Sistemas Operativos

Deadlocks – I
(Interbloqueos)

Sistemas Operativos

- ☑ **Versión: Mayo 2017**
- ☑ **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)

Chapter 7: Deadlocks

- **System Model**
- **Deadlock Characterization**
- **Methods for Handling Deadlocks**
- **Deadlock Prevention**
- **Deadlock Avoidance**
- **Deadlock Detection**
- **Recovery from Deadlock**

Objectivo

1. Desarrollar una descripción de los deadlocks, que impiden que los conjuntos de procesos concurrentes terminen sus tareas

1. Presentar una serie de diferentes métodos para prevenir o evitar los bloqueos en un sistema informático

Definición de Deadlock

- 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

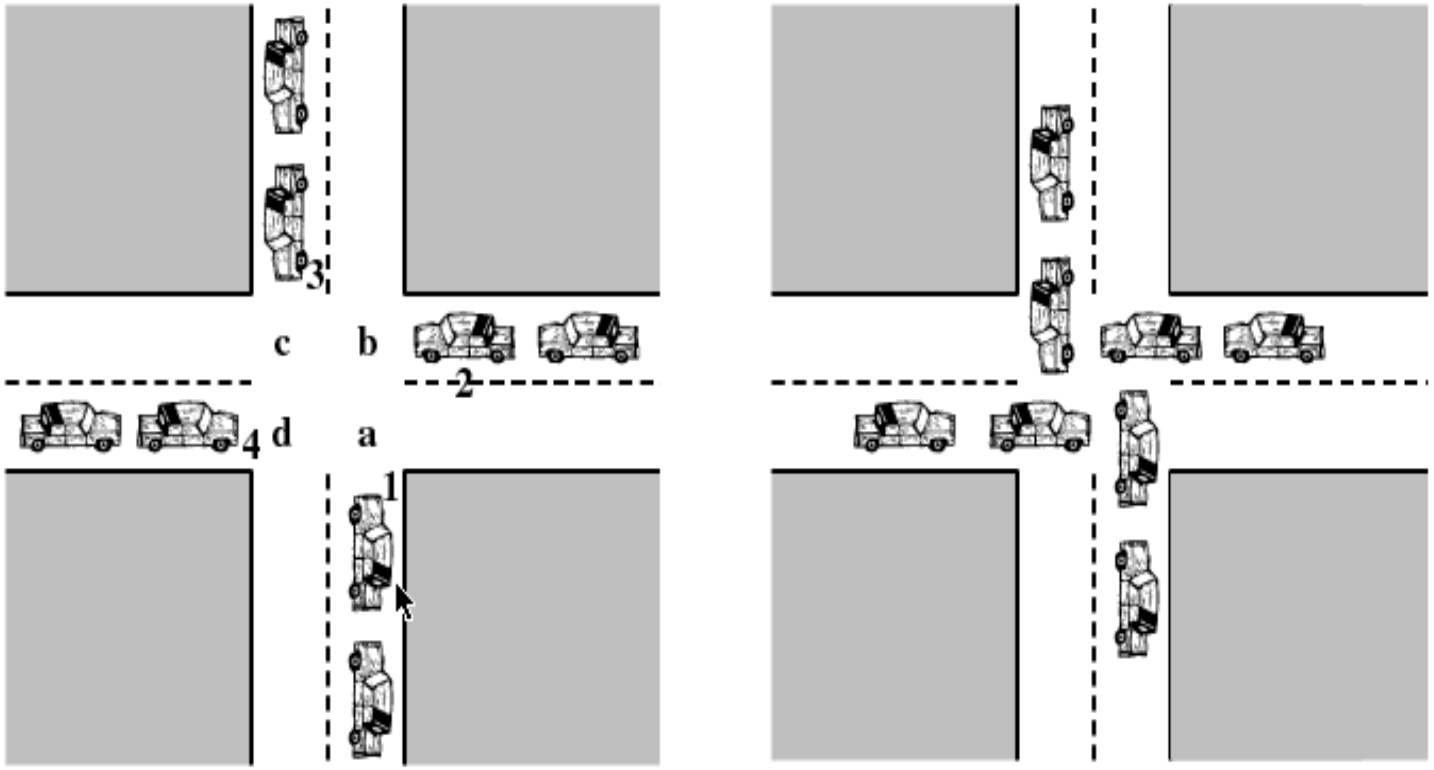
- ☑ Un proceso **A** pide un **scanner**.
- ☑ Un proceso **B** pide una **grabadora** de CD.
- ☑ El proceso **A** pide ahora la **grabadora** de CD
- ☑ El proceso **B** quiere el **scanner**.

