TEMA 16

SISTEMAS OPERATIVOS: GESTIÓN DE PROCESOS

ÍNDICE

1. INTRODUCCIÓN

Concepto de proceso

La definición más general de *proceso* es un programa que se está ejecutando y consta tanto del programa como de los datos y los registros necesarios para llevar adelante la ejecución.

Proceso es cualquier actividad que realiza un procesador. Dicho procesador puede ser compartido por otros procesos siempre que exista un algoritmo de planificación conveniente. Un proceso es la expresión de la ejecución de programas para obtener un resultado determinado.

2. CARACTERÍSTICAS, ESTADOS Y ATRIBUTOS

Una característica fundamental de los procesos concurrentes es la *competencia* que se establece entre ellos cuando han de compartir recursos bien sean físicos o lógicos.

Dicha competencia viene dada por la imposibilidad de que dos procesos puedan actuar de manera simultánea sobre un mismo recurso.

Una segunda propiedad es la *cooperación.*

Cuando dos o más procesos se están llevando a cabo de manera que uno depende del otro es necesario que exista una cooperación entre ambos.

Podríamos definir una tercera característica de los procesos concurrentes que es la de poder formar *familias de procesos* según sean o no interdependientes.

Durante su existencia, un proceso puede pasar por diversas situaciones algunas de las cuales suponen una actividad y otras representan un estado inactivo.

Por tanto podemos definir los siguientes **estados en la ejecución de un proceso**: **En**

ejecución: el proceso se encuentra activo utilizando la CPU. **Preparado/Listo**:

el proceso es capaz de estar activo si tuviese CPU disponible.

Detenido/Bloqueado: en espera de que suceda algún evento tras lo cual recupere su actividad (por ejemplo, una operación de E/S).

Con el fin de lograr un control perfecto en la ejecución de los procesos por parte del S.O. es necesario informar a este de una serie de **atributos propios de cada proceso**:

- *Nombre del proceso*. Denominación indispensable para hacer referencia al mismo desde otro proceso.

- *Estado actual*. Situación en que se encuentra un proceso en un instante dado. Cuando un proceso que se encontraba activo pasa a ser inactivo, todos los parámetros que determinan su situación son guardados en su PCB (Bloque de control de proceso).

- *Prioridad*. En el caso en que la asignación de la CPU a los distintos procesos se efectúe mediante definición de prioridades, será necesario fijar éstas.

- *Derechos*. Cuando son varios los recursos a compartir, es necesario declarar cuales son los derechos que sobre cada uno de los recursos disponibles posee un determinado proceso.

3. PLANIFICACIÓN DE PROCESOS

La planificación de procesos es un conjunto de mecanismos incorporados al S.O. que establecen el orden en que se van a realizar los diferentes procesos.

Los procesos localizados en la memoria pueden entrar y salir de la CPU en cualquier instante, sin que nosotros tengamos control sobre ellos.

Por ello, para calcular los tiempos de estancia y de retorno suponemos que el tiempo que la CPU está trabajando se reparte por igual entre todos los procesos de la memoria.

Concretamente si en la memoria existe un proceso el tiempo de la CPU es todo para él, pero si existen dos, tres o más procesos cada uno recibirá la mitad, tercera o "n" parte del tiempo que la unidad central emplee.

Existirán pues varios contadores de tiempo uno para la unidad central que corresponderá con el tiempo de reloj y otros que señalen el tiempo que lleva cada proceso utilizando el procesador.

3.1. Multiprogramación sin límite de memoria

En este caso los procesos que van llegando entran en memoria sin ninguna dificultad y comparten el tiempo de uso de la unidad central con los procesos que ya estaban en memoria.

El tiempo que la unidad central dedica a cada trabajo se llama adelanto de CPU.

Sí en memoria existen dos procesos, el adelanto de CPU de cada proceso es la mitad del tiempo transcurrido de reloj para la unidad central.

