## Chapitre 3 : INTERBLOCAGE

1. **Introduction** : Dans un système informatique, l'exécution d'un processus nécessite l'utilisation des ressources (*déf chap n°2*). Sachant que le système dispose un nombre limité de ressources et certaines sont non-partageables. L'accès concurrent à ces ressources par les processus risque de créer une situation de blocage.
2. **Définition de l'interblocage (deadlock) :** Un ensemble de processus sont interbloqués si chacun d'eux est bloqué en attente d'un événement qui ne peut être déclenché que par un autre processus de l'ensemble.

   **Exemples :**

   **Exemple1 (*Utilisation croisée de 2 ressources critiques*):**

   | **P1** | **P2** |
   |---|---|
   | **1-** Demande(R1)<br>*<Utilisation R1>*<br>**2-** Demande(R2)<br>*<Utilisation R2>*<br>**3-** Libération (R2)<br>**4-** Libération(R1) | **1-** Demande(R2)<br>*<Utilisation R2>*<br>**2-** Demande(R1)<br>*<Utilisation R1>*<br>**3-** Libération (R2)<br>**4-** Libération(R1) |

   La séquence suivante : 1, 1, 2, 2… induit à un interblocage i.e. P1 attend P2 et P2 attend P1). Les deux processus vont attendre indéfiniment.

   **Exemple2 (Allocation partielle d'une ressource existant en (N) plusieurs exemplaires)**

   Nombre d'unités de ressources disponibles **: Rlibre=4 ;**

   | **P1** | **P2** |
   |---|---|
   | **1-** Demander(3)<br><Obtient 2 unités><br>**2-** Attend 1 unité | **1-** Demander(3)<br><Obtient 2 unités><br>**2-** Attend 1 unité |

   **Rlibre = 0 ;**

3. **Conditions d'interblocage** : Plusieurs conditions sont requises pour qu'il puisse y avoir interblocage :
   a. **Condition1 (*Exclusion Mutuelle*)** : Elle concerne les ressources critiques (un seul exemplaire non partageable) car l'exclusion mutuelle entre processus est nécessaire pour les utiliser.
   b. **Condition2 (*Occupation et attente, Hold and Wait*)** : Un processus qui possède déjà des ressources peut faire de nouvelles demandes d'allocation et donc se bloquer en attente de l'obtention de la nouvelle ressource demandée sans libérer celles qu'il détient.

c. **Condition3 (*Pas de préemption*)** : Elle concerne les ressources non préemptibles[3] (non soumises à réquisition) avec demandes bloquantes. Dans ce cas, une ressource ne peut être retirée à un processus avant qu'il n'ait fini de l'utiliser.
d. **Condition4 (*Attente circulaire*)** : Les processus peuvent être rangés dans une liste circulaire, chaque processus attendant une ressource possédée par le suivant.

4. **Modèles d'interblocage** : Les problèmes de l'interblocage peuvent être classés en une hiérarchie de 5 modèles principaux selon le mode de demande de ressources :
   a. **Modèle ressource simple** : un processus exprime une requête vers au plus une ressource.
   b. **Modèle and** : un processus effectue une requête portant sur plusieurs ressources à la fois. Le processus se bloque jusqu'à l'obtention de toutes les ressources demandées.
   c. **Modèle or** : un processus est bloqué jusqu'à l'obtention d'au moins une des ressources demandées.
   d. **Modèle c(m, n)** : un processus est bloqués jusqu'à l'obtention de n ressources quelconques parmi m ressources.
   e. **Modèle and-or** : c'est une généralisation des modèles **b** et **c**. La requête est une combinaison des opérateurs and et or *(ex : Demande[$R_1$ and ($R_2$ or $R_3$)]).*

Notre étude va se porter sur le ***modèle and***.

