Concurrencia de procesos: Interbloqueo e inanición

Eloy Anguiano Rey
eloy.anguiano@uam.es

Ana González
ana.marcos@uam.es

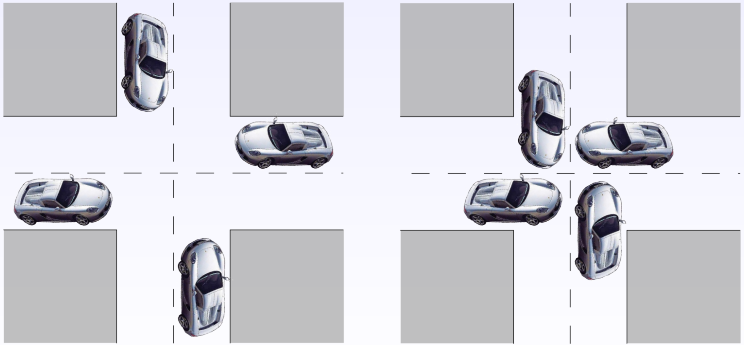
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Definición

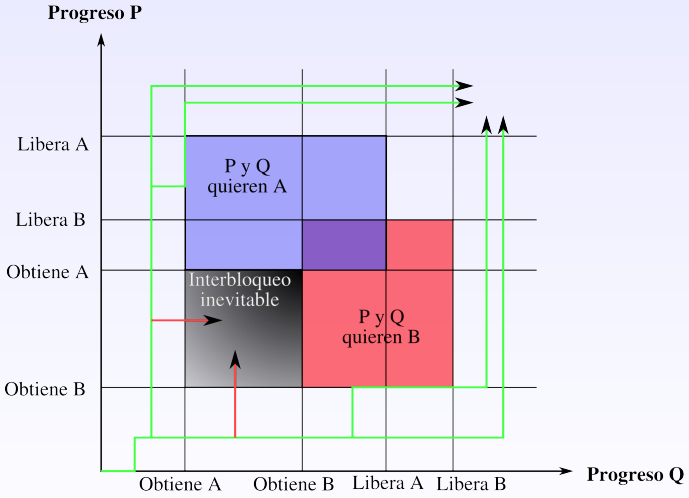
- Bloqueo permanente de un conjunto de procesos que compiten por los recursos o bien se comunican unos con otros.
- No existe una solución eficiente.
- Suponen necesidades contradictorias de recursos por parte de dos o más procesos.
- El bloqueo ocurre cuando un proceso monopoliza el acceso a un recurso y requiere otro recurso que ha sido ya asignado a un segundo proceso, que a su vez necesita el recurso monopolizado por el primer proceso.

Definición
Ejemplos

- Cena de filósofos sin coordinación.
- Ley del estado de Kansas: *"Si dos trenes se aproximan uno al otro en un cruce, ambos harán un alto total y ninguno arrancará de nuevo hasta que el otro haya salido del cruce"*.
- Inversión de prioridades.

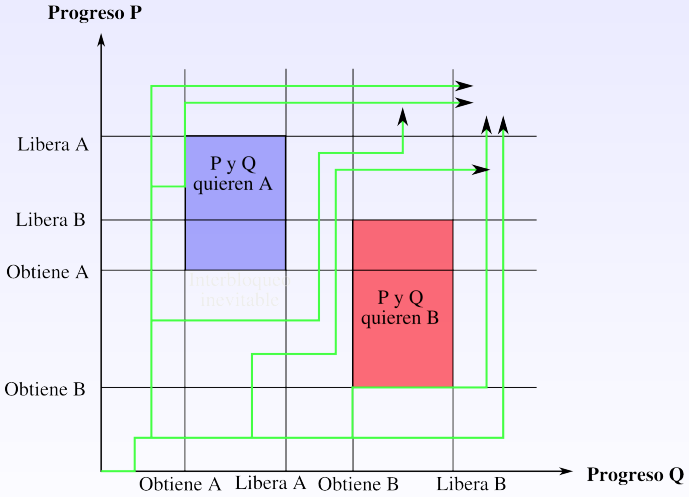


Definición
Estados seguros e inseguros



Definición

Estados seguros e inseguros



Recursos reutilizables
Características

- Pueden ser usados por un proceso y no se agotan con el uso.
- Los procesos obtienen unidades de recursos que liberan posteriormente para que otros procesos las reutilicen.
- Procesadores, canales de E/S, memoria principal y secundaria, archivos, bases de datos y semáforos.
- El interbloqueo se produce si cada proceso retiene un recurso y solicita el otro.

