Sistemas Operativos II

Universidad Nacional ARTURO JAURETCHE

Clase 1

Interbloqueo (Abrazo Mortal – deadlock)

Docentes:
Ing. Jorge Osio
Ing. Eduardo Kunysz

Datos de la cursada

1. La materia consiste de dos módulos en donde se tendrán dos parciales con sus recuperatorio.
2. Se elimina el flotante, pero se podrá recuperar el primer parcial en el recuperatorio del segundo módulo
3. Se debe tener el 80% de asistencia para aprobar la materia
4. Se realizada un trabajo de laboratorio a lo largo de la cursada que contribuirá con el 20% de la nota final
5. Web de la cátedra:

   https://sites.google.com/site/so2unaj/

Definición Interbloqueo

- Los sistemas computacionales están llenos de recursos que pueden ser utilizados por sólo un proceso a la vez. Ejemplo; impresoras, las unidades de cinta, etc
- Si dos procesos utilizan la misma entrada en la tabla del sistema de archivos invariablemente se corrompe el sistema de archivos.
- Todos los sistemas operativos tienen la habilidad de otorgar el acceso exclusivo a ciertos recursos.

Definición Interbloqueo

Para muchas aplicaciones, un proceso necesita acceso exclusivo no sólo a un recurso, sino a varios.

Dos procesos quieren grabar un archivo digitalizado en un CD. El proceso *A pide permiso para utilizar el escáner y se le otorga. El proceso B se programa* de manera distinta y solicita primero el grabador de CDs, y también se le otorga. Ahora *A* pide el grabador de CDs, pero la petición se rechaza hasta que *B lo libere. Por desgracia, en vez de* liberar el grabador de CD, *B pide el escáner. En este punto ambos procesos están bloqueados y permanecerán* así para siempre. A esta situación se le conoce como **interbloqueo.**

Los interbloqueos también pueden ocurrir entre máquinas. Por ejemplo, una red de área local con muchas computadoras.

Ejemplo en un sistema de bases de datos un programa puede bloquear acceso a registros. los interbloqueos pueden ocurrir en recursos de hardware o de software.

Interbloqueo - Recursos

- La clase principal de interbloqueos involucra a los recursos
- Para que el análisis sobre los interbloqueos sea lo más general posible, nos referiremos a los objetos otorgados como **recursos**
- Un recurso puede ser un dispositivo de hardware o una pieza de información
- En resumen, un recurso es cualquier cosa que se debe adquirir, utilizar y liberar con el transcurso del tiempo

Recursos apropiativos y no apropiativos

- Un **recurso apropiativo es uno que** se puede quitar al proceso que lo posee sin efectos dañinos. Por ejemplo, la memoria
- Un **recurso no apropiativo es uno que no se puede quitar a su propietario actual** sin hacer que el cómputo falle. Por ejemplo, si un proceso ha empezado a quemar un CD-ROM y tratamos de quitarle de manera repentina el grabador de CD y otorgarlo a otro proceso, se obtendrá un CD con basura
- Nuestro análisis se enfocará en los recursos no apropiativos.
- Si el recurso no está disponible cuando se le solicita, el proceso solicitante se ve obligado a esperar.

Interbloqueo: Definición

El interbloqueo se puede definir formalmente de la siguiente manera:

Un conjunto de procesos se encuentra en un interbloqueo si cada proceso en el conjunto está esperando un evento que sólo puede ser ocasionado por otro proceso en el conjunto.

Debido a que todos los procesos están en espera, ninguno de ellos producirá alguno de los eventos que podrían despertar a cualquiera de los otros miembros del conjunto, y todos los procesos seguirán esperando para siempre.

Para este modelo suponemos que los procesos tienen **sólo un hilo y que no hay interrupciones** posibles para despertar a un proceso bloqueado.

Adquisición de Recursos

Para ciertos tipos de recursos, los procesos de usuario son los administradores

Permitir que los usuarios administren los recursos

es asociar un semáforo con cada recurso:

```
typedef int semaforo;
semaforo recurso_1;
void proceso_A(void) {
     down(&recurso_1);
     usar_recurso_1();
     up(&recurso_1);}
```

Algunas veces los procesos necesitan dos o más recursos. Estos se pueden adquirir de manera secuencial como se muestra a continuación.

Adquisición de recursos

Código libre de interbloqueos

```
typedef int semaforo;
semaforo recurso_1;
semaforo recurso_2;
void proceso_A(void) {
   down(&recurso_1);
   down(&recurso_2);
   usar_ambos_recursos();
   up(&recurso_2);
   up(&recurso_1); }
void proceso_B(void){
   down(&recurso_1);
   down(&recurso_2);
   usar_ambos_recursos();
   up(&recurso_2);
   up(&recurso_1);}
```

Código con potencial interbloqueo

```
semaforo recurso_1;
semaforo recurso_2;
void proceso_A(void) {
   down(&recurso_1);
   down(&recurso_2);
   usar_ambos_recursos();
   up(&recurso_2);
   up(&recurso_1);
void proceso_B(void) {
   down(&recurso_2);
   down(&recurso_1);
   usar_ambos_recursos();
   up(&recurso_1);
   up(&recurso_2);}
```