5. **Formalisation de l'état d'un système** :
   a. **Modélisation matricielle** :
      1. **Vecteur des ressources maximales** : Nous admettons que le système comprend m classes de ressources ($C_1$ à $C_m$). L'ensemble des ressources existantes peut être représenté par un vecteur : `Rmax = [r1max r2max … rmmax]` dans lequel `rimax` indique le nombre maximale d'exemplaires de la ressource de classe i présente dans le système.
      2. **Vecteurs des ressources disponibles** : Lorsque des allocations ont été éffectuées, le nombre de ressources disponibles est contenu dans un autre vecteur `Available` où `Available[i]` représente le nombre d'unités disponibles de la classe de ressources $C_i$.
      3. **Matrices des unités de ressources allouées** : On dispose, en outre, d'une matrice `Allocate` décrivant les allocations courantes des ressources aux processus dans laquelle `Allocate[i,j]` comptabilise le nombre de ressources de la classe $C_j$ allouées au processus $P_i$.
      4. **Matrices des demandes en attente (requêtes en attentes)** : Les requêtes (demandes) des différents processus sont également décrites par une matrice `Request` dans laquelle `Request[i,j]` désigne le nombre de ressources de la classe $C_j$ demandées et non encore obtenues par le processus $P_i$.

[3] Classes des ressources : Ress réquisitionnables (récupération de la Ress pendant sont utilisation ex : mémoire) et Ress non réquisitionnables (ex : graveur, imprimante…) .

❖ **Notion d'état réalisable** :

L'état d'un système est dit réalisable si les contraintes de cohérence suivantes sont respectées :

1. Le nombre d'unités disponibles est toujours positifs ou nul, et un processus ne peut demander plus de ressources qu'il n'en existe dans le système. À l'initialisation (t=0): `Available = Rmax`.
   La contrainte1 s'écrira : `Available≥[[0]]` et `Request[i,*]≤ Rmax`
2. L'ensemble des ressources détenues par un processus ne peut dépasser l'ensemble des ressources maximales disponibles, et un processus ne peut détenir plus de ressources qu'il n'en a demandé. Dans ce cas la contrainte2 s'écrira : `Allocate[i,*]≤ Rmax` et
   `Rmax≥ Allocate[i,*] + Request[i,*] ≥ Allocate[i,*]`
3. A tout instant, la somme des acquisitions des processus ne peut dépasser le totalité des ressources du système :
   La contrainte3 : $\sum_{i=1}^{n} Allocate[i,*] \leq R\max$

**Remarque** : Le système passe d'un état à un autre par l'intermédiaire de l'une des opérations : Demander, allouer ou libérer comme suit :

- **Demandé** : $Request_i(t+1) = Request_i(t) + Request_i$.
- **Allocation** : $Allocate_i(t+1) = Allocate_i(t) + Allocate_i$ et $Available(t+1) = Available(t) - Allocate_i$.
- **Libération** : $Allocate_i(t+1) = Allocate_i(t) - Allocate_i$ et $Request_i(t+1) = Request_i(t) - Allocate_i$ et $Available(t+1) = Available(t) + Allocate_i$.

**Exemple** :

Un état réalisable peut être décrit par les vecteurs suivants :

`Rmax=[4 2 3 1] et Available=[2 1 0 0]`

Avec les matrices suivantes pour les allocations et les requêtes :

$$Allocate = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix} \qquad Request = \begin{bmatrix} 2 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

b. **Modélisation par graphe** : De même que l'on a représenté l'état du système par un ensemble de vecteurs et matrices, on peut le représenter par un graphe d'état (ou graphe d'allocation de ressources).

Le graphe d'allocation des ressources est un graphe biparti G(V, E) composé de deux types de nœuds et d'un ensemble d'arcs :

- Les processus qui sont représentés par des cercles;
- Les ressources qui sont représentées par des rectangles. Chaque rectangle contient autant de points qu'il y a d'exemplaires de la ressource représentée
- Un arc orienté d'une ressource vers un processus signifie que des unités de la ressource sont allouées au processus
- Un arc orienté d'un processus vers une ressource signifie que des unités de la ressource sont demandées par le processus.

Le graphe indique pour chaque processus les ressources qu'il détient ainsi que celles qu'il demande.

