Capítulo 5

Interbloqueos

La existencia de recursos críticos puede desembocar en una situación no deseada dentro del sistema, el interbloqueo. Éstos afectan al rendimiento, ya que dejan procesos bloqueados que tienen asignados recursos. En este tema veremos cuáles son las condiciones que deben darse para que se produzca este problema, así como las técnicas para abordarlo.

5.1. Introducción

Durante la ejecución de los procesos, estos utilizan un conjunto de recursos. La secuencia de pasos que debe dar el proceso para utilizarlos es la siguiente:

1. **Solicitud** Cuando un proceso solicita un recurso, el sistema operativo comprueba si éste ha sido pedido y asignado a otro. Para ello cuenta con una tabla del sistema donde se registra qué recursos están libres y cuáles asignados, y si es así, a qué proceso lo está. Si un proceso pide un recurso que está asignado a otro, se añade a una cola de procesos que están esperando ese recurso, es decir, pasa a estado bloqueado hasta que esté disponible. Si existen recursos del tipo solicitado, se le asigna.

2. **Uso** El proceso puede operar con el recurso.

3. **Liberación** Una vez que termina su utilización, el proceso libera el recurso.

La solicitud, uso y liberación de los recursos sólo se puede realizar a través de los servicios del sistema. Así, son ejemplos de solicitud y liberación de recursos las llamadas al sistema open y close de apertura y cierre de ficheros, respectivamente. Las llamadas read y write para la lectura y escritura en un fichero son ejemplos de uso de un recurso. Una de las funciones principales del sistema operativo es la gestión de los recursos del sistema, para lo cuál se encarga de asignar, desasignar y llevar el control de éstos. Cuando un proceso necesita un recurso, lo tiene que solicitar al sistema mediante los servicios que éste proporciona.

Los recursos se pueden clasificar atendiendo a distintos criterios:

- En función del número de procesos que lo pueden utilizar:

  Recursos compartibles Pueden ser usados por más de un proceso simultáneamente. Éstos no causan problemas de concurrencia entre procesos, pues pueden usarlos simultáneamente. Ejemplos de este tipo de recurso son los discos, ficheros de sólo lectura, etc.

  Recursos no compartibles o críticos En este caso pueden ser usados por más de un proceso pero no simultáneamente como en el caso anterior. Aquí sí es necesario tener un control de la concurrencia entre los procesos. Ejemplos de este tipo son las impresoras, las cintas magnéticas, etc.

- En función del tiempo de vida del recurso:

  Recursos reutilizables Son aquellos que no se agotan, ni destruyen con su uso. En este caso, los procesos obtienen un recurso que liberan posteriormente para que pueda ser reutilizado por otros procesos. Ejemplos de este tipo de recurso lo constituyen el procesador, memoria principal y secundaria, dispositivos, y estructuras de datos como ficheros, bases de datos, etc. Existe, por norma general, un número fijo de recursos reutilizables.

  Recursos consumibles Son los que pueden ser creados (producidos) y destruidos (consumidos). Normalmente, no hay límite en el número de recursos consumibles de un determinado tipo. Cuando un proceso lo adquiere, éste deja de existir. Ejemplos de este tipo de recurso son las interrupciones, señales, mensajes, e información en un *buffer* de E/S.

En este tema nos vamos a ocupar de los recursos críticos ya que pueden provocar la aparición de una serie de problemas en el sistema.

Cuando un proceso solicita un recurso pueden ocurrir dos cosas, una es que esté disponible, en este caso se le asignará y el proceso podrá continuar su ejecución; otra posibilidad es que el recurso esté asignado a otro proceso, por lo que el primero tendrá que pasar a estado bloqueado a la espera del recurso.

Puede darse una situación en la que los procesos que están bloqueados nunca cambien de estado, porque los recursos que han solicitado están asignados a otros procesos que también están bloqueados. Esta situación se denomina interbloqueo[1].

Podemos definir un interbloqueo como el bloqueo permanente de un conjunto de procesos que compiten por los recursos del sistema. De esta forma, un conjunto de procesos se encuentra en estado de interbloqueo, cuando cada uno de ellos está esperando un suceso que sólo puede ser causado por otro proceso del mismo conjunto, y que nunca se producirá porque todos están bloqueados.

A diferencia de otros problemas que surgen entre los procesos concurrentes, no existe para éste una solución eficiente. En este capítulo, examinaremos la naturaleza del problema del interbloqueo, y la forma en que los sistemas operativos pueden tratarlo.

5.1.1. Ejemplos de interbloqueos

El interbloqueo es una situación que no tiene por qué producirse siempre que se ejecuten los mismos procesos, sino que depende del orden exacto en que se ejecuten las instrucciones. A continuación veremos unos ejemplos.

5.1.1.1. Con recursos reutilizables

Consideremos dos procesos que compiten por el acceso exclusivo de un fichero F del disco y a una unidad de cinta C. Los programas están dedicados a las siguientes tareas repetitivas:

```
P1:   solicitar(F)          P2:   solicitar(C)
      ...                         ...
      solicitar(C)                solicitar(F)
      ...                         ...
      liberar(C)                  liberar(F)
      ...                         ...
      liberar(F)                  liberar(C)
```

[1] También se conoce como abrazo mortal, bloqueo mutuo o *deadlock*.

El interbloqueo puede producirse si cada proceso retiene un recurso y pide el otro. En este ejemplo, si se alternan las instrucciones de cada proceso, $P1$ obtendrá el fichero F pero se quedará esperando la cinta C; mientras que $P2$ obtendrá ésta, pero quedará a la espera del fichero F.

5.1.1.2. Con recursos consumibles

Vamos a considerar dos procesos que se intercambian mensajes y veremos que en este caso también se pueden producir interbloqueos.

```
P1:   receive (P2, M);        P2:   receive (P1, Q);
      send (P2, N);                 send (P1, R);
```

En este caso, ambos procesos se quedan esperando la llegada de un mensaje que nunca llegará.

5.2. Condiciones necesarias

Coffman, Elphick y Shoshani establecen en 1971 que para que se produzca un interbloqueo deben darse simultáneamente las cuatro condiciones siguientes:

1. **Exclusión mutua**: Sólo un proceso puede usar un recurso a un tiempo.
2. **Retener y esperar**: Los procesos retienen los recursos ya asignados, mientras esperan la asignación de otros para continuar con su ejecución.
3. **No apropiación**: No se puede quitar a la fuerza los recursos asignados a un proceso, éste debe liberarlos voluntariamente cuando no los necesite.
4. **Espera circular**: Existe una cadena circular de procesos en la cuál cada proceso tiene uno o más recursos que son requeridos por el siguiente proceso en la cadena. Formalmente, hay un conjunto de procesos en espera $\{p_0, \ldots, p_n\}$, tal que el proceso p_i está esperando un recurso retenido por p_{i+1}, para todo $i = \{0, \ldots, n\}$; y, p_n está esperando un recurso retenido por p_0.

