Système d'Exploitation 2

Chapitre 1 : Les interblocages (Deadlock)

1 ère Année Second Cycle

(partie 2)

Dr. M. Baba Ahmed

Plan (partie 2)

- Méthodes de traitement de l'interblocage
- Reprendre un interblocage que faire après détection de l'interblocage ? (guérison)
- L'évitement des interblocage

1. Cas 1 : ressource avec un seul exemplaire
2. Cas 2 : ressources avec plusieurs exemplaires

- Algorithme du banquier
- Prevention de l'interblocages

Algorithme de détection d'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.
- L'idée donc c'est de le lancer périodiquement,

Quand lancer l'algorithme de détection d'interblocage?

- Dépend de deux facteurs:
  - La fréquence d'apparition des interblocages;
  - Le nombre de processus et de ressources concernés par l'interblocage.

Principaux inconvénients de la détection

- Le coût de l'entretient des graphes
  - Les graphes doivent être mis à jour à chaque allocation ou libération de ressource.
- Le coût de la détection d'interblocage
  - Le nombre de nœuds dans les graphes peut être important.

Reprendre un interblocage
Que faire après détection d'interblocage ?

- Réquisition d'une ressource (Préemption)
- Restaurer un état antérieur
- Suppression ou destruction d'un ou plusieurs processus

Préemption de ressource

- Retirer temporairement une ressource à un processus pour l'attribuer à un autre.
  - Ce type de solution n'est pas souvent possible
  - Dépend du type de ressource (les ressources n'étant pas toutes réquisitionnables)

1. Ressources préemptibles
2. Ressources non préemptibles

Restaurer avec Rollback (retour en arrière)

- Il est possible de placer **des points de reprise** sur les processus:
  - En sauvegardant l'état du processus dans un fichier.
  - Le point de reprise contient l'image mémoire du processus, ainsi que l'état des ressources actuellement attribuées
  - En cas d'interblocage avant de retirer une ressource a un processus le restituer au rollback avant l'acquisition de cette ressource.
- Ce type de reprise est extrêmement coûteux et lourd à mettre en œuvre.

Destruction de processus

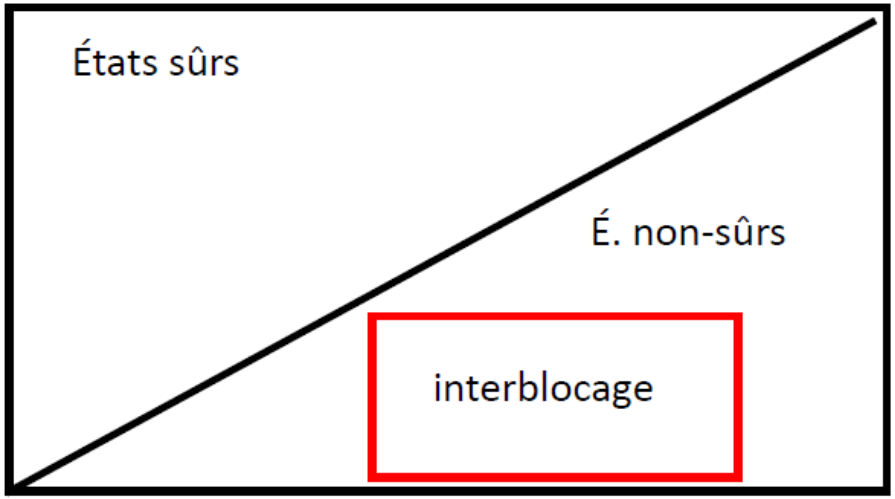
- Tuer tous les processus bloqués
  - Solution radicale
- Si nous tuons un des processus, il libèrera peut-être des ressources nécessaires
  - Problème du choix du processus à tuer.
- Il est préférable de supprimer un processus qui peut être redémarrer depuis le début sans conséquences (rollback)

Eviter les interblocages (Deadlock Avoidance)

- A chaque demande d'allocation de ressources, les système vérifies si cette allocation peut mener à un état non-sûr.
- Si c'est le cas (état non sur) la demande d'allocation est refusée
- Le système examine les séquences d'exécution possibles pour voir si un interblocage est possible

État sur (prudent, sain, certain)

- Un état est sûr si tous les processus peuvent terminer leur exécution (il existe une séquence d'allocations de ressources qui permet à tous les processus de se terminer)
- Ne pas allouer une ressource à un processus si l'état qui en résulte n'est pas sûr



Séquence sûre

- Une séquence de processus **<P_1, P_2, …, P_n>** est sûre si pour chaque Pi, les ressources que Pi peut encore demander peuvent être satisfaites par les ressources couramment disponibles + ressources utilisées par les Pj qui les précèdent.
  - Quand Pi aboutit, Pi+1 peut obtenir les ressources dont il a besoin, terminer, donc
- **<P_1, P_2, …, P_n>** est un ordre de terminaison de processus: tous peuvent se terminer dans cet ordre.

