SISTEMAS OPERATIVOS:

Diseño e implementación

Segunda edición

Andrew S. Tanenbaum
Vrije Universiteit
Amsterdam, Países Bajos

Albert S. Woodhull
Hampshire College
Amherst, Massachusetts

TRADUCCIÓN:

Roberto Escalona
Traductor profesional

REVISIÓN TÉCNICA:

Raymundo Hugo Rangel Gutiérrez
Profesor de Ingeniería en Computación - UNAM

PRENTICE
HALL

MÉXICO • NUEVA YORK • BOGOTÁ • LONDRES • MADRID
MUNICH • NUEVA DELHI • PARÍS • RÍO DE JANEIRO • SINGAPUR
SYDNEY • TOKIO • TORONTO • ZURICH

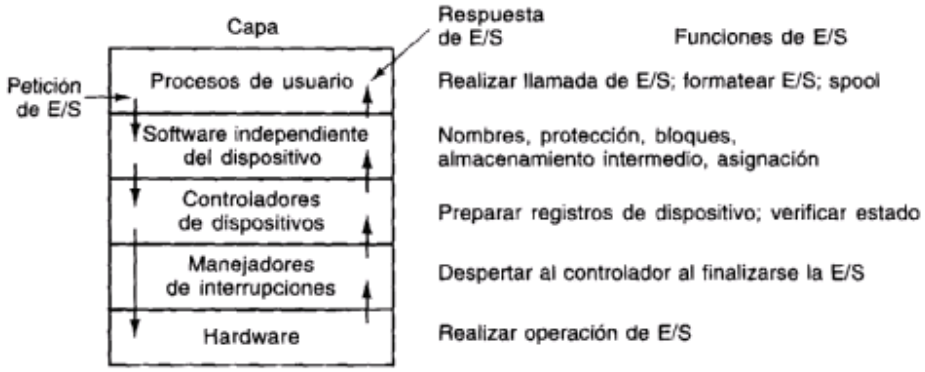

Figura 3-6. Capas del sistema de E/S y funciones principales de cada capa.

3.3 BLOQUEO MUTUO

Los sistemas de cómputo tienen muchos recursos que sólo pueden ser utilizados por un proceso a la vez. Ejemplos comunes de ellos son los graficadores de cama plana, los lectores de CD ROM, las grabadoras de CD-ROM, los sistemas de respaldo en cinta DAT de 8 mm, los formadores de imágenes y las ranuras de la tabla de procesos del sistema. Tener dos procesos escribiendo simultáneamente en una impresora produce basura. Tener dos procesos usando la misma ranura de la tabla de procesos probablemente causaría una caída del sistema. Por ello, todos los sistemas operativos tienen la capacidad de conceder (temporalmente) a un proceso acceso exclusivo a ciertos recursos.

En muchas aplicaciones, un proceso necesita acceso exclusivo no a un recurso, sino a varios. Consideremos, por ejemplo, una compañía de marketing que se especializa en preparar mapas demográficos detallados en un graficador de cama plana de 1 m de ancho. La información demográfica proviene de los CD-ROM que contienen datos censuales y de otro tipo. Supongamos que el proceso A solicita la unidad de CD-ROM y la obtiene. Un momento después, el proceso B solicita el graficador y también lo obtiene. Ahora el proceso A solicita el graficador y se bloquea esperándolo. Por último, el proceso B solicita la unidad de CD-ROM y también se bloquea. En este punto ambos procesos quedan bloqueados y permanecen así eternamente. Esta situación se denomina bloqueo mutuo. No conviene tener b1oqueo un sistema.

Los bloqueos mutuos pueden ocurrir en muchas situaciones además de la petición de dispositivos de E/S de uso exclusivo. Por ejemplo, en un sistema de base de datos, un programa podría tener que poner un candado a varios registros que está usando, a fin de evitar condiciones de competencia. Si el proceso A asegura el registro Rl y el proceso B asegura el registro R2, y luego cada proceso trata de asegurar el registro del otro, también tendremos un bloqueo mutuo. Por tanto, los bloqueos mutuos pueden ocurrir con recursos de hardware o de software.

En esta sección examinaremos los bloqueos mutuos con mayor detenimiento para ver cómo surgen y cómo pueden prevenirse o evitarse. Como ejemplos, hablaremos de adquirir dispositivos

físicos como unidades de cinta, unidades de CD-ROM y graficadores, porque son fáciles de visualizar, pero los principios y algoritmos se aplican igualmente bien a otros tipos de bloqueos mutuos.

3.3.1 Recursos

Los bloqueos mutuos pueden ocurrir cuando se otorga a los procesos acceso exclusivo a dispositivos, archivos, etc. A fin de hacer la explicación de los bloqueos mutuos lo más general posible, nos referiremos a los objetos otorgados como recursos. Un recurso puede ser un dispositivo de hardware (p. ej., una unidad de cinta) o un elemento de información (p. ej., un registro con candado en una base de datos). Una computadora normalmente tiene muchos recursos distintos que se pueden adquirir. Para algunos recursos pueden estar disponibles varios ejemplares idénticos, como tres unidades de cinta. Cuando están disponibles varias copias de un recurso, cualquiera de ellas puede usarse para satisfacer cualquier petición del usuario por ese recurso. En pocas palabras, un recurso es cualquier cosa que sólo puede ser usada por un proceso en un instante dado.

