SISTEMAS OPERATIVOS I

UNIDAD III

BLOQUEOS IRREVERSIBLES (BI)

INSTITUTO TECNOLÓGICO DE MORELIA

Departamento de Sistemas y Computación

Ing. Benito Sánchez Raya

sanchezraya@hotmail.com

Disponible en: www.benito.org.mx

CONTENIDO

1. Recursos
2. Introducción a los Bloqueos Irreversibles (BI)
3. El algoritmo del avestruz
4. Detección de BI y recuperación posterior
5. Cómo evitar los BI
6. Prevención de BI
7. Otros aspectos

1. RECURSOS

- Bloqueo Irreversible (BI)
  - Un conjunto de procesos cae en un BI si cada proceso del conjunto está esperando un suceso que sólo otro proceso del conjunto puede causar.
  - Hay BI a nivel proceso o a nivel computadora
    - en una red
- Recurso
  - Puede ser:
    - Hardware: un dispositivo de E/S.
    - Software: Información (BD, archivos, registros, variables, etc)

1.1. Tipos de recursos

- Recursos expropiables
  - Es un recurso que se le puede quitar a quien lo tiene sin causarle daño.
  - Ejemplo:
    - La memoria se puede expropiar respaldándosela en disco a quien la tenia.
- Recursos no expropiables
  - No se pueden quitar a su dueño sin hacer que su cómputo falle.
  - Ejemplo:
    - Impresora, grabadora de CD/DVDs, etc.

- Normalmente los BI se dan en los recursos no expropiables.
- Secuencia de uso de un recurso
  - Solicita el recurso
  - Si esta disponible:
    - Lo usa
    - Lo libera
  - Si no esta disponible:
    - Devuelve un error
    - Se bloquea
    - Lo intenta más tarde

1.2. Adquisición de recursos

- Normalmente se hace con un semáforo por cada recurso o con mutexs.

```
typedef int semaforo;
semaforo recurso_1;

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

(a)

```
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);
}
```

(b)

Figura 3-1. Uso de un semáforo para proteger recursos. a) Un recurso. b) Dos recursos.

```
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);
    }
```

(a)