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**.

Quieren el Recurso que tiene otro Proceso

Ejemplos



(a) Deadlock possible

(b) Deadlock

Figure 6.1 Illustration of Deadlock

Ejemplos

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)

Modelo del Sistema - Recursos

- **Todo Sistema contiene recursos**
- **Recursos físicos**
  - CPU, memoria, dispositivos.
- **Recursos lógicos**
  - archivos, registros, semáforos, etc.
- **Recursos apropiativos: se le puede quitar al proceso sin efectos dañinos (ej: memoria, CPU).**
- **Recurso no apropiativo: 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**

Modelo del Sistema – Recursos (cont.)

- **Clase de un Recurso:**
  - es el **conjunto** de **instancias** de un **recurso**
- **Ciclo del Recurso**
  1. Solicitud (**Request**)
  2. Uso (**Use**)
  3. Liberación (**Release**)

Deadlock Caracterizacion

Deadlock can arise if 4 conditions hold simultaneously.

1. **Mutual exclusion**: only **1 process** at a **time** can **use 1 resource**
2. **Hold and wait**: a **process holding** at least **one resource** is **waiting** to **acquire additional resources held** by **other processes**
3. **No preemption**: a **resource** can be **released only voluntarily** by the **process holding** it, **after** that **process** has **completed** its task
4. **Circular wait**: there exists a **set {P_0, P_1, …, P_n} of waiting processes** such that P_0 is **waiting** for a **resource** that is **held** by P_1, P_1 is **waiting** for a **resource** that is **held** by P_2, …, P_{n-1} is **waiting** for a **resource** that is **held** by P_n, and P_n is **waiting** for a **resource** that is **held** by P_0.

Representación de procesos y recursos - Grafo

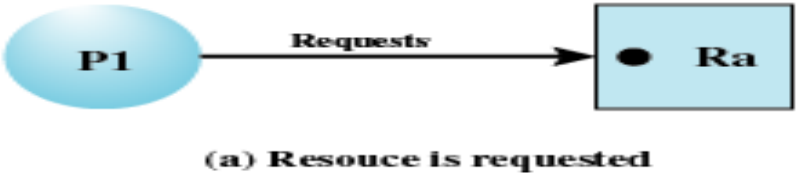
- Se utiliza un **grafo** de **asignación de recursos**.
- Un **set de vértices** y **E** un **set** de **ejes**

vertices partitioned into 2 types:

- $P = \{P_1, P_2, \ldots, P_n\}$, the set of all the **processes** in the **system**
- $R = \{R_1, R_2, \ldots, R_m\}$, the set of **all resource types** in the **system**

request edge – directed edge $P_i \rightarrow R_j$

assignment edge – directed edge $R_j \rightarrow P_{i11}$

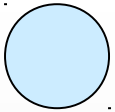


(a) Resouce is requested

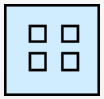
(b) Resource is held

Resource-Allocation Graph (Cont.)

