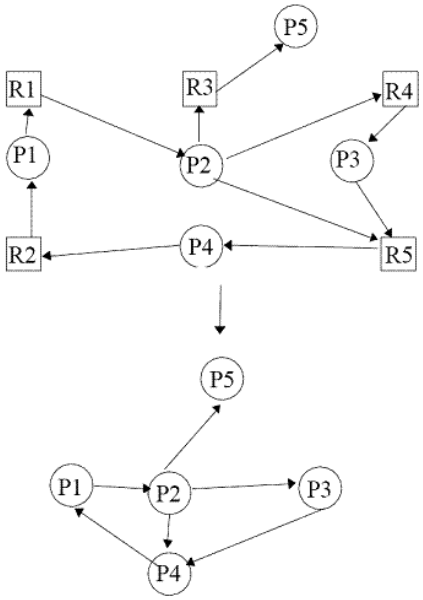
Systèmes d'Exploitation

Interblocage de processus

Version 1



Y. Challal, H. Bettahar, M. Vayssade

Table des matières

Objectifs

- Analyser les situations d'interblocage de processus
- Résoudre le problème d'interblocage de processus
- Modéliser les situations d'interblocage de processus

L'interblocage de processus

I

A. Définition

Définition : Interblocage [Tanenbeaum]

Un ensemble de processus est dans une situation d'interblocage si chaque processus attend un événement que seul un autre processus de l'ensemble peut provoquer. (L'événement signifie ici acquisition ou libération de ressource).

Protocole d'accès aux ressources

On considère qu'un système est constitué d'un nombre fini de ressources devant être distribué parmi un certain nombre de processus concurrents. Les ressources sont groupées en classes (types : Imprimantes, mémoires, registres, fichiers,...). Chaque classe de ressources comporte un nombre fini d'exemplaires (copies identiques).

Pour acquérir une ressource chaque processus suit le protocole suivant :

```
Demander (Ri)
<Utilisation>
Liberer (Ri)
```

Demander() et Liberer() sont généralement des appels systèmes (Open et Close File, Request et release Device, Allocate et Free Memory) Le système doit suivre l'état des ressources (libre, allouée) et décider (suivant une politique) a qui allouer une ressource libre. Si la ressource est occupée, le processus demandeur est mis en attente (de l'événement «libération de ressource»)

Exemple : Interblocage

On considère un système avec :

- Deux périphériques non partageables :
  - Périph1 (exp : imprimante)
  - Périph2 (exp : fichier ouvert en écriture)
- Deux processus P1 et P2 :

P1	P2
1. Demander(Périph1)	1. Demander(Périph2)
2. Demander(Périph2)	2. Demander(Périph1)
<Traitement>	<Traitement>
3. Liberer(Périph1)	3. Liberer(Périph2)
4. Liberer(Périph2)	4. Liberer(Périph1)

Exemple d'interblocage

Les deux processus respectent le protocole d'acquisition de ressources mais si l'on considère une exécution concurrente on peut aboutir à :

P1.1/ P1 obtient le Périph1

P2.1/ P2 obtient le périph2

P1.2/ P1 est mis en attente de la libération de Périph2

P2.2/ P2 est mis en attente de la libération de Périph1

C'est une situation d'interblocage.

B. Conditions nécessaires d'interblocage

Une situation d'interblocage peut survenir SI et SEULEMENT SI les quatre conditions suivantes se produisent simultanément dans un système :

1. **Exclusion mutuelle**: Une ressource au moins doit se trouver dans un mode non partageable et nécessite une EM pour son utilisation.
2. **Occupation et attente**: Il peut exister un processus occupant au moins une ressource et qui attend d'acquérir des ressources supplémentaires détenues par d'autres processus (allocation partielle).
3. **Pas de réquisition :** Les ressources déjà détenues ne peuvent être retirées de force à un processus. Elles doivent être explicitement libérées par le processus qui les détient.
4. **Attente circulaire :** Il existe un ensemble de k processus P1, P2, ..., Pk tels que :
   P1 attend une ressource détenue par P2
   P2 attend une ressource détenue par P3
   ...
   Pk attend une ressource détenue par P1

C. Modélisation de l'allocation des ressources par graphe orienté

Cette modélisation a été proposée par Holt en 1972

Types d'arcs et de noeuds du graphe

Le graphe possède deux types de noeuds :

- Des processus, illustrés par des cercles
- Des ressources représentées par des carrés.

Le graphe possède deux types d'arcs:

- Arc de requête : allant d'un processus à une ressource. Signifie que le processus a demandé la ressource et il est en attente de sa libération.
- Arc d'acquisition : allant d'une ressource à un processus. Signifie que le processus détient la ressource.

