INTERBLOCAGE : NOTIONS et ALGORITHMES

Table des Matières :

1 Préambule

Dans un système multiprogrammé (tâches ou flots) plusieurs tâches (resp flots) peuvent entrer en compétition pour obtenir un nombre fini de ressources ; si, lors de la requête d'une tâche, une ressource n'est pas disponible, la tâche est mise dans l'état « en attente ». Dans l'hypothèse ou la tâche capable de libérer cette ressource est aussi « en attente » parce qu'elle avait sollicité une autre ressource détenu par la première, alors un **interblocage (deadlock)** apparaît.

Objectifs

Montrer les situations qui peuvent aboutir à un interblocage.
Présenter un certain nombre de méthodes qui permettent de se prémunir ou de détecter des situations d'interblocage.

2 Rappels

Un système est composé d'un nombre fini de ressources qui doivent être distribuées sur un ensemble de tâches concurrentes ; ces ressources sont partitionnées en ensemble disjoint selon leur type, composé d'instances identiques : espace mémoire, cœur de processeur, fichiers, disques, imprimantes…

Une requête d'une instance peut être satisfaite par un quelconque élément de l'ensemble. Une tâche émet une requête pour obtenir une ressource, l'utilise, puis la libère. Tout au long de sa vie, une tâche peut nécessiter de nombreuses ressources. Ces opérations de requêtes, utilisation et libération sont réalisées par des appels systèmes : request() and release() pour les périphériques, open() et close() pour les fichiers, allocate() et free() pour la mémoire. D'autres requêtes peuvent concerner des éléments de synchronisation qui ont pour mission de bloquer l'accès à des ressources partagées.

Une table du système gère l'état de chaque ressource : libre ou occupé, l'identification de la tâche concernée, sont ainsi mémorisés. Si la ressource est utilisée par une autre tâche, alors une file d'attente des tâches « en attente » de libération de la ressource est constituée.

Un sous ensemble de tâches est en situation d'interblocage si chaque tâche de l'ensemble est en attente d'un événement qui peut être produit par une autre tâche de cet ensemble. Par exemple, supposons un système disposant de trois imprimantes au total et que trois tâches utilisent chacune une imprimante différente et que chacune des tâches demande une seconde imprimante ; une tâche ne peut être servie que si une imprimante est libérée ce qui ne peut être possible puisque toutes les imprimantes disponibles du système ont été affectées. Cet interblocage concerne des ressources de même type.

De même supposons deux tâches, l'une utilisant une imprimante, l'autre un disque ; si la première demande le disque et la seconde l'imprimante au cours de leur exécution alors un interblocage avec des ressources différentes apparaît.

Ce problème est crucial en application multiflot car, par construction, les flots sont concurrents sur des ressources partagées.

3 Caractérisation de l'interblocage

3.1 Conditions nécessaires

Un interblocage peut se produire sur la conjonction de quatre circonstances ; réciproquement, si l'on peut assurer que l'une de ces circonstances ne peut se produire alors il n'y aura pas interblocage.

1) Chaque tâche requiert une ressource de façon exclusive ; la ressource ne peut être partagée entre plusieurs tâches (ex. une imprimante) ; si une autre tâche tente de prendre cette ressource alors elle sera suspendue tant que la ressource ne sera pas libérée.
2) Chaque tâche ne peut progresser que si elle obtient les ressources requises dynamiquement par des demandes successives ; une tâche peut donc posséder des ressources et être suspendue en attente d'une ressource utilisée par une autre.

3) Une tâche ayant obtenu une ressource la conserve jusqu'à ce qu'elle la libère : il ne peut y avoir préemption de ressources au profit d'une autre.
4) Des tâches sont en attente de ressources allouées à d'autres tâches en formant des dépendances qui conduisent à un cycle : (P0, P1, P2, P3 …Pn, P0) P0 est en attente d'une ressource détenue par P1 qui est en attente d'une ressource détenue par P2 et en fin de liste Pn est en attente d'une ressource détenue par P0.

Ces quatre conditions doivent être réunies pour qu'un interblocage survienne à un instant donné.

