Gestión de recursos: interbloqueo e inanición

Gustavo Romero

Arquitectura y Tecnología de Computadores

7 de enero de 2009

Índice

Lecturas recomendadas

J. Bacon	Operating Systems (11, 18)
A. Silberschatz	Fundamentos de Sistemas Operativos (7)
W. Stallings	Sistemas Operativos (6)
A. Tanuenbaum	Sistemas Operativos Modernos (3)

Gestión de recursos

Motivación

- Los procesos pueden utilizar los **recursos** de los que dispone la máquina sobre las que se ejecutan.
- Los recursos pueden ser de distinta naturaleza:
  - Hardware.
  - Software.
- La forma de utilizar esos recursos puede conducir a situaciones en las que el **progreso** de los procesos involucrados es **imposible** $\Longrightarrow$ bloqueo irreversible o **interbloqueo**.
  - Hardware:
    - A: adquiere el escáner y además la unidad de cd.
    - B: adquiere la unidad de cd y además el escáner.
  - Software:
    - A: bloquea registro X y Z de una BD
    - B: bloquea registro Z y X de una BD.
  - En el tema anterior hemos visto muchos ejemplos...
- El conflicto requiere de al menos **dos agentes activos**.

Tipos de recursos

- Hardware:
  - procesador
  - memoria: RAM y disco
  - red
  - monitor
  - impresora
  - ...
- Software:
  - búfer
  - ficheros
  - directorio
  - ...

También pueden aparecer interbloqueos en los que no están involucrados recursos.

Recursos expropiables y no expropiables

- Un recurso se dice **expropiable** si puede serle **retirado** al proceso que lo posee **sin** causarle **daño** alguno, ej: memoria.
- Un recurso se dice **no expropiable** si no puede serle **retirado** al propietario sin hacer que **falle**, ej: unidad de cd, impresora.

- En los interbloqueos suelen intervenir recursos no expropiables.

Protocolo de utilización de recursos (1)

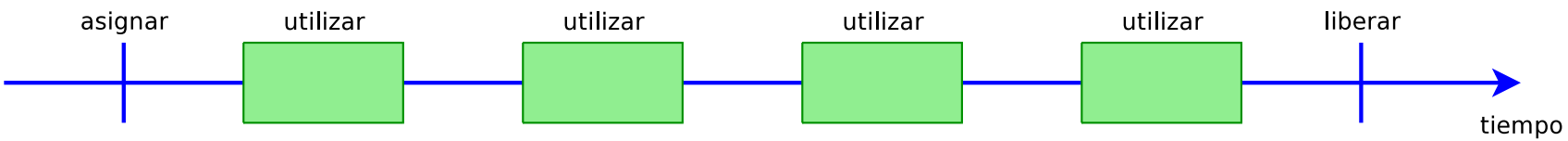
Para utilizar un recurso se sigue este protocolo:

- **solicitar** el recurso.
- **utilizar** el recurso.
- **liberar** el recurso.

Ante una solicitud de recurso que no pueda ser atendida podemos hacer dos cosas:

- **bloquear** el proceso hasta que el recurso quede disponible de nuevo.
- hacer que la solicitud falle y comunicarlo devolviendo un **código de error**.

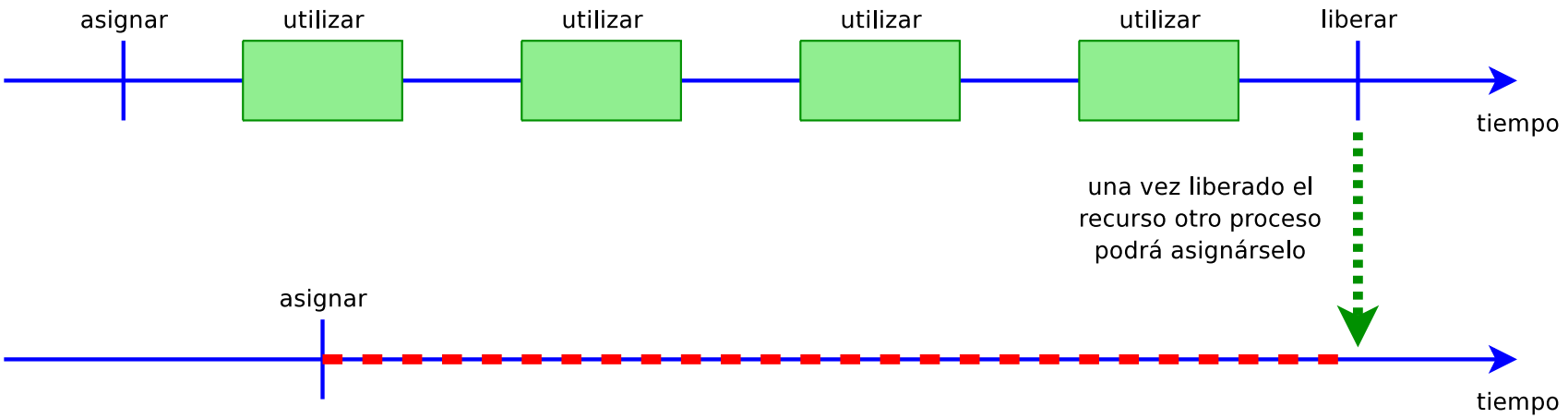
Protocolo de utilización de recursos (2)



La utilización de un recurso es parecida a la gestión de una sección crítica:

- asignar $\sim$ entrar_sección_crítica
- liberar $\sim$ salir_sección_crítica

Protocolo de utilización de recursos (3)



- EL segundo proceso/hebra debe esperar a que el primero acabe de utiliza el recurso y lo libere.
- Sólo entonces el segundo podrá finalizar la asignación del recurso antes de poder utilizarlo.