Recursos reutilizables
Ejemplos

Acceso a zonas de memoria compartida utilizando semáforos

Memoria Compartida
Semáforo Protege_Var1=1
Semáforo Protege_Var2=1
int Var1
int Var2

Proceso 1
...
down(Protege_Var1);
down (Protege_Var2);
Var1 = Var2+1;
up (Protege_Var2);
up (Protege_Var1);

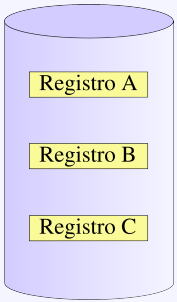
Proceso 2
...
down(Protege_Var2);
down (Protege_Var1);
Var2 = 2*Var1;
up (Protege_Var1);
up (Protege_Var2);

Recursos reutilizables
Ejemplos

Acceso a registros de bases de datos

Instrucciones

1. Proceso 1: Copia Reg. A a Reg. B
2. Proceso 2: Copia Reg. B a Reg. C
3. Proceso 3: Copia Reg. C a Reg. A



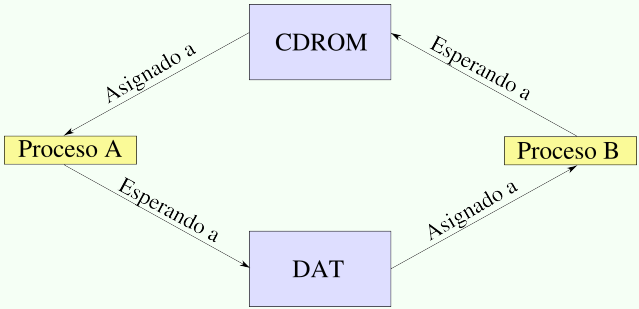
Proceso

1. Pr.1: Solicita Reg. A
2. Pr.1: Cierra Reg. A
3. Pr.2: Solicita Reg. B
4. Pr.2: Cierra Reg. B
5. Pr.3: Solicita Reg. C
6. Pr.3: Cierra Reg. C
7. Pr.1: Solicita Reg. B
8. Pr.2: Solicita Reg. C
9. Pr.3: Solicita Reg. A
10. Pr.1: Espera
11. Pr.2: Espera
12. Pr.3: Espera

Recursos reutilizables
Ejemplos

Acceso a dispositivos

1. Proceso A copia un fichero del CD/ROM a una cinta de backup (DAT).
2. Proceso B copia un fichero de la cinta de backup (DAT) al CD/ROM.



Recursos reutilizables
Ejemplos

Uso de memoria en sistemas multiprogramados

Si el espacio disponible es de 200 KB y se origina la siguiente secuencia de peticiones:

1. Proceso 1: solicita 80 KB
2. Proceso 2: solicita 70 KB
3. Proceso 1: solicita 60 KB
4. Proceso 2: solicita 80 KB

se produce un interbloqueo si ambos procesos avanzan hasta su segunda petición.

Recursos consumibles
Características

- Puede ser creado (producido) y destruido (consumido) por un proceso.
- Interrupciones, señales, mensajes e información en buffers de E/S.
- El interbloqueo se produce si el **receive** es bloqueante.
- Puede darse una combinación de sucesos poco habitual que origine el interbloqueo.

Por transmisión de mensajes

Si el **receive** es bloqueante estos procesos pueden producir interbloqueo:

Proceso 1	**Proceso 2**
...	...
Receive(P2);	Receive(P1);
...	...
Send(P2,M1);	Send(P1,M2);
...	...

