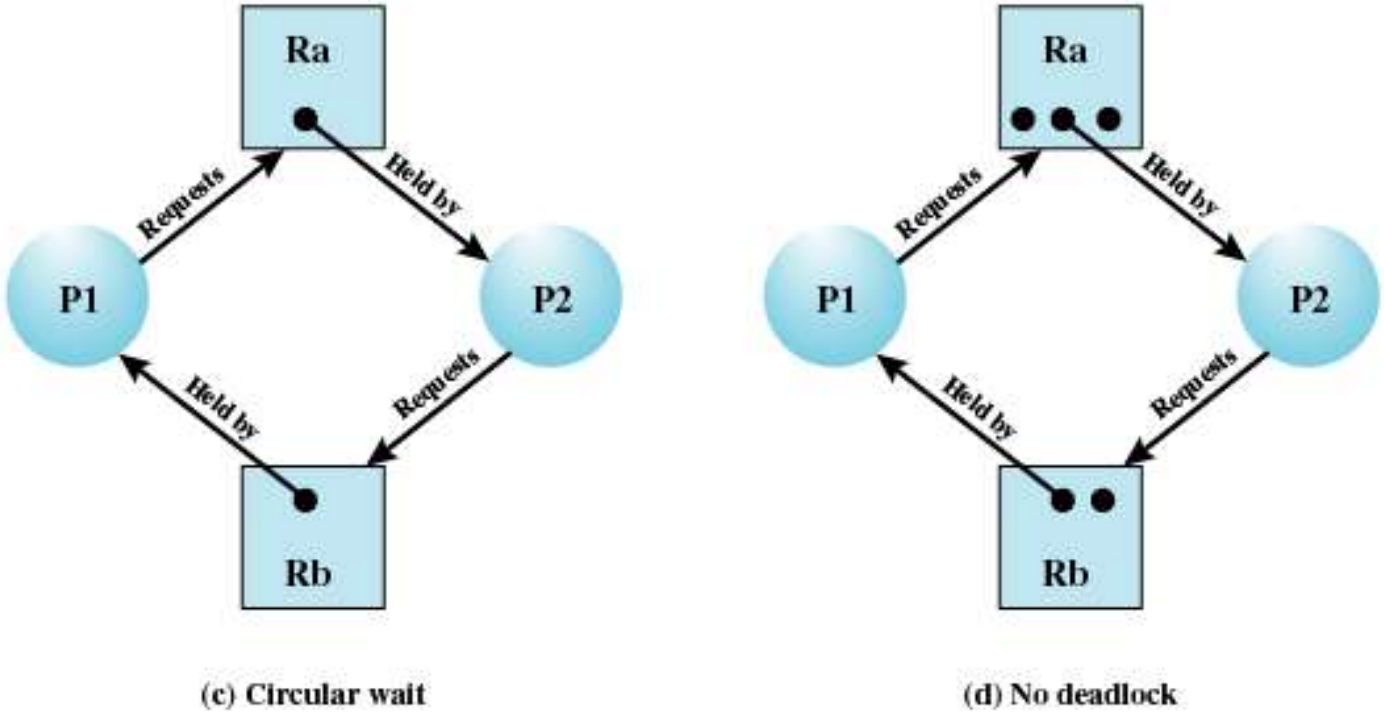
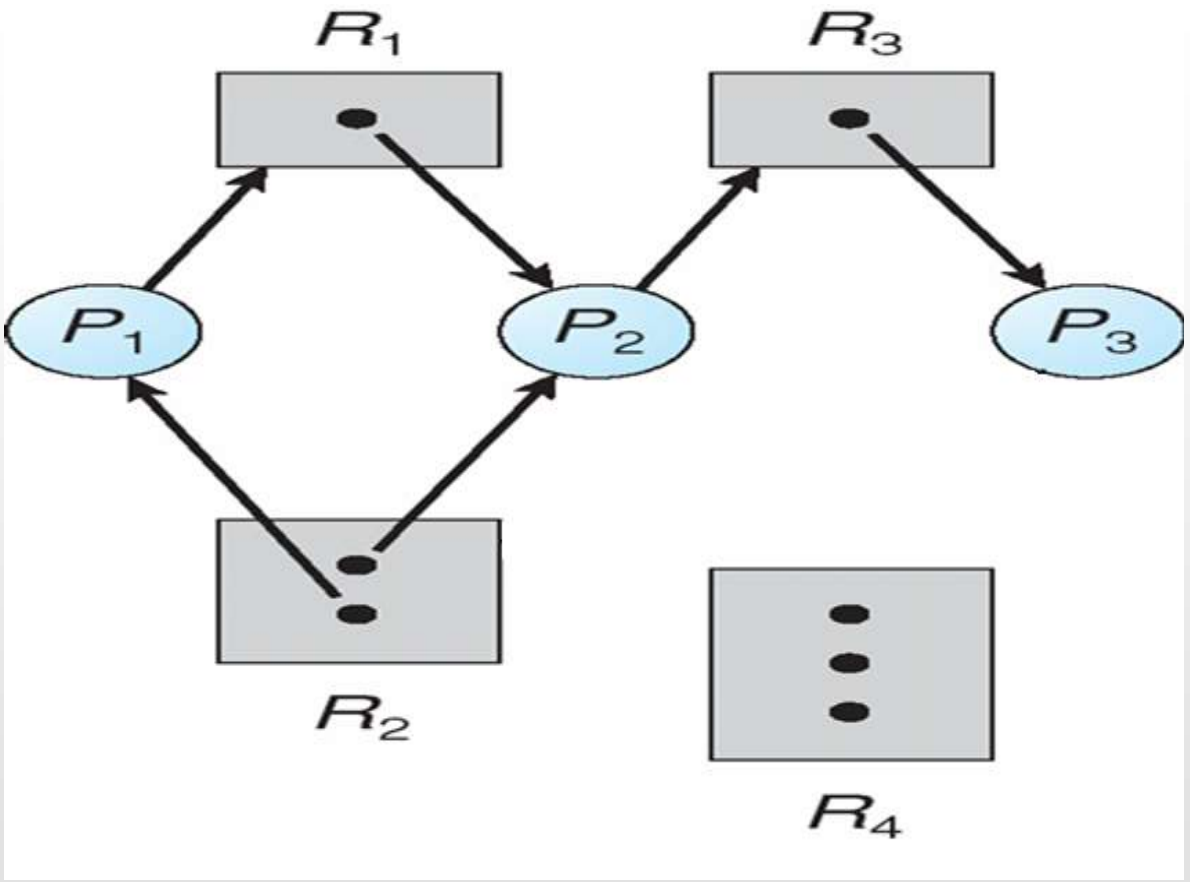