Si les tâches sont en attente de ressources, bloquées par l'allocateur, le système est en interblocage passif (deadlock), alors que si les tâches sont en attente sur une itération qui teste un indicateur de disponibilité, le système est en interblocage actif (livelock) ou « étreinte fatale ».

3.2 Modélisation du problème

Les interblocages peuvent être décrits par un graphe orienté composé d'arcs et de sommets. Les sommets sont de deux types :

- P = {P1, P2, …Pn} l'ensemble des tâches présentes dans le système
- R = {R1,R2,…,Rm} l'ensemble des types de ressources du système.
- Pi → Rj est un arc orienté indiquant que la tâche Pi a demandé une instance de la ressource Rj et qu'elle est attente de cette ressource. (arc requête)
- Rj → Pi est un arc orienté indiquant que la ressource Rj a été allouée à la tâche Pi . (arc affectation)
- Les points représentent différentes instances du même type (le type R2 a deux instances de ressources).

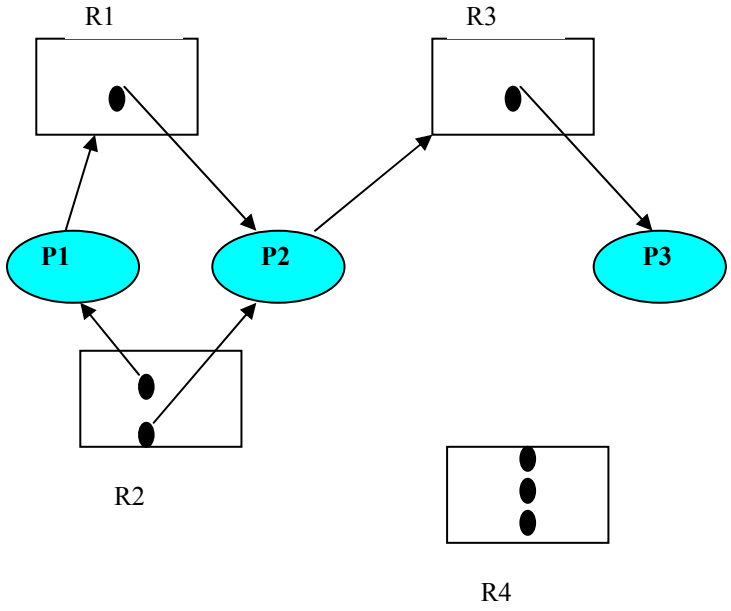

Fig1 : Graphe d'allocation de ressources

Le graphe de la figure 1 traduit :

- La tâche P1 possède une instance de la ressource R2 et est en attente d'une instance d'une ressource de type R1
- La tâche P2 possède une instance de la ressource R1 et une instance de la ressource R2 et est en attente d'une instance d'une ressource de type R3
- La tâche P3 possède l'instance de la ressource de type R3

Il peut être montré que si le graphe ne contient pas de cycle, alors aucune tâche ne sera en interblocage. Si le graphe contient un cycle alors un interblocage peut survenir. Si chaque type de ressource a une seule instance, alors un cycle indique qu'un interblocage s'est produit. Chaque tâche impliquée dans le cycle est bloquée. Dans ce cas, un cycle est une condition nécessaire et suffisante pour l'apparition d'un interblocage.

Ce n'est pas le cas, si plusieurs ressources d'un type donné sont disponibles : un cycle n'entraîne pas forcément un interblocage. C'est une condition nécessaire mais pas suffisante.

En résumé : si un graphe n'a pas de cycle, alors il ne peut y avoir d'interblocage ; si un cycle existe alors il peut ou non y avoir interblocage en fonction du nombre d'instances disponibles de chaque ressource impliquée dans le graphe.

4 Méthodes pour traiter les interblocages

- Utiliser un protocole pour prévenir ou détecter les interblocages, assurant que le système ne sera jamais bloqué
- Etre capable de sortir d'un état provoquant un interblocage
- Simplement ignorer le problème (ce que de nombreux systèmes font en laissant à l'utilisateur le soin de gérer les interblocages !)

