INTERBLOQUEOS

Rodrigo García Carmona
Universidad San Pablo-CEU
Escuela Politécnica Superior

OBJETIVOS

- Conocer **qué** es y **cómo** puede suceder el interbloqueo en los sistemas con múltiples procesos.
- Reconocer las **condiciones** que deben darse para que se produzca el interbloqueo.
- Comprender las principales **alternativas en el tratamiento** de interbloqueos.
  - Prevención.
  - Evasión (o predicción).
  - Detección.
- Conocer las **ventajas** **e** **inconvenientes** de cada uno de los tratamientos.
- Ver ejemplos de problemas clásicos.

CONTENIDO

- Concepto de interbloqueo
- Caracterización formal
  - Modelo de sistema
  - Condiciones de Coffman
  - Representación gráfica
- Técnicas de tratamiento de interbloqueos
  - Prevención
  - Evasión
  - Detección

Bibliografía

- W. Stallings: **Sistemas Operativos**.
  - Capítulo 6.
- A.S. Tanenbaum: **Modern Operating Systems**.
  - Capítulo 6.

CONCEPTO DE INTERBLOQUEO

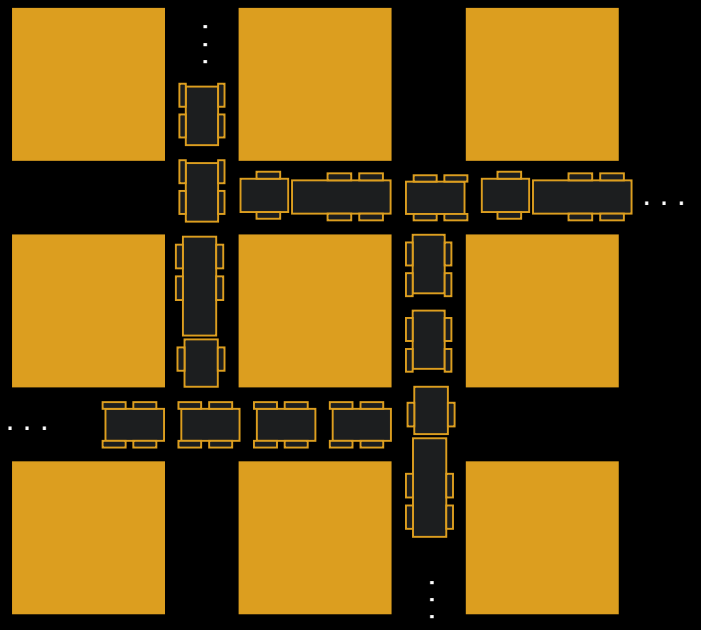
INTERBLOQUEO

- Dados:
  - Un conjunto de procesos ejecutándose en un sistema
  - Un conjunto de recursos que son utilizados por dichos procesos
- Se dice que el conjunto de procesos se encuentra en un estado de **interbloqueo** cuando todos los procesos se encuentran esperando un recurso que mantiene retenido otro proceso del grupo.
- En esa situación:
  - Ningún proceso del grupo puede evolucionar (suspendido eternamente).
  - Ningún proceso podrá obtener los recursos retenidos, puesto que no pueden ser liberados.
- Los interbloqueos constituyen un grave problema.

EJEMPLOS DE INTERBLOQUEO

- El sistema tiene dos unidades de cinta, P1 y P2 tienen cada uno una unidad y necesitan la otra.
- Semáforos SemA y SemB inicializados a 1.
- Situación de tráfico en interbloqueo.

Proceso 1	Proceso 2
P(SemA)	P(SemB)
P(SemB)	P(SemA)



CARACTERIZACIÓN FORMAL

MODELO DE SISTEMA

- Conjunto de **procesos**, identificados por P1, P2, ..., Pi, ..., Pn.
- Conjunto de **recursos**, identificados por R1, R2, ..., Rj, ..., Rm. Estos recursos pueden ser físicos (discos, impresoras, etc.), o lógicos (monitores, semáforos, etc).
  - De cada recurso puede haber una o más **instancias**. Dos recursos se consideran en realidad instancias del mismo recurso si un proceso que solicita dicho recurso considera que puede obtener cualquiera de ellas indistintamente.
- El uso que un proceso hace de un recurso sigue este protocolo:
  - **Petición:** Si el recurso no está disponible, el proceso queda suspendido hasta que lo esté.
  - **Uso:** El recurso es utilizado.
  - **Liberación:** El recurso queda disponible para otros procesos.

