Tema 2. Interbloqueos

Tódolos SO teñen a capacidade de outorgar a un proceso o acceso exclusivo a certos recursos para evitar que se produzan incoherencias.

> **Interbloqueo**: situación na que dous ou máis procesos quedan bloqueados indefinidamente debido a un conflicto entre as súas diferentes necesidades. Polo tanto dise que:
> *Un conxunto de procesos está nun interbloqueo se cada proceso no conxunto está bloqueado esperando un evento que só pode ser ocasionado por outro proceso do conxunto*.

Os interbloqueos ocorren tanto nos recursos de hardware como nos de software, e se deben a que os *procesos compiten* polo emprego deses recursos. Cando ocorre un interbloqueo, tódolos procesos involucrados seguirán *esperando indefinidamente*, polo que supoñeremos que cada proceso só ten un fío e que non hai interrupcións que desperten un proceso bloqueado.

Recursos

> **Recurso**: obxecto (dispositivo hardware, peza de información...) que o SO outorga aos procesos e que se debe adquirir, empregar e liberar posteriormente.

Hai dous **tipos** de recurso:

- **Apropitivos**: *pódeselle quitar* ao proceso que o posúe sen efectos daniños (ex: memoria)
- **Non apropiativos**: *non se lle pode quitar* ao propietario actual sen efectos daniños (ex: CD-ROM)
  *Os máis problemáticos en termos de interbloqueos son os non apropiativos, xa que os conflictos dos apropiativos se poden resolver mediante a asignación dos recursos dun proceso a outro.

Secuencia de accións para solicitar un recurso:

1. *Solicitar* o recurso → 2. *Empregar* o recurso → 3. *Liberar* o recurso

Adquisición de recursos

Para algúns tipos de recursos, é responsabilidade dos procesos de usuario administrar o seu uso, por exemplo, **asociando un semáforo con cada recurso**. Estos semáforos inicialízanse a 1 e implementan a *secuencia de accións* da seguinte forma:

1. *Down* → adquirir recurso
2. *Empregar* recurso
3. *Up* → liberar recurso
   Se os procesos necesitan *dous ou máis recursos*, estes se adquiren de **forma secuencial** (a orde na que se adquiren é importante).

Exemplo

Imaxina unha situación na que temos dous procesos, A e B, e dous recursos. Na figura (a) ambos procesos piden os recursos na mesma orde, e na (b), nunha orde distinta.

- (a): Un dos procesos adquire o primeiro recurso, despois o segundo e realiza o seu traballo. Se o outro proceso quere calquera dos recursos antes de que o primeiro remate con eles, se bloquea esperando.
- (b): Un dos procesos adquire o primer recurso, despois o segundo proceso adquire o segundo recurso: temos un interbloqueo, xa que o primer proceso non pode avanzar sen o segundo recurso e viceversa.

```
typedef int semaforo;
  semaforo recurso_1;
  semaforo recurso_2;

  void proceso_A(void) {
    down(&recurso_1);
    down(&recurso_2);
    usar_ambos_recursos();
    up(&recurso_2);
    up(&recurso_1);
  }

  void proceso_B(void) {
    down(&recurso_1);
    down(&recurso_2);
    usar_ambos_recursos();
    up(&recurso_2);
    up(&recurso_1);
  }
```

(a)

```
semaforo recurso_1;
semaforo recurso_2;

void proceso_A(void) {
  down(&recurso_1);
  down(&recurso_2);
  usar_ambos_recursos();
  up(&recurso_2);
  up(&recurso_1);
}

void proceso_B(void) {
  down(&recurso_2);
  down(&recurso_1);
  usar_ambos_recursos();
  up(&recurso_1);
  up(&recurso_2);
}
```

(b)

Interbloqueos de recursos

Un interbloqueo de recursos ocorre cando **cada proceso espera pola liberación de algún recurso** que está sendo *empregado por outro proceso do conxunto*, polo que ningún dos procesos se pode executar, ningún pode liberar recursos e ningún pode ser despertado.