Adquisición de recursos (1)

- Algunos recursos son administrados por el **usuario**, ej: bases de datos, memoria compartida,...
- Un método de administración podría ser asociar un semáforo a cada recurso.
- El **orden** de adquisición de recursos es **importante**.

un recurso

```
semáforo recurso;

recurso.decrementar();
usar_recurso();
recurso.incrementar();
```

varios recursos simultáneos

```
semáforo recurso1, recurso2;

recurso1.decrementar();
recurso2.decrementar();
usar_recursos();
recurso2.incrementar();
recurso1.incrementar();
```

Adquisición de recursos (2)

2 procesos/hebras tipo núcleo necesitan 2 recursos: interbloqueo en el peor caso

Programa 1 ⟶ ok

```
semáforo recurso1, recurso2;

void* hebra_a(void*){
    recurso1.decrementar();
    recurso2.decrementar();
    usar_recursos();
    recurso2.incrementar();
    recurso1.incrementar();
}

void* hebra_b(void*){
    recurso1.decrementar();
    recurso2.decrementar();
    usar_recursos();
    recurso2.incrementar();
    recurso1.incrementar();
}
```

Programa 2 ⟶ interbloqueo

```
semáforo recurso1, recurso2;

void* hebra_a(void*){
    recurso1.decrementar();
    recurso2.decrementar();
    usar_recursos();
    recurso2.incrementar();
    recurso1.incrementar();
}

void* hebra_b(void*){
    recurso2.decrementar();
    recurso1.decrementar();
    usar_recursos();
    recurso1.incrementar();
    recurso2.incrementar();
}
```

Adquisición de recursos (3)

2 procesos necesitan 6 unidades de las 10 de un recurso: interbloqueo en el peor caso

Proceso 1

```
recurso.decrementar(1);
recurso.decrementar(1);
recurso.decrementar(1);
recurso.decrementar(1);
recurso.decrementar(1);
recurso.decrementar(1);
usar_recurso();
recurso.incrementar(6);
```

Proceso 2

```
recurso.decrementar(1);
recurso.decrementar(1);
recurso.decrementar(1);
recurso.decrementar(1);
recurso.decrementar(1);
recurso.decrementar(1);
usar_recurso();
recurso.incrementar(6);
```

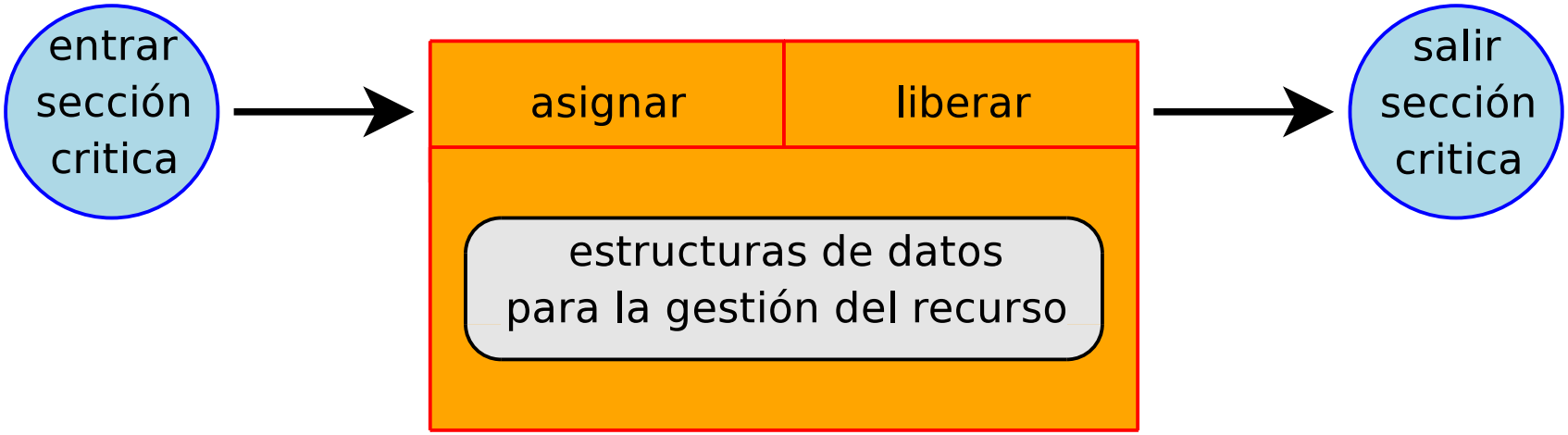
Gestión de recursos: recursos **exclusivos**

- Un recurso se dice exclusivo cuando se asigna como una **unidad única e indivisible**.
- Un gestor de recursos exclusivos podría implementarse como un **monitor** con dos operaciones:
  - `asignar(recurso)`
  - `liberar(recurso)`
- Existe un gran **parecido** entre la gestión de recursos exclusivos y las secciones críticas ejecutadas en exclusión mutua.

Gestión de recursos: recursos **divisibles**

- Un gestor de recursos divisibles también podría implementarse como un **monitor**, pero con dos operaciones ligeramente diferentes:
  - `asignar(recurso,cantidad,localización)`
  - `liberar(recurso,cantidad,localización)`
- Este tipo de recursos pueden asignarse **por partes**, con lo que deberemos especificar tanto la **cantidad** como la **localización** de cada una de las partes.

Gestión de recursos



Gestión de interbloqueos

Definición de interbloqueo

- Definición:

  Un conjunto de procesos/hebras se encuentra interbloqueado si cada uno de ellos está esperando un suceso que sólo otro proceso del conjunto puede causar.

- Habitualmente el suceso esperado es la liberación de algún tipo de recurso o sección crítica.
- Ninguno de los procesos puede...
  - ejecutarse
  - liberar un recurso o sección crítica
  - ser despertado (desbloqueado)

Condiciones **necesarias** para el interbloqueo

Exclusividad: Cada recurso o es asignado exclusivamente a un proceso o está disponible.