Grafos de asignación de recursos
Definición

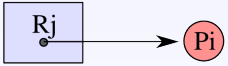
Permiten visualizar la situación (asignación y solicitud) de recursos.

Proceso i:

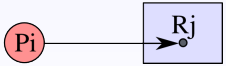
Recurso tipo j (un único recurso):

Recurso tipo j (tres recursos):

Recurso j asignado al proceso i:



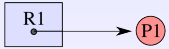
Proceso i a la espera del recurso j:



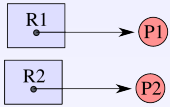
Grafos de asignación de recursos

Ejemplo de grafo con bloqueo

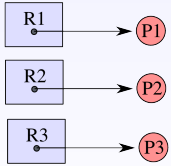
1.- Proceso 1: solicita recurso 1 y se asigna



2.- Proceso 2: solicita recurso 2 y se asigna

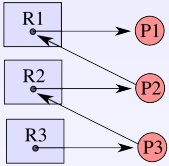


3.- Proceso 3: solicita recurso 3 y se asigna

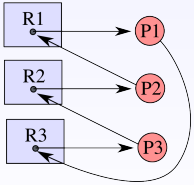


4.- Proceso 2: solicita recurso 1

5.- Proceso 3: solicita recurso 2



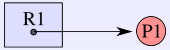
6.- Proceso 1: solicita recurso 3



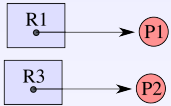
Grafos de asignación de recursos

Ejemplo de grafo sin bloqueo

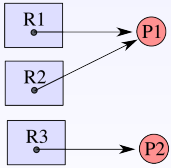
1.- Proceso 1: solicita recurso 1 y se asigna



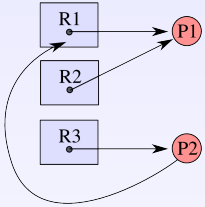
2.- Proceso 2: solicita recurso 3 y se asigna



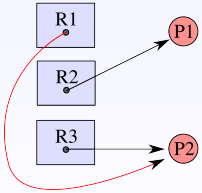
3.- Proceso 1: solicita recurso 2 y se asigna



4.- Proceso 2: solicita recurso 1



5.- Proceso 1: libera recurso 1



Condiciones de interbloqueo

1. **Exclusión mutua**:
   - Sólo un proceso puede usar un recurso cada vez.
2. **Retención y espera**:
   - Hay al menos un proceso que tiene asignado un recurso y se encuentra en espera de que otro proceso libere otro recurso.
3. **No apropiación**:
   - Para dos procesos se puede resolver de la siguiente forma: si a un proceso que retiene ciertos recursos se le deniega una nueva solicitud de un recurso, a dicho proceso no se le puede exigir que libere sus recursos. Necesita que los procesos sean recuperables.
4. **Círculo vicioso de espera**.

Gestión de interbloqueos

1. **No hacer nada** (algoritmo del avestruz) Ej. UNIX. Decisión de diseño, por motivos de flexibilidad. Ej. Nº de archivos abiertos.
2. Permitir que el sistema se bloquee y luego se **recupere** (requiere **detección periódica**).
3. Uso de **protocolos** que aseguren que el sistema nunca se **bloqueará**:
   - Mediante **evasión**: proporcionar al sistema información anticipada acerca de las necesidades de recursos de los procesos, de modo que pueda encontrar secuencias de asignación de CPU a los procesos que eviten los bloqueos.
   - Mediante **prevención**: garantizar que una (o más) de las condiciones necesarias para la formación de bloqueos no se cumpla.