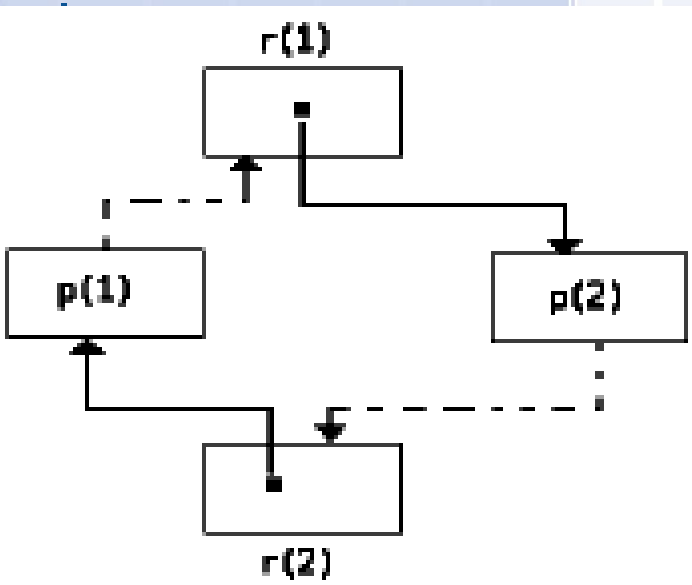
CONDICIONES NECESARIAS PARA EL INTERBLOQUEO (DEADLOCK)

Exclusión Mutua: debe haber recursos no compartidos de uso exclusivo, es decir, cada recurso se asigna a un solo proceso en un momento dado.

Contención y espera(Hold & Wait): un proceso retiene un recurso y espera por otro.

Condición no apropiativa: no es posible quitarle el recurso otorgado a un proceso p
fuerza. Este lo libera voluntariame

Espera circular: cadena circular
dos o más procesos, cada uno de l
cuales espera un recurso retenido
por el siguiente miembro de la cad



Modelado de Interbloqueos mediante Grafos

- Tipos de recursos: R1, R2, R3,.....
- Cada tipo de recurso posee Wi 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).

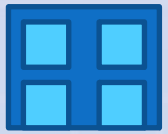
Usarlo : (USE) el proceso puede operar el recurso.

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

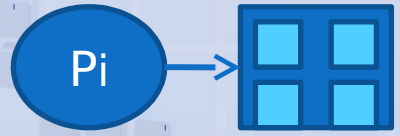
GRAFO DE ASIGNACIÓN DE RECURSOS (elementos)

- Procesos

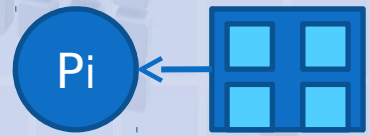
- Tipo de Recurso con 4 instancias



- Pi pide instancia of Rj



- Pi retiene una instancia de Rj



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),...,p(n)} conjunto de Procesos
  - R = {r(1),r(2),.....,r(m)} conjunto de Recursos

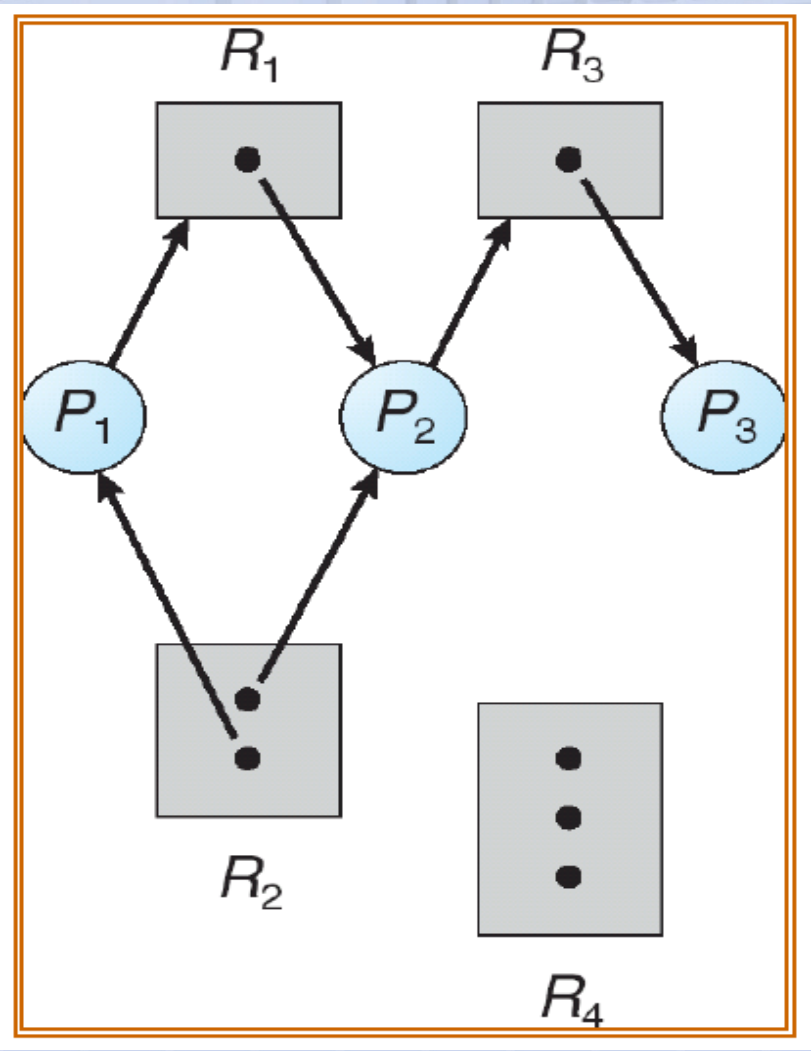
GRAFO DE ASIGNACION DE RECURSOS

Los elementos de E son:

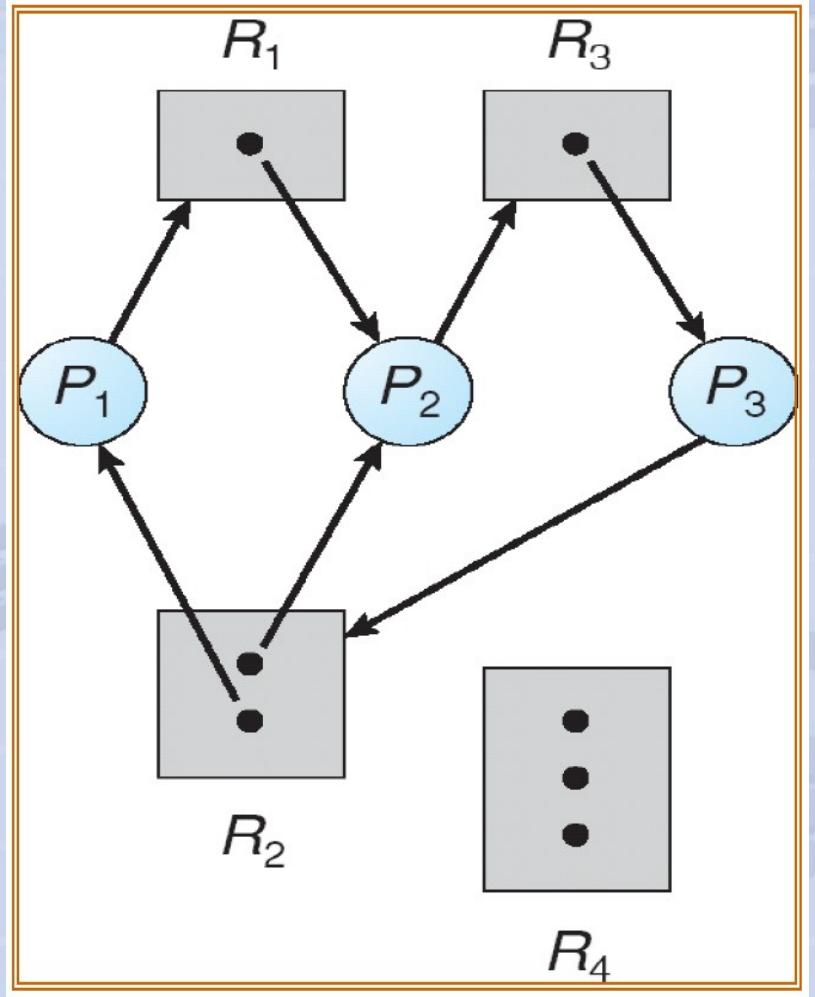
- (p(i),r(j)) =======> p(i) está esperando una instancia del recurso r(j).

- (r(j),p(i)) =======> r(j) (una instancia de) está asignada al proceso p(i).

Ejemplo de un grafo de asignación de recursos



Grafo de asignación de recursos con un Deadlock



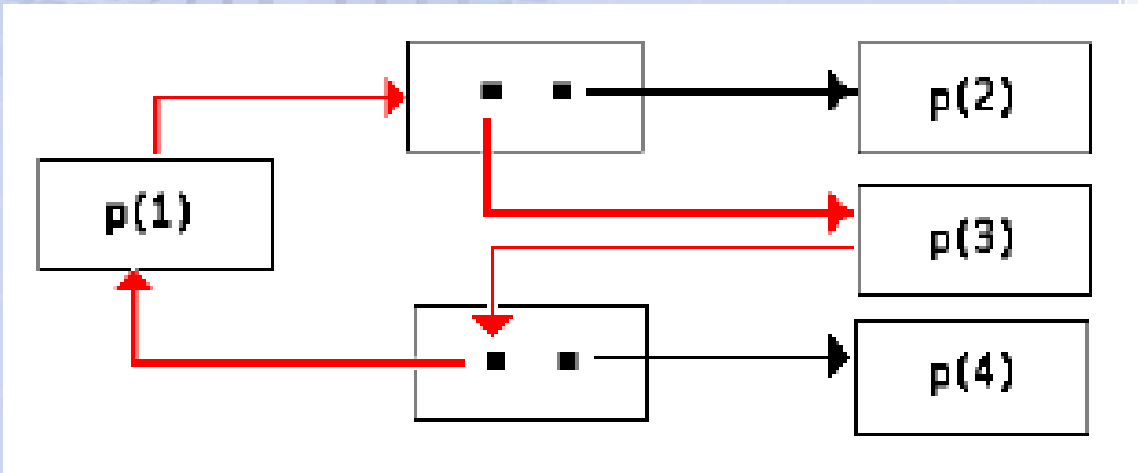
GRAFO DE ASIGNACIÓN DE RECURSOS

- 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 INTERBLOQUEO (DEADLOCK)

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

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

Interbloqueo- Algoritmo del Avestruz

El método más simple es el algoritmo de la avestruz:

meta su cabeza en la arena y pretenda que no hay ningún problema.

Las personas reaccionan a esta estrategia de diversas formas.

