Noyau d'un système d'exploitation INF2610

Chapitre 6: Interblocage

Département de génie informatique et génie logiciel

POLYTECHNIQUE MONTRÉAL
AFFILIÉE À L'UNIVERSITÉ DE MONTRÉAL

Hiver 2019

Interblocage

- **Introduction**
- **Qu'est ce qu'un interblocage ?**
- **Conditions nécessaires pour l'interblocage**
- **Solutions au problème d'interblocage**
  - **Détection et reprise**
  - **Évitement des interblocages**
  - **Prévention des interblocages**

Introduction

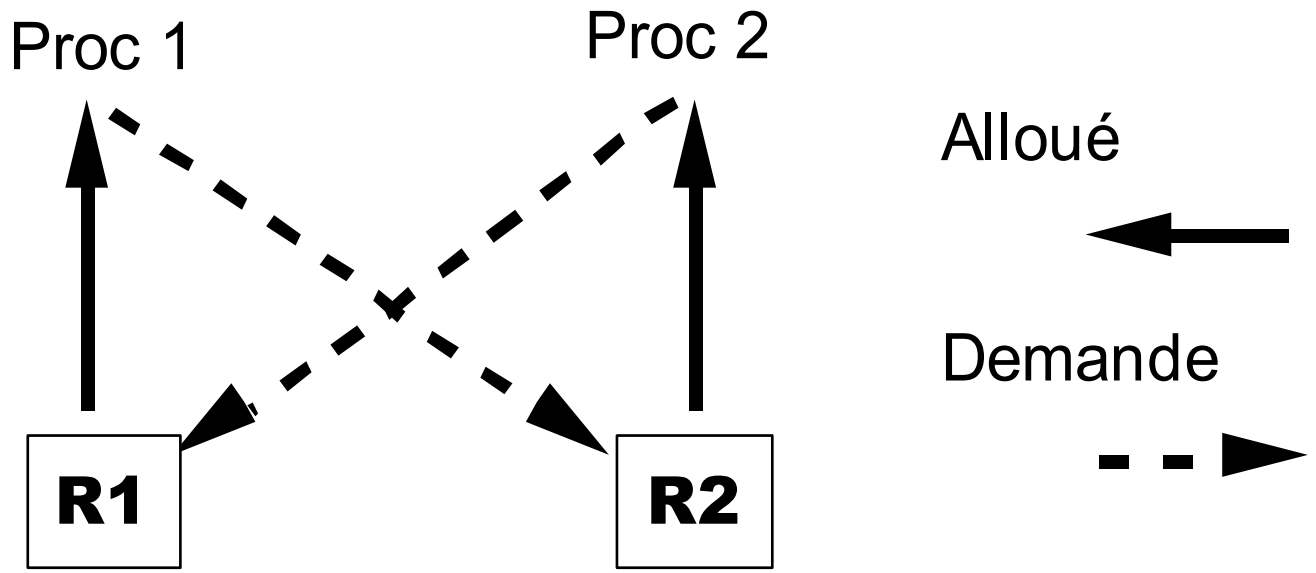
- L'exécution d'un processus nécessite un ensemble de ressources (espace mémoire centrale, espace disque, fichier, périphériques, …) qui lui sont attribuées par le système d'exploitation.

- Des problèmes peuvent survenir, si des processus détiennent des ressources et en demandent d'autres qui sont déjà allouées.

Exemple 1 :

- un processus Proc1 détient une ressource R1 et attend une autre ressource R2 qui est utilisée par 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 ?



- Un ensemble de processus est en interblocage si chaque processus attend la libération d'une ressource allouée à un autre appartenant à l'ensemble.
- Comme tous les processus sont en attente, aucun ne pourra s'exécuter et donc libérer les ressources demandées par les autres. Ils attendront tous indéfiniment.

Conditions nécessaires pour l'interblocage (Coffman, Elphick et Shoshani)

- Exclusion mutuelle : une ressource est soit allouée à un seul processus, soit disponible;
- Détention et attente : les processus qui détiennent des ressources peuvent en demander d'autres;
- Pas de réquisition : les ressources allouées à un processus sont libérées uniquement par le processus (ressources non préemptives);
- Attente circulaire: un ensemble de processus attendant chacun une ressource allouée à un autre.

