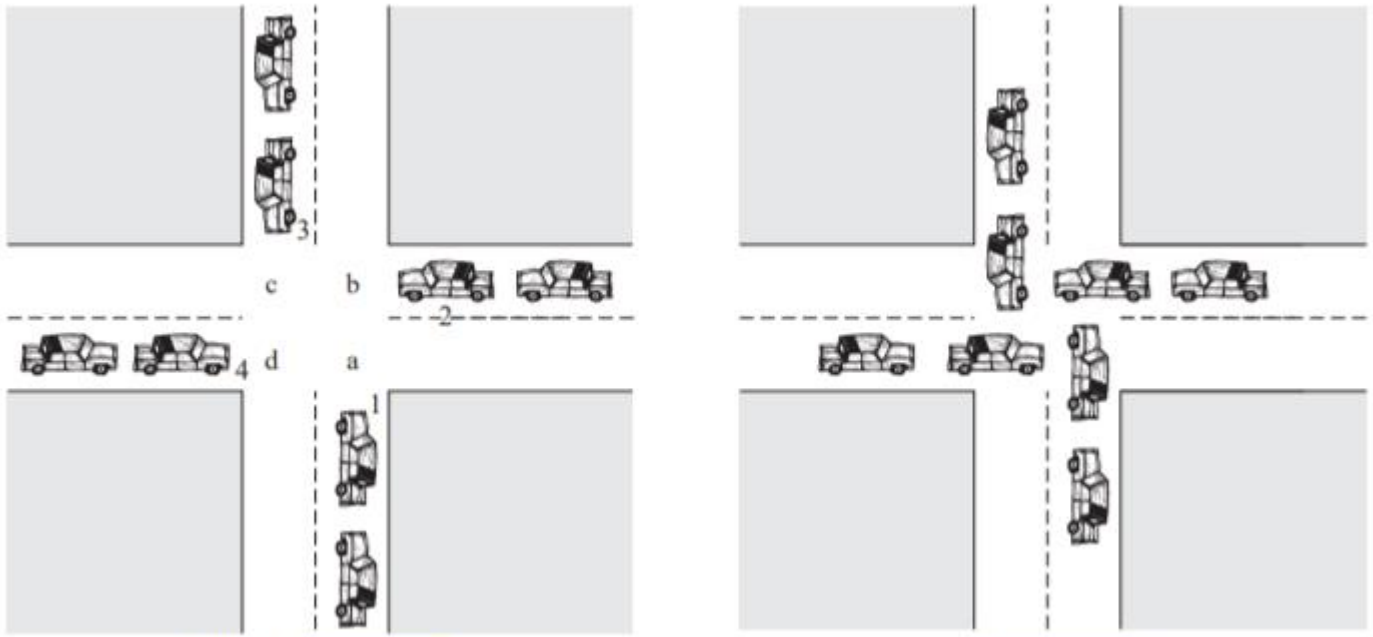
Lección 6

INTERBLOQUEOS



(a) Posible interbloqueo

(b) Interbloqueo

Introducción

- Los sistemas computacionales están llenos de recursos que pueden ser utilizados por sólo un proceso a la vez, por ejemplo las impresoras, las unidades de cinta y las ranuras en los tableros internos del sistema. Cuando dos procesos escriben de manera simultánea en la impresora se producen incoherencias. Si dos procesos utilizan la misma entrada en la tabla del sistema de archivos invariablemente se corrompe el sistema de archivos. En consecuencia, todos los sistemas operativos tienen la habilidad de otorgar (en forma temporal) a un proceso el acceso exclusivo a ciertos recursos.
- Para muchas aplicaciones, un proceso necesita acceso exclusivo no sólo a un recurso, sino a varios. Por ejemplo, suponga que cada uno de dos procesos quiere grabar un documento 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, muchas oficinas tienen una red de área local con muchas computadoras conectadas. A menudo, los dispositivos como escáneres, grabadores de CD, impresoras y unidades de cinta se conectan a la red como recursos compartidos, disponibles para cualquier usuario en cualquier equipo. Si estos dispositivos se pueden reservar de manera remota (es decir, desde el equipo doméstico del usuario), pueden ocurrir los mismos tipos de interbloqueos antes descritos. Las situaciones más complicadas pueden ocasionar interbloqueos que involucren a tres, cuatro o más dispositivos y usuarios.

