Chapitre 13

Algorithmes Distribués & Interblocages

Ce chapitre introduit les problèmes liés à la gestion de processus concurrents. Le problème à resoudre est le partage de ressources entre différents processus asynchrones. Les I.P.C. et les verrous sont deux types d'outils permettant le partage asynchrone de ressources entre processus.

13.1 exemples

13.1.1 Les méfaits des accès concurrents

L'exemple le plus simple est une variable entière partagée par deux processus ou threads, ou bien manipulé par une fonction asyncrone comme un handler de signal. Supposont que l'on définise deux fonctions de manipulation de la variable :

```
int getValue();
void setValue(int );
```

Pour incrémenter la variable, il suffit d'exécuter `setValue(getValue()+1) ;` mais décomposont en `int tmp=getValue() ; setValue(tmp+1) ;` ce qui ne change pas grand chose `tmp` étant s'implement allouée sur la pile dans le premier cas. Regardont l'éxecution suivant du code par deux thread :

```
int tmp1=getValue();
                    | int tmp2= getValue();
setValue(tmp1+1);
                    | setValue(tmp2+1);
```

Que c'est il passé ?
la variable n'a été incrémentée qu'une fois !

Le cas d'un signal le code c de prog

```
#include <stdio.h>
#include <signal.h>
int nbi; // nbre d'incrementation accès atomique
int partage; // nbr d'incrémentation  accès non atomique

int getValue() { return partage; }
```

```
void setValue(int x) { partage = x; }

void handler(int s)
{
int tmp=getValue();
setValue(tmp+1);
nbi++;
}


int
main(int c, char *argv[])
{
struct sigaction sig;
long diff = 0;
sig.sa_handler= handler;
sig.sa_flags = 0;

sigaction(SIGUSR1, &sig, NULL);
fprintf(stderr,"sigusr1= %d\n",SIGUSR1);
fprintf(stderr,"%d %d\n", nbi,partage);

for(;;)
{
        int tmp=getValue();
        setValue(tmp+1);
        nbi++;
if ((partage != nbi) && (diff!= nbi-partage))
        { diff = nbi-partage; fprintf(stderr,"%d\n", nbi-partage); }
}
}
```

13.1.2 Exclusion mutuelle

Problème : il y a une rivière que l'on peut traverser par un gué fait de pierre alignées, où il n'est pas possible de se croiser, et il n'est pas possible de faire demi-tour. Comment doit-t-on organiser le passage ?

Solutions :

1. regarder avant de traverser
2. si deux personnes arrivent en même temps sur chaque rive,
   si elles avancent en même temps $\longrightarrow$ interblocage
   si elles attendent en même temps $\longrightarrow$ interblocage
3. Un remède : un côté prioritaire $\longrightarrow$ famine. En effet si le coté OUEST est prioritaire et qu'un flot continu de personnes arrive de ce côté, les personnes à l'EST sont bloquées indéfiniment.
4. Une solution : alterner les priorités.

Pour des ressources système comme les fichiers, le partage n'est pas géré par le SGF. Il faut donc un mécanisme de partage : les verrous, qui permettent un partage dynamique et partiel (portions de fichiers). Pour un partage entre utilisateurs, on utilise plutôt des outils comme `SCCS, RCS`.

13.2 Mode d'utilisation des ressources par un processus.

Formalisons les opérations réalisables sur une ressource.
- requête : demande bloquante de ressources
- utilisation : lecture/écriture sur la zone verrouillée
- libération : verrou L-type

13.3 Définition de l'interblocage (deadlock)

Un ensemble de processus est en **interblocage** si et seulement si tout processus de l'ensemble est en attente d'un évènement qui ne peut être réalisé que par un autre processus de l'ensemble.

Exemple :

Le processus A possède un verrou de portée [0,400] sur un fichier f, et demande un verrou de portée [800,1000] sur ce même fichier, alors qu'un processus B possède un verrou de portée [600,900] sur le fichier f et demande un verrou de portée [0,33] sur f. Les deux processus sont en interblocage. Dans le cas de la pose de verrous sous UNIX, il y a détection de cet interblocage et la commande `fcntl` échoue.

13.4 Quatre conditions nécessaires à l'interblocage.

Les conditions suivantes sont **nécessaires** pour avoir une possibilité d'interblocage.

Exclusion mutuelle les ressources ne sont pas partageables, un seul processus à la fois peut utiliser la ressource.

Possession & attente il doit exister un processus qui utilise une ressource et qui est en attente sur une requête.

Sans préemption les ressources ne sont pas préemptibles c'est-à-dire que les libérations sont faites volontairement par les processus. On ne peut pas forcer un processus à rendre une ressource. (Contre exemple : le CPU sous Unix est préemptible)

Attente circulaire il doit exister un ensemble de processus P_i tel que P_i attend une ressource possédée par P_{i+1}.

Les quatre conditions sont nécessaires pour qu'une situation d'interblocage ait lieu.
Exercice : montrer que pour les verrous, les quatre conditions tiennent.
Exercice : montrer que si l'une des condition n'est pas vérifiée alors il ne peut y avoir d'interblocage.

13.5 Les graphes d'allocation de ressources

Les graphes d'allocation de ressources permettent de décrire simplement les problèmes d'interblocage.

`G = (N,T) N = P U R`

P : ensemble des processus
R : ensemble des ressources

T est inclus dans RXP U PXR
Soit le couple (x,y) appartenant à T,
si (x,y) appartient à RXP, cela signifie que la ressource x est utilisée par le processus y.
si (x,y) appartient à PXR, cela signifie que le processus x demande la ressource y.