Pour s'assurer qu'un interblocage ne peut se produire, le système peut soit prévenir (**prévention de l'interblocage**) en s'assurant qu'au moins une des quatre conditions nécessaires ne peut se produire soit éviter (**évitement de l'interblocage**) en gérant en avance les ressources qu'une tâche doit utiliser pendant son exécution. Avec des connaissances complémentaires, le système peut décider pour chaque requête si la tâche doit être suspendue ou non. Pour décider si la requête courante peut être satisfaite ou retardée, le système doit considérer les ressources actuellement disponibles, les ressources allouées à chaque tâche, et les prochaines requêtes et libérations effectuées par chaque tâche.

Si le système ne propose ni mécanisme de prévention ni mécanisme d'évitement, alors il doit fournir des algorithmes de détection et de reprise capable de remettre les applications dans un état de bon fonctionnement. Si ce n'est pas, il ne restera plus que …Ctr-Alt-Del ou pire !

4.1 Prévention de l'interblocage

Il suffit qu'une des quatre conditions initiales ne puisse se produire :

4.1.1 Exclusion mutuelle

Concerne des ressources non partageables : une imprimante par exemple, dès qu'elle est allouée à une tâche, doit rester allouer jusqu'à la fin de l'impression. Cette condition est difficilement négociable, les ressources étant par nature soit partageables soit non partageables.

4.1.2 Prendre et attendre

Pour s'assurer qu'une condition qui conduise à prendre une ressource puis attendre pour pouvoir poursuivre ne peut jamais apparaître dans le système, il faut garantir que, lorsqu'une tâche requiert une ressource, elle ne doit pas avoir pris aucune autre ressource. Un protocole simple à utiliser consiste à ce que chaque tâche se voit allouer toutes les ressources nécessaires avant de débuter son exécution ; ceci peut se faire en s'assurant que toutes les requêtes d'allocation de ressources soient satisfaites avant tout autre appel système.

Une alternative consiste à permettre à une tâche d'effectuer une requête lorsqu'elle n'a encore émis aucune autre requête d'allocation.

Problème : faible utilisation des ressources requises (celles qui sont peu utilisées) et possibilité de famines car les ressources nécessaires sont allouées et bloquées par des tâches qui ont préemptées ces ressources antérieurement.

4.1.3 Pas de préemption

La troisième condition implique qu'il n'y ait pas de préemption d'une ressource allouée. Le protocole suivant peut être défini : Si une tâche possède des ressources et fait une nouvelle requête qui ne peut être immédiatement satisfaite (impliquant une attente de cette tâche), alors toutes les ressources actuellement utilisées sont préemptées. En conséquence, ces ressources sont libérées ; les ressources préemptées sont rajoutées à la liste des ressources pour lesquelles la tâche est en attente. La tâche ne pourra être reprise que lorsque toutes les ressources initiales seront disponibles à nouveau en plus de celle qui a provoqué l'attente.

De façon alternative, si une tâche requiert des ressources, on teste d'abord si elles sont disponibles : si oui, elles sont alors allouées ; si non, on regarde si elles sont alloués à une autre tâche qui serait en attente d'autres ressources : dans ce cas, ces ressources sont récupérées de la tâche en attente et allouées à la nouvelle ; dans le cas contraire, la tâche qui demande ces ressources doit être suspendue ; en conséquence, les ressources ainsi bloquées peuvent être allouées à d'autres tâches qui en feraient ultérieurement la demande. Une tâche ne peut être re activée que si les nouvelles ressources requises sont allouées et doit être capable de retrouver toutes les ressources dont elle disposait avant sa suspension. Ce protocole est souvent appliqué aux ressources dont l'état peut être sauvé et restauré ultérieurement tels les registres de l'UC et la mémoire.

4.1.4 Attente en présence de cycle

Une façon de s'assurer qu'une telle condition ne peut survenir consiste à définir une topologie sur les types de ressources qui permette de définir un ordre, puis de s'assurer que chaque tâche émette des requêtes sur des ressources dans un ordre croissant.
Soir R = {R1, R2, …, Rn) un ensemble de types de ressources ; à chaque type on affecte un entier permettant de définir un ordre ; il existe donc une fonction F : R -> N.

On définit alors le protocole suivant pour prévenir l'interblocage : chaque tâche peut disposer de ressources seulement dans l'ordre croissant de l'énumération. Une tâche disposant de ressources de type F(Ri) ne peut disposer que de ressources F(Rj) tel que j > i. Si plusieurs instances d'une même ressource sont demandées alors toutes les instances sont allouées dans la même requête. La démonstration par l'absurde de l'absence de cycles peut être faite simplement.