Condicións para os interbloqueos de recursos

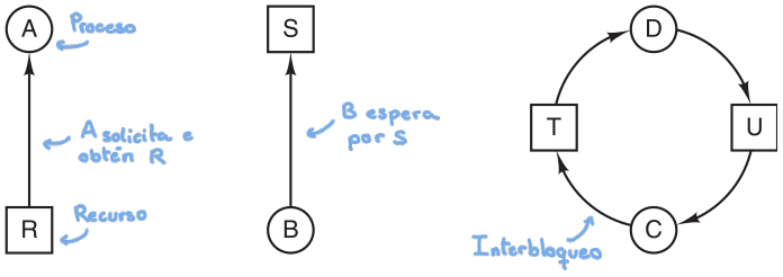
Para que ocorra un interbloqueo deben aplicarse todas as condicións seguintes:

1. **Condición de exclusión mutua** → cada proceso *se asigna a un só proceso ou está dispoñible*.
2. **Condición de contención e espera** → os procesos que conteñen recursos outorgados previamente, poden *solicitar novos recursos*.
3. **Condición non apropiativa** → *os recursos outorgados* previamente *non se lle poden quitar a un proceso pola forza*, senon que *deben ser liberados* de explicitamente polo proceso que os contén.
4. **Condición de espera circular** → debe haber unha *cadea circular* de dous ou máis *procesos*, cada un *esperando por un recurso contido polo seguinte* proceso da cadea.

Modelado de interbloqueos

Para **representar interbloqueos** gráficamente empréganse **grafos dirixidos** da seguinte forma:

- Os **nodos** poden ser de dous tipos: *procesos* (circulos) ou *recursos* (cadrados)
- Os **arcos** en dirección *recurso→proceso*, significa que o recurso está *asignado* ao proceso; mentres que os arcos en dirección *proceso→recurso*, significa que o proceso está *bloqueado* á espera do recurso.



O sistema operativo *non ten que executar os procesos nunha orde especial*, polo que se ao aoutorgar unha petición específica, se pode producir un interbloqueo, o SO *pode suspender o proceso* sen outorgar a solicitude ata que sexa seguro.

PROCESOS

A
Solicitud R
Solicitud S
Liberación R
Liberación S

(a)

B
Solicitud S
Solicitud T
Liberación S
Liberación T

(b)

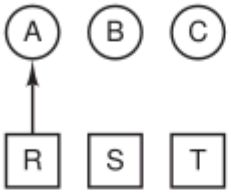
C
Solicitud T
Solicitud R
Liberación T
Liberación R

(c)

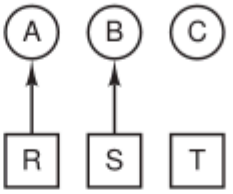
PLANIFICACIÓN 1

1. A solicita a R
2. B solicita a S
3. C solicita a T
4. A solicita a S
5. B solicita a T
6. C solicita a R
   interbloqueo

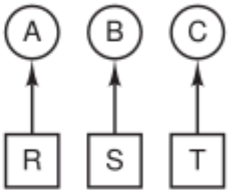
(d)



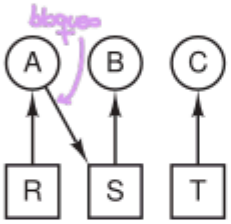
(e)



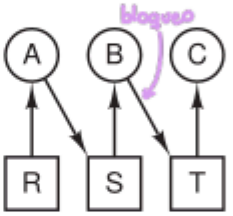
(f)



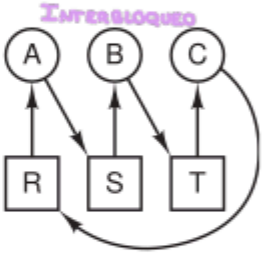
(g)



(h)



(i)

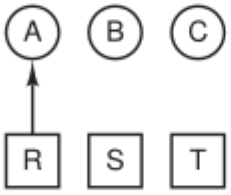


(j)

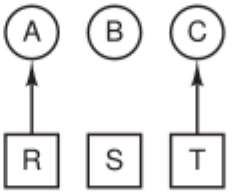
PLANIFICACIÓN 2

1. A solicita a R
2. C solicita a T
3. A solicita a S
4. C solicita a R
5. A libera a R
6. A libera a S
   no hay interbloqueo