- Los **matemáticos** la encuentran totalmente inaceptable y dicen que los interbloqueos se deben prevenir a toda costa.

- Los **ingenieros** preguntan con qué frecuencia se espera el problema. Si ocurren interbloqueos en promedio de una vez cada cinco años, la mayoría de los ingenieros no estarán dispuestos a reducir considerablemente el

Interbloqueo- Algoritmo del Avestruz

Para que este contraste sea más específico, considere un sistema operativo que bloquea al proceso llamador cuando no se puede llevar a cabo una llamada al sistema en un dispositivo físico debido a que el dispositivo está muy ocupado. Si un proceso realiza un "open" exitosamente de la unidad de CD-ROM y otro la impresora, y después cada proceso trata de abrir el otro recurso y se bloquea en el intento, tenemos un in[illegible] Pocos sistemas actuales detectarán esto.

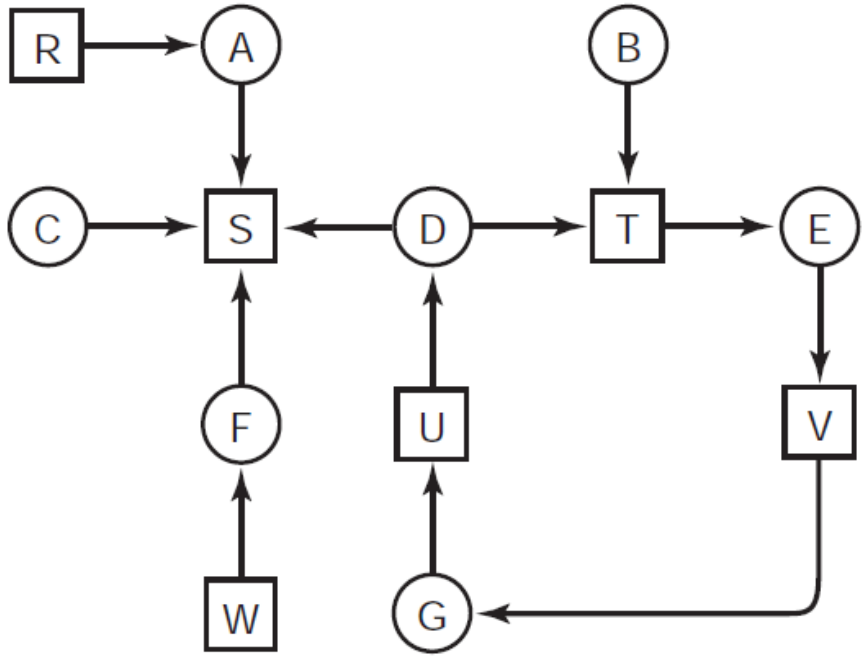
Interbloqueo - DETECCIÓN Y RECUPERACIÓN

Una segunda técnica es la detección y recuperación.

- Cuando se utiliza esta técnica, el sistema no trata de evitar los interbloqueos.
- En vez de ello, intenta detectarlos cuando ocurran y luego realiza cierta acción para recuperarse después del hecho.
- A continuación analizaremos algunas de las formas en que se pueden detectar los interbloqueos, y ciertas maneras en que se puede llevar a cabo una recuperación de los mismos.

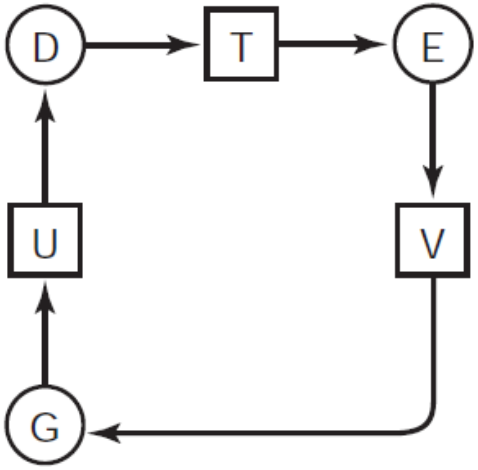
DETECCION DE **DEADLOCK**

- **Utilizando grafos**



(a)

(a) Un gráfico de recursos.



(b)

(b) Un ciclo extraído de (a).

DETECCION DE **DEADLOCK**

Algoritmo simple de detección (detección de ciclos)

L: Lista de nodos y lista de arcos

1. N como nodo inicial
2. Inicializar L vacía y arcos desmarcados
3. Nodo actual al fin de L

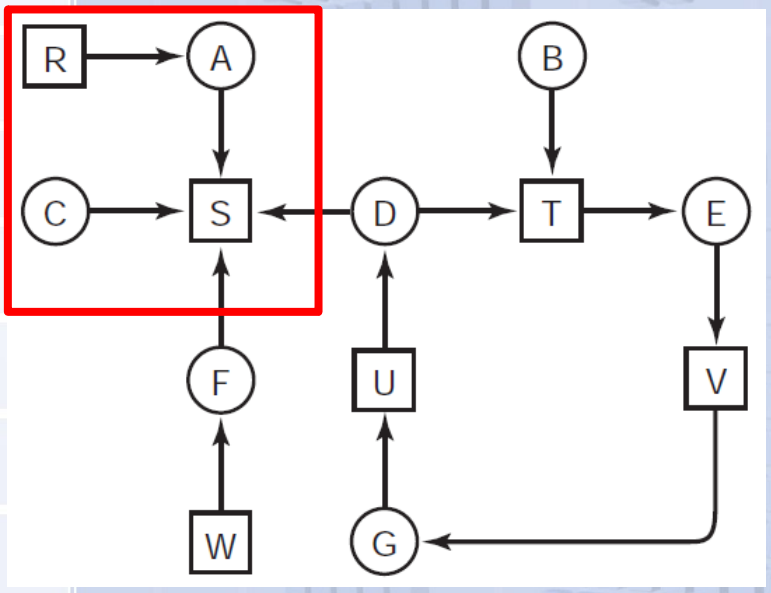
   ¿Nodo aparece dos veces en L? **Si ->FIN -> CICLO**

   No->Continúa
4. Del nodo actual :¿arcos salientes desmarcados?