4.2 Evitement de l'interblocage

Les algorithmes de prévention de l'interblocage sont fondés sur des restrictions sur l'émission des requêtes de sollicitations de ressources. Ces restrictions conduisent à ce qu'au moins une des conditions pour que l'interblocage se produise ne puisse apparaître avec comme corollaire une faible utilisation des ressources et éventuellement des famines pour certaines tâches en attente de ressources bloquées.

Une alternative consiste à fournir des informations complémentaires sur l'utilisation des ressources pour effectuer des choix pertinents : le modèle le plus simple et le plus utile consiste à ce que chaque tâche déclare le maximum de ressources de chaque type nécessaire. A partir de ces informations il est possible de construire un algorithme qui garantit que le système n'entrera jamais en interblocage. Un tel algorithme définit l'approche par évitement de l'interblocage. Cet algorithme examine dynamiquement l'état de l'allocation des ressources pour assurer qu'un cycle conduisant à un blocage ne peut survenir ; cet état est définit par le nombre de ressources disponibles et allouées et les demandes maximum des tâches.

4.2.1 Etat sains

Un état est dit sain, si le système peut allouer les ressources pour chaque tâche dans un ordre quelconque tout en évitant un interblocage ; formellement, un système est dans un état sain, s'il existe une suite saine. Une suite de tâches <P1, P2, …,Pn> est une suite saine pour l'état courant d'allocation si, pour chaque Pi, les requêtes que Pi peut faire peuvent être satisfaites avec les ressources disponibles plus les ressources détenues par toutes les tâches Pj, avec j < i.

Dans cette situation, si les ressources nécessaires pour Pi ne sont pas immédiatement disponibles alors Pi doit attendre que tous les Pj aient terminé ; dès lors, Pi peut obtenir toutes ses ressources, elle achève son exécution,

libère les ressources allouées et termine. Quand Pi termine, Pi+1 peut obtenir les ressources nécessaires et ainsi de suite. Si une telle suite n'existe pas alors le système est dit non sain.

Un état sain ne peut être en interblocage ; par contre un état non sain peut conduire (mais non nécessairement) à un interblocage.

Exemple :
Considérons un système ayant 12 disques et 3 tâches (P0, P1, P2) ; P0 doit disposer de 10 disques, P1 de 4 et P2 peut en nécessiter jusqu'à 9.
Soit à t0, P0 possède 5 disques, P1 possède 2 disques ainsi que P2 ; 3 disques sont donc libres.

A l'instant t0, le système est dans un état sain. La suite <P1, P0, P2> satisfait la condition de système sain.
P1 peut obtenir tous ses disques nécessaires (2 parmi les 3 libres), puis terminer et relâcher ses 4 ressources (le système dispose alors de 5 ressources libres) ; puis P0 peut prendre ces 5 ressources et disposer des 10 ressources nécessaires puis terminer ; le système dispose alors de 10 ressources libres ; P2 peut alors disposer des 9 ressources nécessaires en utilisant 7 ressources complémentaires ; il reste alors 3 ressources libres ; lorsque P2 termine, les 12 disques deviennent disponibles.

Si maintenant on suppose qu'à l'instant t1, P2 demande et reçoit un disque de plus.
Donc, à t1, P0 possède 5 disques, P1 2 disques et P2 3 disques ; 2 disques sont libres.
La suite <P1, P0, P2> ne satisfait plus la condition de système sain.
En effet, P1 obtient les 2 disques libres, termine et libère ses 4 disques ; P0 ne peut obtenir les 10 disques nécessaires (puisqu'il en possède 5 et ne peut en obtenir que 4) et donc va devoir attendre. De même P2 doit disposer de 6 ressources complémentaires qu'il ne peut avoir et doit donc attendre engendrant un blocage. Cette situation a été provoquée par l'allocation d'un disque de plus à P2 restreignant à 2 le nombre de disques libres.

