CHAPITRE 3 : LE PROBLEME DE L'INTERBLOCAGE

Le but de ce chapitre est de présenter le problème de l'interblocage (deadlock) : les conditions de son apparition, et les différentes méthodes qui ont été proposées pour le traiter.

3.1 PRESENTATION :

Un système informatique possède un nombre fini de ressources qui doivent être distribuées parmi un certain nombre de processus concurrents. Les ressources sont groupées en plusieurs types, lesquels peuvent exister en plusieurs instances identiques. L'espace mémoire, le processeur, les périphériques sont des exemples de types de ressources. Par exemple, si un système a 2 processeurs, on dira que le type de ressource processeur possède 2 instances, et si le système est doté de 5 imprimantes, on dira que le type de ressource imprimante possède 5 instances.

Dans des conditions normales de fonctionnement, un processus ne peut utiliser une ressource qu'en suivant la séquence suivante :

Requête – Utilisation - Libération

- *La requête* : le processus fait une demande pour utiliser la ressource. Si cette demande ne peut pas être satisfaite immédiatement, parce que la ressource n'est pas disponible, le processus demandeur se met en état attente jusqu'à ce que la ressource devienne libre.
- *Utilisation* : Le processus peut exploiter le ressource.
- *Libération* : Le processus libère la ressource qui devient disponible pour les autres processus éventuellement en attente.

Définition de l'interblocage :

Un ensemble de processus est dans une situation d'interblocage si chaque processus de l'ensemble attend un événement qui ne peut être produit que par un autre processus de l'ensemble.

Exemple : Un système possède une instance unique de chacun des deux types de ressources R1 et R2. Un processus P1 détient l'instance de la ressource R1 et un autre processus P2 détient l'instance de la ressource R2. Pour suivre son exécution, P1 a besoin de l'instance de la ressource R2, et inversement P2 a besoin de l'instance de la ressource R1. Une telle situation est une situation d'interblocage.

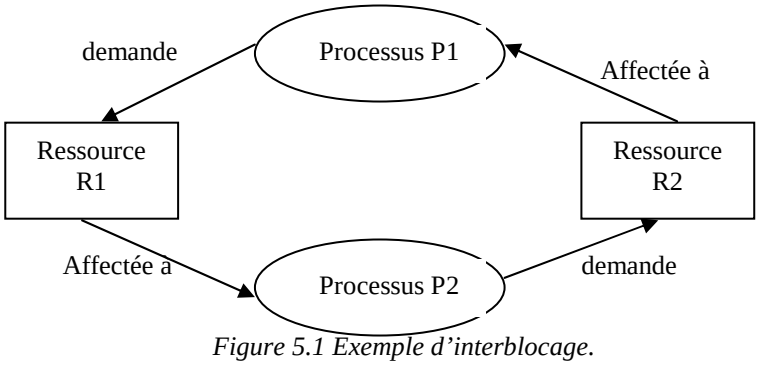


Figure 5.1 Exemple d'interblocage.

Les interblocages sont évidemment indésirables. Dans un interblocage, les processus ne terminent jamais leur exécution et les ressources du système sont immobilisés, empêchant ainsi d'autres travaux de commencer.

Une situation d'interblocage peut survenir si les quatre conditions suivantes se produisent simultanément (Habermann) :

1. *Accès exclusif* : Les ressources ne peuvent être exploitées que par un seul processus à la fois.
2. *Attente et occupation* : Les processus qui demandent de nouvelles ressources gardent celles qu'ils ont déjà acquises et attendent la satisfaction de leur demande
3. *Pas de réquisition* : Les ressources déjà allouées ne peuvent pas être réquisitionnées.
4. *Attente circulaire* : Les processus en attente des ressources déjà allouées forment une chaîne circulaire d'attente.

Définition de la famine :

On dit qu'un processus est dans une situation de famine (starvation) s'il attend indéfiniment une ressource (qui est éventuellement occupée par d'autre processus).

Notons que l'interblocage implique nécessairement une famine, mais le contraire n'est pas toujours vrai.

3.2 GRAPHE D'ALLOCATION DES RESSOURCES :

On peut décrire l'état d'allocation des ressources d'un système en utilisant un graphe. Ce graphe est composé de N nœuds et de A arcs.