```
    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);
    }
```

(b)

Figura 3-2. a) Código sin bloqueos irreversibles. b) Código con un bloqueo irreversible potencial.

- El orden como se pidan los recursos es crucial para entrar o no, a un BI
  - Ver figura anterior.
- Si dos piden el mismo no se bloquea.

A → R A → S B → R B → S	A → R A ← R B → S B ← S A → S B → R	A → R B → R B ← R A → S B → S B ← S
Original	BI	No hay BI

2. INTRODUCCIÓN A LOS BI

- Para que sea un BI los procesos solo tienen un subproceso.
- No hay interrupciones que activen a un bloqueado.

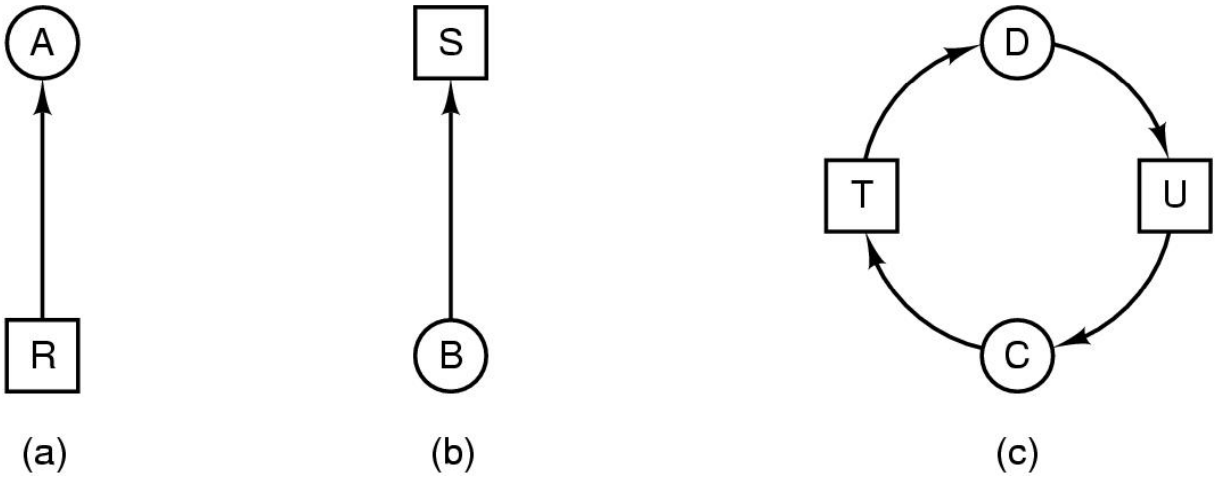
2.1. Condiciones para el BI

1. Exclusión mutua
   - Cada recurso esta asignado solo a un proceso, o esta libre.
2. Retención y espera
   - Los procesos que tienen recursos ya otorgados pueden solicitar otros.
3. De no expropiación
   - Los recursos ya otorgados no pueden arrebatarse, deben liberarse explícitamente.
4. Espera circular
   - Debe haber una cadena circular de dos o más procesos.

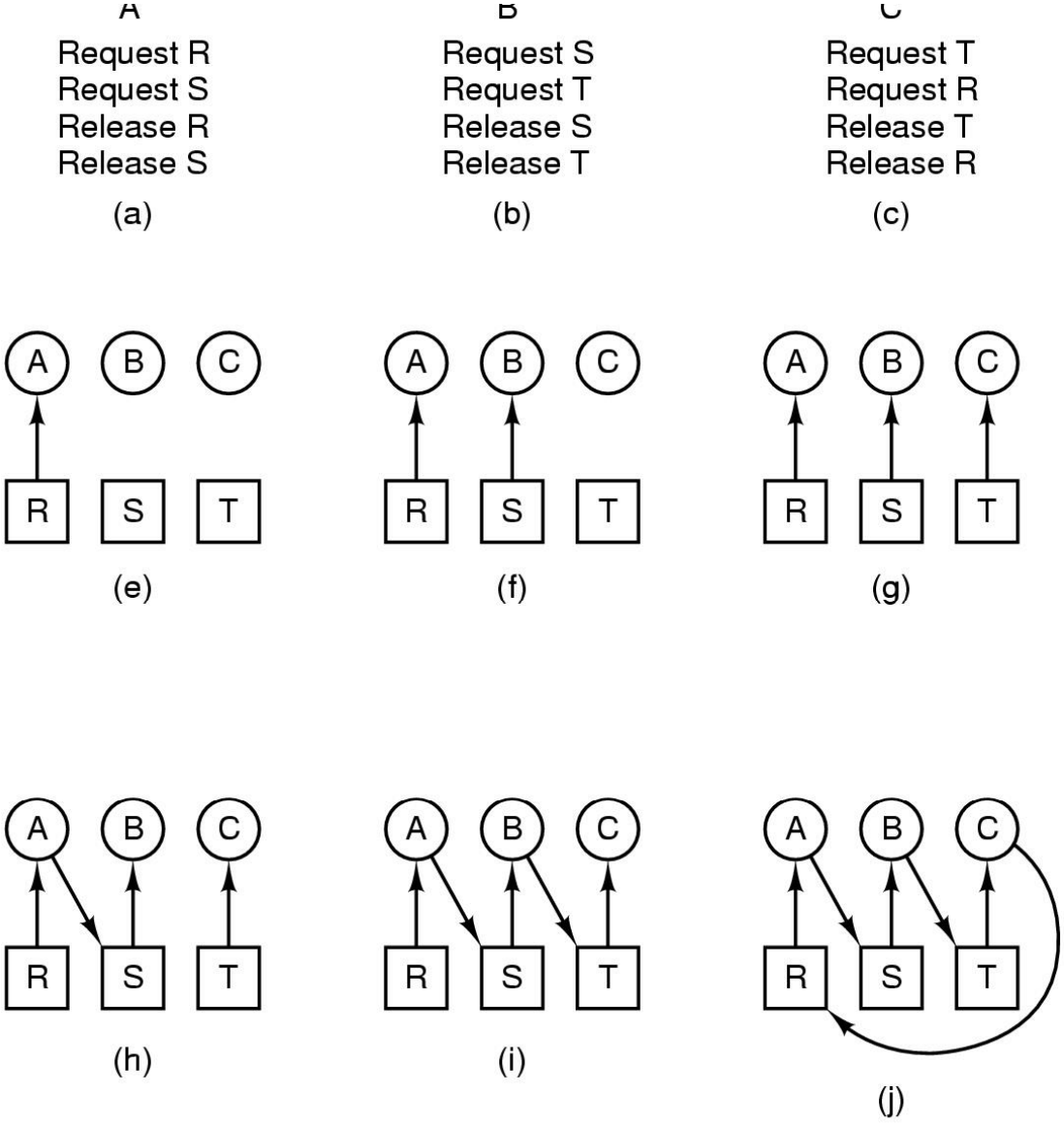
Deben ocurrir las cuatro para que haya un BI

2.2. Modelo de BI

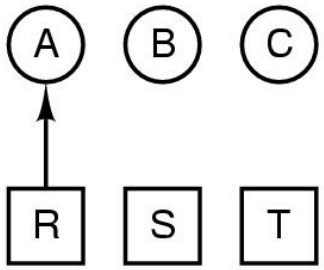
- Se representan con grafos.



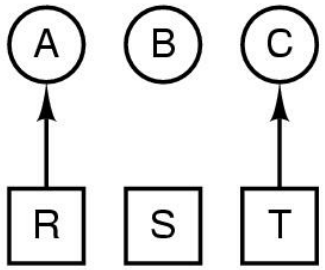
a) Posesión de un recurso b) Solicitud de un recurso c) Bloqueo Irreversible


A
Request R
Request S
Release R
Release S
(a)
B
Request S
Request T
Release S
Release T
(b)
C
Request T
Request R
Release T
Release R
(c)
1. A requests R
2. B requests S
3. C requests T
4. A requests S
5. B requests T
6. C requests R
deadlock
(d)
A B C
R S T
(e)
A B C
R S T
(f)
A B C
R S T
(g)
A B C
R S T
(h)
A B C
R S T
(i)
A B C
R S T
(j)

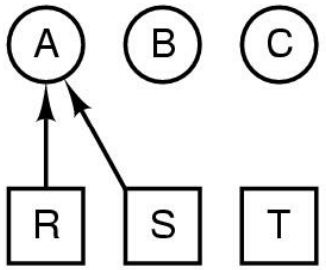
1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S
   no deadlock



(k)

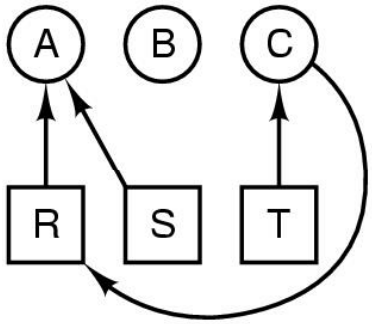


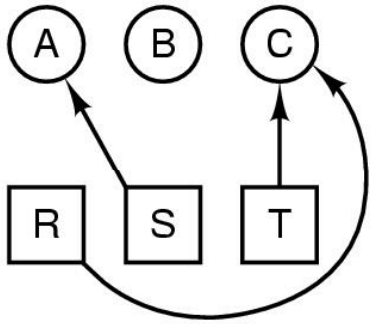
(l)

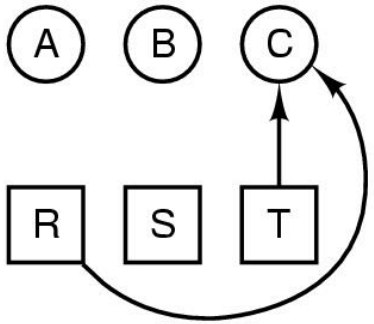


(m)

(n)







- **Estrategias para el manejo de BI**
  1. Ignorar el problema
  2. Dejar que sucedan, detectarlos y recuperarse de ellos.
  3. Evitar que sucedan, con una asignación cuidadosa de recursos
  4. Prevenirlos, anulando una de las cuatro condiciones para un BI.

3. ALGORITMO DEL AVEZTRUZ

- En un sistema real ya funcionando no son tan comunes los BI.
- La mayoría de los SO pueden tener BI que ni siquiera se detectan.
- Se tienen limitantes que pueden originar BI:
  a) La cantidad de procesos y subprocesos activos al mismo tiempo.
  b) Número máximo de archivos abiertos
  c) El espacio de intercambio en disco.