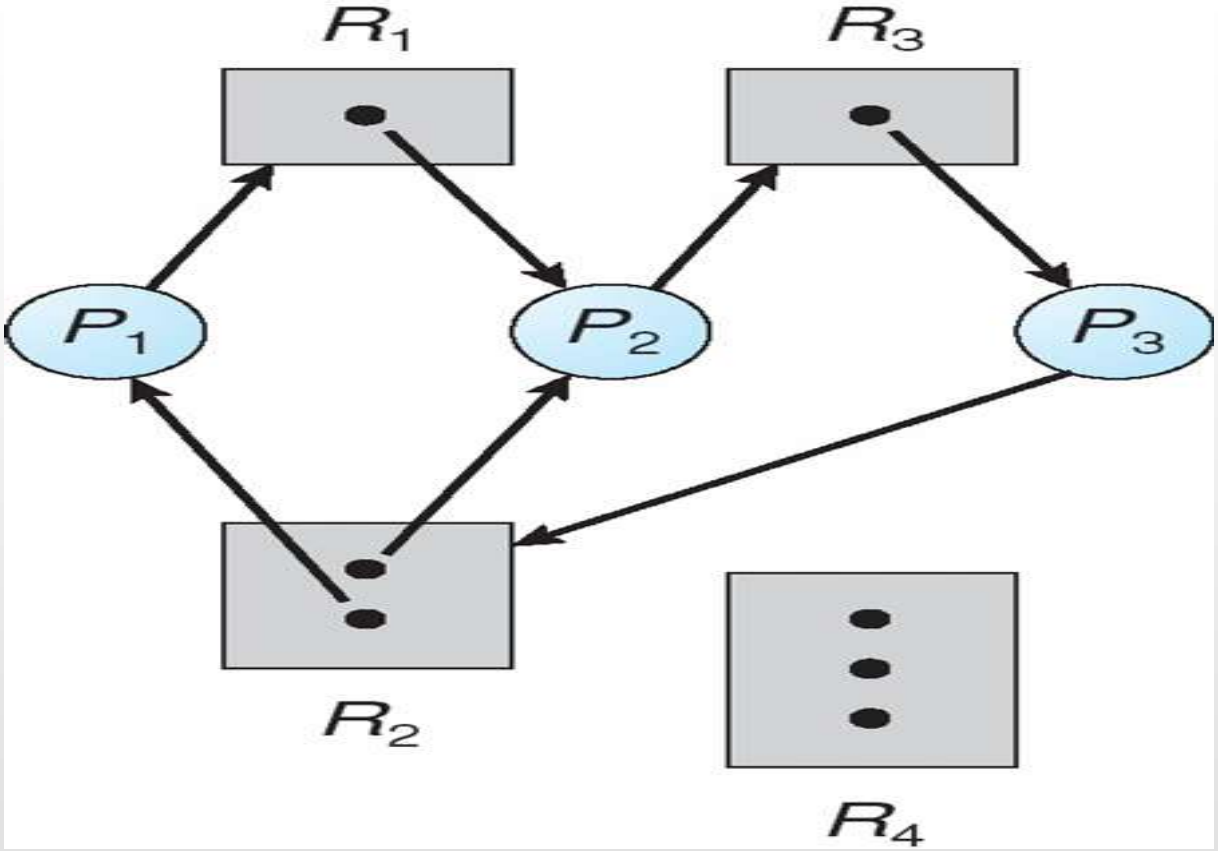
Figure 6.5 Examples of Resource Allocation Graphs