Solutions au problème d'interblocage

- Les détecter et y remédier.

- Les éviter en allouant les 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.

- Les prévenir en empêchant l'apparition de l'une des quatre conditions nécessaires à leur existence.

Remarque :

- En général, ce problème est ignoré par les systèmes d'exploitation car le prix à payer pour les éviter ou les traiter est trop élevé pour des situations qui se produisent rarement.

Détection et reprise

- Dans ce cas, le système ne cherche pas à empêcher les interblocages. Il tente de les détecter et d'y remédier.

- Pour détecter les interblocages, il construit dynamiquement le graphe d'allocation des ressources du système qui indique les attributions et les demandes de ressources.

- Le système vérifie s'il y a des interblocages :
  - A chaque modification du graphe suite à une demande d'une ressource (coûteuse en termes de temps processeur).
  - Périodiquement ou lorsque l'utilisation du processeur est inférieure à un certain seuil (la détection peut être tardive).

Détection et reprise (2)
Graphe d'allocation des ressources

- Le graphe d'allocation des ressources est un graphe biparti (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 la ressource est allouée au processus.
  - Un arc orienté d'un processus vers une ressource signifie que le processus est bloqué en attente de la ressource.

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

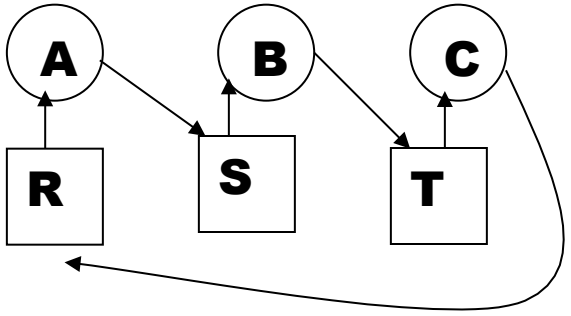
- La détection est réalisée en réduisant le graphe.

Détection et reprise (3)
Graphe d'allocation des ressources

Exemple 2 : Soient trois processus A, B et C qui utilisent trois ressources R, S et T comme suit :

A	B	C
Demande R	Demande S	Demande T
Demande S	Demande T	Demande R
Libère R	Libère S	Libère T
Libère S	Libère T	Libère R

- Si les processus sont exécutés séquentiellement A suivi de B suivi C, il n'y aurait pas d'interblocage.
- On atteindrait une situation d'interblocage, si les instructions sont exécutées dans cet ordre :
  - A demande R
  - B demande S
  - C demande T
  - A demande S
  - B demande T
  - C demande R



Détection et reprise (4)

- Lorsque le système détecte un interblocage, il doit le supprimer, ce qui se traduit généralement par la réalisation de l'une des opérations suivantes :
  - Retirer temporairement une ressource à un processus pour l'attribuer à un autre.
  - Restaurer un état antérieur (retour arrière) et éviter de retomber dans la même situation.
  - Supprimer un ou plusieurs processus.

- La reprise n'est pas évidente.

Exercice 1

Considérons l'attribution des ressources suivante :

A détient R et demande S ;

B demande T ;

C demande S ;

D détient U et demande S et T ;

E détient T et demande V ;

F détient W et demande S ;

G détient V et demande U.

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

Exercice 2

Considérons un système gérant quatre processus, P1 à P4, et trois types de ressources R1, R2 et R3 (3 R1, 2 R2 et 2 R3). L'attribution des ressources :

- 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.

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

Évitement des interblocages

- Dans ce cas, lorsqu'un processus demande une ressource, le système doit déterminer si l'attribution de la ressource est sûre (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 sûr si tous les processus peuvent terminer leurs exécutions (il existe un ordre d'allocation de ressources qui permet à tous les processus de se terminer).

- Il faut connaître à l'avance les besoins en ressources de chaque processus (ce qui est en général impossible).

Évitement des interblocages
Comment déterminer si un état est sûr ?