Gestión de interbloqueos

Detección de interbloqueos

Bloqueos con un solo recurso de cada tipo

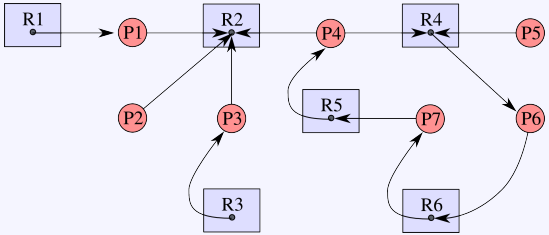
Para cada nodo de la gráfica de procesos/recursos se crea una lista de nodos vacía L que se rellena ejecutando el siguiente algoritmo:

1. Se añade el nodo activo a la lista **L** y se comprueba si el nodo está dos veces en la lista. Si está dos veces la gráfica tiene un ciclo (y por tanto hay un bloqueo).
2. Desde el nodo activo se buscan arcos que salgan del mismo. Si los hay se pasa a **3**, si no a **4**.
3. Se elige un arco que no haya sido recorrido, se marca como recorrido y se marca el nodo apuntado por el arco como activo. Se vuelve a **1**.
4. Se ha llegado a un punto donde no se puede avanzar. Se vuelve al nodo anterior y se marca como activo. Se pasa al paso. **1**. Si el nodo al que se vuelve es el inicial se acaba. No hay bloqueo.

Gestión de interbloqueos

Detección de interbloqueos

Ejemplo



Se van comprobando todos los caminos:

1. $L(R_1) = \langle R_1 \rangle \langle R_1, P_1 \rangle \langle R_1, P_1, R_2 \rangle \langle R_1, P_1 \rangle \langle R_1 \rangle$ **No hay bloqueo**
2. $L(P_1) = \langle P_1 \rangle \langle P_1, R_2 \rangle \langle P_1 \rangle$ **No hay bloqueo**
3. $L(R_2) = \langle R_2 \rangle$ **No hay bloqueo**
4. $L(P_4) = \langle P_4 \rangle \langle P_4, R_2 \rangle \langle P_4 \rangle \langle P_4, R_4 \rangle \langle P_4, R_4, P_6 \rangle \langle P_4, R_4, P_6, R_6 \rangle$ $\langle P_4, R_4, P_6, R_6, P_7 \rangle \langle P_4, R_4, P_6, R_6, P_7, R_5 \rangle \langle P_4, R_4, P_6, R_6, P_7, R_5, P_4 \rangle$ Hay un ciclo luego **hay bloqueo**

Gestión de interbloqueos

Detección de interbloqueos

Bloqueos con varios recursos de cada tipo

- Vector de **Recursos existentes** $E = \{E_1, E_2, E_3, ..., E_m\}$
- Vector de **Recursos disponibles** $A = \{A_1, A_2, A_3, ..., A_m\}$
- Matriz de **asignación actual** $C = \begin{Bmatrix} C_{11} & C_{12} & \cdots & C_{1m} \\ C_{21} & C_{22} & \cdots & C_{2m} \\ \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & \cdots & C_{nm} \end{Bmatrix}$
- Matriz de **solicitudes** $R = \begin{Bmatrix} R_{11} & R_{12} & \cdots & R_{1m} \\ R_{21} & R_{22} & \cdots & R_{2m} \\ \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & \cdots & R_{nm} \end{Bmatrix}$

Gestión de interbloqueos
Detección de interbloqueos

Bloqueos con varios recursos de cada tipo

- $\sum_i C_{ij} + A_j = E_j$

Algoritmo

1. Se busca un proceso no marcado P_i que cumpla que $\forall x,\ R_{ix} \leq A_x$. Si no hay procesos sin marcar, se acaba el algoritmo (**no hay bloqueo**).
2. Si se encuentra un proceso que cumpla esa condición, se marca, se suma la línea C_{ix} a A y se vuelve a 1.
3. Si no existe ese proceso, se acaba el algoritmo (**hay bloqueo**).

