PROGRAMACION CONCURRENTE

I.4. Patologías en programas concurrentes

J.M. Drake

Notas:

Patologías de los programas concurrentes

- Patologías características de los programa concurrente son:
  - **Propiedades de seguridad**: No debe ejecutar algo que conduzca a un error:
    - Ejecuta una operación imposible
    - Entra en una sección crítica cuando está otro proceso.
    - Un proceso no respeta un punto de sincronismo.
    - Un proceso permanece a la espera de un evento que nunca se producirá.
  - **Propiedades de vivacidad**: Las sentencias que se ejecuten deben contribuir a un avance constructivo al objetivo del programa:
    - Bloqueos activos: Dos procesos ejecutan sentencias que no hacen avanzar al programa.
    - Aplazamiento indefinido: Un programa se queda sin tiempo de procesador para avanzar.
- Las mejoras en seguridad y vivacidad suelen tener efectos contrapuestos:
  - La práctica de la ingeniería software pone énfasis en el diseño de la seguridad. Se asegura que el código no hace nada imprevisto o peligroso.
  - El mayor tiempo en el diseño de una aplicación concurrente se emplea en aspectos relacionados con la vivacidad. Evitar bloqueos, incrementar el rendimientos, etc.

Notas:

Todo programa concurrente debe satisfacer dos tipos de propiedades:

•Propiedades de seguridad ("safety"): estas propiedades son relativas a que en cada instante de la ejecución no debe haberse producido algo que haga entrar al programa en un estado erróneo.

•Dos procesos no deben entrar simultáneamente en una sección crítica.

•El consumidor no debe consumir el dato antes de que el productor los haya producido.

•El productor no debe producir un dato mientras que el buffer de comunicación esté lleno.

•Propiedades de vivacidad ("liveness"): esta propiedades son relativas de que cada sentencia que se ejecute conduce en algún modo a un avance constructivo para alcanzar el objetivo funcional del programa. Són en general muy dependientes de la política de planificación que se utilice. Ejemplos de propiedades de vivacidad son:

•No deben producirse bloqueos activos (livelock). Por bloqueos activos se entienden conjuntos de procesos que ejecutan de forma continuada sentencias que no conducen a un progreso constructivo.

•Aplazamiento indefinido (starvation): consiste en el estado al que puede llegar un programa que aunque potencialmente puede avanzar de forma constructiva, no lo hace como consecuencia de que no se le asigna tiempo de procesador en la política de planificación.

Seguridad en sistemas concurrentes

- Las prácticas seguras de programación concurrente son generalizaciones de las practicas de programación secuencial segura.
- La seguridad tiene un componente temporal: en los programas concurrentes hay que garantizar que sólo se accede a los objetos cuando tienen un estado coherente (consistente).
- Un objeto es coherente si satisface todos los invariantes entre su atributos que son inherentes a su naturaleza.
- En los programas concurrentes hay que analizar todos los invariantes que debe satisfacer.
- Un objeto no debe ser utilizado mientras se encuentra en un estado transitorio incoherente.
- Un ejemplo son los conflictos de almacenamiento a bajo nivel:
  - Conflicto de lectura/escritura: Un thread no puede escribir un nuevo valor en un atributo, mientras otro thread lo está leyendo.
  - Conflicto de escritura/escritura: Dos threads concurrentes no pueden escribir un mismo atributo.

Notas:

Las prácticas seguras de programación concurrente son generalizaciones de prácticas de programación secuencial seguras y sólidas. La seguridad en diseños concurrentes agrega una dimensión temporal a las nociones comunes de seguridad.

Una diferencia entre seguridad en programas secuenciales y concurrentes es que la mayoría de las comprobaciones de seguridad no pueden ser comprobadas estáticamente por los compiladores. Esto tiene consecuencias importantes: Un programa se comprueba incorrecto en la compilación no puede ejecutarse, mientras que si el fallo solo es detectable en fase de ejecución, sólo puede ser previsto a fin de evitar sus consecuencias.