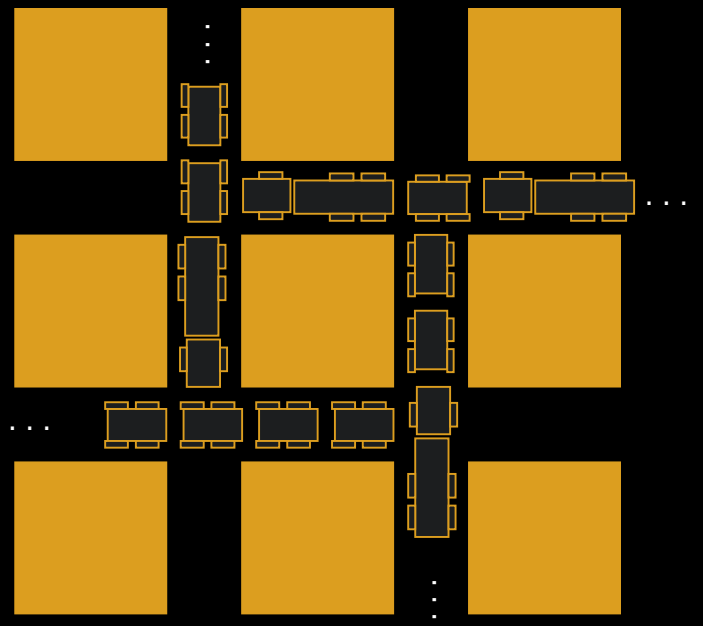
CONDICIONES DE COFFMAN

- Se ha demostrado que las siguientes cuatro condiciones son necesarias (aunque no suficientes) para que se produzca un interbloqueo:
  - **Exclusión mutua:** Al menos un recurso debe ser utilizado en exclusión mutua.
  - **Retener y esperar:** Debe haber al menos un proceso que retenga un recurso y que haya pedido algún otro recurso que posea otro proceso, por lo que estará esperando.
  - **No expropiación:** El sistema no puede arrebatar los recursos que ha asignado previamente a los procesos. Un proceso mantiene retenido un recurso hasta que deja de utilizarlo y lo libera voluntariamente.
  - **Espera circular:** Debe existir un conjunto de procesos {P1, P2, ..., Pi, ..., Pn} tal que P1 se encuentra esperando un recurso que retiene P2, P2 espera un recurso que retiene P3, ..., Pn-1 espera un recurso que mantiene Pn y Pn espera un recurso que mantiene P1.
- Si todas ellas se cumplen **simultáneamente**, el sistema se encuentra en situación de **riesgo de sufrir un interbloqueo**.
- La 4ª deriva de las 3 anteriores.

EJEMPLOS DE LAS CONDICIONES DE COFFMAN

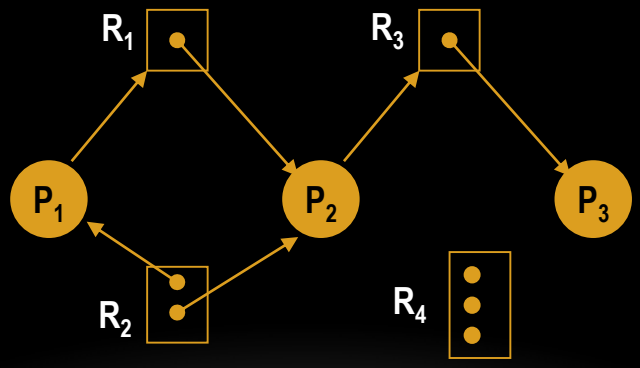
Cada sección de calle se considera un recurso.

1. **Exclusión mutua:** sólo un vehículo puede ocupar una sección de calle.
2. **Retener y esperar:** cada vehículo ocupa una sección de la calle y está esperando para moverse a la siguiente sección.
3. **No expropiación:** no se puede quitar una sección de la calle ocupada por un vehículo. El vehículo la liberará cuando se mueva a la siguiente sección.
4. **Espera circular:** cada vehículo está esperando a que se mueva el vehículo de delante.



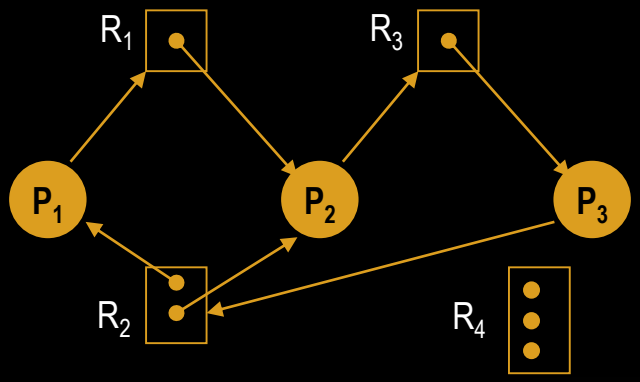
REPRESENTACIÓN GRÁFICA

- Grafo de asignación de recursos.
- Una asignación concreta de recursos puede ser representada de forma gráfica mediante un grafo en el que:
  - **Existen dos tipos de nodos:** círculos (procesos) y cuadrados (recursos). Los cuadrados poseen tantos puntos en el interior como instancias haya de dicho recurso.
  - **Los arcos son dirigidos:** Si van de un proceso a un recurso, indican petición, y si van de un recurso a un proceso, indican asignación. En este último caso, el arco va en realidad de una instancia concreta (punto) al proceso.



INTERPRETANDO EL GRAFO

- Interpretación del grafo:
  - Si no existen ciclos (dirigidos), entonces no hay interbloqueo.
  - Si existe algún ciclo dirigido, entonces:
    - Si hay sólo una instancia de cada recurso, entonces hay un interbloqueo.
    - Si hay más de una instancia, entonces puede haber un interbloqueo.
- En el ejemplo anterior ahora P3 solicita R2.