Algorithme du banquier : état est caractérisé par quatre tableaux.

$$E = \begin{pmatrix} R1 & R2 & R3 & R4 \\ 4 & 2 & 3 & 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix}$$

$$Alloc = \begin{matrix} P1 \\ P2 \\ P3 \end{matrix} \begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{pmatrix}$$

$$Req = \begin{matrix} P1 \\ P2 \\ P3 \end{matrix} \begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{pmatrix}$$

1. Rechercher un processus Pi non marqué dont la rangée Pi 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 Pi de Alloc à A, marquer le processus ;
4. Si tous les processus sont marqués alors l'état est sûr et l'algorithme se termine, sinon aller à l'étape 1.

A : ressources disponibles,

E : ressources du système,

Alloc : ressources attribuées,

Req : ressources nécessaires non encore obtenues.

Exercice 3

L'état courant est-il sûr ?

Peut-on accorder 2 ressources R1 à P3 ?

Peut-on accorder 1 ressource R1 à P2 ?

Prévention des interblocages

Pour prévenir les interblocages, on doit faire en sorte que l'une des quatre conditions nécessaires à leur existence ne soit jamais satisfaite.

- Pas d'exclusion mutuelle : impossible car certaines ressources sont à usage exclusif.
- Pas de « détention et attente » : Il faudrait que toutes les ressources nécessaires à un processus soient demandées et allouées à la fois. Le processus ne doit pas détenir des ressources et en demander d'autres.
  - Il est difficile de prévoir les besoins du processus.
  - Problème de famine.
- Préemption : n'est pas raisonnablement traitable pour la plupart des ressources sans dégrader profondément le fonctionnement du système. On peut cependant l'envisager pour certaines ressources dont le contexte peut être sauvegardé et restauré.

Prévention des interblocages (2)

- Pas d'attente circulaire, si on parvient à :
  - établir un ordre total entre les ressources et
  - imposer, à chaque processus, la règle de demande de ressources suivante : Un processus peut demander une ressource Rj seulement si toutes les ressources qu'il détient sont inférieures à Rj

Exemple 3 : problème des philosophes

- Les fourchettes sont ordonnées f0 < f1 < f2 < f3 < f4. Les philosophes doivent prendre les fourchettes en ordre croissant :
  - Le philosophe 0 doit demander la fourchette f0 puis f4
  - Le philosophe 1 doit demander la fourchette f0 puis f1
  - Le philosophe 2 doit demander la fourchette f1 puis f2
  - Le philosophe 3 doit demander la fourchette f2 puis f3
  - Le philosophe 4 doit demander la fourchette f3 puis f4.

Prévention des interblocages (3)

Exemple 3 : problème des philosophes (suite)

```
Semaphore Fourch[5] ={1,1,1,1,1}
```

```
 Philosophe0()
{   penser();
    P(Fourch[0]);
    P(Fourch[4]);
    manger();
    V(Fourch[0]);
    V(Fourch[4]);
}
```

```
Philosophe (num in [1,4])
{    penser();
     P(Fourch[num-1]);
     P(Fourch[num]);
     manger();
     V(Fourch[num-1]);
     V(Fourch[num]);
}
```

Problème de famine

- Le problème de famine est un autre problème inhérent à la gestion de ressources.

- L'allocation de ressources est indéfiniment retardée.

- Pour éviter ce problème :
  - Mémoriser les demandes dans une file pour les traiter selon la discipline FIFO.
  - Limiter le temps d'allocation de chaque ressource.
  - Augmenter progressivement la priorité d'un processus avec le temps d'attente (si la priorité est un critère d'ordonnancement).

Exercice 4

- On dispose d'un mécanisme d'enregistrement à un ensemble de cours, tel que :
  - tout 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.

- Le bureau des affaires académiques souhaite donc mettre en place un système de permutation de cours, permettant à un étudiant de changer de cours. Il vous sollicite pour vérifier si l'implémentation que vous avez proposée il y a un an (avant se suivre le cours INF2610) est correcte :

Exercice 4 (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 4 (suite)

Cette implémentation est incorrecte pour plusieurs raisons :

- On désinscrit toujours l'utilisateur de son premier cours, même s'il ne peut pas s'inscrire au deuxième cours.
- Du fait que l'on verrouille le deuxième cours alors qu'on a déjà un verrou sur le premier, on a un risque d'interblocage si deux étudiants lancent simultanément la fonction avec les deux mêmes valeurs de cours, mais dans l'ordre inverse : chacun verrouillera d'abord le cours que l'autre voudra verrouiller ensuite, et il ne sera donc pas possible de sortir de cet interblocage.
- On ne verrouille pas le deuxième cours avant de faire le test pour savoir s'il est plein.