La seguridad en programas concurrentes añade una dimensión temporal a las tecnicas de diseño relativas a la seguridad. Se debe evitar el acceso a ciertas operaciones de objetos mientras estos mantienen estados no coherentes. Un estado coherente es aquel en el que todos los atributos del objeto posee valores legales y significativos. Para poder determinar los estados no coherentes de un objeto y poder evitar el acceso a ellos en ese estado, deben establecerse a nivel conceptual todos los invariantes asociados al objeto. Un así podrá comprobarse un estado como coherente si satisface todos sus invariantes.

Ejemplos de conflictos de lectura y escritura que pueden ser causa de errores son:

•Un objeto se representagráficamente por un hilo mientras está siendo actualizado.

•Un thread realiza una operación de transferencia sobre un saldo bancario mientras otro está haciendo un ingreso.

•Un elemento de una lista es utilizado mientras que aun no se ha establecifo su enlace al siguiente elemento.

Atributos y seguridad

- Los problemas de consistencia que comprometen la seguridad de un objeto provienen de definiciones de los atributos realizados a alto nivel y hechas sin tener en consideración su representación física.
- Deben introducirse algunas restricciones de representación para incrementar la seguridad:
  - Representaciones directas de valor.
  - Representaciones de valores derivados.
  - Representaciones lógicas de estado.
  - Variables sobre el estado de ejecución.
  - Variables históricas.
  - Variables de seguimiento de evolución
  - Referencias a relaciones.
  - Referencias a representaciones de objetos.

Notas:

Las categorías de campos y restricciones que deben tomarse en cualquier clase para mejorar el rendimiento e incrementar la seguridad son:

•Representaciones directas de valor: Hay veces que se introducen atributos que resumen propiedades globales de un objeto. Ej: Longitud de una lista.

•Representaciones de valores derivados: Los campos que se utilizan para eliminar o reducir al mínimo la necesidad de cálculos o invocaciones de métodos.

•Representaciones lógicas de estado: Reflejos del estado de control lógico de un objeto.

•Variables sobre el estado de ejecución: Campos que registran el estado dinámico de bajo nivel de un objeto. Ej: variable que representa un mensaje recibido y pendiente de ser tratado.

•Variables históricas: Representación de los estados previos que ha seguido un objeto. Ej: Registro histórico de mensajes recibidos.

•Variables de seguimiento de evolución: Un identificador, una referencia a objeto o marca de tiempo que caracteriza la evolución de un objeto.

•Referencias a relaciones: Campos que apuntan a otros objetos que interaccionan con el objeto anfitrión..

•Referencias a representaciones de objetos: Punteros que son almacenados por un objeto, para que a través de él se acceda a otros objetos relacionados con él.

Vivacidad y sistemas concurrentes

- En un programa concurrente hay muchas causas por las que un thread en estado de ejecución, no se ejecuta:
  - **Bloqueo**: La ejecución se suspende porque se requiere un recurso que está siendo utilizado por otro thread.
  - **Espera**: La ejecución se suspende a la espera de un evento, bien de temporización o bien procedente de otro thread.
  - **Entrada**: La suspensión se suspende en espera de un evento de entrada o salida
  - **Suspensión**: Se suspende la ejecución porque otro thread de mayor prioridad se está ejecutando.
  - **Fallo**: Se suspende porque se ha producido una excepción en el propio thread.