L'ensemble des nœuds est partitionné en deux types :

- P={P1, P2, …, Pm} : l'ensemble de tous les processus
- R={R1, R2, …, Rn} l'ensemble de tous les types de ressources du système

Un arc allant du processus Pi vers un type de ressource Rj est noté Pi→Rj ; il signifie que le processus Pi a demandé une instance du type de ressource Rj. Un arc du type de ressourcé Rj vers un processus Pi est noté Rj→Pi ; il signifie qu'une instance du type de ressource Rj a été alloué au processus Pi.

Un arc Pi→Rj est appelé arc de requête. Un arc Rj→Pi est appelé arc d'affectation.

Graphiquement, on représente chaque processus Pi par un cercle et chaque type de ressource Rj comme une rectangle. Puisque chaque type de ressource Rj peut posséder plus d'une instance, on représente chaque instance comme un point dans le rectangle.

Un arc de requête désigne seulement le rectangle Rj, tandis que l'arc d'affectation doit aussi désigner un des points dans le rectangle.

Quand un processus Pi demande une instance du type de ressource Rj, un arc de requête est inséré dans le graphe d'allocation des ressources. Quand cette requête peut être satisfaite, l'arc de requête est instantanément transformé en un arc d'affectation. Quand plus tard, le processus libère la ressource l'arc d'affection est supprimé.

Exemple : L'état d'allocation d'un système est décrit par les ensembles suivants :

- Ensemble des processus P={P1, P2, P3}
- Ensemble des ressources R={R1, R2, R3, R4}
- Ensemble des arcs A={P1→R1, P2→R3, R1→P2, R2→P2, R2→P1, R3→P3}

Le nombre d'instances par ressources est donné par ce tableau :

Type de ressources	Nombre d'instances
R1	1
R2	2
R3	2
R4	3

Voici le graphe d'allocation des ressources associé à ce système :

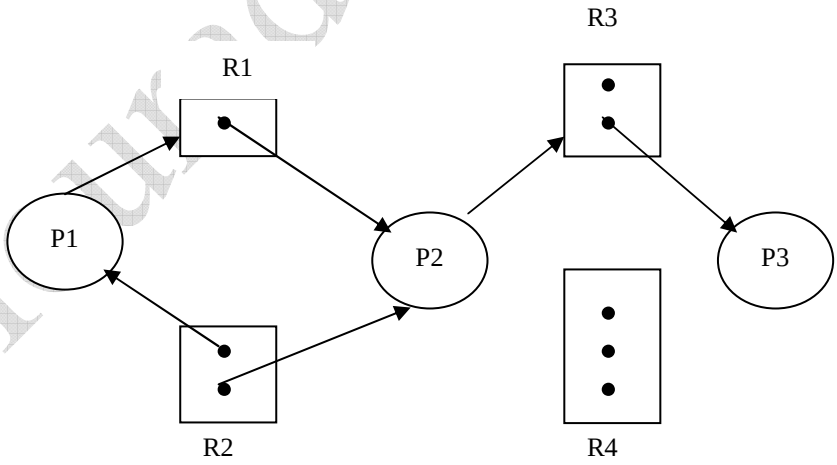


Figure 5.2 Graphe d'allocation des ressources.

On peut faire la lecture suivante sur ce graphe :

- Le processus P1 détient une instance de la ressource R2 et demande une instance de la ressource R1
- Le processus P2 détient une instance de R1 et attend une instance de R3

- Le processus P3 détient une instance de la ressource R3

Si le graphe d'allocation contient un *circuit*, alors il *peut* exister une situation d'interblocage. Considérons les exemples suivants :