Introducción

- Por ejemplo, en un sistema de bases de datos, un programa puede tener que bloquear varios registros que esté utilizando para evitar condiciones de competencia. Si el proceso *A* bloquea el registro *R1* y el proceso *B* bloquea el registro *R2*, y después cada proceso trata de bloquear el registro del otro, también tenemos un interbloqueo. Por ende, los interbloqueos pueden ocurrir en los recursos de hardware o de software.

6.1 RECURSOS

- Una clase principal de interbloqueos involucra a los recursos, por lo que para empezar nuestro estudio veremos lo que son. Los interbloqueos pueden ocurrir cuando a los procesos se les otorga acceso exclusivo a los dispositivos, registros de datos, archivos, etcétera.
- Un recurso puede ser un dispositivo de hardware (por ejemplo, una unidad de cinta) o una pieza de información (como un registro bloqueado en una base de datos). En resumen, un recurso es cualquier cosa que se debe adquirir, utilizar y liberar con el transcurso del tiempo.
- **6.1.1 Recursos apropiativos y no apropiativos**
- Los recursos son de dos tipos: apropiativos y no apropiativos.
- Un **recurso apropiativo** es uno que se puede quitar al proceso que lo posee sin efectos dañinos. La memoria es un ejemplo de un recurso apropiativo.
- Por el contrario, un **recurso no apropiativo** es uno que no se puede quitar a su propietario actual sin hacer que el cómputo falle. 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.
- En general, los interbloqueos involucran a los recursos no apropiativos.

- La secuencia de eventos requerida para utilizar un recurso se proporciona a continuación, en un formato abstracto.
    1. Solicitar el recurso.
    2. Utilizar el recurso.
    3. Liberar el recurso.
- Si el recurso no está disponible cuando se le solicita, el proceso solicitante se ve obligado a esperar. En algunos sistemas operativos, el proceso se bloquea de manera automática cuando falla la solicitud de un recurso, y se despierta cuando el recurso está disponible. En otros sistemas, la solicitud falla con un código de error y depende del proceso que hizo la llamada decidir si va a esperar un poco e intentar de nuevo.
- Un proceso al que se le ha negado la petición de un recurso por lo general permanece en un ciclo estrecho solicitando el recurso, después pasa al estado inactivo y después intenta de nuevo. Aunque este proceso no está bloqueado, para toda intención y propósito es como si lo estuviera, debido a que no puede realizar ningún trabajo útil. En nuestro siguiente análisis vamos a suponer que cuando a un proceso se le niega un recurso solicitado pasa al estado inactivo.
- La naturaleza exacta de solicitar un recurso es en gran medida dependiente del sistema. En algunos sistemas se proporciona una llamada al sistema request para permitir que los procesos pidan los recursos en forma explícita. En otros, los únicos recursos que conoce el sistema operativo son los archivos especiales que sólo un proceso puede tener abiertos en un momento dado. Éstos se abren mediante la llamada al sistema open ordinaria. Si el archivo ya está en uso, el proceso que llama es bloqueado hasta que su propietario actual lo cierra.