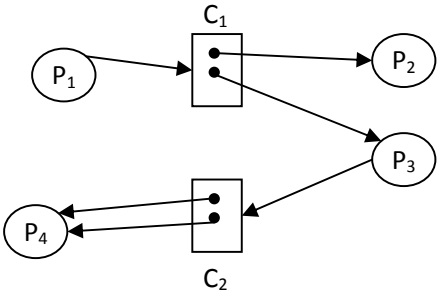
**NB** : Interblocage = trouver un cycle dans le graphe.

**Exemple** :

Soit un système à 2 classes de ressources et 4 processus caractérisé par l'état suivant : `Rmax=[2 2] et Available=[0 0]`

$$Allocate = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad Request = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

Le graphe d'allocation correspondant :



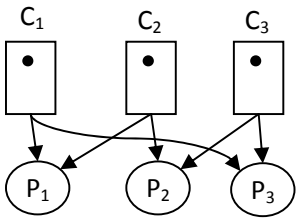

**Exercice1** : Soient trois processus P1, P2 et P3 qui utilisent trois classes de ressources $C_1$, $C_2$ et $C_3$ comme suit :

| **P1** | **P2** | **P3** |
|---|---|---|
| Demander($C_1$) | Demander($C_2$) | Demander($C_3$) |
| Demander($C_2$) | Demander($C_3$) | Demander($C_1$) |
| Libérer($C_1$) | Libérer($C_2$) | Libérer($C_3$) |
| Libérer($C_2$) | Libérer($C_3$) | Libérer($C_1$) |

① Si les processus sont exécutés séquentiellement P1 suivi de P2 suivi de P3, il n'y aurait pas d'interblocage.

② Supposons que l'exécution des processus est gérée par un ordonnanceur circulaire. On atteindrait une situation d'interblocage, si les instructions sont exécutées dans cet ordre :

$P_1$ demande $C_1$ ; $P_2$ demande $C_2$ ; $P_3$ demande $C_3$ ; $P_1$ demande $C_2$ ; $P_2$ demande $C_3$ ; $P_3$ demande $C_1$ ;




Dans ce graphe, $P_1$ dispose d'une instance (un exemplaire) de la classe $C_1$ et demande un exemplaire de la classe de ressources $C_2$, déjà, allouée à $P_2$. Ce dernier attend une ressource de $C_3$, déjà, allouée à $P_3$ et P2 est en attente d'une ressource de $C_1$ allouée à $P_1$.

$P_1$ ----> $C_2$ ---->$P_2$---->$C_3$ ---->$P_3$ ---->$C_1$----> $P_1$.

Un cycle apparaît est aucune ressources des trois classes n'est disponible, donc un ***interblocage***.

**Remarques** :

- Si chaque ressource existe en un seul exemplaire, alors un interblocage existe si le graphe d'allocation des ressources contient un cycle.
- L'existence d'un cycle dans le graphe d'allocation n'est pas une CNS pour détecter les interblocages si une ressource peut exister en plusieurs exemplaires.

**Exercice** :

Supposons un système avec 7 processus et 6 classes de ressources.

Considérons l'attribution des ressources suivante :

$P_1$ détient $C_1$ et demande $C_2$ ; $P_2$ demande $C_3$ ; $P_3$ demande $C_2$ ; $P_4$ détient $C_4$ et demande $C_2$ et $C_3$; $P_5$ détient $C_3$ et demande $C_5$ ; $P_6$ détient $C_6$ et demande $C_2$ ; $P_7$ détient $C_5$ et demande $C_4$.

Construire le graphe d'allocation des ressources. Y a-t-il un interblocage ? Si oui, quels sont les processus concernés ?

- ❖ **Autres représentations** : L'état du système peut être représenté par un graphe (*graphe d'attente*) dont les nœuds représentent les processus et les arcs ($P_i$, $P_j$) qui indiquent que $P_i$ attend au moins une ressource allouée à $P_j$.

