Sistemas Operativos I

Procesos
Semáforos y Monitores
Interbloqueo

Edwin Salvador

19 de noviembre de 2015

Sesión 8

Contenido I

Semáforos

- Los semáforos ayudan a que dos procesos puedan cooperar por medio de simples señales.
- Así sería posible detener un proceso en un lugar específico hasta que reciba una señal específica.
- Para esta señalización se utilizan variables especiales llamadas semáforos. Así, para transmitir una señal vía el semáforo **s**, el proceso ejecutará la primitiva **semWait(s)**; El proceso se suspenderá hasta que la transmisión se realice.

Contenido I

Semáforo general o con contador

- Para lograr el efecto deseado, el semáforo se puede ver como una variable que contiene un entero sobre el cual sólo están definidas **3 operaciones:**
  1) Un semáforo puede ser inicializado a un valor **no negativo.**
  2) La operación **semWait** decrementa el valor del semáforo. Si el valor pasa a ser negativo, entonces el proceso que está ejecutando semWait se bloquea. En otro caso, el proceso continúa su ejecución.
  3) La operación **semSignal** incrementa el valor del semáforo. Si el valor es menor o igual que cero, entonces se desbloquea uno de los procesos bloqueados en la operación semWait.

Definición de las primitivas del semáforos

```
struct semaphore {
    int cuenta;
    queueType cola;
}
void semWait(semaphore s)
{
    s.cuenta—;
    if (s.cuenta < 0)
    {
        poner este proceso en s.cola;
        bloquear este proceso;
    }
}
void semSignal(semaphore s)
{
    s.cuenta++;
    if (s.cuenta <= 0)
    {
        extraer un proceso P de s.cola;
        poner el proceso P en la lista de listos;
    }
}
```

Contenido I

Semáforos binarios

- Es una versión más restringida de los semáforos. También se lo conoce como **mutex**.
- Solo puede tomar valores 0 y 1 y está definido por las siguientes 3 operaciones:
  1) Puede ser inicializados a 0 o 1.
  2) La operación **semWaitB** comprueba el valor del semáforo. Si el valor es cero, entonces el proceso que está ejecutando semWaitB se bloquea. Si el valor es uno, entonces se cambia el valor a cero y el proceso continúa su ejecución.
  3) La operación **semSignalB** comprueba si hay algún proceso bloqueado en el semáforo. Si lo hay, entonces se desbloquea uno de los procesos bloqueados en la operación **semWaitB**. Si no hay procesos bloqueados, entonces el valor del semáforo se pone a uno.

Definición de las primitivas del semáforo binario.

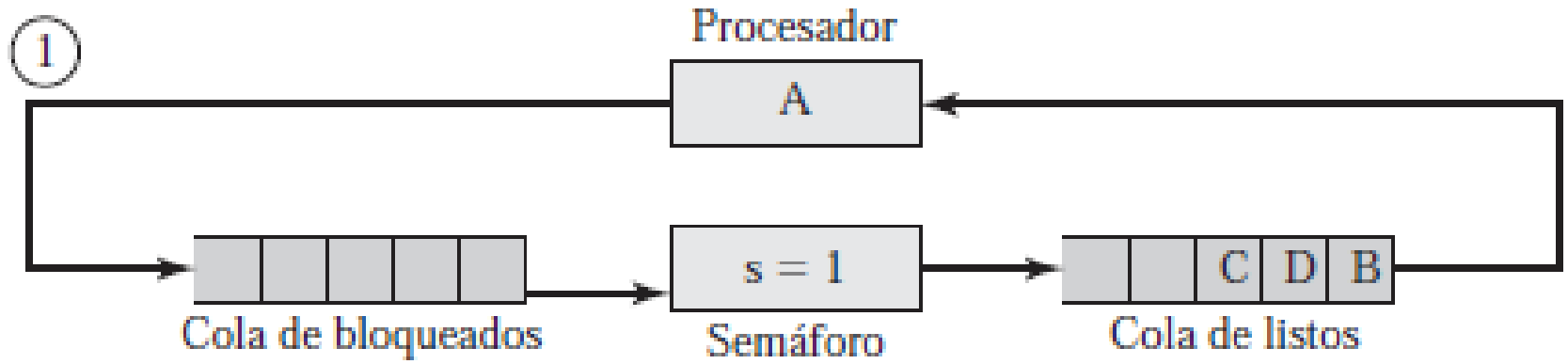
```
struct binary_semaphore {
    enum {cero, uno} valor;
    queueType cola;
};
void semWaitB(binary_semaphore s)
{
    if (s.valor == 1)
        s.valor = 0;
    else
        {
            poner este proceso en s.cola;
            bloquear este proceso;
    }
}
void semSignalB(binary_semaphore s)
{
    if (esta_vacia(s.cola))
        s.valor = 1;
    else
    {
            extraer un proceso P de s.cola;
            poner el proceso P en la lista de listos;
    }
}
```

Semáforos

- Para ambos tipos de semáforos, general o binario, se utiliza una cola para mantener los procesos esperando por el semáforo.
- La política más favorable para saber que proceso se debe extraer de la cola es FIFO, el proceso que lleve más tiempo bloqueado es el primero en ser extraído de la cola.
- Un semáforo que incluya esta política se lo llama **semáforo fuerte**. Si un semáforo no especifica el orden que los procesos deben salir de la cola se lo llama **semáforo débil**.

