Sistemas Operativos

Deadlocks – I
(Interbloqueos)

Sistemas Operativos

- ☑ **Versión: Abril 2020**
- ☑ **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:

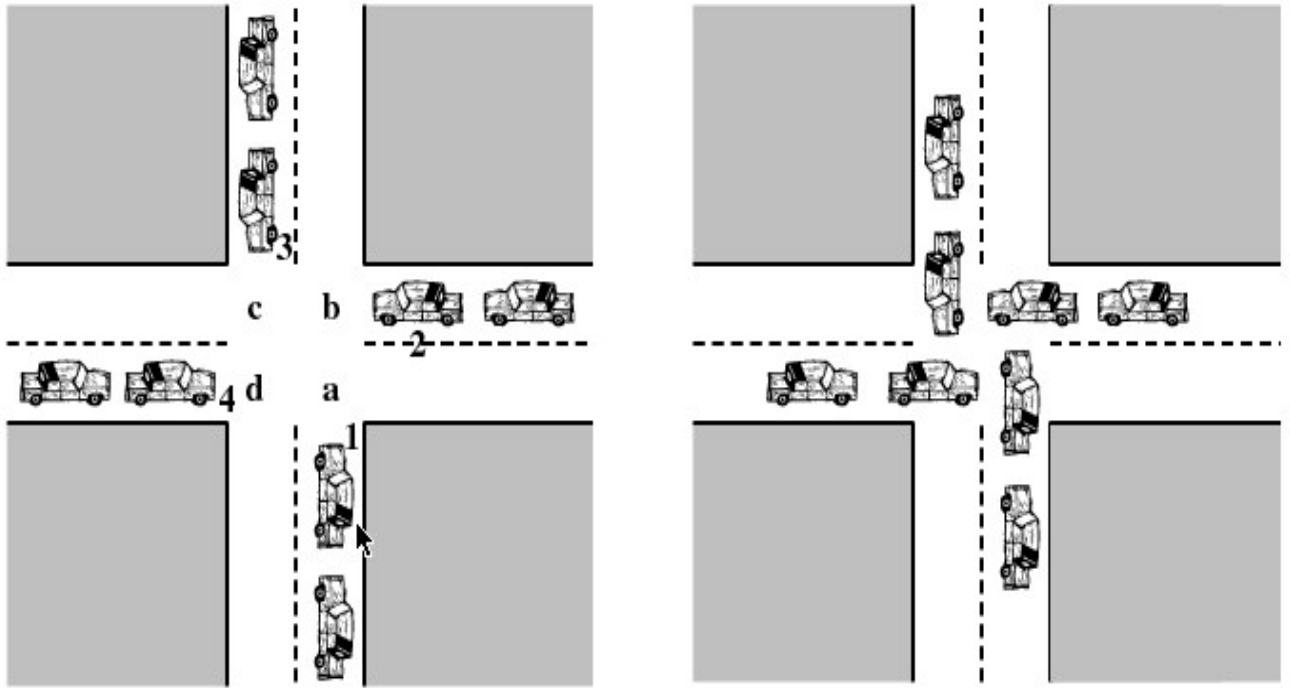
- ☑ Un proceso A pide un scanner.
- ☑ Un proceso B pide una grabadora de CD.
- ☑ El proceso A pide ahora la grabadora de CD
- ☑ El proceso B quiere el scanner.

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

Step	Action
p_0	Request (D)
p_1	Lock (D)
p_2	Request (T)
p_3	Lock (T)
p_4	Perform function
p_5	Unlock (D)
p_6	Unlock (T)

Process Q

Step	Action
q_0	Request (T)
q_1	Lock (T)
q_2	Request (D)
q_3	Lock (D)
q_4	Perform function
q_5	Unlock (T)
q_6	Unlock (D)