6.1.2 Adquisición de recursos

- Para ciertos tipos de recursos, como los registros de una base de datos, es responsabilidad de los procesos de usuario administrar su uso. Una manera de permitir que los usuarios administren los recursos es asociar un semáforo con cada recurso. Estos semáforos se inicializan con 1. Se pueden utilizar mutexes de igual forma. Los tres pasos antes listados se implementan como una operación down en el semáforo para adquirir el recurso, usarlo y finalmente realizar una operación up en el recurso para liberarlo.
- **6.2 INTRODUCCIÓN A LOS INTERBLOQUEOS**
- 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. La condición sin interrupciones es necesaria para evitar que un proceso, que de cualquier otra forma estaría en interbloqueo, sea despertado por una alarma, por ejemplo, y después ocasione eventos que liberen a otros procesos en el conjunto.
- El **interbloqueo de recursos**. Es probablemente el tipo más común, pero no el único. Primero estudiaremos los interbloqueos de recursos con detalle, y después regresaremos brevemente a los demás tipos de interbloqueos al final del capítulo.

- **6.2.1 Condiciones para los interbloqueos de recursos**
- Deben aplicarse cuatro condiciones para un interbloqueo (de recursos):

1. Condición de exclusión mutua. Cada recurso se asigna en un momento dado a sólo un proceso, o está disponible.
2. Condición de contención y espera. Los procesos que actualmente contienen recursos que se les otorgaron antes pueden solicitar nuevos recursos.
3. Condición no apropiativa. Los recursos otorgados previamente no se pueden quitar a un proceso por la fuerza. Deben ser liberados de manera explícita por el proceso que los contiene.
4. Condición de espera circular. Debe haber una cadena circular de dos o más procesos, cada uno de los cuales espera un recurso contenido por el siguiente miembro de la cadena.

- En general, se utilizan cuatro estrategias para lidiar con los interbloqueos.

1. Sólo ignorar el problema. Tal vez si usted lo ignora, él lo ignorará a usted.
2. Detección y recuperación. Dejar que ocurran los interbloqueos, detectarlos y tomar acción.
3. Evitarlos en forma dinámica mediante la asignación cuidadosa de los recursos.
4. Prevención, al evitar estructuralmente una de las cuatro condiciones requeridas.

- **6.3 EL ALGORITMO DE LA AVESTRUZ**
- No hacer nada.
- **6.4 DETECCIÓN Y RECUPERACIÓN DE UN INTERBLOQUEO**
- 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.
- **6.4.1 Detección de interbloqueos con un recurso de cada tipo**
- Vamos a empezar con el caso más simple: sólo existe un recurso de cada tipo. Dicho sistema podría tener un escáner, un grabador de CD, un trazador (plotter) y una unidad de cinta, pero no más que un recurso de cada clase. Se puede usar un algoritmo basado en nodos.
- **6.4.2 Detección del interbloqueo con varios recursos de cada tipo**
- Cuando existen varias copias de algunos de los recursos, se necesita un método distinto para detectar interbloqueos. Se puede usar un algoritmo basado en matrices para detectar interbloqueos entre *n* procesos.
- **6.4.3 Recuperación de un interbloqueo**
- En caso de detección de interbloqueos se necesita alguna forma de recuperarse y hacer funcionar el sistema otra vez. Podemos hacer 3 cosas: Apropiarse del recurso de nuevo, hacer un retroceso o eliminar el proceso.

6.5 CÓMO EVITAR INTERBLOQUEOS

- Para evitar interbloqueos necesitamos cierta información de antemano.

- **6.5.1 Trayectorias de los recursos**
  - Los principales algoritmos para evitar interbloqueos se basan en el concepto de los estados seguros.

- **6.5.2 Estados seguros e inseguros**
- Se dice que un estado es **seguro** si hay cierto orden de programación en el que se puede ejecutar cada proceso hasta completarse, incluso aunque todos ellos solicitaran de manera repentina su número máximo de recursos de inmediato.