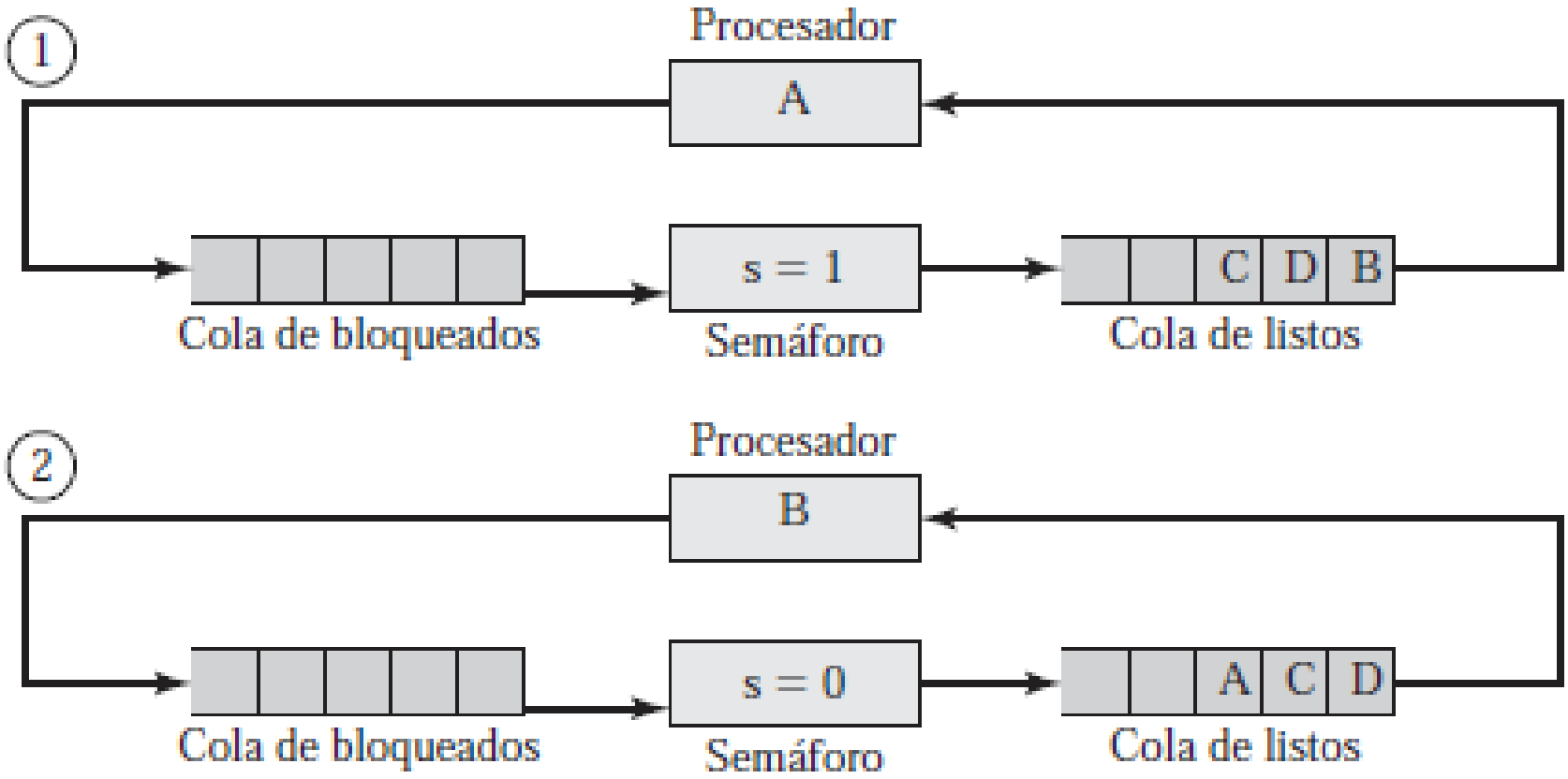
Ejemplo de mecanismo de semáforo fuerte

- Supongamos que tenemos 4 procesos A, B, C y D.
- A, B y C dependen de un resultado de D.
- Inicialmente A está ejecutando; B, C y D están listos; y el contador del semáforo está en 1 lo que indica que uno de los resultados de D está disponible.



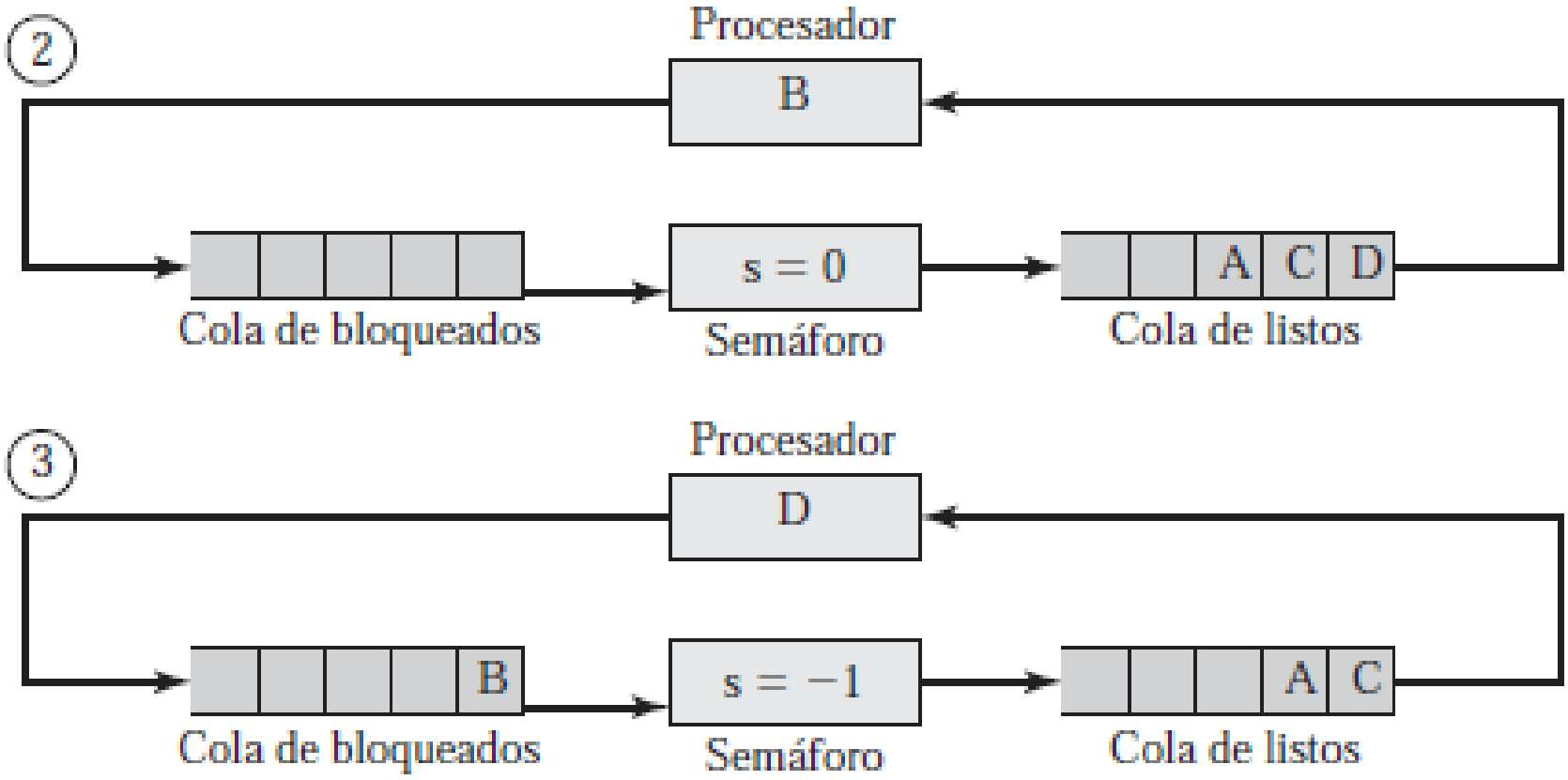
Ejemplo de mecanismo de semáforo fuerte

- A realiza una operación **SemWait** sobre **s** (decrementa a 0) y A puede continuar su ejecución y luego se adjunta a la cola de listos.