Los recursos son de dos tipos: expropiables y no expropiables. Un recurso expropiable es uno que se puede arrebatar al proceso que lo tiene sin que haya efectos adversos. La memoria es un ejemplo de recurso expropiable. Consideremos, por ejemplo, un sistema con 512K de memoria de usuario, una impresora y dos procesos de 512K que quieren imprimir algo. El proceso A solicita y obtiene la impresora, y comienza a calcular los valores que va a imprimir, pero antes de que haya terminado el cálculo excede su cuanto de tiempo y es intercambiado a disco.

Ahora se ejecuta el proceso B e intenta, sin éxito, adquirir la impresora. Aquí tenemos una situación de bloqueo mutuo en potencia, porque A tiene la impresora y B tiene la memoria, y ninguno puede proceder sin el recurso que el otro tiene. Por fortuna, es posible quitarle la memo ria a B (expropiarla) intercambiando B a disco e intercambiando A a la memoria. Ahora A puede ejecutarse, imprimir, y por último liberar la impresora. No hay bloqueo mutuo.

En contraste, un recurso no expropiable no puede quitársele a su poseedor actual sin hacer que el cómputo falle. Si un proceso ya comenzó a imprimir salidas y se le quita la impresora para dársela a otro proceso, se obtendrá basura como salida. Las impresoras no son expropiables.

En general, en los bloqueos mutuos intervienen recursos no expropiables. Los bloqueos mutuos en potencia en los que intervienen recursos expropiables casi siempre pueden resolverse reasignando los recursos de un proceso a otro. Por tanto, nuestro tratamiento se centrará en los recursos no apropiables.

La secuencia de sucesos que se requiere para usar un recurso es:

1. Solicitar el recurso.
2. Usar el recurso.
3. Liberar el recurso.

Si el recurso no está disponible cuando se solicita, el proceso que realiza la solicitud tiene que esperar. En algunos sistemas operativos, el proceso se bloquea automáticamente cuando una petición

de recurso falla, y se le despierta cuando el recurso está disponible. En otros sistemas, la petición falla con un código de error, y toca al proceso invocador esperar un poco e intentarlo de nuevo.

3.3.2 Principios del bloqueo mutuo

El bloqueo mutuo puede definirse formalmente como sigue:

Un conjunto de procesos está en bloqueo mutuo si cada proceso del conjunto está esperando un evento que sólo otro proceso del conjunto puede causar.

Puesto que todos los procesos están esperando, ninguno de ellos puede causar ninguno de los eventos que podrían despertar a cualquiera de los demás miembros del conjunto, y todos los proce sos continúan esperando indefinidamente.

En la mayor parte de los casos, el evento que cada proceso está esperando es la liberación de algún recurso que actualmente está en poder de otro miembro del conjunto. Dicho de otro modo, cada miembro del conjunto de procesos mutuamente bloqueado está esperando un recurso que está en poder de otro proceso en bloqueo. Ninguno de los procesos puede ejecutarse, ninguno puede liberar ningún recurso, y ninguno puede ser despertado. Ni el número de los procesos ni el número y tipo de los recursos poseídos y solicitados son importantes.

Condiciones para el bloqueo mutuo

Coffman et al. (1971) demostraron que deben cumplirse cuatro condiciones para que haya un bloqueo mutuo:

1. Condición de exclusión mutua. Cada recurso está asignado únicamente a un solo proceso o está disponible.

2. Condición de retener y esperar. Los procesos que actualmente tienen recursos que les fueron otorgados previamente pueden solicitar nuevos recursos.

3. Condicion de no expropiacion. No es posible quitarle por la fuerza a un proceso los recursos que le fueron otorgados previamente. El proceso que los tiene debe liberarlos explícitamente.

4. Condición de espera circular. Debe haber una cadena circular de dos o más procesos, cada uno de los cuales está esperando un recurso retenido por el siguiente miembro de la cadena.

Deben estar presentes estas cuatro condiciones para que ocurra un bloqueo mutuo. Si una o más de estas condiciones está ausente, no puede haber bloqueo mutuo.

HoIt (1972) mostró cómo pueden modelarse estas cuatro condiciones usando grafos dirigidos. Los grafos tienen dos clases de nodos: procesos, que se indican con círculos, y recursos, que se indican çon cuadrados. Un arco que va de un nodo de recurso (cuadrado) a uno de proceso (círculo) indica que el recurso fue solicitado previamente por el proceso, le fue concedido, y actualmente está en su poder. En la Fig. 3-7(a) el recurso R está asignado actualmente al proceso A.

