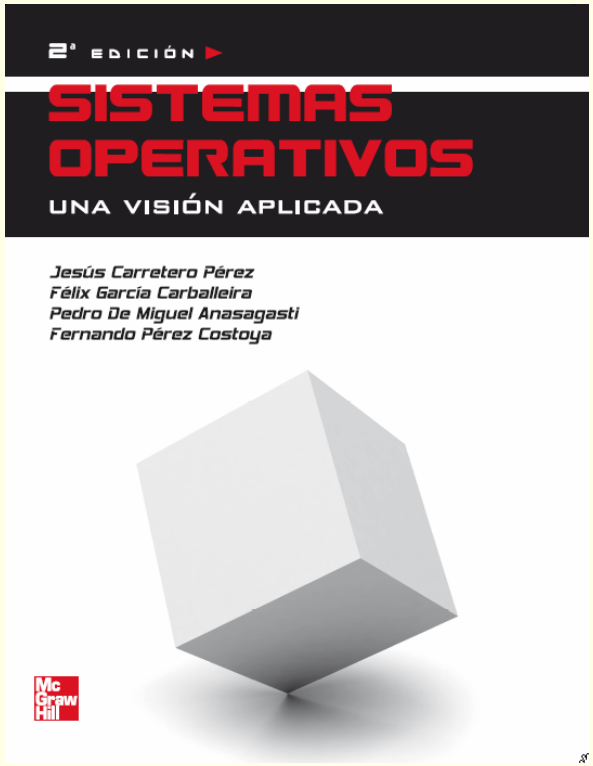
Sistemas operativos
2ª edición



Capítulo 7
Interbloqueos
(extracto de las transparencias del libro)

Contenido

- Introducción
- Modelo general del sistema
- Definición y tratamiento del interbloqueo
- Detección y recuperación del interbloqueo
- Prevención del interbloqueo
- Predicción del interbloqueo
- Tratamiento del interbloqueo en los sistemas operativos

Introducción

- Procesos compiten por recursos y se comunican
- Conflicto puede causar bloqueo indefinido

  $\Rightarrow$ **Interbloqueo** (*deadlock*)
- Conocido y estudiado desde hace mucho tiempo
  - Poca repercusión práctica en SSOO
- Aparece en otras disciplinas informáticas

Tipos de recursos

- Físicos vs Lógicos:
  - UCPs, memoria, dispositivos. etc.
  - archivos, semáforos, mutex, cerrojos, mensajes, señales, etc.
- Reutilizables o consumibles
  - ¿Sigue existiendo después de usarse?
- Uso dedicado o compartido
  - ¿Pueden usarlo varios procesos simultáneamente?
- Con uno o múltiples ejemplares
  - ¿Existen múltiples ejemplares del mismo recurso?
- Expropiables o no expropiables
  - ¿Es factible expropiar el recurso cuando se está usando?

Recursos reutilizables

- Todos los recursos físicos
- Algunos lógicos (archivos, cerrojos, mutex, ...)

Primer ejemplo de interbloqueo

Proceso P_1	Proceso P_2
Solicita(C)	Solicita(I)
Solicita(I)	Solicita(C)
Uso de rec.	Uso de rec.
Libera(I)	Libera(C)
Libera(C)	Libera(I)

Ejecución con interbloqueo

P_1: solicita(C)
P_2: solicita(I)
P_2: solicita(C) $\rightarrow$ bloqueo
P_1: solicita(I) $\rightarrow$ interbloqueo

Ejecución sin interbloqueo

P_1: solicita(C)
P_1: solicita(I)
P_2: solicita(I) $\rightarrow$ bloqueo
P_1: libera(I)
P_2: solicita(C) $\rightarrow$ bloqueo
P_1: libera(C)
P_2: libera(C)
P_2: libera(I)

Recursos consumibles

- Proceso genera recurso y otro lo consume
- Recursos asociados a comunicación y sincronización
  - mensajes, señales, semáforos, ...
- Ejemplo: Interbloqueo inevitable ("estructural")

Proceso P_1	Proceso P_2	Proceso P_3
Enviar(P_3)	Recibir(P_1)	Recibir(P_2)
Recibir(P_3)	Enviar(P_3)	Enviar(P_1)
Enviar(P_2)		Recibir(P_1)

Recursos reutilizables y consumibles

- En general, procesos usan ambos tipos de recursos
  - No hay solución general eficiente
- Exposición se centra en recursos reutilizables
- Ejemplo mixto:

Proceso P_1	Proceso P_2
Solicita(C)	Solicita(C)
Enviar(P_2)	Recibir(P_1)
Libera(C)	Libera(C)