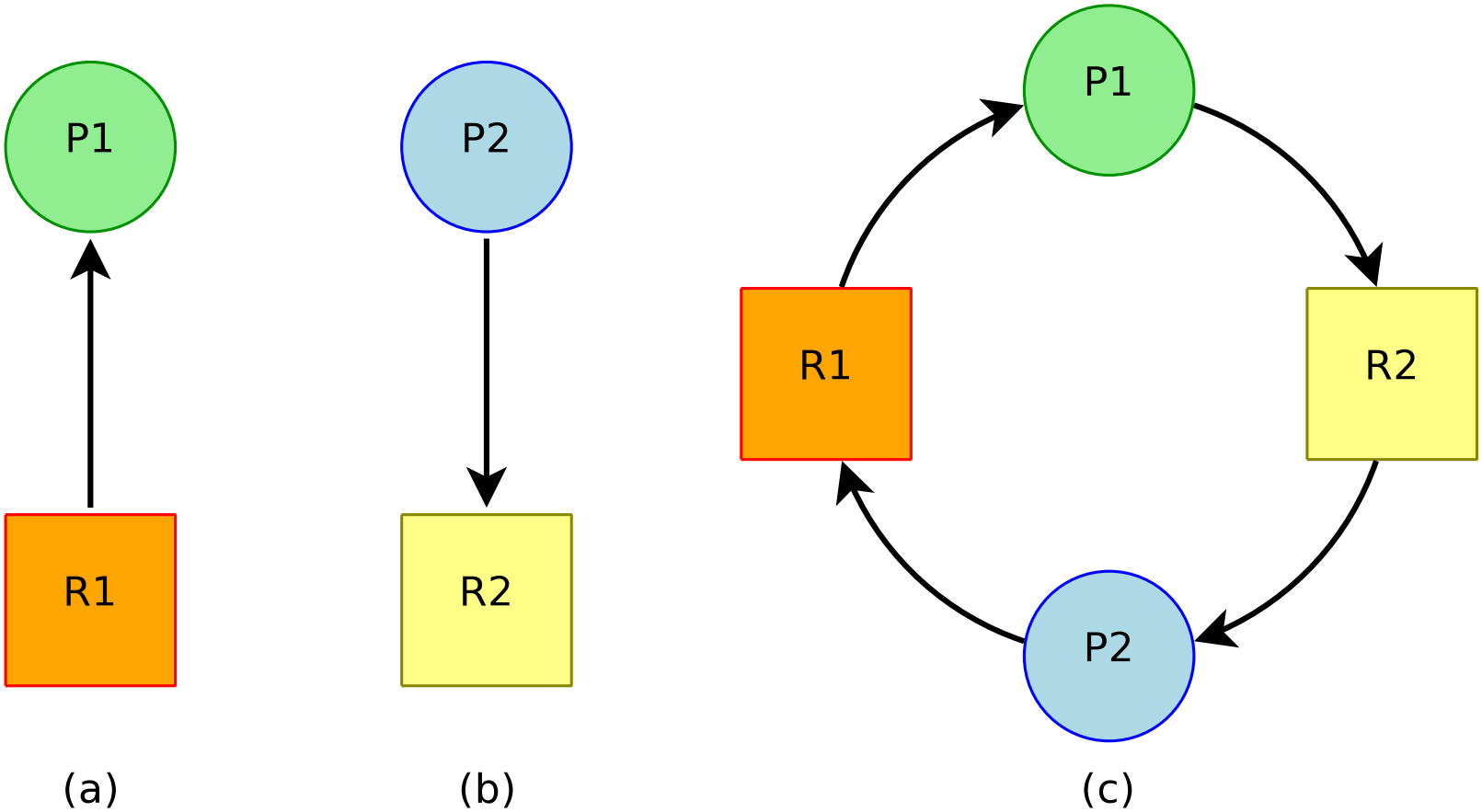
Retención y espera: Los procesos que tiene recursos asignados pueden solicitar nuevos recursos.

No expropiación: Los recursos ya asignados no pueden ser arrebatados al proceso sino que este debe liberarlos de forma voluntaria.

Espera circular: Debe existir una cadena circular de dos o más procesos cada uno de los cuales espera por un recurso en manos del siguiente proceso.

Condiciones necesarias pero no suficientes (Coffman, 1971).

Grafo de asignación de recursos



a) El recurso R1 ha sido asignado al proceso P1.
b) El proceso P2 espera a que el recurso R2 le sea asignado.
c) Interbloqueo.

Resolución de interbloqueos

Prevenir: anular una de las cuatro condiciones necesarias para la aparición de un interbloqueo.

Evitar: asignar recursos de forma cuidadosa para que no pueda aparecer el interbloqueo.

Detectar y recuperar: intentar detectar el interbloqueo y tomar medidas para resolverlo en caso de que aparezca.

Ignorar: no hacer nada con la esperanza de que no aparezca el interbloqueo.

Prevenir: atacar la condición de exclusión mutua

- Algunos recursos necesitan ser utilizados de forma **exclusiva**, ej: impresora, teclado, pila,...
- Con algunos de estos recursos podemos hacer **"spooling"**:
  - todo proceso cree utilizar de forma inmediata y exclusiva el recurso.
  - sólo el demonio del recurso lo utiliza realmente.
  - no aplicable a todos los recursos, ej: tabla de procesos.
- Otros dispositivos pueden ser **virtualizados**:
  - multiplexación del uso del procesador en el tiempo.
- Tanto el spooling como la virtualización introducen **nuevos problemas** en el sistema, ej: competición por espacio en memoria y/o disco.

Prevenir: atacar la condición de retener y esperar

- El proceso debe **solicitar por adelantado** todos los recursos que va a necesitar.
- Si consigue todos los recursos podrá finalizar y devolverlos.
- Si no puede conseguirlos todos simultáneamente deberá esperar.
- Inconvenientes:
  - Todo proceso debe conocer por adelantado que recursos va a necesitar.
  - No se hace un uso óptimo de los recursos.
  - La espera para poder disponer de todos los recursos puede ser larga.