Las tres primeras condiciones son necesarias, pero no suficientes, para que exista interbloqueo. La cuarta condición es en realidad, una consecuencia potencial de las tres primeras. Es decir, dado que se producen las tres

primeras condiciones, puede ocurrir una secuencia de eventos que desemboque en una espera circular irresoluble. Esta espera circular es de hecho la definición de un interbloqueo. A pesar de ser la definición, para que no se pueda resolver la espera circular han de cumplirse las condiciones anteriores.

Existe una diferencia fundamental entre las tres primeras condiciones y la última. Las tres primeras son decisiones de política del sistema operativo, mientras que la cuarta es una circunstancia que podría darse o no en función de la secuencia de peticiones y liberaciones de recursos por parte de los procesos.

5.3. Modelado del interbloqueo

En 1972, Holt demostró cómo pueden ser modeladas las cuatro condiciones mediante los denominados **grafos de asignación de recursos**, en los que se indican las asignaciones y peticiones de recursos en un sistema.

Estos grafos usan la siguiente notación gráfica:

- Los círculos representan procesos.
- Los cuadrados grandes representan clases o tipos de recursos críticos.
- Los círculos pequeños dentro de los anteriores indican las unidades que existen de cada tipo de recurso.
- Los arcos orientados representan:

  Asignación Si el arco va del vértice que representa una instancia de un tipo de recurso al que representa un proceso, indica que el recurso está asignado al proceso (figura 5.1(a)).

  Solicitud Si el arco va de un vértice que representa un proceso a uno que representa un tipo de recurso, indica una petición del recurso (figura 5.1(b)).

  Producción Si el arco va de un vértice que representa un tipo de recurso a un proceso, indica que éste es productor del recurso (figura 5.1(c)).

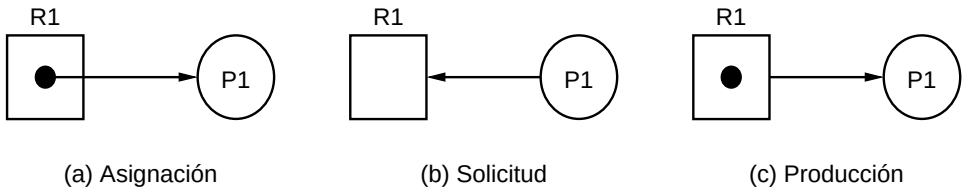

Figura 5.1: Notación para el modelado de interbloqueos

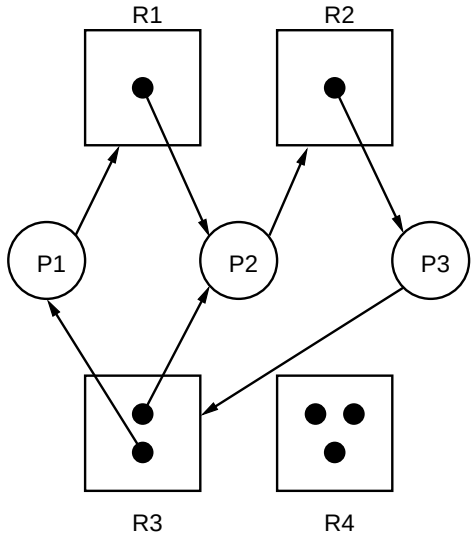

Figura 5.2: Grafo de asignación de recursos con interbloqueo

Este grafo de asignación de recursos nos permite determinar de una forma fácil si en un sistema existe un interbloqueo o no. Así, si el grafo no tiene un ciclo, entonces el sistema no se encuentra en un estado de interbloqueo. Por otro lado, si está interbloqueado el grafo presenta al menos un ciclo, donde se encuentran implicados los procesos y recursos que lo forman, como se puede ver en la figura 5.2. Sin embargo, la presencia de un ciclo no implica la existencia de un interbloqueo. Así, la figura 5.3 presenta un ciclo, pero los procesos P1 y P4 pueden finalizar y liberar sus recursos, de forma que P2 y P3 pueden continuar con su trabajo.

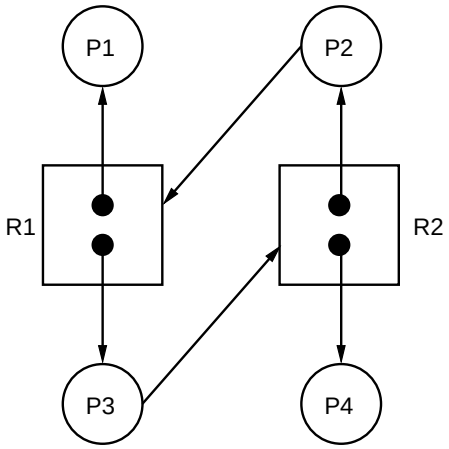

Figura 5.3: Grafo de asignación de recursos con ciclo pero sin interbloqueo

5.4. Estrategias para tratar los interbloqueos

Dado que las peticiones y liberaciones de recursos se realizan en un orden impredecible, es muy difícil diseñar una política infalible con un coste aceptable, al contrario que con la exclusión mutua.

Los sistemas operativos pueden afrontar los interbloqueos siguiendo una de las tres estrategias siguientes:

- Utilizar un protocolo que asegure que no van a aparecer interbloqueos en el sistema. Esto se puede conseguir de dos formas diferentes, mediante lo que se conoce como **prevención** de interbloqueos, que intenta evitar que éstos aparezcan eliminando una de las cuatro condiciones necesarias; o bien, mediante los métodos de **predicción**[2], éstos hacen un reparto cuidadoso de los recursos, intentando predecir antes de asignar uno, si esta asignación puede conducir a una situación de interbloqueo.

- Dejar que aparezcan interbloqueos y utilizar métodos para detectarlos y, si existen, hacer que desaparezcan. Éstos son los denominados métodos de **detección** y **recuperación**.

- Otros sistemas no se preocupan del problema en absoluto. Si el interbloqueo aparece en el sistema habrá que hacerlo desaparecer manualmente. Esta estrategia es empleada por muchos sistemas operativos,

[2]La expresión inglesa *avoidance* que en la mayor parte de la bibliografía en español sobre interbloqueos se traduce como evitación, se ha traducido como predicción, ya que prevenir y evitar se pueden considerar sinónimos en español.

incluyendo a UNIX y LINUX. Es aceptable en el caso de que los interbloqueos ocurran con muy poca frecuencia.