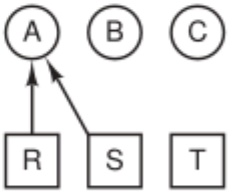
(k)



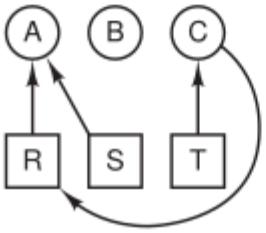
(l)

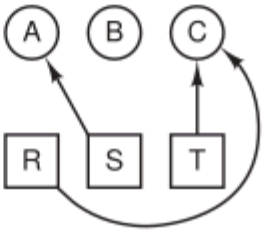


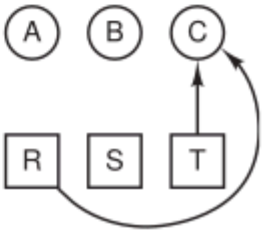
(m)



(n)







Estratexias para lidiar cos interbloqueos

Hai catro estratexias:

- Ignorar o problema
- Detección e recuperación
- Evitalos de forma dinámica
- Prevención

Ignorar o problema: O algoritmo da avestruz

Simplemente *ignórase o problema*: "Tal vez si usted lo ignora, él lo ignorará a usted".
É o máis habitual (Windows, Linux...)

Detección e recuperación

En vez de intentar evitar os interbloqueos, o sistema intenta detectalos para recuperarse despois.

Detección de interbloqueos con un recurso de cada tipo

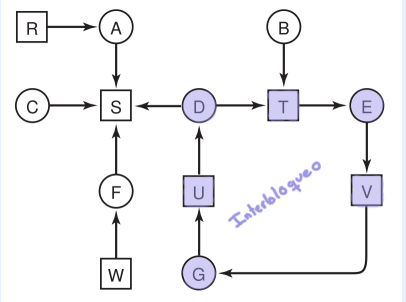
É o caso máis simple: *se o grafo de recursos ten algún ciclo, entón existe un interbloqueo*, e se non existen ciclos, entón o sistema non está en interbloqueo.

Exemplo

Sistema con sete procesos (A-G) e seis recursos (R-W):

1. O proceso A contén a R e quere a S.
2. O proceso B non contén ningún recurso pero quere a T.
3. O proceso C non contén ningún recurso pero quere a S.
4. O proceso D contén a U e quere a S e a T.
5. O proceso E contén a T e quere a V.
6. O proceso F contén a W e quere a S.
7. O proceso G contén a V e quere a U.

   Para *saber se o sistema está en interbloqueo* e, en caso afirmativo, *qué procesos están involucrados*, podemos construir o **grafo de recursos**:



Para *formalizar a detección de interbloqueos* empregamos **algoritmos para detectar ciclos en digrafos**.
*ff dixo que nn facia falta saber os algoritmos

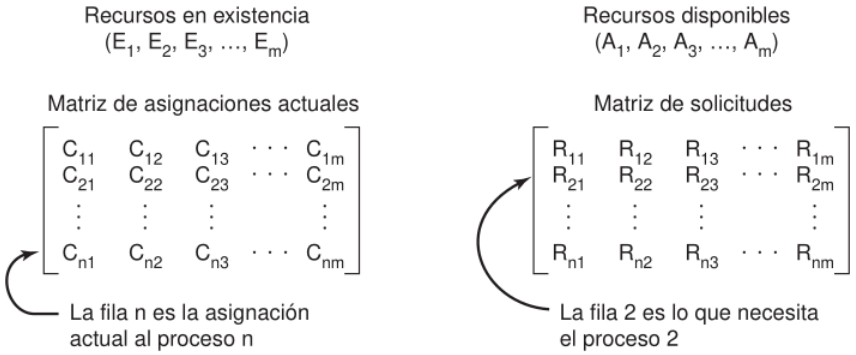
Detección de interbloqueos con varios recursos de cada tipo

Para detectar interbloqueos entre n procesos, de P_1 a P_n, con m tipos de recursos, tendo E_i recursos de tipo i (sendo $i \leq m$), necesitamos un **algoritmo baseado en matrices**.