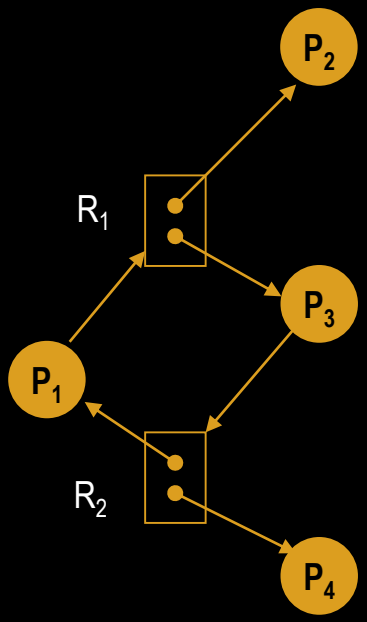


P_1 está esperando un recurso que está asignado a P_2, que espera un recurso que posee P_3, que espera un recurso que poseen P_1 y P_2.

¡INTERBLOQUEO!

OTRO EJEMPLO

- Grafo de asignación de recursos con un ciclo pero sin interbloqueo:



TÉCNICAS DE TRATAMIENTO DE INTERBLOQUEOS

TÉCNICAS PARA TRATAR CON INTERBLOQUEOS

- **Ignorar** el problema (*Ostrich algorithm*), asumiendo que dicha situación nunca se dará en el sistema. Es la aproximación que mantienen muchos sistemas operativos, incluido Unix.
- Emplear algún algoritmo o protocolo que asegure que nunca se va a poder producir un interbloqueo. Esta solución puede adoptar dos formas alternativas:
  - **Prevención:** consiste en conseguir que no puedan darse alguna de las cuatro condiciones de Coffman. De esta forma, el interbloqueo no puede llegar a producirse.
  - **Evasión:** consiste en llevar la cuenta de los recursos disponibles en el sistema, los recursos que poseen los procesos y los que pueden llegar a solicitar. Cada vez que se hace una petición de un recurso, el sistema analiza toda esa información para conceder (o denegar) dicho recurso.
- Utilizar un algoritmo que pueda detectar una situación de interbloqueo (**detección**) y seguir alguna técnica que permita deshacer dicha situación (recuperación).

PREVENCIÓN (I)

- **Exclusión mutua:** No es posible eliminar esta condición, pues la mayoría de recursos son inherentemente no compartibles.
  - ¿Qué hacer para imprimir? ...
- **Retener y esperar:** Para deshacer esta condición, se debe obligar a los procesos a :
  1. Utilizar los recursos de uno en uno, liberando cada recurso antes de solicitar el siguiente.
  2. Solicitar todos sus recursos de una vez, al principio de su ejecución.
  - Estas aproximaciones tienen dos problemas:
    - Baja utilización de los recursos, puesto que estarán retenidos desde el principio de la ejecución de los procesos, pero no se estarán utilizando en todo momento.
    - Inanición de los procesos que necesiten muchos recursos solicitados muy frecuentemente por los demás procesos.

PREVENCIÓN (II)

- **No expropiación:**
  - Para eliminar esta restricción se puede aplicar el siguiente algoritmo: Cuando un proceso P solicita recursos que no están libres, el sistema examina si los poseen procesos que a su vez están esperando. En ese caso, el sistema se apropia de dichos recursos y se los ofrece a P.
  - Los procesos a los que se les ha arrebatado recursos sólo podrán continuar cuando obtengan de nuevo dichos recursos, más los que estaban esperando.
  - Sólo funciona con recursos cuyo estado se pueda salvar y recuperar.
- **Espera circular:** Esta condición se puede romper si imponemos un orden total a los recursos, y se obliga a que los procesos soliciten siempre los recursos siguiendo dicho orden. Es decir:
  - Definimos la función f: R -> N, que asocia a cada recurso un número natural.
  - Un proceso que posee un recurso Ri puede solicitar otro recurso Rj si y sólo si f(Ri) < f(Rj).
  - De difícil implementación en sistemas con muchos recursos.
  - Bastante ineficiente.

EVASIÓN

- Las técnicas de prevención no son utilizables en algunas situaciones, presenta algunos inconvenientes y resulta confusa de utilizar.
- La evasión (también llamada predicción) permite las tres primeras condiciones de Coffman, pero controla que no se pueda producir la cuarta.
- Para este tipo de técnicas, el sistema debe conocer:
  - La cantidad **total** de recursos disponibles en el sistema.
  - La cantidad **actual** de recursos disponibles en el sistema.
  - La **necesidad máxima** de recursos de los procesos.
  - La **asignación actual** de recursos a procesos.
- Dos técnicas:
  - Denegación de inicio de proceso.
  - Denegación de asignación de recurso.
    - Concepto de estado seguro.
    - Algoritmo del banquero.

INFORMACIÓN SOBRE EL SISTEMA

- Suponemos:
  - n procesos.
  - m recursos.
- Estructuras de datos necesarias:
  - **Recursos[m]:** Máximo número de instancias de cada recurso.
  - **Disponible[m]:** Número de instancias disponibles de cada recurso.
  - **Asignados[n,m]:** Número de instancias de cada recurso actualmente asignadas a cada proceso.
  - **Necesarios[n,m]:** Número máximo de peticiones de instancias de cada recurso que cada proceso puede hacer.

DENEGACIÓN DE INICIO DE PROCESO Y DE ASIGNACIÓN DE RECURSO

- **Denegación de inicio de proceso:**
  - Evita el inicio de procesos.
  - Se impide que un proceso se inicie si los recursos máximos necesarios son superiores a los disponibles.
  - Incluso aunque…
    - No hubiera llegado a usarlos todos.
    - Pueda ir avanzando con los disponibles.
- **Denegación de asignación de recurso:**
  - Evita la asignación de recursos.
  - Se impide que un recurso se asigne si esto llevaría a un estado inseguro.
  - Se emplea el algoritmo del banquero.