Si P_2 obtiene C $\rightarrow$ interbloqueo

Modelo del sistema

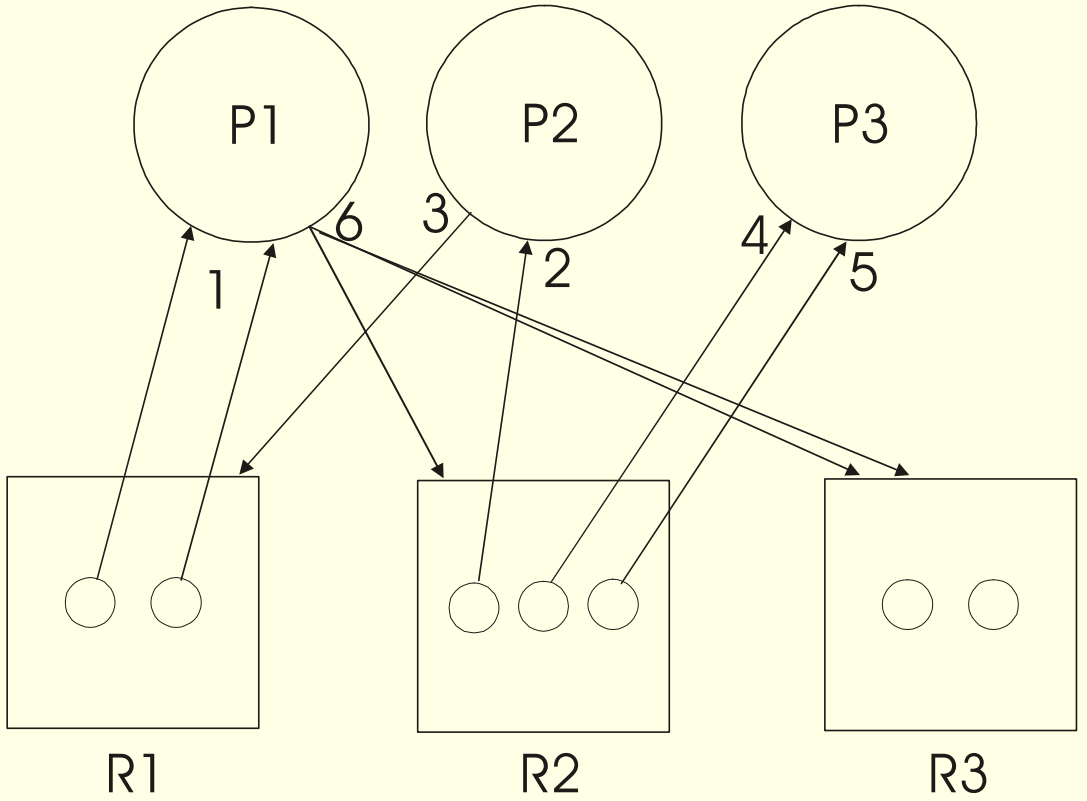
- Conjunto de procesos o *threads*
- Conjunto de recursos de uso exclusivo (N unidades/recurso)
- Relaciones entre procesos y recursos:
  - Asignación: nº unidades asignadas a cada proceso
  - Pendientes: nº unidades pedidas pero no asignadas
- Primitivas genéricas:
  - *Solicitud* ($R_1[U_1]$,...,$R_n[U_n]$)
  - *Liberación* ($R_1[U_1]$,...,$R_n[U_n]$)
- Carácter dinámico del sistema:
  - Procesos y recursos aparecen y desaparecen
- Representación mediante grafo o matriz