5.4.1. Prevención

Esta técnica fue propuesta por Havender en 1968. Consiste en diseñar un sistema de tal forma que se elimine toda posibilidad de que ocurra un interbloqueo. Para ello, hemos de asegurarnos de que una de las cuatro condiciones (véase el apartado 5.2) no se dé en el sistema. En la práctica sólo se proporcionan métodos para eliminar las tres últimas, dado que un sistema operativo debe proporcionar exclusión mutua para el acceso a los recursos críticos.

Los métodos para prevenir el interbloqueo pueden ser de dos tipos. Los indirectos que consisten en impedir la aparición de la condición de retener y esperar o la de no apropiación, y los directos que evitan la aparición de la espera circular.

5.4.1.1. Negación de la condición de retener y esperar

Para asegurarnos de que en un sistema nunca se cumple la condición de retener y esperar, tenemos que garantizar que siempre que un proceso solicita un recurso no retenga otros. Esto se puede conseguir mediante distintas estrategias.

Todo o nada

Consiste en forzar a los procesos a pedir todos los recursos que van a necesitar para su terminación al principio, y bloquearlos mientras esto no pueda ser garantizado. De esa forma, si está disponible el conjunto de recursos que necesita un proceso, entonces el sistema puede asignárselos y éste puede seguir su ejecución. Si algún recurso no está disponible, el proceso deberá esperar. Mientras el proceso espera no puede tener ningún recurso. Con esto se evita la condición de retener y esperar, y por tanto, no puede aparecer el interbloqueo.

La forma de implementarla es hacer que las peticiones de recursos se hagan al comienzo, independientemente de si los va a utilizar inmediatamente o no. Para ello se debe disponer de una llamada al sistema que permita solicitar un conjunto de recursos simultáneamente; asimismo, la liberación de éstos se hará en bloque al finalizar la ejecución del proceso. Esta solución es ineficiente debido a dos causas:

- Baja utilización de los recursos debida a que éstos pueden estar asignados, pero inactivos durante un largo período de tiempo, durante el cuál se niega el acceso a otros procesos. Por ejemplo, supongamos un proceso que tras leer unos datos de un fichero, realiza cálculos y por último, al final de su ejecución imprime los resultados. Mientras realiza las operaciones de cálculo tiene asignados el fichero y la impresora sin que ningún otro proceso pueda usarlos, puesto que no puede liberar ningún recurso hasta que no termine.

- Posibilidad de inanición de un proceso que requiera recursos muy utilizados, ya que puede esperar mucho tiempo para conseguirlos, aunque podría empezar a ejecutarse sólo con algunos de ellos. En el ejemplo anterior, el proceso podría haber empezado su ejecución sólo con el fichero, sin necesidad de esperar la impresora.

División de peticiones

Para disminuir los inconvenientes anteriores se puede dividir el proceso en fases que se ejecuten de manera relativamente independiente, y aplicar la estrategia anterior a cada una de ellas. El proceso pide todos los recursos que va a necesitar durante la ejecución de cada fase. Cuando termina una, libera todos los recursos asignados y solicita los de la siguiente. De esta forma, se reduce el desperdicio de recursos, pero hace más complicado el trabajo del programador, al requerir un trabajo extra en el diseño de las aplicaciones. En el caso descrito en el apartado anterior se podían definir tres fases, una de lectura de datos en la que sólo necesitará el fichero, otra de cálculo en la que no necesita ningún recurso y, por último, una de impresión de resultados en la que utilizará la impresora.

Petición incremental de recursos y liberación de éstos

El proceso va pidiendo los recursos a medida que los necesita; si un recurso no se le puede conceder porque no está disponible, liberará voluntariamente todos los que tenía asignados hasta el momento. El proceso se ocupa de dejar los recursos en el estado adecuado.

El problema que se presenta con esta estrategia es que algunos recursos no pueden ser liberados y readquiridos posteriormente de una forma fácil. Por ejemplo, si se han realizado cambios irreversibles en la memoria o en ficheros, el simple hecho de devolver el recurso puede corromper el sistema.

Por tanto, la liberación de un recurso sólo tiene sentido si la integridad del sistema no se ve afectada, y cuando el gasto debido a las operaciones de guardar y restaurar los estados de los procesos y los recursos sea aceptablemente pequeño.

5.4.1.2. Negación de la condición de no apropiación

La condición de no apropiación puede ser negada permitiendo la apropiación, es decir, autorizando al sistema a retirar el control de ciertos recursos a un proceso bloqueado. De este modo, si un proceso solicita un recurso que no está disponible porque está asignado a otro proceso, entonces el sistema puede apropiarlo si el proceso al que está asignado se encuentra bloqueado. El empleo de este tipo de estrategia podría seguir el algoritmo 5.1 cada vez que un proceso solicita un recurso.

Algoritmo 5.1 Negación de la no apropiación

```
si está disponible
entonces
  se asigna el recurso al proceso
si no si asignado a procesos en espera de recursos
entonces
  el sistema se apropia del recurso
  el sistema asigna el recurso al proceso solicitante
si no
  el proceso pasa a estado de espera del recurso
fin si
```

Esta apropiación es involuntaria desde el punto de vista del proceso afectado, por lo que el sistema operativo debe encargarse de guardar el estado del recurso apropiado y añadir éste a la lista de los solicitados por el proceso. Éste volverá a ejecutarse solamente cuando pueda obtener de nuevo sus recursos anteriores, así como los nuevos que está solicitando.

La apropiación sólo es posible para ciertos tipos de recursos, que permitan guardar de forma fácil su estado, un ejemplo es la memoria principal cuyo contenido puede ser almacenado en memoria secundaria. Sin embargo, otro tipo de recursos, tales como ficheros parcialmente actualizados, no pueden ser apropiados sin corromper el sistema. Esto hace que la apropiación de recursos sea aún más difícil que la liberación voluntaria de la que se habló antes.

5.4.1.3. Negación de la condición de espera circular

Una forma de evitar la espera circular es mediante la siguiente estrategia:

1. Definir una ordenación de los tipos de recursos. Para ello, asignamos a cada tipo de recurso un número entero único, que nos permite compararlos y determinar si uno precede al otro en nuestro orden. Más

formalmente, sea $R = \{r_1, \ldots, r_n\}$ el conjunto de tipos de recursos, en los que establecemos el siguiente orden: $r_1 < r_2 < \ldots < r_n$.