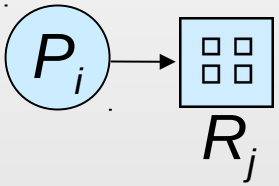
Process



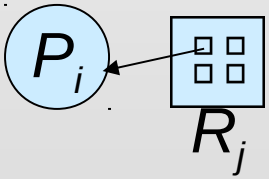
Resource Type with 4 instances



P_i* requests instance of *R_j



P_i* is holding an instance of *R_j



Ejemplo

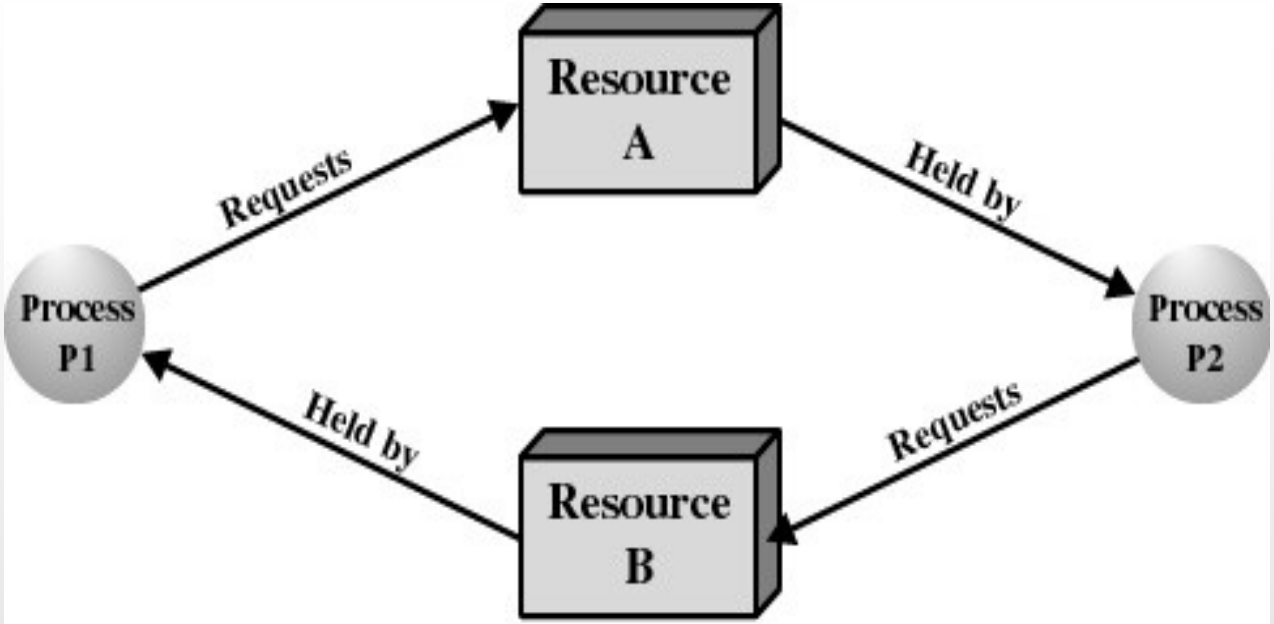
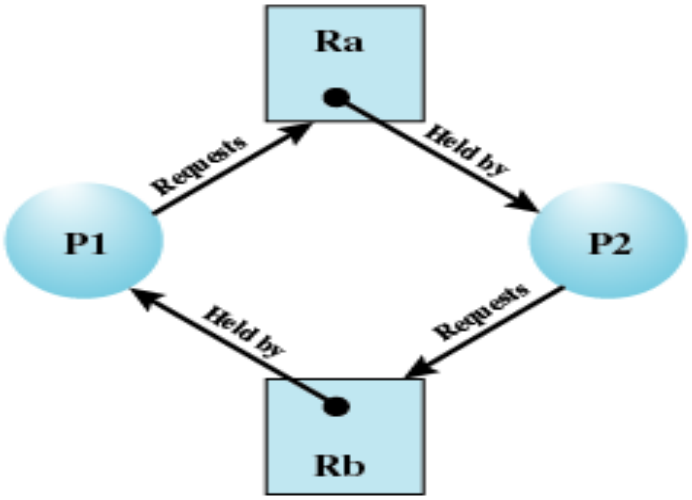
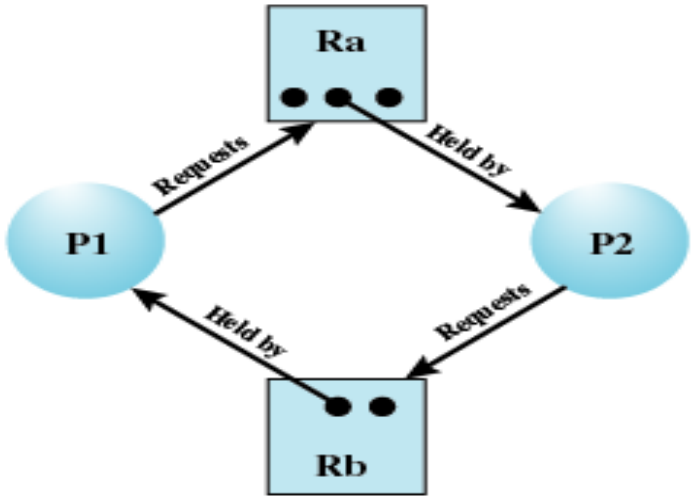