Ejemplo 1 de representación con grafo

3 proc y 3 rec

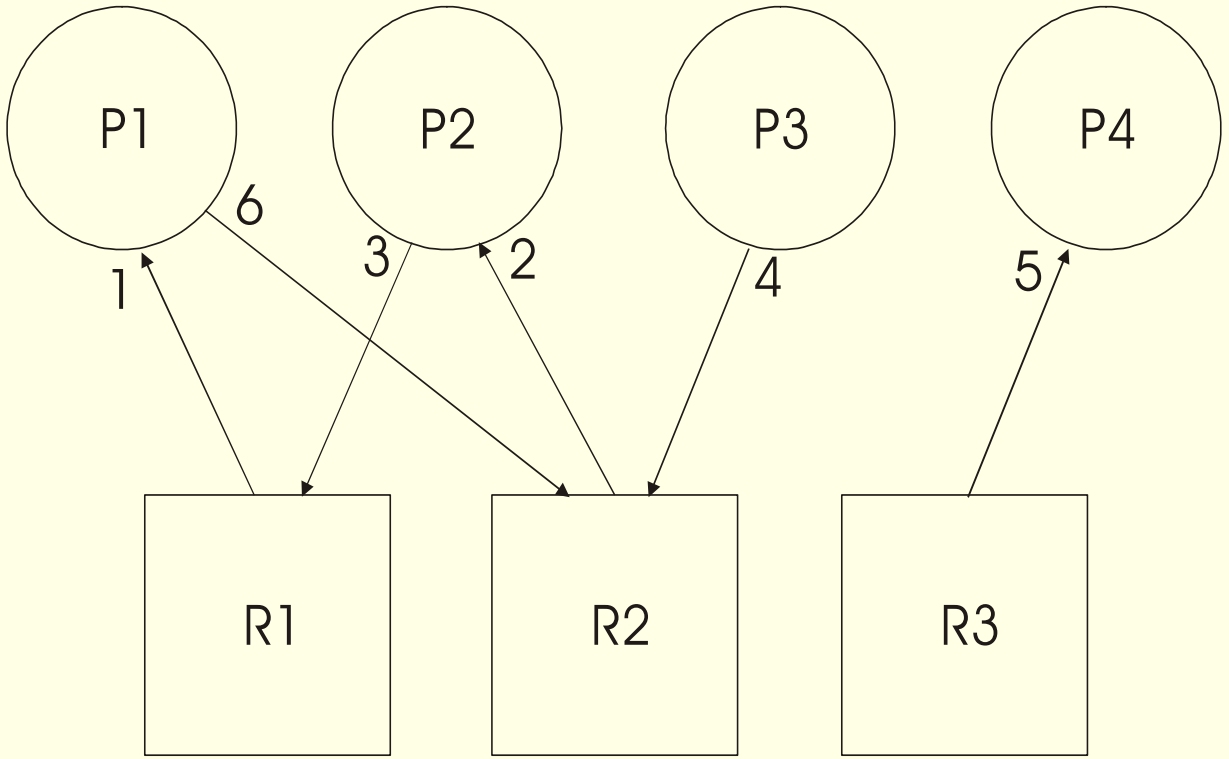
$R_1(2), R_2(3), R_3(2)$

1. P_1: solicita($R_1[2]$) → solicita 2
2. P_2: solicita($R_2[1]$)
3. P_2: solicita($R_1[1]$) → bloq
4. P_3: solicita($R_2[1]$)
5. P_3: solicita($R_2[1]$)
6. P_1: solicita($R_2[1], R_3[2]$) → bloq

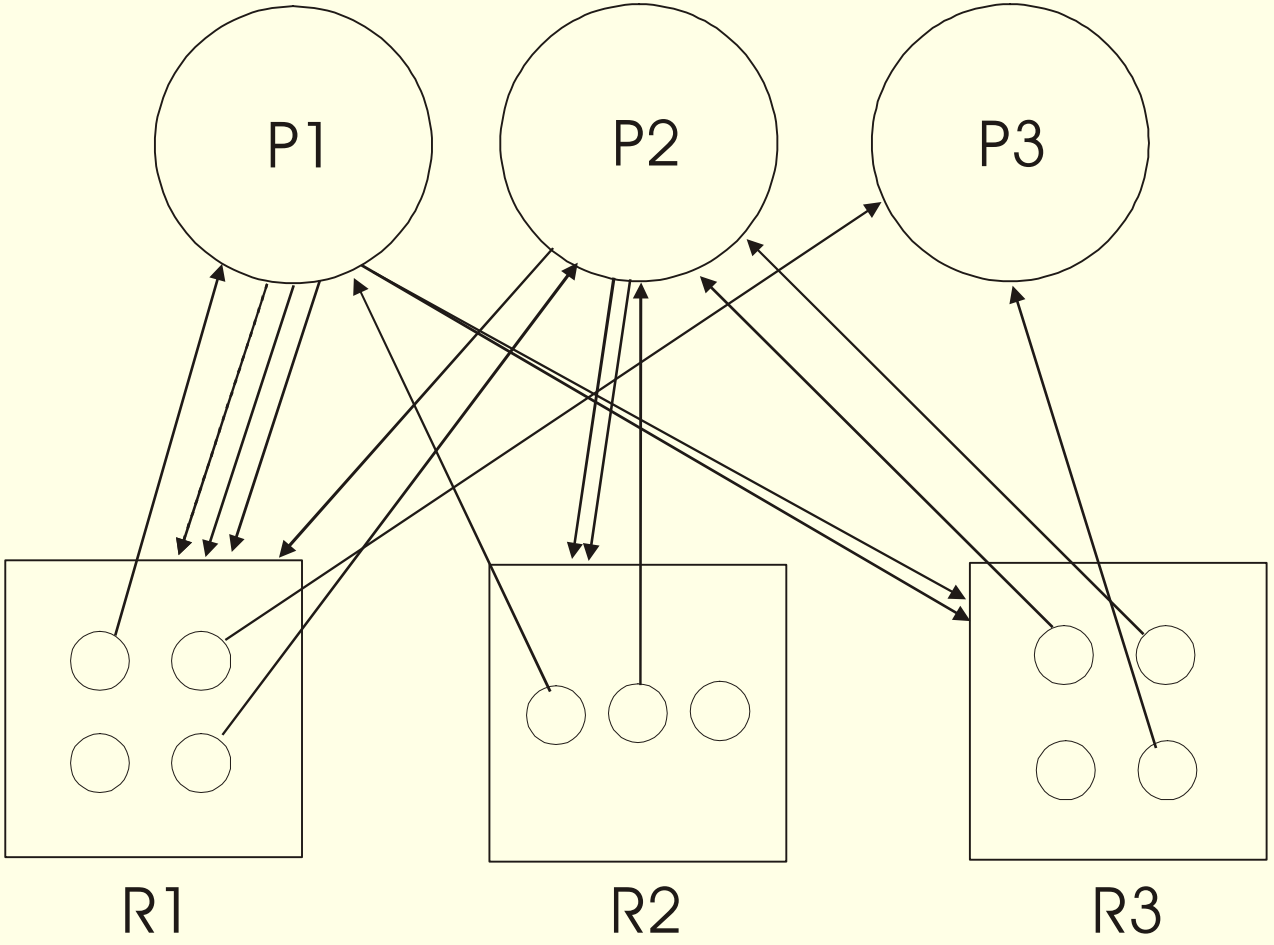


Ejemplo 2 de representación con grafo (1 unid/rec)

1. P_1: solicita(R_1)
2. P_2: solicita(R_2)
3. P_2: solicita(R_1) $\rightarrow$ bl.
4. P_3: solicita(R_2) $\rightarrow$ bl.
5. P_4: solicita(R_3)
6. P_1: solicita(R_2)..