Gestión de interbloqueos

Detección de interbloqueos

Ejemplo

Recursos existentes

$$E = \{4, 2, 3, 1\}$$

Recursos disponibles

$$A = \{2, 1, 0, 0\}$$

Matriz de solicitudes

$$R = \begin{Bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{Bmatrix}$$

Matriz de asignación actual

$$C = \begin{Bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{Bmatrix}$$

Algoritmo

1.- $R_1 \not\leq A, R_2 \not\leq A, R_3 \leq A$
2.- $A = \{2, 2, 2, 0\}$
1.- $R_1 \not\leq A, R_2 \leq A$
2.- $A = \{4, 2, 2, 1\}$
1.- $R_1 \leq A$
2.- $A = \{4, 2, 3, 1\}$
1.- No hay procesos sin marcar. **Sin bloqueo**

Gestión de interbloqueos
Recuperación de interbloqueos

1. Recuperación mediante apropiación:
   - Se selecciona un proceso (o varios) y se le 'requisan' los recursos para cederlos a otros procesos bloqueados.
2. Recuperación mediante Rollback:
   - Se almacena en un fichero periódicamente el estado del proceso (imagen de memoria) y estado de los recursos utilizados. Al producirse un bloqueo se detecta los recursos que son necesarios y un proceso que tenga alguno de esos recursos se interrumpe y se retrasa hasta el punto de verificación anterior a la solicitud del recurso (puede repetirse el bloqueo).

Gestión de interbloqueos
Recuperación de interbloqueos

3. Recuperación mediante eliminación de procesos. Hay dos posibilidades de actuación:
   - Se abortan todos los procesos bloqueados.
   - Se selecciona un procesos bloqueado o uno no bloqueado que tenga recursos necesarios. Elegir procesos que se puedan reiniciar (si existen). Se termina de eliminar procesos cuando se desbloquee el sistema.
   - Criterios para la selección de procesos a terminar:
     - Menor prioridad.
     - Menor líneas de salida producidas al terminar.
     - Menor tiempo de CPU consumido.
     - Mayor tiempo restante.
     - Menor número de recursos asignados.
     - Tipo de recursos asignados.
     - Mayor número de recursos necesarios para terminación.
   - Problemas:
     - Si el proceso está actualizando un archivo, su eliminación puede causar inconsistencias.
     - Si el procesos está imprimiendo, se reinicia la impresora.

Gestión de interbloqueos

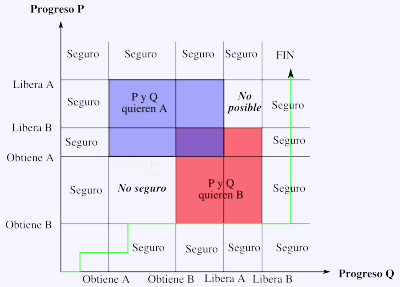
Evasión de interbloqueos

Para que se pueda realizar evasión de interbloqueos es necesario que se cumplan estas dos condiciones:

- La condición es que todos los procesos declaren sus necesidades totales de recursos al principio.
- Un recurso se concede sólo mientras se permanece en un estado seguro.

Análisis de trayectoria de recursos

Es muy compleja para más de dos recursos y dos procesos.



Gestión de interbloqueos
Prevención de interbloqueos

La única forma real de evitar bloqueos consiste en garantizar que no se cumpla una de las condiciones necesarias para la formación de bloqueos

1. Prevención de la exclusión mutua:
   - Problema: hay recursos que necesariamente requieren la exclusión mutua:
     - Archivos con permiso de escritura.
     - La tabla de procesos.
     - La tabla de nodo-i activos.
     - La memoria.
     - El espacio en disco.
     - ...

Gestión de interbloqueos
Prevención de interbloqueos

2. Prevención de la condición de Detenerse y Esperar:
   - Se exige a los procesos que soliciten todos los recursos al comenzar su ejecución.
   - Problemas:
     - Un proceso no 'sabe' a priori que recursos va a utilizar.
     - Uso muy ineficiente de los recursos: los recursos se acaparan desde el inicio, aunque no se usen hasta el final del proceso.
     - Se puede producir la inanición: procesos no bloqueados pueden no tener nunca acceso a un recurso acaparado por otro.