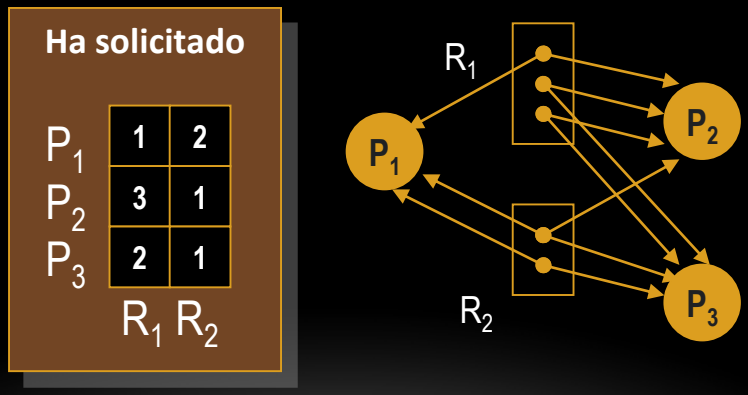
ESTADO SEGURO (I)

- **Estado seguro:** un estado (de asignación de recursos) se considera seguro si en él no hay posibilidad de interbloqueo.
- Para que un estado sea seguro, es necesario que exista al menos una secuencia segura.
- Una **secuencia segura** es una cierta ordenación de los procesos que asegure que se completa la ejecución de todos ellos.
  - Al menos uno puede ejecutarse, y al acabar liberará recursos, que utilizará otro proceso… y así sucesivamente.
  - Es una visión pesimista, asume que todos los procesos necesitarán de todos los recursos al mismo tiempo.
- Si dicha secuencia existe, como mucho cada proceso tendrá que esperar a que liberen sus recursos los procesos que van antes que él en la secuencia.

ESTADO SEGURO (II)

- Espacios seguro e inseguro, e interbloqueo:

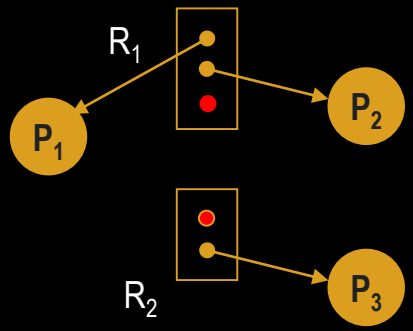
- Secuencia segura:
  - Es una cierta ordenación de los procesos que cumple que los recursos que aún puede pedir cualquiera de los Pi pueden ser satisfechos con los recursos libres más los recursos retenidos por los Pj, j<i.



ESTADO SEGURO (III)

Asignado	R_1	R_2
P_1	1	0
P_2	1	0
P_3	0	1

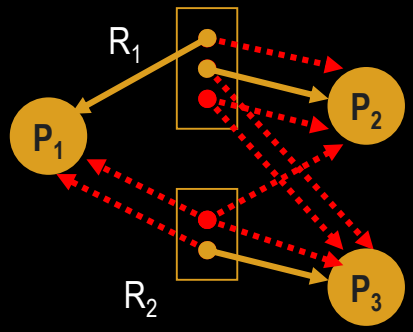
ESTADO DE ASIGNACIONES ACTUAL



No existe una Secuencia segura

ESTADO INSEGURO

Aún puede solicitar	R_1	R_2
P_1	0	2
P_2	2	1
P_3	2	1



Dependiendo de cuándo se soliciten los recursos y cuando se liberen, podrá darse interbloqueo o no.

ALGORITMO DEL BANQUERO

- Cuando un proceso realice una petición de recursos, el sistema se los concederá **sólo** en el caso de que la petición mantenga al sistema en un estado seguro.
- En dicha petición se comprueba que...
  1. El proceso no pida más recursos que los que originalmente necesitaba.
  2. Que el recurso pedido está disponible.
  3. Que tras la asignación el sistema queda en estado seguro.
     - ¿Hay algún proceso que pueda terminar su ejecución con los recursos restantes?
     - Tras acabar dicho proceso, y liberar por tanto sus recursos, ¿habría otro proceso que pudiera terminar su ejecución?
     - Repetir hasta que se encuentre una forma en la que todos los procesos puedan terminar su ejecución.

CÓDIGO DEL ALGORITMO DEL BANQUERO (I)

Estado del sistema

```
struct estado {
   int recursos[m];
   int disponibles[m];
   int necesidad[n][m];
   int asignacion[n][m];
}
```

Algoritmo de asignación

```
if (asignacion[i,*] + peticion[i,*] > necesidad[i,*])
   ERROR
else if (peticion[*] > disponibles[*])
   SUSPENDER
else {
   estado test;
   test->recursos = recursos;
   test->necesidad = necesidad;
   test->asignacion[i,*] = asignacion[i,*] + peticion[*];
   test->disponibles[*] = disponibles[*] – peticion[*];
}
if (seguro(test)) {
   ASIGNACIÓN
}
else
   SUSPENDER
```