Ejemplo 3 de representación con grafo



Definición y tratamiento de interbloqueo

- Conjunto de procesos tal que cada uno está esperando un recurso que sólo puede liberar otro proceso del conjunto
- Tratamiento del interbloqueo:
  - Detección y recuperación. Se detecta y se recupera
    - Coste de algoritmo + pérdida del trabajo realizado
  - Prevención. Asegura que no ocurre fijando reglas
    - Infrautilización de rec.: se deben pedir antes de necesitarlos
  - Predicción. Asegura que no ocurre basándose en conocimiento de necesidades futuras de los procesos
    - Dificultad de conocer futuro
    - Coste de algoritmo + Infrautilización de recursos
  - Ignorar el problema
    - Utilizada por la mayoría de los SS.OO.

Detección del interbloqueo

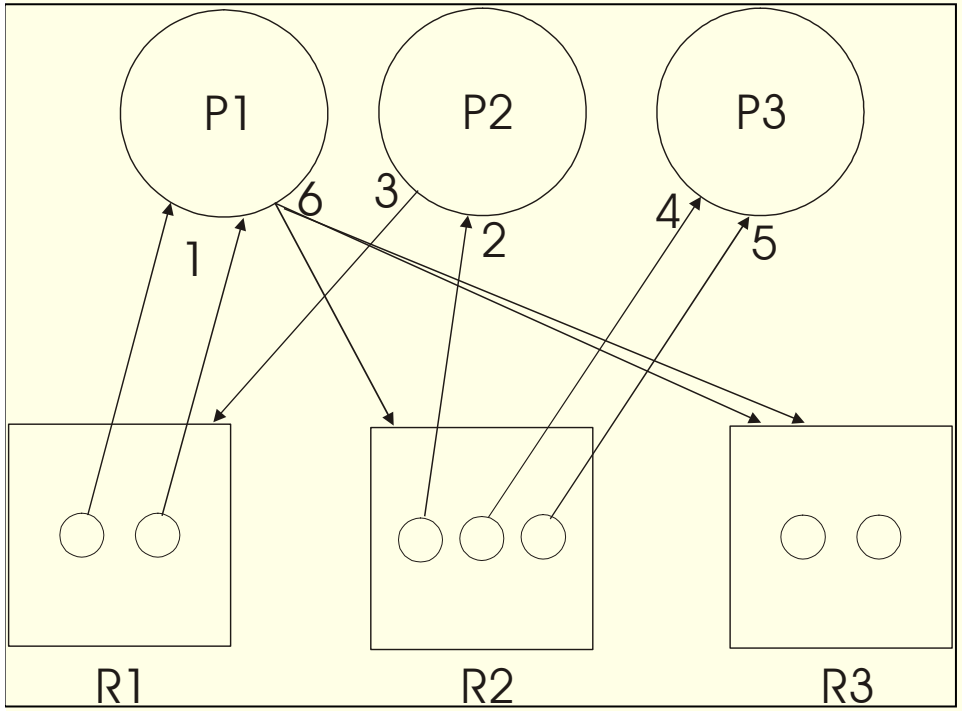
- Algoritmo que comprueba si se cumplen condiciones de interbl.
- Condiciones necesarias pero no suficientes (Coffman):
  - **Exclusión mutua**.
  - **Retención y espera**.
  - **Sin expropiación**.
  - **Espera circular**.
- Si restricciones (1 unid/rec), cond. son necesarias y suficientes
  - Ejemplo 2: Ciclo en grafo → espera circular → interbloqueo
  - Ejemplo 1: Ciclo en grafo → espera circular → No interbloq.
  - Ejemplo 3: Ciclo en grafo → espera circular → interbloq.
    - Se necesita condición necesaria y suficiente para sist. general

Condición necesaria y suficiente

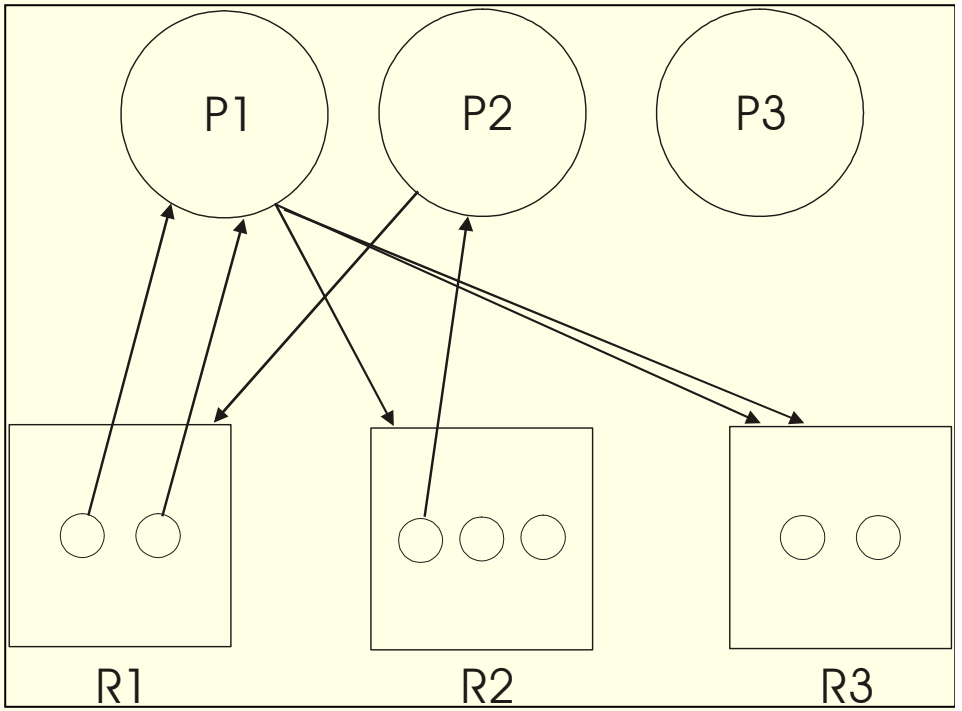
- Proceso no bloqueado debería devolver recursos en el futuro
  - Recursos liberados desbloquearían otros procesos...
- Reducción (*Red*) del sistema por proceso P
  - Si se pueden satisfacer necesidades de P con r. disponibles
  - Nuevo estado hipotético donde P ha liberado todos sus rec.
- Condición necesaria y suficiente:
  - $\exists$ secuencia de *Red* desde estado actual $\subset$ todos los procesos
  - Si no: procesos $\not\subset$ están en interbloqueo