Gestión de interbloqueos
Prevención de interbloqueos

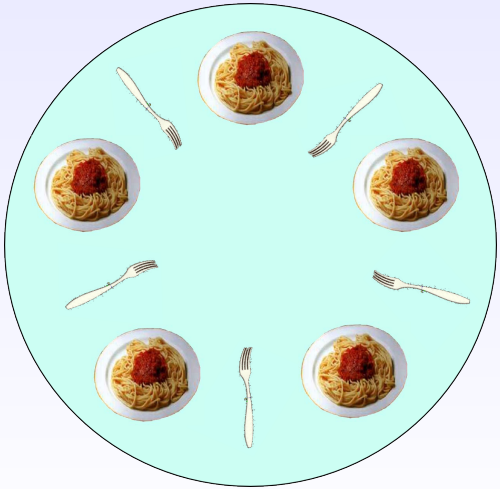
3. Prevención contra la condición de no apropiación:
   - Si un proceso solicita un recurso no disponible, se interrumpe y se le expropia de todos sus recursos.
   - El proceso se reinicia cuando se le pueden proporcionar todos los recursos que necesita.
   - Sólo es realizable con recursos cuyo estado puede ser fácilmente almacenable/recuperable:
     - Registros de CPU.
     - Memoria.

Gestión de interbloqueos
Prevención de interbloqueos

4. Prevención contra la condición de espera circular:
   1. Los recursos sólo pueden utilizarse uno a uno. Esto no es posible en muchos casos: ej. copia de un fichero de cinta a disco.
   2. Los recursos se numeran, de modo que sólo se puedan solicitar en un orden determinado.
   3. Variante: Sólo se pueden solicitar recursos que tengan un número mayor del último que se haya solicitado.

   - Problema: cuando hay un número alto de recursos, no es fácil diseñar un orden que satisfaga a todos los procesos.

Cena de los filósofos



Cena de los filósofos
Primer intento de solución


```
1   #define N 5
2
3   void filosofo (void) {
4       int i;
5       while (1) {
6           piensa();
7           coge_palillo(i);
8           coge_palillo((i+1) % N);
9           come();
10          deja_palillo(i);
11          deja_palillo((i+1) % N);
12      }
13  }
```

Posible bloqueo mutuo

Cena de los filósofos
Segundo intento de solución


```
1   #define N 5
2
3   void filosofo (void) {
4       int i;
5       while (1) {
6           piensa();
7           coge_palillo(i);
8           if(libre((i+1) % N)) {
9               coge_palillo((i+1) % N);
10              come();
11              deja_palillo(i);
12              deja_palillo((i+1) % N);
13          }
14          else deja_palillo(i);
15      }
16  }
```

Posible inanición si los filósofos actúan simultáneamente (poco probable pero posible)

Cena de los filósofos

Solución


```
1  #define N  5
2  #define IZQD  (i-1) %N
3  #define DCHA  (i+1) %N
4  #define PENSANDO 0
5  #define HAMBRIENTO 1
6  #define COMIENDO 2
```

```
1  typedef int semaforo;
2  int estado[N];
3  semaforo mutex;
4  semaforo S[N];
```

```
1  void filosofo(int i)
2  {
3    while (1) {
4      piensa();
5      coge_palillos(i);
6      come();
7      deja_palillos(i);
8    }
9  }
10
11 void coge_palillos(int i)
12 {
13   down (&mutex);
14   estado[i]= HAMBRIENTO;
15   intento(i);
16   up (&mutex);
17   down (&S[i]);
18 }
```

```
1  void deja_palillos(int i)
2  {
3    down (&mutex);
4    estado[i]= PENSANDO;
5    intento(IZQD);
6    intento(DCHA);
7    up (&mutex);
8  }
9
10 void intento (int i)
11 {
12   if(estado[i] == HAMBRIENTO &&
13         estado[IZQD] != COMIENDO &&
14         estado[DCHA] != COMIENDO) {
15     estado[i] = COMIENDO;
16     up(&S[i]);
17   }
18 }
```

Mecanismos de concurrencia
En UNIX

- Tubos (pipes).
- Mensajes.
- Memoria compartida.
- Semáforos.
- Señales.

Mecanismos de concurrencia
En Windows 2000

- Proceso.
- Hilo.
- Archivo.
- Entrada de consola.
- Notificación de cambio de archivo.
- Mutante.
- Semáforo.
- Suceso.
- Temporizador.