2. Exigir que los procesos pidan los recursos en orden creciente de enumeración. Esto es posible mediante dos estrategias:

   - Si un proceso solicita un recurso de tipo r_i, después sólo podrá solicitar recursos de tipo r_j, si y sólo si $r_j > r_i$. Cuando se necesiten varios ejemplares del mismo tipo de recurso, sólo se debe hacer una petición para todos.
   - Siempre que un proceso solicite un recurso de tipo r_i debe liberar todos los recursos r_k que tengan un índice mayor, es decir, $r_k > r_i$.

Esta estrategia presenta ciertas dificultades a la hora de implementarla:

- La ordenación de los tipos de recursos debe ser lógica para evitar esperas innecesarias, de tal modo, que siga el orden en el que se usan normalmente los recursos. Los procesos que los necesiten en orden diferente al previsto, deberán adquirirlos y conservarlos durante bastante tiempo antes de utilizarlos, lo cuál implica un bajo rendimiento o utilización de los recursos. Por tanto, hemos de buscar la ordenación óptima de éstos.
- Si se agregan nuevos tipos de recursos al sistema hemos de establecer una nueva ordenación. Además, en algunos casos, puede ser necesario modificar las diferentes aplicaciones para adaptarse al nuevo orden.
- La ordenación afecta a la capacidad del programador para escribir libre y fácilmente el código de sus aplicaciones, pues ha de solicitar los distintos recursos en función de ésta.

5.4.2. Predicción

Los métodos de prevención se basan en negar alguna de las condiciones necesarias para que se produzcan interbloqueos, restringiendo de cierta manera las peticiones de recursos. Esto conduce a un uso ineficiente de éstos.

Otro enfoque para resolver el problema de los interbloqueos es la predicción. Este método permite que se den las tres primeras condiciones, pero cuando un proceso pide un recurso el sistema estudia si su asignación puede producir un interbloqueo o no.

Para aplicar una estrategia de este tipo, se necesita disponer de información sobre las peticiones de recursos que van a realizar todos los procesos del sistema.

5.4.2.1. El algoritmo del banquero

Existen diversos algoritmos de predicción caracterizados por la mayor o menor cantidad de información previa necesaria. El más conocido es el **algoritmo del banquero**, propuesto por Dijkstra en 1965, llamado así porque introduce a un banquero que hace préstamos y recibe pagos de una fuente dada de capital. Éste necesita conocer el número de recursos disponibles y asignados, así como la demanda máxima de los procesos. Antes de ver este algoritmo, empezaremos definiendo una serie de conceptos relacionados con él.

Consideremos un sistema de n procesos y m tipos de recursos diferentes. Definiremos los siguientes vectores y matrices:

$$Recursos = [R_i] = (R_1, R_2, \ldots, R_m)$$
$$Disponible = [D_i] = (D_1, D_2, \ldots, D_m)$$

$$Demanda = [Dm_{ij}] = \begin{pmatrix} Dm_{11} & Dm_{12} & \ldots & Dm_{1m} \\ Dm_{21} & Dm_{22} & \ldots & Dm_{2m} \\ \ldots & \ldots & \ldots & \ldots \\ Dm_{n1} & Dm_{n2} & \ldots & Dm_{nm} \end{pmatrix}$$

$$Asignación = [A_{ij}] = \begin{pmatrix} A_{11} & A_{12} & \ldots & A_{1m} \\ A_{21} & A_{22} & \ldots & A_{2m} \\ \ldots & \ldots & \ldots & \ldots \\ A_{n1} & A_{n2} & \ldots & A_{nm} \end{pmatrix}$$

El vector *Recursos* señala la cantidad total de cada tipo de recurso que hay en el sistema. El vector *Disponible* da la cantidad de cada recurso no asignada actualmente. La matriz *Demanda* indica las necesidades máximas de cada proceso para cada recurso. Es decir, Dm_{ij} da la demanda del proceso p_i para el recurso r_j. Esta información debe ser declarada por el proceso de antemano para que se pueda utilizar el algoritmo del banquero. Del mismo modo, A_{ij} es el número de unidades del recurso r_j asignadas al proceso p_i. Con estas estructuras se pueden establecer las siguientes relaciones:

1. Todos los recursos o están asignados o están disponibles.

$$\forall i, \sum_{k=1}^{n} A_{ki} + D_i = R_i$$

2. Un proceso no puede solicitar más unidades de cada tipo de recurso de las disponibles en el sistema.

$$\forall k, i, Dm_{ki} \leq R_i$$

3. A los procesos no se les puede asignar mayor cantidad de recursos que los declarados inicialmente.

$$\forall k, i, A_{ki} \leq Dm_{ki}$$

Las estructuras definidas anteriormente nos permiten definir el **estado del sistema**, donde se refleja qué recursos tiene asignados cada proceso. A partir de éste podemos obtener la matriz $Necesidad$ que representará el número de instancias de cada recurso que podría necesitar un proceso. Más formalmente:

$$\forall i, j, N_{ij} = Dm_{ij} - A_{ij}$$

Se dice que un sistema está en **estado seguro**, si existe un orden en el cuál todos los procesos pueden finalizar su ejecución, sin que se produzca un interbloqueo. Si este orden no existe nos encontramos en un **estado inseguro**.

Formalmente, un sistema se encuentra en estado seguro si existe una secuencia de procesos $< p_1, \ldots, p_n >$, donde cada proceso p_i puede finalizar con los recursos que tiene asignados más los disponibles más los recursos asignados a todos los procesos p_j, para todo $j < i$.

Un estado seguro es un estado sin interbloqueo. Del mismo modo, un estado de abrazo mortal es un estado inseguro. Sin embargo, un estado inseguro no implica necesariamente la existencia del interbloqueo, simplemente indica que una secuencia desafortunada de eventos puede ocasionar un interbloqueo. En la figura 5.4 podemos ver la relación existente entre estos conceptos.

Esto nos sugiere una estrategia de predicción de interbloqueo, que consiste en asegurar que el sistema esté siempre en un estado seguro. El algoritmo del banquero es una estrategia de este tipo.

El estado inicial en el que no se ha asignado ningún recurso y todos están disponibles, es siempre seguro. Partiendo de éste, cuando un proceso realiza una petición, se simula su concesión, se actualiza el estado del sistema, y se determina si nos encontramos en un estado seguro. Si es así, se concede la solicitud y, si no, se bloquea al proceso hasta que sea seguro.