Alg. de detección para Ejemplo 1 (1/2)

Estado inicial



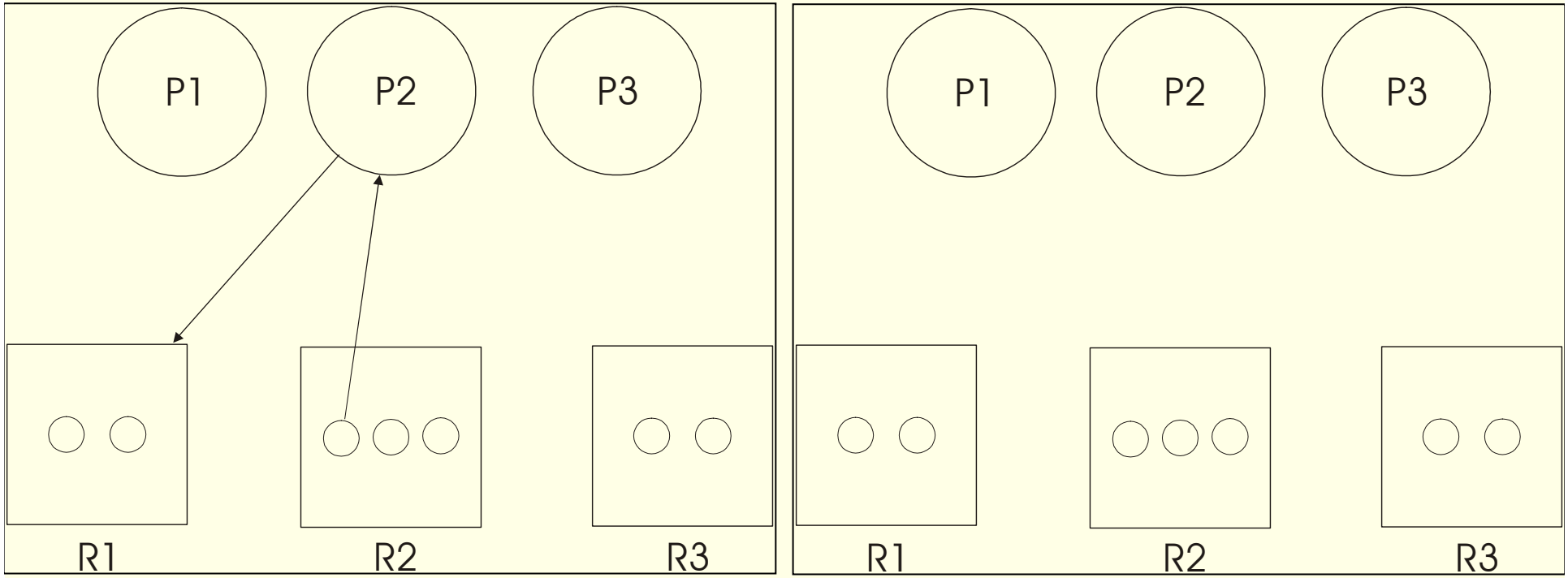
Reducción por P3



Alg. de detección para Ejemplo 1 (2/2)