Notas:

Fallos de vivacidad

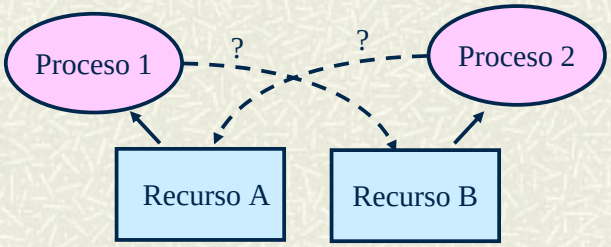
- Generalmente una falta temporal de ejecución es normal y aceptable. Sin embargo una falta permanente o ilimitada de ejecución es un problema que impide el correcto funcionamiento del programa:
  - **Interbloqueo**: Dependencia circular entre threads, en el que cada uno requiere para ejecutarse un recurso que posee otro.
  - **Perdida de señales**: Un thread permanece inactivo porque comenzó a esperar a un evento después de que el evento se haya producido.
  - **Mutex anidados**: Un thread permanece suspendido en un mutex que es requerido por otro thread que debe tomarlo para activar su ejecución.
  - **Fallo continuado**: Una actividad falla repetidamente.
  - **Inanición**: La CPU que tiene que ejecutar el thread está permanente ocupada por la ejecución de otros threads que tienen mayor prioridad de jecución.
  - **Falta de recursos**: El sistema no dispone de los recursos que se necesitan el threads para su ejecución.
  - **Fallo distribuido**: El thread requiere para ejecutarse el acceso a una maquina remota que nos está accesible.

Notas:

Interbloqueos

- Los bloqueos están relacionados con la reserva no ordenada de recursos a los que se acceden con exclusión mutua.

- Ejemplo:
  - Un proceso P_1 requiere para avanzar disponer de los recursos R_A y R_B.
  - Otro proceso P_2 requiere también los recursos R_A y R_B.
  - El proceso P_1 toma R_A y queda a la espera de R_B.
  - El proceso P_2 toma R_B y queda a la espera de R_A.
  - Los procesos P_1 y P_2 permanecen indefinidamente bloqueados.



Notas:

Bloqueos

Los bloqueos están en gran medida ligados con la necesidad de los diferentes procesos de un programas concurrentes de utilizar en régimen de exclusión mutua conjuntos de recursos. Para el uso exclusivo de un recurso, un proceso debe llevar a cabo una secuencia de tres pasos:

- Solicitar el recurso.
- Usar el recurso.
- Liberar el recurso.

El solicitud y reserva de forma no ordenada de varios recursos por parte de diferentes procesos, puede conducir a situaciones en que unos procesos consigan los recursos que también necesitan otros procesos, los cuales a su vez necesitan para avanzar recursos que ya han sido reservados por los primeros.

Un ejemplo típico de bloqueo es:

Consideremos un programa con dos procesos P1 y P2 concurrentes que en un momento dado necesitan disponer simultáneamente de dos recursos R1 y R2 que deben ser utilizado con exclusión mutua. Considérese que en un momento dado de ejecución del programa, P1 ha reservado R1 y ha solicitado R2, estando a la espera de que le sea concedido. Así mismo, P2 ha reservado R2, y ha solicitado R1, estando a la espera que le sea concedido. Obviamente esta es una situación de bloqueo de la que P1 y P2 no pueden salir.

Condiciones bajo las que hay bloqueos.

- Para que un bloqueo pueda ocurrir, se requiere que se produzcan simultáneamente estas condiciones
  - Los procesos utilizan recursos bajo régimen de exclusión mutua.
  - Los procesos mantienen reservados los recursos tomados mientras esperan los otros recursos que necesita.
  - No existe capacidad de despojar a los procesos de los recursos ya tomados.
  - Existe una cadena circular de requerimientos y reservas que conducen al bloqueo.
- Esta condiciones suelen existir, pero también se pueden eludir:
  - Puede establecerse que si un thread se bloquea por falta de un recurso, este libere todos los que tenía tomado en espera del que falta.
  - Puede establecerse que los thread tomen en bloque todos los recursos que necesita.
  - El controlador puede tener capacidad de retirar la cesión de los recursos asignados a los procesos bloqueados.
  - Puede establecer un orden de reserva de recursos que haga imposible las dependencias circulares.

Notas:

Se han identificado cuatro condiciones necesarias y suficientes para que en un programa se puedan producirse bloqueos (Coffman, 1971):

•Los procesos afectados necesitan recursos compartidos que deben ser usados bajo exclusión mutua.