Elementos:

- $E \rightarrow$ vector de *recursos existentes* $\rightarrow$ número total de instancias de cada recurso
- $A \rightarrow$ vector de *recursos disponibles* $\rightarrow$ numero de instancias de cada recurso sin asignar
- C $\rightarrow$ matriz de *asignacións actuais*
- R $\rightarrow$ matriz de *peticións*

Desta forma queda:



Pasos do algoritmo simplificado:

1. Inicializánse tódolos *procesos como non marcados* e se toma A como o *vector de recursos dispoñibles*.
2. *Búscase un proceso $P*{i}$ sen marcar tal que para cada recurso j, $R[i][j] \leq A[j]$_
3. Se existe, asumimos que P_i pode rematar, polo que *o marcamos e actualizamos* $A = A+$ fila i de C.
4. Volvemos ao paso 2 ata que non atopemos ningún proceso que cumpra a condición. **Todos os procesos non marcados ao final están en interbloqueo**.

Recuperacón por medio de apropiación

Consiste en **quitarlle temporalmente un recurso ao seu propietario actual** e *outorgarllo a outro proceso*, para despois *devolverllo* ao seu propietario orixinal, polo que pode requerir unha intervención manual.

> Soe ser dificil ou imposible recuperarse desta forma.

Recuperación a través de retroceso

Consiste en facer que os procesos realicen **puntos de comprobación** de forma periódica, é dicir, que vaian *escribindo o seu estado nun arquivo* para *poder reinicialo máis tarde*. O punto de comprobación contén a *imaxe da memoria* e o *estado do recurso* (qué recursos están asignados ao proceso nun momento dado). Para que sexan máis efectivos, *os novos puntos de comprobación escríbense en novos arquivos*, de forma que se acumule unha **secuencia completa** a medida que o proceso se execute.
Cando se detecta un interbloqueo, é doado ver que recursos se necesitan, polo que **para recuperarse**, un proceso que posúe un recurso necesario só ten que *iniciarse nun dos seus puntos de comprobación anteriores* e *o recurso se lle asigna a un dos procesos do interbloqueo*.

Recuperación mediante a eliminación de procesos

Consiste en **eliminar un ou máis procesos**: ou ben un dos *procesos do ciclo*, ou ben un que *non estea no ciclo*, para poder liberar os seus recursos (este elíxese con coidado). Sempre que sexa posible, é mellor eliminar un proceso que poida volver a ser executado dende o inicio sen efectos daniños.

Métodos para evitar interbloqueos

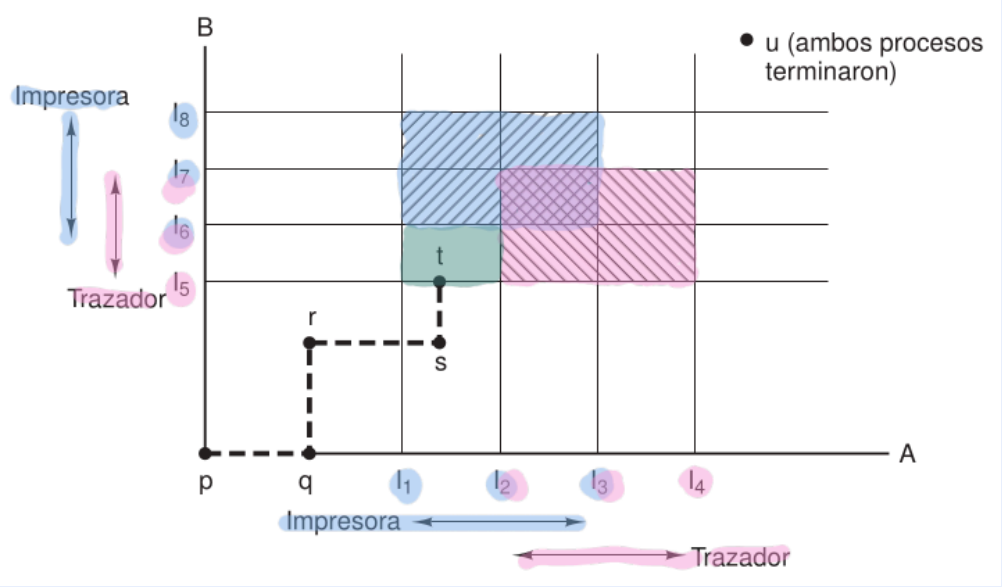
Na maioría dos sistemas, os recursos se solicitan dun en un, polo que o sistema debe ser capaz de *decidir se é seguro outorgar un recurso*, e realizar a asignación só en caso afirmativo.