Prevenir: atacar la condición de no expropiación

- ¿A quién expropiar?
  - ¿Al proceso que **solicita** un recurso ya asignado?
  - ¿Al proceso **propietario** del recurso solicitado?
- La expropiación requiere la capacidad de **almacenar** el estado del recurso expropiado para su posterior **devolución** $\longrightarrow$ no siempre posible.
- La expropiación de alguno recursos puede producir resultados **erróneos**, ej: impresora.

¿Se le ocurre alguna forma de hacer una impresora expropiable?

Prevenir: atacar la condición de espera circular

- 1ª solución: permitir a cada proceso la asignación de **un único recurso** $\longrightarrow$ demasiado restrictiva.
- 2ª solución: imponer un **orden estricto** entre recursos para su asignación:
  - Una vez que un proceso recibe un recurso i, puede volver a solicitar otros si y sólo si su número de orden j es mayor, $j > i$.
  - Aplicable a múltiples procesos: el que tenga el recurso con el mayor número de orden siempre podrá finalizar.
  - Puede ser imposible encontrar una ordenación que satisfaga a todo el mundo.
- De esta forma en el grafo de asignación de recurso nunca podrá aparecer un ciclo con lo que nunca se producirá un interbloqueo.

Viabilidad de la prevención

condición	estrategia
exclusión mutua	multiplexación
retener y esperar	solicitar recursos por adelantado
no expropiación	expropiar recursos
espera circular	ordenación de recursos

- Solución demasiado **restrictiva**, ej: asignación en orden.
- **Utilización** de recursos demasiado **baja**, ej: solicitar al principio y no poder liberar hasta la finalización.
- Demasiada **sobrecarga**, ej: la expropiación requiere tiempo y espacio para almacenar y restaurar.
- Consecuencia $\longrightarrow$ ninguna de estas soluciones se emplea en sistemas comerciales.

Evitar interbloqueos (1)

- Actuar sobre las decisiones que el sistema toma en tiempo de ejecución con respecto a si la creación de una **nueva hebra** o una **nueva solicitud** de recurso podría conducir a un interbloqueo.
- Dos políticas son posibles:
  - **No iniciar** una **nueva hebra** si la suma de todos los recursos que necesita más los recursos ya asignados a otras hebras puede conducir a un interbloqueo.
  - **No conceder** solicitudes de **nuevos recursos** si su asignación puede llevar el sistema a un interbloqueo.
- Siempre hemos de tener en cuenta el peor caso posible.
- La mayor restricción es **conocer por adelantado** los recursos que va a necesitar un proceso.

Modelo teórico para evitar interbloqueos (1)

n hebras: $H_1, H_2, \cdots H_n$

m tipos de recursos: $R_1, R_2, \cdots R_m$

cantidad total de cada recurso: $R = (r_1, r_2, \cdots, r_m)$

cantidad de recursos disponibles $D = (d_1, d_2, \cdots, d_m)$

matriz de solicitudes:
$$S = \begin{pmatrix} S_{11} & S_{12} & \cdots & S_{1m} \\ S_{21} & S_{22} & \cdots & S_{2m} \\ . & . & \cdots & . \\ S_{n1} & S_{n2} & \cdots & S_{nm} \end{pmatrix}$$

matriz de asignación:
$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ . & . & \cdots & . \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{pmatrix}$$

Modelo teórico para evitar interbloqueos (2)

Invariantes y restricciones:

- Los recursos o bien están disponibles o bien han sido asignados:

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

- Ninguna hebra solicita más recursos de los que posee el sistema:

$$S_{ki} \leq R_i \quad \forall k, i$$

- Ninguna hebra recibe más recursos de los que solicita:

$$A_{ki} \leq S_{ki} \quad \forall k, i$$

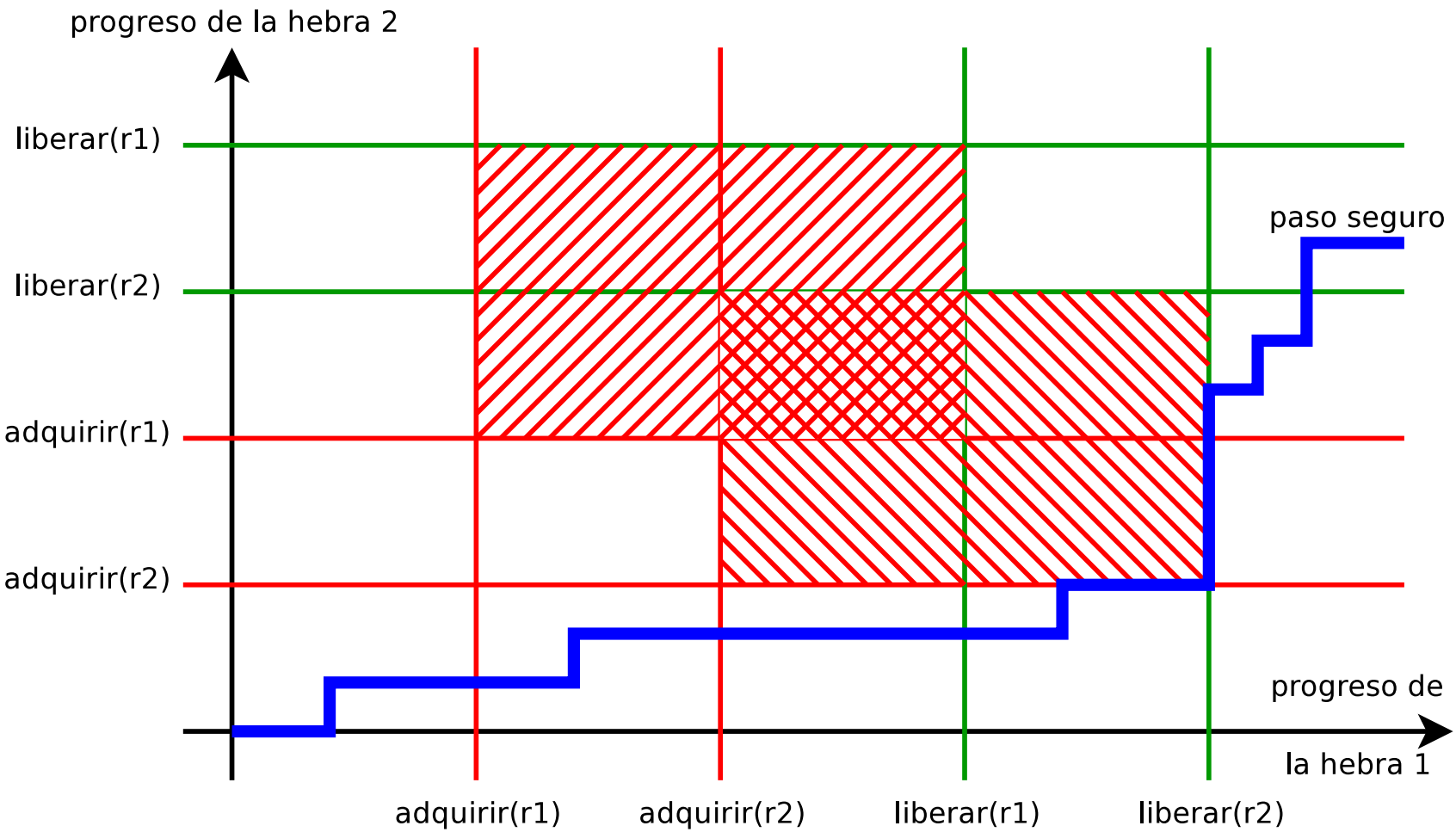
Modelo teórico para evitar interbloqueos (3)