   Si -> ir al paso 5 No -> ir al paso 6
5. Elegir arco saliente desmarcado y marcarlo. Paso 3
6. Si **¿nodo = inicial? => fin sin ciclos.**

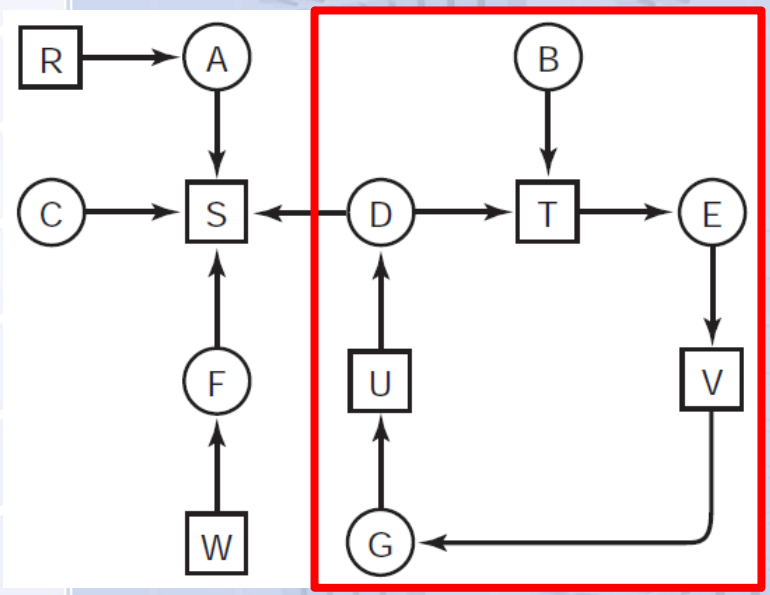
   ¿nodo ≠ inicial? => punto muerto retroceder nodo anterior

DETECCION DE **DEADLOCK**



- Inicio: R y L vacia.
- R-> L
- Avanzamos a A => L=[R,A]
- Avanzamos a S => L=[R,A,S]
- S no tiene arcos salientes =>Regresamos a A.
- A no tiene arcos salientes desmarcados => a R. FIN

DETECCION DE **DEADLOCK**



- Inicio: R y L vacia.
- R-> L
- Avanzamos a A => L=[R,A]
- Avanzamos a S => L=[R,A,S]
- S no tiene arcos salientes =>Regresamos a A.
- A no tiene arcos salientes desmarcados => a R. FIN

- Comenzamos con B.
- Se completa la lista hasta D => L = [B, T, E, V, G, U, D]
- Si elegimos S llegamos a un punto muerto y regresamos a D
- Si elegimos T y actualizamos la lista: L = [B, T, E, V, G, U, D, T]
- T se repite => Ciclo

DETECCION DE DEADLOCK

ALGORITMO DE SHOSHANI Y COFFMAN (1970)

- **Disponible :** Vector de longitud M. Disponible(j) = k, indica que existen k instancias disponibles del recurso r(j).

- **Asignación :**Matriz de NxM. Define el número de recursos de cada tipo actualmente tomados por cada proceso. Asignación(i,j,) = k, dice que el proceso p(i) tiene k instancias del recurso r(j).

- **Requerimiento o Espera:** Matiz de NxM. 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) ≠ 0 => Finish(i) = Falso sino
Finish(i) = Verdadero

P2. Encontrar un i tal que

a) Finish(i) = F

b) Requerimiento(i) ≤ 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

DETECCION DE DEADLOCK

ALGORITMO DE SHOSHANI Y COFFMAN (1970)

Ejemplo: Se tie[illegible] **7 de A, 2, d[illegible] y 6 de C**

Asignados			
P	A	B	C
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2

Requeridos		
A	B	C
0	0	0
2	0	2
0	0	0
1	0	0
0	0	2

Disponibles		
A	B	C
0	0	0

Para T_0

1. Work[M] = [0 0 0]
   Finish[N] = [F F F F]

2. P0

3. Work[M] = [0 1 0]
   Finish[N] = [V F F F]

2. P2

3. Work[M] = [3 1 3]
   Finish[N] = [V F V F]

No Deadlock

Secuencia
P0,P2,P1,P3,P4
Work[M] = [7 2 6]

DETECCION DE DEADLOCK

ALGORITMO DE SHOSHANI Y COFFMAN (1970)

Ejemplo: Se tie[illegible] **7 de A, 2, de B y 6 de C**

Asignados			
P	**A**	**B**	**C**
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2

Requeridos		
A	**B**	**C**
0	0	0
2	0	2
0	0	1
1	0	0
0	0	2

Disponibles		
A	**B**	**C**
0	0	0

Para T_1

1. Work[M] = [0 0 0]
   Finish[N] = [F F F F]

2. P0

3. Work[M] = [0 1 0]
   Finish[N] = [V F F F]

2. P2
   Req(i) ≤ Work ??? **NO**

Si bien se pueden asignar los recursos de P0, no es suficiente para el resto

Deadlock
Entre
P1, P2, P3 y P4

RECUPERACION FRENTE DEADLOCK (técnicas)

- **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** (apropiación de recursos)

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** (a través del retroceso)

- 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** (eliminación de procesos)

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.