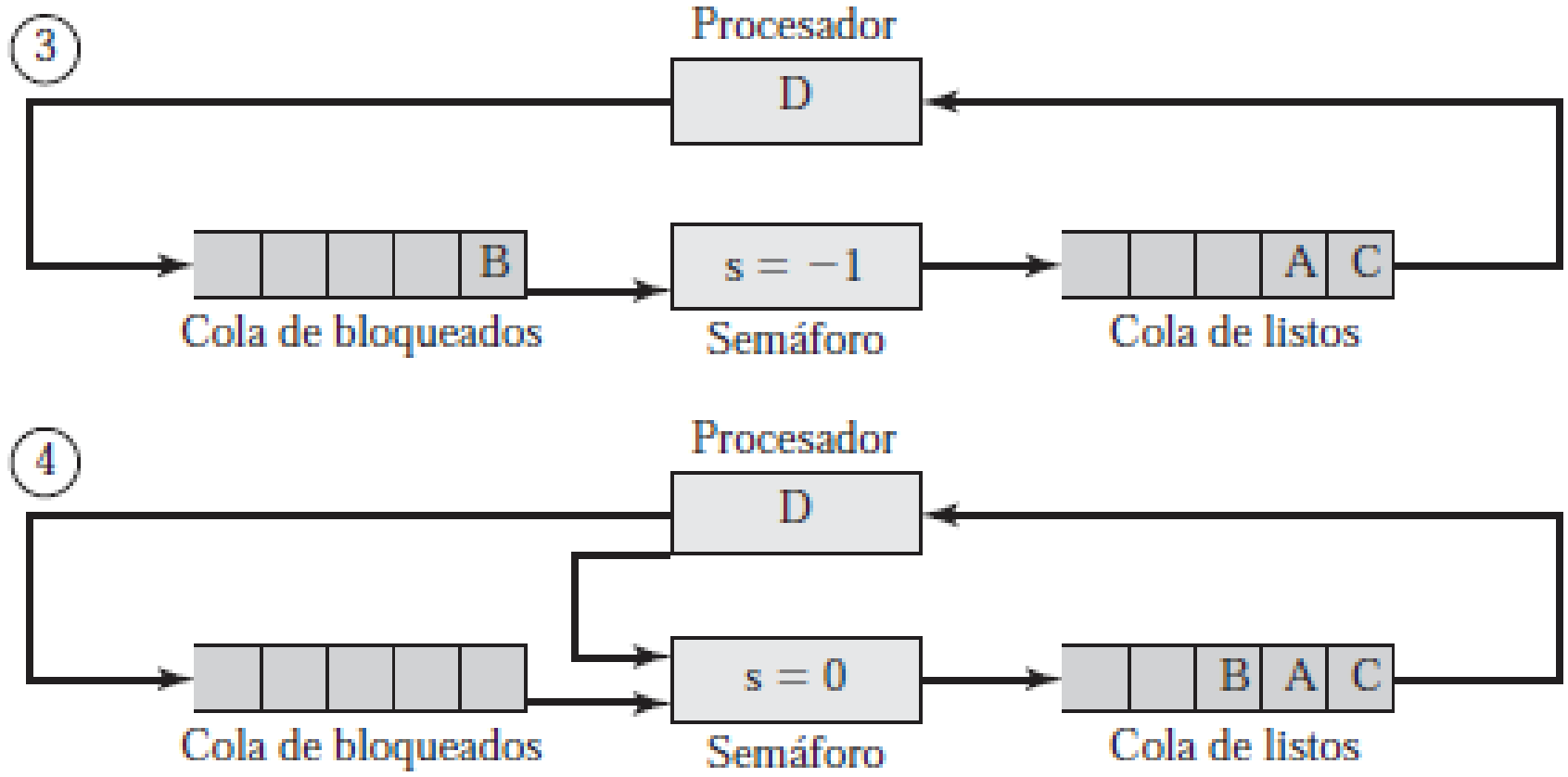
Ejemplo de mecanismo de semáforo fuerte

- B se ejecuta y finalmente realiza una operación **semWait** y es bloqueado. Esto permite que D ejecute.