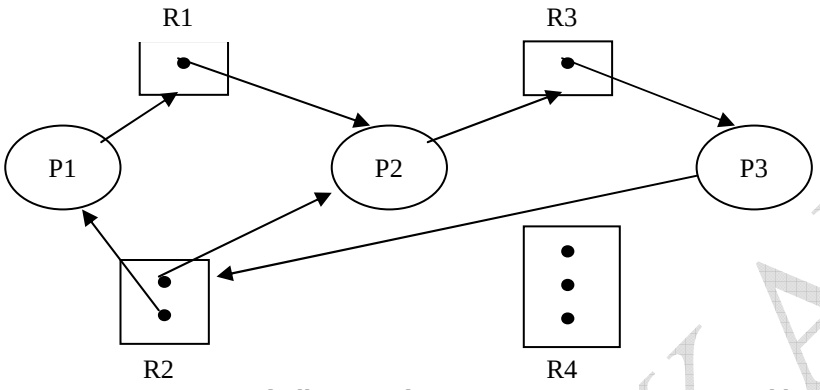


Figure 5.3 Graphe d'allocation des ressources avec circuit et interblocage

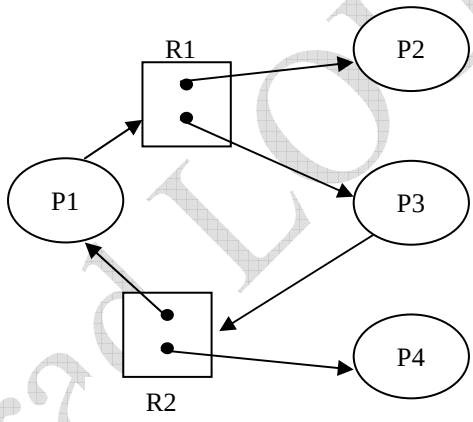


Figure 5.4 Graphe d'allocation des ressources avec circuit sans interblocage

3.3 METHODES DE TRAITEMENT DES INTERBLOCAGES :

Il existe essentiellement trois méthodes pour traiter le problème de l'interblocage : *Prévention, Evitement*, *Détection-Guérison*.

- ***Prévention*** : On élimine complètement le risque d'interblocage en faisant en sorte que l'une des ses quatre conditions d'apparition ne soit pas vérifiée.
- ***Evitement*** : On évite le risque d'interblocage en veillant à que le système évolue uniquement entre états sains.
- ***Détection-Guérison*** : On attend que l'interblocage arrive, on le détecte puis on applique une méthode de guérison.

3.4 PREVENTION DES INTERBLOCAGES :

Les méthodes de prévention des interblocages s'assurent que l'une, au moins, des quatre conditions d'apparition de l'interblocage n'est pas vérifiée.

1. *Exclusion mutuelle* : Un processus ne doit jamais attendre une ressource partageable (exemple : il faut autoriser autant de processus que possible pour la lecture d'un fichier). Cependant, il n'est pas possible de prévenir les interblocages en niant la condition de l'exclusion mutuelle : certaines ressources sont non partageables (exemple : il n'est pas possible de partager un fichier entre plusieurs rédacteurs).
2. *Occupation et attente* : Pour s'assurer que la condition d'occupation et d'attente ne se produit jamais dans le système, on doit garantir qu'à chaque fois qu'un processus qui requiert une ressource, il n'en détient aucune autre. Autrement dit, un processus ne doit demander des ressources supplémentaires qu'après avoir libéré les ressources qu'il occupe déjà.

   Par exemple, imaginons un processus qui copie des données d'une unité de bandes vers un fichier du disque, trie le fichier puis imprime le résultat. Si l'on doit demander toutes les ressources au début du processus, celui-ci doit dès le départ requérir l'unité de bande, le fichier du disque et l'imprimante. Ainsi, il gardera l'imprimante pendant toute l'exécution, même s'il n'en a besoin qu'à la fin. L'autre façon de faire, serait d'affecter au processus uniquement l'unité de bande et le fichier sur disque, au début de son exécution. Il copie de l'unité de bande vers le disque et les libère ensuite tous les deux. Le processus doit à nouveau demander le fichier sur disque et l'imprimante. Après avoir copié le fichier sur l'imprimante, il libère ces deux ressources et se termine.
3. *Pas de réquisition* : Pour garantir que cette condition ne soit pas vérifiée, on peut utiliser le protocole suivant : Si un processus détenant certaines ressources en demande une autre qui ne peut pas lui être immédiatement allouée, toutes les ressources actuellement alloues à ce processus sont réquisitionnées. C'est à dire que ses ressources sont implicitement libérées.

   Les ressources réquisitionnées sont ajoutées à la liste des ressources pour lesquelles le processus attend. Le processus démarrera seulement quand il pourra regagner ses anciennes ressources, ainsi que les nouvelles qu'il requiert.