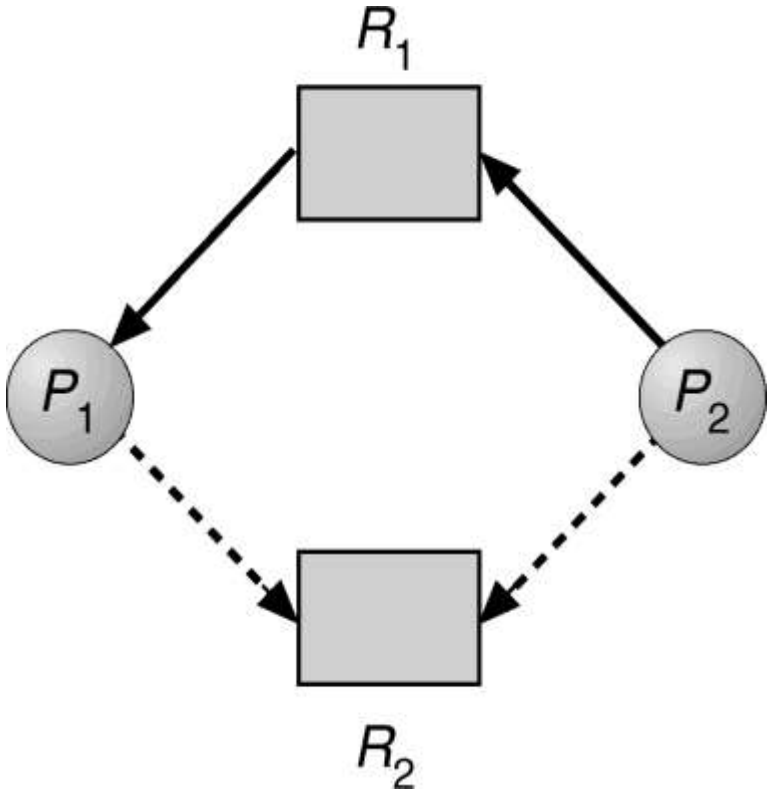
Cas d'un seul exemplaire par ressource: Algorithme d'allocation de ressources

- Il faut maintenant prendre en considération:
  - les requêtes possibles dans le futur (chaque processus doit déclarer ça)
- Arête demande P_i - > R_j indique que le processus P_i peut demander la ressource R_j (ligne à tirets)