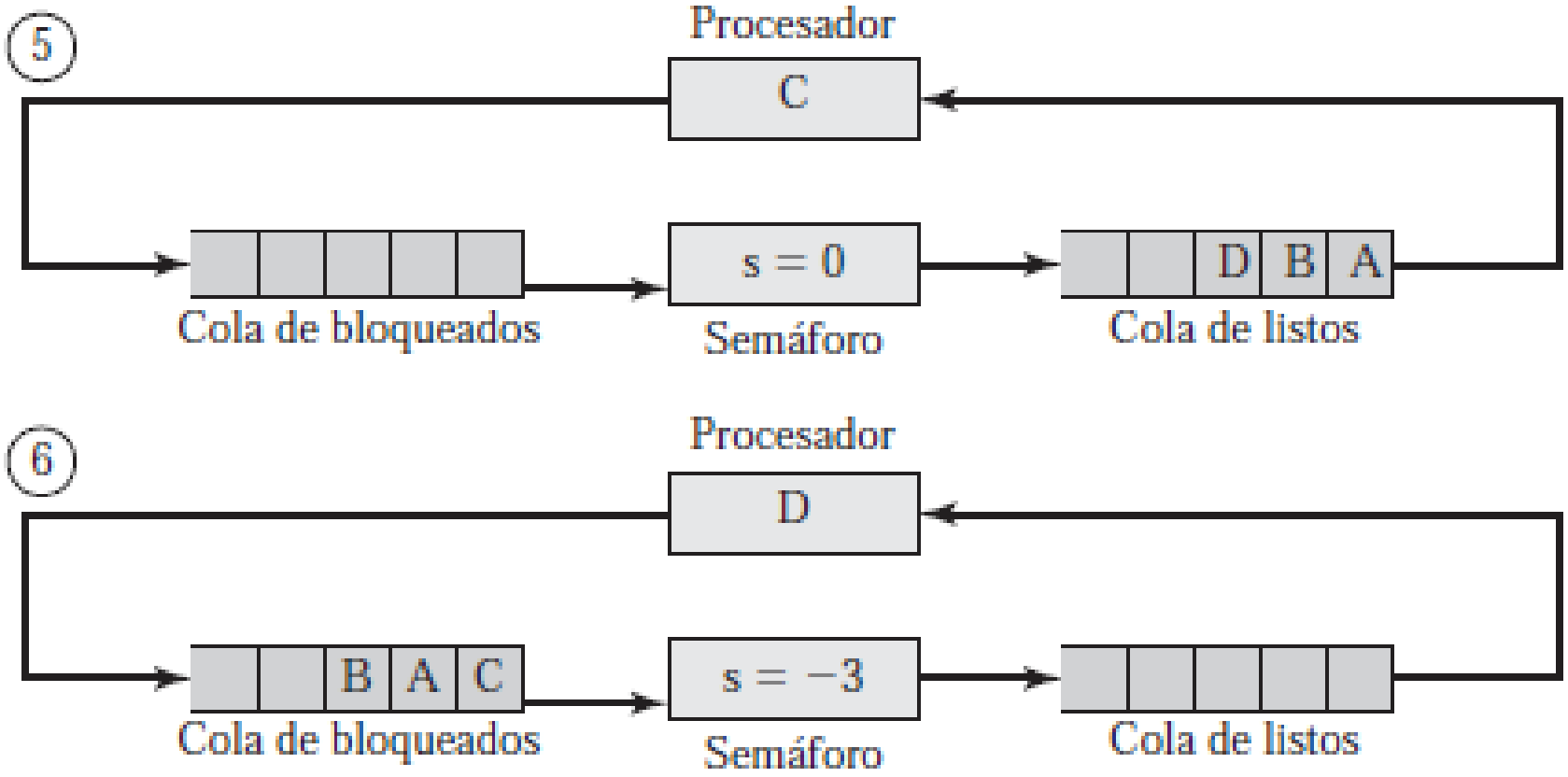
Ejemplo de mecanismo de semáforo fuerte

- Cuando D completa, existe un nuevo resultado disponible y realiza una instrucción **semSignal**.
- Esto permite a B moverse a la cola de listos.



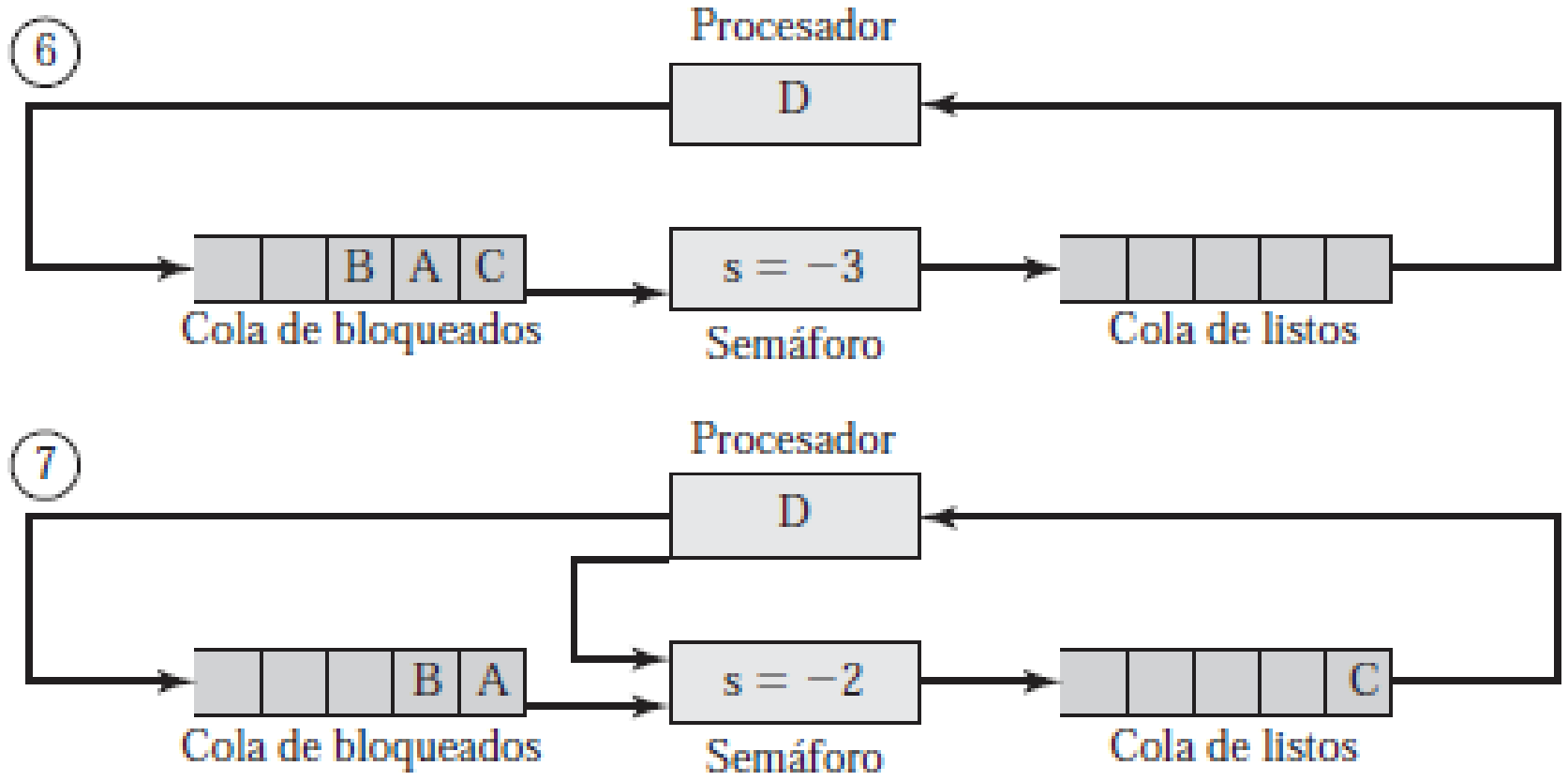
Ejemplo de mecanismo de semáforo fuerte

- Luego D se adjunta a la cola de listos y esto permite que C ejecute.
- C ejecuta una instrucción **semWait** y se bloquea. De igual manera pasa con A y B.



