

INF2610

Noyau d'un système d'exploitation



Chapitre 6 - Interblocage

Sommaire

- Qu'est qu'un interblocage ?
- Solutions au problème d'interblocage
 - Détection des interblocages et reprise
 - Évitement des interblocages
 - Prévention des interblocages
- Exercice

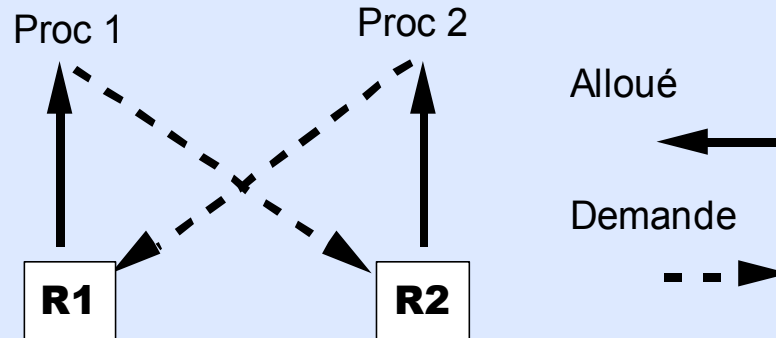
Qu'est ce qu'un interblocage ?

- Un ensemble de processus est en interblocage si chacun des processus de cet ensemble est en attente d'un événement qui ne peut être causé que par un autre processus du même ensemble.
- Comme tous les processus de cet ensemble sont en attente, aucun ne pourra s'exécuter et générer les événements nécessaires à leurs activations. Ils attendront tous indéfiniment.
- Si les attentes d'événements sont passives alors il s'agit d'un interblocage passif (deadlock). Sinon l'interblocage est qualifié d'actif (livelock).
- Dans le contexte de gestion des ressources,
 - l'événement attendu par chaque processus est la libération suivie de l'allocation d'une ressource au processus,
 - les ressources sont partagées en exclusion mutuelle, et
 - lorsqu'une ressource est allouée à un processus, il disposera de cette ressource jusqu'à ce qu'il la libère.

Qu'est ce qu'un interblocage ?

Exemple 1 :

- Un processus Proc1 détient une ressource R1 et attend une autre ressource R2 qui est allouée à un autre processus Proc2 ;
- Le processus Proc2 détient la ressource R2 et attend la ressource R1.



- On a une situation d'**interblocage** (Proc1 attend Proc2 et Proc2 attend Proc1). Les deux processus vont attendre indéfiniment.

Qu'est ce qu'un interblocage ?

- Quatre conditions sont nécessaires à l'interblocage (conditions de Coffman) :
 1. **Exclusion mutuelle** : Une ressource est soit allouée à **un seul processus**, soit disponible.
 2. **Détention et attente** : Les processus qui détiennent des ressources **peuvent en demander d'autres**.
 3. **Pas de réquisition** : Une ressource allouée est uniquement libérée par le processus qui la détient (ressources non-préemptives).
 4. **Attente circulaire** : Chaque processus d'un ensemble est en attente d'une ressource détenue par un autre processus de ce même ensemble.
- ➔ Si l'une de ces conditions n'est pas satisfaite alors il n'y a pas d'interblocage.
- ➔ Si ces conditions sont toutes satisfaites, à un instant donné, alors il y a un interblocage.

Solutions au problème d'interblocage

1. Détection des interblocages et reprise ;
 - À l'aide d'un graphe d'allocation des ressources.
2. Évitement des interblocages
 - Algorithme du banquier
3. Prévention des interblocages
 - En s'assurant que l'une des quatre conditions nécessaires ne soit **jamais satisfaite**.

Détection des interblocages et reprise



- Cette solution vise à détecter les interblocages et à les supprimer. Les ressources sont allouées si elles sont disponibles.
- Elle se base sur la construction dynamique du graphe d'allocations des ressources.
- Ce graphe indique pour chaque processus les ressources qu'il détient et celles qu'il attend.
- La vérification de la présence d'interblocages est réalisée :
 - à chaque modification du graphe suite à la demande d'une ressource ou
 - périodiquement lorsque l'utilisation du processeur est inférieure à un certain seuil.
- La détection est réalisée en réduisant le graphe.