- Windows y Unix los ignoran, prefieren un BI de vez en cuando a que haya limitantes con los recursos.

4. DETECCIÓN DE BI Y RECUPERACIÓN POSTERIOR

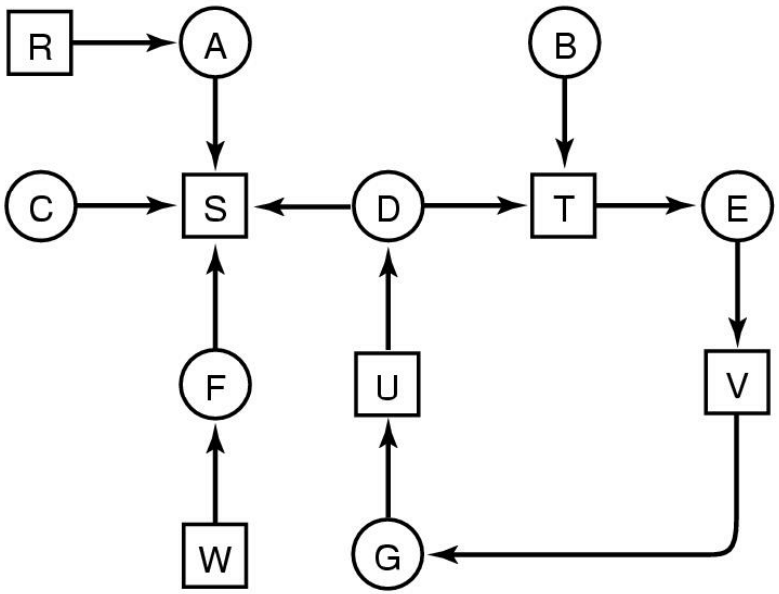
- No se intenta prevenir, solo esperar a que suceda, detectarlo y recuperarse del bloqueo.

4.1. Detección de BI con un recurso de cada tipo

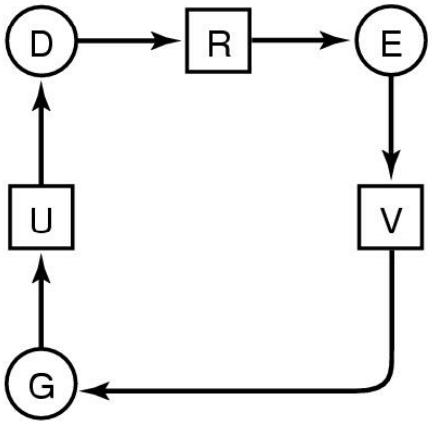
- Solo hay un recurso de cada tipo
- Se construye un grafo, si hay uno o más ciclos, existe un BI de lo contrario no lo hay.

- Ejemplo:
  - Son 7 procesos (A, B, C, D, E, F, G)
  - Son 6 recursos (R, S, T, U, V, W)
    1. A tiene a R, quiere S
    2. B quiere a T
    3. C quiere a S
    4. D tiene a U, quiere S y T
    5. E tiene T, quiere V
    6. F tiene W, quiere S
    7. G tiene V, quiere U

- El grafo quedaría:



(a)



(b)

- Visualmente es fácil observar un BI.
- Algoritmo formal para detectar BI
  - Usa L = Estructura de datos que es una lista de nodos.
  - Se van marcando los arcos analizados.
  - Se inicia haciendo el recorrido a partir de cualquier nodo, y todos deben analizarse.
  - Si pasa por un nodo previamente visitado, existe un ciclo, por lo que hay un BI.
  - Si llega un nodo sin salida, se regresa al nodo anterior.
  - Si hay dos salidas, al azar se toma una.
  - Si al ir regresando, retorna a la raíz, quiere decir que no hay ciclos.

- Para el grafo anterior:
  - Para R quedaría:
    - L = {R}, {R, A}, {R, A, S}, se retorna para ver si hay más salidas {R, A}, {R}
  - Para B quedaría:
    - L = {B}, {B, T}, {B, T, E}, {B, T, E, V}, {B, T, E, V, G}, {B, T, E, V, G, U}, {B, T, E, V, G, U, D}, {B, T, E, V, G, U, D, S}, {B, T, E, V, G, U, D}, {B, T, E, V, G, U, D, T}, hay BI porque se repite T.

4.2. Detección de BI con múltiples recursos de cada tipo

- n = Número de procesos
- m = Tipos distintos de recursos
- E = Vector de recursos existentes
- A = Vector de recursos disponibles
- C = Matriz de asignación actual ($C_{proceso,\ recurso}$)
- R = Matriz de solicitudes ($R_{proceso,\ recurso}$)

Resources in existence
$(E_1, E_2, E_3, \ldots, E_m)$

Resources available
$(A_1, A_2, A_3, \ldots, A_m)$

Current allocation matrix

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix}$$

Row n is current allocation to process n

Request matrix

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

Row 2 is what process 2 needs

- Se debe cumplir: $\sum_{i=1}^{n} C_{ij} + A_j = E_j$
- Se basa en la comparación de vectores.
  - A y B son vectores
  - A ≤ B indica que cada elemento de A es menor o igual que su correspondiente en B.