Ejemplo de mecanismo de semáforo fuerte

- Esto permite que D retoma la ejecución y cuando tenga otro resultado disponible, ejecute una instrucción **SemSignal**.
- Así, C pasa a la lista de listos. Los siguientes ciclos liberarán a A y B del estado bloqueado.



Contenido I

Solución al problema de exclusión mutua usando semáforos

- Consideremos *n* procesos identificados como *P(i)*, los cuales necesitan acceder al mismo recurso. Cada proceso tiene una **sección crítica** que accede al recurso.
- Cada proceso ejecuta un **semWait(s)** justo antes de entrar en su sección crítica.
- Si el valor de **s** pasa a ser negativo, el proceso se bloquea. Si el valor es 1, entonces se decrementa a 0 y el proceso entra en su sección crítica inmediatamente.
- Dado que **s** ya no es positivo, ningún otro proceso será capaz de entrar en su sección crítica.

Solución al problema de exclusión mutua usando semáforos

- El semáforo se inicializa a 1. Así, el primer proceso que ejecute un semWait será capaz de entrar en su sección crítica inmediatamente, poniendo el valor de s a 0.
- Cualquier otro proceso que intente entrar en su sección crítica la encontrará ocupada y se bloqueará, poniendo el valor de s a -1. Si otros procesos intentan entrar, esto decrementará el valor de **s**.
- Cuando el proceso que inicialmente entró en su sección crítica salga de ella, **s** se incrementa y uno de los procesos bloqueados (si hay alguno) se extrae de la lista de procesos bloqueados asociada con el semáforo y se pone en estado Listo. Cuando sea planificado por el sistema operativo, podrá entrar en la sección crítica.

Semáforos

- Los semáforos proporcionan una herramienta potente y flexible para conseguir exclusión mutua y para la coordinación de procesos.
- Puede ser difícil producir un programa correcto utilizando semáforos por que las operaciones semWait y semSignal pueden estar dispersas a través de un programa y resulta difícil ver el efecto global de estas operaciones sobre los semáforos a los que afectan.

Contenido I

Monitores

- Un monitor es un módulo software consistente en uno o más procedimientos, una secuencia de inicialización y datos locales.
- Proporcionan una funcionalidad equivalente a la de los semáforos pero es más fácil de controlar.
- Ciertos lenguajes de programación tienen monitores implementados como bibliotecas para que sean usadas directamente por el programador.

Contenido I

Monitor con señal

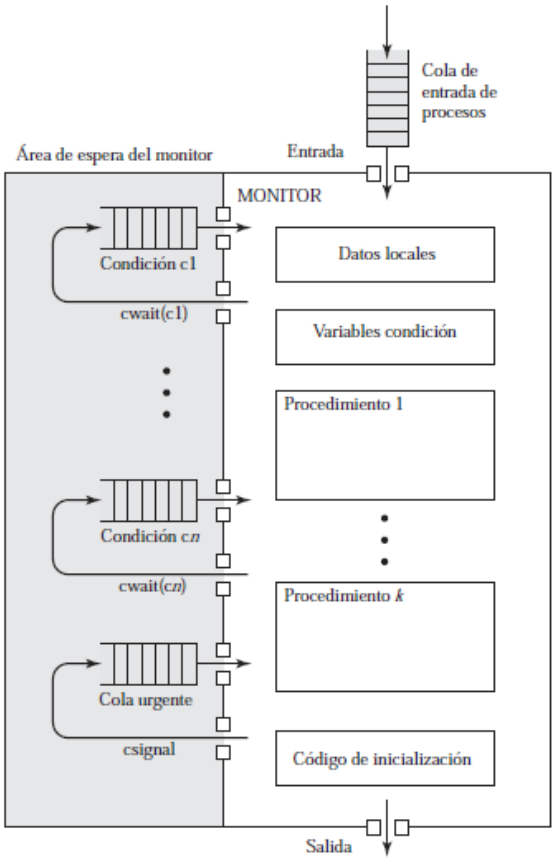
- Las principales características de un monitor son las siguientes:
  1) Las variables locales de datos son sólo accesibles por los procedimientos del monitor y no por ningún procedimiento externo.
  2) Un proceso entra en el monitor invocando uno de sus procedimientos.
  3) Sólo un proceso puede estar ejecutando dentro del monitor al tiempo; cualquier otro proceso que haya invocado al monitor se bloquea, en espera de que el monitor quede disponible.
- Las dos primeras características guardan semejanza con las de los objetos en el software orientado a objetos. De hecho, en un sistema operativo o lenguaje de programación orientado a objetos puede implementarse inmediatamente un monitor como un objeto con características especiales.
- El monitor facilita la exclusión mutua al tener la disciplina de sólo un proceso a la vez. Si el monitor representa un recurso solo un proceso puede acceder al recurso a la vez.

Sincronización en el monitor

- Si un proceso invoca un monitor y mientras está en el se bloquea, el monitor debería ofrecer la funcionalidad de bloquear el proceso y liberar el monitor para que otro proceso pueda acceder. Cuando el proceso inicial se desbloquee, el proceso debe ser retomado y acceder al monitor en el mismo punto en el que se suspendió.
- Para lograr la sincronización, un monitor utiliza las **variables de condición** que solo pueden ser accedidas desde el monitor.
- Estas variables de condición son un tipo de datos especial en los monitores que se manipulan mediante dos funciones:
  - **cwait(c):** Suspende la ejecución del proceso llamante en la condición c. El monitor queda disponible para ser usado por otro proceso.
  - **csignal(c):** Retoma la ejecución de algún proceso bloqueado por un cwait en la misma condición. Si hay varios procesos, elige uno de ellos; si no hay ninguno, no hace nada.

Operaciones *wait* y *signal*

- Hay que notar que las operaciones wait y signal de los monitores son diferentes de las de los semáforos.
- Si un proceso en un monitor ejecuta la instrucción **csignal(c)** y no hay ningún proceso esperando en la variable condición, la señal se pierde.
- Si un proceso que está ejecutando en el monitor detecta un cambio en la variable condición x, realiza un csignal(x), que avisa del cambio a la correspondiente cola de la condición.



Desventajas del monitor con señal

- Requiere que si hay al menos un proceso en una cola de una condición, un proceso de dicha cola ejecuta inmediatamente cuando otro proceso realice un **csignal** sobre dicha condición.
- Esto implica que el proceso que realizó el **csignal** debe salir inmediatamente del monitor o bloquearse dentro del monitor.
- Esto tiene dos desventajas:
  - Si el proceso no ha terminado con el monitor, entonces se necesitarán dos cambios: uno para bloquear el proceso y otro para retomarlo cuando el monitor esté disponible.
  - La planificación de procesos asociada con una señal debe ser altamente fiable. Lo cuál es muy difícil de lograr, ya que puede un proceso podría fallar justo antes de ejecutar el **csignal** lo cual podría dejar a algunos procesos colgados indefinidamente.