Graphe d'allocation de ressources

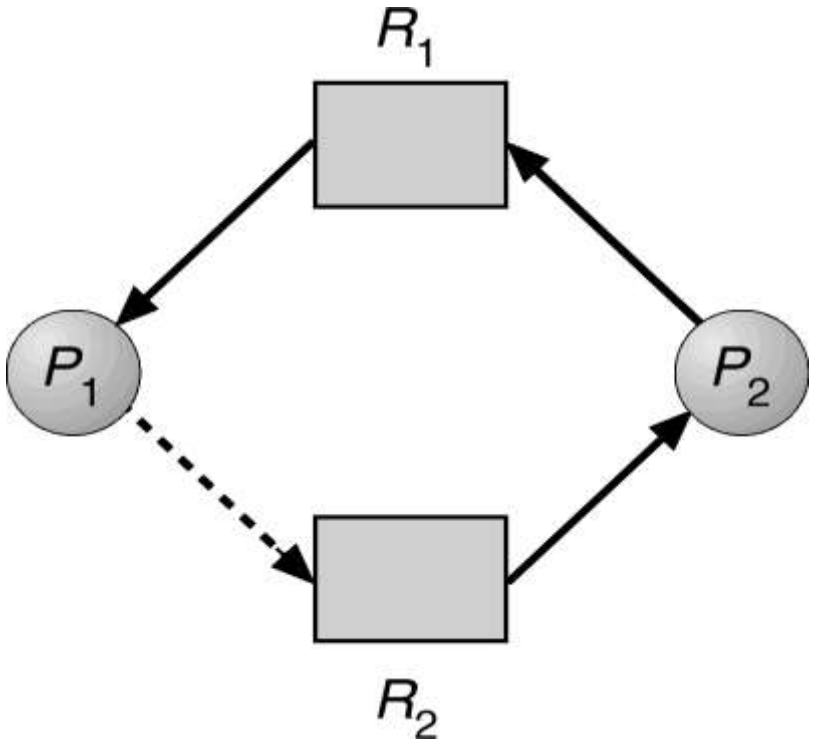
Ligne continue : requête courante

Tirets : requête possible dans le futur



Exemple (un état non-sûr)

Si P1 demande R2, ce dernier ne peut pas lui être donné, car ceci peut causer un cycle dans le graphe: P1 req R2



Cas de plusieurs exemplaires :structures de données

- Available : Vecteur de longueur m indiquant le nombre de ressources disponibles de chaque type.
  - **Available[j]=k**: veut dire que le type de ressources **Rj** possède k instances disponibles.
- Max : Matrice n x m définissant la demande maximale de chaque processus.
  - Si **Max[i, j]=k**: cela veut dire que le processus **Pi** peut demander au plus k instances du type de ressources **Rj**.

Cas de plusieurs exemplaires : structures de données

- Allocation : Matrice n x m définissant le nombre de ressources de chaque type de ressources actuellement alloué à chaque processus.
  - Si **Allocation[i, j]=k**: cela veut dire que l'on a alloué au processus **Pi** k instances du type de ressources **Rj.**
- Need : Matrice n x m indiquant les ressources restant à satisfaire à chaque processus.
  - 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.

Cas de plusieurs exemplaires : structures de données

- Request : Matrice n x m indiquant les ressources supplémentaires que les processus viennent de demander.
  - Si **Request[i, j]=k**: cela veut dire que le processus **Pi** vient de demander k instances supplémentaires du type de ressources **Rj**.

Algorithme du Banquier

- Les processus sont comme des clients qui désirent emprunter de l'argent (ressources) à la banque…
- Un banquier ne devrait pas prêter de l'argent s'il ne peut pas satisfaire les besoins de tous ses clients

Algorithme du Banquier

- Pour décider si la requête doit être accordée, l'algorithme du banquier teste si cette allocation conduira le système dans un état sûr:
  - accorder la requête si l'état est sûr
  - sinon refuser la requête
- Un état est sûr si et seulement s'il existe une séquence {P1..Pn} où chaque Pi est alloué toutes les ressources dont il a besoin pour terminer
  - ie: nous pouvons toujours exécuter tous les processus jusqu'à terminaison à partir d'un état sûr

Algorithme du Banquier (Demande de ressource)

Début

Etape 1 : Si $Request_i <= Need_i$ Alors Aller à l'étape 2

Sinon erreur : le processus a excédé ses besoins maximaux

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

Etape 3 : Sauvegarder l'état du système (les matrices Available, Allocation et Need).

Allouer les ressources demandées par le processus P_i 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

Fin.

Cet algorithme est appelé a chaque fois qu'un processus fait une demande de ressource

Algorithme du Banquier (Vérification état sain)

Début
Available : Tableau[m] de Entier
Finish : Tableau[n] de Logique (booleen)
Etape 1 : **Initialisation**
Finish :=Faux ;
Etape 2 : Trouver i tel que : Finish[i]=faux et $Need_i$<=Available
Si un tel i n'existe pas aller à l'étape 4.
Etape 3 : Available :=Available + $Allocation_i$
Finish[i] :=Vrai // processus marqué
Aller à l'étape 2
Etape 4 : Si Finish=Vrai (pour tout i) Alors
Verification_Etat_Sain :=Vrai
Sinon si Finish = faux alors
Verification_Etat_Sain :=Faux
Finsi
Fin.