Recursos

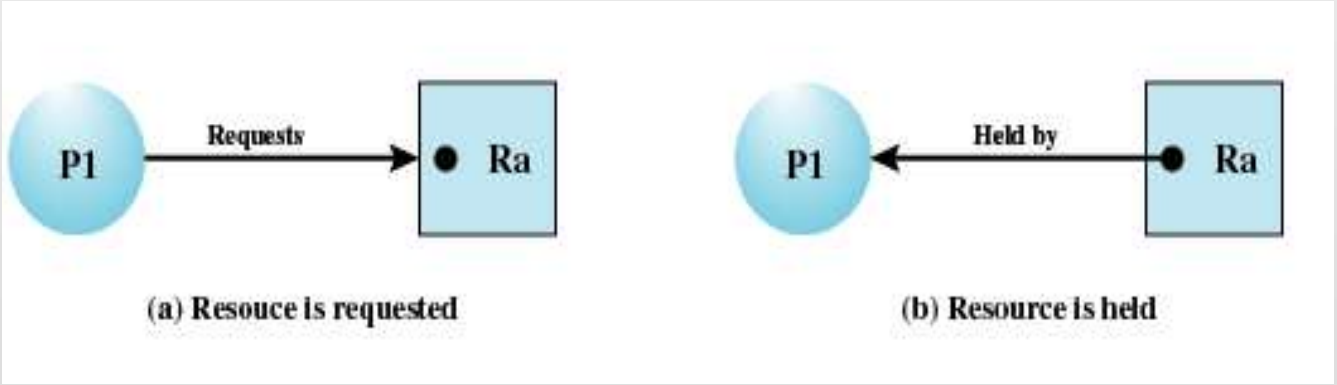
- Todo Sistema contiene recursos
- **Recursos físicos**
  - CPU, memoria, dispositivos.
- **Recursos lógicos**
  - archivos, registros, semáforos, etc.
- **Recursos apropiativos**: se le pueden quitar al proceso sin efectos dañinos (ej: memoria, CPU).
- **Recursos no apropiativos**: si se le saca al proceso, éste falla (interrumpir una escritura a CD, impresora).
- Cada recurso Rj puede tener Wi **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**.

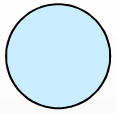
Representación de procesos y recursos

- ☑ Para poder representar la asignación de recursos, se utiliza un **Grafo de Asignación de recursos**.
- ☑ El gráfico **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