6. **Traitements des interblocages** : [os_05_deadlocks, ch6inf26Aut10, C8_interblocage, C9_interblocage ]Plusieurs stratégies peuvent être utilisées face à l'interblocage à savoir : Prévention, évitement ou la détection/guérison[4].

   1. **Stratégies de Prévention** : Le risque d'interblocage survient dès que des processus qui possèdent des ressources peuvent en demander de nouvelles. Si on exclut cette éventualité, il n'y a plus de risque d'interblocage. Trois techniques peuvent être employées en ce sens :
      - ♦ Allouer toutes les ressources en une seule fois. Il n'y a plus d'interblocage, mais des ressources peuvent être immobilisées alors qu'elles ne sont pas réellement utilisées.
      - ♦ Libérer toutes les ressources affectées à un processus avant toute nouvelle demande ; en général, le processus redemande simultanément toutes les ressources qu'il a été forcé de libérer, plus la nouvelle.
      - ♦ Méthode des classes ordonnées
        - ○ Les ressources sont organisées en classes C1, C2, · · · , Cn
        - ○ Dans une classe, les ressources sont allouées en une seule fois
        - ○ Les ressources doivent être demandées dans l'ordre des classes.

        Cette technique est une amélioration de l'allocation en une seule fois ; elle est très utilisée pour l'allocation des ressources matérielles en rangeant les ressources dans l'ordre croissant de leur coût (pour diminuer le coût d'immobilisation). On peut ainsi demander d'abord des périphériques, puis de la mémoire et enfin l'unité centrale.

[4] La guérison de l'interblocage vise à reprendre l'exécution du système dans un état cohérent et sain.

2. **Stratégie d'Évitement**[5] : L'évitement est une méthode d'allocation des ressources avec précaution. Si l'allocation d'une ressource peut conduire à un interblocage, elle est retardée jusqu'à ce qu'il n'y ait plus de risque. En effet, Dans ce cas, lorsqu'un processus demande une ressource, le système doit déterminer si l'attribution de la ressource est sûre (*mènent vers un état fiable*). Si c'est le cas, il lui attribue la ressource, Sinon, la ressource n'est pas accordée.

   Cette méthode impose à tout processus de déclarer ses besoins (annonces) avant son exécution.

   **Définition (état fiable)**: Un état est fiable si tous les processus peuvent terminer leur exécution (il existe un ordre d'allocation de ressources qui permet à tous les processus de se terminer). En effet, il existe une suite fiable de processus **S = $P_1P_2,\ldots,P_n$** tq : rang($P_i$) = i, i=1..n et

$$Annonce[i,*] - Allocate[i,*] \leq Available + \sum_{k=1}^{i} Allocate[k,*]$$

   **Propriétés des suites fiables**

   Lorsque l'état du système est fiable (non intebloqué) alors on peut construire au moins une suite fiable complète de processus.

- **Méthode d'évitement (Algorithme du Banquier) :**

  Un état de système est définie par :
  - *Rmax[1..m]* : Nombre total, initial, de ressources.
  - *Annonce[p,*]* : Nombre maximal de ressources nécessaires au processus p.
  - *Allocate[p,*]* : Nombre de ressources couramment allouées au processus p.
  - Available*[1..m]* : Nombre de ressources disponibles.

**Algorithme Banquier**
**Debut**
(1) Chercher $P_i$ non marqué tel que : **Need[i,*]=(Annonce[i,*] – Allocate[i,*]) ≤ Aavailable**.
(2) Si $P_i$ n'existe pas:
(3) Aller en 7.
(4) **Available ← Available + Allocate[i,*]**.
(5) Marquer $P_i$.
(6) Aller en 1.
(7) Si il reste un processus non marqué:
(8) Il est en situation d'interblocage.
**Fin.**

**Remarque :** Si tous les processus sont marqués, selon l'algorithme du Banquier, alors l'état du système est fiable. Sinon, l'état n'est pas fiable.