Si no tenemos en cuenta los tiempos de E/S, en memoria y CPU no existe ningún tiempo de retardo: cuando llega un proceso a la cola entra en ejecución.

No es necesario pues ningún tipo de planificación de los procesos para entrar en memoria.

Para aclarar las ideas veamos el siguiente ejemplo de la figura donde se representan seis procesos con sus horas de llegada y el tiempo de CPU que necesitan. Los tiempos de ejecución y las horas de llegada están en forma decimal para facilitar los cálculos. Es decir, las 8,50 significará las 8 y media.

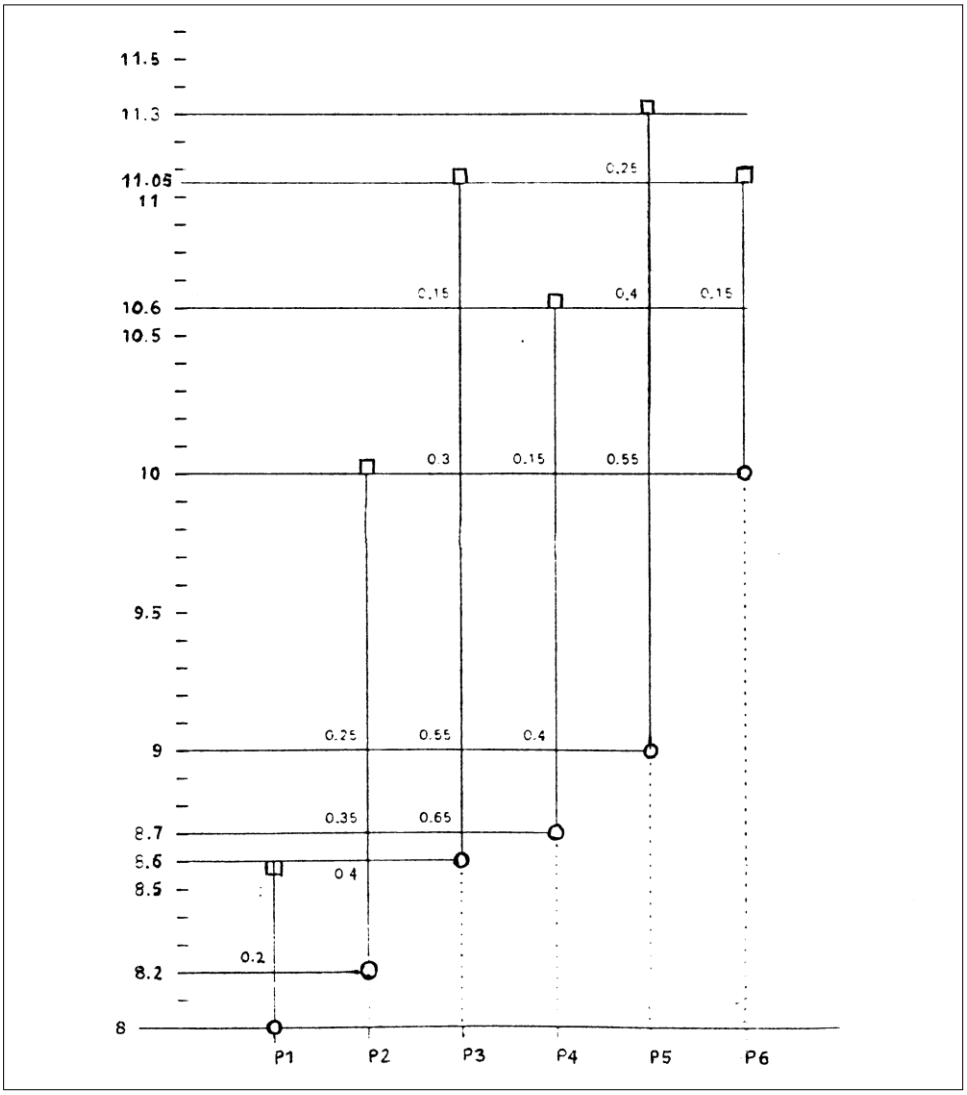
PROCESO	HORA LLEGADA	TIEMPO EJECUCION
P1	8.00	0,4
P2	8.20	0,6
P3	8.60	0,7
P4	8.70	0,5
P5	9.00	0,8
P6	10.00	0,3