- **6.5.3 El algoritmo del banquero para un solo recurso**

- **6.5.4 El algoritmo del banquero para varios recursos**

6.6 CÓMO PREVENIR INTERBLOQUEOS

- Evitar los interbloqueos es algo en esencia imposible, debido a que se requiere información sobre las peticiones futuras, que no se conocen, ¿cómo evitan los sistemas reales el interbloqueo?
- La respuesta es volver a las cuatro condiciones establecidas por Coffman y colaboradores (1971) para ver si pueden proporcionarnos una pista. Si podemos asegurar que por lo menos una de estas condiciones nunca se cumpla, entonces los interbloqueos serán estructuralmente imposibles (Havender, 1968).
- **6.6.1 Cómo atacar la condición de exclusión mutua**
- Si ningún recurso se asignara de manera exclusiva a un solo proceso, nunca tendríamos interbloqueos. No obstante, es igual de claro que al permitir que dos procesos escriban en la impresora al mismo tiempo se producirá un caos. Al colocar la salida de la impresora en una cola de impresión, varios procesos pueden generar salida al mismo tiempo. En este modelo, el único proceso que realmente solicita la impresora física es el demonio de impresión. Como el demonio nunca solicita ningún otro recurso, podemos eliminar el interbloqueo para la impresora.
- Los demonios comúnmente se programan para imprimirse sólo hasta después que esté disponible el archivo de salida completo. Sin embargo, esta decisión en sí podría provocar un interbloqueo. ¿Qué ocurriría si dos procesos llenaran cada uno una mitad del espacio de la cola de impresión disponible con datos de salida y ninguno terminara de producir su salida completa?
- Sin embargo, hay un germen de una idea aquí, que se aplica con frecuencia: **evite asignar un recurso cuando no sea absolutamente necesario, y trate de asegurarse que la menor cantidad posible de procesos reclamen ese recurso.**

- **6.6.2 Cómo atacar la condición de contención y espera**
- Si podemos evitar que los procesos que contienen recursos esperen por más recursos, podemos eliminar los interbloqueos. Una forma de lograr esta meta es requerir que todos los procesos soliciten todos sus recursos antes de empezar su ejecución. Si todo está disponible, al proceso se le asignará lo que necesite y podrá ejecutarse hasta completarse. Si uno o más recursos están ocupados, no se asignará nada y el proceso sólo esperará.
- Un problema inmediato con este método es que muchos procesos no saben cuántos recursos necesitarán hasta que hayan empezado a ejecutarse. De hecho, si lo supieran se podría utilizar el algoritmo del banquero. Otro problema es que los recursos no se utilizarán de una manera óptima con este método. Tome como ejemplo un proceso que lee datos de una cinta de entrada, los analiza por una hora y después escribe en una cinta de salida y traza los resultados. Si todos los recursos se deben solicitar de antemano, el proceso ocupará la unidad de cinta de salida y el trazador por una hora.
- Sin embargo, algunos sistemas de procesamiento por lotes de mainframes requieren que el usuario liste todos los recursos en la primera línea de cada trabajo. Después el sistema adquiere todos los recursos de inmediato, y los mantiene hasta que el trabajo termina. Aunque este método impone una carga sobre el programador y desperdicia recursos, evita los interbloqueos.
- Una manera ligeramente distinta de romper la condición de contención y espera es requerir que un proceso que solicita un recurso libere temporalmente todos los recursos que contiene en un momento dado. Después puede tratar de obtener todo lo que necesite a la vez.

- **6.6.3 Cómo atacar la condición no apropiativa**
- También es posible atacar la tercera condición (no apropiativa). Si a un proceso se le ha asignado la impresora y está a la mitad de imprimir su salida, quitarle la impresora a la fuerza debido a que el trazador que necesita no está disponible es algo engañoso como máximo, e imposible en el peor caso. Sin embargo, ciertos recursos se pueden virtualizar para evitar esta situación. Al colocar en una cola de impresión en el disco la salida de la impresora y permitir que sólo el demonio de impresión tenga acceso a la impresora real, se eliminan los interbloqueos que involucran a la impresora, aunque se crea uno para el espacio en disco. No obstante, con los discos grandes es muy improbable quedarse sin espacio.
- Sin embargo, no todos los recursos se pueden virtualizar de esta manera. Por ejemplo, los registros en las bases de datos o las tablas dentro del sistema operativo se deben bloquear para poder utilizarse, y ahí es donde se encuentra el potencial para el interbloqueo.