Figure 6.5 Circular Wait

Ejemplo – Varias Instancias



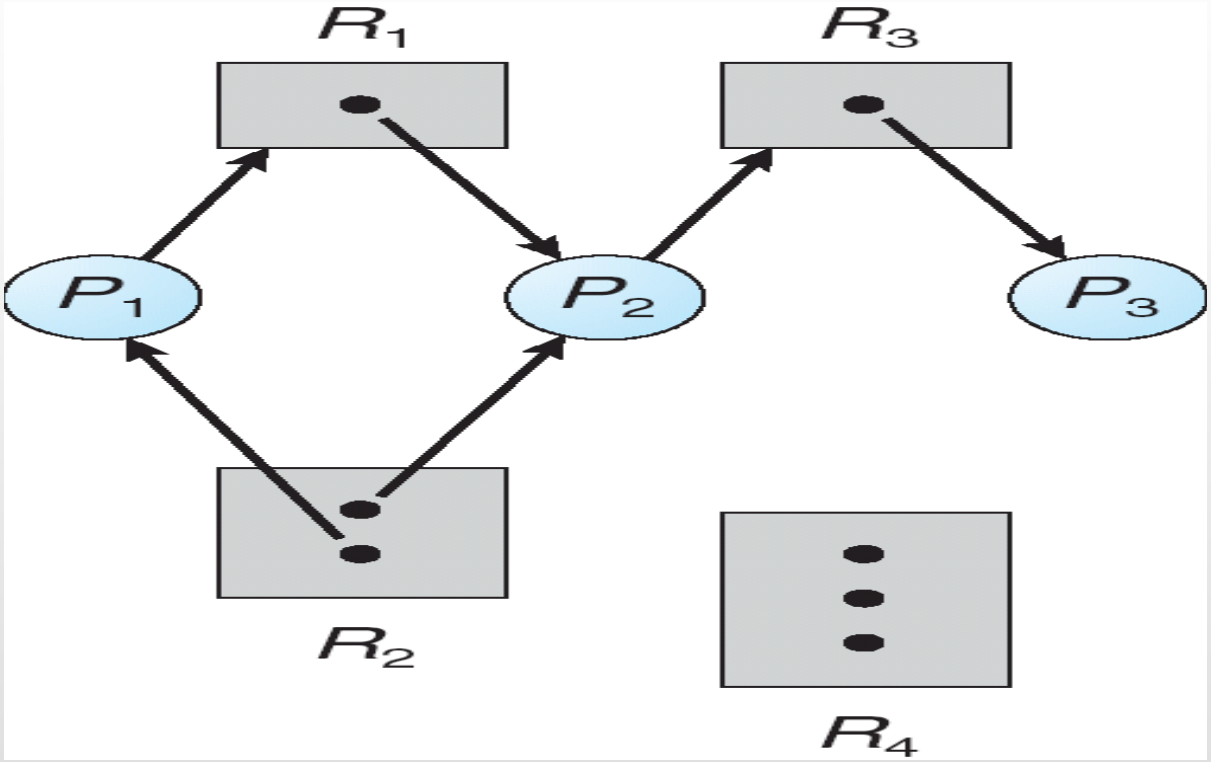
(c) Circular wait



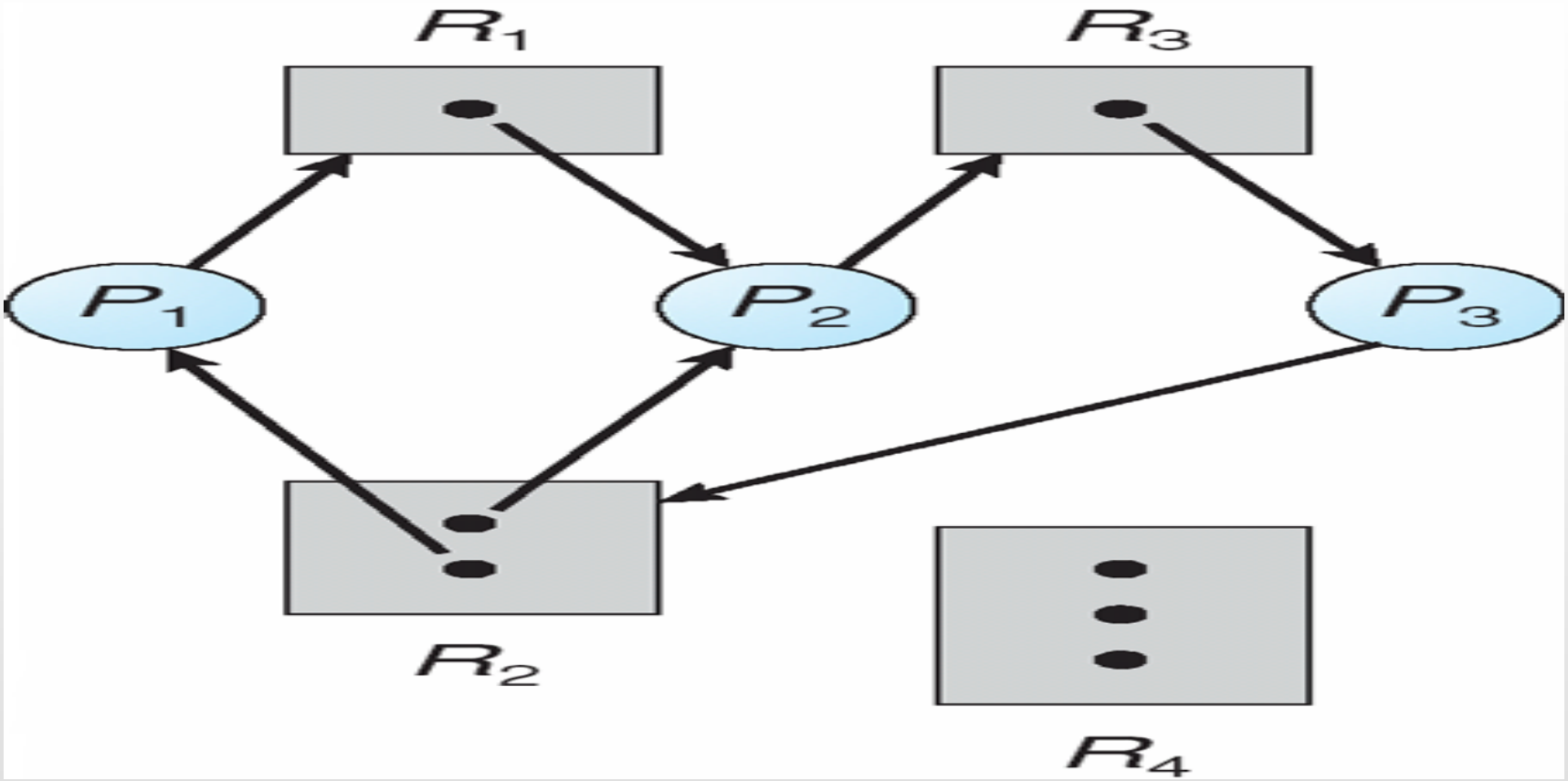
(d) No deadlock

Figure 6.5 Examples of Resource Allocation Graphs

Example of a Resource Allocation Graph



Resource Allocation Graph With A Deadlock



Cuantos ciclos hay?

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 que hay romper para que no haya deadlock

- **Condiciones:**
  1. **Exclusión mutua** (solo **1 proceso por vez** puede **usar** el **recurso**)
  2. **Retención y espera** (**P1 tiene** al menos **R1** y **espera** otro **R2** que **posee** otro **P2**)
  3. **No apropiación**
  4. **Espera circular**

LOGRAR QUE ALGUNA NO SE CUMPLA!!

Métodos para el tratamiento del Deadlock

- **Asegurar que el Sistema NUNCA se entrará en estado de Deadlock**
  1. **Prevenir que ocurra el** Deadlock
  2. **Evitar que ocurra el** Deadlock
  3. **Permitir que ocurra el Deadlock, Detectar y luego Recuperarse**
  4. **Ignorar el problema (y esperar que nunca ocurra un deadlock como los SO actuales incluido Linux Windows)**

prevention <> avoidance (el NUNCA)

1. **Prevention (<u>PREVENIR</u> la formación del interbloqueo):** que **por lo menos 1 de las 4 condiciones no pueda mantenerse**. Se imponen **<u>restricciones</u>** en la forma en que los procesos **<u>REQUIEREN los recursos</u>**.
2. **Avoidance (<u>EVITAR</u> la formación del interbloqueo):** **asignar cuidadosamente los recurso**s, **manteniendo información** actualizada sobre **requerimiento** y **uso** de **recursos**. (NO ES BUENA SOLUCIÓN)