Pi détient la ressource Ri Pi demande la ressource Ri

Représentation de l'allocation de ressources

1. Cas d'un seul exemplaire de ressource

Exemple

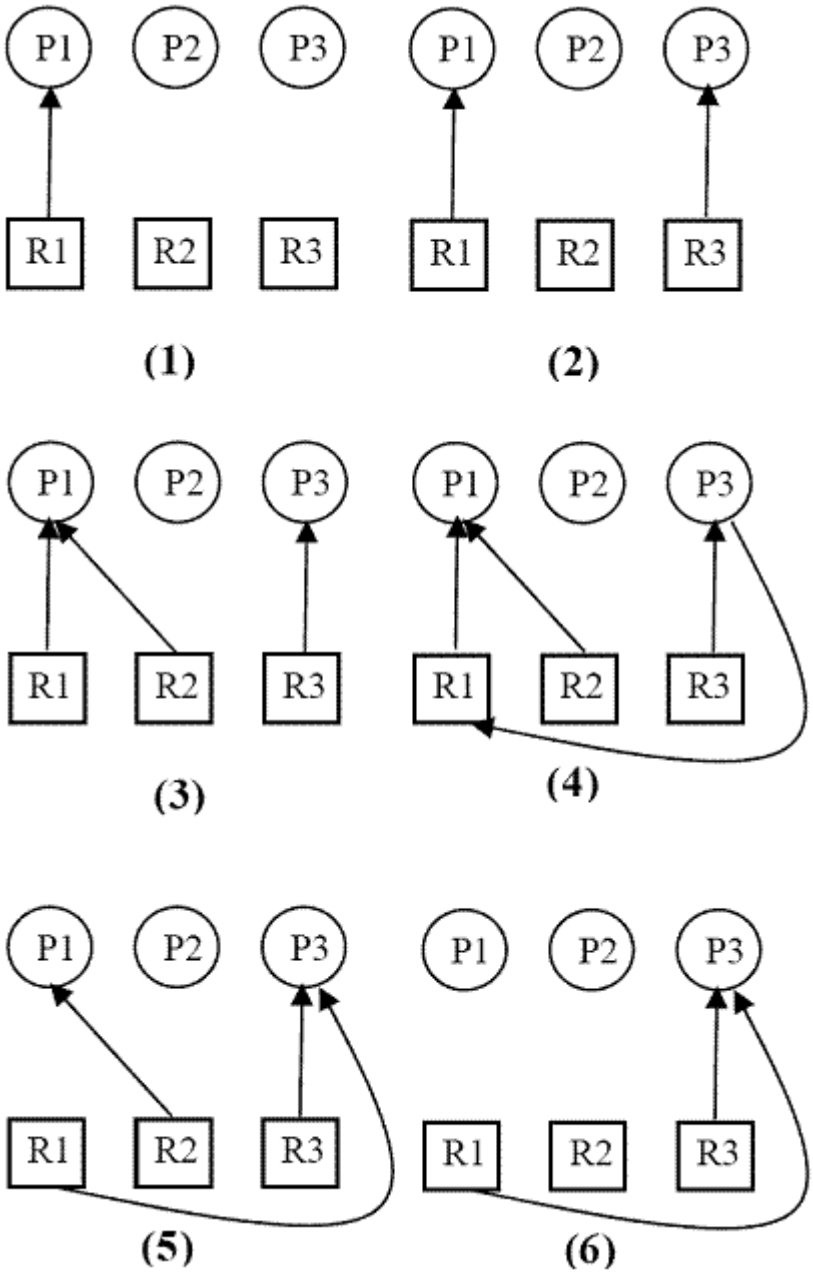
Un système est composé de trois processus P1, P2 et P3 et de trois ressources R1, R2 et R3. Les demandes et les libérations de ressources sont les suivantes :

P1	P2	P3
Demande R1	Demande R2	Demande R3
Demande R2	Demande R3	Demande R1
Libération R1	Libération R2	Libération R3
Libération R2	Libération R3	Libération R1