Traxectorias dos recursos (ver exame 2022)

Nos diagramas de traxectorias dos recursos, *cada punto no diagrama representa un estado* conxunto dos dous procesos, e, cun só procesador, as rutas deben ser verticais ou horizontais, nunca diagonais.

Exemplo

Neste diagrama móstranse as traxectorias de dous procesos competindo por dous recursos.



- Punto p $\rightarrow$ ningún dos procesos executou instruccións.
- Optamos por executar primeiro o proceso A (eixo das x) $\rightarrow$ punto q
- Optamos por executar o proceso B $\rightarrow$ punto r (camiño vertical)
- A solicita a impresora (cruza I_1) e se lle concede $\rightarrow$ traxectoria r-s
- B solicita o trazador (en I_5) $\rightarrow$ punto t

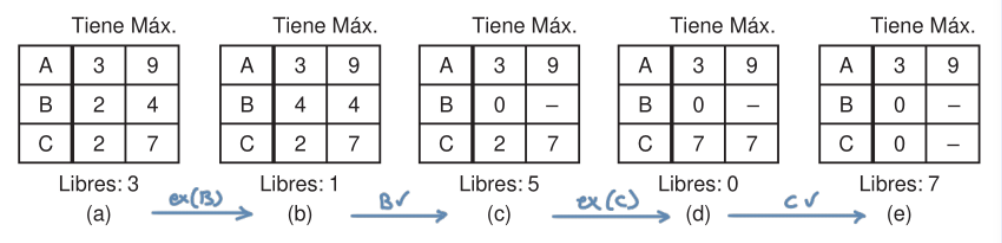
A rexión azul representa a zona na que ambos procesos teñen a impresora, e a rexión rosa, cando ambos teñen o trazador, polo que ambas zonas son imposibles debido á exclusión mutua. Se o sistema entra na zona delimitada polo cadrado verde, entrará en interbloqueo cando chegue á intersección entre I_2 e I_6, xa que tanto A como B solicitan recursos que xa están asignados, o que fai que o cadrado verde sexa inseguro e non se deba entrar en el. Polo tanto, cando o sistema chega ao punto t, o único seguro é executar A ata que chegue a I_4 e libere ambos recursos.

Estados seguros e inseguros

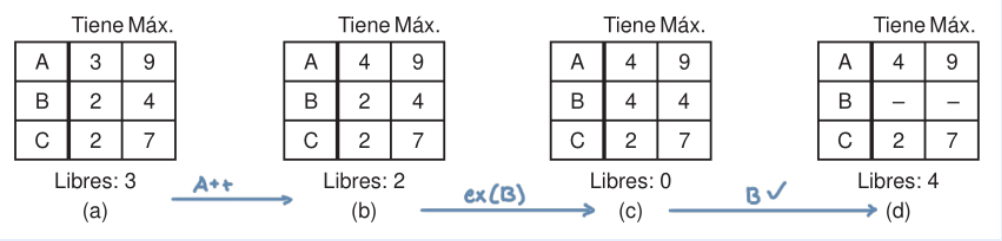
Os principais algoritmos pra evitar interbloqueos baséanse no concepto de **estados seguros**, é decir, estados nos que *hai certa orde* de programación na que *se pode executar cada proceso ata completalo*, aínda que todos solicitaran de maneira repentina todos os seus recursos de inmediato.

Exemplo

Na seguinte imaxe, o estado (a) é seguro xa que existe unha secuencia de asignacións que permite completar tódolos procesos, é decir, o sistema pode evitar un interbloqueo mediante unha programación cuidadosa: executamos B → executamos C



Se decidisemos cederlle unha unidade de recurso a A, obteríamos o estado (b) da figura seguinte:



Nese estado, B poderíase completar, pero despois , dado que só quedarían libres 4 recursos e dado que tanto A e C necesitan 5, non hai secuencia que garantice que os procesos se completarán, polo que o sistema non debería ter outorgado a primeira petición de A.