[5] Appelée aussi, stratégie de prévention dynamique

**Exemple1** : À instant t donné, l'état du système peut être le suivant :

`Rmax=[10 5 7], Available=[3 3 2]`

$$Annonce = \begin{bmatrix} 7 & 5 & 3 \\ 3 & 2 & 2 \\ 9 & 0 & 2 \\ 2 & 2 & 2 \\ 4 & 3 & 3 \end{bmatrix} \qquad Allocate = \begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

*Question* : Cet état est-il un état faible ?

**Exemple2** :

Considérons un système de 4 processus ($P_1$, $P_2$, $P_3$ et $P_4$) et disposant de 10 unités d'une ressource (**Rmax =[10]**). Le vecteur des annonces est : **Annonce= [6 5 4 7].** Supposons que des allocations aient été effectuées et décrites par le vecteur **Allocate = [1 1 2 4]** ; le nombre de ressources libres est alors **Available = [2]**.

*Question* : Déterminez s'il y a interblocage dans le système (*Cet état est-il un état fiable* ?).

(1) **Need[3,*]=Annonce[3,*]–Allocate[3,*]≤Available.**
**→$P_3$ est marqué, Available=Available+Allocate[3,*]=[4].**

(2) **Need[2,*]=Annonce[2,*]–Allocate[2,*]≤Available.**
**→P2 est marqué, Available=Available+Allocate[2,*]= [5].**

(3) **Need[1,*]=Annonce[1,*]–Allocate[1,*]≤Available.**
**→P1 est marqué, Available=Available+Allocate[1,*]=[6].**

(4) **Need[4,*]=Annonce[4,*]–Allocate[4,*]≤Available.**
**→P4 est marqué, Available=Available+Allocate[4,*]=[10].**

L'état du système est fiable, pas d'interblocage.

3. **Détection/Guérison** : Selon cette approche, aucune mesure préventive n'est prise : on laisse les processus évoluer librement, si une situation d'interblocage est détectée alors une méthode de guérison est appliquée pour remettre le système dans un état cohérent.

**A. Cas de ressources en plusieurs exemplaires :** La méthode se base sur la détection de **l'état sain** du système ; un **état sain** est un état où l'interblocage est impossible.

- **Définition (état sain)** : Le système est dit dans l'état sain si on peut allouer les ressources dans un certain ordre tel que tous les processus du système peuvent s'exécuter jusqu'à leur fin sans intrblocage. D'une manière plus formelle ceci revient à trouver une suite S dite saine et complète de processus **$S = P_1P_2,\ldots,P_n$** tq : rang($P_i$) = i, i=1..n, suite ordonnée contenant les n processus du système et telle que pour tout processus Pi de cette suite les requêtes de Pi en attente peuvent être totalement satisfaites en prenant l'ensemble des ressources disponibles et des ressources détenues par l'ensemble des processus qui précèdent Pi (de rang infèrieur à rang(Pi)). Donc cette propriété s'écrira pour tout processus $P_i$ de rang i dans S (avec Card(S)=n) :