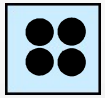


Representación de procesos y recursos

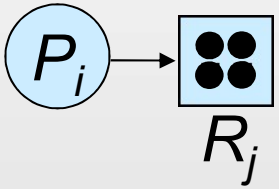
Proceso



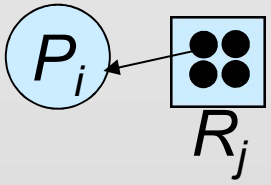
Tipo de Recurso con 4 instancias



P_i solicita una instancia de R_j



P_i tiene asignada una instancia de R_j



Ejemplo – Varias Instancias

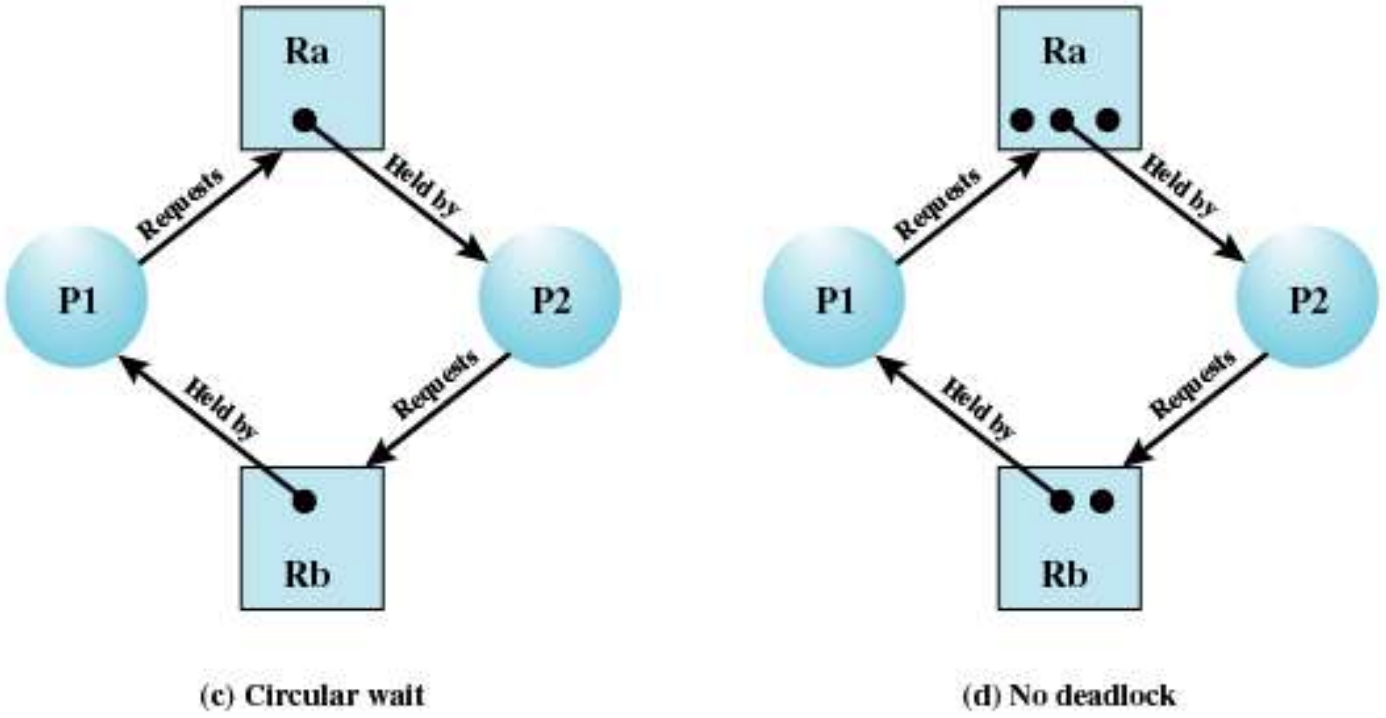
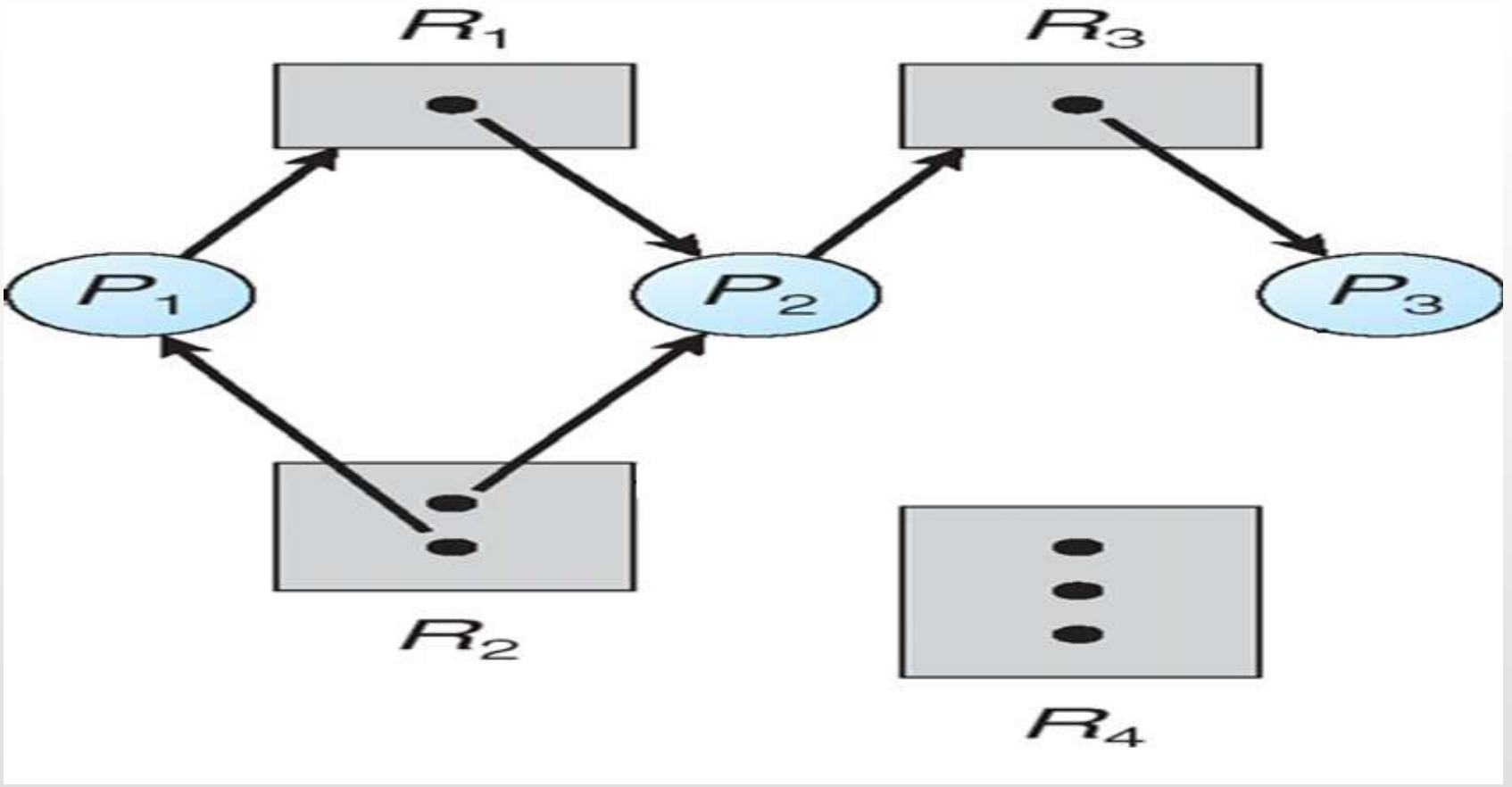