Observación: un estado inseguro non é un estado en interbloqueo per se, senon que simplemente é un estado que non *garante* que todos os procesos terminen.

Algoritmo do banqueiro para un só recurso (ver exame 2022)

É un algoritmo proposto por Dijkstra en 1965 que pode evitar interbloqueos. Búscanse **procesos que poidan ser satisfeitos** e se determina se os *recursos liberados* cando remate poden *satisfacer a outro de maneira recursiva*.

Exemplo

Neste exemplo, tanto o (a) como o (b) son estados seguros, mentres que o (c) é inseguro:

	Tiene	Máx.
A	0	6
B	0	5
C	0	4
D	0	7

Libres: 10

(a)

	Tiene	Máx.
A	1	6
B	1	5
C	2	4
D	4	7

Libres: 2

(b)

	Tiene	Máx.
A	1	6
B	2	5
C	2	4
D	4	7

Libres: 1

(c)

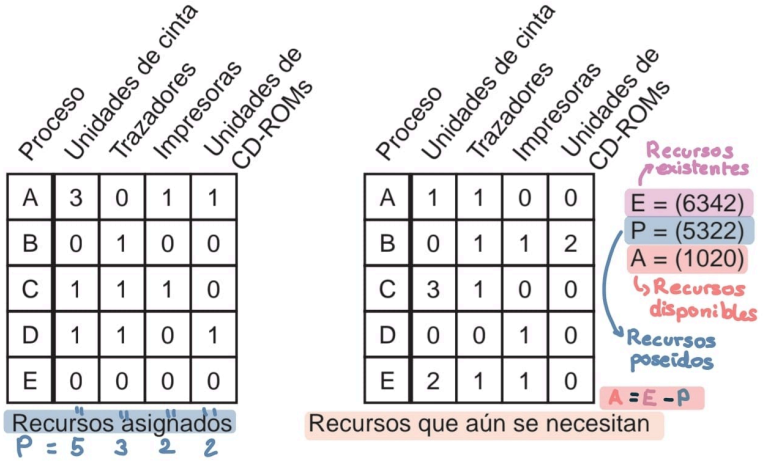
(b) é un estado seguro porque o banqueiro pode outorgarlle recursos aos procesos, por exemplo, na seguinte orde: C → B → A → D

(c) é un estado inseguro porque só queda un recurso libre, e todos os procesos precisan máis dun recurso para executarse, polo que se todos os procesos solicitasen o seu número máximo de recursos o banqueiro non podería satisfacer a ningún deles e se produciría un interbloqueo.

Así, o algoritmo do banqueiro **considera cada petición a medida que vai ocorrendo, e analiza se outorgala produce un estado seguro**: en caso afirmativo outorga a petición, e senón a pospón. Para **saber se un estado é seguro**, o banqueiro comproba se ten os *recursos suficientes* para satisfacer a algun dos seus clientes: en caso afirmativo, *vai facendo a simulación de outorgar as peticións* a cada un dos seus clientes. Se todos os "préstamos" son satisfactorios, entón a petición inicial conduce a un estado seguro e se pode outorgar.

Algoritmo do banqueiro para varios recursos

Para aplicar o algoritmo do banqueiro para varios recursos, necesitamos unha matriz de **recursos asignados**, outra de **recursos necesitados**, e tres vectores: de **recuros existentes** (E), de **recursos posuídos** (P) e de **recursos dispoñibles** (A).



É evidente que $A_i = E_i - P_i$, é decir, que os recursos dispoñibles son os existentes menos os posuídos.

Algoritmo para compobar se un estado é seguro:

1. Buscar unha *fila R* (un proceso) cuxos *recursos que aínda se necesitan* sexan *menores ou iguais que os recursos dispoñibles (A)*. Se non existe, entón o sistema entrará en interbloqueo en algún momento.
2. Supoñer que *o proceso da filla R obtén tódolos recursos* que necesita e remata. Marcamos ese *proceso como rematado* e *sumámoslle os seus recursos ao vector de recursos dispoñibles A*.
3. *Repetimos os pasos 1 e 2* ata que todos os procesos estean marcados como *terminados* ou ata que haxa un *interbloqueo*.