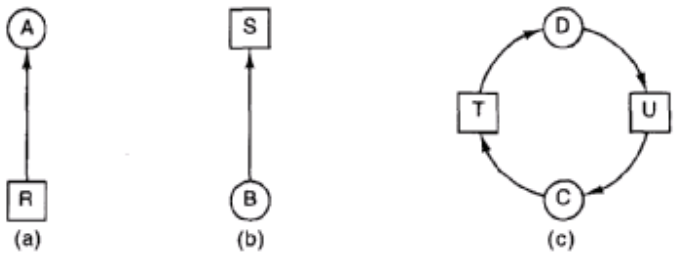

Figura 3-7. Grafos de asignación de recursos. (a) Retención de un recurso. (b) Petición de un recurso. (c) Bloqueo mutuo.

Un arco de un proceso a un recurso indica que el proceso está bloqueado esperando ese recurso. En la Fig. 3-7(b) el proceso B está esperando el recurso S. En la Fig. 3-7(c) vemos un bloqueo mutuo: el proceso C está esperando el recurso T, que actualmente está en poder del proceso E), El proceso D no va a liberar el recurso T porque está esperando el recurso U, que está en poder de C. Ambos procesos esperarán eternamente. Un ciclo en el grafo implica que hay un bloqueo mutuo en el que intervienen los procesos y recursos del ciclo. En este ejemplo, el ciclo es C-T-D-U-C.

Examinemos ahora un ejemplo de cómo pueden usarse los grafos de recursos. Imagine que tenemos tres procesos, A, 8 y C, y tres recursos, R, S y T. Las peticiones y liberaciones de los tres procesos se muestran en la Fig. 3-8(a)-(c). El sistema operativo está en libertad de ejecutar cualquier proceso no bloqueado en cualquier instante, de modo que podría decidir ejecutar A hasta que A terminara todo su trabajo, luego 8 hasta su finalización y por último C.

Este ordenamiento no da lugar a ningún bloqueo mutuo (porque no hay competencia por los recursos) pero tampoco tiene paralelismo. Además de solicitar y liberar recursos, los procesos calculan y realizan LIS. Cuando los procesos se ejecutan secuencialmente, no existe la posibilidad de que mientras un proceso está esperando LIS el otro pueda usar la CPU. Por tanto, ejecutar los procesos en forma estrictamente secuencial podría no ser óptimo. Por otro lado, si ninguno de los procesos realiza LIS, el algoritmo del primer trabajo más corto es mejor que el round robin, así que en algunas circunstancias lo mejor podría ser ejecutar todos los procesos secuencialmente.

Supongamos ahora que los procesos realizan tanto LIS como cálculos, de modo que el round robín es un algoritmo de planificación razonable. Las peticiones de recursos podrían presentarse en e1 orden que se indica en la Fig. 3-8(d). Si esas sE/S peticiones se llevan a cabo en ese orden, los

grafos de recursos resultantes son los que se muestran en la Fig. 3-8(e)-Q). Después de hacerse la petición 4, A se bloquea esperando S, como se aprecia en la Hg. 3-8(h). En los dos pasos B y C siguientes también se bloquean, dando lugar en última instancia a un ciclo y al bloqueo mutuo de la Fig. 3-8(j).

Sin embargo, como ya hemos mencionado, el sistema operativo no está obligado a ejecutar los procesos en un orden específico. En particular, si la concesión de una petición detenninada pudiera dar pie a un bloqueo mutuo, el sistema operativo podrá simplemente suspender el proceso sin dar respuesta a la petición (o sea, no planificar el proceso) hasta que pueda hacerlo sin peligro. En la Hg. 3-8, si el sistema operativo tiene conocimiento de un bloqueo mutuo inminente, podría suspender B en lugar de otorgarle S. Al ejecutarse sólo A y C obtendríamos las peticiones y liberaciones de la Fig. 3-8(k) en lugar de las de la Hg. 3-8(d). Esta secuencia da lugar a los grafos de recursos de la Hg. 3-8(l)-(q), que no conducen a bloqueo mutuo.

Después del paso (q), ya se puede otorgar S a B porque A ya terminó y C tiene todo lo que necesita. Incluso si B llegara a bloquearse al solicitar T, no podría ocurrir un bloqueo mutuo. B simplemente esperará hasta que C termine.

Más adelante en el capítulo estudiaremos un algoritmo detallado para tomar decisiones de asignación que no conducen a bloqueos mutuos. Lo que es importante entender ahora es que los grafos de recursos son una herramienta que nos permite ver si una secuencia de petición./libera ción dada conduce o no al bloqueo. Basta con indicar las peticiones y liberaciones paso por paso, determinando después de cada paso si el grafo contiene ciclos. Si los contiene, tenemos un bloqueo mutuo; si no, no hay bloqueo. Aunque nuestro tratamiento de los grafos de recursos corresponde al caso en que sólo hay un recurso de cada tipo, es posible generalizar los grafos para manejar múltiples recursos del mismo tipo (Holt, 1972).