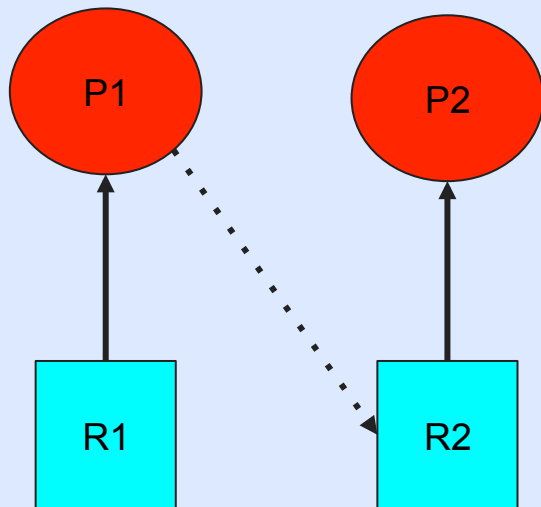
Détection des interblocages et reprise

Graphe d'allocation des ressources

- Le graphe d'allocation des ressources est un graphe biparti composé de deux types de nœuds :
 - ceux qui représentent, sous forme de cercles, les processus et
 - ceux qui représentent, sous forme de carrés, les ressources.
- Un arc connectant une ressource à un processus signifie que la ressource est allouée au processus.
- Un arc connectant un processus à une ressource signifie que le processus est en attente de la ressource.

Détection des interblocages et reprise

Graphe d'allocation des ressources - Exemple



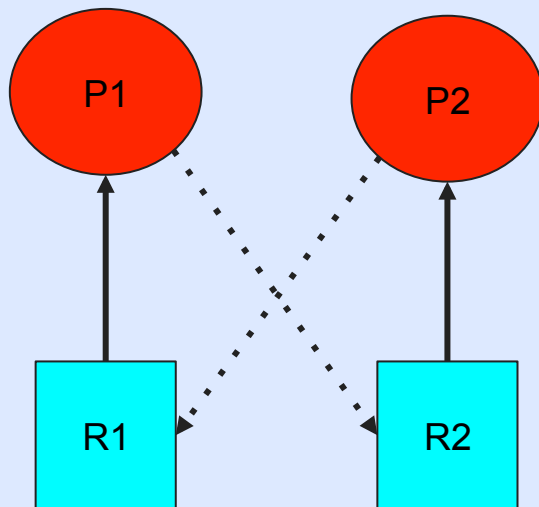
Selon ce graphe,

- la ressource R1 est allouée au processus P1,
- la ressource R2 est allouée au processus P2, et
- le processus P1 demande la ressource R2.

Détection des interblocages et reprise

Graphe d'allocation des ressources - Situation d'interblocage

Selon ce graphe,



- la ressource R1 est allouée au processus P1,
- la ressource R2 est allouée au processus P2,
- le processus P1 demande la ressource R2, et
- le processus P2 demande la ressource R1.

➔ Interblocage.

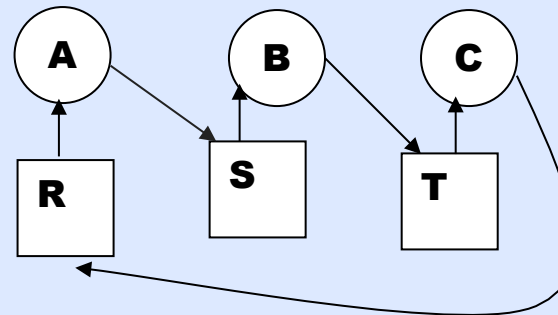
➔ Si chaque ressource existe en un seul exemplaire, il y a un interblocage dans l'état courant ssi il y a un cycle dans le graphe.

Détection des interblocages et reprise

Exemple 2 : 3 processus A, B et C utilisent, en exclusion mutuelle, 3 ressources R, S et T.