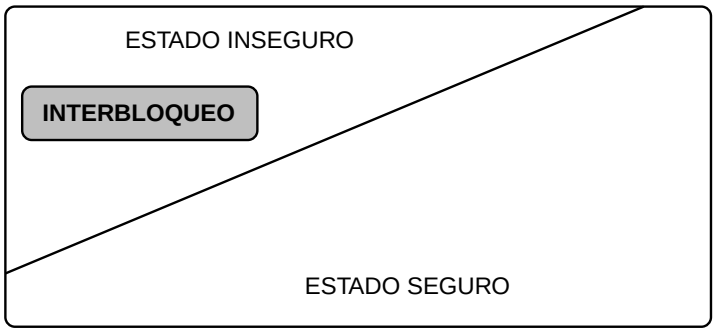

Figura 5.4: Relaciones entre estado seguro, inseguro e interbloqueo

Algoritmo 5.2 Algoritmo del banquero

```
Sea P_i, el proceso que hace una petición
si ∃ un recurso R_k, tal que Solicitud[i][k] >Necesidad[i][k]
entonces
  Error, el proceso P_i supera su demanda máxima inicial
fin si
si ∃ un recurso R_k, tal que Solicitud[i][k] >Disponible[i][k]
entonces
  bloquear al proceso P_i porque no hay recursos disponibles
fin si
para todo recurso, R_j
hacer
  Disponible[j] = Disponible[j] - Solicitud[i][j]
  Asignación[i][j] = Asignación[i][j] + Solicitud[i][j]
  Necesidad[i][j] = Necesidad[i][j] - Solicitud[i][j]
fin para
ejecutar algoritmo de seguridad
si Sistema == SEGURO
entonces
  asignar los recursos al proceso P_i
si no
  para todo recurso, R_j
  hacer
    Disponible[j] = Disponible[j] + Solicitud[i][j]
    Asignación[i][j] = Asignación[i][j] - Solicitud[i][j]
    Necesidad[i][j] = Necesidad[i][j] + Solicitud[i][j]
  fin para
  bloquear al proceso P_i
fin si
```

Para implementar este algoritmo es necesario disponer de cierta información, que vendrá dada por las estructuras anteriores más la matriz *Solicitud*. Ésta contiene las solicitudes de recursos que realizan los distintos procesos. Así, $Solicitud_{ij}$ indica la solicitud del proceso p_i del recurso r_j. El algorit-

mo 5.2 muestra una posible implementación. Siempre que un proceso pide recursos, se comprueba, en primer lugar, si esta petición está dentro de las necesidades máximas declaradas inicialmente, si no fuera así se produciría un error. Posteriormente se determina si existen recursos suficientes para atenderla, en caso afirmativo se simula la asignación comprobándose si el sistema queda en estado seguro. Si no había recursos o el sistema llega a estado inseguro se bloquea al proceso.

Para determinar si el sistema se encuentra o no en un estado seguro podemos utilizar las estructuras vistas anteriormente. El algoritmo 5.3 muestra el mecanismo que podría emplearse para esta comprobación.

Algoritmo 5.3 Algoritmo de seguridad

```
para todo recurso, R_j
hacer
  Trabajo[j] = Disponible[j];
fin para
para todo proceso, P_i
hacer
  Acabar[i] = Falso;
fin para
mientras ∃ un proceso P_i, tal que Acabar[i] == Falso y
        ∀ recurso R_j, Necesidad[i][j] ≤ Trabajo[j]
hacer
  Acabar[i] = Verdadero;
  para todo recurso, R_j
  hacer
    Trabajo[j] = Trabajo[j] + Asignación[i][j];
  fin para
fin mientras
Sistema = SEGURO;
para todo proceso, P_i
hacer
  si Acabar[i] == Falso
  entonces
    Sistema = INSEGURO;
  fin si
fin para
```

El algoritmo del banquero garantiza que se terminarán los procesos avanzando de estado seguro en estado seguro para evitar el interbloqueo, permitiendo las condiciones de exclusión mutua, retener y esperar, y no apropiación.

Ejemplo de utilización del algoritmo del banquero

Para comprender mejor el algoritmo, vamos a ver cómo se aplica. Consideremos la situación actual del sistema dada por las siguientes estructuras:

	Asignación		
	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	2
P4	2	1	1
P5	0	0	2

	Demanda		
	R1	R2	R3
P1	7	5	3
P2	3	2	2
P3	9	0	2
P4	2	2	2
P5	4	3	2

Disponible	
R1	3
R2	3
R3	2

En primer lugar se calculará la matriz Necesidad, que viene dada por la diferencia entre las matrices de Demanda y Asignación. Ésta es la que se muestra a continuación:

	Necesidad		
	R1	R2	R3
P1	7	4	3
P2	1	2	2
P3	6	0	0
P4	0	1	1
P5	4	3	0

Antes de aplicar el algoritmo del banquero, lo primero que debemos tener en cuenta es si el sistema se encuentra en un estado seguro. Para ello, creamos las estructuras de datos necesarias para aplicar el algoritmo.

	Asignación		
	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	2
P4	2	1	1
P5	0	0	2

	Necesidad		
	R1	R2	R3
P1	7	4	3
P2	1	2	2
P3	6	0	0
P4	0	1	1
P5	4	3	0

Trabajo	
R1	3
R2	3
R3	2

Acabar	
P1	F
P2	F
P3	F
P4	F
P5	F

Podemos ver que $P2$ no ha terminado y existen recursos disponibles (vector Trabajo) para cubrir sus necesidades, por tanto este proceso podrá finalizar sin problemas. De esta forma, las estructuras resultantes son (se destaca en negrita las variaciones respecto a la situación anterior):

	Asignación		
	R1	R2	R3
P1	0	1	0
P2	**0**	**0**	**0**
P3	3	0	2
P4	2	1	1
P5	0	0	2

	Necesidad		
	R1	R2	R3
P1	7	4	3
P2	**0**	**0**	**0**
P3	6	0	0
P4	0	1	1
P5	4	3	0

Trabajo	
R1	5
R2	**3**
R3	2

Acabar	
P1	F
P2	**V**
P3	F
P4	F
P5	F

Tras $P2$, la secuencia de procesos $P4$, $P1$, $P3$ y $P5$, pueden terminar sin problemas, por lo que podemos afirmar que el sistema se encuentra en estado seguro. Supongamos que el proceso $P2$, solicita una unidad del recurso $R1$ y otra del recurso $R3$. Veamos si se le puede conceder inmediatamente. Como la solicitud de $P2$ es menor que su demanda máxima inicial y existen recursos disponibles para atenderla, simulamos la asignación, obteniendo las siguientes estructuras:

	Asignación		
	R1	R2	R3
P1	0	1	0
P2	**3**	**0**	**1**
P3	3	0	2
P4	2	1	1
P5	0	0	2

	Necesidad		
	R1	R2	R3
P1	7	4	3
P2	**0**	**2**	**1**
P3	6	0	0
P4	0	1	1
P5	4	3	0

Disponible	
R1	**2**
R2	3
R3	**1**

Falta comprobar si el sistema se encuentra en estado seguro. Basta con observar, que la secuencia de procesos $P2$, $P4$, $P1$, $P3$ y $P5$, pueden finalizar correctamente, por lo que le podemos asignar los recursos solicitados al proceso $P2$. Si a continuación el proceso $P1$ solicita dos unidades del recurso $R1$ y tres del recurso $R2$, no se le podrán asignar ya que conduciría a un estado inseguro. Para ello, aplicamos el algoritmo del banquero, que tras la asignación simulada posee las siguientes estructuras:

	Asignación		
	R1	R2	R3
P1	**2**	**4**	**0**
P2	3	0	1
P3	3	0	2
P4	2	1	1
P5	0	0	2

	Necesidad		
	R1	R2	R3
P1	**5**	**1**	**3**
P2	0	2	1
P3	6	0	0
P4	0	1	1
P5	4	3	0

Disponible	
R1	**0**
R2	**0**
R3	1

Como puede verse los procesos no pueden terminar su ejecución porque no se pueden atender sus solicitudes de recursos al no haber suficientes recursos disponibles, por tanto se llega a un estado inseguro, por lo que no se asignan los recursos solicitados.

Inconvenientes del algoritmo del banquero

El algoritmo del banquero ha sido utilizado para predecir interbloqueos en sistemas con pocos recursos, y es una política menos restrictiva que la prevención. Sin embargo, es poco práctico para la mayoría de los sistemas por varias razones:

- Los procesos deben declarar por anticipado el número máximo de re-

cursos que van a necesitar. Esto es factible en un sistema por lotes, pero en un sistema interactivo es muy difícil de conocer.

- El número de recursos de cada clase debe permanecer constante. Esto es difícil de conseguir ya que, bien por averías, bien por mantenimiento preventivo, no podemos contar con un número de recursos siempre constante.
- Los procesos deben liberar explícitamente los recursos antes de terminar.
- El algoritmo tiene un alto coste en tiempo cuando el número de procesos y de peticiones es muy elevado, ya que se necesitan $m \times n^2$ operaciones, donde n es el número de procesos y m el de recursos.
- El algoritmo garantiza que todas las peticiones serán concedidas dentro de un intervalo de tiempo finito, pero no especificado. Está claro que, en sistemas reales, se necesitan garantías mucho mayores que ésta.
- De forma similar, el algoritmo supone que los procesos devolverán los recursos dentro de un intervalo de tiempo finito. También en este caso se necesitan garantías mucho mayores que ésta para sistemas reales.
- El algoritmo asume el peor de los casos y, por lo tanto, realiza una pobre utilización de recursos manteniendo sin asignar recursos vitales, con tal de permanecer en estado seguro.
- Castiga a los procesos que requieren muchos recursos, de tal modo que los que necesiten pocos pueden provocar la inanición de los primeros.
- Por otro lado, los procesos a considerar deben ser independientes; esto es, el orden en que se ejecuten no debe estar forzado por condiciones de sincronización. Por este motivo, la ejecución estrictamente secuencial de los procesos, es la peor condición a imponer.

5.4.3. Detección y recuperación

Las estrategias de prevención y predicción son muy poco flexibles a la hora de asignar los recursos porque imponen muchas restricciones, dando lugar a una baja utilización de éstos. Sin embargo, las estrategias de detección no imponen restricciones al acceso a los recursos por parte de los procesos, permitiéndose la aparición de los interbloqueos. En estos sistemas es necesario disponer de un algoritmo que compruebe la existencia de interbloqueo, y otro que nos permita hacerlo desaparecer en caso de que exista.

La comprobación de la existencia del interbloqueo implica una cierta sobrecarga en el sistema. Por un lado, requiere un gasto extra en tiempo de

procesamiento para ejecutar el algoritmo y, por otro, necesita un coste para mantener la información sobre la situación actual del sistema.

Antes de estudiar las distintas estrategias de detección y recuperación, el diseñador del sistema debe plantearse la siguiente cuestión, ¿cuándo debe realizarse la comprobación o control de la existencia del interbloqueo? Existen varias posibilidades:

- Cada vez que se deniega una solicitud. Esto permite detectar rápidamente la existencia de interbloqueo pero consume mucho tiempo de CPU.
- A intervalos regulares. En este caso habría que ajustar el tamaño del intervalo, teniendo en cuenta que si es demasiado pequeño también se consumirá mucho tiempo de CPU, y si es muy grande, se pueden haber producido muchos interbloqueos, siendo más difícil la recuperación.
- Cuando desciende el rendimiento del sistema. Se evita ejecutar innecesariamente el algoritmo, pero hay que estar controlando continuamente el rendimiento del sistema.

Una vez detectado un interbloqueo en un sistema, se debe seguir alguna estrategia para su recuperación. Ésta implica la pérdida de parte o de la totalidad del trabajo realizado por algunos procesos, pero este es el coste de este método.

5.4.3.1. Detección por grafos

En el apartado 5.3 vimos cómo se pueden representar gráficamente los procesos y recursos del sistema. Esta representación nos sirve para determinar si existe o no un interbloqueo. Para ello se aplica una técnica de **reducción de grafos**. Un grafo puede ser reducido por el proceso P_i, si:

1. No espera ningún recurso.
2. Está esperando un recurso del que existen unidades disponibles.

Cuando un grafo es reducido por el proceso P_i libera todas las unidades de los recursos asignados. Se dice que un grafo es **irreducible** si no puede ser reducido por algún proceso. Un grafo es **completamente reducible** si existe una serie de transformaciones que reducen el grafo a una situación donde no existen aristas de solicitudes ni de asignaciones.