   Ce protocole présente malheureusement au moins deux inconvénients :

   - Lenteur dans l'utilisation des ressources puisqu'on peut allouer plusieurs ressources et ne pas les utiliser pendant longtemps.
   - La famine est possible puisqu'un processus peut être retardé indéfiniment parce que l'une des ressources qu'il demande est occupée.
4. *Attente circulaire* : On peut garantir que la condition de l'attente circulaire ne se vérifie jamais, en imposant un ordre total sur les types de ressources et on forçant chaque processus à demander les ressources dans un ordre croissant d'énumération. Par exemple, on numérote les types de ressources du système, comme suit :

   - O(Unité de bandes)=1
   - O(Unité de disque)=2
   - O(Imprimante)=3
   - O(Lecteur CD-ROM)=4
   - O(Lecteur Disquette)=5
   - … etc.

   Pour prévenir l'interblocage, on peut imaginer le protocole suivant : Un processus ne peut acquérir une ressource d'ordre i que s'il a déjà obtenu toutes les ressources nécessaires d'ordre j (j<i). Par exemple, un processus désirant utiliser l'unité de bandes et l'imprimante, doit d'abord demander l'unité de bande ensuite l'imprimante. On peut démontrer que de cette façon, il n'y a aucun risque d'interblocage.

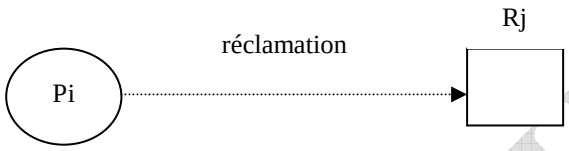
   Cependant cet algorithme est inefficace, il peut conduire à une monopolisation inutile d'une ressource (un processus peut détenir une ressource alors qu'il n'en a pas besoin).

3.5 EVITEMENT DES INTERBLOCAGES :

3.5.1 PRINCIPE :

Si on dispose d'un système d'allocation de ressources avec une seule instance pour chaque type de ressource, on peut utiliser une variante du graphe d'allocation de ressources pour la prévention des interblocages.

En plus des arcs de requêtes et d'affectation, on introduit un nouveau type d'arc appelé *arc de réclamation*. Un arc de réclamation Pi→Rj indique que le processus Pi peut demander la ressource Rj dans le futur (il est représenté en pointillé).

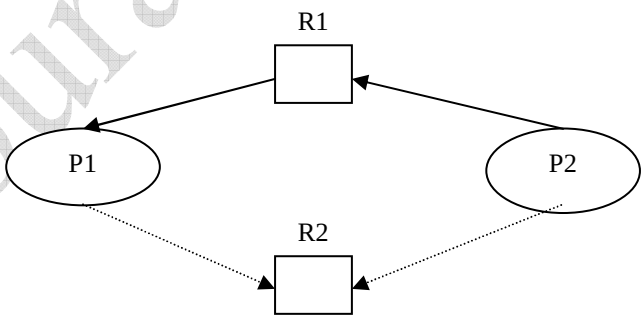


Quand le processus Pi demande réellement la ressource Rj, l'arc de réclamation Pi→Rj est transformé en arc de requête. De même, lorsqu'une ressource est libérée par Pi, l'arc d'affectation Rj→Pi est reconverti en arc de réclamation Pi→Rj.

On peut éviter que l'interblocage arrive en empêchant Pi qui demande la ressource Rj, n'obtienne pas cette ressource si la transformation de l'arc de requête Pi→Rj en arc d'affectation provoque un circuit dans le graphe.

S'il n'existe pas de circuit, l'allocation de la ressource laissera le système dans un état *sain*. Si l'on trouve un circuit, l'allocation de la ressource laissera le système dans un état *malsain*. Le processus Pi devra donc attendre pour que sa requête soit satisfaite.

Exemple : Soit le graphe d'allocation suivant :



Supposons que P2 demande R2. Bien que R2 soit libre, on ne peut l'allouer à P2, puisque cette action créerait un circuit dans le graphe.