Demandes de ressources par trois processus

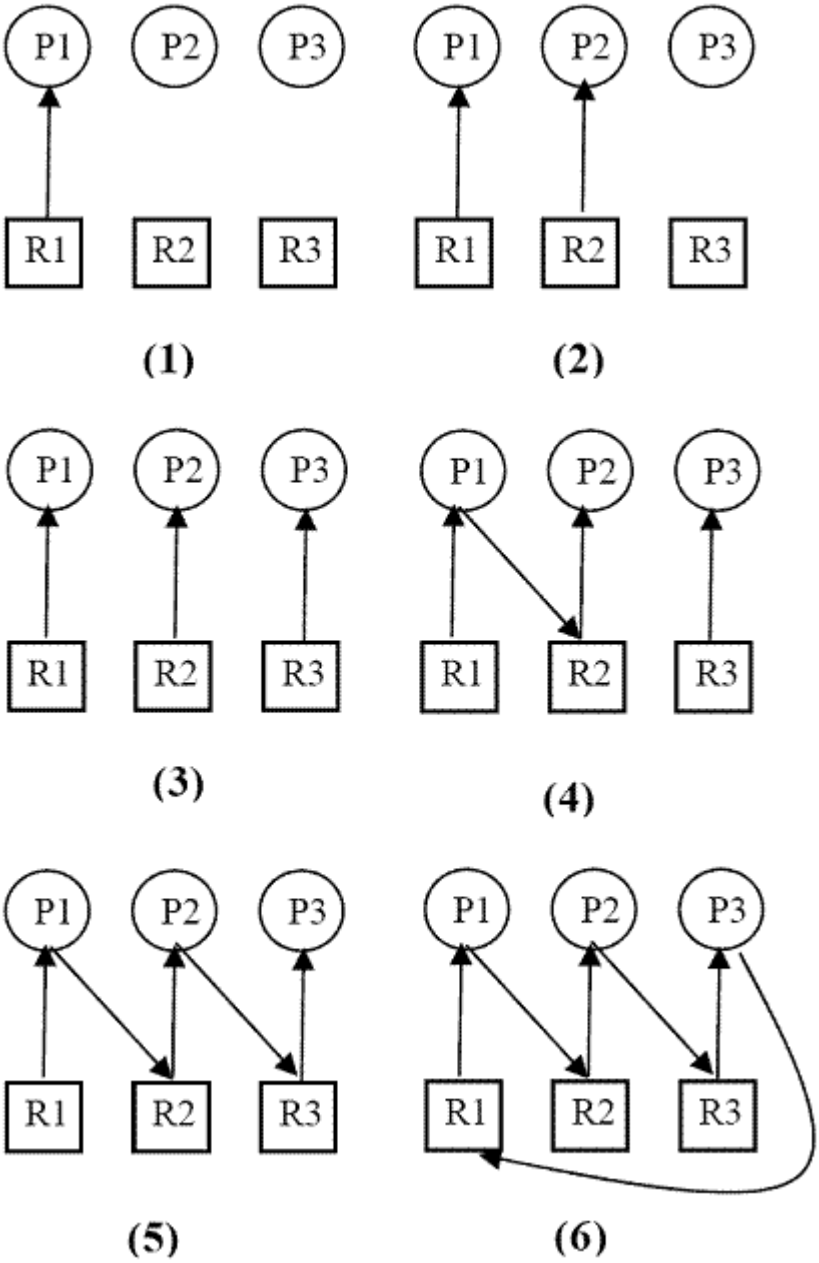
Suivant l'ordre d'exécution des instructions le système peut entrer dans une situation d'interblocage ou non.

Cas1 : pas d'interblocage :



Cas 1 : Pas d'interblocage

Cas 2 : interblocage



Cas d'exécution avec interblocage

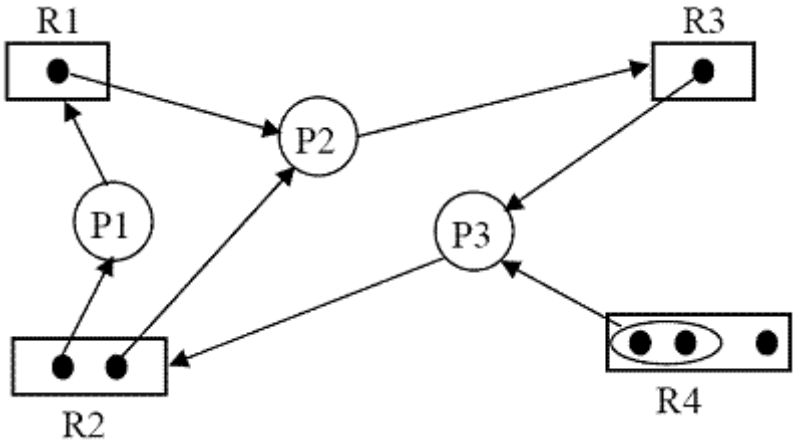
Théorème

Si chaque type de ressource possède exactement un seul exemplaire alors : Il y a situation d'interblocage SSI le graphe d'allocation possède un circuit.

2. Cas de plusieurs exemplaires par type de ressources

Représentation des exemplaires d'une ressource

Dans les carrés représentant les ressources on ajoute des points pour donner le nombre d'exemplaires. Les arcs d'allocation lient désormais les processus aux exemplaires (points) de la ressource.



Représentation des exemplaires d'une ressource

Cycle1 : R2, P2, R3, P3, R2 ==> interblocage

Cycle2 : R2, P1,R1,P2,R3,P3,R2 ==> interblocage.