Ejemplo – Alocación de recursos – Varias Instancias



¿Hay Ciclos/Bucles?

Ejemplo – Alocación de recursos - Varias Instancias



¿Hay Ciclos/Bucles?

Hechos Basicos

- Si el grafo no contiene ciclos → NO hay interbloqueo
- Si el grafo contiene un ciclo:
  - Si sólo hay una instancia por tipo de recurso → SI hay interbloqueo
  - Si hay varias instancias por tipo de recurso → hay posibilidad de deadlock.

HAY QUE ELIMINAR EL DEADLOCK!!

Condiciones para que se cumpla deadlock

- ☑ **Coffman, 1971**
- ☑ **Condiciones:**
  1. **Exclusión mutua**: En un instante de tiempo dado, solo un proceso puede utilizar una instancia de un recurso
  2. **Retención y espera**: Los procesos deben mantener los recursos asignados y esperar por la asignación de los nuevos requeridos
  3. **No apropiación**: Los recursos no pueden ser quitados a un proceso que actualmente los posea
  4. **Espera circular**: El proceso forma parte de una lista circular en la que cada proceso de la lista está esperando por al menos un recurso asignado a otro proceso de la lista
- ☑ **Para estar en presencia de un Deadlock se deben cumplir todas!**

https://dl.acm.org/doi/abs/10.1145/356586.356588?casa_token=WZOrjUa0qDwAAAAA:phVv40v2uU4z1lmR3fNWeV-1odpTzJSk5wTHyamhrWerKFLYZGkPsCgYlCY4mRX2G7bzwcCyfEsDHA

Métodos para el tratamiento del deadlock

- ☑ **Existen distintos enfoques con el fin de llevar adelante el manejo de Deadlocks:**

1. **Usar un protocolo que asegure que NUNCA se entrará en estado de deadlock**
   - 1.1 **Prevenir:** Que no se cumple alguna de las 4 condiciones
   - 1.2 **Evitar:** Tomar decisiones de asignación en base al estado del sistema
2. **Permitir el estado de deadlock y luego recuperar**
3. **Ignorar el problema y esperar que nunca ocurra un deadlock**

"prevention" <> "avoidance"

- **Prevention (prevenir la formación del interbloqueo):**
  - Que por lo menos una de las condiciones no pueda mantenerse.
  - Se imponen restricciones en la forma en que los procesos REQUIEREN los recursos.
- **Avoidance (evitar la formación del interbloqueo):**
  - Asignar cuidadosamente los recursos, manteniendo información actualizada sobre requerimiento y uso de recursos.