Reducción por P1

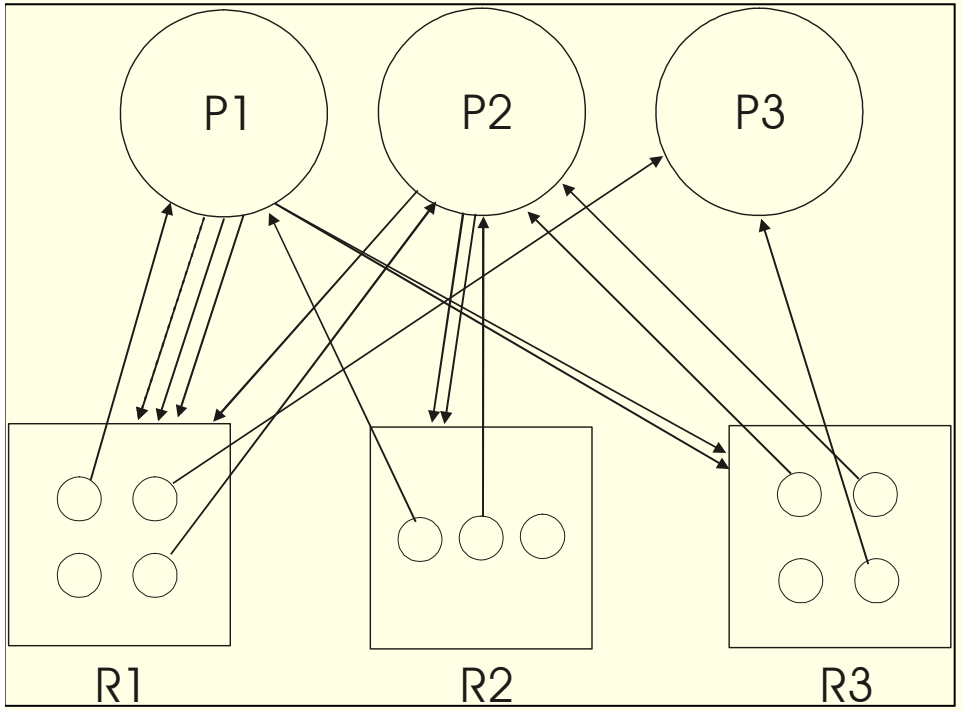
Reducción por P2



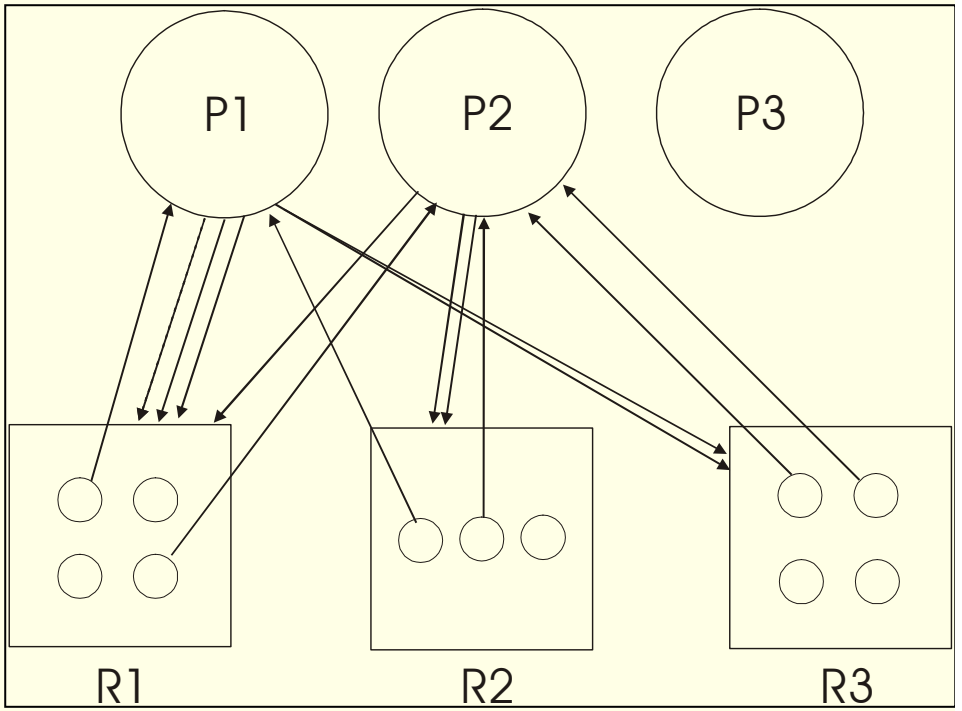
No hay interbloqueo

Alg. de detección para Ejemplo 3

Estado inicial



Reducción por P3



No más reducciones: Interbloqueo

Activación del algoritmo de detección

- Supervisión continua:
  - Por cada petición que no puede satisfacerse
  - Puede tener coste demasiado alto
- Supervisión periódica:
  - Guiada por tiempo y/o por detección de síntomas

Recuperación del interbloqueo

- Quitar recursos a procesos hasta eliminar interbloqueo
- Alternativas:
  - "Retroceder en el tiempo": Requiere puntos de recuperación
  - Abortar proceso perdiendo todo su trabajo realizado
- Selección de procesos basada en:
  - prioridad, nº de recursos asignados al proc., t. de ejecución,...
- Estrategia adecuada para bases de datos