•Los procesos mantienen reservados los recursos que le son concedidos, mientras que espera que le sean concedidos los que le faltan.

•No existe capacidad de despojar a los procesos de recursos ya concedidos.

•Existe una cadena circular de requerimientos y reservas que conducen al bloqueo.

Tratamiento de los bloqueos

Existen dos estrategias para evitar bloqueos:

- Se **introduce un gestor** que arbitra el acceso a los recursos y detecta los bloqueos y que cuando lo encuentra arbitra una solución.
- Se **diseñan los programas libres de bloqueos**. Esta admite dos opciones:
  - Se **evitan los bloqueos** mediante técnicas anticipativas: (Política del banquero)
  - Se **diseñan los programas** estructuralmente libres de bloqueos (Técnicas formales).

Notas:

Tratamiento de los bloqueos:

Existen dos estrategias para tratar los sistemas concurrentes con posibles bloqueos:

•Introducir componentes en el programa que puedan detectar cuando un bloqueo se ha producido, y disponga de medios para sacar al sistema del mismo.

•Diseñar los programas con una estructura en la que los bloqueos no son posibles. En este caso, caben a su vez dos opciones:

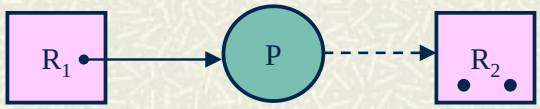
•Evitar los bloqueos: en las que se toman medidas con anticipación en el programa para que el bloqueo no pueda tener lugar.

•Prevenir los bloqueos: consiste en diseñar el sistema teóricamente con tal estructura que no sean posible alcanzar en él, situaciones de bloqueo.

Las técnicas de detección/recuperación de bloqueos, y de evitación/prevención de bloqueos no tienen que ser excluyentes. Es frecuente, que en un mismo sistema se utilice un método para tratar un cierto tipo de bloqueos, y se utilice el otro método para evitar otros tipos de bloqueos.

Grafo de reserva de recursos

- Es un grafo orientado que representa el estado de asignación y requerimiento de recursos por parte de los procesos de una aplicación:
  - Nudos: Existe un nudo en el grafo por cada proceso y por cada recurso de la aplicación.
  - Arcos:
    - Cuando un recurso está asignado a un proceso se establece un arco (continuo) del nudo que representa el recurso al nudo que representa al proceso.
    - Cuando un proceso tiene requerido un recurso se establece un arco (discontinuo) del nudo que representa el proceso al nudo que representa el recurso.
  - Cuando un recurso tiene varias replicas, se introducen tantos puntos como replicas tenga. Cada una de ellas puede ser asignada independientemente.



Notas:

Los **métodos de detección/recuperación** de bloqueos se caracterizan por permitir que se produzcan los bloqueos, e introducir elementos con capacidad de detectar ese estado cuando se produce, y disponer de herramientas para resolverlo.

En el ejemplo de la figura se muestra un proceso P y dos recursos R1 y R2. El recurso R2 ofrece dos replicas, luego puede ser potencialmente asignado a dos procesos que lo requiera.

En el grafo de asignación de recursos de la figura, expresa:

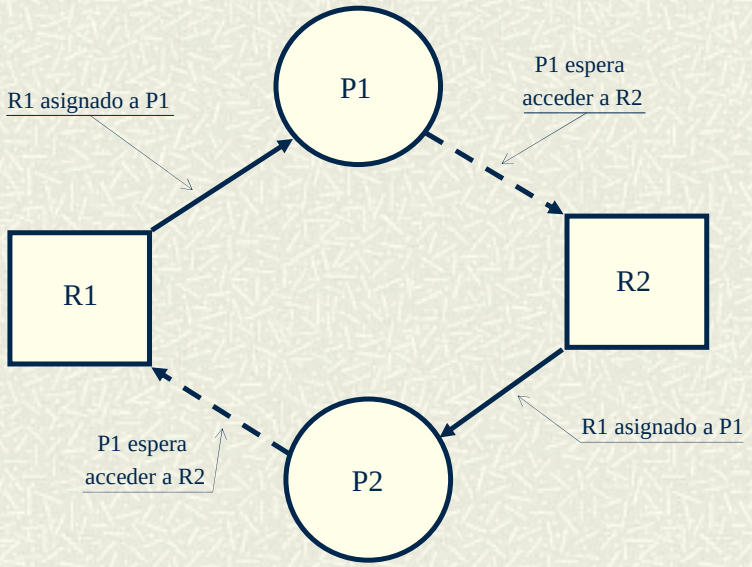
•El recurso R1 ha sido asignado al proceso P.