$$Request[i,*] \leq Available + \sum_{rang\ j=1}^{i-1} Allocate[j,*]$$

- **Propriétés des suites saines :**
  - Lorsque l'état du système est sain (non interbloqué) alors, on peut construire au moins une suite saine complète de processus.
  - Lorsque l'état du système est sain (non interbloqué) alors toute suite saine partielle (incomplète) peut être prolongée en une suite saine complète de processus.
- **Conséquences** : Pour détecter un interblocage dans un système, il suffit de réaliser un algorithme permettant de construire une suite saine de processus S. Si la suite S est complète (Card(S) = n) alors le système est dans un état sain et il n'y pas d'interblocage. Si S est incomplète et ne peut plus être prolongée en une suite saine complète alors l'état n'est pas sain et il y a interblocage : dans ce cas si Card(S) = k<n alors il y a (n-k) processus interbloqué.
- **Équivalence dans le graphe d'allocation de ressources** :
  Un processus est dit réducteur du graphe si toutes ses requêtes peuvent être satisfaites, i.e. si tous ses arcs de requêtes en attente peuvent être transformés en arcs d'allocation. Dans le graphe d'état, une suite saine de processus est une suite de processus réducteurs du graphe. L'état sera sain si la suite processus réducteurs est complète (contient tous les processus du graphe) i.e. le graphe est totalement réduit. Dans le cas contraire (graphe partiellement réduit) on n'obtient que k processus réducteurs (k<n) alors les (n-k) processus restants (non réducteurs du graphe) sont donc interbloqués.
- **Méthode de détection (Algorithme TestSain)** :
  Un état de système est définie par :
  - *Rmax[1..m]* : Nombre total, initial, de ressources.
  - *Request[p,*]* : Nombre de ressources demandées et non encore obtenues par le processus p.
  - *Allocate[p,*]* : Nombre de ressources couramment allouées au processus p.
  - Available*[1..m]* : Nombre de ressources disponibles.

**Algorithme TestSain**
**Debut**
(1) Chercher $P_i$ non marqué tel que : **Request[i,*] ≤ Available**.
(2) si $P_i$ n'existe pas:
(3) aller en 7.
(4) **Available ← Available + Allocate[I,*]**.
(5) Marquer $P_i$.
(6) Aller en 1.
(7) si il reste un processus non marqué:
(8) il est en situation d'interblocage.
**Fin.**

**Remarque :**

- Si tous les processus sont marqués, selon l'algorithme de TestSain, alors l'état du système est sain. Sinon, il y a interblocage et tous les processus non marqués sont interbloqués.
- La compléxité de cet algorithme est polynomiale : O ($mxn^2$) où m : nombre de classe de ressource, n : nombre de processus.

**Exemple** : Soit un système à 2 classes de ressources et 4 processus caractérisé par l'état suivant : **Rmax = [2 2] , Available = [0 0]**

$$Allocate = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad Request = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

**Question :** *Cet état est-il un état sain* ?

(1) : **Request[2,*] ≤ Available :**
**→marqué $P_2$ : Available = Available + Allocate[2,*].**

(2) : **Request[4,*] ≤ Available :**
**→marqué $P_4$ : Available = Available + Allocate[4,*].**

(3) : **Request[3,*] ≤ Available :**
**→marqué $P_3$ : Available = Available + Allocate[3,*].**

(4) : **Request[1,*] ≤ Available :**
**→marqué $P_1$ : Available = Available + Allocate[1,*].**

Tous les processus sont marqués, l'état du système est sain et donc pas d'interblocage.

**B. Cas de ressources en un seul exemplaire** : Dans ce cas, pour réduire le nombre d'opérations de l'algorithme de test de l'état sain, on utilise une variante du graphe d'état appelée ***graphes des attentes***. Ce dernier graphe est obtenu en éliminant du graphe d'allocation de ressources tous les sommets de type ressource et en ne gardant que les sommets de type processus. Ensuite, on effectue la fusion des arcs appropriés :

($P_i$, $R_k$) et ($R_k$, $P_j$) ----> ($P_i$, $P_j$) : ceci indique que $P_i$ attend une ressource détenue par $P_j$.

- **Méthode de détection** :
  1. Construire le graphe des attentes,
  2. Détecter un interblocage revient à rechercher au moins un cycle (circuit) dans le graphe des attentes. L'absence de cycle indique un état sain (pas d'interblocage) ; sinon, les processus interbloquéq sont ceux formant le cycle.

**Remarque :** La complexité de l'algorithme devient maintenant : O($n^2$)

- **Mise en œuvre de la détection** : Le premier problème qui se pose est de choisir quand lancer l'algorithme de détection d'interblocage.

  La réponse à cette question dépend de deux facteurs :

  - La fréquence d'apparition des interblocages.
  - Le nombre moyen de processus interbloqués.

  Il faut trouver un critère permettant d'éviter un coût trop élevé pour le système, i.e. qui évite de trop pénaliser le rendement du système (temps moyen d'exécution d'un processus).