La figura siguiente nos muestra el desarrollo de los procesos suponiendo que no hay multiprogramación y utilizamos un algoritmo FIFO (primero en entrar, primero en salir),

PROCESO	HORA LLEGADA	TIEMPO DE EJECUCION	HORA DE INICIO	HORA FINAL	TIEMPO DE RETORNO
P1	8.00	0,4	8.00	8.40	0.40
P2	8.20	0,6	8.40	9.00	0.80
P3	8.60	0,7	9.00	9.70	1.10
P4	8.70	0,5	9.70	10.20	1.50
P5	9.00	0,8	10.20	11.00	2.00
P6	10.00	0.3	11.00	11.03	1.30

$$Tm = \frac{0.4+0.8+1.1+1.5+2.0+1.3}{6} = \frac{7.1}{6} = 1.2$$

Si el sistema fuese multiprogramado nos resultaría la figura siguiente:



3.2. Multiprogramación con memoria limitada

Cuando la memoria no es suficiente para contener todos los procesos es necesario algún tipo de planificación para decidir qué proceso o procesos son los que entran primero.

Es necesario tener un mayor conocimiento de los procesos como puede ser su prioridad, su tiempo de ejecución o cualquier otro detalle que facilite la elección del algoritmo apropiado.

Además necesitaremos saber la memoria que ocupa cada proceso que suele ser complejo ya que según la ejecución puede variar de tamaño.

Vamos a suponer resueltos esos pequeños inconvenientes y que tenemos una memoria de 100 Kbytes donde debemos colocar los procesos de la siguiente figura.

NUMERO PROCESO	HORA LLEGADA	TIEMPO CPU	MEMORIA	PRIORIDAD
1	0	1	20 K	1
2	0.5	1	60 K	2
3	1	0.5	10 K	3
4	1.5	2	30 K	3
5	2	0.2	20 K	1
6	2.5	0.1	20 K	2

Para decidir que procesos son los primeros en ejecutarse les dotamos de una prioridad. El desarrollo de los mismos queda reflejado en la siguiente tabla:

INSTANTE	TIEMPO CPU/PROCESO	PROCESOS
0		1 (1,0)
	0,5	
0.5		1 (0,5) (2 (1,0)
	0,25	
1		1 (0,25) 2 (0,75) 3 (0,5)
	0,16	
1.5		1 (0,09) 2 (0,59) 3 (0,34)
	0,09	
1.77		1 (0) 2 (0 5) 3 (0,25)
	0,12	
2		2 (0,38) 3 (0,13) 5 (0,2)
	0,13	
2.4		2 (0,25) 3 (0) 5 (0,07)
	0,05	
2.5		2 (0,20) 5 (0,02) 6 (0,1)
	0,02	
2.56		2 (0,18) 5 (0) 6 (0,08)
	0,08	
2.72		2 (0,1) 6 (0) 4 (2,0)
	0,1	
2.93		2 (0) 4 (1 9)
	1,9	
4.83		4 (0)

En la primera columna de esta figura y de arriba hacia abajo representamos el tiempo transcurrido señalando los instantes en que acontece algún suceso.

En la segunda el adelanto de CPU para cada proceso implicado.

En la siguiente representamos en nombre del proceso y entre corchetes el tiempo que aún le queda para concluir su ejecución.

4. CONCURRENCIA DE PROCESOS

En un sistema multiprogramado con un único procesador, los procesos se intercalan en el tiempo para dar la apariencia de ejecución simultánea. Aunque no se consigue un proceso paralelo real y aunque se produce cierta sobrecarga en los intercambios de procesos de un sitio a otro, la ejecución intercalada produce beneficios importantes en la eficiencia del procesamiento y en la estructuración de los programas.