Théorème

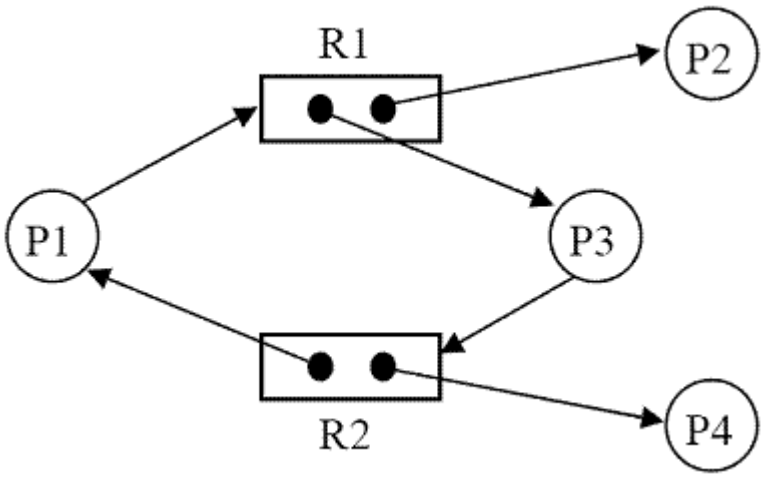
Dans le cas de plusieurs exemplaires par type de ressource : Si le graphe d'allocation est sans circuit alors aucun processus n'est dans une situation d'interblocage.

Attention

Malheureusement dans le cas de plusieurs exemplaires par type de ressource la réciproque n'est pas vraie, en voici un contre-exemple.

Exemple : Circuit sans interblocage

Le système de cet exemple n'est pas dans une situation d'interblocage malgré la présence d'un circuit dans le graphe.



Présence de cycle sans interblockage

Cycle : P1,R1,P3,R2,P1 , mais pas d'interblocage, Pourquoi ? P2 et P4 peuvent terminer et libérer R1 et R2.

Il faut utiliser une autre méthode de modélisation dans le cas de plusieurs exemplaires de ressources.

D. Modélisation de l'allocation de ressources à travers des matrices

Structures de données

Le système est composé de n processus P1, P2, ...,Pn

Et de m classes (types) de ressources R1, R2, ...,Rm

Cette représentation fait intervenir les structures de données suivantes (qui sont une implémentation du graphes d'allocation) :

- **Vecteur des capacités:**

  N : Tableau[1..M] d'entiers

  N[i] = le nombre maximum d'exemplaires de la ressource Ri (disponible au départ)

- **Vecteur des ressources disponibles:**

  DISPONIBLE : Tableau[1..M] d'entiers

  DISPONIBLE[j]=nombre d'exemplaires libres de la ressource Rj

  Au départ ce vecteur est égal au vecteur de capacités N.

- **Matrice d'allocation :**

  ALLOCATION :Tableau[1..N, 1..M] d' entiers;

  ALLOCATION[i,j]=nombre d'exemplaires de la ressource Rj alloués au processus Pi.

- **Matrice des demandes en attente :**

  DEMANDE: Tableau[1..N, 1..M] De Entier ;

  DEMANDE[i,j] = nombre d'exemplaires de la ressource Rj demandés et non obtenus par le processus Pi.

Définition : Blocage

Soit un système S comprenant n processus P1, P2, ... Pn. L'état du système à un instant donné est un interblocage s'il existe un sous-ensemble D non vide de processus tel que, pour tout Pi appartenant à D, l'inégalité :

$$DEMANDE[i] \leqslant DISPONIBLE + \sum_{j \notin D} ALLOCATION[j]$$

est fausse

Chaque processus Pi de D est bloqué.

Remarque : Condition de blocage équivalente

En utilisant l'inéquation :

$$DISPONIBLE = N - \sum_{j=1}^{n} ALLOCATION[j]$$

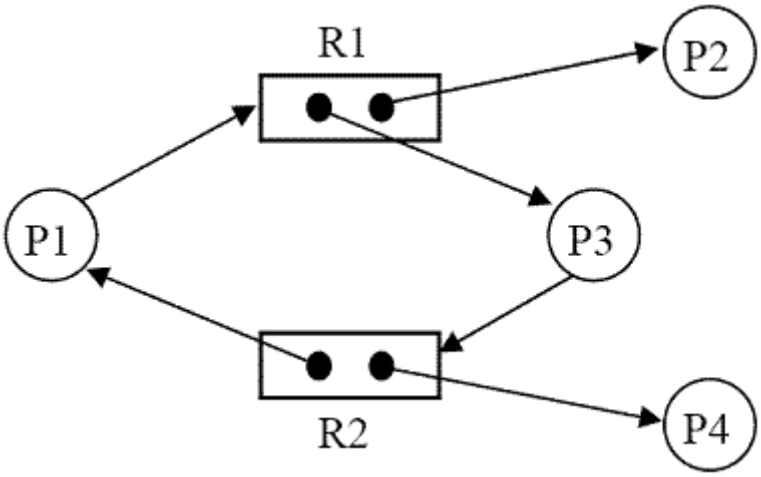
La condition de blocage est équivalente à la suivante :

$$DEMANDE[i] \leqslant N - \sum_{k \in D} ALLOCATION[k]$$

est fausse

Exemple

Soit le graphe d'allocation de ressources suivant :



Graphe d'allocation

Ce graphe peut être traduit dans le modèle matriciel comme suit :