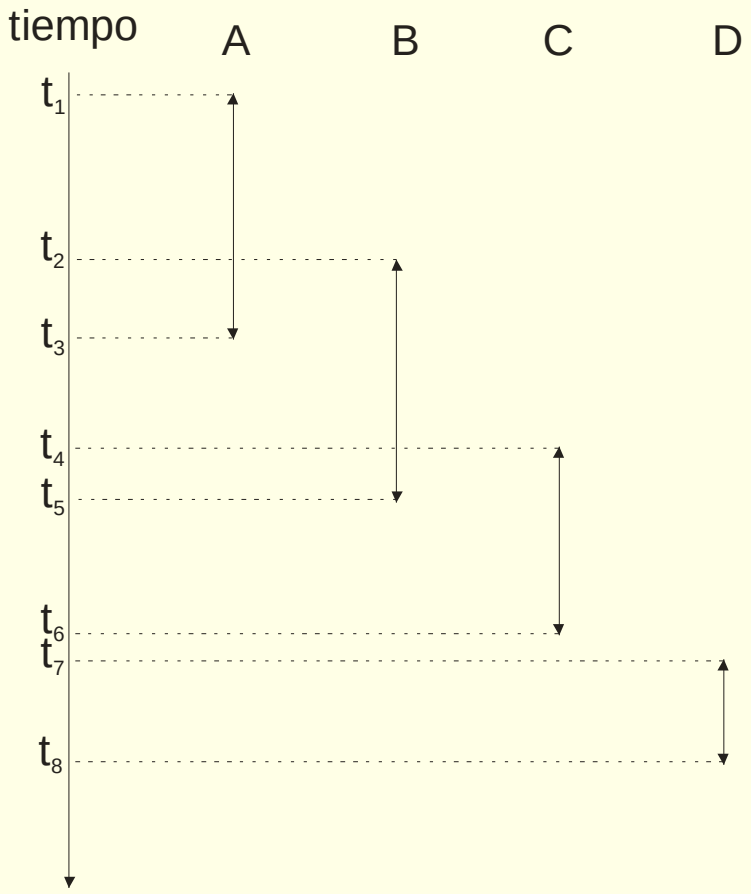
Prevención del interbloqueo

- Que no se cumpla una condición necesaria
- "Exclusión mutua" y "sin expropiación" no se pueden relajar
  - Dependen de carácter intrínseco del recurso
- Las otras dos condiciones son más prometedoras

Prevención evitando "retención y espera"

- Sólo se puede pedir si no se tiene
  - Infrautilización

t_1: solicita(A,B,C)
(t_1,t_2): sólo utiliza A
(t_2,t_3): utiliza A y B
t_3: Libera(A)
(t_3,t_4): sólo utiliza B
(t_4,t_5): utiliza B y C
t_5: Libera(B)
(t_5,t_6): sólo utiliza C
t_6: Libera(C)
t_7: solicita(D)
(t_7,t_8): sólo utiliza D
t_8: Libera(D)



Prevención evitando "espera circular"

- Método de las peticiones ordenadas:
  - Establece orden de recursos del sistema
    - Según forma de uso más frecuente
  - Restricción: Proceso sólo puede pedir en orden
- Conlleva infrautilización:
  - Si A<B<C<D $\rightarrow$ Proceso pide justo cuando necesita
  - Si A>B>C>D $\rightarrow$ Proceso pide todo en t_1

Predicción del interbloqueo

- Punto sin retorno: P1 y P2 obtienen su primer recurso
- No concede 1 de esas peticiones → sistema en "estado seguro"

Proceso P_1	Proceso P_2
Solicita(C)	Solicita(I)
Solicita(I)	Solicita(C)
Uso de rec.	Uso de rec.
Libera(I)	Libera(C)
Libera(C)	Libera(I)

Concepto de estado seguro

- "Aunque procesos pidiesen de golpe necesidades máximas hay un orden de ejecución tal que cada proceso pueda obtenerlas"
- Similar a detección pero con necesidades máximas
- Estado seguro:
  - No interbl. usando como solicitudes las necesidades máx.
- Conocimiento a priori no da información sobre uso real

Estado inseguro → condición necesaria pero no suficiente

Proceso P_1	Proceso P_2
Solicita(C)	Solicita(I)
Libera(C)	Solicita(C)
Solicita(I)	Libera(C)
Libera(I)	Libera(I)

Algoritmo de predicción

- Algoritmo del banquero de Dijkstra (1965)
- Estructuras de datos requeridas:
  - Vector D (dim p): D_i cuántas unid de R_i hay disponibles
  - Matriz A (dim pxr): A[i,j] unid de R_j asignadas a P_i
  - Matriz N (dim pxr): N[i,j] unid adicionales de R_j puede necesitar P_i
    - Es la diferencia entre necesidades máx. y unidades asignadas
    - Inicialmente contiene necesidades máx. de cada proceso
- Solicitud de recursos de Pi: ¿Todos disponibles?
  - Sí. Por cada Rj:
    - A[i,j]= A[i,j]+ Uj y N[i,j]= N[i,j]- Uj (Uj unidades solicitadas de R_j)
- Liberación de recursos de Pi:
  - Por cada Rj:
    - A[i,j]= A[i,j]- Uj y N[i,j]= N[i,j]+ Uj (Uj unidades liberadas de R_j)

Algoritmo del banquero

```
S=∅;
Repetir {
    Buscar Pi ∉ en S tal que N[i]≤D;
    Si Encontrado {
        Reducir por Pi: D = D + A[i]
        Añadir Pi a S;
    }
} Mientras (Encontrado)
Si (S==P) El estado es seguro
Si no: El estado no es seguro
```

Estrategia de predicción

- Proceso realiza petición de recursos disponibles:
  - Nuevo estado provisional transformando A y N
  - Algoritmo del banquero sobre nuevo estado
  - Si seguro, se asignan recursos
  - Si no, se bloquea al proceso sin asignarle los recursos

Ejemplo de algoritmo del banquero (1/3)