3.5.2 ETAT SAIN :

On vient de voir que le principe des méthodes d'évitement de l'interblocage est de garder le système toujours dans un état sain. Nous donnons maintenant une définition formelle de ce qu'est un état sain :

Définition d'un état sain :

Un système est dans un état sain s'il existe une séquence saine. Une séquence de processus <P1, P2, .., PN> est une séquence saine pour l'état d'allocation courant si, pour chaque Pi, les requêtes de ressources de Pi peuvent être satisfaites par les ressources couramment disponibles, plus les ressources détenues par tous les Pj , avec j<i.

En effet, dans cette situation, si les ressources demandées par Pi ne sont pas immédiatement disponibles, Pi peut attendre jusqu'à la terminaison de tous les Pj. Une fois qu'ils ont fini, Pi peut obtenir les ressources nécessaires, achever sa tâche et rendre les ressources allouées. Quand Pi termine, Pi+1 peut obtenir les ressources manquantes, et ainsi de suite. Si une telle séquence n'existe pas, on dit que le système est dans état malsain, c'est à dire qu'il y a un risque d'apparition d'interblocage.

Exemple : un système possède 12 unités de bandes magnétiques et 3 processus P0, P1 et P2. On suppose que les besoins des trois processus en ressources sont: 10 pour P0, 4 pour P1, 9 pour P2. D'autre part, on suppose qu'à l'instant t0 l'état d'allocation des ressources par les processus est le suivant : 5 pour P0, 2 pour P1 et 2 pour P2. Le nombre d'unités de bandes libres est donc égal à 3.

Processus	Besoins maximaux	Besoins actuellement satisfaits	Besoins restant à satisfaire
P0	10	5	5
P1	4	2	2
P2	9	2	7

Nombre de ressources (unités de bandes) disponibles : 03.

A l'instant t0, le système est dans un état sain. En effet, la séquence <P1, P0, P2> est saine, puisque l'on peut allouer immédiatement au processus P1 toutes ses ressources manquantes et les rendre ensuite. Dans ce cas le système disposerait de 5 unités de bandes, le processus P0 peut donc obtenir toutes ses ressources manquantes et les rendre ensuite. Le système disposerait alors de 10 unités de bandes et enfin le processus P2 peut obtenir ses unités de bandes et les rendre, le système disposerait donc de 12 unités de bandes.

Il est possible de passer d'un état sain à un état malsain. Supposons qu'à l'instant t1 le processus P2demande et qu'on lui accorde 1 unité de bandes de plus. L'état du système serait alors le suivant :

Processus	Besoins maximaux	Besoins actuellement satisfaits	Besoins restant à satisfaire
P0	10	5	5
P1	4	2	2
P2	9	3	6

Nombre de ressources (unités de bandes) disponibles : 02.

Le système n'est plus dans un état sain. En effet, on ne peut allouer toutes ses unités de bandes qu'au processus P1. Quand il les rendra, le système ne disposera que de 04 unités de bandes. Comme le processus P0 peut demander 5 unités et le processus P2 7 unités, le système n'aura pas suffisamment de ressources pour les satisfaire. Ou pourra alors avoir un cas d'interblocage, puisque P0 et P2 seront retardés indéfiniment. Pour éviter cette situation, il ne fallait pas accorder à P1 la ressource qu'il a demandée ; il fallait le faire attendre.

3.5.3 ALGORITHME DU BANQUIER :

L'algorithme du banquier (Habermann et Dijkstra) est un algorithme d'évitement des interblocages qui s'applique dans le cas général où chaque type de ressources possède plusieurs instances. Le nom de

l'algorithme a été choisi parce que cet algorithme pourrait s'appliquer dans un système bancaire pour s'assurer que la banque ne prête jamais son argent disponible de telle sorte qu'elle ne puisse plus satisfaire tous ses clients.

Quand un nouveau processus entre dans le système, il doit déclarer le nombre maximal d'instances de chaque type de ressources dont il aura besoin. Ce nombre ne doit pas excéder le nombre total de ressources du système. Au cours de son exécution, quand un processus demande un ensemble de ressources, l'algorithme vérifie si cela gardera toujours le système dans un état sain. Dans l'affirmative la demande est accordée, dans la négative la demande est retardée.