ALLOCATION DISPONIBLE

	R1	R2
P1	0	1
P2	1	0
P3	1	0
P4	0	1

R1	R2
0	0

DEMANDE

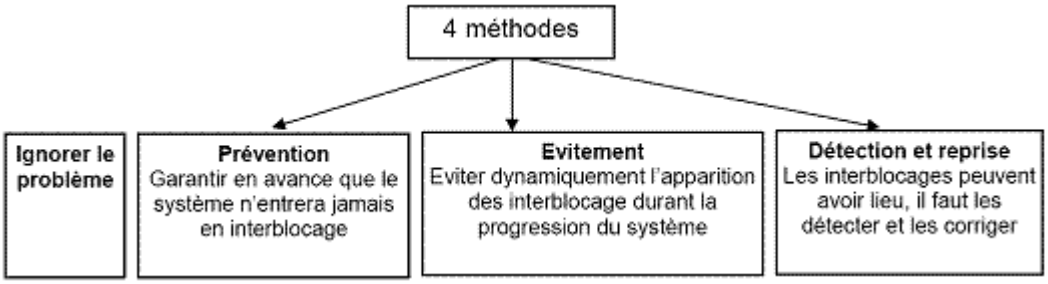
	R1	R2
P1	1	0
P2	0	0
P3	0	1
P4	0	0

Représentation matricielle du graphe précédent

E. Traitement des interblocages

Quatre méthodes de traitement d'interblocages

Il existe quatre approches pour traiter les interblocages :



SOlutions au bropblèmes d'interblocage

F. Ignorer l'interblocage

Définition : Ignorer l'interblocage

Cette méthode consiste à ignorer complètement les interblocages. En fait la plupart des systèmes d'exploitation, y compris UNIX et WINDOWS, se contentent d'ignorer le problème en partant du principe que la plupart des utilisateurs préfèrent être confrontés à un interblocage occasionnel plutôt que d'être soumis à des règles qui limiteraient le nombre de ressources (processus, fichiers, mémoire,..) par utilisateur.

C'est une attitude empirique qui est fondée sur la fréquence constatée du phénomène et sur la gravité des conséquences. Si le système est un ordinateur en multi-programmation qui doit être relancé toutes les semaines en moyenne et que des blocages ne se produisent qu'une fois par mois, ignorer le problème peut être une solution raisonnable.

Exemple : Interblocage sous UNIX

Sur unix, si on essaie de créer des process par des fork() depuis plusieurs process, ceci alors que la table des process est presque pleine, les demandeurs vont boucler en recevant chaque fois le status "création impossible". Les auteurs du système ont considéré que cette situation était exceptionnelle et qu'il était préférable de ne pas alourdir la création de process dans le cas général pour traiter en plus ce cas particulier.

Exemple : Pilote automatique d'un avion

Si l'application concernée est, par exemple, le contrôle du pilote automatique d'un avion, il est probable que l'on choisira l'option inverse : écrire plus de code, mais refuser qu'il puisse se produire un blocage quelles que soient les circonstances.

G. Prévention de l'interblocage

Pour garantir que les interblocages ne se produisent jamais d'une manière définitive, il suffit que l'une des 4 conditions nécessaires pour l'occurrence d'un interblocage ne soit jamais vérifiée. Examinons chaque possibilité:

Prévenir l'exclusion mutuelle

Il n'est pas possible d'éviter l'EM pour prévenir les interblocages. Certaines ressources sont intrinsèquement non partageables.

Prévenir l'occupation et attente

Pour éviter l'allocation partielle il faut garantir qu'à chaque fois qu'un processus demande une ressource il n'en détient aucune autre. Deux solutions sont possibles :

- **Allocation globale :** Un processus ne peut commencer son exécution que s'il dispose de toutes les ressources nécessaires à l'avance. Il ne peut libérer une ressource que si elle ne lui est plus nécessaire.

  Inconvénient : Diminution ou suppression du parallélisme.
- **Allocation par étapes :** Chaque processus ne peut demander des ressources que s'il ne dispose d'aucune ressource. Le processus doit donc demander ses ressources par étape et les libérer à la fin de chaque étape.

  Inconvénient : Mauvaise utilisation des ressources

Prévenir la non préemption

Pour éviter la non préemption deux approches sont possibles :

- Tout processus qui demande des ressources non disponibles dans l'immédiat se bloque mais le système lui retire toutes les ressources qu'il détient déjà.
- Si un processus demande de nouvelles ressources non disponibles le système les retire à d'autres processus bloqués en attente d'autres ressources.

Dans les deux cas le processus auquel on retire des ressources ne peut être relancé que si toutes les ressources (anciennes plus nouvelles) sont disponibles.

Inconvénient : Risque de privation.

Prévenir l'attente circulaire

Pour éviter l'attente circulaire, la solution est d'imposer un ordre total sur l'acquisition des ressources. On affecte à chaque classe de ressource une priorité par une fonction F telle que F(Ri) est un nombre indiquant l'ordre de la classe de ressource Ri.

Règle : Un processus ne peut acquérir une ressource de la classe Ri que s'il a déjà acquis toutes les ressources de classe Rj telles que F(Rj)<F(Ri).