En general, son cuatro las estrategias que se emplean para manejar el bloqueo mutuo:

1. Simplemente hacer caso omiso del problema.
2. Detección y recuperación.
3. Evitarlo de manera dinámica, mediante una asignación cuidadosa de los recursos.
4. Prevención, negando estructuralmente una de las cuatro condiciones necesarias. Examinaremos cada uno de estos métodos por turno en las siguientes cuatro secciones.

3.3.3 El algoritmo del avestruz

La estrategia más sencilla es el algoritmo del avestruz: meter la cabeza en la arena y pretender que el problema no existe. La gente reacciona a esta estrategia de diversas maneras. Los matemáticos la encuentran totalmente inaceptable y dicen que los bloqueos mutuos deben prevenirse a toda costa. Los ingenieros preguntan con qué frecuencia se espera que se presente el problema, qué tan seguido se cae el sistema por otras razones, y qué tan grave es un bloqueo mutuo. Si ocurren bloqueos mutuos una vez cada 50 años en promedio, pero las caídas del sistema debido a fallas de hardware, errores del compilador y defectos del sistema operativo ocurren una vez al mes, la mayoría de los ingenieros no estarían dispuestos a pagar un precio sustancial en términos de reducción del rendimiento o de la comodidad a fin de evitar los bloqueos mutuos.

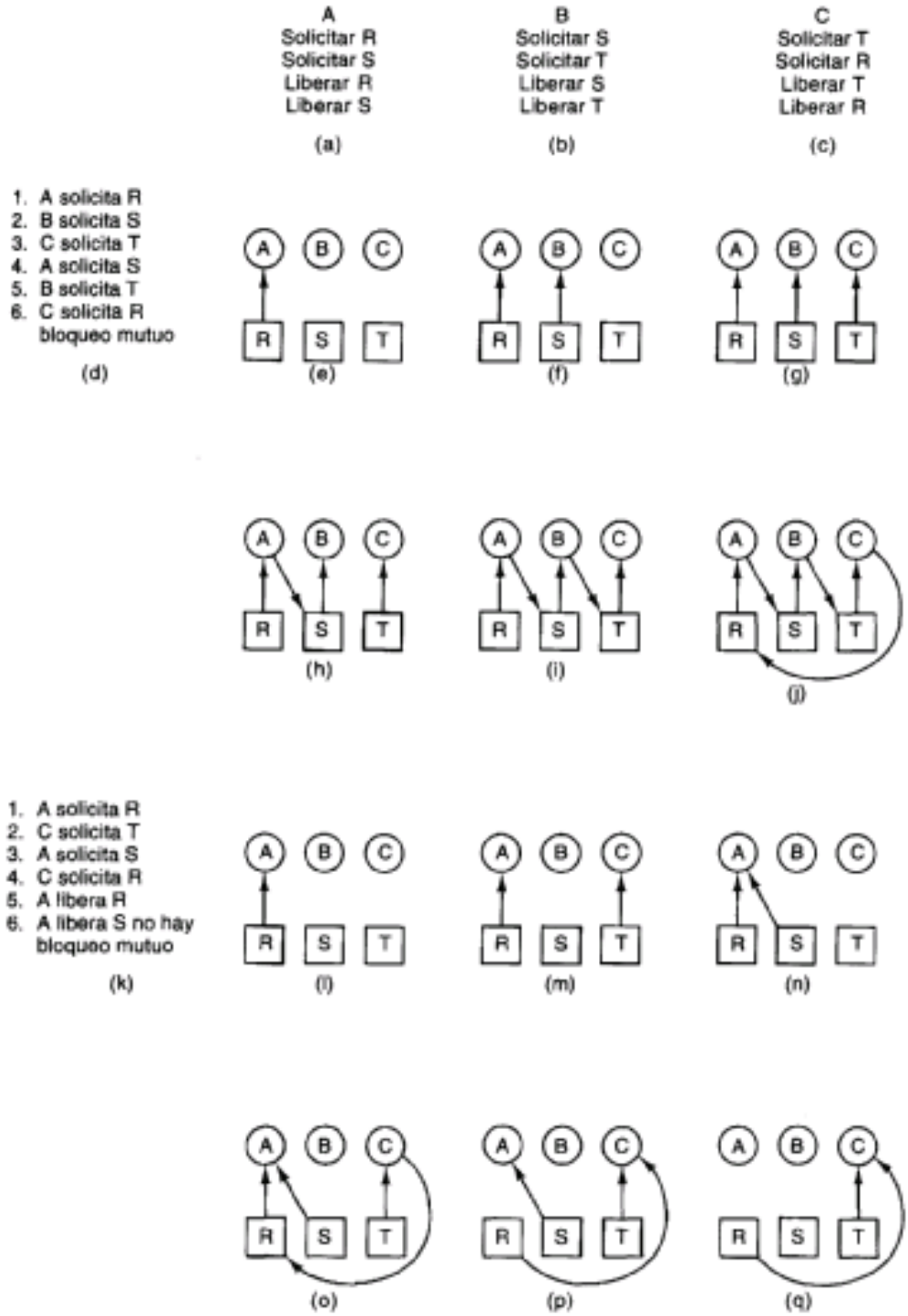

Figura 3-8. Ejemplo de cómo ocurre un bloqueo mutuo y cómo puede evitarse.