- **6.6.4 Cómo atacar la condición de espera circular**
- La espera circular se puede eliminar de varias formas. Una de ellas es simplemente tener una regla que diga que un proceso tiene derecho sólo a un recurso en cualquier momento. Si necesita un segundo recurso, debe liberar el primero. Para un proceso que necesita copiar un enorme archivo de una cinta a una impresora, esta restricción es inaceptable.
- Otra manera de evitar la espera circular es proporcionar una numeración global de todos los recursos, como se muestra en la figura 6-13(a). Ahora la regla es ésta: los procesos pueden solicitar recursos cada vez que quieran, pero todas las peticiones se deben realizar en orden numérico. Un proceso puede pedir primero una impresora y después una unidad de cinta, pero tal vez no pueda pedir primero un trazador y después una impresora.

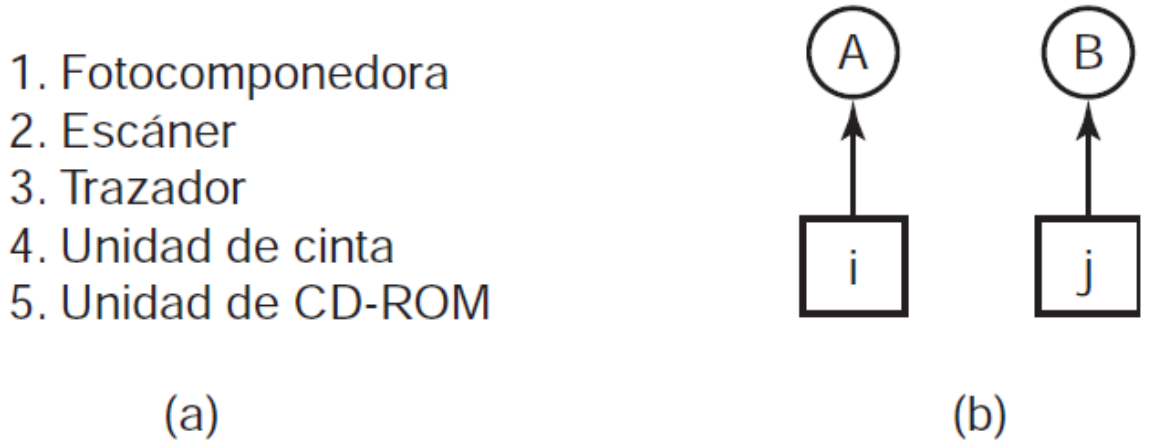


Figura 6-13. (a) Recursos ordenados en forma numérica. (b) Un gráfico de recursos.

- Con esta regla, el gráfico de asignación de recursos nunca puede tener ciclos. Veamos por qué es esto cierto para el caso de dos procesos, en la figura 6-13(b). Podemos obtener un interbloqueo sólo si *A* solicita el recurso *j* y *B* solicita el recurso *i*. Suponiendo que *i* y *j* sean recursos distintos, tendrán diferentes números. Si $i > j$, entonces *A* no puede solicitar a *j* debido a que es menor de lo que ya tiene. Si $i < j$, entonces *B* no puede solicitar a *i* debido a que es menor de lo que ya tiene. De cualquier forma, el interbloqueo es imposible.
- Con más de dos procesos se aplica la misma lógica. En cada instante, uno de los recursos asignados será el más alto. El proceso que contiene ese recurso nunca pedirá un recurso que ya esté asignado. Terminará o, en el peor caso, solicitará recursos con mayor numeración, los cuales están disponibles. En un momento dado, terminará y liberará sus recursos. En este punto, algún otro proceso contendrá el recurso más alto y también podrá terminar. En resumen, existe un escenario en el que todos los procesos terminan, por lo que no hay interbloqueo presente.
- Una variación menor de este algoritmo es retirar el requerimiento de que los recursos se adquieran n una secuencia cada vez más estricta, y simplemente insistir que ningún proceso puede solicitar un recurso menor del que ya contiene. Si al principio un proceso solicita 9 y 10, y después libera ambos recursos, en efecto está empezando otra vez, por lo que no hay razón de prohibirle que solicite el recurso 1.
- Aunque el ordenamiento numérico de los recursos elimina el problema de los interbloqueos, puede ser imposible encontrar un ordenamiento que satisfaga a todos. Cuando los recursos incluyen entradas en la tabla de procesos, espacio en la cola de impresión del disco, registros bloqueados de la base de datos y otros recursos abstractos, el número de recursos potenciales y usos distintos puede ser tan grande que ningún ordenamiento podría funcionar.