- Iniciar una nueva hebra o asignar un nuevo recurso si y sólo si:

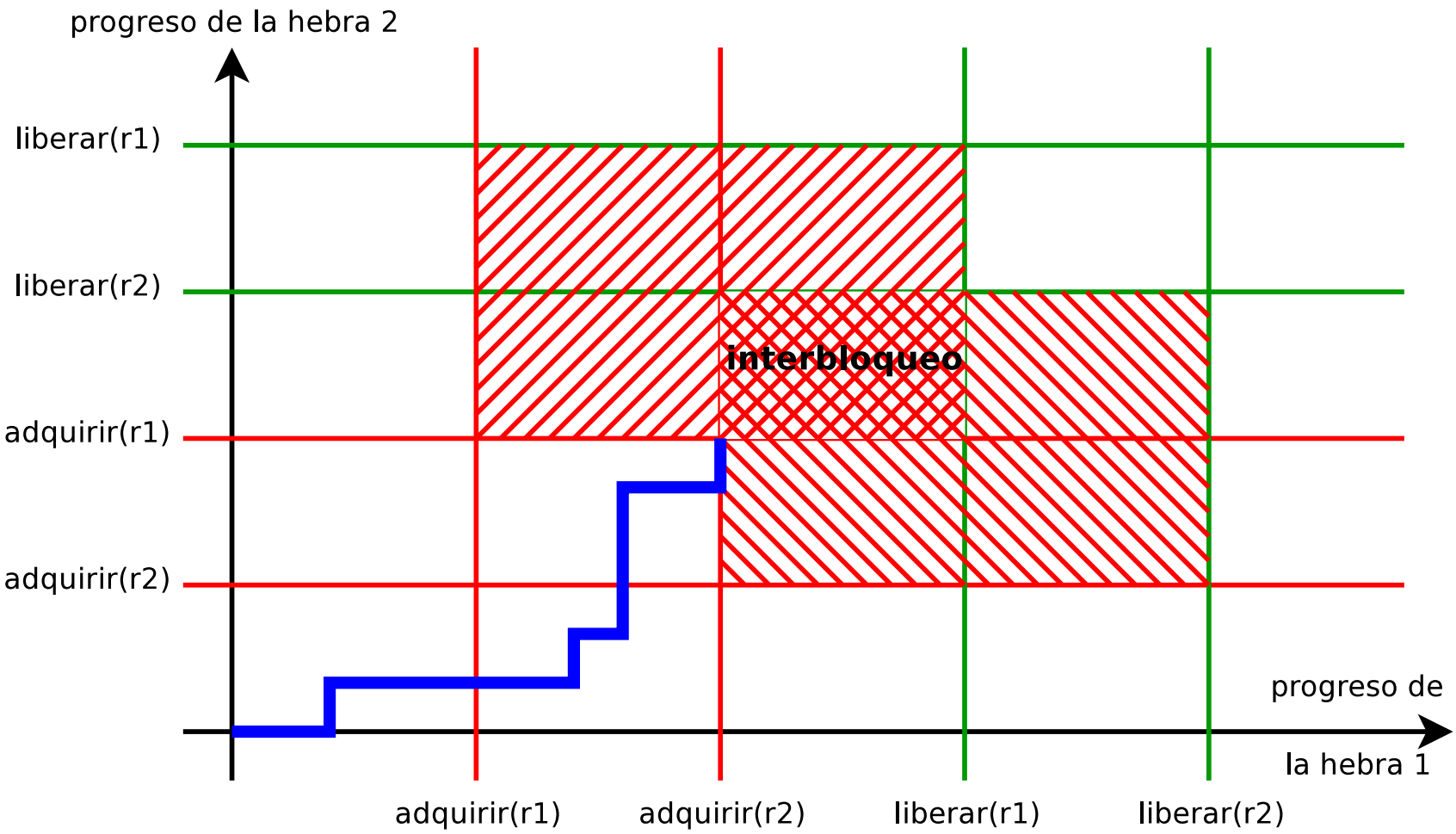
$$R_i \geq S_{n+1,i} + \sum_{k=1}^{n} S_{ki} \quad \forall i$$

- Esta política es muy **pesimista** porque asume que todas las hebras solicitarán sus recursos al **mismo tiempo**.
- En la práctica sin embargo suele pasar que...
  - Algunos de los recursos nunca son solicitados.
  - Pocas hebras necesitarán todos sus recursos a la vez.
- Es necesario un **algoritmo más optimista y mejor** $\Longrightarrow$ **Algoritmo del banquero**.