Exercice 4 (suite)

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

Exercice 4 (suite)

```
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 ();
}
```

Exercice 5

- Deux processus P1 et P2 partagent un fichier fich. Supposez que :
  - P1 a posé un verrou exclusif sur les octets [0,10[et demande un verrou exclusif sur les octets [18,30[sur fich.
  - P2 a posé un verrou exclusif sur les octets [12,28[sur fich et demande un verrou exclusif sur les octets [0,5[sur fich.

  Les deux processus sont-ils en interblocage ?
- Sous UNIX/linux, certains interblocages liés à la pose de verrous exclusifs sur des fichiers sont détectés et « évités ».
- La fonction **lockf** qui permet de poser des verrous peut échouer avec le message d'erreur : « **Resource deadlock avoided** ».

int lockf(int fd, int cmd, off_t len);

Si cmd = **F_LOCK** alors verrouiller l'accès aux *len* octets contigus en commençant par l'octet courant.

➔ **F_UNLOCK** pour déverrouiller l'accès.

Exercice 5 (suite)

```
int main()
{    // créer et remplir un fichier
     char c[38]="0123456789abcdefghijklmnopqrstuvwxyz\n";
     int fd= open("fich.txt",O_RDWR|O_CREAT, S_IRWXU);
     write(fd,&c,38);
     write(1,&c,38);
      close(fd);

  if (fork()==0)  // processus fils
   {   fd= open("fich.txt",O_RDWR);
       if( lockf(fd,F_LOCK,10) ==-1)
       {      printf("fils : %s  =>  exit(1)\n", strerror(errno)); exit(1);  }
       printf( "fils : verrou sur les octets [0,10[\n");
       write(fd,"xxxxxxxxxx",10);
       sleep(1);
       lseek(fd, 18, SEEK_SET);
       if( lockf(fd,F_LOCK,12) ==-1)
      {      printf("fils : %s  =>  exit(1)\n", strerror(errno)); exit(1); }
       printf( "fils : verrou sur les octets [18,30[\n");
       close(fd);
       printf("fils : fin normale \n");
   }
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <fcntl.h>
```

Exercice 5 (suite)

```
else { // processus père
        fd= open("fich.txt",O_RDWR);
        lseek(fd, 12, SEEK_CUR);
        if( lockf(fd,F_LOCK,16) ==-1)
        {     printf("parent : %s => exit(1)\n", strerror(errno)); exit(1);}
        printf( "pere : verrou sur les octets [12,28[\n");
        write(fd,"yyyyyyyyyyyyyyyy",16);
        sleep(1);
        lseek(fd, 0, SEEK_SET);
        if( lockf(fd,F_LOCK,5) ==-1)
        {     printf("parent :  %s => exit(1) \n", strerror(errno)); exit(1);}
        printf( "pere :  verrou sur les octets [0,5[\n");
        wait(NULL);
        lseek(fd,0,SEEK_SET);
        char x;
        while(read(fd,&x,1)>0)
              write(1,&x,1);
        close(fd);
        unlink("fich.txt");
        printf("pere : fin normale \n");
   }
   exit(0);
}
```

```
$ gcc lockftest.c -o lockftest
$ ./lockftest
0123456789abcdefghijklmnopqrstuvwxyz
pere : verrou sur les octets [12,28[
fils : verrou sur les octets [0,10[
fils : Resource deadlock avoided  => exit(1)
pere :  verrou sur les octets [0,5[
xxxxxxxxxxabyyyyyyyyyyyyyyyystuvwxyz
pere : fin normale
```

Lectures suggérées

Chapitre 6 Deadlocks (pp 435-463)
Modern operating Systems, 4nd edition, Andrew S. Tanenbaum, publié par Pearson Education, Prentice-Hall, 2016. **Livre disponible dans le dossier Slides Aut 2018 du site moodle du cours.**
).