Para hacer este contraste más específico, UNIX (y MINIX) sufren potencialmente de bloqueos mutuos que ni siquiera se detectan, o mucho menos que se rompan automáticamente. El número total de procesos que hay en el sistema está determinado por el número de entradas de la tabla de procesos, así que las ranuras de la tabla de procesos son un recurso finito. Si un FORK falla porque la tabla está llena, una estrategia razonable para el programa que ejecuta el FORK sería esperar un tiempo aleatorio e intentarlo otra vez.

Supongamos ahora que un sistema UNIX tiene 100 ranuras para procesos. Se están ejecutando 10 programas, cada uno de los cuales necesita crear 12 (sub)procesos. Una vez que cada proceso ha creado 9 procesos, los 10 procesos originales y los 90 nuevos han agotado la tabla.

Ahora, cada uno de los 10 procesos originales se encuentra en un ciclo infinito en el que se bifurcan y fallan: bloqueo mutuo. La probabilidad de que esto suceda es pequeñísima, pero podría suceder. ¿Deberemos abandonar los procesos y la llamada FORK para eliminar el problema?

El número máximo de archivos abiertos está restringido de forma similar por el tamaño de la tabla de nodos-i, así que ocurre un problema similar cuando la tabla se llena. El espacio de intercambio (swapping) en el disco es otro recurso limitado. De hecho, casi todas las tablas del sistema operativo representan un recurso finito. ¿Debemos cancelar todos estos recursos porque podría suceder que una colección de n procesos solicitara un del total, y luego cada uno tratara de adquirir uno más?

La estrategia de UNIX consiste en hacer caso omiso del problema bajo el supuesto de que la mayoría de los usuarios preferirán un bloqueo mutuo ocasional en lugar de una regla que restrinja a todos los usuarios a un proceso, un archivo abierto, y una de cada cosa. Si los bloqueos mutuos pudieran eliminarse gratuitamente, no habría mucho que discutir. El problema es que el precio es alto, y principalmente consiste en poner restricciones molestas a los procesos, como veremos en breve. Así, enfrentamos un desagradable trueque entre comodidad y correción, y mucha disco sión acerca de qué es más importante.

3.3.4 Detección y recuperación

Una segunda técnica es la detección y recuperación. Cuando se usa esta técnica, el sistema no hace otra cosa que no sea vigilar las peticiones y liberaciones de recursos. Cada vez que un recurso se solicita o libera, se actualiza el grafo de recursos, y se determina si contiene algún ciclo. Si se encuentra uno, se termina uno de los procesos del ciclo. Si esto no rompe el bloqueo mutuo, se termina otro proceso, continuando así hasta romper el ciclo, Un método un tanto más burdo consiste en no mantener siquiera el grafo de recursos, y en vez de ello verificar periódicamente si hay procesos que hayan estado bloqueados continuamente durante más de, digamos, una hora. A continuación se terminan esos procesos.

La detección y ecuperación es la estrategia que a menudo se usa en las macrocomputadoras, sobre todo los sistemas por lotes en los que terminar y luego reiniciar un proceso suele ser aceptable. Sin embargo, se debe tener cuidado de restaurar todos los archivos modificados a su estado original, y revertir todos los demás efectos secundarios que pudieran haber ocurrido.

3.3.5 Prevención del bloqueo mutuo

La tercera estrategia para manejar el bloqueo mutuo consiste en imponer restricciones apropiadas abs procesos de modo que el bloqueo mutuo sea estructuralmente imposible. Las cuatro condiciones planteadas por Coffman et al. (1971) señalan algunas posibles soluciones. Si podemos asegurar que al menos una de esas condiciones nunca se satisfaga, el bloqueo mutuo será imposi ble (Havender, 1968).

Ataquemos primero la condición de exclusión mutua. Si ningún recurso se asignara de manera exclusiva a un solo proceso, jamás tendríamos bloqueo mutuo. Sin embargo, es igualmente obvio que permitir a dos procesos escribir en la impresora al mismo tiempo conduciría al caos. Si colocamos en spool las salidas a la impresora, varios procesos podrán generar salidas al mismo tiempo. En este modelo, el único proceso que realmente solicita la impresora física es el demonio de la impresora. Puesto que el demonio nunca solicita otros recursos, podemos eliminar el blo queo mutuo para la impresora.