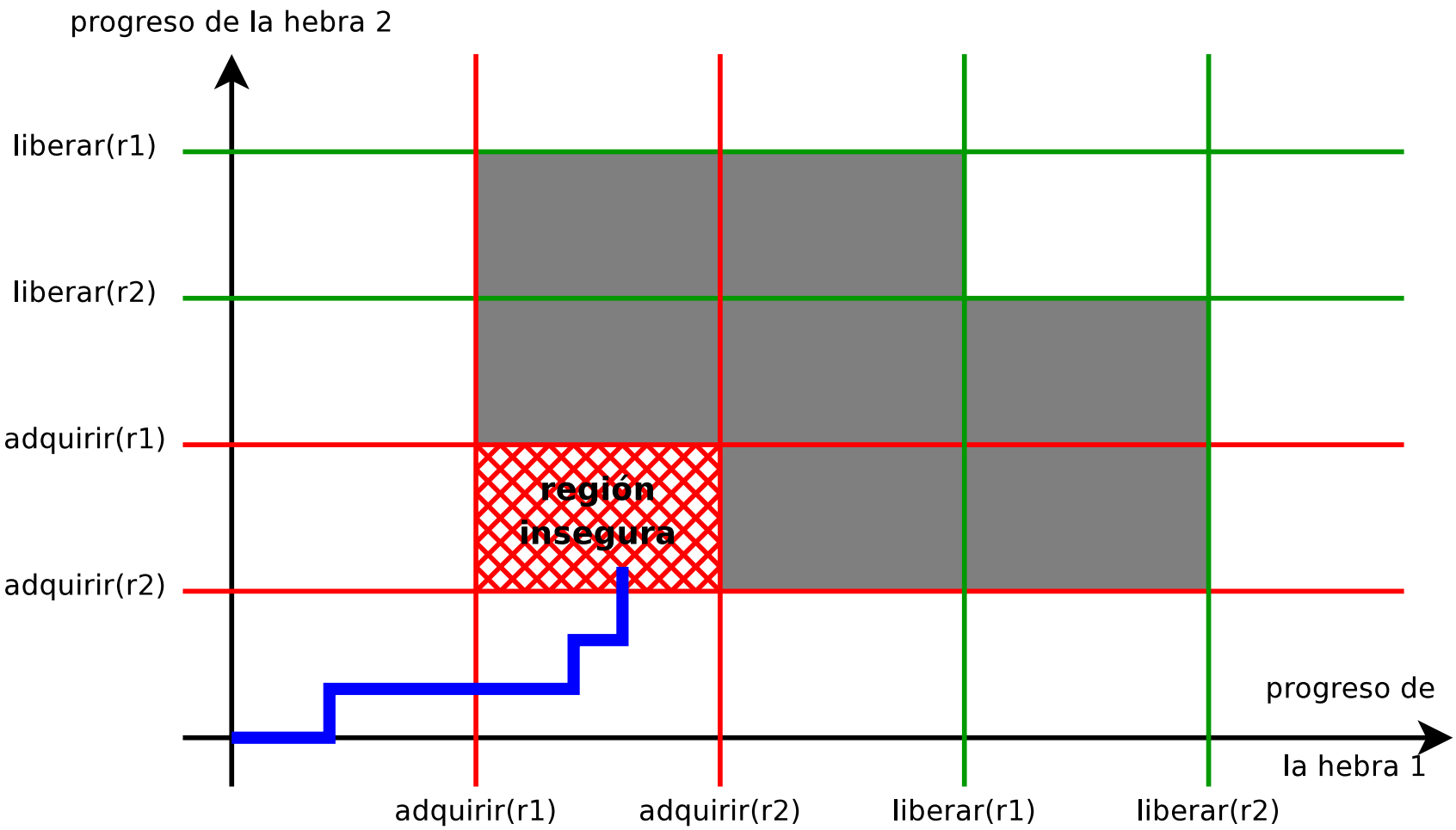
Evitar interbloqueos: estados seguros e inseguros (1)