- Estado actual del sistema (es seguro):

$$A=\begin{bmatrix}1 & 1 & 0\\ 0 & 1 & 2\\ 1 & 0 & 0\end{bmatrix} \quad N=\begin{bmatrix}3 & 0 & 2\\ 2 & 2 & 0\\ 1 & 1 & 2\end{bmatrix} \quad D=\begin{bmatrix}2 & 1 & 2\end{bmatrix}$$

- Secuencia de peticiones:
  - P_3 solicita 1 unidad de R_3
  - P_2 solicita 1 unidad de R_1

Ejemplo de algoritmo del banquero (2/3)

- P_3 solicita 1 unidad de R_3: Nuevo estado provisional

$$A=\begin{bmatrix}1 & 1 & 0\\ 0 & 1 & 2\\ 1 & 0 & 1\end{bmatrix} \quad N=\begin{bmatrix}3 & 0 & 2\\ 2 & 2 & 0\\ 1 & 1 & 1\end{bmatrix} \quad D=\begin{bmatrix}2 & 1 & 1\end{bmatrix}$$

- ¿Estado seguro?

1. $S=\varnothing$
2. P_3: ya que $N[3]\leq D \rightarrow D=D+A[3]=[312] \rightarrow S=\{P_3\}$
3. P_1: ya que $N[1]\leq D \rightarrow D=D+A[1]=[422] \rightarrow S=\{P_3, P_1\}$
4. P_2: pues $N[2]\leq D \rightarrow D=D+A[2]=[434] \rightarrow S=\{P_3, P_1, P_2\}$

Se acepta petición: estado provisional → estado del sistema

Ejemplo de algoritmo del banquero (3/3)

- P_2 solicita 1 unidad de R_1: Nuevo estado provisional

$$A=\begin{bmatrix}1 & 1 & 0\\ 1 & 1 & 2\\ 1 & 0 & 1\end{bmatrix} \quad N=\begin{bmatrix}3 & 0 & 2\\ 1 & 2 & 0\\ 1 & 1 & 1\end{bmatrix} \quad D=\boxed{\begin{matrix}1 & 1 & 1\end{matrix}}$$

- ¿Estado seguro?

1. $S=\varnothing$
2. P_3: ya que $N[3] \leq D \rightarrow D=D+A[3]=[212] \rightarrow S=\{P_3\}$
3. No hay Pi tal que $N[i] \leq D \rightarrow$ Estado inseguro

No se acepta petición:

Se bloquea al proceso y se restaura estado del sistema

Limitaciones de estrategias de predicción

- Conocimiento a priori de necesidades máximas
  - Difícil de obtener
  - Basado en peor caso posible
- Necesidades máximas no expresan uso real de recursos
- Infrautilización de recursos
  - Niegan uso de recurso aunque esté libre

Tratamiento del interbloqueo en los SSOO

- Mayoría lo ignora o no da solución general
- Distinción entre dos tipos de recursos:
  - Recursos internos (propios del SO)
    - Uso restringido a una activación del sistema operativo
    - P. ej. *spinlocks* y semáforos internos
    - Interbloqueo puede causar colapso del sistema
  - Recursos de usuario (usados en modo usuario)
    - Uso durante tiempo impredecible
    - P. ej. dispositivo dedicado o semáforo de aplicación
    - Interbloqueo afecta a procesos y recursos involucrados

Tratamiento del interbloqueo en los SSOO

- Tratamiento de recursos internos
  - Código del SO apenas se modifica
    - Interbloqueo → error de programación de SO
  - Uso de estrategias de prevención es adecuado
    - Tiempo de uso es breve y acotado
    - Solución habitual: ordenamiento de recursos
- Tratamiento de recursos de usuario
  - Código de procesos que usan recursos es impredecible
  - No hay tratamiento general para todos los recursos
    - Prevención → Infrautilización
    - Predicción → Dificultad de conocer información a priori
    - Detección y recuperación → Demasiada sobrecarga
      - Aunque sí se usa para controlar un tipo de recurso específico

Ejemplo de interbloqueo (1/2)

```
// Versión con posible interbloqueo
struct nodo {
    struct nodo *siguiente;
    /* otros campos */
};
struct lista {
    pthread_mutex_t mutex_lista;
    struct nodo *primer_nodo;
};
void mover_elemento_de_lista(struct lista *origen, struct lista* destino,
    struct nodo *elemento, int posicion_destino) {
    pthread_mutex_lock(&origen->mutex_lista);
    pthread_mutex_lock(&destino->mutex_lista);
    /* mueve el elemento a la lista destino */
    pthread_mutex_unlock(&origen->mutex_lista);
    pthread_mutex_unlock(&destino->mutex_lista);
}
```

Ejemplo de interbloqueo (2/2)

```
// Versión libre de interbloqueos
void mover_elemento_de_lista(struct lista *origen, struct lista* destino,
    struct nodo *elemento, int posicion_destino) {
    if (origen < destino) {
        pthread_mutex_lock(&origen->mutex_lista);
        pthread_mutex_lock(&destino->mutex_lista);
    }
    else {
        pthread_mutex_lock(&destino->mutex_lista);
        pthread_mutex_lock(&origen->mutex_lista);
    }
    /* mueve el elemento a la lista destino */
    pthread_mutex_unlock(&origen->mutex_lista);
    pthread_mutex_unlock(&destino->mutex_lista);
}
```