A		B		C	
(1)	Demande R	(2)	Demande S	(3)	Demande T
(4)	Demande S	(5)	Demande T	(6)	Demande R
	/*traitement*/		/*traitement*/		/*traitement*/
	Libère R		Libère S		Libère T
	Libère S		Libère T		Libère R

- Si les processus sont exécutés séquentiellement, il n'y aurait pas d'interblocage.
- Si les ressources sont demandées dans l'ordre indiqué en rouge, on aurait une situation d'interblocage :



Détection des interblocages et reprise

- Dans le cas général, le système vérifie s'il y a un interblocage en réduisant le graphe d'allocation des ressources.
 - La réduction du graphe consiste à vérifier s'il existe un ordre d'allocation des ressources à partir de l'état courant (du graphe) qui permet à tous les processus de se terminer.
 - Lorsque le système détecte un interblocage, il doit sortir de cette situation en :
 - retirant temporairement une ressource à un processus,
 - restaurant un état antérieur tout en évitant de retomber dans la même situation, ou
 - supprimer un ou plusieurs processus.
- ➔ La reprise n'est pas évidente (coûteuse ou encore non acceptable dans certains contextes).

Détection des interblocages et reprise – Exercice 1

- Construisez le graphe d'allocation des ressources correspondant à l'état suivant :
 - P1 détient R et demande S
 - P2 demande T
 - P3 demande S
 - P4 détient U et demande S et T
 - P5 détient T et demande V
 - P6 détient W et demande S
 - P7 détient V et demande U
- Y a-t-il un interblocage ? Si oui, quels sont les processus concernés ?

Détection des interblocages et reprise – Exercice 2

- Un système dispose de 3 R1, 2 R2 et 2 R3. Supposez que 4 processus P1, P2, P3 et P4 partagent ces ressources et que l'état actuel d'attribution des ressources est :
 - P1 détient une ressource de type R1 et demande une ressource de type R2.
 - P2 détient 2 ressources de type R2 et demande une ressource de type R1 et une ressource de type R3.
 - P3 détient 1 ressource de type R1 et demande une ressource de type R2.
 - P4 détient 2 ressources de type R3 et demande une ressource de type R1.
- Construisez le graphe d'allocation des ressources. Y a-t-il un interblocage ? Si oui, quels sont les processus concernés ?

Détection des interblocages et reprise



Outils google :

ThreadSanitizerDeadlockDetector (<https://github.com/google/sanitizers/wiki/ThreadSanitizerDeadlockDetector>) :

« The deadlock detector maintains a directed graph of lock acquisitions. If a lock B is acquired while a lock A is being held by the same thread, a directed edge $A \Rightarrow B$ is added to the lock acquisition graph. A potential deadlock is reported when there is a cycle in the graph. »

ThreadSanitizer (<https://clang.llvm.org/docs/ThreadSanitizer.html>) :

« ThreadSanitizer (aka TSan) is a data race detector for C/C++. Data races are one of the most common and hardest to debug types of bugs in concurrent systems. A data race occurs when two threads access the same variable concurrently and at least one of the accesses is write. »

Évitement des interblocages

- Cette solution vise à éviter les interblocages en analysant toute demande d'allocation de ressources et en refusant toutes celles qui risquent de mener vers un interblocage.
 - Elle se base sur la notion d'état sûr (ou sain).
 - Un état est dit **sûr** si tous les processus peuvent terminer leurs exécutions dans le pire cas (c-à-d tous les processus demandent en même temps toutes les ressources manquantes à leurs exécutions complètes).
- ➔ Il faut connaître à l'avance les besoins en ressources de chaque processus, ce qui est généralement impossible.

Évitement des interblocages

- Lorsqu'un processus demande une ressource, le système doit déterminer si l'attribution de la ressource mène vers un état sûr.
 - Si c'est le cas, il lui attribue la ressource. Sinon, la ressource n'est pas accordée.
 - Un état est défini par 4 tableaux :
 - E: tableau des ressources du système (disponibles + louées),
 - A: tableau des ressources disponibles,
 - Alloc: tableau des ressources allouées à chaque processus, et
 - Req: tableau des ressources potentiellement nécessaires à chaque processus mais non encore détenues.
- $A = E - (\text{Alloc}(P1) + \dots + \text{Alloc}(Pn))$.