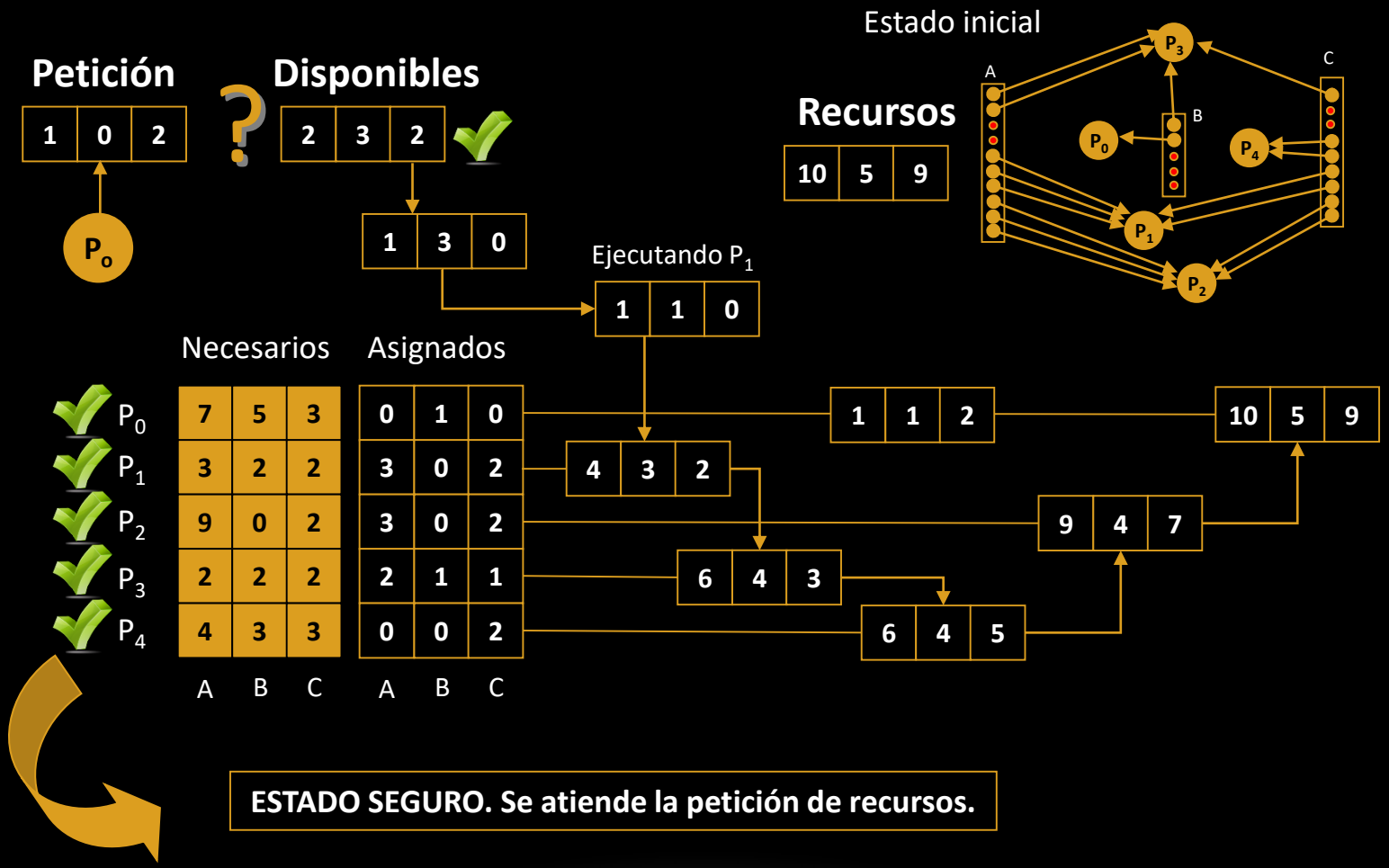
CÓDIGO DEL ALGORITMO DEL BANQUERO (II)

Algoritmo de comprobación de estado

```
boolean seguro (estado E) {
   int disponibles_actual[m];
   procesos resto;                          // Estructura con el resto de procesos
   disponibles_actual = disponibles;
   boolean posible = true;
   while (posible) {
      boolean encontrado = false;
      for (RECORRE RESTO) {
         Pi ES EL PROCESO ACTUAL EN EL RECORRIDO
         if (necesidad[i,*] – asignacion[i,*] <= disponibles_actual) {
            encontrado = true;
            break;
         }
      }
      if (encontrado) {
         disponibles_actual = disponibles_actual + asignacion[i,*];
         ELIMINA Pi DE RESTO
      }
      else
         posible = false;
   }
   if (RESTO ESTÁ VACÍO)
      return true;
   else
      return false;
}
```

EJEMPLO DE USO DEL ALGORITMO DEL BANQUERO (I)

EJEMPLO DE USO DEL ALGORITMO DEL BANQUERO (II)