Prevención de interbloqueos

Para previr interbloqueos temos que asegurar que, polo menos, **nunca se cumpra unha das condicións** para estes (ver arriba).

Atacar a condición de exclusión mutua

Se **ningún recurso se asignara de maneira exclusiva a un só proceso**, nunca teríamos interbloqueos; pero esto non é posible ca maioría de recursos xa que sería un caos. Por exemplo, no caso dunha **impresora**, sería inviable permitir que tódolos procesos escribisen nela á vez, polo que neste modelo, o *único proceso que solicita realmente a impresora física* é o **demonio de impresión**; e, como este nunca solicita ningún outro recurso, o *interbloqueo para a impresora queda eliminado*.
Unha idea que se aplica con frecuencia é a de **evitar asignar un recurso cando non sexa estrictamente necesario**, e tratar de **asegurar que a menor cantidade posible de procesos reclamen ese recurso**.

Atacar a condición de contención e espera

Trátase de **evitar que os procesos que conteñen recursos esperen por máis recursos**, polo que se debería requerir que tódolos procesos *soliciten todos os seus recursos antes de empezar a execución*.
Non obstante, o **problema** deste método é que moitos procesos *non saben cantos recursos necesitarán ata que empezen a executarse* (se o souberan, poderíase empregar o algoritmo do banqueiro) . Ademáis, así os recursos *non se empregarían de maneira óptima*.
Unha forma distinta de atacar a condición de contención e espera é *requerir que un proceso que solicita un recurso libere termporalmete os recursos que contén* nun momento dado, para despois tratar de obter todo o que necesite á vez.

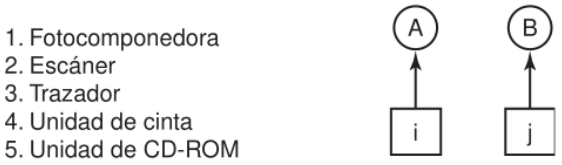
Atacar a condición non apropiativa

Trátase de **virtualizar os recursos** de forma que *só un proceso teña acceso ao recurso real*, pero todos teñan acceso ao recurso "virtual". Non obstante, **non todos os recursos se poden virtualizar** así.

Atacar a condición de espera circular

Hai varias formas de atacar esta condición:

- Ter unha regra que diga que **un proceso ten dereito a un só recurso en calquera momento**, polo que *se necesita outro recurso, debe liberar o anterior*.
- Proporcionar unha **numeración global de tódos os recursos**, de forma que os procesos poidan *solicitar recursos* sempre que queiran, pero realizando as peticións *en orde numérica*. Así, o grafo de asignación de recursos nunca terá ciclos (nunca se producirá un interbloqueo).



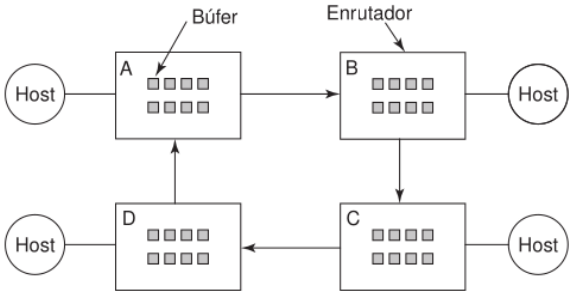
Outras cuestións

Bloqueo de dúas fases

*Ver BDII

Interbloqueos de comunicacións

É un tipo de interbloqueo que pode ocorrer nos **sistemas de comunicacións**, onde os procesos se comunican entre sí mediante o *envío de mensaxes*. Para romper estes interbloqueos introdúcense os *tempos de espera* (timeout) e os **protocolos** (mais info nos apuntamentos de Redes).



Bloqueo activo

O **bloqueo activo** (**livelock**) ocorre cando *ningún dos procesos se bloquea, pero tampouco progresa*, polo que é funcionalmente equivalente a un interbloqueo.

Inanición

Ocorre cando **certos procesos nunca reciben atención aínda que non estean nun interbloqueo**. Pódese *evitar* mediante unha *política de asignación de recursos tipo FIFO*.