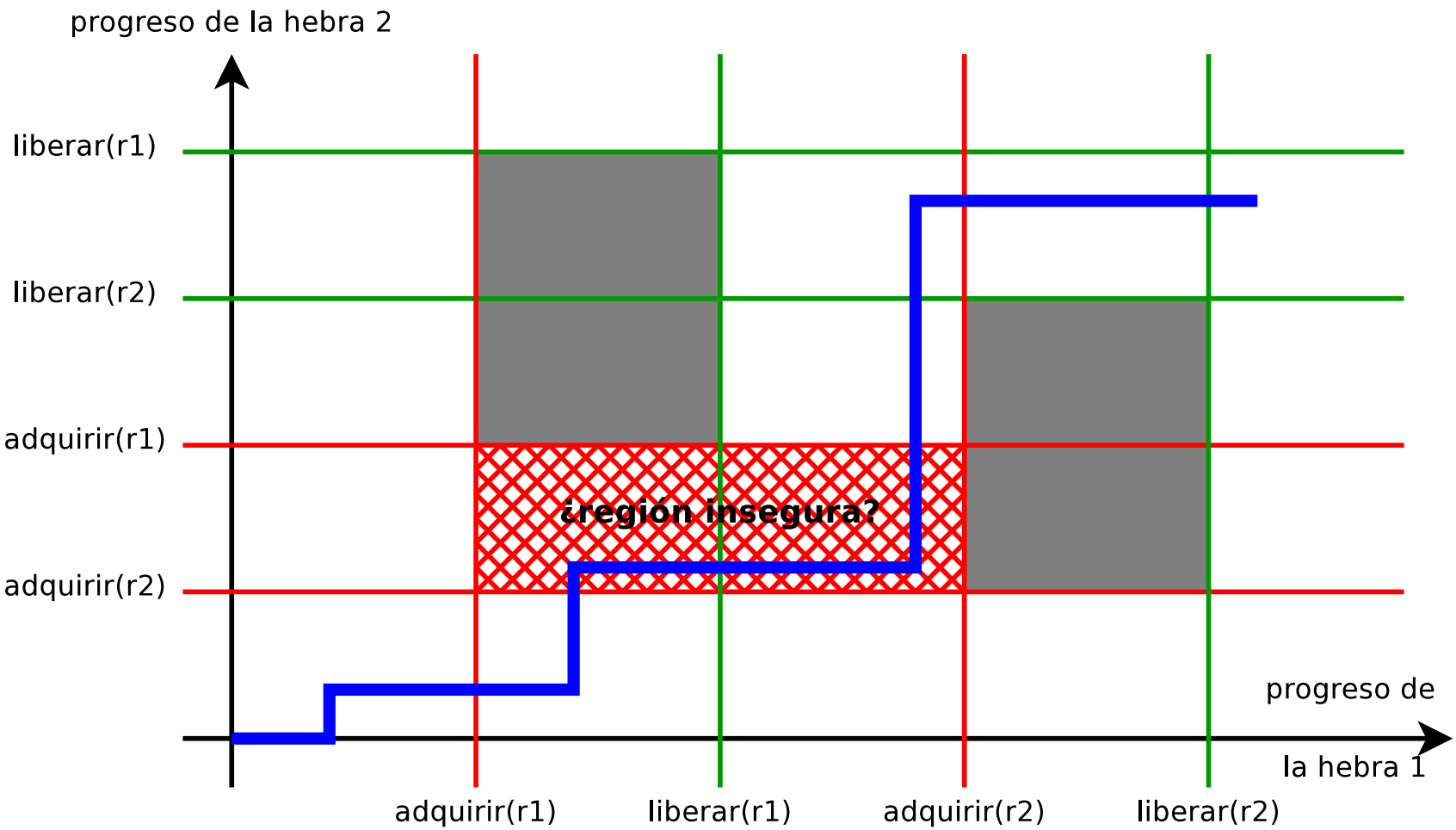
Evitar interbloqueos: estados seguros e inseguros (2)



Evitar interbloqueos: estados seguros e inseguros (3)



Evitar interbloqueos: estados seguros e inseguros (4)



Estado seguro

Definición: El estado de un sistema con n hebras es **seguro** mientras exista al menos una **secuencia de ejecución** que permita a **todas finalizar**.

Formalmente:

El estado del sistema es seguro si y sólo si existe una permutación $< H_{k_i}, H_{k_2}, \cdots H_{k_n} >$ dentro de $\{H_1, H_2, \cdots H_n\}$

$$\forall i \epsilon \{1, 2, \cdots n\} : \quad S_{k_i} - A_{k_i} \leq D_i + \sum_{s=1}^{i-1} A_{k_s}$$

o

$$\forall i \epsilon \{1, 2, \cdots n\} : \quad S_{k_i} - A_{k_i} \leq R_i - \sum_{s=1}^{n} A_{k_s}$$

El algoritmo del banquero $O(n^2 m)$

```
respuesta_t banquero()
{
  respuesta_t respuesta = desconocida;
  hebra HI[N] = H; // hebras interbloqueadas, inicialmente todas

  for(int i = 0; i < N; ++i) // para cada hebra
    if (H[i] pertenece HI && S[i,*] - A[i,*] < D[*]) // ¿asignable?
    {
      sacar H[i] de HI[i]; // sacar hebra de interbloqueadas
      D[*] = D[*] - A[i,*]; // actualizar recursos disponibles
      if (HI vacio)
        respuesta = seguro;
    }
    else
    {
        respuesta = inseguro;
    }
  return respuesta;
}
```

Detección de interbloqueos (1)