- Ningún proceso esta marcado en un principio
- Al ir avanzando el algoritmo, se van marcando los procesos.
- Al finalizar el algoritmo los procesos que no estén marcados están en un BI.

- Algoritmo:
  1. Buscar un proceso no marcado, Pi, para el cual la i-enésima fila de R sea menor o igual que A.
  2. Si se haya tal proceso, añadir la i-enésima fila de C a A, marcar el proceso y regresar al paso 1.
  3. Si no existe tal proceso, el algoritmo termina.

- Es decir, se escoge un proceso el cual pueda ejecutarse hasta terminar, con los recursos disponibles.
- Al terminar de ejecutarse libera los recursos y marca el proceso como terminado.
- Si al final todos se pueden ejecutar, no hay BI.
- Si algunos procesos no se pueden marcar hay un BI.

Tape drives Plotters Scanners CD Roms

E = (4 2 3 1)

Tape drives Plotters Scanners CD Roms

A = (2 1 0 0)

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

- En la figura anterior:
  - El primero y segundo proceso no se puede satisfacer.
  - El tercer proceso si se puede satisfacer
    - Termina y libera sus recursos, quedando A=(2 2 2 0)
  - Ahora sí, el segundo proceso se puede ejecutar y liberar todos sus recursos, A=(4 2 2 1)
  - Y ahora si se ejecuta el proceso restante
  - No hay BI.
- Cada cuando ejecutar los algoritmos de detección de BI
  - Cada que se pida un proceso
  - Cada ciertos minutos
  - Cada que el rendimiento del CPU baje de cierto umbral

4.3. Cómo recuperarse de un BI

1. Mediante expropiación
   - Consiste en arrebatarle el recurso a un proceso, y después reanudarlo.
   - Depende del recurso, de preferencia un recurso expropiable.
2. Por eliminación de procesos
   - Más burda pero eficaz
   - Elimina un proceso, si no lo rompe, se elimina otro y así sucesivamente.
   - Podría ser uno que no este en el ciclo, uno de baja prioridad, uno que pueda reiniciarse, etc.

3. Mediante reversión

- Estableciendo puntos de verificación
- Deben ser archivos nuevos constantemente creados.
- El punto de verificación contiene:
  - La memoria (pila, registros, etc)
  - Estado de los recursos y procesos
- Al detectar un BI se busca un punto de verificación antes de haber solicitado el recurso para reanudar a partir de ahí.

5. CÓMO EVITAR LOS BI

- Objetivo: Evitar los BI mediante la asignación cuidadosa de recursos.
- Sin tener que darle todos los recursos que requiere, como el método anterior.

5.2. Estados seguros e inseguros

- Estado seguro
  - Cuando no se ha caído en un BI, y existe algún orden de calendarización en el cual todos los procesos puedan ejecutarse hasta terminar.