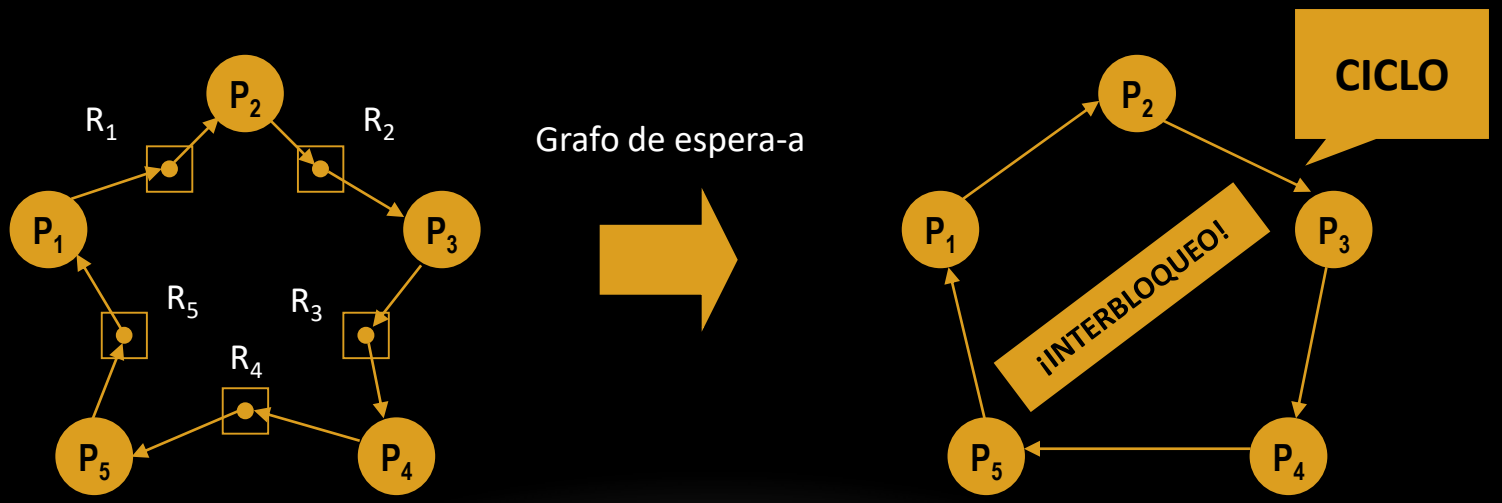


PROBLEMAS DE LA EVASIÓN

- Es necesario conocer información privilegiada con antelación.
- Los procesos deben ser independientes:
  - Si el orden de ejecución está condicionado a los requisitos de sincronización, el sistema no es libre de elegir una secuencia segura.
- El número de recursos e instancias es fijo e inalterable:
  - No se pueden añadir recursos nuevos…
  - …ni eliminarlos.

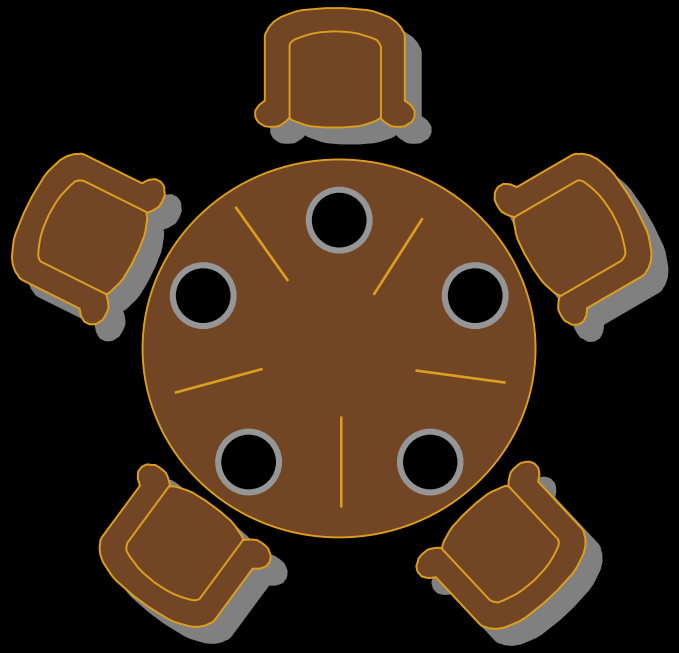
DETECCIÓN EN RECURSOS DE INSTANCIA ÚNICA

- Caso más sencillo.
- Grafo de “espera-a” (wait-for)
  - Se basa en el grafo de asignación de recursos.
  - Se eliminan los nodos correspondientes a recursos, y se ajustan los arcos de forma que habrá un arco del proceso P_i al proceso P_j si P_j posee un recurso que P_i ha solicitado.
  - Existirá interbloqueo si y sólo si hay un ciclo en el grafo.



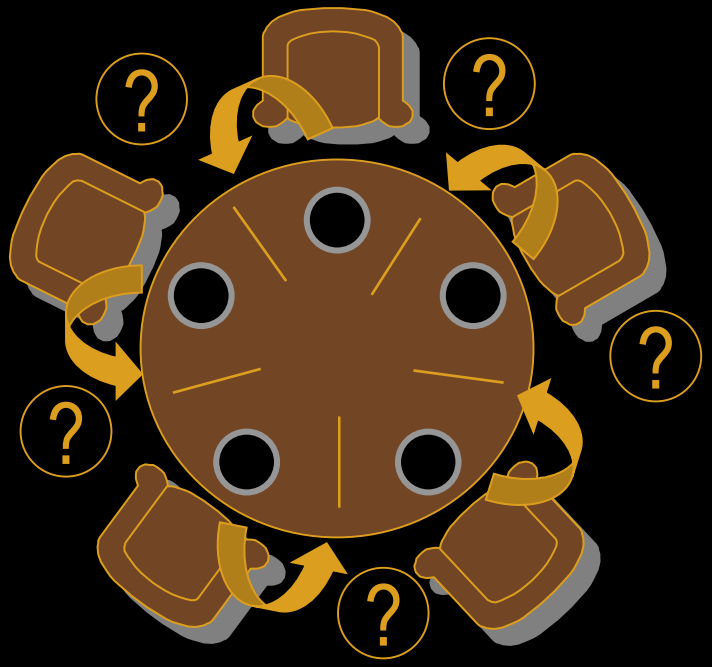
PROBLEMA DE LOS CINCO FILÓSOFOS (I)