L'idée pour assurer qu'un système peut toujours rester dans un état sain est relativement simple : Initialement le système est dans un état sain ; lorsqu'une tâche demande une ressource disponible, le système doit s'assurer si la ressource peut être allouée ou si la tâche doit être mise en attente. La requête est satisfaite si et seulement si le système reste dans un état sain. En conséquence l'utilisation de ressources peut s'avérer plus faible qu'elle ne pourrait l'être sans prendre l'assurance d'éviter les interblocages.

4.2.2 L'algorithme du banquier

Cet algorithme de prévention dynamique d'interblocage a été conçu par Dijkstra et Habermann dans la fin des années 60 ; son nom provient de l'analogie avec le comportement d'un banquier qui doit garder suffisamment de ressources pour pouvoir satisfaire l'ensemble des requêtes de ses clients, quitte à refuser certaines demandes qui ne lui permettraient pas par la suite de satisfaire les autres.

Lorsqu'une tâche entre dans le système, elle déclare le nombre maximum d'instances qu'elle peut être amenée à demander. Ce nombre évidemment ne peut dépasser le nombre de ressources disponibles dans le système.
Lorsqu'un utilisateur demande un ensemble de ressources, le système doit déterminer si cette allocation laisse le système dans un état sain ; si oui, les ressources sont allouées, si non, la tâche est mise en attente tant que toutes les ressources nécessaires ne sont pas disponibles (libérées par les autres tâches).

Cet algorithme nécessite les structures de données suivantes :
Soit *n* le nombre de tâches dans le système et *m* le nombre de types de ressources.

Disponible[m] : un vecteur d'entiers de taille m dont chaque entrée *j* mémorise le nombre de ressources disponibles *Rj*. Disponible[j] = k signifie que k ressources de type *Rj* sont disponibles.

Max[n,m] : une matrice qui modélise le maximum de demandes pour chaque tâche à un instant donné.
Si Max[i,j] = k alors la tâche *Pi* peut demander jusqu'à *k* instances de ressources de type *Rj*

Allocation[n,m] : une matrice qui indique le nombre de ressources de chaque type déjà allouée à chaque tâche. Si Allocation[i,j] = k, alors la tâche *Pi* possède, à l'instant considéré, *k* instances de ressources *Rj*.

Besoin[n,m] : une matrice qui indique le besoin en ressources pour chaque tâche à l'instant considéré.

Si Besoin[i,j] = k, alors la tâche *Pi* peut avoir besoin de *k* ressources complémentaires de type *Rj*. Notons que Besoin[i,j] = Max[i,j] - Allocation[i,j].

Soit Z et Y deux vecteurs de taille n. On établit Z ≤ Y ssi Z[i] ≤ Y[i] ∀ι ε [1,n]. On note $Allocation_i$ le vecteur spécifiant les ressources actuellement utilisées par la tâche Pi.
$Besoin_i$ définit les ressources additionnelles que la tâche Pi peut être amenée à requérir.

L'algorithme suivant, dit algorithme du banquier permet de déterminer si le système est dans un état sain.

Soit Work et Finish deux vecteurs de taille m et n tels que Work[i] = $Disponible_i$ et Finish[i] = true dès que $Allocation_i$ est réalisé.

Algorithme

1. Work = Disponible et Finish[i] = false pour tout i
2. Rechercher un i tel que :
   a. (Finish[i] = = false) et
   b. $Besoin_i \leq Work$

   S'il n'existe pas de i alors goto 4
3. Work = Work + $Allocation_i$
   Finish[i] = true
   Goto 2
4. If (Finish[i] = = true) pour tout i alors le système est dans un état stable.

L'algorithme qui permet de déterminer si les requêtes peuvent être honorées en laissant le système dans un état stable est alors :

Soit $Request_i$ le vecteur de requêtes pour la tâche Pi. Si $Request_i$ [j] = = k alors la tâche Pi souhaite k instances de la ressource Rj. Lorsqu'une requête est émise par la tâche Pi, les actions suivantes sont prises :

1. Si $Request_i \leq Besoin_i$ goto 2. Sinon, erreur, dépassement de ressources disponibles
2. $Request_i \leq$ Disponible, goto 3. Sinon, Pi attend la disponibilité de ressources
3. L'état est alors modifié ainsi :
   i. Disponible = Disponible - $Request_i$
   ii. $Allocation_i = Allocation_i + Request_i$
   iii. $Besoin_i = Besoin_i - Request_i$

Si l'état est sain, la transaction est effectuée et la tâche Pi reçoit ses ressources. Si l'état nouveau est non sain, alors la tâche Pi doit attendre que $Request_i$ soit possible, et l'état précédent est restauré.