•El proceso P ha requerido el recurso R2

Para detectar el bloqueo se requiere disponer de un método de representar la situación en que se encuentra la reserva de los recursos. El método más utilizado es el **grafo de reserva de recursos**, este es un grafo orientado en el que los nudos son tanto los recursos reservados o solicitados y los procesos que los solicitan o aceptan. Un arco de un proceso a un recurso, significa que el proceso ha solicitado el recurso, mientras que un arco de un recurso a un proceso representa una asignación del recurso al proceso.

Métodos de detección de recursos.

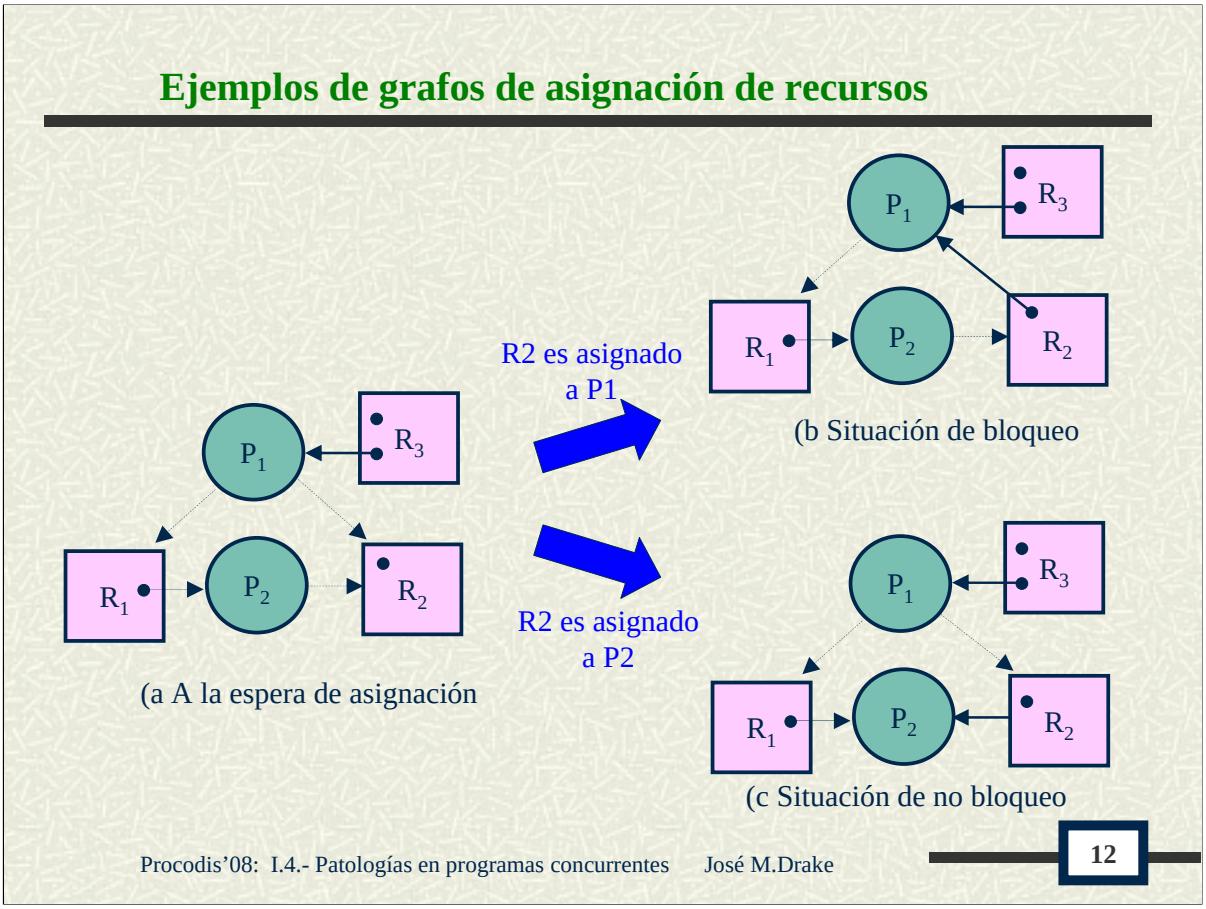
Grafo de reserva de recursos.