Interbloqueo – Como evitarlo

En nuestro análisis de detección de interbloqueos hicimos la suposición de que cuando un proceso

pide recursos, los pide todos a la vez. *Sin embargo, en la mayoría de* los sistemas los recursos se solicitan uno a la vez. El sistema debe ser capaz de decidir si es seguro otorgar un recurso o si no lo es, y realizar la asignación sólo cuando sea seguro.

Por ende, surge la pregunta: ¿Hay algún algoritmo que siempre pueda evitar un interbloqueo al realizar la elección correcta todo el tiempo?

La respuesta es un sí calificado: podemos evitar los interbloqueos, pero sólo si hay cierta información disponible de antemano. A continuación examinaremos las formas de evitar el interbloqueo mediante una asignación cuidadosa de los recursos.

ALGORITMOS PARA EVITAR EL Interbloqueo

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. Disponible(j) = k, indica que existen k instancias disponibles del recurso r(j).
- **MAX :** Matriz de NxM. Define la máxima demanda de cada proceso.

MAX(i,j) = k, dice que p(i) requiere a lo sumo k instancias del recurso r(j).

- **Asignación :**Matriz de NxM. Define el número de recursos de cada tipo actualmente tomados por cada proceso.

Asignación(i,j,) = k, dice que el proceso p(i) tiene k instancias del recurso r(j).

- **Necesidad :**Matriz de NxM. Define el resto de necesidad de recursos de cada proceso.

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). Requerimiento (i)(j) = k, indica que p(i) requiere k instancias del recurso r(j).

Banquero y Seguridad

De lo cual se desprende que:

Necesidad(i,j) = MAX(i,j) - Asignación(i,j)

Para simplificar utilizaremos la notación siguiente:

Si X e Y son dos vectores:

X ≤ Y sii X(i) ≤ Y(i) ∀ i y

X < Y sii X ≤ Y , y X ≠ Y.

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

- Asignación(i) define todos los recursos asignados por el proceso p(i).
- Requerimiento(i) es el vector de requerimientos de p(i).

Banquero y Seguridad

Cuando se requiere un recurso se realizan primero estas verificaciones:

1. Si Requerimiento(i) ≤ Necesidad(i) seguir, sino ERROR (se pide más que lo declarado en MAX(i)).
2. Si Requerimiento(i) ≤ 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,.....,n (Procesos).

Finish(i) va marcando cuáles son los procesos ya controlados (si están en V) y bcon 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) Finish(i) = F y

b) Necesidad(i) ≤ Work

si no existe ese i , ir a Paso 4.

Paso 3.Work = Work + asignación(i)

Finish(i) = V (por Verdadero);

ir a Paso 2.

Paso 4. Si Finish(i) = V para todo i, el sistema está en estado "SAFE", o sea, encontramos la secuencia de procesos. Requiere MxNxN 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.

Asignación	A	B	C
p(0)	0	1	0
p(1)	2	0	0
p(2)	3	0	2
p(3)	2	1	1
p(4)	0	0	2

Disponible	A	B	C
	3	3	2

MAX	A	B	C
	7	5	3
	3	2	2
	9	0	2
	2	2	2
	4	3	3

Necesidad = MAX - Asig

A	B	C
7	4	3
1	2	2
6	0	0
0	1	1
4	3	1

Requerimiento de P(1)= <1, 0, 2 >

ALGORITMO DEL BANQUERO (ejemplo)

Para decidir que puedo garantizar la satisfacción de este requerimiento:

a) Veo que Req(i) ≤ Disponible ((1,0,2) ≤ (3,2,2))

b) Veo que Req(i) ≤ Necesidad ((1 0 2) ≤ (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 algoritmos de Seguridad

ALGORITMO DEL BANQUERO (ejemplo del algoritmo de seguridad)