4.3 Algorithmes de détection de l'interblocage

Dans le cas ou des solutions de prévention ou d'évitement de l'interblocage ne sont pas déployées il faut mettre en place des moyens pour détecter une telle situation et revenir à un point de reprise correct.

4.3.1 Une seule instance de chaque type de ressources

Le graphe d'allocation de ressources (Fig.1) est transformé en supprimant les sommets ressources et fusionnant les arcs tâches-ressources en un arc tâches-tâches. Ce nouveau graphe est appelé graphe d'attente.
Ainsi un arc de Pi vers Pj dans un tel graphe indique que Pi est en attente que Pj libère une ressource que Pi a demandée. Un arc de Pi vers Pj existe si et seulement si le graphe d'allocation des ressources contient deux arcs Pi vers Rq et Rq vers Pj pour la ressource Rq.

Un interblocage intervient ssi il existe un cycle dans un tel graphe. Pour détecter une telle situation, le système doit maintenir un graphe d'attente et tester régulièrement la présence de cycles dans ce graphe. Un tel algorithme est de complexité $O(n^2)$, n étant le nombre d'arcs du graphe.

4.3.2 Plusieurs instances de chaque type de ressources

L'algorithme précédent ne peut être utilisé ; cet algorithme utilise les mêmes structures que celles de l'algorithme du banquier.
Disponible[m] : un vecteur d'entiers de taille m dont chaque entrée *j* mémorise le nombre de ressources disponibles *Rj*. Disponible[j] = k signifie que k ressources de type *Rj* sont disponibles.

Allocation[n,m] : une matrice qui indique le nombre de ressources de chaque type déjà allouée à chaque tâche à un instant donné. Si Allocation[i,j] = k, alors la tâche *Pi* possède actuellement *k* instances de ressources *Rj*.

Request[n,m] : une matrice qui indique la requête en cours en nombre de ressources pour chaque tâche à un instant donné. Si Request[i,j] = k, alors la tâche *Pi* demande *k* ressources complémentaires de type *Rj*.

Algorithme

Soit Work et Finish deux vecteurs de taille m et n tels que Work[i] = $Disponible_i$ et Finish[i] = true dès que $Allocation_i$ est réalisé.

1. Work = Disponible ; si ($Disponible_i$!= 0) alors Finish[i] = false $\forall \iota \varepsilon [0,n-1]$ sinon Finish[i] = true.
2. Rechercher un i tel que :
   a. (Finish[i] = = false) et
   b. $Request_i \leq$ Work

   S'il n'existe pas de i alors goto 4
3. Work = Work + $Allocation_i$
   Finish[i] = true
   Goto 2
4. If (Finish[i] = = false) pour un i 0<= i <n alors le système est en interblocage. La tâche Pi est bloquée.

Exemple
Soit 5 tâches (P0, P1, P2, P3, P4) et trois types de ressources (A B C) composées respectivement de (7, 2, 6) individus.
A t0 le système à l'état suivant :

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

La suite <P0, P2, P3, P1, P4> conduit, en utilisant l'algorithme précédent aux valeurs de
Work :
[0 0 0] , [0 1 0] , [3 1 3] , [5 2 4] , [7 2 4] , [7 2 6] impliquant un système sans interblocage.

Supposons alors qu'à l'instant t1 P2 émette une requête de plus sur C.

	Allocation	Requête	Disponible
	A B C	A B C	A B C
P0	0 1 0	0 0 0	0 0 0
P1	2 0 0	2 0 2	
P2	3 0 3	**0 0 1**	
P3	2 1 1	1 0 0	
P4	0 0 2	0 0 2	

L'algorithme précédent donne :
Work :
[0 0 0] , [0 1 0], puis il n'existe pas de valeur de Requête pour satisfaire Requête <= Work donc P1, P2, P3, P4 sont en interblocage.