Notas:

En el diagrama de reserva de recursos, un bloqueo se detecta como un bucle en el grafo:

- Si todos los recursos que participan en el grafo son de réplica única, la condición necesaria y suficiente para que exista un interbloqueo es que exista un un bucle. Los procesos y recursos en el bucle son los procesos y recursos que participan en el interbloqueo.
- Si algunos de los recursos que participan en el bucle tienen replica, la existencia del bucle es condición necesaria, pero no suficiente.

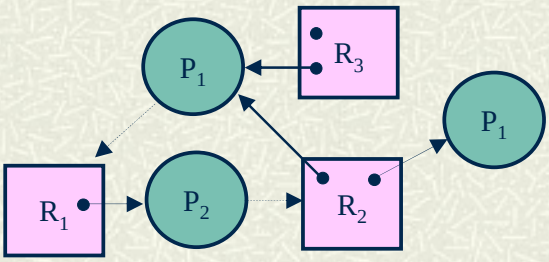


Notas:

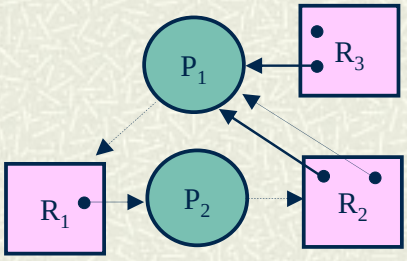
Se muestra la evolución de un grafo de asignación de recursos:

1. La situación es transitoria. Los procesos P1 y P2 han requerido el recurso R2 que se encuentra libre. No hay aún bloqueo, pero está pendiente una decisión de asignación.
2. El recurso ha sido asignado al proceso P1 y se produce una situación de bloqueo, en las que intervienen P_1,P_2, R_1 y R_2.
3. El recurso ha sido asignado a P2, no existe bloqueo. P2 puede continuar su ejecución.

Ejemplos con recursos con replicas



(aunque hay bucle, no hay bloqueo



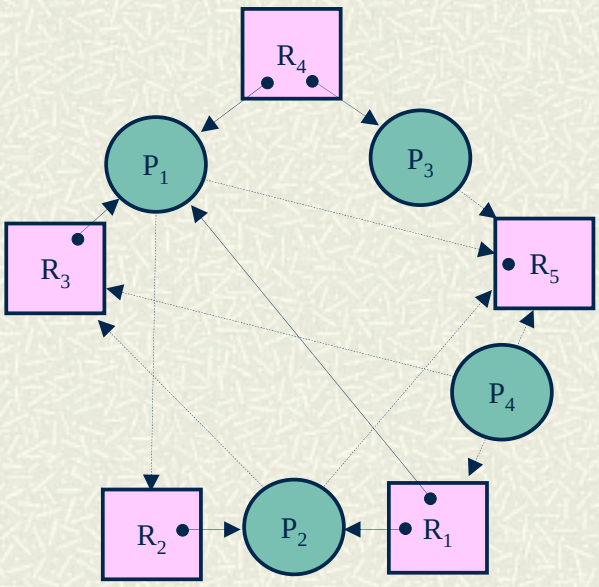
(Hay bucle, y hay bloqueo

Notas:

1. En este caso, aunque existe el bucle P_1-R_1-P_2-R_2, la naturaleza de R2 con dos replicas hace que no se produzca interbloueo. Cuando P_3 finalice de utilizar la replica de R_2, la liberará y P_2 podrá continuar. Por tanto, en este estado no hay bloqueo.