Prevenir: Condición 1 de exclusión mutua

- **Si ningún recurso se asignara de manera exclusiva (todos los R no fueran exclusivos) ⇒ no habría interbloqueo.**
- **NO SIEMPRE SE PUEDE!**
- **Considerar que hay recursos que son compartibles (ej. archivos read only, RAM, CPU)**
- **Considerar que hay recursos que no son compartibles (ej. Impresoras).**
  - **Ejemplo: impresora y proceso de impresión.**
- **Mantener la exclusión mutua para los recursos NO compartibles**
- **Los recursos compartibles NO requieren mantener la exclusión mutua**

Prevenir: Condición 2 de retención y espera

- **Se basa en que si un proceso requiere un recurso, DEBE LIBERAR otros.**
- **Alternativas:**
  - El Proceso **DEBE requerir** y **reservar TODOS** los **Recursos** a **usar ANTES** de comenzar la **ejecución** (precedencia de los system calls que hacen el requerimiento antes de cualquier otra system call)
  - El Proceso puede **requerir recursos SÓLO** cuando **NO tiene ninguno**. **(libero todos)**
- **Desventajas**
  - **Baja utilización** de recursos
  - Posibilidad de **inanición** de alguno de los procesos (**starvation, o espera infinita**)

Prevenir: Condición 3 de no apropiación

- No siempre se puede atacar esta condición. **No siempre se puede quitar un recurso**. No es una buena solución
- **Posible solución:**
  - Si un **recurso no puede asignarse** a un **proceso** y **queda** en **wait**, se **liberan todos** sus **recursos**.
  - Los **recursos apropiados** se **agregan** a la **lista** de **recursos** que está **esperando**
  - El **proceso** se **reiniciará sólo** cuando pueda **recuperar todos** sus **recursos** (antiguos y nuevos)

Prevenir: Condición 4 de Espera circular

- Se define un **ordenamiento** de **todos** los **recursos**. Los **procesos deben** requerir **recursos** en un **orden** numérico **ascendente**.
  - Sea **F:R→N,** N conjunto de los naturales.
  - **F** asigna un **numero único** a **cada recurso** (los **números pequeños** para **recursos muy usados**).
  - Un **proceso**, que **ya tiene Ri** puede **requerir Rj si y solo si** ***F(Rj) > F(Ri)*** ***(creciente)***

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.

NO PUEDE PEDIR ALGO CON NUMERO INFERIOR

Los diversos métodos para evitar el interbloqueo se sintetizan en la figura 6-14.

Condición	Método
Exclusión mutua	Evitar recurso se asigne exclusivo
Contención y espera	Solicitar todos los recursos al principio
No apropiativa	Quitar los recursos
Espera circular	Ordenar los recursos en forma numérica

Figura 6-14. Resumen de los métodos para evitar interbloqueos.

Evitar/Avoidance Deadlocks

Requiere que el SO tenga información ANTES

- **El SO cuenta con información sobre el uso de los recursos**
  - **cómo se requieren**
  - en qué **momento** del sistema **son requeridos**
  - la **demanda máxima** de recursos, etc.
- **Desventajas:**
  - puede producir una **baja utilización** de los **recursos**
  - puede producir una **baja performance** del **sistema**

Sobre información de los recursos

- **Conocer** la **secuencia** de **solicitud**, **uso** y **liberación** de cada **recurso** requerido por el **proceso**.
- 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**.

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 Pi**, los **recursos** que **Pi** puede aún **solicitar** puedan ser **satisfechos** por los **recursos disponibles + los recursos mantenidos** por **todos** los **Pj, con j <I.**