En la figura 5.5(a) tenemos un ejemplo donde existen tres recursos y cuatro procesos. En ella, podemos observar que P1 tiene dos recursos de

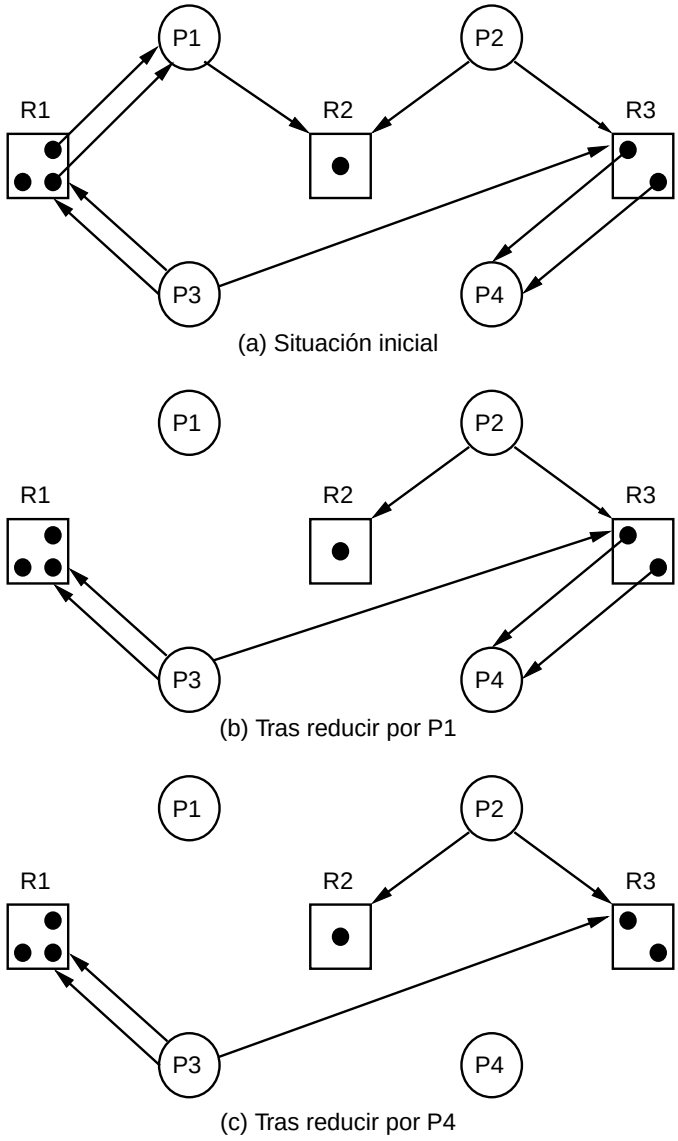

Figura 5.5: Ejemplo de reducción de un grafo

tipo R1 asignados, y existe un recurso de tipo R2 disponible para atender la única solicitud que está haciendo. Por tanto, es posible reducir el grafo por este proceso. La nueva situación aparece en la figura 5.5(b). En ella, P4 tiene todos sus recursos asignados, por lo que se puede reducir el grafo. La figura 5.5(c) muestra el grafo tras reducir por P4. Finalmente, aunque no se muestra, el grafo se puede reducir por P2 y P3, al existir recursos para ambos.

5.4.3.2. Algoritmo de detección

El método gráfico descrito anteriormente es difícil de implementar en un sistema operativo. A continuación vamos a ver un algoritmo más simple, que consiste en investigar toda posible secuencia de asignación para los procesos que aún tengan que completarse.

Algoritmo 5.4 Algoritmo de detección

```
para todo proceso, P_i
hacer
  Bloqueado[i] = Verdadero
fin para
para todo recurso, R_j
hacer
  Trabajo[j] = Disponible[j]
fin para
mientras ∃ un proceso P_i, tal que Bloqueado[i] == Verdadero y
        ∀ recurso, R_j, Solicitud[i][j] ≤ Trabajo[j]
hacer
  para todo recurso, R_j
  hacer
    Bloqueado[i] = Falso
    Trabajo[j] = Trabajo[j] + Asignación[i][j]
  fin para
fin mientras
Interbloqueo = Falso
para todo proceso, P_i
hacer
  si Bloqueado[i] == Verdadero
  entonces
    Interbloqueo = Verdadero
  fin si
fin para
```

El algoritmo 5.4 muestra una posible implementación. Como se puede apreciar es muy parecido al algoritmo 5.3 empleado para determinar si un sistema está en estado seguro, empleando las mismas estructuras.

Ejemplo de detección del interbloqueo

Supongamos el sistema representado por las siguientes estructuras, donde ningún recurso es consumible:

	Asignación		
	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	3
P4	2	1	1
P5	0	0	2

	Solicitud		
	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	0
P4	1	0	0
P5	0	0	2

Disponible	
R1	0
R2	0
R3	0

Comprobemos si el sistema presenta un interbloqueo. Para ello creamos las estructuras siguientes:

	Asignación		
	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	3
P4	2	1	1
P5	0	0	2

	Solicitud		
	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	0
P4	1	0	0
P5	0	0	2

Trabajo	
R1	0
R2	0
R3	0

Bloqueado	
P1	V
P2	V
P3	V
P4	V
P5	V

Escogemos $P1$, pues no está esperando ningún recurso y puede finalizar su tarea:

	Asignación		
	R1	R2	R3
P1	**0**	**0**	**0**
P2	2	0	0
P3	3	0	3
P4	2	1	1
P5	0	0	2

	Solicitud		
	R1	R2	R3
P1	**0**	**0**	**0**
P2	2	0	2
P3	0	0	0
P4	1	0	0
P5	0	0	2

Trabajo	
R1	0
R2	**1**
R3	0

Bloqueado	
P1	**F**
P2	V
P3	V
P4	V
P5	V

A continuación, la secuencia $P3$, $P4$, $P2$ y $P5$ pueden finalizar, por lo que el sistema no presenta un interbloqueo, debido a que todos los procesos pueden atender sus solicitudes. Sin embargo, si la situación inicial fuera la siguiente ($P3$ solicita un recurso más de tipo $R3$):

	Asignación		
	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	3
P4	2	1	1
P5	0	0	2

	Solicitud		
	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	1
P4	1	0	0
P5	0	0	2

Trabajo	
R1	0
R2	0
R3	0

Bloqueado	
P1	V
P2	V
P3	V
P4	V
P5	V

En esta nueva situación sólo $P1$ podría finalizar, pero el resto de procesos no podrían continuar al encontrarse en interbloqueo.

5.4.3.3. Recuperación de un interbloqueo

Una vez detectado el interbloqueo tenemos dos opciones para hacerlo desaparecer. Una consiste en terminar uno o más procesos para romper la espera circular, la otra en apropiarse de recursos de uno o más procesos implicados en el interbloqueo.