2.En este caso hay bucle P_1-R_1-P_2-R_2, y también existe bloqueo.

Estructura de datos para la gestión de grafos de asignación.



R

2	1	1	2	1

R: Recurso en sistema

R

P

1	0	1	1	0
1	1	0	0	0
0	0	0	1	0
0	0	0	0	0

A: Matriz de asignación

R

P

0	1	0	0	1
0	0	1	0	1
0	0	0	0	1
1	0	1	0	1

B: Matriz de requerimientos

R

0	0	0	0	1

V: Recursos disponibles

Notas:

A fin de detectar estados de bloqueo en un grafo de asignación de recurso, este se puede representar mediante dos matrices:

- R: Vector de recursos del sistema: Representa el número de replicas de cada tipo de recursos que se encuentra disponible en el sistema.
- A: Matriz de asignación: representa los recursos que han sido asignados a cada procesador.
- B: Matriz de requerimientos: Representa los recursos que han sido requerido por cada procesador. y aún no les ha sido asignados.
- V: Vector de recursos disponibles en el sistema: Representa los recursos que aún no han sido asignados. Resulta de restar al vector R cada fila de la matriz A

APLICACION AL EJEMPLO:

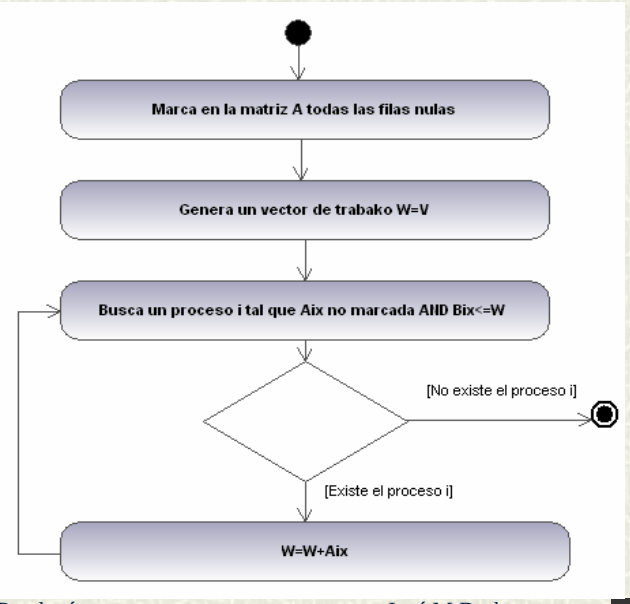
1) Se marca el procesador 4
2) Se calcula W=[0,0,0,0,1]
3) Se busca el prceso i=4: no está marcado y B3x=[0,0,0,0,1] <= W=[0,0,0,0,1]
4) Se marca el proceso 3 yse revalua W= W*A3x = [0,0,0,1,1]

3) Se nusca un nuevo proceso i no marcado y tal que Bix<=[0,0,0,1,1]. No existe luego se termina.

Existe un bloqueo en el que participa P1 y P2. Efectivamente existe el bucle P1-R2-P2-R3-P1

Un algoritmo para detecciones de bloqueos.

- El objetivo del algoritmo es buscar los procesos que no son parte de un bloqueo.