puede usar los recursos disponibles más los que liberan los procesos anteriores

Es decir

Estado sano o seguro

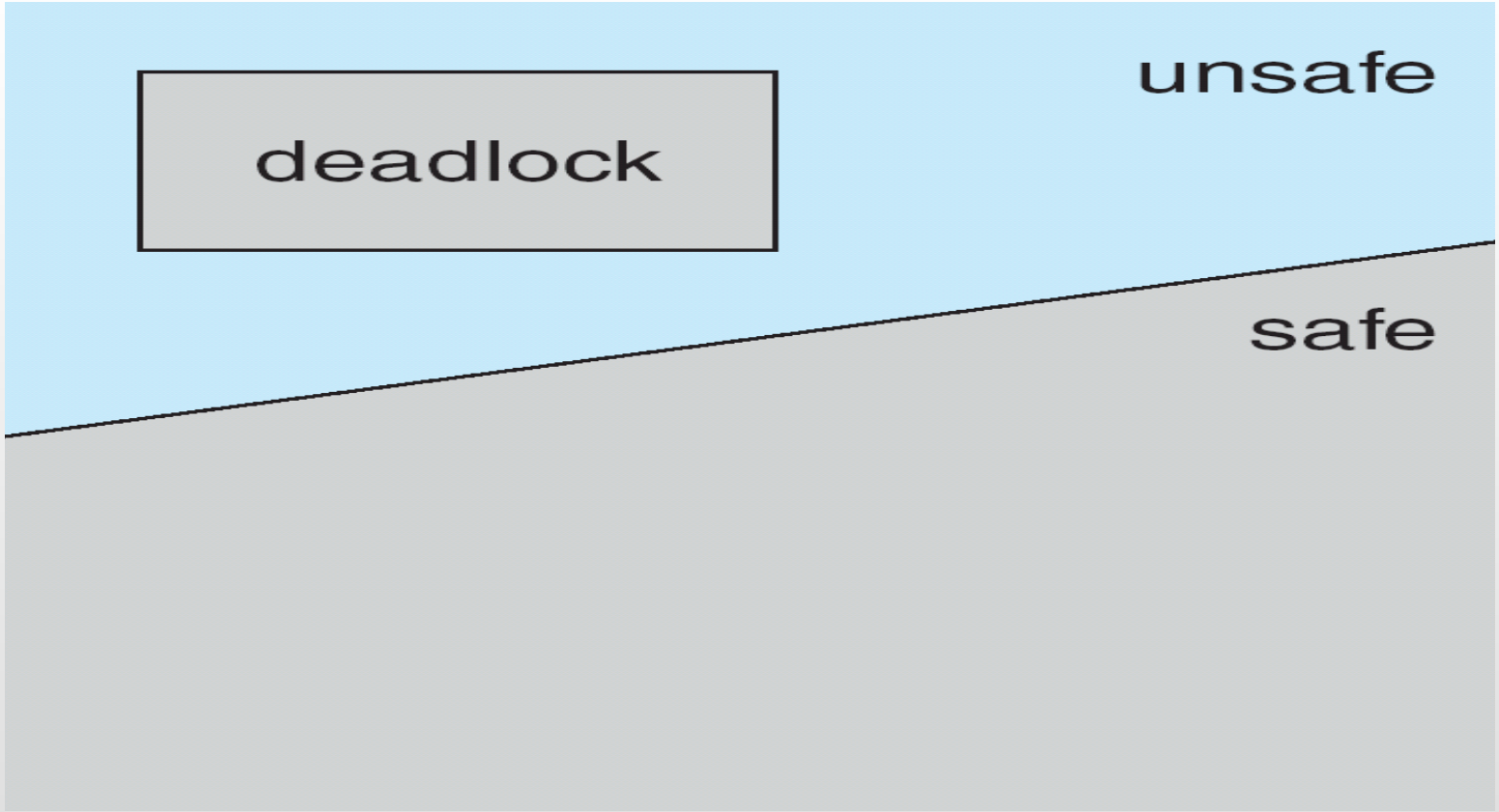
- Si los **recursos** necesarios de **Pi no** están **disponibles** inmediatamente, entonces **Pi** puede **esperar** hasta que **todos** los **Pj** hayan **terminado**.
- Cuando **Pj** está **terminado**, **Pi** puede **obtener** los **recursos necesarios**, **ejecutar**, **devolver** los **recursos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

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

Importante!

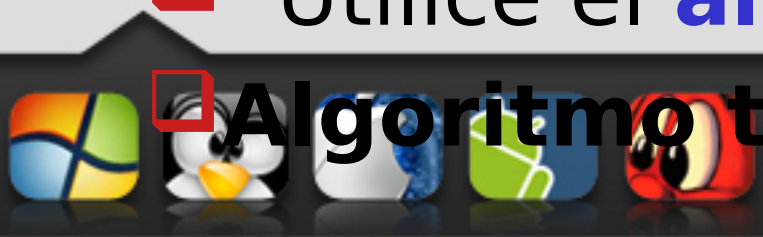
- Un **estado seguro** garantiza que **NO hay deadlock**.
- **SI** hay **deadlo**ck, **estoy en estado inseguro**.
- En **estado inseguro hay** posibilidad que **SI haya deadlock**.
  - **No todo estados inseguros** es **deadlock**
- **EVITAR/AVOIDANCE**: trata de **nunca** entrar a **estado inseguro**. **Garantiza** lo **seguro** entonces **no hay deadlo**ck.