  - Ejemplo:
    - Con 10 instancias.

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

(b)

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5

(c)

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0

(d)

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7

(e)

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

(d)

- Este sería un estado inseguro.
- En un estado inseguro no se ofrece garantía que los procesos terminen.
- Un estado inseguro no necesariamente es un BI

5.3. Algoritmo del banquero para un solo recurso

- Se examina cada solicitud en el momento que se hace, y se analiza si esto llevará a un estado seguro, si es así lo concede sino lo pospone.
- Para ver si un estado es seguro:
  - Checa si hay suficiente recursos para satisfacer el proceso.
  - Verifica que proceso (cliente) esta más cercano al limite.

- Ejemplo: 10 instancias de las 22 requeridas.

  c) podría ser un estado inseguro mas no un BI.

  porque algún proceso podría liberar algunos

  Un BI podría ser cuando todos los procesos solicitan el total de recursos.

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

5.4. Algoritmo del banquero para múltiples recursos

C

Process	Tape drives	Plotters	Scanners	CD ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Recursos asignados

R

Process	Tape drives	Plotters	Scanners	CD ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Recursos que aún se necesitan

E = (6342)
P = (5322)
A = (1020)

E=Existentes
P=Poseídos
A=Disponibles

- Igual que el caso de un solo recurso, se requiere conocer las necesidades totales de los procesos antes de ejecutarse.
- Requiere que la cantidad de procesos sea fija.
- Los recursos pueden desaparecer/aparecer “en caliente”
- No se usa en la práctica

- Algoritmo para ver si un estado es seguro o inseguro:
  1. Buscar una fila R, cuyas necesidades de recursos insatisfechas sean menores o iguales que el vector A. Si no existe tal fila, en algún momento el sistema caerá en un BI porque ningún proceso puede terminar.

     Si hay mas de una, escoger cualquiera.
  2. Suponer que el proceso de la fila escogida solicita todos los recursos que necesita y termina. Este proceso se marca como terminado y todos sus recursos se suman al vector A.
  3. Repetir 1 y 2, hasta que todos los procesos se hayan marcado como terminados (lo que quiere decir que el estado inicial era seguro) o hasta que haya un BI.

- A la figura anterior:
  - El estado actual es seguro
  - Supongamos que el proceso B solicita un escáner
