Chapitre 2 Synchronisation

1. LE PROBLEME DE L'EXCLUSION MUTUELLE

On appelle processus indépendants des processus ne faisant appel qu'à des ressources locales. On appelle processus parallèles pour une ressource des processus pouvant utiliser simultanément cette ressource. Lorsque la ressource est critique (ou en accès exclusif), on parle d'exclusion mutuelle (par exemple, sur une machine monoprocesseur, l'UC est une ressource en exclusion mutuelle).

Def.: On appelle section critique la partie d'un programme où la ressource est seulement accessible par le processus en cours. Il faut s'assurer que deux processus n'entrent jamais en même temps en section critique sur une même ressource. A ce sujet, aucune hypothèse ne doit être faite sur les vitesses relatives des processus.

Exemple : la mise à jour d'un fichier (deux mises à jour simultanées d'un même compte client).La section critique comprend :

- lecture du compte dans le fichier,
- modification du compte,
- reecriture du compte dans le fichier.

1.1 Algorithme1 :

```
int tour ; /* variable de commutation de droit a la section critique */
/* valeur 1 pour P1, 2 pour P2 */
main ()
{
        tour = 1; /* P1 peut utiliser sa section critique */
        parbegin
                p1();
                p2() ;
        parend
}
/*************************************************************************/
p1()
{
for ( ; ; )
{
        while (tour == 2); /* c'est au tour de P2 ; P1 attend */
        crit1;
        tour = 2; /* on redonne l'autorisation a P2 */
        reste1;
}
}
/*************************************************************************/
p2()
{
for (; ; )
{
        while (tour == 1); /* c'est au tour de P1 ; P2 attend */
        crit2;
        tour = 1; /* on redonne l'autorisation a P1 */
        reste2 ;
}
}
```

Avantages:

- l'exclusion mutuelle est satisfaite. Pour chaque valeur de tour, une section critique et une seule peut s'exécuter, et ce jusqu'a son terme.
- l'interblocage est impossible puisque tour prend soit la valeur 1, soit la valeur 2 (Les deux processus ne peuvent pas être bloques en même temps).
- la privation est impossible: un processus ne peut empêcher l'autre d'entrer en section critique puisque tour change de valeur a la fin de chaque section critique.

Inconvénients :

- P1 et P2 sont contraints de fonctionner avec la même fréquence d'entrée en section critique
- Si l'exécution de P2 s'arrête, celle de P1 s'arrête aussi: le programme est bloqué.

1.2 Algorithme2

Chaque processus dispose d'une clé d'entrée en section critique (c1 pour P1, c2 pour P2). P1 n'entre en section critique que si la clé c2 vaut 1. Alors, il affecte 0 à sa clé c1 pour empêcher P2 d'entrer en section critique.

```
int c1, c2 ;
/* cles de P1 et P2 - valeur 0: le processus est en section critique */
/* valeur 1: il n'est pas en section critique */
main ()
{
        c1=c2 = 1; /* initialement, aucun processus en section critique */
        parbegin
                p1() ;
                p2 ();
        parend
}
/*************************************************************************/
p1 ()
{
for ( ; ; )
{
        while (c2 == 0); /* P2 en section critique, P1 attend */
        c1 = 0 /* P1 entre en section critique */
        crit1 ;
        c1 = 1; /* P1 n'est plus en section critique */
        reste1 ;
}
}
/*************************************************************************/
p2 ()
{
for ( ; ; )
{
        while (c1 == 0); /* P1 en section critique, P2 attend */
        c2 = 0; /* P2 entre en section critique */
        crit2 ;
        c2 = 1 ; /* P2 n'est plus en section critique */
        reste2;
}
}
```

Avantage :

On rend moins dépendants les deux processus en attribuant une clé de section critique à chacun.

Inconvénients :

Au début c1 et c2 sont à 1. P1 prend connaissance de c2 et met fin a la boucle while. Si la commutation de temps a lieu à ce moment, c1 ne sera pas à 0 et P2 évoluera pour mettre c2 à 0, tout comme le fera irrémédiablement P1 pour c1.

La situation c1 = c2 = 0 qui en résultera fera entrer simultanement P1 et P2 en section critique : l'exclusion mutuelle ne sera pas satisfaite.

Si l'instruction ci = 0 était placée avant la boucle d'attente, l'exclusion mutuelle serait satisfaite, mais on aurait cette fois interblocage.