Comment vérifier si un état est sûr ? → Algorithme du banquier

Évitement des interblocages

Algorithme du banquier :

1. Rechercher un processus P_i non marqué dont la rangée P_i de Req est inférieure ou égale à A ;
2. Si un tel processus n'existe pas alors l'état est non sûr. L'algorithme se termine.
3. Sinon, ajouter la rangée P_i de Alloc à A et marquer le processus P_i ;
4. Si tous les processus sont marqués alors l'état est sûr et l'algorithme se termine, sinon aller à l'étape 1.

État : S

		R1	R2	R3	R4
E =		(4	2	3	1)
A =		(2	1	0	0)

		R1	R2	R3	R4
	P1	0	0	1	0
Alloc =	P2	2	0	0	1
	P3	0	1	2	0

		R1	R2	R3	R4
	P1	2	0	0	1
Req =	P2	1	0	1	0
	P3	2	1	0	0

S est-il sûr ?

Évitement des interblocages

État S :

E	R1	R2	R3	R4
	4	2	3	1

Alloc		R1	R2	R3	R4
	P1	0	0	1	0
	P2	2	0	0	1
	P3	0	1	2	0

A	R1	R2	R3	R4
	2	1	0	0

Req		R1	R2	R3	R4
	P1	2	0	0	1
	P2	1	0	1	0
	P3	2	1	0	0

S est-il sûr ?



Processus marqué

Évitement des interblocages

S est-il sûr ?

A

R1	R2	R3	R4
2	1	0	0

Req

	R1	R2	R3	R4
P1	2	0	0	1
P2	1	0	1	0
P3	2	1	0	0

Alloc

	R1	R2	R3	R4
P1	0	0	1	0
P2	2	0	0	1
P3	0	1	2	0

- $\text{Req}(P1) \leq A$? Non
- $\text{Req}(P2) \leq A$? Non
- $\text{Req}(P3) \leq A$? Oui
- Ajouter Alloc(P3) à A :

A

R1	R2	R3	R4
2	2	2	0

- Marquer P3 et aller à l'étape 1.

Évitement des interblocages

S est-il sûr ?

A	R1	R2	R3	R4
	2	2	2	0

Req		R1	R2	R3	R4
	P1	2	0	0	1
	P2	1	0	1	0
	P3	2	1	0	0

Processus marqué

Alloc

	R1	R2	R3	R4
P1	0	0	1	0
P2	2	0	0	1
P3	0	1	2	0

- $\text{Req}(P1) \leq A$? Non
- $\text{Req}(P2) \leq A$? Oui
- Ajouter $\text{Alloc}(P2)$ à A :

A	R1	R2	R3	R4
	4	2	2	1


- Marquer le processus P2 puis aller à l'étape 1.

Évitement des interblocages

S est-il sûr ?

A	R1	R2	R3	R4
	4	2	2	1

Req		R1	R2	R3	R4
	P1	2	0	0	1
	P2	1	0	1	0
	P3	2	1	0	0

 Processus marqué

Alloc

	R1	R2	R3	R4
P1	0	0	1	0
P2	2	0	0	1
P3	0	1	2	0

- $\text{Req}(P1) \leq A$? Oui
- Ajouter $\text{Alloc}(P1)$ à A

A	R1	R2	R3	R4
	4	2	3	1

- Marquer le processus P1.



Processus marqué

Évitement des interblocages

S est-il sûr ?

A	R1	R2	R3	R4
	4	2	2	1

Req		R1	R2	R3	R4
	P1	2	0	0	1
	P2	1	0	1	0
	P3	2	1	0	0

- Tous les processus sont marqués.
L'état est sûr.



Processus marqué

Évitement des interblocages - Exemple

- À l'état S, le système reçoit une demande de ressource R1 du processus P1.
- Va-t-il accorder la ressource R1 au processus P1 ?