Soient *m* le nombre de types de ressources du système, et *n* le nombre de processus. Pour fonctionner, l'algorithme maintient plusieurs structures de données :

- **Available** : C'est un Vecteur de longueur m indiquant le nombre de ressources disponibles de chaque type. Ainsi, si Available[j]=k, cela veut dire que le type de ressources Rj possède k instances disponibles.
- **Max** : C'est une matrice nxm définissant la demande maximale de chaque processus. Ainsi, Si Max[i, j]=k, cela veut dire que le processus Pi peut demander au plus k instances du type de ressources Rj.
- **Allocation** : C'est une matrice nxm définissant le nombre de ressources de chaque type de ressources actuellement alloué à chaque processus. Ainsi si Allocation[i, j]=k, cela veut dire que l'on a alloué au processus Pi k instances du type de ressources Rj.
- **Need** : C'est une matrice nxm indiquant les ressources restant à satisfaire à chaque processus. Ainsi, si Need[i, j]=k , cela veut dire que le processus Pi peut avoir besoin de k instances au plus du type de ressources Rj pour achever sa tâche.
- **Request** : C'est une matrice nxm indiquant les ressources supplémentaires que les processus viennent de demander. Ainsi, si Request[i, j]=k, cela veut dire que le processus Pi vient de demander k instances supplémentaires du type de ressources Rj.

De ce qui précède, on peut remarquer que :

1. Ces structures de données peuvent varier dans le temps, en taille et en valeur.
2. La matrice *Need* peut être calculée à partir des matrices *Max* et Allocation :

$$Need[i, j]=Max[i, j] – Allocation[i, j]$$

Notations : Pour des raisons pratiques, on utilise les notations suivantes :

- $Allocation_i$: Vecteur désignant les ressources actuellement allouées au processus Pi.
- $Need_i$: Vecteur désignant les ressources supplémentaires que le processus Pi peut encore demander.
- $Request_i$: Vecteur désignant les ressources supplémentaires que le processus Pi vient de demander.
- X<=Y : X et Y sont des vecteurs si X[i]<=Y[i] pour chaque i allant de 1 à n.

L'algorithme du Banquier peut être scindé en deux parties complémentaires : Un algorithme de requête de ressources et un algorithme de vérification si l'état du système est sain.

Nous donnons maintenant le code de chacune des deux parties.

```
Algorithme de requête de ressources
Début
        /* Cet algorithme est appelé à chaque fois qu'un processus fait une demande de ressources
```

Etape 1 : Si $Request_i <= Need_i$
Alors Aller à l'étape 2
Sinon erreur : le processus a excédé ses besoins maximaux
Finsi

Etape 2 : Si $Request_i <= Available$
Alors Aller à l'étape 3
Sinon Attendre : les ressources ne sont pas disponibles.
Finsi

Etape 3 : Sauvegarder l'état du système (les matrices Available, Allocation et Need).
Allouer les ressources demandées par le processus Pi en modifiant l'état du système de la manière suivante :

$Available := Available - Request_i$
$Allocation_i := Allocation_i + Request_i$
$Need_i := Need_i - Request_i$

Si Verification_Etat_Sain=Vrai
Alors L'allocation est validée
Sinon L'allocation est annulée ; Restaurer l'ancien Etat du système
Finsi
Fin.

L'algorithme suivant est une fonction qui renvoie la valeur *Vrai* si le système est dans un état sain, *Faux* sinon.

```
Algorithme Verification_Etat_Sain
Début
        Work : Tableau[m] de Entier ;
        Finish : Tableau[n] de Logique ;

        Etape 1 : Work :=Available
                Finish :=Faux ;

        Etape 2 : Trouver i tel que : Finish[i]=faux et Need_i<=Work
                Si un tel i n'existe pas aller à l'étape 4.

        Etape 3 : Work :=Work + Allocation_i
                Finish[i] :=Vrai
                Aller à l'étape 2

        Etape 4 : Si Finish=Vrai (pour tout i)
                        Alors Verification_Etat_Sain :=Vrai
                        Sinon Verification_Etat_Sain :=Faux
                Finsi
Fin.
```

Remarque : Cet algorithme peut demander mxn^2 opérations pour décider si un état est sain.