Avantage : Améliorer la gestion des ressources en plaçant les ressources coûteuses dans les classes supérieures

Inconvénient : Priorité statique, difficulté de passage à l'échelle (gérer la numérotation des ressources très nombreuses : enregistrements dans une base de données).

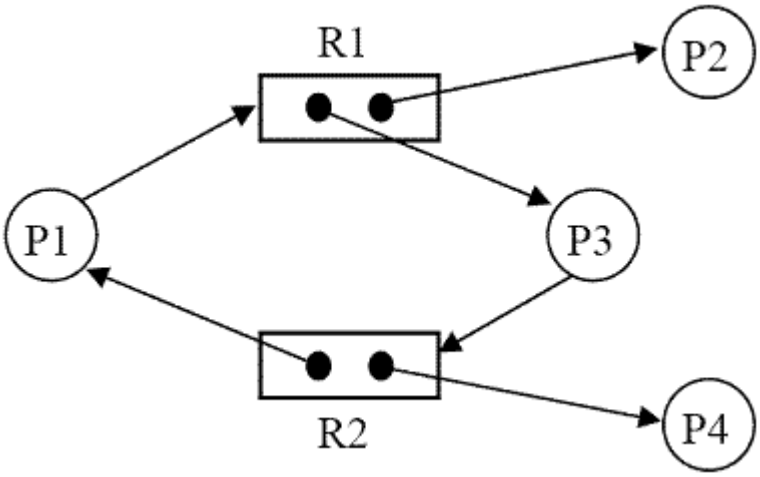
H. Evitement des interblocages

Pour éviter dynamiquement tout interblocage dans un système la solution la plus simple est l'algorithme du Banquier dont le principe est d'évaluer le risque d'interblocage pouvant être provoqué par une demande de ressource. Si une demande présente ce risque le système doit la mettre en attente même si elle peut être satisfaite avec les ressources disponibles.

Définition : Séquence saine

Une séquence de processus P1, P2, ..., Pn est dite séquence saine si pour chaque processus Pi, les demandes de ressources de Pi peuvent être satisfaite par les ressources disponibles plus les ressources détenues par tous les Pj avec j<i.

Exemple : Séquence saine



Séquence saine

Dans ce système, la séquence P4, P3, P2, P1 est une séquence saine.

Définition : Etat sain

Un Système est dans un état sain s'il existe une séquence saine de tous les processus du système.

Remarque : Etat sain vs. état malsain

Un état sain est un état exempt d'interblocage.

Un état malsain (état dans lequel on ne peut trouver une séquence saine) peut conduire à un interblocage.

1. Algorithme du Banquier

Principe

Lorsqu'un processus demande un ensemble de ressources, le système doit déterminer si l'allocation de ces ressources laissera le système dans un état sain. Si c'est le cas, les ressources sont allouées; sinon le processus doit attendre que d'autres processus libèrent suffisamment de ressources.

Structures de données

- Le vecteur DISPONIBLE
- Les matrices ALLOCATION et DEMANDE
- De plus chaque processus doit déclarer le nombre maximal de rescousses de chaque type dont il a besoin :

  MAX : Tableau[1..N, 1...M] d'entiers est la matrice des annonces.

  MAX[i,j]=nombre maximum de rescousses de la classe Rj nécessaires pour l'exécution du processus Pi
- On définit aussi :

BESOIN : Tableau[1..N,1..M] d'entiers;

La matrice des besoins ou des ressources manquantes.

BESOIN = MAX - ALLOCATION

Notations

- X,Y : Tableau [1..N] d'entiers; X<=Y ssi X[i]<=Y[i] pour tout i=1,N
- Mat : Tableau [1..N, 1..M] d'entiers;

  Mat_i désigne la ligne i de la matrice Mat. Ainsi : $ALLOCATION_i$ (resp. $BESOIN_i$) désigne les ressources actuellement allouées au processus Pi (resp. les ressources supplémentaires que le processus Pi peut encore demander pour achever sa tâche).

Algorithme de détermination d'état sain

```
Travail :Tableau[1..M] d'entiers ;
Fini : Tableau[1..N] de Booléen ;
1.  Travail :=DISPONIBLE ;
    Pour i=1 jusqu'à N faire Fini[i]=faux ;
2.  Trouver i tel que Fini[i] = faux et BESOINi <= Travail
    Si i n'existe pas aller à 4
3.  Travail=Travail+Allocationi ;
    Fini[i]=vrai ;
    aller à 2
4. Si Fini[i]=vrai pour tout i Alors le système est dans un
état sain ;
```