E

R1	R2	R3	R4
4	2	3	1

Alloc

	R1	R2	R3	R4
P1	0	0	1	0
P2	2	0	0	1
P3	0	1	2	0


A

R1	R2	R3	R4
2	1	0	0

Req

	R1	R2	R3	R4
P1	2	0	0	1
P2	1	0	1	0
P3	2	1	0	0

- Il va accorder R1 à P1 uniquement si l'allocation de R1 à P1 mène vers un état sûr

 Processus marqué

Évitement des interblocages - Exemple

- L'état S' successeur de S par l'allocation de R1 à P1 :

E'

R1	R2	R3	R4
4	2	3	1

Alloc'

	R1	R2	R3	R4
P1	0 1	0	1	0
P2	2	0	0	1
P3	0	1	2	0

A'

R1	R2	R3	R4
2 1	1	0	0

Req'

	R1	R2	R3	R4
P1	2 1	0	0	1
P2	1	0	1	0
P3	2	1	0	0

- Req'(P1) <= A ? Non; Req'(P2) <= A ? Non; Req'(P3) <= A ? Non.
- S' est non sûr → la demande de R1 par P1 va être rejetée par le système.

Prévention des interblocages

- Cette solution vise à s'assurer qu'au moins, l'une des quatre conditions nécessaires à l'interblocage, n'est jamais satisfaite.

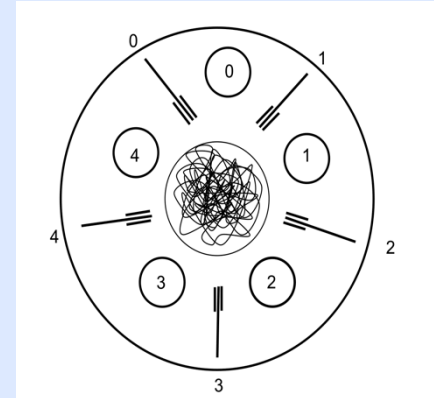
Condition	Comment l'empêcher ?
Exclusion mutuelle	Impossible de l'empêcher car les ressources sont en général à usage exclusif (imprimantes, mémoires, processeurs, etc.).
Détention et attente	Chaque processus demande à la fois toutes les ressources dont il a besoin. Cependant, <ul style="list-style-type: none">a. il est difficile de connaître à l'avance les besoins des processus etb. il y a un risque de famine.

Prévention des interblocages

Condition	Comment l'empêcher ?
Pas de réquisition	L'état de la ressource est sauvegardé avant de la préempter puis restaurer ultérieurement. Cette solution est envisageable pour certaines ressources (processeurs, mémoires, etc.) mais pas pour d'autres (imprimantes).
Attente circulaire	Une solution consiste à : <ul style="list-style-type: none">- ordonner les ressources, et- imposer, à chaque processus, la règle suivante: Un processus peut demander une ressource R_j seulement si toutes les ressources qu'il détient sont inférieures à R_j selon l'ordre établi.

Prévenir les interblocages : Problème des philosophes

./run.sh Synchronisation/
Philosophes



- Chaque philosophe demande à la fois la fourchette à gauche ainsi que celle à sa droite.

➔ Pas de détention et attente

Problème des philosophes

```
#define N 5
#define G (i+1)%N
#define D i
#define libre 1
#define occupe 0
int fourchettes[N] = { libre, libre, libre,
                      libre, libre };
Semaphore mutex;
```

```
Philosophe(int i) {
    int cp=0;
    while (1) {    penser(); // sleep(1);
        P(mutex);
        if (fourchettes[G] == libre && fourchettes[D] == libre) {
            fourchettes[G] = occupe; fourchettes[D] = occupe;
            V(mutex);
            printf("philosophe %d mange pour la %d ieme\n", i, cp++);
            manger(); // sleep(1);
            printf("philosophe %d a fini de manger\n", i);
            P(mutex);
            fourchettes[G] = libre; fourchettes[D] = libre;
            V(mutex);
        } else {    V(mutex); }
    }
}
```