- En una mesa hay cinco filósofos.
- Cada uno de ellos puede hacer únicamente dos cosas: quedarse quieto para pensar y comer.
- El protocolo que sigue cada uno de los filósofos para comer es el siguiente:
  1. Coger el palillo derecho de la mesa.
  2. Coger el palillo izquierdo.
  3. Comer.
  4. Dejar el palillo derecho en la mesa.
  5. Dejar el palillo izquierdo.
- Cada palillo es un recurso reutilizable serie. Hay únicamente cinco palillos para los cinco filósofos.



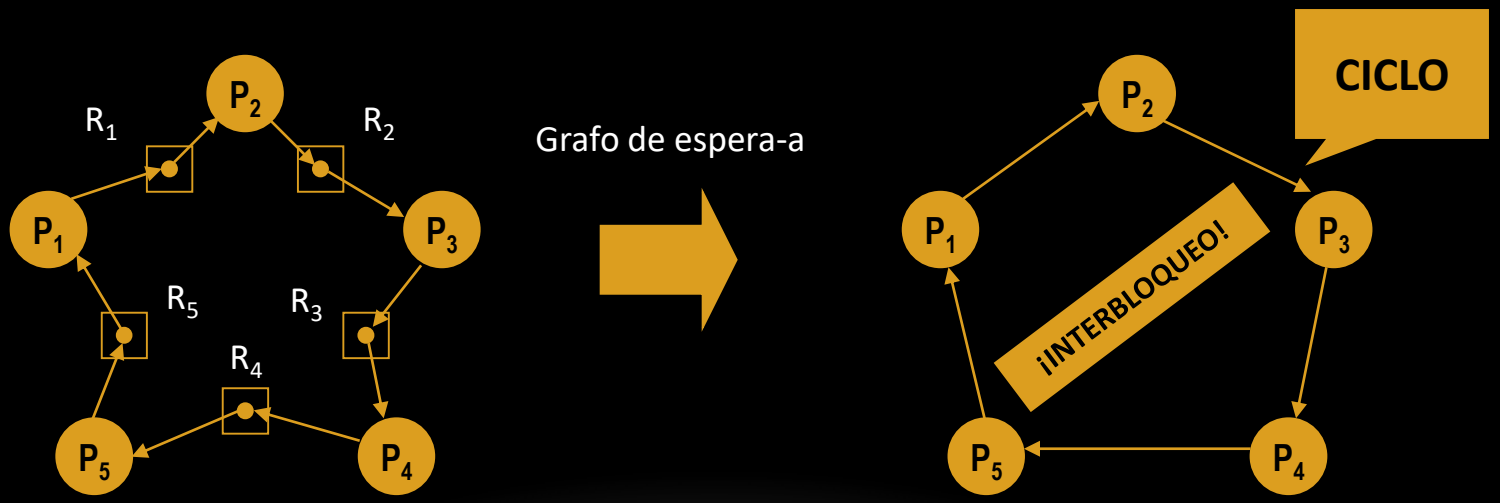
PROBLEMA DE LOS CINCO FILÓSOFOS (II)

- ¿Qué ocurre si cada uno coge el palillo derecho (i) e intenta coger el izquierdo (i+1)?
- Los cinco procesos han ejecutado P(palillo(i)) y se quedan esperando en P(palillo((i+1)%5)). Esta situación es de **interbloqueo**.
- Soluciones triviales:
  - Permitir que sólo se sienten a la mesa cuatro filósofos de forma simultánea.
  - Cada filósofo coge ambos palillos a la vez o no coge ninguno.
  - Modificar el protocolo del filósofo para comer: Los filósofos pares cogen primero el palillo derecho y los impares el izquierdo.



DETECCIÓN EN EL PROBLEMA DE LOS CINCO FILÓSOFOS

- Caracterización formal: Grafo de asignación de recursos.
  - Existen cinco procesos (filósofos)
  - Además, cinco recursos (palillos) de instancia única:
    - Los recursos no son todos iguales, puesto que cada proceso solicita un palillo en concreto (el de su derecha o el de su izquierda). No podemos considerar un único recurso con múltiples instancias, puesto que cada proceso no puede aceptar cualquier instancia.



DETECCIÓN EN RECURSOS DE MÚLTIPLES INSTANCIAS

- Caso más complejo. Se usa un algoritmo de detección.
- Estructuras de datos necesarias:
  - **Recursos[m]:** Máximo número de instancias de cada recurso.
  - **Disponible[sm]:** Número de instancias disponibles de cada recurso.
  - **Asignados[n,m]:** Número de instancias de cada recurso actualmente asignadas a cada proceso.
  - **Solicitudes[n,m]:** Número de peticiones de instancias de cada recurso que cada proceso ha hecho.
- Cuándo se invoca este algoritmo. Alternativas:
  - **Cada vez que una petición de recursos no puede ser concedida inmediatamente:** Produce más sobrecarga pero es más fácil de detectar qué procesos provocan el interbloqueo puesto que como mucho habrá un único interbloqueo. Consume más tiempo de procesador.
  - **Cada cierto intervalo de tiempo:** Esta alternativa permite medir la sobrecarga, pero si existen varios ciclos será más difícil determinar qué procesos están involucrados en cada uno de los interbloqueos. Consume menos tiempos de procesador.