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

Exemple (Vérification état sain)

Allocation

	R1	R2	R3
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

Max

	R1	R2	R3
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

Need

	R1	R2	R3
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

Need = Max-Allocation

Available

R1	R2	R3
3	3	2

Exemple (Vérification état sain)

Allocation

	R1	R2	R3
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

Max

	R1	R2	R3
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

Need

	R1	R2	R3
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

(Need Pi<=Available)
(Available ←Available+Alloc Pi)
Système est dans un état sain
Il existe une séquence d'éxècution qui est saine
{P1>P3>P4>P2>P0}
On peut trouvé plusieurs séquences saines

Available

R1	R2	R3
3	3	2

Existing

R1	R2	R3
10	5	7

	3	3	2
P1	5	3	2
P3	7	4	3
P4	7	4	5
P2	10	4	7
P0	10	5	7

Exemple (Demande de ressource)

Allocation

	R1	R2	R3
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

Max

	R1	R2	R3
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

Need

	R1	R2	R3
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

Need = Max-Allocation

Les requêtes des processus (Request) :

T1 : P1 demande (1,0,2)
T2: P4 demande (3,3,0)
T3 : P0 demande (0,2,0)

Available

R1	R2	R3
3	3	2

Exemple (Demande de ressource)

Allocation

	R1	R2	R3
P0	0	1	0
P1	3	0	2
P2	3	0	2
P3	2	1	1
P4	0	0	2

Max

	R1	R2	R3
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

Need

	R1	R2	R3
P0	7	4	3
P1	0	2	0
P2	6	0	0
P3	0	1	1
P4	4	3	1

Need = Max-Allocation

Les requêtes des processus (Request) :

T1 : P1 demande (1,0,2)
T2: P4 demande (3,3,0)
T3 : P0 demande (0,2,0)

Available

R1	R2	R3
2	3	0

Available = Available – requestPi
Allocation = AllocationPi + request Pi
Need = NeedPi –requestPi

Exemple (Demande de ressource)

Allocation

	R1	R2	R3
P0	0	1	0
P1	3	0	2
P2	3	0	2
P3	2	1	1
P4	0	0	2

Max

	R1	R2	R3
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

Need

	R1	R2	R3
P0	7	4	3
P1	0	2	0
P2	6	0	0
P3	0	1	1
P4	4	3	1

Need = Max-Allocation

Les requêtes des processus (Request) :

T1 : P1 demande (1,0,2)
T2: P4 demande (3,3,0)
T3 : P0 demande (0,2,0)

Available

R1	R2	R3
2	3	0

Réappliquer l'algorithme (Vérification état sain) avec les nouvelles modifications

Exemple (Demande de ressource)

Allocation

	R1	R2	R3
P0	0	1	0
P1	3	0	2
P2	3	0	2
P3	2	1	1
P4	0	0	2

Max

	R1	R2	R3
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

Need

	R1	R2	R3
P0	7	4	3
P1	0	2	0
P2	6	0	0
P3	0	1	1
P4	4	3	1

Need = Max-Allocation

Les requêtes des processus (Request) :

T1 : P1 demande (1,0,2)
T2: P4 demande (3,3,0)
T3 : P0 demande (0,2,0)

Available

R1	R2	R3
2	3	0

La séquence {P1>P3>P4>P0>P2} est saine, on peut donc accorder la demande de ressource faite par P1

Exemple (Demande de ressource)

Allocation

	R1	R2	R3
P0	0	1	0
P1	3	0	2
P2	3	0	2
P3	2	1	1
P4	0	0	2

Max

	R1	R2	R3
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

Need

	R1	R2	R3
P0	7	4	3
P1	0	2	0
P2	6	0	0
P3	0	1	1
P4	4	3	1

Need = Max-Allocation

Les requêtes des processus (Request) :

T1 : P1 demande (1,0,2)
T2: P4 demande (3,3,0)
T3 : P0 demande (0,2,0)

Available

R1	R2	R3
2	3	0

La requête de P4 est rejeté (ou mise en attente), les ressources ne sont pas disponible (Request(P4)>Available)

Exemple (Demande de ressource)

Allocation

	R1	R2	R3
P0	0	1	0
P1	3	0	2
P2	3	0	2
P3	2	1	1
P4	0	0	2

Max

	R1	R2	R3
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

Need

	R1	R2	R3
P0	7	4	3
P1	0	2	0
P2	6	0	0
P3	0	1	1
P4	4	3	1

Need = Max-Allocation

Les requêtes des processus (Request) :

T1 : P1 demande (1,0,2)
T2: P4 demande (3,3,0)
T3 : P0 demande (0,2,0)

Available

R1	R2	R3
2	3	0

La requête de P0 n'est pas accordé car l'état obtenue est malsain

Critiques de l'Algorithme du Banquier

- Coûteux : L'algorithme est très coûteux en temps d'exécution et en mémoire
  - Il faut maintenir plusieurs matrices, et déclencher à chaque demande de ressource, l'algorithme de vérification de l'état sain qui demande beaucoup d'operations
- 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).