Prevenir: 1. Condición de exclusión mutua

> ***Exclusión mutua**: En un instante de tiempo dado, solo un proceso puede utilizar una instancia de un recurso*

- ☑ Si **ningún recurso se asignara de manera exclusiva** (no siempre se puede), no habría interbloqueo.
- ☑ Considerar que hay recursos compartibles (archivos read only, memoria) y no compartibles (impresora).
- ☑ **Mantener la exclusión mutua para los no compartibles puede resultar complejo:**
  - ➢ Se pueden implementar esquemas de encolamiento y que el recurso sea manejado por un proceso global (spooler). De esta manera no se bloquea el recurso no compartible ← el spooler podría llenarse y bloquear procesos…
- ☑ Los recursos compartibles NO requieren mantener la exclusión mutua

Prevenir: 2. Condición de retención y espera

> ***Retención y espera**: Los procesos deben mantener los recursos asignados y esperar por la asignación de los nuevos requeridos*

- ☑ **Se basa en que si un proceso requiere un recurso que no está disponible, debe liberar otros.**
- ☑ **Alternativas:**
  1. Un proceso **debe requerir y reservar todos los recursos a usar antes de comenzar la ejecución** (precedencia de las system calls que hacen el requerimiento antes de cualquier otra system call)
  2. El proceso **puede requerir recursos sólo cuando no tiene ninguno** (al comienzo de su ejecución generalmente).
- ☑ **Desventajas**
  - ✓ **Baja utilización de recursos**
  - ✓ Posibilidad de **inanición** de alguno de los procesos (starvation, o espera infinita)

Prevenir: 3. Condición de no apropiación (cont.)

> ***No apropiación**: Los recursos no pueden ser quitados a un proceso que actualmente los posea*

- ☑ No siempre se puede atacar esta condición, ya que **no siempre puede expropiarse un recurso a un proceso**.
- ☑ Posibles soluciones:
  3. Virtualizar recursos:
     - ➢ El proceso no accede directamente al recurso, sino que accede a un demonio que lo administra.
     - ➢ Solo el demonio tiene acceso al recurso, y a medida que los procesos requieren el recurso, interactúan con el demonio quien almacena los trabajos en una cola (spooler)
     - ➢ De esta manera se logra que el recurso físico pueda ser "utilizado" por varios procesos al mismo tiempo (una forma ficticia de compartir el recurso, ya que en realidad solo 1 proceso a la vez lo está utilizando)

Prevenir: 4. Condición de Espera circular

> ***Espera circular:*** *El proceso forma parte de una lista circular en la que cada proceso de la lista está esperando por al menos un recurso asignado a otro proceso de la lista.*

- ☑ Se define un **ordenamiento de los recursos**. Luego, **un proceso puede requerir recursos en un orden numérico ascendente**.
- ☑ Sea **F:(R→N)**, N conjunto de los naturales.
- ☑ **La función F** asigna un numero único a cada recurso (los números pequeños para recursos muy usados).
- ☑ **Un proceso, que ya tiene R_i puede requerir R_j si y solo si** $F(R_i) < F(R_j)$ *(solicita recursos con numero mayor a los ya asignados)*
- ☑ Como no se puede dar que $F(R_j) > F(R_i)$ y $F(R_j) < F(R_i)$, **podemos garantizar que no habrá un bucle en el grafo de asignación de recursos**

Ejemplo: Prevención en Espera circular

- Supongamos que se han definido los siguientes valores:

  F(CD)=1; F(disco duro)=4, F(impresora)=7

- Un proceso que ya tiene asignado el disco, puede pedir la impresora (pues F(impresora)> F(disco duro).
- Si ya tiene la impresora, no puede solicitar el CD.

Evitar Deadlocks

- **Requiere que el SO tenga información ANTES sobre el uso de recursos**
- **El SO cuenta con información sobre el uso de los recursos**
  - Cómo son requeridos por parte de los procesos
  - En qué momento son requeridos
  - La demanda máxima, etc.
- La técnica se basa en **tomar decisiones acerca de la asignación de los recursos, con el fin de que no se llegue a un estado de Deadlock**.
- En todo momento el SO toma decisiones dinámicas acerca de la asignación. **Si en algún momento se evalúa que podría entrarse en un estado de Deadlock, la asignación de recursos es denegada**.
- Desventajas en implementación: **puede producir una baja utilización de los recursos** y de la performance del sistema debido a los cálculos que deben realizarse

Estado sano o seguro

- **Un sistema está en un estado seguro si el SO puede asignar recursos a cada proceso de un conjunto de alguna manera, evitando el deadlock**.
- Cuando un proceso solicita un recurso disponible, el sistema **DEBE DECIDIR** si la asignación inmediata deja el sistema en un estado seguro.
- **Debe haber una secuencia "cadena segura" de TODOS procesos <P0, P1 , , Pn >**, que puedan ejecutarse con *todos* los recursos disponibles sin que haya deadlock.