Paso 1 WORK = < 2 3 0 > y Finish(i) = F para todo i

Paso 2 Existe i? Tomemos P1 porque < 0 2 0 > ≤ < 2 3 0 > y Finish(1)=F

Paso 3 WORK = WORK + ASIG P(1) = < 2 3 0 > + < 3 0 2 > = < 5 3 2>

Finish(1) = V

Paso 2 Existe i? Tomemos P3 porque < 0 1 1 > ≤ < 5 3 2 > y Finish(3)=F

Paso 3 WORK = WORK + ASIG P(3) = < 5 3 2 > + < 2 1 1 > = < 7 4 3>

Finish(3) = V

Paso 2 Existe i? Tomemos P4 porque < 4 3 1 > ≤ < 7 4 3 > y Finish(4)=F

Paso 3 WORK = WORK + ASIG P(4) = < 7 4 3 > + < 0 0 2 > = <

ALGORITMO DEL BANQUERO - Algunas consideraciones

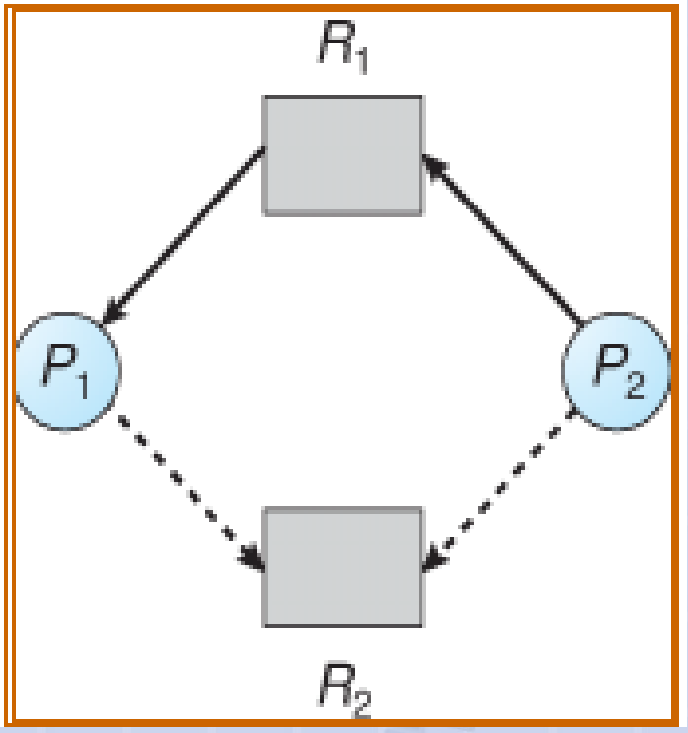
El algoritmo del Banquero impone fuertes restricciones al sistema ya que **solo permite asignar recursos si el estado resultante es seguro**. **Podrían asignarse** recursos pasando por **estados inseguros** y finalmente **no llegar** a entrar en deadlock pero **el Banquero no lo permite**.

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?

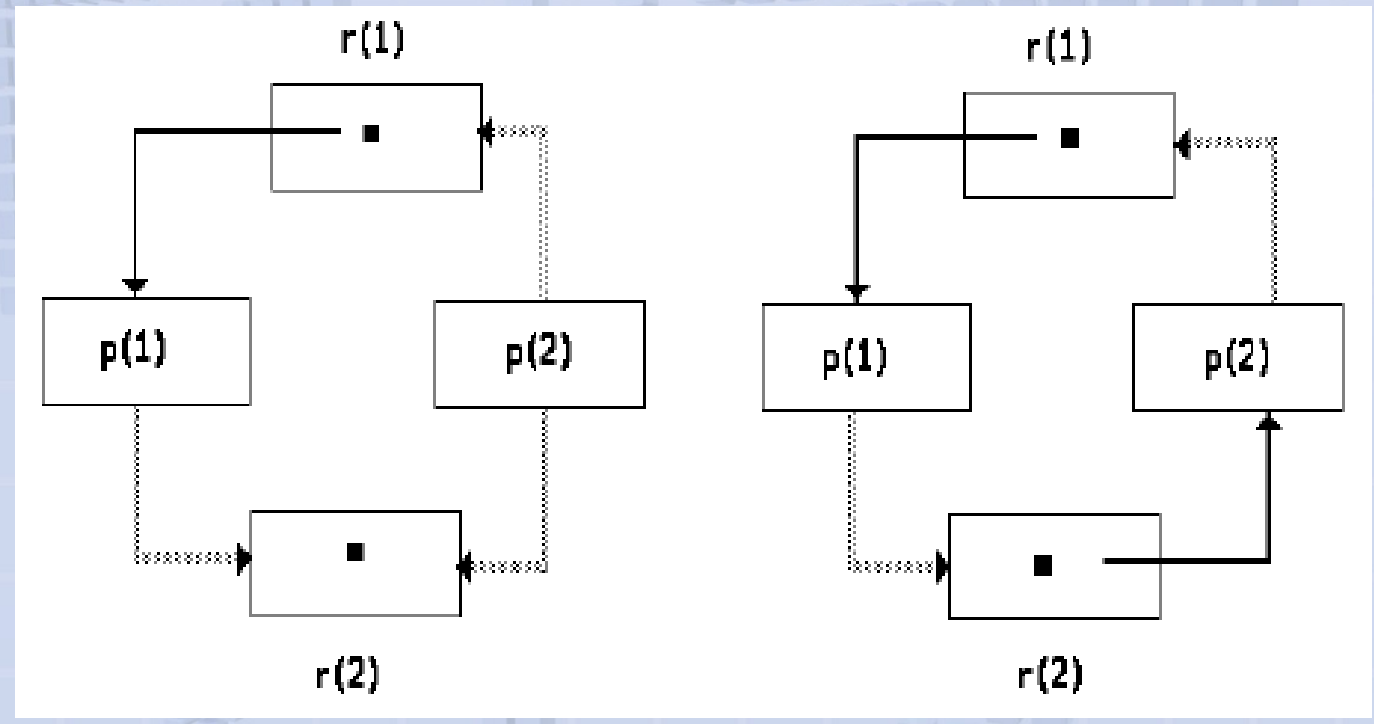
Algoritmo para recursos de una sola instancia

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.