En un sistema con varios procesadores, no sólo es posible intercalar los procesos, sino también superponerlos en el tiempo.

Para estudiar las diferencias entre la ejecución secuencial y ejecución concurrente nos serviremos del siguiente *ejemplo*:

Vamos a preparar las cartas de aviso al cobro del impuesto local de la contribución urbana.

El programa realizará las siguientes operaciones:

- Recogida de datos.
- Almacenamiento de los mismos.
- Preparación de las etiquetas.
- Cálculos.
- Impresión de las cartas de pago.

El proceso de impresión de las cartas solo podrá realizarse cuando se hayan hecho los cálculos y preparado las etiquetas. Por ello se establece la relación: "Los cálculos preceden a la impresión" llamada relación de precedencia.

En la figura siguiente se refleja el desarrollo de los procesos secuenciales.

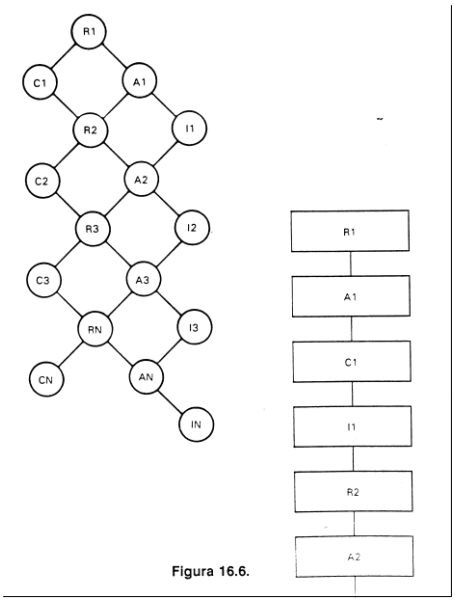


Figura 16.6.

Sin embargo, mientras se preparan las etiquetas se pueden ir realizando los cálculos o leyendo datos de otro contribuyente. Dichos procesos se realizan de forma simultánea y se denominan procesos concurrentes.

5. SINCRONIZACIÓN DE PROCESOS

En la ejecución concurrente de procesos se daban relaciones de forma que algunos procesos no pueden empezar su ejecución hasta que otros hayan concluido.

Para que un grupo de procesos funcione de forma adecuada se deben sincronizar sus actividades para asegurar las relaciones de precedencia de los procesos implicados.

Una forma de lograrlo es mediante la *generación y envío de señales.*

Cuando un proceso está a la espera de señales su ejecución es suspendida por el S.O. hasta que llegan las señales requeridas.

Entre los procesos implicados debe darse una cooperación para la ejecución de tareas comunes y una competencia para la utilización de los recursos compartidos.

6. EXCLUSIÓN MUTUA

Lo que se pretende con los sistemas que utilizan la multiprogramación es optimizar el uso de los recursos disponibles, y para ello los procesos se van intercalando para aprovechar los tiempos muertos de la CPU.

Pero si un proceso abandona la CPU cuando una determinada variable tiene un valor, y otro proceso entra en escena pudiendo modificar dicha variable, estamos ante el caso de variables compartidas y a estas secciones de programa se les denomina **secciones críticas**.

Una solución pasa por permitir el acceso al recurso compartido al proceso que ejecuta la sección critica impidiendo el mismo a los demás procesos su ejecución hasta el termino del mismo.

Dicha solución se le conoce como **exclusión mutua**.

Como la idea es que mientras que se ejecuta una sección crítica el proceso no sea interrumpido, podemos establecer una serie de condiciones para que se de la exclusión mutua.