Decimos Estado Sano o Seguro

- El sistema está en estado seguro si existe secuencia <P1, P2, ..., Pn> (cadena segura) para TODOS los procesos del sistema
- Si existe una secuencia, significa que se pueden satisfacer los requerimientos, teniendo e cuenta lo actualmente asignado a los procesos en ejecución

Estado sano o seguro

- Si los **recursos necesarios de P_i no están disponibles inmediatamente, entonces P_i puede esperar** hasta que todos los P_j hayan terminado.
- Cuando P_j está terminado, P_i puede obtener los recursos necesarios, ejecutar, devolver los recursos asignados y finalizar.
- Cuando P_i termina, P_i +1 puede obtener sus recursos necesarios, y así sucesivamente

Si no se puede construir esta secuencia, el estado del sistema es inseguro

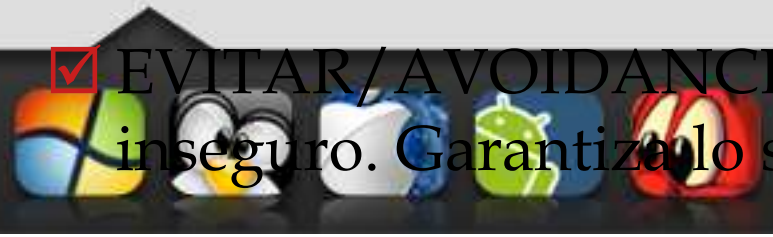
Importante!

- ☑ Un estado seguro garantiza que **NO** hay deadlock.

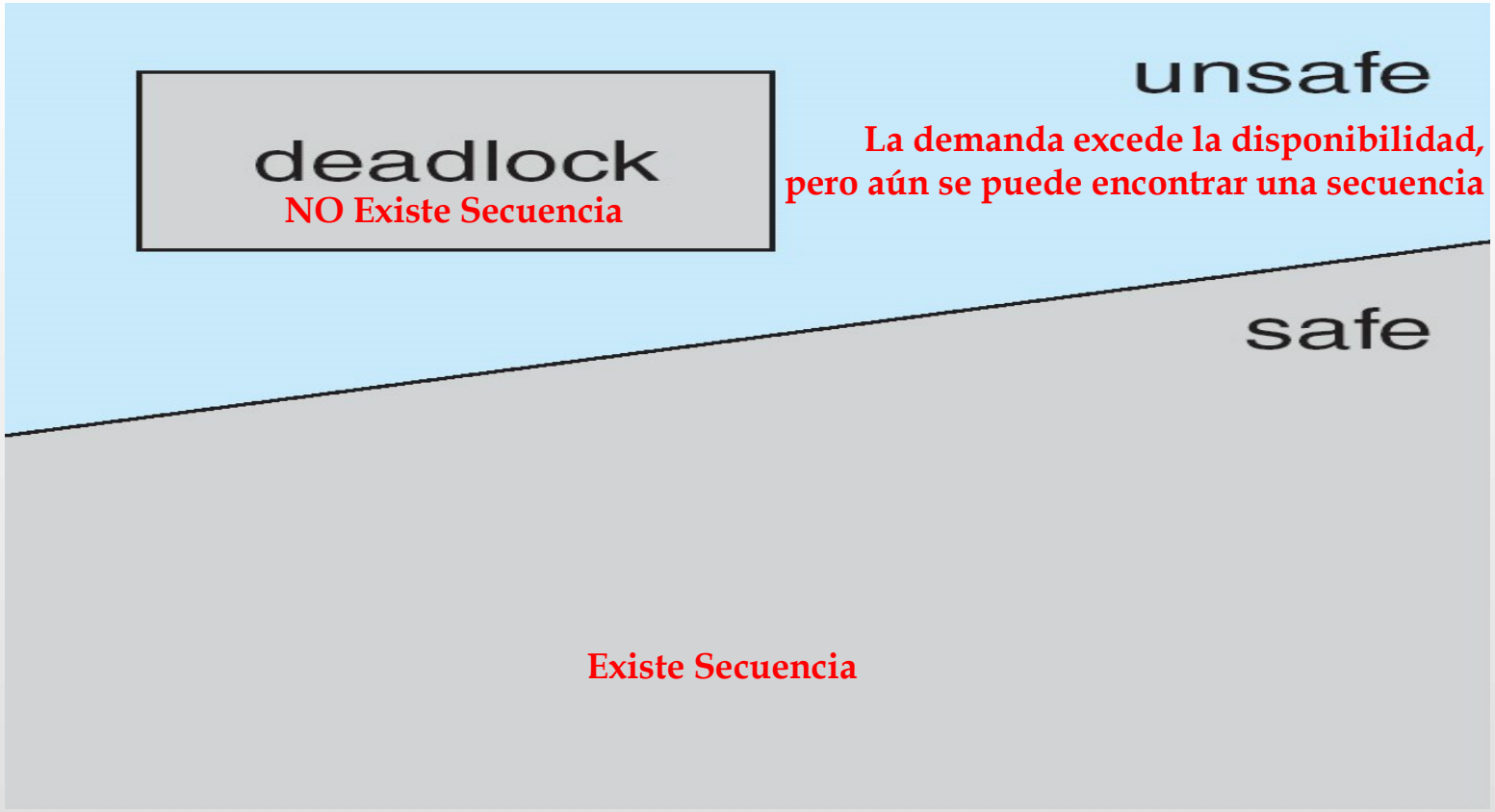
- ☑ En estado inseguro **hay posibilidad** que haya deadlock.
  - ➢ No todo estado inseguros es deadlock
  - ➢ Puede ocurrir que no exista cadena de asignación temporalmente, pero ante la terminación de un proceso, se liberen recursos y se pueda construir la cadena