Desafortunadamente, no todos los recursos se pueden manejar con spool (la tabla de procesos no se presta muy bien que digamos a ello). Además, la competencia misma por obtener espacio de disco para spool puede dar lugar al bloqueo. ¿Qué sucedería si dos procesos llenaran cada uno con sus salidas la mitad del espacio de spool disponible y ninguno terminara? Si el demonio se progra mara de modo que comenzara a imprimir antes de que toda la salida estuviera en spool, la impre sora podría permanecer ociosa si un proceso de salida decidiera esperar varias horas después de la primera ráfaga de salida. Por esta razón, los demonios se programan de modo que sólo impriman si ya está disponible todo el archivo de salida. Ninguno de los procesos terminaría, y tendríamos un bloqueo mutuo por el disco.

La segunda de las condiciones planteadas por Coffman et al. se ve más prometedora. Si pode mos evitar que los procesos que retienen recursos esperen para obtener más recursos, podremos eliminar los bloqueos mutuos. Una forma de lograr este objetivo es exigir que todos los procesos soliciten todos sus recursos antes de iniciar su ejecución. Si todo está disponible, se asignará al proceso todo lo que necesita y éste podrá ejecutarse hasta finalizar. Si uno o más recursos están ocupados, no se asignará nada y el proceso simplemente esperará.

Un problema inmediato de esta estrategia es que muchos procesos no saben cuántos recur sos van a necesitar antes de iniciar su ejecución. Otro problema es que los recursos no se aprovecha rán de manera óptima. Tomemos como ejemplo un proceso que lee datos de una cinta de entrada, los analiza durante una hora, y luego escribe una cinta de salida y además grafica los resultados. Si todos los recursos se solicitaran por adelantado, el proceso mantendría inaccesibles la unidad de cinta de salida y el graficador durante una hora.

Una forma un poco distinta de romper la condición de retener y esperar es exigir que un proceso que solicita un recurso libere primero temporalmente todos los recursos que está rete niendo en ese momento. Sólo si la petición tiene éxito podrá el proceso recibir de vuelta los recursos originales.

Atacar la tercera condición (no expropiación) es aún menos prometedor que atacar la segunda. Si a un proceso se le asignó la impresora y apenas ha imprimido la mitad de sus salidas,

quitarle forzosamente la impresora porque un graficador que se necesita no está disponible daría lugar a un desastre.

Sólo queda una condición. Se puede eliminar la espera circular de varias formas. Una de ellas consiste sencillamente en tener una regla que diga que un proceso sólo tiene derecho a un solo recurso en un instante dado. Si el proceso necesita un segundo recurso, deberá liberar el primero. Para un proceso que necesita copiar un archivo enorme de una cinta a una impresora, esta restricción es inaceptable.

Otra forma de evitar la espera circular es crear una numeración global de todos los recursos, como se muestra en la Fig. 3-9(a). Ahora la regla es ésta: los procesos pueden solicitar recursos cuando quieran, pero todas las peticiones deben hacerse en orden numérico. Un proceso puede solicitar primero una impresora y después una unidad de cinta, pero no puede solicitar primero un graficador y luego una impresora.

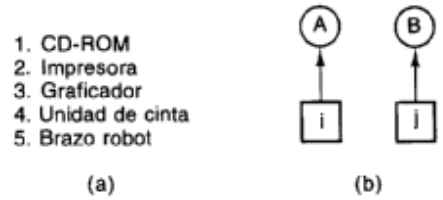

Figura 3-9. (a) Recursos ordenados numéricamente. (b) Un grafo de recursos.

Con esta regla, el grafo de asignación de recursos nunca puede tener ciclos. Veamos por qué esto se cumple para el caso de dos procesos [Fi.g 3-9(b)] Podemos tener un bloqueo mutuo sólo si A solicita el recurso j y B solicita el recurso i. Si suponemos que i y j son recursos distintos, tendrán diferente número. Si i >1, no se permitirá a A solicitar j. Si i <j, no se permitirá a B solicitar i. De cualquier manera, el bloqueo mutuo es imposible.

La misma lógica es válida para múltiples procesos. En todo instante, uno de los recursos asignados será el más alto. El proceso que esté reteniendo ese recurso nunca podrá solicitar uno que ya está asignado; o bien finalizará, o en el peor de los casos solicitará recursos con un número aún más alto, todos los cuales están disponibles. Tarde o temprano, ese proceso finalizará y liberará sus recursos. En este punto, algún otro recurso estará reteniendo el recurso más alto y también podrá finalizar. En pocas palabras, existe una situación en la que todos los procesos terminan, de modo que no hay bloqueo mutuo.

Una variación menor de este algoritmo consiste en omitir el requisito de que los recursos se adquieran en orden estrictamente creciente y sólo insistir en que ningún proceso solicite un recurso con un número menor que el del recurso que ya tiene en su poder. Si un proceso inicial- mente solicita los recursos 9 y 10, y luego los libera, en efecto estará de nuevo iniciando desde el principio, y no habrá razón para prohibirle ahora que solicite el recurso 1.