Exemple : Un système possède 5 processus (P0, P1, P2, P3, P4) et 3 types de ressources (A, B, C). Le type de ressources A possède 10 instances, le type de ressources B possède 5 instances et le type de ressources C possède 7 instances. A l'instant T0, l'état des ressources du système est décrit par les matrices *Allocation*, *Max* et *Available* suivantes :

Matrice : Allocation

	A	B	C
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

Matrice : Max

	A	B	C
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

Matrice : Available

A	B	C
3	3	2

Le contenu de la matrice Need peut être déduit par calcul, Need=Max – Allocation

Matrice : Need

	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

On peut vérifier que le système est dans un état sain. En effet, la séquence <P1, P3, P4, P2, P0> est saine.

Supposons qu'à l'instant T1 le processus P1 demande une instance supplémentaire du type de ressources A et deux instances du type de ressources C. Nous avons alors : $Request_1$=(1, 0, 2).

Afin de décider si la requête peut être immédiatement accordée, on doit d'abord vérifier que $Request_1$<=Available, ce qui est vrai. On enregistre l'état du système (le contenu des matrices), puis on supposera que la requête a été satisfaite et on arrive à l'état suivant :

Matrice : Allocation

	A	B	C
P0	0	1	0
P1	3	0	2
P2	3	0	2
P3	2	1	1
P4	0	0	2

Matrice : Need

	A	B	C
P0	7	4	3
P1	0	2	0
P2	6	0	0
P3	0	1	1
P4	4	3	1

Matrice : Available

A	B	C
2	3	0

On doit alors déterminer si le nouvel état est sain en appliquant l'algorithme de vérification de l'état sain, ce qui est vrai (la séquence <P1, P3, P4, P0, P2> est saine). On peut donc accorder la demande de ressource faite par P1.

A l'instant T2, une requête (3, 3, 0) arrive du processus P4. Cette requête est immédiatement rejetée parce que les ressources ne sont pas disponibles.

A l'instant T3, une requête (0, 2, 0) arrive du processus P0. Cette requête ne sera pas accordée parce que l'état obtenu est malsain.

Critique de l'algorithme du Banquier :

Bien qu'il a l'avantage d'éviter les interblocages, l'algorithme du banquier a néanmoins quelques inconvénients :

- *Coûteux* : L'algorithme est en effet très coûteux en temps d'exécution et en mémoire pour le système Puisqu'il faut maintenir plusieurs matrices, et déclencher à chaque demande de ressource, l'algorithme de vérification de l'état sain qui demande mxn^2 opération. (m est le nombre de types de ressources et n est le nombre de processus).
- *Théorique* : L'algorithme exige que chaque processus déclare à l'avance les ressources qu'il doit utiliser, en type et en nombre. Cette contrainte est difficile à réaliser dans la pratique.
- *Pessimiste* : L'algorithme peut retarder une demande de ressources dès qu'il y a risque d'interblocage (mais en réalité l'interblocage peut ne pas se produire).

3.6 DETECTION ET GUERISION DES INTERBLOCAGES :

Si un système n'emploie pas d'algorithme pour prévenir les interblocages, il peut se produire une situation d'interblocage. Dans ce cas le système doit fournir :

- Un algorithme qui examine l'état du système pour déterminer s'il s'est produit un interblocage.
- Un algorithme pour guérir (corriger) l'interblocage.

3.6.1 DETECTION DE L'INTERBLOCAGE :

Pour détecter un interblocage dans un système disposant de plus d'une instance pour chaque type de ressources, on peut utiliser l'algorithme de détection suivant, qui s'inspire de l'algorithme du banquier. . Cet algorithme est lancé périodiquement. Il utilise les structures de données suivantes :