Notas:

1) Se marcan todas las filas de la matriz de asignación A que son nulas. Esto es obvio, puesto que un proceso que no tiene asignados recursos no puede ser parte de un bloqueo.
2) Se genera el vactor de trabajo W y se inicializa al vector V de recursos disponibles.
3) Se busca el proceso i, tal que corresponda a una fila de A que no este marcada, y que satisfaga en la matriz de requerimientos que Bix<=W
4) Si no existe un proceso i tal como el buscado en (3 se ha terminado el proceso.
5) Si existe un proceso i tal como el buscado en (3, se marca la fila i de la matriz A y se revalua W=W+Ai
6) Se vuelve al punto 3)

Los bloqueos que existan corresponden a los procesadores que corresponden a las filas de la matriz A que no están marcadas

Criterios de arbitrar un bloqueo.

- Detectado un bloqueo:
  - Abortar todos los procesos afectados por el bloqueo.
  - Abortar de uno en uno, los procesos que intervienen en el bloqueo, hasta que el mismo desaparece.
  - Liberar desde fuera los recursos afectados, uno a uno hasta que el bloqueo desaparece.

- Criterios de liberación de los recursos:
  - La prioridad de los procesos.
  - El tiempo que cada proceso lleve ya de ejecución.
  - El tiempo que se prevé que aún resta para concluir el proceso.
  - El número de recursos que requieren los recursos.
  - La naturaleza de los recursos que son requeridos por cada proceso.

Notas:

Una vez que un bloqueo ha sido detectado, se suelen utilizar tres procedimientos de recuperación:

•Abortar todos los procesos afectados por el bloqueo.

•Abortar de uno en uno, los procesos que intervienen en el bloqueo, hasta que el mismo desaparece.

•Liberar desde fuera los recursos afectados, uno a uno hasta que el bloqueo desaparece.

Cuando los procesos se abortan, los recursos se liberan, y luego se reinician de nuevo los procesos. Obviamente esto supone una fuerte penalización temporal, por lo que es muy importante tener criterios para elegir la secuencia en que los procesos se van abortando. Algunos de estos criterios pueden ser:

•La prioridad de los procesos.

•El tiempo que cada proceso lleve ya de ejecución.

•El tiempo que se prevé que aún resta para concluir el proceso.

•El número de recursos que requieren los recursos.

•La naturaleza de los recursos que son requeridos por cada proceso.

Algoritmo del banquero.

- Existe un gestor que administra los recursos del sistema.
- Se conocen el número máximo de recursos que puede requerir cada proceso.
- El gestor no entrega un recurso si no tiene garantía de que aún quedan recursos para que al menos uno de los procesos en ejecución puedan terminar.

Proceso	Requiere
A	4
B	6
C	8
Recursos: 12	

Estado admitido
A ← 1
B ← 4
C ← 5
Restan: 2

Estado no admitido
A ← 2
B ← 4
C ← 5
Restan: 1

Notas:

Una estrategia muy utilizada es el **algoritmo del banquero.** En este algoritmo, cada proceso debe declarar el número máximo de recursos que puede llegar a requerir, y la concepción de recursos se conceden solo si se verifica que tras la concesión quedan recursos para que al menos alguno de los procesos pueda evolucionar de forma segura aún cuando requiriera el número máximo de recursos.

Ejemplo: Supóngase que existen tres procesos A, B y C, y se dispone de12 disponibilidades de un cierto recursos. El máximo De las disponibilidades de recursos declaradas por cada proceso son:

Proceso	#Rec.Decl
A	4
B	6
C	8

Con este algoritmos las siguientes situaciones de concesión de recursos son respectivamente admitida y no admitida: A

Estado admitido

Proceso	#Rec.Decl.	#Rec. Conc.
A	4	1
B	6	4
C	8	5

Este estado es admitido, ya que aún quedan dos recursos, por lo que en el peor caso B podría concluir sin bloqueo, y luego concluirían los otros.

Estado no admitido

Proceso	#Rec.Decl.	#Rec. Conc.
A	4	2
B	6	4
C	8	5

Este estado no es admitido, ya que solo queda un recurso, y cabe la posibilidad de que todos los procesos quedasen bloqueados si en el peor caso todos pidieran el máximo de recursos.