6.7 OTRAS CUESTIONES

- En esta sección analizaremos varias cuestiones relacionadas con los interbloqueos. Éstas incluyen el bloqueo de dos fases, los interbloqueos sin recursos y la inanición.
- **6.7.1 Bloqueo de dos fases**
- En muchos sistemas de bases de datos, una operación que ocurre con frecuencia es solicitar bloqueos sobre varios registros y después actualizar todos los registros bloqueados.
- Cuando hay varios procesos en ejecución al mismo tiempo, hay un peligro real de interbloqueo. A menudo, a este método se le conoce como **bloqueo de dos fases**. En la primera fase, el proceso trata de bloquear todos los registros que necesita, uno a la vez. Si tiene éxito pasa a la segunda fase, realizando sus actualizaciones y liberando los bloqueos. No se realiza ningún trabajo real en la primera fase.
- Si durante la primera fase se necesita cierto registro que ya esté bloqueado, el proceso sólo libera todos sus bloqueos e inicia la primera fase desde el principio.
- Sin embargo, esta estrategia no es aplicable en general. Por ejemplo, en los sistemas de tiempo real y los sistemas de control de procesos, no es aceptable sólo terminar un proceso a la mitad debido a que un recurso no está disponible, y empezar todo de nuevo. Tampoco es aceptable iniciar de nuevo si el proceso ha leído o escrito mensajes en la red, actualizado archivos o cualquier otra cosa que no se pueda repetir con seguridad.
- El algoritmo funciona sólo en aquellas situaciones en donde el programador ha ordenado las cosas con mucho cuidado, para que el programa se pueda detener en cualquier punto durante la primera fase y luego se reinicie.

- **6.7.2 Interbloqueos de comunicaciones**
- Aunque los interbloqueos de recursos son el tipo más común, no son el único. Otro tipo de interbloqueo puede ocurrir en los sistemas de comunicaciones (como las redes), en donde dos o más procesos se comunican mediante el envío de mensajes. Un arreglo común es que el proceso *A* envía un mensaje de petición al proceso *B*, y después se bloquea hasta que *B* envía de vuelta un mensaje de respuesta. Suponga que el mensaje de petición se pierde. *A* se bloquea en espera de la respuesta. *B* se bloquea en espera de una petición para que haga algo. Tenemos un interbloqueo. Sólo que éste no es el clásico interbloqueo de recursos. *A* no está en posesión de un recurso que *B* quiere, ni viceversa. De hecho, no hay recursos a la vista. Pero es un interbloqueo de acuerdo con nuestra definición formal, ya que tenemos un conjunto de (dos) procesos, cada uno bloqueado en espera de un evento que sólo el otro puede provocar. A esta situación se le conoce como **interbloqueo de comunicación**.
- La técnica que por lo general se puede utilizar para romper los interbloqueos de comunicación: los tiempos de espera.
- En la mayoría de los sistemas de comunicación de red, cada vez que se envía un mensaje del que se espera una respuesta, también se inicia un temporizador. Si el temporizador termina su conteo antes de que llegue la respuesta, el emisor del mensaje asume que éste se ha perdido y lo envía de nuevo (una y otra vez, si es necesario). De esta forma se evita el interbloqueo.
- Desde luego que, si el mensaje original no se perdió pero la respuesta simplemente se retrasó, el receptor destinado puede recibir el mensaje dos o más veces, tal vez con consecuencias indeseables.
- Piense en un sistema bancario electrónico en el que el mensaje contiene instrucciones para realizar un pago. Es evidente que no se debe repetir (y ejecutar) varias veces, sólo porque la red es lenta o el tiempo de espera es demasiado corto. El diseño de las reglas de comunicación para que todo funcione bien, que se conocen como **protocolo.**