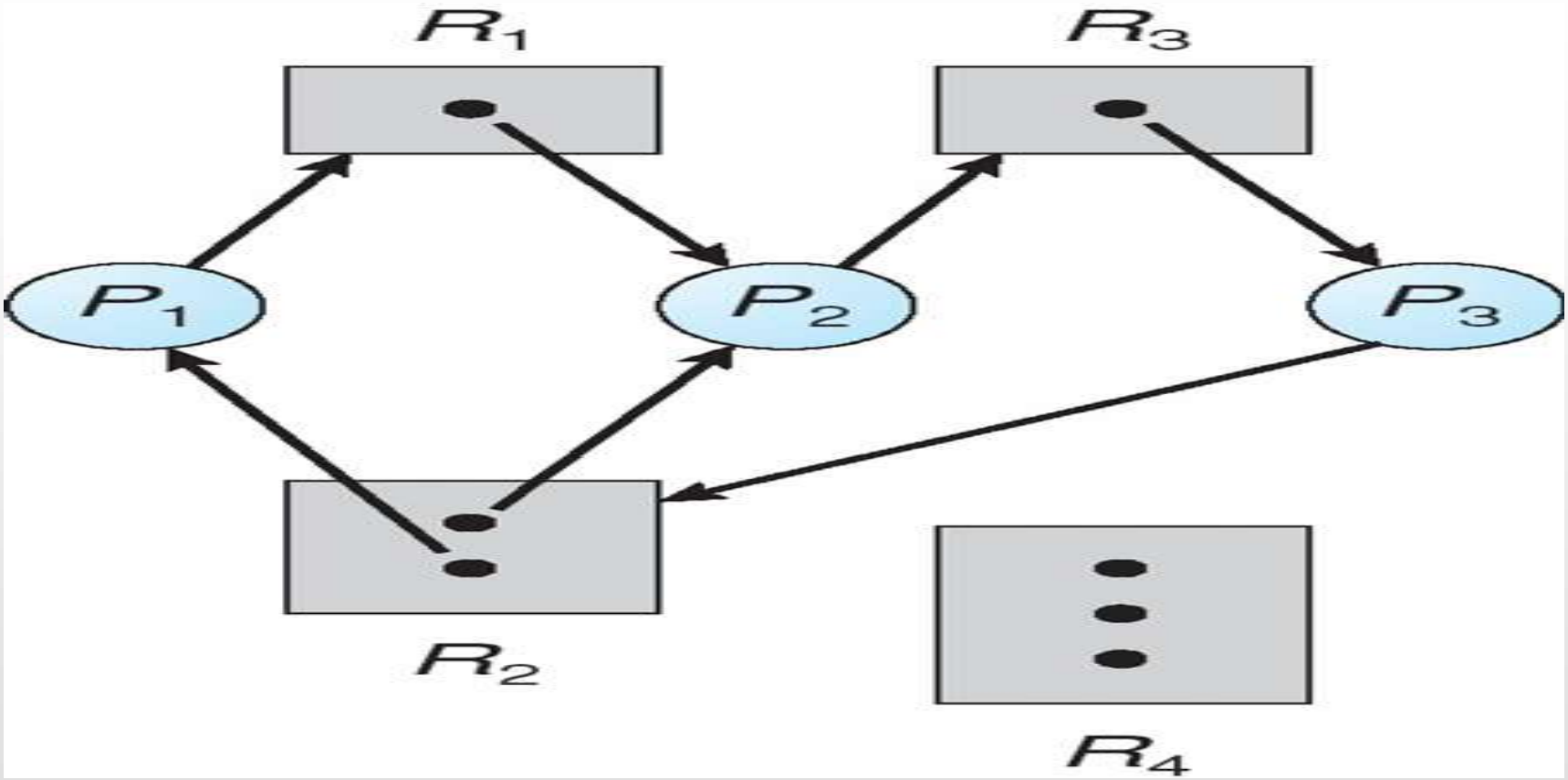
Figure 6.5 Examples of Resource Allocation Graphs