Contenido I

Monitor con notificación y difusión

- **Notificación (cnotify)**
  - Provoca que la cola de la condición **x** sea notificada, pero el proceso que señaló continúa ejecutando.
  - Así, el proceso en cabeza de la cola de la condición será retomado en un momento futuro conveniente, cuando el monitor esté disponible.
  - Esto evita cambios de proceso extra.
- **Difusión (cbroadcast)**
  - Provoca que todos los procesos esperando en una condición pasen a estado Listo.
  - Esto es conveniente en situaciones donde un proceso no sabe cuántos otros procesos deben ser reactivados. Así, todos los procesos en espera de la condición serán avisados para que lo intenten de nuevo.

Ventajas de monitor con notificación y difusión

- Es menos propensa a error ya que cada procedimiento comprueba la variable condición después de ser señalado. Por lo tanto un proceso puede señalar o difundir incorrectamente sin causar un error en el programa señalado. El programa señalado comprobará la variable relevante y si la condición no se cumple, volverá a esperar.
- Se presta a un enfoque más modular de la construcción de programas.

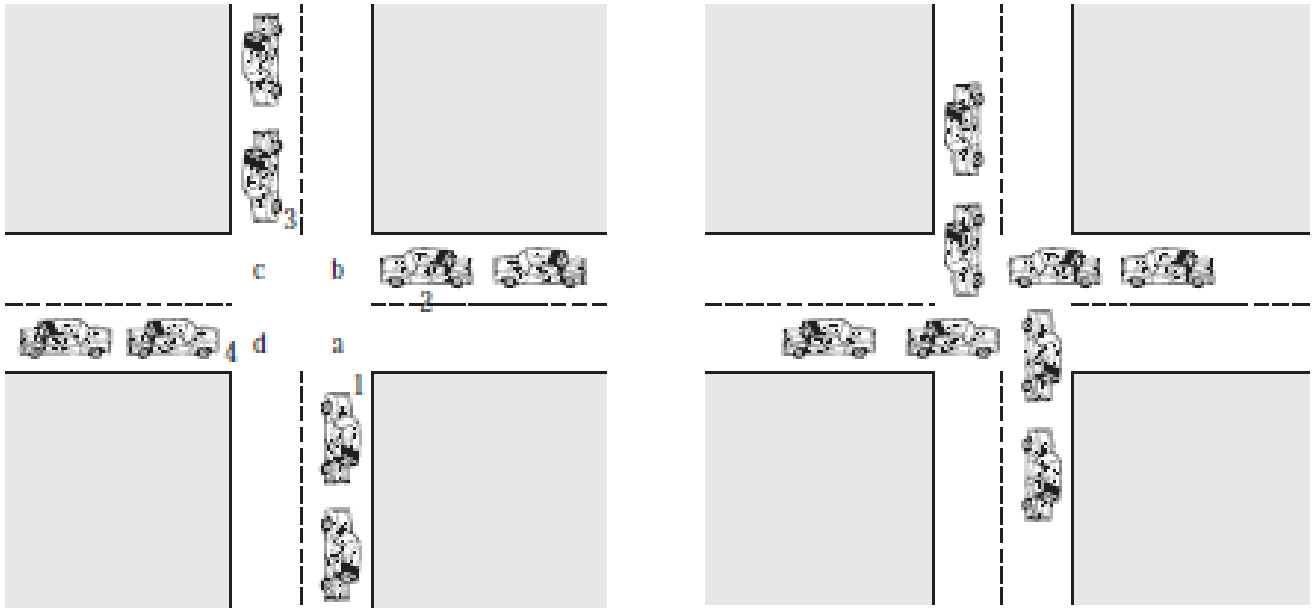
Contenido I

Contenido I

Fundamentos del interbloqueo

- Un conjunto de procesos está interbloqueado cuando cada proceso del conjunto está bloqueado esperando un evento (normalmente la liberación de algún recurso requerido) que sólo puede generar otro proceso bloqueado del conjunto.
- El interbloqueo es permanente porque no puede producirse ninguno de los eventos. No existe una solución eficiente.
- Un interbloqueo involucra necesidades de recursos conflictivas de dos o más procesos.

Ilustración de interbloqueo



(a) Posible interbloqueo (b) Interbloqueo

- Todos los autos llegan al mismo tiempo.
- En un cruce de 4 caminos un auto debe dar paso al autos de la derecha. Esto es efectivo si solo hay 3 autos. En este caso todos los auto dan esperan al de la derecha (interbloqueo).
- Si todos irrespetan la norma anterior y pasan lentamente, cada uno bloqueará a otro auto (interbloqueo).

Interbloqueo en un computador

- Si tenemos dos procesos P y Q ejecutando y requieren el uso exclusivo de recursos durante un tiempo. P(Solicita A, Solicita B, Libera A, Libera B) y P (Solicita B, Solicita A, Libera B, Libera A).
- Las posibles opciones de ejecución son:
  1. Q adquiere B y, a continuación, A, y, más tarde, libera B y A. Cuando P continúe su ejecución, será capaz de adquirir ambos recursos.
  2. Q adquiere B y, a continuación, A. P ejecuta y se bloquea al solicitar A. Q libera B y A. Cuando P continúe su ejecución, será capaz de adquirir ambos recursos.
  3. Q adquiere B y, a continuación, P adquiere A. El **interbloqueo** es inevitable, puesto que cuando la ejecución continúe, Q se bloqueará a la espera de A y P a la de B.
  4. P adquiere A y, a continuación, Q adquiere B. El **interbloqueo** es inevitable, puesto que cuando la ejecución continúe, Q se bloqueará a la espera de A y P a la de B.
  5. P adquiere A y, a continuación, B. Q ejecuta y se bloquea al solicitar B. P libera A y B. Cuando Q continúe su ejecución, será capaz de adquirir ambos recursos.
  6. P adquiere A y, a continuación, B, y, más tarde, libera A y B. Cuando Q continúe su ejecución, será capaz de adquirir ambos recursos.