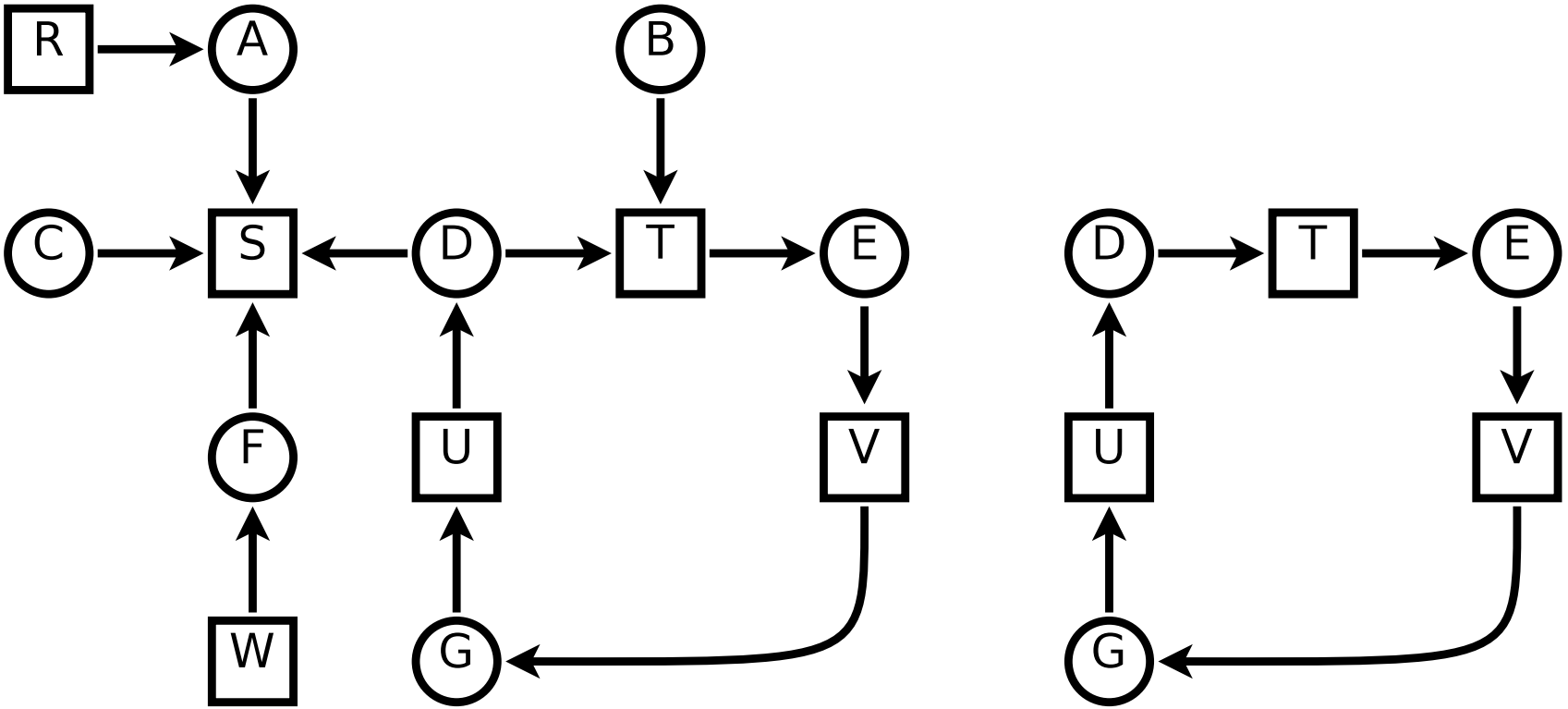
- Como parece que prevenir o evitar los interbloqueos no funciona demasiado bien, vamos a intentar detectarlos para poder deshacerlos.
- Una solución es modificar el algoritmo del banquero para relajar sus condiciones y hacerlo más eficiente:

  P[n][m]: matriz de solicitudes de recursos pendientes de asignar.

  `if(H[i]` ϵ `HI && P[i,*]` $\leq$ `D[*])...`

- En caso de interbloqueo HI contendrá la lista de hebras causantes.

Detección de interbloqueos (2)



(a) Estudie los recursos asignados y solicitados.

(b) Ciclo descubierto en el grafo de recursos $\longrightarrow$ interbloqueo.

Uso del algoritmo de detección de interbloqueos

¿Cuándo y con qué frecuencia utilizar el algoritmo de detección?

- En el caso extremo de que una solicitud de recurso **no** pueda ser **atendida**
  - ¿Qué coste tendría esto para el sistema?
  - ¿Cuánto incrementaría la complejidad del sistema?
  - ¿Cómo afecta a la temporización?
- **Periódicamente**, pero... ¿con qué periodo?
  - ¿Cuánto tiempo estamos dispuestos a esperar tras un interbloqueo para detectarlo?
  - ¿Cuántos recursos del sistema estamos dispuestos a utilizar en el proceso de detección?

Recuperación de un interbloqueo (1)

¿Qué hacer una vez detectado un interbloqueo? $\longrightarrow$ **finalizar procesos**

- Finalizar **todos** los procesos interbloqueados.
  - Es rápido pero desperdicia mucho trabajo.
- Finalizar procesos **de uno en uno** hasta que desaparezca el interbloqueo.
  - Este método consume más recursos.
  - Es mejor en cuanto a la cantidad de trabajo aprovechado.
  - ¿En qué orden finalizar los procesos?
- **Expropiar recursos.**
  - ¿De qué forma seleccionar el proceso víctima?
  - ¿Permite el sistema volver atrás y recuperarse a un proceso de una expropiación?
  - ¿Cómo protegernos de la inanición?

Recuperación de un interbloqueo (2)

- Recuperación mediante **expropiación**:
  - Tomar el recurso en conflicto de otro proceso.
  - Método aplicable sólo a recursos expropiables.
- Recuperación mediante **vuelta atrás**:
  - Almacenar el estado de los procesos periódicamente.
  - Reiniciar el proceso en uno de los estados previos al interbloqueo.
- Recuperación mediante **finalización**:
  - Método burdo pero eficaz.
  - Eliminar procesos del ciclo que condujo al interbloqueo hasta permitir a algún otro continuar.
  - Si disponemos de información elegir procesos reiniciables (compilador/base de datos).

Recuperación de un interbloqueo (3)

Criterios para elegir el proceso a finalizar:

- Tamaño de las peticiones pendientes.
- Cantidad de recursos asignados.
- Prioridad.
- Nivel del proceso: usuario/sistema.
- Tiempo de ejecución acumulado.
- Tiempo de ejecución restante.
- Escriba el que más le guste.

Ignorar los interbloqueos

El algoritmo del avestruz:

- Suponer que no existe el problema del interbloqueo.
- Razonable si...
  - los interbloqueos suceden muy raramente.
  - el coste de la prevención y la evitación es demasiado alto.
- Linux y Windows utilizan este método.
- Es un compromiso entre conveniencia y corrección.

Políticas combinadas

Los enfoques anteriores pueden combinarse:

- Agrupar los recursos en un cierto número de categorías y ordenarlas, ej:
  - área de intercambio
  - recursos de proceso: dispositivos de E/S, ficheros,...
  - memoria principal
- Utilizar prevención de cola circular para evitar interbloqueos entre estas categorías de recursos.
- Utilizar el método más apropiado contra interbloqueos dentro de cada categoría de recursos.