  Les principaux inconvénients de la détection sont :

  - L'overhead ou proportion de temps importante prise par le système pour l'entretien des graphes et la détection de l'interblocage (le graphe doit être mis à jour à chaque allocation ou libération de ressource).
  - Le nombre de cycles des graphes peut être important.

- **Guérison de l'interblocage:** La guérison de l'interblocage vise à reprendre l'exécution du système dans un état cohérent et sain. Ceci ne peut être fait qu'en récupérant des ressources déjà allouées : il y aura donc réquisition ou retrait forcé appliquée à ces ressources. Ceci entraine une transformation de ces ressources en ressources préemptibles (sauvegarde de copies totales et/ou partielles éventuelles des ressources). Une fois le problème de la préemption est résolu, il reste 3 problèmes à résoudre :

  1. **Choix du processus victimes** : le choix doit porter sur le processus qui engendre un cout minimal en doit tenir compte de priorité des processus, leur évolution (temps écoulé, temps restant, ressources utilisées et ressources restantes).
  2. **Problème de retour arrière (Roll Back) :** il faut effectuer une « marche arrière » au processus victimes i.e. le réexécuter à partir d'un état cohérent antérieur à celui où il était (cet état cohérent doit être sain). Généralement, pour éviter de recommencer complètement l'exécution du processus victime on utilise souvent la technique des points de reprise « check points ». Le système mémorise (généralement sur disque) un cliché complet de l'espace mémoire occupé par le processus et de l'état des ressources utilisées par le processus de façon à pouvoir reprendre son exécution dans un état cohérent. Ce qui nécessite un espace mémoire considérable.
  3. **Problème de privation (Famine):** La privation concernée ici est le rsique de choisir trop fréquemment le même processus victimes. Diverses méthodes ont été proposées pour y remédier :
     - Techniques du privilège circulant ou jeton qui protège le processus qui le détient contre la destruction forcée.
     - Utilisation d'un facteur de coût croissant avec le nombre de destructions forcées : on choisira donc comme victime le processus qui a le plus petit facteur de coût.

## 4. Conclusion : Approche combinée de traitement de l'interblocage

Le problème de choisir entre les méthodes de traitement de l'interblocage ne peut se résoudre qu'en évaluant le coût (rendement du système) engendré par l'adoption de chacune de ces approches en tenant compte des facteurs : fréquence des interblocages, nombre de processus interbloqués. En fait, aucune approche n'est totalement satisfaisante pour résoudre à elle seule tous les cas d'interblocage.

L'expérience prouve qu'il est en général meilleur de combiner plusieurs méthodes de préventions et de détection/guérison pour résoudre le problème de l'interblocage dans le système. Il est donc recommandé de partitionner ce problème en plusieurs sous-problèmes et de trouver pour chacun d'eux la méthode la plus adéquate.

On peut illustrer cette démarche sur l'exemple suivant :

Soit un système partitionné en 4 sous-ensembles de ressources :

1. Ressources internes utilisées par le système pour gérer les processus (ex : PCB, structures représentatives internes,…)
2. Mémoire centrale.
3. Ressources communes aux travaux (périphériques, fichiers,…).
4. Espace réservé au SWAP[6] en mémoire secondaire.

On associé à chaque sous-ensemble une méthode spécifique de traitement de l'interblocage :

1. **Sous-ensemble1** : prévention par ordonnancement des ressources internes (classes ordonnées)
2. **Sous-ensemble2** : prévention par préemption (éviter la condition 3 d'interblocage par SWAP).
3. **Sous-ensemble3** : prévention dynamique (algorithme du Banquier)
4. **Sous-ensemble4** : prévention par allocation globale (éviter la condition 2 d'interblocage par pré-allocation d'espace en mémoire secondaire pour chaque job en déclarant une taille maximale).

[6] SWAP : opération de transfert de contenu entre deux types de mémoires (ex : stocker le contenu non actif de la RAM dans le disque dur (mémoire virtuelle))