- ☑ Si hay deadlock, estoy en estado inseguro.

- ☑ EVITAR/AVOIDANCE: trata de nunca entrar a estado inseguro. Garantiza lo seguro entonces no hay deadlock.



Estado seguro, inseguro y Deadlock



Ejemplo 1

- ☑ En un sistema existen 3 procesos **P0, P1 y P2 que** comparten **12** unidades de cinta.
- ☑ Según la tabla, hay 3 cintas libres.
- ☑ ¿En qué estado está el sistema?

Procesos	Máximo a usar	en uso
P_0	10	5
P_1	4	2
P_2	9	2

La demanda excede la disponibilidad (hay un estado inseguro), pero ¿podemos encontrar una secuencia de asignación de recursos?

Ejemplo 1 (cont.)

- En un sistema existen 3 procesos **P0, P1 y P2** que comparten **12** unidades de cinta.
- Según la tabla, hay 3 cintas libres.
- ¿En qué estado está el sistema?

Procesos	Máximo a usar	en uso
P_0	10	5
P_1	4	2
P_2	9	2

P1 → P0 → P2 , el sistema está en estado seguro

Ejemplo 2

- En un sistema existen 3 procesos **P0, P1 y P2** que comparten **12** unidades de cinta.
- Según la tabla, hay 3 cintas libres.
- ¿En qué estado está el sistema?

Procesos	Máximo a usar	en uso
P_0	12	5
P_1	4	2
P_2	9	2

La demanda excede la disponibilidad (hay un estado inseguro), pero ¿podemos encontrar una secuencia de asignación de recursos?

Ejemplo 2

- ☑ En un sistema existen 3 procesos **P0, P1 y P2** que comparten **12** unidades de cinta.
- ☑ Según la tabla, hay 3 cintas libres.
- ☑ ¿En qué estado está el sistema?

Procesos	Máximo a usar	en uso
P_0	12	5
P_1	4	2
P_2	9	2

No hay una cadena segura → El sistema está en Deadlock

Algoritmos para evitar el deadlock

1. **Instancia única de un tipo de recurso**
   - **Algoritmo** que **determina** el **estado seguro** de un sistema.
   - Utilizar un **grafo de asignación de recursos.**
   - **Se debe encontrar una secuencia segura (evitar los bucles)**
2. **Múltiples instancias de un tipo de recurso**
   - Se utiliza el **algoritmo del banquero**
   - **Algoritmo teórico**
   - **Busca encontrar una secuencia segura de asignación**

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 necesitaría
- ☑ 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.

Estructuras asociadas

- ***n*:** cantidad de procesos
- ***m*:** cantidad de tipos de recursos
- ***disponible*:** 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 R_j .

Estructuras asociadas

- ***asignación*:** matriz de n x m que indica cuantos recursos tiene asignados cada proceso. Asignacion(i,j)=k, indica que hay k instancias del recurso Rj asignadas a Pi .
- ***max*:** matriz de n x m que indica la cantidad máxima re recursos que un proceso necesitará. Max(i,j)=k, indica que Pi necesitará en total k instancias del recurso Rj .
- ***need*:** matriz de n x m que indica la cantidad de recursos que le faltan a un proceso para completar su ejecución (need=max-asignacion). Need(i,j)=k, indica que Pi necesitará k instancias mas de las que ya tiene, del recurso Rj .