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!**

http://www.ccs.neu.edu/home/pjd/cs7600-s10/Tuesday_January_26_01/p67-coffman.pdf

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

> ***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:
  1. Si un proceso requiere un recurso que no está disponible, **todos los recursos que tiene asignado ese proceso son apropiados y se bloquea el proceso**. Se desbloquea cuando el recurso pedido y los que tenía están nuevamente disponibles.
  2. Si un proceso requiere un recurso que no está disponible, **el sistema chequea si ese recurso lo tiene asignado un proceso que esté bloqueado a la espera de otro recurso. Si es así, el recurso es apropiado del proceso bloqueado y asignado al solicitante**. Si no es así, el proceso solicitante es bloqueado hasta que el recurso esté disponible

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 $F(R_j) > F(R_i)$**
- ☑ 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

Sobre información de los recursos

- ☑ **Es necesario conocer la secuencia de solicitud, uso y liberación** de cada recurso requerido por el proceso.
- ☑ Ante cada requerimiento **se evalúa la incidencia de la asignación** (posibles consecuencias)
- ☑ **Requiere que cada proceso declare el número máximo de recursos de cada tipo que pueda necesitar**.
- ☑ El algoritmo de prevención de interbloqueo **examina dinámicamente el estado de asignación de recursos para asegurar que nunca puede haber una condición de espera circular**. (posibles consecuencias)
- ☑ El estado de asignación de recursos se define por el número de recursos disponibles y asignados, y las demandas máximas de los procesos

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 de tal manera que para cada P_i, los recursos que P_i puede aún solicitar puedan ser satisfechos por los recursos disponibles + los recursos mantenidos por todos los P_j, con $j < i$ (los recursos asignados a los procesos que actualmente se están ejecutando con todos sus requerimientos satisfechos).