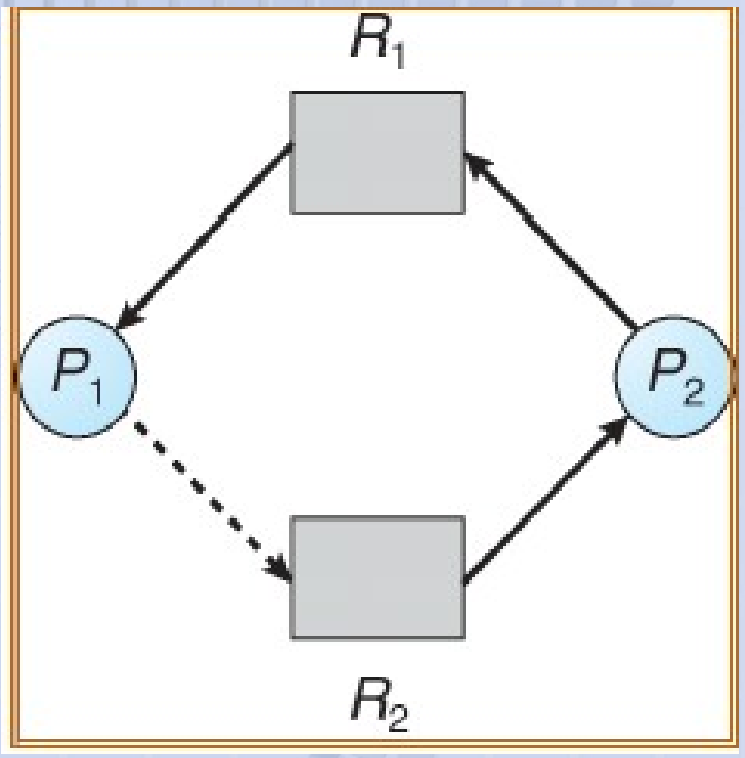


Algoritmo para recursos de una sola instancia



Algoritmo para recursos de una sola instancia

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



Interbloqueo – Como prevenirlo

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),....,r(n)}

F: R --> N

F(impresora)=1 , F(disco)=5 , F(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 órden 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 <p(1),p(2), ...,p(n)> 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

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

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

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

1° P(1) necesita 2 (hay disponibles 3). Luego libera 5
2° P(0) necesita 5 (hay disponibles 5). Luego libera 10
3° P(2) necesita 9 (hay disponibles 10)

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

Solo P(1) puede satisfacer sus requerimientos, aquí

no existe una secuencia segura, pues ni P(0), ni P(2)

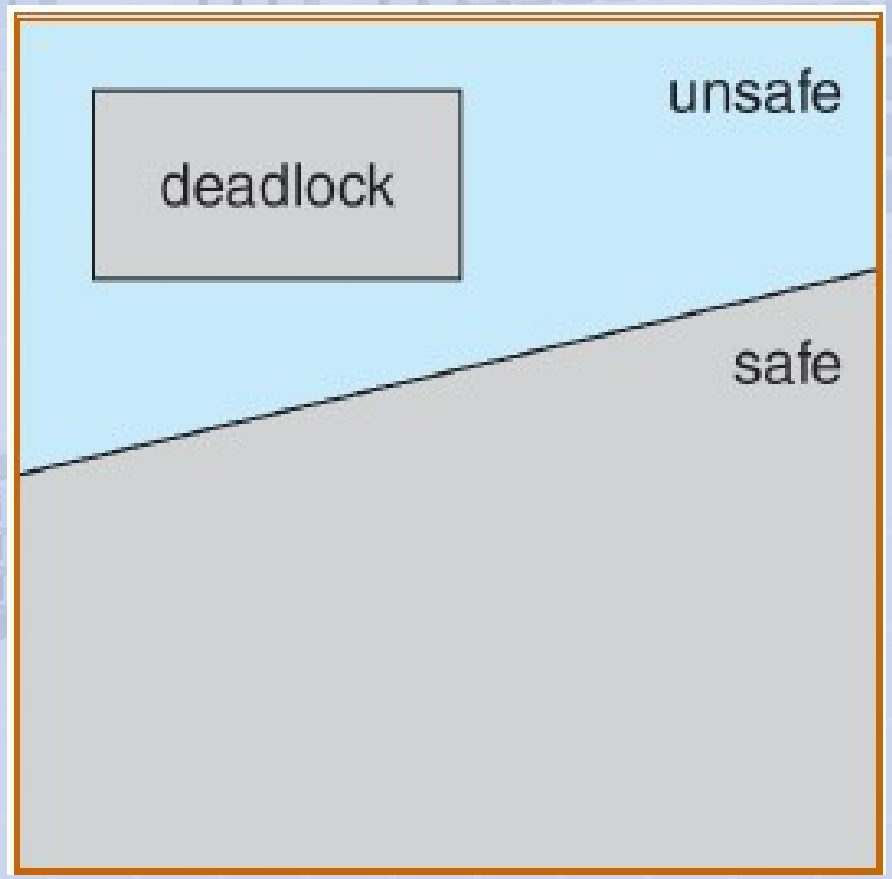
pueden obtener la totalidad de recursos que requieren, entonces podemos desembocar en un estado de "deadlock". Si se hubiera respetado la secuencia "segura", esto no se habría producido

Estado Seguro

Resumiendo:

- 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

Seguro, Inseguro , Deadlock



Bibliografía

Cap 6, tanembaum "Sistemas Operativos modernos"

Cap 7, Silberschatz "Fundamentos de sistemas operativos"