Terminación de procesos

Con esta estrategia eliminamos procesos, recuperándose los recursos que tenían asignados con objeto de que el sistema pueda continuar con normalidad. Para ello tenemos varias opciones:

1. Eliminar todos los procesos interbloqueados. Es una de las soluciones más adoptadas en los sistemas operativos, pero presenta un coste muy elevado.
2. Ir abortando sucesivamente todos los procesos interbloqueados hasta que desaparezca el abrazo mortal, es decir, hasta que haya suficientes recursos libres como para continuar sin interbloqueo la ejecución de los restantes procesos. Es necesario establecer un orden a la hora de seleccionar los procesos, normalmente se seguirá un criterio de coste mínimo. Tras la eliminación de cada uno habrá que ejecutar un algoritmo de detección para determinar si aún existe el interbloqueo.

En la segunda estrategia la decisión de eliminar un proceso no es fácil y pueden seguirse diversos criterios con objeto de que la pérdida que sufra el sistema sea mínima. Algunos de éstos pueden ser:

- Cuántos procesos se verán implicados en la reanudación.
- Cantidad y calidad de los recursos utilizados (qué tipo, si son fáciles de apropiar, etc.).

- La prioridad del proceso.
- Tiempo de ejecución transcurrido.
- Cuántos recursos más necesita para finalizar.

Apropiación de recursos

Consiste en retirar o apropiarse de los recursos de uno de los procesos interbloqueados para dárselos a otro de los implicados, permitiéndole que termine y salga del interbloqueo. En este caso hay que afrontar las siguientes cuestiones:

Selección de la víctima Determinar qué procesos y qué recursos son apropiados, siempre siguiendo el criterio del mínimo coste, al igual que cuando se abortan procesos. También se ha de evitar la inanición, para que no siempre se apropie al mismo proceso.

Apropiación del recurso Hemos de tener en cuenta que el proceso al que le quitemos el recurso, en un futuro tendrá que continuar y habrá que asignarle de nuevo dicho recurso como si no hubiera pasado nada. Es por este motivo por lo que sólo se pueden apropiar ciertos recursos, al igual que en la prevención cuando se niega la condición de no apropiación (ver apartado 5.4.1.2).

5.5. Resumen

La utilización de recursos críticos por parte de los procesos puede ocasionar la aparición de una situación conocida como interbloqueo. Ésta se puede definir como el bloqueo permanente de un conjunto de procesos que compiten por los recursos del sistema o que se comunican entre ellos.

Las técnicas clásicas para abordar este problema se clasifican en tres grandes grupos: prevención, predicción y detección. La elección de una u otra dependerá del sistema y de la información que poseamos sobre él. No existe una solución óptima al problema, ya que todas ellas implican o una pobre utilización de los recursos o bien, una sobrecarga en el sistema. De este modo, en muchos sistemas donde se supone que su probabilidad de aparición es pequeña no se aplica ninguna estrategia, y en caso de aparecer se opta por reiniciar el sistema.

Los métodos de prevención se caracterizan por negar una de las condiciones necesarias para que se produzca el interbloqueo. Son muy restrictivos en

la asignación de recursos a los procesos, proporcionando un uso ineficiente de éstos.

Las estrategias de predicción se basan en el conocimiento previo de cierta información, con objeto de determinar si existe una secuencia de eventos que pueda conducir o no a un interbloqueo. De esta forma, cuando un proceso solicita un recurso, se determina si su asignación puede conducir o no a un interbloqueo. En el primer caso se le deniega el acceso al recurso. El algoritmo de predicción más conocido es el del banquero.

Por el contrario, las estrategias de detección son más liberales en el sentido de no imponer ninguna restricción ni tener conocimiento alguno de los procesos. Estos algoritmos permiten la aparición del interbloqueo en el sistema, proporcionan un método para su detección, y si lo detectan, intentan hacer que desaparezca con el menor coste posible.

5.6. Ejercicios

1. ¿Qué diferencias existen entre las estrategias de prevención, predicción y detección?

2. ¿Cuál es la diferencia principal entre el interbloqueo y la inanición? ¿En qué se parecen?

3. ¿Es posible que aparezca un interbloqueo en el que esté implicado un único proceso? ¿Y con una única clase o tipo de recurso?

4. Diga si las siguientes afirmaciones son verdaderas o falsas, razonando su respuesta:

   a) Un sistema que en un momento dado presenta espera circular está en interbloqueo.

   b) El algoritmo del banquero está limitado porque el número de recursos ha de ser fijo.

   c) La exclusión mutua es una condición que no se puede eliminar.

   d) Se puede prevenir el interbloqueo mediante la asignación de prioridades, de tal modo que un recurso se asignará al proceso de mayor prioridad.

5. En un grafo de asignación de recursos, ¿es posible la existencia de un ciclo sin que exista un interbloqueo? Razone la respuesta. En caso afirmativo explique por qué y dibuje una gráfica de ejemplo.

6. Las estrategias de predicción necesitan cierta información tal como la demanda máxima de recursos que van a necesitar los procesos. Pero,

¿cómo podríamos determinar esta demanda máxima en un sistema por lotes y en un sistema interactivo?

7. Considere la siguiente instantánea de un sistema que emplea un método de predicción basado en el algoritmo del banquero:

	Asignación			
	R1	R2	R3	R4
P1	0	0	1	2
P2	1	0	0	0
P3	2	3	5	4
P4	0	6	3	1
P5	0	0	1	4

	Demanda			
	R1	R2	R3	R4
P1	0	0	1	2
P2	1	7	5	0
P3	3	3	5	6
P4	0	6	5	1
P5	10	6	5	6

Disponible	
R1	1
R2	5
R3	2
R4	0

Conteste de forma razonada a las siguientes preguntas:

a) ¿Cuál es el contenido de la matriz *Necesidad*?

b) ¿Está el sistema en estado seguro?

c) Si llega la solicitud del proceso P2 = (0, 4, 2, 0), ¿puede ser satisfecha sin problema?

d) Si a continuación llega la solicitud P5 = (1, 0, 0, 0), ¿podría ser satisfecha sin problema?

e) ¿Y si a continuación llega la solicitud P4 = (1, 0, 0, 0)?

8. Suponga un sistema que aplica una estrategia de detección y recuperación de interbloqueos. La situación inicial viene representada por las siguientes estructuras:

	Asignación			
	R1	R2	R3	R4
P1	0	0	1	2
P2	1	0	0	0
P3	1	2	1	1
P4	0	1	2	2
P5	0	0	1	3

	Solicitud			
	R1	R2	R3	R4
P1	0	0	1	2
P2	1	2	1	0
P3	2	1	2	0
P4	0	3	1	2
P5	0	3	2	3

Disponible	
R1	1
R2	2
R3	2
R4	0

Se pide:

a) Modele mediante un grafo de asignación de recursos la situación actual del sistema.

b) ¿Existe interbloqueo? En caso afirmativo indique los procesos implicados.