6.1. Algoritmos de exclusión mutua

El algoritmo más elemental para conseguir la exclusión mutua entre procesos consiste en disponer de una variable global, accesible a todos los procesos implicados, en la que se recoge la capacitación de un proceso para ejecutar su sección critica, impidiendo a los demás el acceso a la CPU hasta que el proceso primero no termine su ejecución excluyente. En dicho instante el proceso primero cederá la CPU a otro mediante la actualización del valor de la variable.

De lo dicho antes, podemos extraer que la alternancia entre secciones se realiza sin tener en cuenta la conveniencia del sistema.

A su vez, si el primer proceso falla los demás quedaran bloqueados al no poder acceder a la variable que gobierna todo el proceso.

Para resolver dichos problemas se concibió un segundo algoritmo cuya idea fundamental era la de sustituir la variable global de turno por una serie de variables, una por proceso, que permiten a un proceso determinado señalar a los demás el hecho de encontrarse realizando su sección crítica.

Cada proceso antes de entrar en su sección crítica, consulta el estado de los demás simbolizado en su variable de estado.

6.2. Algoritmo de Dekker

Dijkstra presentó un algoritmo de exclusión mutua para dos procesos que diseño el matemático holandés Dekker. Esta es la primera solución correcta conocida, del tipo software, al problema de la sección crítica para dos procesos.

6.3. Semáforos

Una mejora cualitativa para manejar las secciones críticas fue presentada por Dijkstra que propuso un mecanismo lógico llamado *semáforo* que garantiza la exclusión mutua.

El principio fundamental es el siguiente: Dos o más procesos pueden cooperar por medio de simples señales, de forma que se pueda obligar a detenerse a un proceso en una posición determinada hasta que reciba una señal específica. Para la señalización se utilizan variables especiales denominadas semáforos. Para transmitir una señal por el semáforo *s*, los procesos ejecutan la primitiva *signal(s)*. Para recibir una señal del semáforo *s*, los procesos ejecutan la primitiva *wait(s)*.

Un semáforo puede tener el valor cero, lo que indica que no se han guardado desbloqueos o algún valor positivo si quedan pendientes uno o más desbloqueos. Se pueden contemplar los semáforos como variables que tienen un valor entero sobre el que se definen las tres operaciones siguientes:

1. Un semáforo puede inicializarse con un valor no negativo.
2. La operación ***wait*** verifica si el valor es mayor que cero. Si es así, decrementa el valor del semáforo (es decir, utiliza un desbloqueo almacenado). Si el valor es cero, el proceso que ejecuta el wait se bloquea.
3. La operación ***signal*** incrementa el valor del semáforo. Si uno o más procesos estaban bloqueados en ese semáforo, el sistema elige uno de ellos y le permite completar su operación wait.

Aparte de estas tres operaciones, no hay otra forma de examinar o manipular los semáforos.

El semáforo en general se utiliza para gestionar los recursos físicos del sistema.

Un *semáforo binario* sólo puede tomar los valores 0 y 1. Es más sencillo de implementar y tienen prácticamente la misma potencia de expresión que los semáforos generales.

En los semáforos se emplea una cola para mantener los procesos esperando en el semáforo. La política más equitativa es la FIFO: el proceso que ha estado bloqueado durante más tiempo se libera de la cola.

6.4. Algoritmo de Camport

Se denomina *algoritmo de la panadería*. Es similar a la planificación de una tienda en la que cada cliente toma un número y se le sirve a aquel que tenga el número más bajo.

Sin embargo este algoritmo no prevé que dos procesos puedan tomar el mismo número.

7. TRATAMIENTOS DE HARDWARE

Tanto en los semáforos como en los anteriores algoritmos, la exclusión mutua de los procesos, sólo estaría asegurada si todos los procesos siguieran las particulares reglas del juego implantadas.

Si uno solo de los procesos olvidase proteger su sección crítica, estaría comprometiendo la, perfecta, ejecución de los demás procesos, al poder ejecutar su sección crítica en concurrencia con otras. Por tanto no aseguraríamos la coherencia general del sistema.