Estado sano o seguro

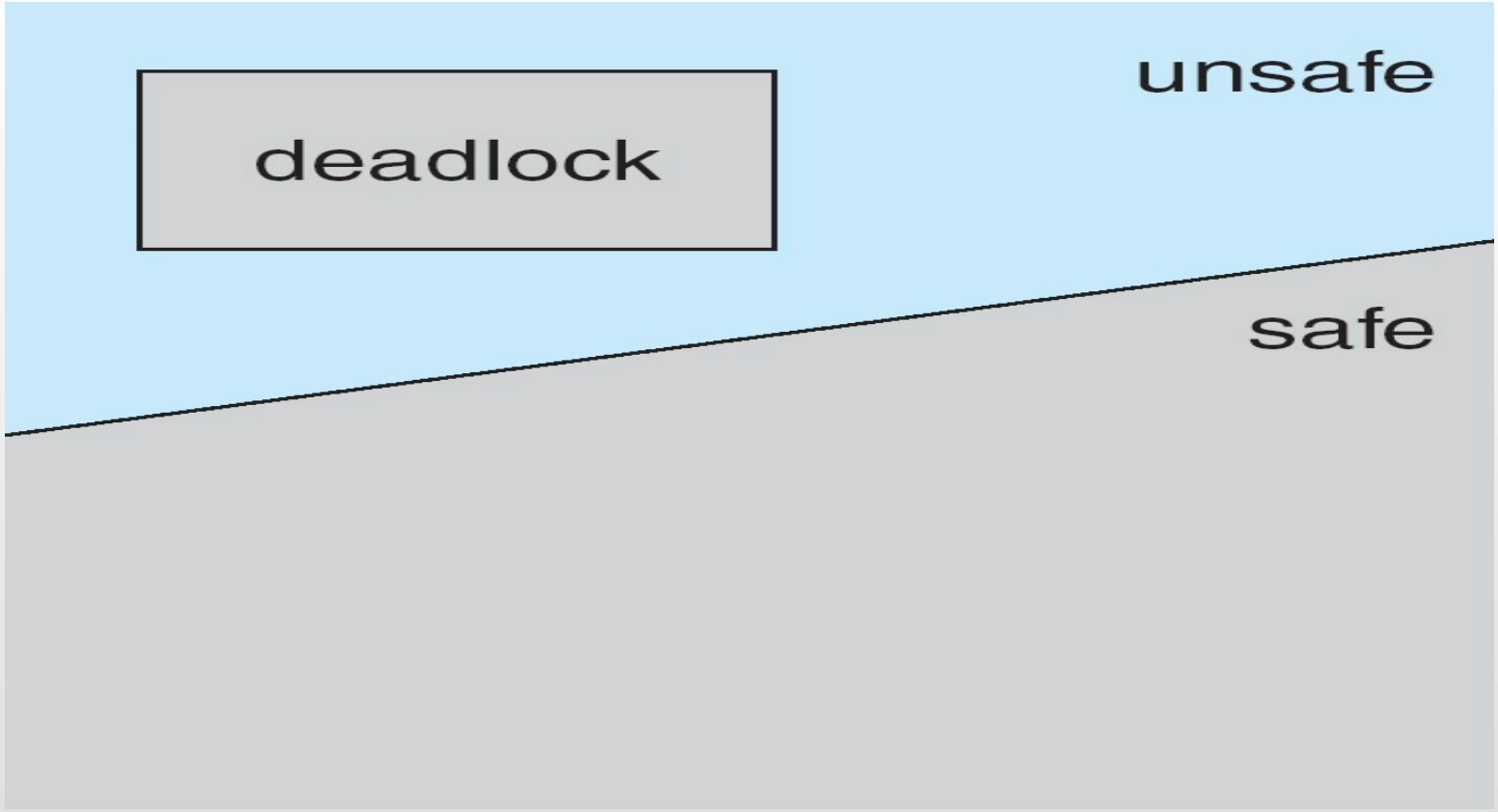
- Si los **recursos necesarios de P_i no están disponibles inmediatamente, entonces P_i puede esperar** hasta que todos los P_j hayan terminado.
- Cuando P_j está terminado, P_i puede obtener los recursos necesarios, ejecutar, devolver los recursos asignados y finalizar.
- Cuando P_i termina, P_i +1 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.
- ☑ Si hay deadlock, estoy en estado inseguro.
- ☑ En estado inseguro **hay posibilidad** que haya deadlock.
  - ➢ No todo estado inseguros es deadlock
- ☑ 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

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

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 Estado inseguro

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 .
- ***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 R_j asignadas a P_i .
- ***max*:** matriz de n x m que indica la cantidad máxima re recursos que un proceso necesitará. Max[i,j]=k, indica que P_i necesitará en total k instancias del recurso R_j .
- ***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 P_j necesitará k instancias mas de las que ya tiene, 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 $X \leq Y$ si y solo si $X(i) \leq Y(i)$, 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