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Asignación

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Max

	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

Necesidad

R1	R2	R3
0	1	1

Disponible

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.
- Para este algoritmo, tomaremos filas de las matrices como si fueran vectores.
- Recursos asignados a P_i representados por el vector $Asig_i$ que es la fila i de la matriz Asig.

Algoritmo del Banquero

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Max

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Asignacion

	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

Necesidad

R1	R2	R3
0	1	1

Vector de disponibles D

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Max

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Asignacion

	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

Necesidad

R1	R2	R3
0	1	1

Vector de disponibles D

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Max

	R1	R2	R3
P1	1	0	0
P2	6	1	3
P3	2	1	1
P4	0	0	2

Asignacion

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

Necesidad

R1	R2	R3
0	1	0

Vector de disponibles D

Algoritmo del Banquero

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Max

	R1	R2	R3
P1	1	0	0
P2	6	1	3
P3	2	1	1
P4	0	0	2

Asignacion

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

Necesidad

R1	R2	R3
0	1	0

Vector de disponibles D

(continuación)

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Max

	R1	R2	R3
P1	1	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Asignacion

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

Necesidad

R1	R2	R3
6	2	3

Vector de disponibles D

- El algoritmo continúa con el fin de evaluar si existe una secuencia segura. En tal caso, podemos decir que el sistema se encuentra en un estado seguro

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Max

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	0

Asignacion

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	0

Necesidad

R1	R2	R3
9	3	6

Vector de disponibles D

Detección y Recuperación

- ☑ Permitir el estado de deadlock y luego recuperar
- ☑ La técnica **determina** si ocurrió un deadlock, y en tal caso busca **recuperarse** del mismo. Para ello vamos a contar con:
  1. Algoritmo que examine si ocurrió un deadlock
  2. Algoritmo para recuperación del deadlock

Detección

- ☑ Con recursos con una sola instancia
  - ✓ Análisis del grafo de asignación
- ☑ Con recursos de varias instancias
  - ✓ Algoritmo del Banquero

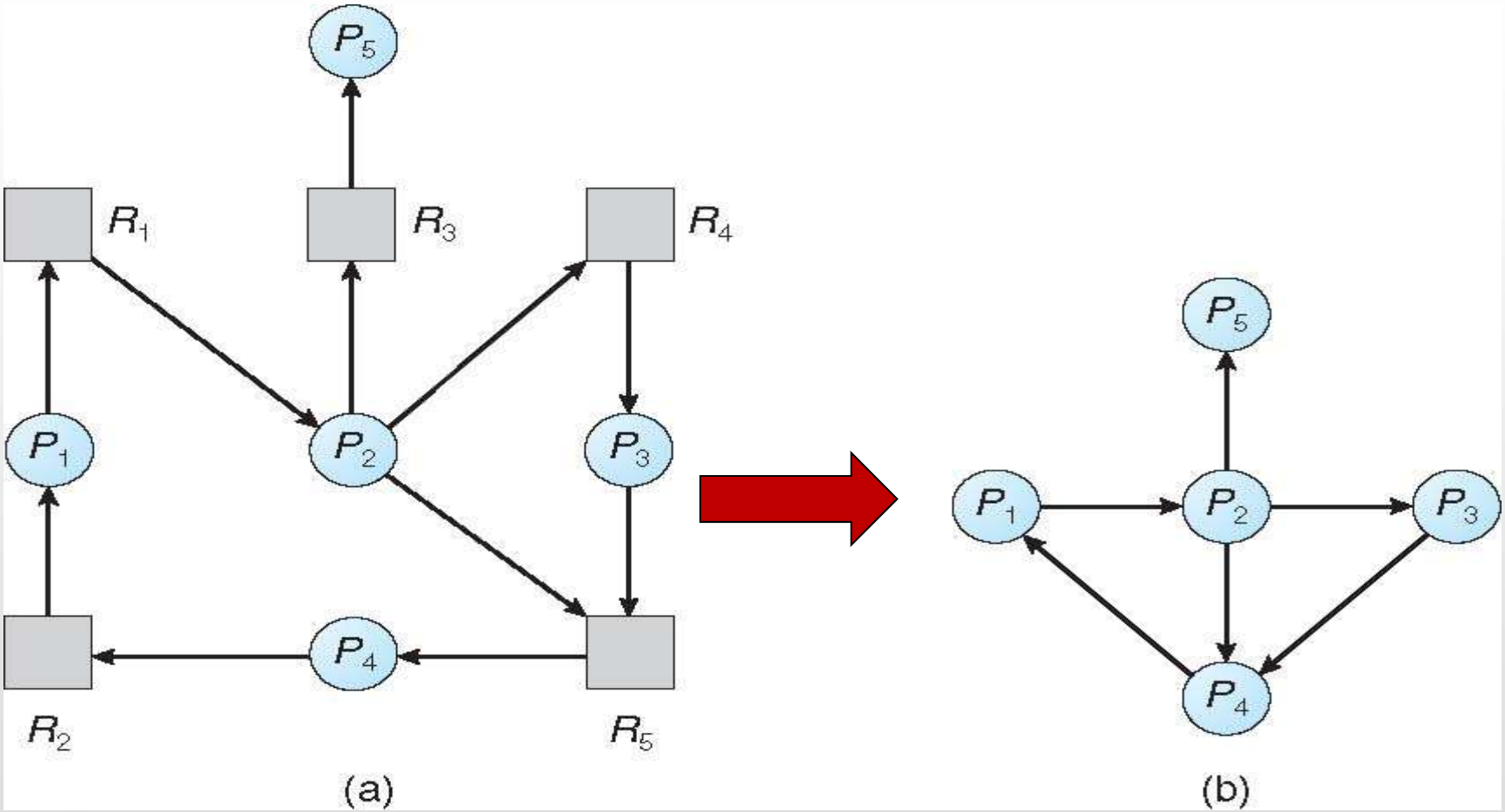
Detección. Instancia única de cada tipo de recurso