Safe, Unsafe, Deadlock State



Algoritmos para evitar el deadlock

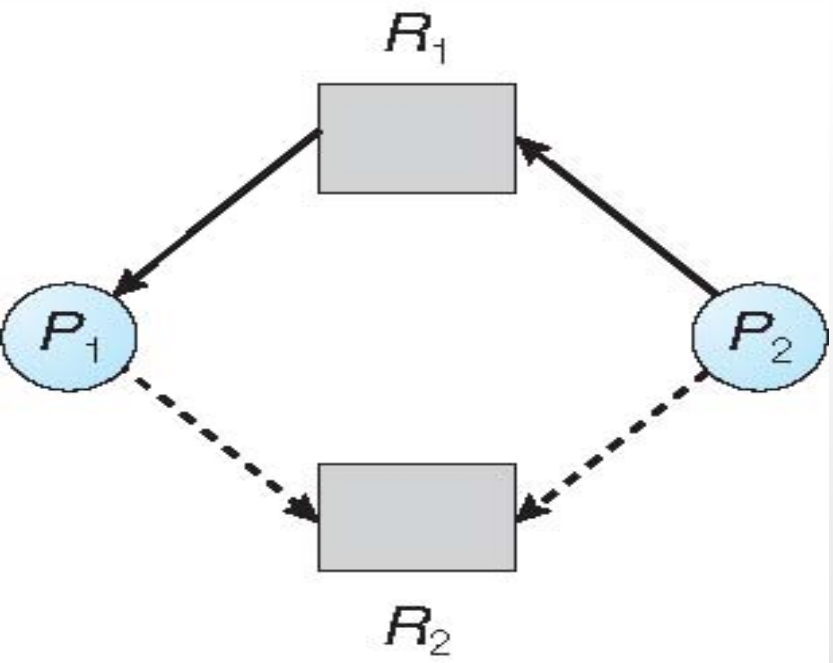
1. **Instancia única** de **un tipo** de **recurso**
   - **Algoritmo** que **determina** el **estado seguro** de un sistema.
   - Utilizar un **grafo** de **asignación** de **recursos**. **Encuentro secuencia**
2. **Múltiples instancias** de **un tipo** de **recurso**
   - Utilice el **algoritmo del banquero**
   - **Algoritmo teórico**



Resource-Allocation Graph

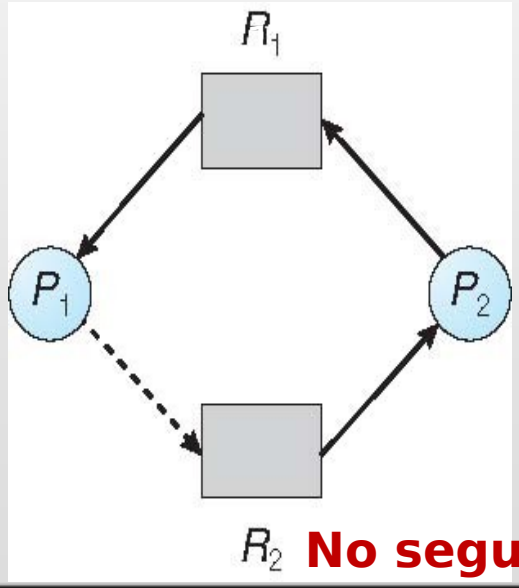
- Los **recursos deben** ser **reclamados ANTES** en el **sistema. Se arma un grafo**
- La **línea de reclamación Pi → Rj** indica que el **proceso Pj** puede **solicitar** el **recurso Rj**; (**línea punteada)**
- La **línea de reclamación/punteada** se **convierte** en **línea de solicitud** cuando un **proceso solicita** un **recurso (línea continua)**
- **línea de solicitud convertida** a **línea de asignación** cuando el **recurso** se **asigna** al **proceso Rj → Pi**
- Cuando un **recurso** es **liberado** por un proceso, la **línea de asignación** se vuelve a **convertir** en un **línea de reclamación/punteada**

Resource-Allocation Graph



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

- The **request** can be **granted only** if converting the **request edge** to an **assignment edge** does **not result** in the formation of a **cycle** in the **resource allocation graph**



No seguro

Próxima clase continuamos con Deadlock