Para llevar a efecto la ejecución concurrente de procesos, es necesario introducir mecanismos de programación que tengan en consideración la posibilidad de que dos secciones críticas quieran ejecutarse concurrentemente.

Una vez quedó clara esta necesidad, tanto los desarrolladores de software como los diseñadores de hardware intentaron aportar mecanismos al conjunto de instrucciones de la propia máquina.

Estas instrucciones son decodificadas directamente por la CPU y son programables desde un lenguaje de bajo nivel.

7.1. DI/EI: Inhabilitación de interrupciones

Esta doble instrucción *Interrupción no permitida/Interrupción permitida*, actúa sobre la CPU como un interruptor.

En una máquina monoprocesador los procesos sólo pueden intercalarse. Un proceso se continuará ejecutando hasta que solicite un servicio del sistema operativo o hasta que sea interrumpido. Por tanto, para garantizar la exclusión mutua, es suficiente con impedir que un proceso sea interrumpido.

Cuando la CPU recibe de un proceso la instrucción DI queda bloqueada para cualquier otro proceso, ejecutando en exclusiva el proceso que mandó esta instrucción.

Esta instrucción al actuar directamente sobre la CPU detiene la ejecución de todos los demás procesos, tengan o no secciones críticas en situación de ejecutarse.

Sin embargo, cuando el sistema tenga más de un procesador, es posible (y habitual) que haya más de un proceso ejecutándose al mismo tiempo. En este caso, inhabilitar las interrupciones no garantiza la exclusión mutua.

7.2. TS: Test and Set

La instrucción de TS, *comparar y asignar*, es la implantación física más fiel al mecanismo propuesto por la solución lógica "semáforo".

Esta instrucción resuelve situaciones de intromisión, haciendo indivisibles los dos pasos de comprobar si la variable semáforo estaba en estado "libre" y de asignar el nuevo valor a la variable.

Actúa como una eficaz protección de la memoria en sistemas de multitarea y multiusuario. Cuando hay zonas de datos accesibles para la escritura y lectura por varios procesos es necesario que el acceso a estas zonas sea regulado para que no se de la situación de simultaneidad al actualizar los mismos.

Para ello utiliza un bit de acceso por zona de datos definida; si el bit está a cero lo pone a uno y devuelve cierto. Entonces el proceso empieza a utilizar en exclusiva esa zona de datos. En otro caso, el valor no se modifica y se devuelve falso. La función TS se ejecuta inmediatamente en su totalidad, es decir, no está sujeta a interrupciones.

7.3. CS: Intercambiar

Esta instrucción responde a la búsqueda de otras vías que no fueran la de exclusión mutua para asegurar la coherencia de la ejecución de procesos concurrentes.

Se apoya en la idea de controlar, cuando un proceso quiere actualizar la variable en conflicto, si otro ya lo ha hecho. Si es así el proceso vuelve a intentarlo, hasta que por fin procede a la actualización de dicha variable y continúa.

Esta instrucción intercambia el contenido de un registro con el de una posición de memoria. Durante la ejecución de la instrucción, se bloquea el acceso a la posición de memoria de cualquier otra instrucción que haga referencia a la misma posición.

La utilización de TS es recomendable cuando es previsible se vayan a producir muchas situaciones de colisión.

Cuando no se prevé gran número de situaciones conflictivas se utiliza CS.

8. INTERBLOQUEO

8.1. Definición y características

Se puede definir el interbloqueo como el bloqueo *permanente* de un conjunto de procesos que compiten por los recursos del sistema o bien se comunican unos con otros.

Todos los interbloqueos suponen demandas contradictorias de recursos por parte de dos o más procesos.

Un interbloqueo siempre tiene solución si se retiran los procesos que la producen; esta retirada se denomina reanudación.

Otras soluciones pueden ser:

* Establecer protocolos para asegurar que no se produce el interbloqueo.
* Asignar prioridades aunque esta solución puede producir inanición.
* Mecanismo que nos informe el estado de todos los recursos.