- **Mantener** un **grafo wait-for (Grafo de espera)**
  - ✓ **Nodos = Procesos**
  - ✓ $P_i \rightarrow P_j$, si P_i ***espera que*** P_j **libere recurso** R_q

- **El algoritmo se basa en buscar periódicamente un ciclo en el grafo.**

Si hay un ciclo, existe un bloqueo

Detección. Instancia única de cada tipo de recurso



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)?
- La **frecuencia** con la que se **debe ejecutar** el **algoritmo** de **detección** de **interbloqueos**, **suele** ser un **parámetro** del **sistema**.
- Los motores de BD también poseen algoritmos de detección de Deadlocks. Por ejemplo SQLserver lo corre cada 5 segundos

Cuando Uso del Algoritmo de Detección

- Una **posibilidad extrema** es **comprobar** el estado **cada vez** que se solicita un recurso y estos **no pueden ser asignados**. **Consume mucha CPU**
- Si el **algoritmo** de **detección** se **invoca arbitrariamente**, puede **haber muchos ciclos** en el **gráfo** de **recursos** y, por lo tanto, **no podríamos** decir cuál de los **muchos procesos bloqueados** "**causó**" el **bloqueo**.
- Hay que encontrar un término medio
- Cuando la utilización del Procesador desciende su actividad por debajo de un determinado porcentaje de utilización (recordar que 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:
- En cualquiera de los casos, las alternativas son:
  - **Abortar** 1 o más **Procesos** para **romper ciclo**
  - **Expropiar recursos** a 1 o más **Procesos del ciclo**

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.
    - 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 (puede estar actualizando un archivo).
  - Previo a terminar el proceso hay que hacerlo lelgar a un estado seguro (check Point) de modo que al reanudarlo quede consistente
- Se presenta un nuevo problema de política o decisión: selección de la víctima
- Se debe seleccionar aquel proceso cuya terminación represente el costo mínimo para el sistema.
- Hay que asegurarse de no seleccionar siempre al mismo proceso (inanición)

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.
- ☑ **Tiempo faltante para la liberación de recursos asignados**
- ☑ ¿**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)

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.
- Hay sobrecarga en tiempo de ejecución de la Detección
- Hay procesos que son reiniciados o rollback lo que genera demoras en la ejecución.

- La Detección y Recuperación de interbloqueos puede ser atractiva en sistemas con una baja probabilidad de interbloqueos.
- 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 100% 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.
- Se dividen los recursos del sistema en "clases de recursos":
  - Para cada clase se aplica el método más adecuado de manejo de interbloqueo
  - Depende de si soportan apropiaciones
  - Depende de si se puede predecir el comportamiento del recurso