- **Available** : C'est un Vecteur de longueur m indiquant le nombre de ressources disponibles de chaque type. Ainsi, si Available[j]=k, cela veut dire que le type de ressources Rj possède k instances disponibles.
- **Allocation** : C'est une matrice nxm définissant le nombre de ressources de chaque type de ressources actuellement alloué à chaque processus. Ainsi si Allocation[i, j]=k, cela veut dire que l'on a alooué au processus Pi k instances du type de ressources Rj.
- **Request** : C'est une matrice nxm indiquant les ressources supplémentaires que les processus viennet de demander. Ainsi, si Request[i, j]=k, cela veut dire que le processus Pi vient de demander k instances supplémentaires du type de ressources Rj.

```
Algorithme Détection d'interblocage
Début
        Work : Tableau[m] de Entier ;
        Finish : Tableau[n] de Logique ;

        Etape 1 : Work :=Available
                Pour i=1 jusqu'à N
                Faire
                        Si Allocation_i<>0
                                Alors Finish[i] :=Faux
                                Sinon Finish[i]:=Vrai
                        Finsi
                Fait ;

        Etape 2 : Trouver un indice i tel que : Finish[i]=Faux et Request_i<=Work
                Si un tel i n'existe pas aller à l'étape 4.

        Etape 3 : Work :=Work + Allocation_i
                Finish[i] :=Vrai
                Aller à l'étape 2

        Etape 4 : Si Finish[i]=Faux  (pour un certain i)
                        Alors Le système est dans un état d'interblocage
                        Sinon Le système n'est pas dans un état d'interblocage
                Finsi
Fin.
```

Exemple : L'état d'allocation des ressources d'un système est donné par le contenu des trois matrices suivantes : *Available, Allocation* et *Request*.

Matrice : Allocation

	A	B	C	D
P0	1	0	1	0
P1	2	0	0	1
P2	0	1	2	0

Matrice : Request

	A	B	C	D
P0	2	0	0	1
P1	1	0	1	0
P2	2	1	0	0

Matrice : Available

A	B	C	D

5	2	3	1

En appliquant l'algorithme de détection , on trouvera que les contenus successifs des matrices Work et Finish sont :

Itération 1 :

Work

5	2	3	1

Finish

Faux	Faux	Faux

Indice choisi i=0

Itération 2 :

Work

6	2	4	1

Finish

Vrai	Faux	Faux

Indice choisi i=1

Itération 3 :

Work

8	2	4	2

Finish

Vrai	Vrai	Faux

Indice choisi i=2

Itération 4 :

Work

8	3	6	2

Finish

Vrai	Vrai	Vrai

Le vecteur Finish a la valeur Vrai, donc le système n'est pas en situation d'interblocage.

Supposons maintenant que le processus P0 fait une demande supplémentaire de 3 instances de la ressource B, et P2 demande en plus 4 instances de la ressource C. Les données du problème deviennent donc :

Matrice : Allocation

	A	B	C	D
P0	1	0	1	0
P1	2	0	0	1
P2	0	1	2	0

Matrice : Request

	A	B	C	D
P0	2	3	0	1
P1	2	0	1	0
P2	2	1	4	0

Matrice : Available

A	B	C	D
5	2	3	1

Une exécution de l'algorithme de détection fera apparaître les contenus suivants :

Itération 1 :

Work

5	2	3	1

Finish

Faux	Faux	Faux

Indice choisi i=1

Itération 2 :

Work

7	2	3	2

Finish

Faux	Vrai	Faux

Le déroulement de l'algorithme s'arrête avec le vecteur Finish contenant deux valeurs fausses : P0 et P2 sont en interblocage.

Critique de l'algorithme de détection :

L'algorithme de détection des interblocages est très coûteux s'il est exécuté après chaque demande de ressources. En effet, il sera aussi coûteux que l'algorithme d'évitement du Banquier. L'idée donc c'est de le lancer périodiquement, mais comment choisir cette période ?.

3.6.2 GUERISON DE L'INTERBLOCAGE :

Il existe plusieurs solutions pour corriger un interblocage si le système détecte qu'il en existe un.

- *Correction manuelle* : Le système alerte l'opérateur qu'il s'est produit un interblocage, et l'invite à le traiter manuellement (en relançant le système par exemple).
- *Terminaison de processus* : On peut éliminer un interblocage en arrêtant un ou plusieurs processus. On peut choisir d'arrêter tous les processus, ou bien de les arrêter un à un jusqu'à éliminer l'interblocage.
- *Réquisition de ressources :* Pour éliminer l'interblocage, en procédant à la réquisition d'une ou plusieurs ressources, en les enlevant à un processus et en les donnant à un autre jusqu'à ce que l'interblocage soit éliminé.