Aunque el ordenamiento numérico de los recursos elimina el problema de los bloqueos mutuos, puedc ser imposible encontrar un ordenamiento que satisfaga a todo mundo. Cuando los recursos incluyen ranuras de la tabla de procesos, espacio de disco para spool, registros de base de

datos con candado y otros recursos abstractos, el número de recursos potenciales y de posibles usos puede ser tan grande que ningún ordenamiento resulte práctico.

En la Fig. 3-10 se resumen las diferentes estrategias para prevenir los bloqueos mutuos.

Condición	Estrategia
Exclusión mutua	Poner todo en spool
Retener y esperar	Solicitar inicialmente todos los recursos
No expropiación	Quitar los recursos
Espera circular	Ordenar los recursos numéricamente

Figura 3-10. Resumen de estrategias para prevenir el bloqueo mutuo.

3.3.6 Evitar los bloqueos mutuos

En la Fig. 3-8 vimos que el bloqueo mutuo se evitaba no imponiendo reglas arbitrarias a los procesos sino analizando con detenimiento cada petición de recurso para ver si se puede satisfa cer sin peligro. Surge la pregunta: ¿hay algún algoritmo que siempre pueda evitar el bloqueo mutuo tomando la decisión correcta en todos los casos? La respuesta es que sí se puede evitar el bloqueo mutuo, pero sólo si se cuenta con cierta información por adelantado. En esta sección examinaremos formas de evitar los bloqueos mutuos mediante una asignación cuidadosa de los recursos.

El algoritmo del banquero para un solo recurso

Un algoritmo de planificación que puede evitar el bloqueo mutuo se debe a Dijkstra (1965) y se conoce como algoritmo del banquero. Este algoritmo toma como modelo la forma en que un banquero de una ciudad pequeña podría tratar con un grupo de clientes a los que ha concedido líneas de crédito. En la Fig. 3-li(a) vemos cuatro clientes, a cada uno de los cuales se ha otorgado cierto número de unidades de crédito (p. ej., 1 unidad = 1K dólares). El banquero sabe que no todos los clientes van a necesitar su crédito máximo de inmediato, así que sólo ha reservado 10 unidades en lugar de 22 para atenderlos, (En esta analogía, los clientes son procesos, las unidades de crédito son, digamos, unidades de cinta, y el banquero es el sistema operativo.)

Los clientes atienden sus respectivos negocios, haciendo peticiones de préstamos de vez en cuando. En un momento dado, la situación es la que se muestra en la Fig. 3-11(b). Una lista de los clientes junto con el dinero que ya se les prestó (unidades de cinta que ya se les asignaron) y el crédito máximo disponible (número máximo de unidades de cinta que se necesitarán al mismo tiempo posteriormente) se denomina estado del sistema respecto a la asignación de recursos.

Se dice que un estado es seguro si existe una secuencia de otros estados que conduzca a una situación en la que todos los clientes obtienen préstamos hasta sus límites de crédito (todos los procesos obtienen todos sus recursos y finalizan). El estado de la Fig. 3-11(b) es seguro porque, quedándole dos unidades, el banquero puede posponer todas las peticiones excepto la de Miguel,

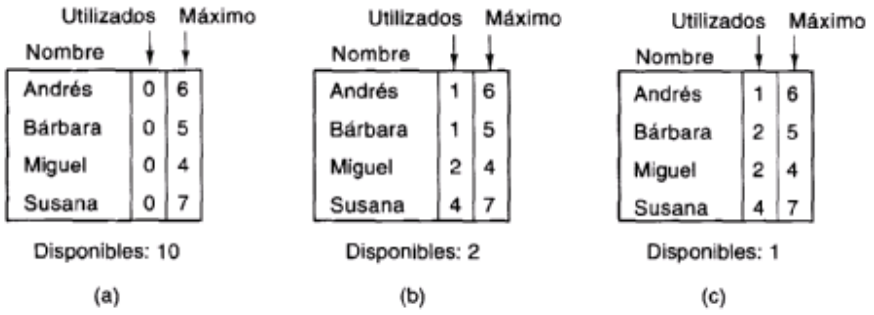

Figura 3-11. Tres estados de asignación de recursos: (a) Seguro. (b) Seguro. (c) Inseguro.

de modo que dejará que éste finalice y libere sus cuatro recursos. Teniendo cuatro recursos disponibles, el banquero puede otorgar a Susana o a Bárbara las unidades que necesiten, etcétera.

Consideremos lo que sucedería si se concediera a Bárbara una unidad más en la Fig. 3-11(b); tendríamos la situación de la Fig. 3-11(c) que es insegura. Si todos los clientes pidieran repentina mente sus préstamos máximos, el banquero no podría satisfacer ninguna de sus peticiones, y tendríamos un bloqueo mutuo. Un estado inseguro no tiene que dar pie al bloqueo mutuo, ya que un cliente podría no necesitar toda su línea de crédito disponible, pero el banquero no puede contar con que esto sucederá.