Los recursos se dividen en varios tipos cada uno de los cuales consta de un número de instancias.

Un recurso puede ser una impresora. Si hay varias impresoras se dice que el tipo impresora tiene varias instancias. Si al solicitar una instancia de un tipo de recurso esta solicitud es satisfecha por cualquier instancia del tipo, se dice que las instancias son idénticas.

Un conjunto de procesos se encuentra en estado de interbloqueo cuando cada proceso del conjunto está esperando un suceso que solo puede ser causado por otro proceso del conjunto.

Puede producirse interbloqueo si y sólo si se dan simultáneamente las siguientes **condiciones**:

- *Exclusión mutua*: Al menos un recurso se retiene en forma no compartible.

- *Retener y esperar*: Tiene que existir un proceso que retenga al menos un recurso y que esté esperando para adquirir otros recursos que actualmente estén retenidos.

- *No-apropiación*: La propiedad de los recursos no puede ser revocada.

- *Espera circular*: Tiene que existir un conjunto de procesos en espera, cada uno de los cuales retiene, al menos, un recurso que necesita el siguiente proceso de la cadena.

8.2. Métodos para tratar interbloqueos

Para tratar los interbloqueos existen dos tipos de métodos. Aquellos que intentan que no se produzcan y los que los tratan una vez que se han producido.

Entre los primeros destacan la prevención y la predicción.

En cuanto a los segundos hablaremos de detección y recuperación.

Prevención

Consiste en lograr que al menos una de las cuatro condiciones necesarias no se verifique.

En general, la primera de las cuatro condiciones no puede anularse. La condición de retención y espera puede prevenirse exigiendo que todos los procesos soliciten todos los recursos que necesiten a un mismo tiempo y bloqueando el proceso hasta que todos los recursos puedan concederse simultáneamente. La condición de no apropiación puede prevenirse de varias formas, por ejemplo, obligando a que los procesos liberen sus recursos cuando se les deniega una nueva solicitud. Por último, la condición de espera circula puede prevenirse definiendo una ordenación lineal de los tipos de recursos.

Predicción

Los métodos de prevención del interbloqueo suelen dar como efecto secundario una baja utilización de los recursos y por tanto un bajo rendimiento del sistema.

Con la predicción del interbloqueo, se decide dinámicamente si la petición actual de asignación de un recurso podría, de concederse, llevar potencialmente a un interbloqueo. Necesita, por tanto, conocer las peticiones futuras de recursos. Existen dos enfoques:

No iniciar un proceso si sus demandas pueden llevar a un interbloqueo.
No conceder una solicitud de incrementar los recursos de un proceso si esta asignación puede llevar a interbloqueo.

Detección

Cuando no se puede asegurar que no se va a producir un interbloqueo es necesario disponer de esquemas de detección y recuperación.

Para detectar si existe un interbloqueo en el sistema se ejecuta periódicamente algún algoritmo de detección que permite detectar la condición de espera circular.

La frecuencia de su uso dependerá del sistema y si hay más o menos tendencia al interbloqueo.

Recuperación

Para recuperar un sistema que ha caído en interbloqueo es necesario que uno o varios procesos detengan su ejecución y regresen a algún punto anterior de la misma.

Se necesita alguna estrategia de recuperación. Las técnicas siguientes son posibles enfoques, enumeradas en orden creciente de sofisticación:

1. Abandonar todos los procesos bloqueados.
2. Retroceder cada proceso bloqueado hasta algún punto de control definido previamente y volver a ejecutar todos los procesos.
3. Abandonar sucesivamente los procesos bloqueados hasta que deje de haber interbloqueo.
4. Apropiarse de recursos sucesivamente hasta que deje de haber interbloqueo.

9. BIBLIOGRAFÍA

Stallings, W. ***Sistemas operativos*** Prentice Hall, 2ed., 1997

Tanenbaum, Andrew S. ***Sistemas operativos: Diseño e implementación*** Prentice Hall, 1990