- **6.7.3 Bloqueo activo**
- En ciertas situaciones se utiliza el sondeo (ocupado en espera). Esta estrategia se utiliza a menudo cuando se va a usar la exclusión mutua por un tiempo muy corto, y la sobrecarga de la suspensión es grande en comparación con realizar el trabajo. Considere una primitiva atómica en la que el proceso que llama prueba un mutex y lo sostiene o devuelve una falla.
- Ahora imagine un par de procesos que utilizan dos recursos, como se muestra en la figura 6-16. Cada uno necesita dos recursos y ambos utilizan la primitiva de sondeo *entrar_region* para tratar de adquirir los bloqueos necesarios. Si falla el intento, el proceso sólo intenta otra vez. En la figura 6-16, si el proceso *A* se ejecuta primero y adquiere el recurso 1, y después se ejecuta el proceso 2 y adquiere el recurso 2, sin importar quién se ejecute a continuación, no progresará más, pero ninguno de los procesos se bloqueará. Sólo utiliza su quantum de CPU una y otra vez sin progresar, pero sin bloquearse tampoco. Por ende, no tenemos un interbloqueo (debido a que ningún proceso está bloqueado), pero tenemos algo funcionalmente equivalente al interbloqueo: el **bloqueo activo** (*livelock*).

```
void proceso_A(void) {
entrar_region(&recurso_1);
entrar_region(&recurso_2);
usar_ambos_recursos();
salir_region(&recurso_2);
salir_region(&recurso_1);
}
```

```
void proceso_B(void) {
entrar_region(&recurso_2);
entrar_region(&recurso_1);
usar_ambos_recursos();
salir_region(&recurso_1);
salir_region(&recurso_2);
}
```

- **Figura 6-16.** El estado de ocupado en espera puede conducir al bloqueo activo.

- **6.7.4 Inanición**
- Un problema muy relacionado con el interbloqueo y el bloqueo activo es la **inanición**. En un sistema dinámico, las peticiones de recursos ocurren todo el tiempo. Se necesita cierta política para decidir acerca de quién obtiene qué recurso y cuándo. Esta política, aunque parece razonable, puede ocasionar que ciertos procesos nunca reciban atención, aun cuando no estén en interbloqueo.
- Como ejemplo, considere la asignación de la impresora. Imagine que el sistema utiliza cierto algoritmo para asegurar que la asignación de la impresora no produzca un interbloqueo. Ahora suponga que varios procesos la quieren al mismo tiempo. ¿Quién debe obtenerla?
- Un posible algoritmo de asignación es otorgar la impresora al proceso con el archivo más pequeño a imprimir (suponiendo que esta información esté disponible). Este método maximiza el número de clientes satisfechos y parece razonable. Ahora considere lo que ocurre en un sistema ocupado, cuando un proceso tiene que imprimir un archivo enorme. Cada vez que la impresora esté libre, el sistema buscará y elegirá el proceso con el archivo más pequeño. Si hay un flujo constante de procesos con archivos cortos, el proceso con el archivo enorme nunca recibirá la impresora. Simplemente se pospondrá de manera indefinida, aun cuando no está bloqueado.
- La inanición se puede evitar mediante el uso de una política de asignación de recursos del tipo “primero en llegar, primero en ser atendido”. Con este método, el proceso que espere más tiempo será el que se atienda primero. A su debido tiempo, cualquier proceso dado se convertirá en el más antiguo y, por ende, obtendrá el recurso que necesita.