    - Sería un estado seguro, termina D, A o E, etc.
  - Supongamos que el proceso B solicita una escáner y E el otro.
    - Conduciría a un BI.
    - Por lo que la solicitud de E debería aplazarse.

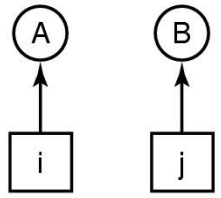
6. PREVENCIÓN DE BI

- Como los mecanismos para evitar los BI son poco irreales y es difícil evitar caer en un BI, se opta por la prevención
- Consiste en evitar que al menos una de las cuatro condiciones básicas no se cumpla.
  a) Exclusión mutua:
     - Un recurso está asignado a un proceso o libre.
  b) Retención y espera
     - Si ya tienen procesos pueden pedir más.
  c) De no expropiación
     - No pueden arrebatárseles, deben liberarlos explícitamente.
  d) De espera circular
     - Cadena circular de 2 o más procesos.

- Exclusión mutua
  - No asignar en forma exclusiva todos los recursos
    - Hacerlo a través de spooling
  - Asignar un recurso hasta que sea estrictamente necesario.
  - Que la cantidad de procesos que soliciten cierto recurso sea la menor posible.
- Retención y espera
  - Exigir que todos los procesos pidan al inicio todos los recursos que requerirán (casi nunca se puede y no es óptimo, excepto en algunos procesamientos por lotes)
  - Otra opción sería que antes de pedir un recurso suelte todos e intente obtenerlos todos a la vez (con el nuevo).

- De no expropiación
  - Arrebatar los recursos
  - En recursos no expropiables es imposible.
- Condición de espera circular
  - Que un proceso solo pueda tener un recurso a la vez, al solicitar uno, que suelte el otro (imposible)
  - Numerar los recursos, y que los procesos al solicitarlos los vayan tomando en orden ascendente.
    - Si tiene el 3, puede pedir uno mayor al 3.
    - Con esto se evitarían ciclos, porque estará accediendo a recursos libres.
  - Sería difícil encontrar un orden que satisfaga a todos los procesos.

1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD Rom drive



7. OTROS ASPECTOS

- Bloqueos de dos fases
  - Sobre todo en base de datos

  1ª Fase: Bloqueo de registros

  Hasta que pueda bloquearlos todos.

  2ª Fase: Actualización
- BI que no son por recursos
  - Puede ser sólo entre procesos
    - Uno espere a que otro lo “despierte”
    - Caso Productor-Consumidor con despertar perdido

- Inanición
  - Puede haber procesos que no están en un BI y aun así nunca sean atendidos, por lo que mueren de inanición.
  - Una solución es usar un algoritmo de calendarización que lo evite.

REFERENCIA:

- Sistemas Operativos Modernos, Segunda Edición

  TANENBAUM

  Prentice Hall