ALGORITMO DE DETECCIÓN DE INTERBLOQUEOS DE RECURSOS DE MÚLTIPLES INSTANCIAS

Estado del sistema

```
struct estado {
   int recursos[m];
   int disponibles[m];
   int asignados[n][m];
   int solicitudes[n][m];
}
```

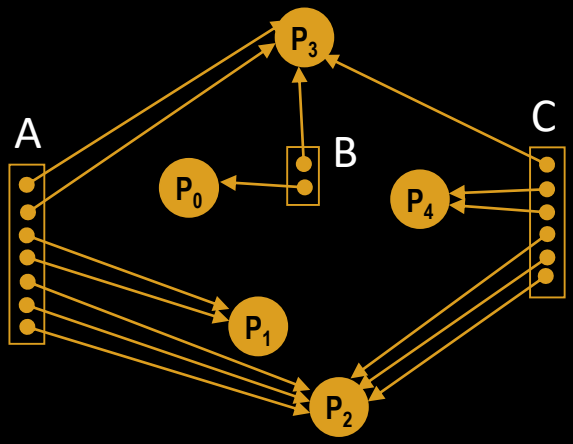
Algoritmo de detección

```
boolean deteccion() {
   int no_bloqueado[n];
   int trabajo[m] = disponible[m];
   int i;
   for (i=0; i<n, i++) {                  // Sin nada asignado no
      if (asignados[i]!=0)                // pueden estar bloqueados
         no_bloqueado[i] = false;
      else
         no_bloqueado [i] = true;
   }
   boolean encontrado = true;
   while(encontrado) {      // Repetir si nuevos procesos no bloqueados
      encontrado = false;
      for (i=0; i<n, i++) {               // Para cada proceso
         int j;
         boolean haPedidoMas = false;          // Si no ha pedido
         for (j=0; j<m, j++) {                 // más recursos de los
            if (solicitudes[i,j] > trabajo[j]) // disponibles
               haPedidoMas = true;
         }
         if (no_bloqueado[i]==false && haPedidoMas==false) {
            trabajo = trabajo + asignados[i];
            no_bloqueado [i] = true;           // No está bloqueado
            encontrado = true;
      }
   }
   for (i=0; i<n, i++) {
      if (no_bloqueado[i] == false)
         return true;
   }
   return false;
}
```

Aquellos procesos para los que no_bloqueado[i] sea falso forman parte de un interbloqueo

EJEMPLO DEL ALGORITMO

- ¿Existe interbloqueo en los siguientes dos casos?:



Recursos

7	2	6

Disponibles

0	0	0

Asignados

	A	B	C
P_0	1	0	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

Solicitudes 1

	A	B	C
P_0	0	0	0
P_1	2	0	2
P_2	0	0	0
P_3	1	0	0
P_4	0	0	2

Solicitudes 2

	A	B	C
P_0	0	0	0
P_1	2	0	2
P_2	0	0	2
P_3	1	0	0
P_4	0	0	2

RECUPERACIÓN DE INTERBLOQUEOS

- Alternativas:
  - **Terminación de procesos:**
    - Terminación de todos los procesos interbloqueados.
    - Terminación iterativa de procesos, hasta que el interbloqueo desaparece.
  - **Expropiación de recursos:** El sistema expropia de recursos de los procesos interbloqueados, hasta que desaparece el interbloqueo.
- Problemas a resolver:
  - **Selección de una víctima:** A quién se elige para apropiarse de sus recursos.
  - **Vuelta atrás:** El proceso al que se le quitan los recursos ha de ser devuelto a un estado seguro. La solución trivial es abortar dicho proceso.
  - **Inanición:** Hay que considerar que no se debería quitar siempre los recursos al mismo proceso, sobre todo si la vuelta atrás supone abortarlo y obligarle a empezar desde el principio.

RESUMEN DE TÉCNICAS PARA TRATAR INTERBLOQUEOS

Técnica	Política de asignación de recursos	Esquemas	Ventajas principales	Desventajas principales
Prevención	Conservadora: los recursos están poco ocupados.	Solicitud de todos los recursos a la vez.	Funciona bien con procesos que realizan una sola ráfaga de actividad. No es necesaria la expropiación.	Ineficiencia Retrasos en el inicio de los procesos Se deben conocer las necesidades futuras de recursos.
		Expropiación.	Conveniente cuando se aplica a recursos cuyo estado se puede salvar y restaurar fácilmente.	Expulsa más habitualmente de lo necesario.
		Ordenación de recursos.	Aplicación factible por medio de chequeos durante la compilación. No hace falta procesamiento durante la ejecución, ya que el problema se resuelve durante el diseño del sistema.	No permite las solicitudes incrementales de recursos.
Evasión (Predicción)	A medio camino entre la detección y la prevención.	Manipulación para encontrar al menos un camino seguro.	No es necesaria la expropiación.	Deben conocerse las demandas futuras de recursos. Los procesos pueden bloquearse durante largos periodos.
Detección	Muy liberal; los recursos solicitados se conceden cuando es posible.	Comprobación periódica del interbloqueo.	Nunca retrasa el inicio de un proceso. De fácil adaptación al entorno.	Pérdidas inherentes a la expropiación.