Algorithme de requête

```
1. Si DEMANDEi <= BESOINi Alors Aller à 2
    Sinon Erreur : excéder sa demande maximale
2. Si DEMANDEi <=DISPONIBLE Alors Aller à 3
    Sinon Attente
3. /* On suppose que le système a alloué les ressources
demandées par le processus Pi */
    DISPONIBLE = DISPONIBLE - DEMANDEi
    ALLOCATIONi= ALLOCATIONi + DEMANDEi
    BESOINi= BESOINi - DEMANDEi
4. Si cet état est Sain Alors allocation avec succès
    Sinon /* Restaurer l'état initial */
    DISPONIBLE = DISPONIBLE + DEMANDEi
    ALLOCATIONi = ALLOCATIONi - DEMANDEi
    BESOINi = BESOINi + DEMANDEi
```

Exemple

Soit un système comportant 5 processus (P0, P1, P2, P3, P4) et 3 classes de ressources (R0, R1, R2). L'état du système est :

ALLOCATION

	R0	R1	R2
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

MAX

	R0	R1	R2
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

DISPONIBLE

R0	R1	R2
3	3	2

BESOIN = MAX – ALLOCATION

	R0	R1	R2
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

Etat du système

Le système est dans un état sain : la séquence P1, P3, P4, P2, P0 est saine.

P1 fait une nouvelle demande. DEMANDE1 = (1,0,2). DEMANDE1 <=DISPONIBLE puisque (1,0,2)<=(3,3,2)

On suppose que la requête a été satisfaite et on obtient l'état suivant :

ALLOCATION

	R0	R1	R2
P0	0	1	0
P1	3	0	2
P2	3	0	2
P3	2	1	1
P4	0	0	2

BESOIN

	R0	R1	R2
P0	7	4	3
P1	0	2	0
P2	6	0	0
P3	0	1	1
P4	4	3	1

DISPONIBLE

R0	R1	R2
2	3	0

Nouvel état du sytsème

Selon l'algorithme cet état est sain : la séquence P1, P3, P4, P0, P2 est saine. Cette

requête peut être satisfaite.

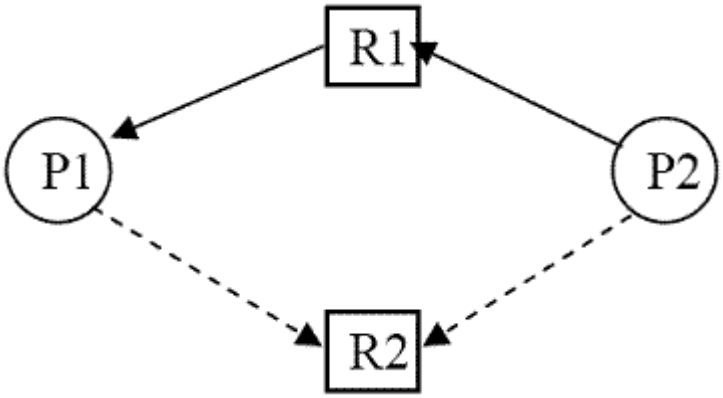
2. Cas d'un seul exemplaire par ressource

Méthode : Recherche d'un cycle dans le graphe d'allocation prévisionnelle

En plus de l'algorithme du banquier, il est possible dans ce cas d'utiliser un algorithme plus simple qui consiste à rechercher un cycle dans un graphe.

Le graphe est une variante du graphe d'allocation dans lequel figurent les arcs qui expriment les besoins futurs des processus. Un arc de besoin est représenté comme un arc de requête en direction mais il est en pointillé. Un arc de besoin (Pi, Rj) indique que le processus Pi peut avoir à demander la ressource Rj. Tous les besoins doivent être déclarés à priori. Ainsi, lorsque Pi commence son exécution, tous ses arcs besoins figurent sur le graphe d'allocation. Lorsque Pi demande Rj, la requête peut lui être attribuée si en convertissant l'arc de requête (Pi, Rj) en un arc d'acquisition (Rj, Pi) aucun cycle ne se forme.

Exemple



Graphe d'allocation prévisionnelle

Dans ce système, si P2 demande R2, un cycle se forme dans le graphe : le système ne va pas attribuer R2 à P2.

I. Détection des interblocage et reprise

Un système qui ne prévoit pas les interblocages doit fournir :

1. Un algorithme qui examine l'état du système pour déterminer s'il s'est produit un interblocage; et
2. Un algorithme pour le corriger.