Prévention de l'interblocage

- Pour qu'un interblocage puisse se produire, il faudrait que les 4 conditions d'interblocage soient vérifiées:
  - **Exclusion Mutuelle**
  - **Possession et attente**
  - **Pas de réquisition**
  - **Attente circulaire**
- Pour prévenir l'interblocage il suffit d'éliminer une condition d'entre elles.

Éliminer « l'exclusion mutuelle »

- Afin d'éviter l'exclusion mutuelle, il est parfois possible de sérialiser les requêtes portant sur une ressource

Exemple : deux processus souhaitent imprimer le contenu du même fichier (avec interblocage)

Solution : les deux processus « spoolent » leurs travaux dans un répertoire spécialisé qui vont être traité en série l'un après l'autre

Spooling : consiste a mettre des informations dans une file d'attente avant de les envoyer a un périphérique

- Rarement faisable dans le cas général (ressources critiques)

Éliminer « possession et attente »

- Pour éliminer cette condition, on doit s'assurer que le processus ayant fait la requête ne dispose pas d'autres ressources.
- Deux approches possibles:
  1. Allouer toutes les ressources demandées d'un seul coup (ou rien)
  2. Imposer la libération des ressources.

1. L'allocation globale

- Demander toutes les ressources dont on a besoin à accès exclusif d'un seul coup
- Principe: rendre atomique (indivisible) la séquence d'acquisition des ressources
  - On obtient toutes les ressources ou aucune ("tout ou rien"),
  - La phase d'acquisition étant elle-même une section critique

Inconvénients

- Il est généralement impossible de prédire les ressources qui peuvent être demandées
- Un processus risque d'attendre longtemps avant d'obtenir ses ressources.
- Il peut y avoir un gaspillage éventuel de ressources.
  - On oblige le processus à demander des ressources à un moment où il n'en a pas besoin.

Éliminer « pas de réquisition »

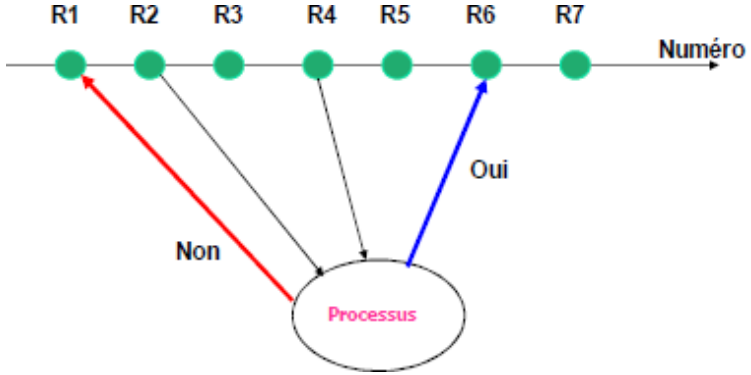
- On suppose de récupérer de force certaines ressources utilisées par un processus
  - Soit en terminant le processus
  - Soit en étant capable de le faire revenir en arrière
  - Nécessite de prévoir des « points de reprise » (rollback)
- Autoriser la réquisition n'est pas toujours souhaitable ou bien est délicate …

Exemples de problèmes de réquisition

- Difficile d'interrompre un processus qui en train d'utiliser une imprimante !
  - Car annuler les effets de l'utilisation de la ressource supposerait d'effacer des lignes déjà imprimées !!!
- Difficile d'annuler une transaction sur une Base de données (avec verrouillage d'enregistrements)
  - Le « rollback » nécessite l'usage d'un journal pour enregistrer les modifications effectuées.

Éliminer « l'attente circulaire »

- Solution possible : numérotation des ressources
  - Les processus peuvent demander toutes les ressources dont ils ont besoin, à condition de respecter l'ordre croissant de la numérotation des ressources.
  - Permet de briser la condition d'attente circulaire.
  - Si un processus détient des ressources d'un type donné, il ne peut demander que des ressources dont les numéros sont plus élevés.



Numérotation des ressources

- Avantage: évite les inconvénients de la méthode d'allocation globale et améliore, donc, la gestion des ressources.
- Inconvénient: Priorité fixée de manière statique entre les classes de ressources

Ce qui peut être difficile a mettre en œuvre