A un moment donné, c1 et c2 seraient nuls simultanément P1 et P2 exécuteraient leurs boucles d'attente indéfiniment.

1.3 **Algorithme3**

Lorsque les deux processus veulent entrer en section critique au même moment, l'un des deux renonce temporairement.

```
int c1,c2; /* 0 si le processus veut entrer en section critique, 1 sinon */
main ()
{
        c1= c2 = 1; /* ni P1,ni P2 ne veulent entrer en section critique au depart */
        parbegin
                p1();
                p2 () ;
        parend
}
/*************************************************************************/
p1()
{
for ( ; ; )
{
        c1 = 0; /* P1 veut entrer en section critique */
        while (c2 == 0) /* tant que P2 veut aussi entrer en section critique ... */
        {
                c1 = 1; /* ....P1 abandonne un temps son intention... */
                c1 = 0; /* ..... puis la réaffirme */
        }
        crit1;
        c1 = 1; /* fin de la section critique de P1 */
        reste1 ;
}
}
/*************************************************************************/
p2()
{
for ( ; ; )
{
        c2 = 0 ; /* P2 veut entrer en section critique */
        while (c1 == 0) /* tant que P1 veut aussi entrer en section critique ... */
        {
                c2 = 1; /* .. P2 abandonne un temps son intention .... */
                c2 = 0 ; /* ..... puis la réaffirme */
        }
        crit2;
        c2 = 1 ; /* fin de la section critique de P2 */
        reste2 ;
}
}
```

Commentaires :
- l'exclusion mutuelle est satisfaite. (cf. ci-dessus)
- il est possible d'aboutir a la situation ou c1 et c2 sont nuls simultanement. Mais il n'y aura pas interblocage car cette situation instable ne sera pas durable (Elle est liée a la commutation de temps entre ci = 1 et ci = 0 dans while).
- il y aura donc d'inutiles pertes de temps par famine limitée.

1.4 Algorithme de DEKKER

DEKKER a propose un algorithme issu des avantages des 1ere et 3eme solutions pour résoudre l'ensemble du problème sans aboutir à aucun inconvénient. Par rapport à l'algorithme précédent, un processus peut réitérer sa demande d'entrée en section critique, si c'est son tour.

```
int tour , /* valeur i si c'est au tour de Pi de pouvoir entrer en section critique */
c1,c2 ; /* valeur 0 si le processus veut entrer en section critique,1 sinon */
main ()
{
        c1 = c2 = tour = 1; /* P1 peut entrer en section critique, mais... */
        parbegin /* ... ni P1, ni P2 ne le demandent */
                p1 () ;
                p2 () ;
        parend
}
p1 ()
{
for ( ; ; )
{
        c1 = 0; /* P1 veut entrer en section critique */
        while (c2 == 0) /* tant que P2 le veut aussi..... */
        if (tour == 2) /* ..... si c'est le tour de P2 ..... */
        {       c1 = 1; /* ..... P1 renonce ..... */
                while (tour == 2) ; /* ..... jusqu'a ce que ce soit son tour ..... */
                c1 = 0; /* ..... puis réaffirme son intention */
        }
crit1;
tour = 2; /* C'est le tour de P2 */
c1 = 1; /* P1 a achevé sa section critique */
reste1;
}
}
/*************************************************************************/
p2 ()
{
for (; ; )
{
        c2 = 0; /* P2 veut entrer en section critique */
        while (c1 == 0) /* tant que P1 le veut aussi ..... */
        if (tour == 1) /* ..... si c'est le tour de P1 ..... */
        {       c2 = 1; /* ..... P2 renonce ..... */
                while (tour == 1); /* ..... Jusqu'à ce que ce soit son tour .... */
                c2 = 0; /* ..... puis réaffirme son intention */
        }
crit2;
tour = 1; /* C'est le tour de P1 */
c2 = 1; /* P2 a achevé sa section critique */
reste2;
}
}
```

Remarques :
- Si p1 veut entrer en section critique (c1 = 0), alors que p2 le veut aussi (c2 = 0), et que c'est le tour de p1 (tour = 1), p1 insistera (while (c2 == 0) sera actif). Dans p2, la même boucle aboutira à c2 = 1 (renoncement temporaire de p2).