Prévenir les interblocages : Problème des philosophes

- Les fourchettes sont ordonnées comme suit :
 $f_0 < f_1 < f_2 < f_3 < f_4$
 - Chaque philosophe demande les fourchettes à sa gauche et à sa droite en commençant par la plus petite selon l'ordre établi.
- Pas d'attente circulaire.

```
#define N 5      // nombre de philosophes
#define G (i+1)%N // fourchette gauche du philosophe pour i=0 à 4
#define D i      // fourchette droite du philosophe i
```

```
Semaphore f[N]={1,1,1,1,1};
```

```
Philosophe(int i) {
    int cp=0;
    while (1) {
        sleep(1);    // penser
        if(G<D) { P(f[G]); P(f[D]);}
        else {      P(f[D]); P(f[G]); }
        printf("philosophe %d mange pour la %d ieme\n", i, cp++);
        sleep(1);    // manger
        printf("philosophe %d a fini de manger\n", i);
        V(f[G]); V(f[D]);
    }
}
```

Exercice 3

- On dispose d'un mécanisme d'enregistrement à un ensemble de cours, tel que :
 - un étudiant ne peut être inscrit à plus de trois cours, et
 - chaque cours a un nombre limité de places.
- Un étudiant inscrit déjà à trois cours peut s'il le souhaite en abandonner un, pour en choisir un autre dans la limite des places disponibles.
- Si cet échange n'est pas possible, l'étudiant ne doit pas perdre les cours auxquels il est déjà inscrit.
- On vous propose la fonction EchangeCours suivante :

Exercice 3 (suite)

```
void EchangeCours (PUtilisateur utilisateur, PCours cours1, cours2) {  
    cours1->verrouille (); // verrouille l'accès à l'objet cours1  
    cours1->desinscrit (utilisateur);  
    if (cours2->estPlein() == false) {  
        cours2->verrouille (); // verrouille l'accès à l'objet cours2  
        cours2->inscrit (utilisateur);  
        cours2->deverrouille (); //déverrouille l'accès à l'objet cours2  
    }  
    cours1->deverrouille (); //déverrouille l'accès à l'objet cours1  
}
```

- **Vérifiez si l'implémentation est correcte** : Si elle est correcte, **expliquez pourquoi**, en montrant comment est géré le cas où deux étudiants (ou plus) veulent accéder en même temps au système. Si elle est incorrecte, **listez et expliquez les problèmes**, et **proposez** une solution qui fonctionne.

Exercice 3 (solution)

Cette implémentation n'est pas correcte car :

- Elle va désinscrire l'utilisateur de son premier cours, même si elle ne peut pas l'inscrire au deuxième cours ;
- Elle verrouille les cours dans un ordre (le premier cours puis le deuxième cours). Un interblocage est possible si un autre appel à la fonction verrouille les mêmes cours dans l'ordre inverse.
- Elle ne verrouille pas le deuxième cours avant le test «if (cours2->estPlein()==false) ».

Exercice 3 (1^{ère} solution)

Éviter « attente circulaire »

```
void EchangeCours (PUtilisateurs utilisateur, PCours cours1, cours2) {  
    if (cours1 < cours2 )  
    {        cours1->verrouille(); cours2->verrouille (); }  
    else if (cours1 > cours2 )  
        { cours2->verrouille(); cours1->verrouille (); }  
    else return;  
  
    if (cours2->estPlein() == false)  
    {    cours2->inscrit (utilisateur);  
        cours1->desinscrit (utilisateur);  
    }  
    cours1->deverrouille ();  
    cours2->deverrouille ();  
}
```

Exercice 3 (2^{ième} solution)

Éviter « détention et attente »

```
void EchangeCours (PUtilisateurs utilisateur, PCours cours1, cours2) {  
    cours2->verrouille ();  
    if (cours2->estPlein() == false) {  
        cours2->inscrit (utilisateur);  
        cours2->deverrouille ();  
        cours1->verrouille ();  
        cours1->desinscrit (utilisateur);  
        cours1->deverrouille ();  
    }  
    else  
        cours2->deverrouille ();  
}
```

Lecture suggérée

- **Chapitre 4.2 : Synchronisation des processus**

Introduction aux systèmes d'exploitation - Cours et exercices en GNU/Linux, Hanifa Boucheneb & Juan-Manuel Torres-Moreno, 216 pages, édition ellipses, 2019, ISBN : 9782340029651.

- **Capsules de Vittorio : Algorithme du banquier**

https://www.youtube.com/channel/UCffP7k2AXCttKadZcHsqYg?view_as=subscriber