Interbloqueo en un computador

- El interbloqueo solo se produce cuando los dos procesos necesitan los dos recursos al mismo tiempo de manera exclusiva.
- Por ejemplo, si P no necesitara a A y B al mismo tiempo (Solicita A, Libera A, Solicita B, Libera B), y Q(Solicita B, Solicita A, Libera B, Libera A), entonces no hay forma de que se produzca un interbloqueo.

Recursos reutilizables

- Un recurso reutilizable es aquél que sólo lo puede utilizar de un proceso en cada momento y que no se destruye después de su uso.
- Ejemplos: procesadores, canales de E/S, memoria principal y secundaria, dispositivos, y estructuras de datos como ficheros, bases de datos y semáforos.
- Recursos reutilizables en un interbloqueo: D = fichero de disco y C = una unidad de cinta. Los procesos requieren uso exclusivo de D y C.

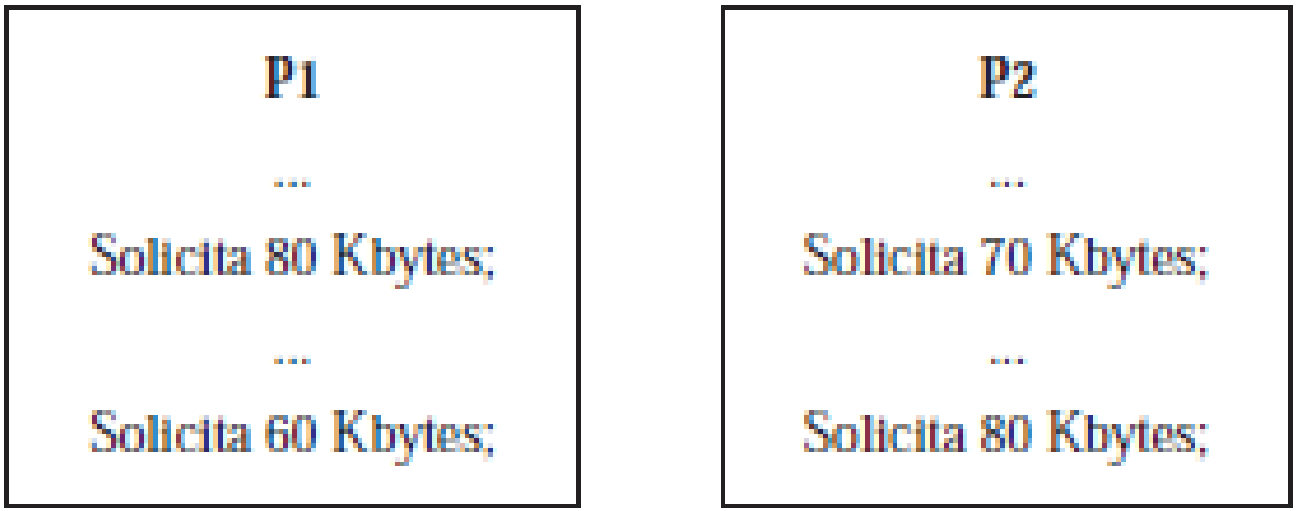
Paso	Acción
p_0	Solicita (D)
p_1	Bloquea (D)
p_2	Solicita (C)
p_3	Bloquea (C)
p_4	Realiza función
p_5	Desbloquea (D)
p_6	Desbloquea (C)

Paso	Acción
q_0	Solicita (C)
q_1	Bloquea (C)
q_2	Solicita (D)
q_3	Bloquea (D)
q_4	Realiza función
q_5	Desbloquea (C)
q_6	Desbloquea (D)

- Como debería intercalarse la ejecución de los procesos para que se produzca un interbloqueo? $p_0\ p_1\ q_0\ q_1\ p_2\ q_2$

Ejemplo de interbloqueo con recurso reutilizable

- Otro ejemplo está relacionado con las peticiones de reserva de memoria principal. Supongamos que disponemos de 200 Kbytes y tenemos los procesos P1 y P2 que realizan las peticiones:



- Si los dos procesos llegan a su segunda petición se producirá un interbloqueo.
- Es difícil tratar con este tipo de interbloqueo si no se conoce la cantidad de memoria que se va a solicitar.
- La mejor manera de solucionarlos será utilizando memoria virtual.

Recursos consumibles

- Los recursos consumibles son los que pueden crearse (producirse) y destruirse (consumirse).
- No hay límite en el número de recursos consumibles de un determinado tipo.
- Un proceso productor puede crear un número ilimitado de estos recursos.
- Cuando un proceso consumidor adquiere un recurso, el recurso deja de existir.
- Ejemplos: las interrupciones, las señales, los mensajes y la información en buffers de E/S.

Ejemplo de interbloqueo con recursos consumibles

- Consideremos dos procesos P1 y P2, de tal manera que cada proceso intenta recibir un mensaje del otro y, a continuación, le envía un mensaje:

```
P1
...
Recibe (P2);
...
Envía (P2, M1);
```

```
P2
...
Recibe (P1);
...
Envía (P1, M2);
```

- Si la función Recibe es bloqueante, entonces se produce el interbloqueo.

Condiciones para el interbloqueo

- Se deben presentar 3 condiciones para que se produzca un interbloqueo:
  1. **Exclusión mutua**. Solo un proceso puede utilizar un recurso.
  2. **Retención y espera.** Un proceso puede mantener los recursos asignados mientras espera la asignación de otros recursos.
  3. **Sin expropiación.** No se puede forzar la expropiación de un recurso a un proceso que lo posee.
- Por varios motivos estas tres condiciones son altamente deseables en un sistema. La exclusión mutua asegura la coherencia de resultados. La no expropiación permite que un proceso pueda ser restaurado a un estado previo y repetir sus acciones.
- Si estas tres condiciones están presentes, se puede producir un interbloqueo pero también puede que no. Para que realmente se produzca un interbloqueo se requiere una cuarta condición:
  4. **Espera circular:** Existe una lista de procesos, de tal manera que cada proceso posee al menos un recurso necesitado por el siguiente proceso de la lista.

Contenido I

Técnicas para el tratamiento del interbloqueo

Existen tres estrategias para el tratamiento del interbloqueo:

- Prevenir
- Predecir
- Detectar

Prevención del interbloqueo

- Consiste en diseñar un sistema de manera que se excluya la posibilidad del interbloqueo.
- Existen dos categorías dentro de los métodos de prevención:
  - **Indirecto** previene la aparición de una de las tres primeras condiciones.
  - **Directo** previene que se produzca la cuarta condición.