1. Détection

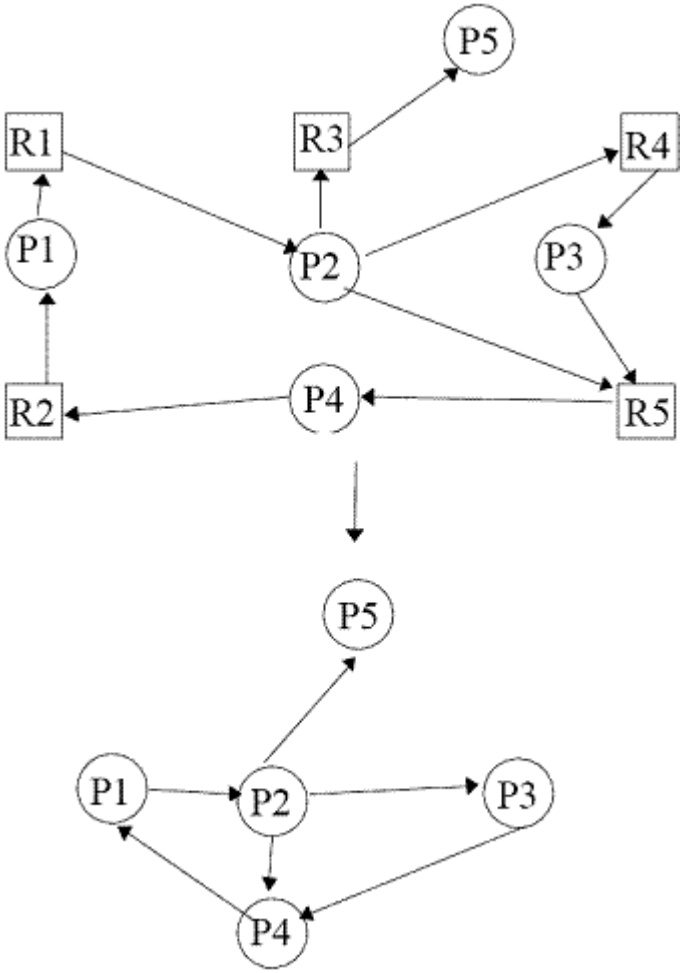
On distingue les deux cas : un seul exemplaire par ressource et plusieurs exemplaires par ressource.

a) Cas à un seul exemplaire par type de ressource

Graphe d'attente

Si toutes les ressources possèdent une seule instance, l'algorithme de détection d'interblocage utilise une variante du graphe d'allocation : le graphe d'attente (Wait for Graph) qui est obtenu en éliminant les noeuds de type ressource et en fusionnant les arcs (requête, affectation) en arcs attente.

Exemple : Graphe d'attente



Transformation d'un graphe d'allocation en un graphe d'attente

Méthode : Graphe d'attente et interblocage

Un interblocage se produit dans le système SSI le graphe d'attente contient un circuit. Le système doit donc maintenir le graphe d'attente et appeler périodiquement un algorithme qui cherche un circuit dans le graphe.

b) Cas de plusieurs exemplaires par type de ressource

Principe

L'algorithmique de détection ressemble à celui du banquier. Elle est dûe à Shoshani et Coffman.

Structures de données

- Le vecteur DISPONIBLE

- Les matrices ALLOCATION et DEMANDE

Algorithme de détection de l'interblocage

```
Travail : Tableau[1..M] d'entiers ;
Fini : Tableau[1..N] de Booléen ;
1. Travail :=DISPONIBLE ;
    Pour i=1 jusqu'à N faire
      Si ALLOCATIONi ≠ 0 Alors Fini[i]=faux ;
      Sinon Fini[i]=vrai ;
2. Trouver i tel que Fini[i]= faux et DEMANDEi <= Travail
    Si i n'existe pas aller à 4
3. Travail=Travail+ ALLOCATIONi;
    /*Pi termine et libère ces ressources */
    Fini[i]=vrai ;
    aller à 2
4. Si Fini[i]=Faux pour certain i Alors le système est dans
un état d'interblocage.
    De plus, chaque processus Pi tel que Fini[i]=Faux est en
situation d'interblocage;
```

Exemple

Soit un système comportant 5 processus (P0, P1, P2, P3, P4) et 3 classes de ressources (R0, R1, R2). L'état du système est :

ALLOCATION

	R0	R1	R2
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

DEMANDE

	R0	R1	R2
P0	0	0	0
P1	2	0	2
P2	0	0	0
P3	1	0	0
P4	0	0	2

DISPONIBLE

R0	R1	R2
0	0	0

Etat du système

En exécutant l'algorithme en trouve que la séquence P0, P2, P3, P1, P4 va donner Fini[i] à vrai pour tout i. Donc le système n'est pas en interblocage

2. Reprise

Défaire un interblocage

Il existe deux méthodes pour défaire un interblocage :

1. **Terminaison de processus :** On parle d'avortement de processus. On peut là aussi :
   - Avorter tous les processus en situation d'interblocage et prendre leur ressource ; ou

- Avorter un processus à la fois, jusqu'à ce que le circuit d'interblocage soit éliminé.

2. **Réquisition de ressources :** Réquisitionner successivement certaines ressources et les donner à d'autres processus jusqu'à ce que le circuit d'interblocage soit défait. Il faut pour cela répondre aux question suivantes:
   - Quelle ressource réquisitionner et quel processus choisir ? (ex : Celui qui s'est exécuté le moins ou celui qui possède très peu de ressources, etc.)
   - Que faire avec le processus qui ne peut plus avancer (parce que on lui a retiré ses ressources) ?
     - Retour arrière : Le ramener à un état sain antérieur et reprendre son exécution (Roll back). Ceci nécessite une sauvegarde périodique des états sain (checkpoints ou journal d'exécution)
     - Avortement : l'avorter et le relancer de nouveau