- Cet algorithme peut être généralisé a n processus.
- De manière générale, les algorithmes par attente active présentent un défaut commun : les boucles d'attente et le recours fréquent à la variable tour gaspillent du temps UC.

Nouvelle idee :
Mettre en sommeil un processus qui demande à entrer en section critique des lors qu'un autre processus y est déjà. C'est l'attente passive.

1.5 Les sémaphores

La résolution des problèmes multitâches a considérablement progresse avec l'invention des sémaphores par E.W. DIJKSTRA en 1965. Il s'agit d'un outil puissant, facile à implanter et à utiliser.
Les sémaphores permettent de résoudre un grand nombre de problèmes liés à la programmation simultanée, notamment le problème de l'exclusion mutuelle.

Un sémaphore est une structure à deux champs :
- une variable entière, ou valeur du sémaphore. Un sémaphore est dit binaire si sa valeur ne peut être que 0 ou 1, général sinon.
- une file d'attente de processus ou de taches.

Dans la plupart des cas, la valeur du sémaphore représente à un moment donné le nombre d'accès possibles à une ressource.

Seules deux fonctions permettent de manipuler un sémaphore :
- *P (s) ou down (s) ou WAIT (s)*
- *V (s) ou up (s) ou SIGNAL (s)*

1.5.1 P (s)

La fonction P(S) décrémente le sémaphore d'une unité à condition que sa valeur ne devienne pas négative.

```
debut
a ← 1 /* a est un registre */
TestAndSet (&a, &verrou) /* copie le contenu de verrou (variable globale
initialisée à 0) dans a et range 1 dans le mot mémoire verrou.
TestAndSet est ininterruptible Exécutée par le matériel, elle permet de
lire et d'écrire un mot mémoire */
tant que a = 1 /* si a = 0, le verrou est libre et on passe, sinon le
verrou est mis */
TestAndSet (&a, &verrou)
fin tant que
si (valeur du semaphore > 0)
alors décrémenter cette valeur
sinon - suspendre l'exécution du processus en cours qui a appelé P(s),
- placer le processus dans la file d'attente du semaphore,
- le processus passe de l'état ACTIF à l'état ENDORMI.
finsi
verrou ← 0
fin
```

2.5.2 V (s)

La fonction V(S) incrémente la valeur du sémaphore d'une unité si la file d'attente est vide et si cette incrémentation est possible.

```
debut
a ← 1 /* a est un registre */
TestAndSet (&a, &verrou)
tant que a = 1 /* si a = 0, le verrou est libre et on passe, sinon le
verrou est mis */
TestAndSet (&a, &verrou)
fin tant que
si (file d'attente non vide)
alors
- choisir un processus dans la file d'attente du sémaphore,
- réveiller ce processus. Il passe de l'état ENDORMI à l'état ACTIVABLE.
sinon incrémenter la valeur du sémaphore si c'est possible
finsi
verrou ← 0
fin
```

Remarques:

1. Du fait de la variable globale verrou, les fonctions P et V sont ininterruptibles (on dit aussi atomiques). Les deux fonctions P et V s'excluent mutuellement. Si P et V sont appelées en même temps, elles sont exécutées l'une après l'autre dans un ordre imprévisible. Dans un système multiprocesseur, les accès à la variable partagée verrou peuvent s'entrelacer. Il est donc nécessaire de verrouiller le bus mémoire chaque fois qu'un processeur exécute TestAndSet.
2. On supposera toujours que le processus réveillé par V est le premier entré dans la file d'attente, donc celui qui est en tête de la file d'attente.
3. L'attente active sur a consomme du temps UC.

1.5.3 Application des sémaphores à l'exclusion mutuelle

Avec le schéma de programme précédent :

```
SEMAPHORE s; /* déclaration très symbolique */
main ()
{
SEMAB (s,1); /* déclaration très symbolique ; initialise le sémaphore
binaire s à 1 */
parbegin
p1 () ;
p2 () ; /* instructions très symboliques
parend */
}
/*************************************************************************
*****/
p1 () /* premier processus */
{
for ( ; ; )
{
P (s);
.................. /* section critique de p1 */
V (s);
.................. /* section non critique de p1 */
}
}
/*************************************************************************
****/
p2 () /* second processus */
```

```
{
for ( ; ; )
{
P (s);
.................. /* section critique de p2 */
V (s);
.............. /* section non critique de p2 */
}
}
```