Técnicas de prevención de las condiciones

- Veamos como se pueden eliminar cada una de las condiciones:
  - **Exclusión mutua:** Por lo general no es posible eliminarla. Aunque algunos recursos como los ficheros permiten múltiples accesos de lectura, estos necesitan solo un accesos de escritura.
  - **Retención y espera:** Se puede eliminar al obligar que un proceso solicite al mismo tiempo todos sus recursos requeridos. El proceso se bloquea hasta que todos lo recursos puedan ser asignados. Es ineficiente por 2 razones:
    - Se puede bloquear el proceso por mucho tiempo
    - Los recursos pueden estar inutilizados por mucho tiempo.

    También puede ser que el proceso no conozca anticipadamente todos los recursos que necesita.
  - **Sin expropiación:** Existen dos maneras:
    - Si un proceso mantiene varios recursos y se le deniega una nueva petición, este proceso debe liberar sus recursos y luego solicitarlos de nuevo si es necesario.
    - Si un proceso solicita un recurso que otros proceso mantiene, el SO puede obligar al proceso a liberar sus recursos.

Técnicas de prevención de las condiciones

- **Espera circular:** se puede impedir definiendo un orden lineal entre los distintos tipos de recursos. Se puede asociar un índice a cada tipo de recurso, R_1 y R_2. Así, si A y B necesitan de R_1 y R_2 y A ha pedido R_1, B tendrá que esperar a que R_1 esté disponible para luego pedir R_2, no podrá pedir R_2 antes que R_1. También puede ser ineficiente, realentizando los procesos y denegando accesos innecesariamente a un recurso.

Contenido I

Predicción del interbloqueo

- A diferencia que en la prevención, la predicción permite las tres condiciones necesarias pero toma decisiones razonables para asegurarse de que nunca se alcanza el punto del interbloqueo.
- Permite más **concurrencia** que en la prevención.
- Decide dinámicamente si la petición actual de reserva de un recurso podrá potencialmente causar un interbloqueo. Por lo tanto, requiere el conocimiento de las futuras solicitudes de recursos del proceso.

Técnicas de predicción

- Existen dos técnicas para para predecir el interbloqueo:
  - **Denegar el inicio de un proceso** Solo se inicia un proceso si se pueden satisfacer las necesidades máximas de todos los procesos actuales más las del nuevo proceso. No es óptima ya que asume el peor caso en donde todos lo procesos solicitarán sus necesidades máximas simultáneamente.
  - **Denegar la asignación de más recursos:** También se la conoce como *El algoritmo del banquero*. Asegura que el sistema de procesos y recursos está siempre en un **estado seguro**.
    - Un proceso solicita un conjunto de recursos,
    - *Supone* que se concede la petición
    - Actualiza el estado del sistema
    - Determina si el resultado es un estado seguro.
    - En caso afirmativo, se concede la petición. Caso contrario, se bloquea el proceso hasta que sea seguro conceder la petición.sistema.

    Nota: Un estado seguro es aquél en el que todos los procesos pueden ejecutarse al completo.

Contenido I

Detección del interbloqueo

- A diferencia de la prevención (muy conservadora), la detección no limita el acceso a los recursos ni restringe las acciones de los procesos.
- Los recursos pedidos se conceden a los procesos siempre que sea posible.
- Periódicamente, el SO realiza un algoritmo que le permite detectar la condición de espera circular.
- Esta comprobación de la existencia de interbloqueo se puede hacer con tanta frecuencia como cada vez que se realiza un petición de recurso, o con menos frecuencia dependiendo de la probabilidad de que ocurra uno.
- La comprobación en cada petición tiene la ventaja de detectar tempranamente los interbloqueos y que el algoritmo es sencillo. Sin embargo, estas comprobaciones consumen tiempo del procesador.

Algoritmo de detección del interbloqueo

- Un algoritmo utilizado comúnmente utiliza una matriz $Asignacion$, un vector $Disponibles$ y una matriz $Solicitud$ que representa la cantidad de recursos solicitados por los procesos.
- El algoritmo marca los procesos que no están en un interbloqueo.
- Si al final existen procesos sin marcar, esos procesos están en un interbloqueo.
- El objetivo del algoritmo es encontrar un proceso cuyas peticiones puedan satisfacerse con los recursos disponibles, asumir que se conceden estos recursos y que el proceso se ejecuta hasta terminar y liberar los recursos. Luego, el algoritmo busca otro proceso para ejecutar.
- Este algoritmo **no previene** el interbloqueo, la ocurrencia de un interbloqueo dependerá del orden en que se concedan las peticiones.
- Este algoritmo simplemente determina la existencia de un interbloqueo.

Recuperación de un interbloqueo

Detección del interbloqueo

- Cuando el algoritmo detecta un interbloqueo, debe aplicar una de las siguientes estrategias para recuperarlo:
  1. Abortar todos los procesos involucrados en el interbloqueo. Esta el la solución más utilizada en los SO.
  2. Retroceder cada proceso en interbloqueo a un punto de control establecido y rearrancar todos los procesos. Requiere una implementación de retroceso y rearranque. El problema es que el interbloqueo puede repetirse aunque gracias al indeterminismo del procesamiento concurrente puede que esto no suceda.
  3. Abortar sucesivamente los procesos en interbloqueo hasta que éste deje de existir. El orden de abortarlos debe basarse en criterios que impliquen el coste mínimo. Después de cada aborto debe invocarse al algoritmo para comprobar que ya no existe el interbloqueo.
  4. Expropiar sucesivamente los recursos hasta que eje de existir el interbloqueo. De igual manera se debe minimizar el coste y se debe verificar que el interbloqueo ha dejado de existir. El proceso expropiado debe retroceder a un punto previo a la adquisición del recurso.

Contenido I

Una estrategia integrada de tratamiento

- Cada una de las estrategias presentadas tienen ventajas y desventajas. Por lo tanto podríamos utilizar diferentes estrategias en diferentes situaciones.
- Por ejemplo:
  - Agrupar los recursos en diversas clases de recursos diferentes.
  - Utilizar la estrategia del orden lineal para prevenir la espera circular.
  - Usar el algoritmo más apropiado dentro de una clase de recursos.
- Se podrían tener las siguientes clases de recursos (en ese orden):
  1. **Espacio de intercambio** almacenamiento secundario (prevención mediante asignación de todos los recursos al mismo tiempo o predicción)
  2. **Recursos del proceso** dispositivos asignables como ficheros (predicción, prevención por ordenamiento)
  3. **Memoria principal** asignable a procesos en páginas o segmentos (prevención por expropiación).
  4. **Recursos internos** canales E/S (prevención por ordenamiento)