Por tanto, el algoritmo del banquero consiste en considerar cada petición en el momento en que se presenta y ver si su satisfacción conduce o no a un estado seguro. Si es así, se concede lo solicitado; si no, se pospone la petición. Para determinar si un estado es seguro, el banquero verifica si tiene suficientes recursos para satisfacer al cliente que está más cerca de su máximo. Si los tiene, se supone que esos préstamos ya fueron pagados, y a continuación se verifica el cliente que ahora está más cerca de su límite, y así sucesivamente. Si todos los préstamos pueden pagarse tarde o temprano, el estado es seguro y puede satisfacerse la petición inicial.

Trayectorias de recursos

El algoritmo anterior se describió en términos de una sola clase de recursos (p. ej., sólo unidades de cinta o sólo impresoras, pero no algunas de cada clase). En la Fig. 3-12 vemos un modelo para manejar dos procesos y dos recursos, por ejemplo, una impresora y un graficador. El eje horizontal representa el número de instrucciones ejecutadas por el proceso A. El eje vertical representa el número de instrucciones ejecutadas por el proceso B. En I A solicita una impresora; en ‘2’ A necesita un graficador. La impresora y el graficador son liberados en 13 e 14, respectivamente. El proceso J necesita el graficador de 15 a 17, y la impresora, de ‘6 a 18.

Cada punto del diagrama representa un estado conjunto de los dos procesos. Inicialmente, el estado está en p, donde ningún proceso ha ejecutado todavía instrucciones. Si el planificador decide ejecutar A primero, llegamos al punto q, en el que A ya ejecutó cierto número de instrucciones, pero Ji todavía no ejecuta ninguna. En el punto q la trayectoria se vuelve vertical, lo que

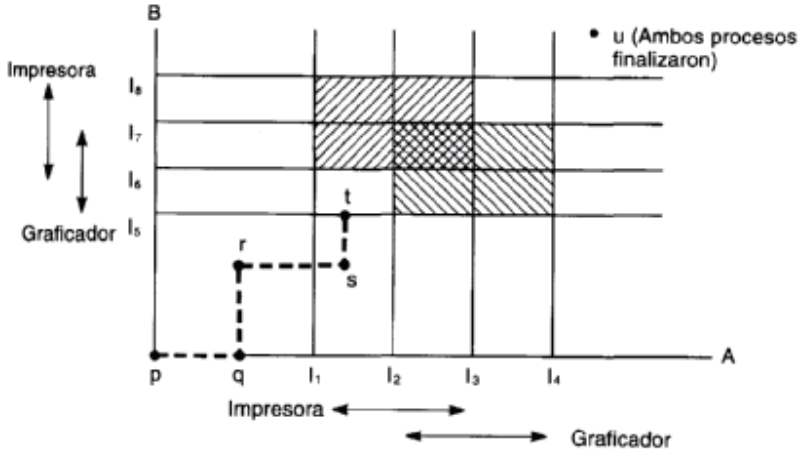

Figura 3-12. Dos trayectorias de recursos de procesos.

indica que el planificador decidió ejecutar B. Con un solo procesador, todas las trayectorias deben ser horizontales o verticales, nunca diagonales. Además, el movimiento siempre es hacia el norte o hacia el este, nunca hacia el sur o hacia el oeste (los procesos no pueden ejecutarse hacia atrás).

Cuando A cruza la línea I en la trayectoria de r a s, solicita y recibe la impresora. Cuando B llega al punto t, solicita el graficador.

Las regiones sombreadas son de especial interés. La región con líneas inclinadas del suroeste al noreste indican que ambos procesos tienen la impresora. La regla de exclusión mutua hace que sea imposible entrar en esta región. De forma similar, la región sombreada en la otra dirección indica que ambos procesos tienen el graficador, y es igualmente imposible.

Si el sistema alguna vez entra en el cuadro delimitado por 1 e ‘2 a los lados e 15 e ‘6 de arriba a abajo, nos encontraremos ante un bloqueo mutuo cuando llegue a la intersección de ‘2 e 16. En este punto, A está solicitando el graficador y B está solicitando la impresora, y ambos dispositivos ya están asignados. Todo el cuadro es inseguro y no debe entrarse en él. En el punto ¿t lo único seguro es ejecutar el proceso A hasta que llegue a 14. De ahí en adelante, cualquier trayectoria a u es buena.

El algoritmo del banquero para múltiples recursos

Este modelo gráfico es difícil de aplicar al caso general de un número arbitrario de procesos y un número arbitrario de clases de recursos, cada una con múltiples ejemplares (p. ej., dos graficadores, tres unidades de cinta). No obstante, el algoritmo del banquero puede generalizarse para este fin. En la Fig. 3-13 se muestra cómo funciona.

En la figura vemos dos matrices. La de la izquierda muestra cuántos ejemplares de cada recurso se han asignado actualmente a cada uno de los cinco procesos. La matriz de la derecha muestra cuántos recursos necesita todavía cada proceso para poder finalizar. Al igual que en el