

Programmation systèmes avancée

Wieslaw Zielonka

zielonka@irif.fr

Les verrous des fichiers
les verrous flock

Verrous de fichier

Pourquoi les verrous ?

Problème d'accès parallèle de plusieurs processus à un fichier, "race condition".

Verrouillage de fichier

Verrous :

- poser un verrou
- effectuer une opération (ou des opérations) de lecture/écriture sur un fichier
- enlever le verrou.

Deux types de verrous :

- les verrous consultatifs (advisory) : un processus peut ignorer les verrous posés par d'autres processus et accéder au fichier. Pour que les verrous puissent avoir un effet tous les processus doivent coopérer et utiliser le mécanisme de verrouillage. Par défaut dans le système UNIX les verrous sont consultatifs, ce sont les seuls verrous dans la norme POSIX.
- les verrous obligatoires (mandatory) : le processus qui essaie d'accéder au fichier verrouillé et qui n'utilise pas le mécanisme de verrouillage sera contraint à respecter les conditions imposées par le verrouillage.

Verrous de fichier

Si un processus modifie le fichier (ou une partie du fichier) d'autres processus ne peuvent ni lire ni écrire.

Le processus écrivain possède le verrou exclusif.

Plusieurs processus peuvent lire le fichier en même temps, le même fichier ne peut pas être modifié en même temps.

Les processus lecteurs possèdent le verrou partagé.

Verrous flock()

Les verrous BSD, ne font pas partie de POSIX mais implémentés dans presque tous les systèmes UNIX.

```
#include <sys/file.h>

int flock(int fd, int operation)
```

pose le verrou ou lève le verrou **sur la totalité du fichier**.
Retourne 0 en cas de succès, -1 sinon.

- fd - un descripteur de fichier valide

Verrous flock() - opérations

opération	description
LOCK_SH	placer le verrou partagé sur un fichier dont le descripteur est fd
LOCK_EX	placer le verrou exclusif sur le fichier fd
LOCK_UN	déverrouiller le fichier dont le descripteur est fd
LOCK_NB	faire une demande de verrou non bloquante

Verrous flock()

Plusieurs processus peuvent posséder en même temps les **verrous partagés** sur le même fichier. On pose le verrou partagé pour effectuer une **lecture**.

A chaque moment seulement au plus un processus peut posséder un **verrou exclusif** sur un fichier, d'autres processus ne peuvent pas avoir ni verrou exclusif ni partagé pendant ce temps. On pose le verrou exclusif pour effectuer des **écritures** et/ou des lectures.

Il est possible de convertir un verrou d'un type (partagé/exclusif) vers un autre type (exclusif/partagé) avec l'appel à flock() en spécifiant le nouveau type de verrou.

Conversion de verrou partagé vers exclusif peut être bloquante (s'il y a d'autres processus possédant les verrous partagés). La conversion de verrou n'est pas atomique, elle se déroule en deux étapes

1. la libération de verrou que le processus possède suivi de
2. la tentative d'obtention de nouveau verrou.

Mais entre 1 et 2 un autre processus peut obtenir le verrou donc notre processus peut perdre le verrou après 1 et être bloqué sur l'opération 2.

Verrous flock()

L'opération d'acquisition de verrou est normalement bloquante (le processus bloque jusqu'à l'obtention de verrou).

On ajoute LOCK_NB pour la rendre **non-bloquante**:

```
int n = flock( fd, LOCK_EX | LOCK_NB );  
if( n == -1 && errno == EWOULDBLOCK ){  
    // c'est le cas ou le verrou n'est pas acquis parce que  
    // d'autres processus détiennent un verrou non-compatible  
    // avec le verrou exclusif demandé.  
    .....  
}
```

De la même façon LOCK_SH | LOCK_NB à utiliser pour la demande de verrou partagé non-bloquante.

Verrous flock() et fork, exec

Les verrous flock() sont associés à la table de fichier ouvert.

Donc si deux descripteurs pointent vers la même entrée dans la table de fichiers ouverts alors les deux processus possèdent le même verrou.

Conséquences:

- A. processus A obtient le verrou sur un fichier xyz.txt
 - B. processus A fait fork() et crée un processus enfant B
 - C. les deux processus A et B possèdent le verrou (**même dans le cas de verrou exclusif**).
-

Ce comportement permet de passer le verrou flock() du parent vers l'enfant de manière atomique :

1. processus A obtient le verrou et fait fork() en créant l'enfant B
2. le parent fait close() sur le descripteur (**sans déverrouillage!**), à partir de ce moment seulement l'enfant détient le verrou
3. l'enfant fait exec() -- le verrou est préservé à travers de exec().

Verrous flock() et dup/dup2

Quand un processus duplique un descripteur avec dup() ou dup2() les deux descripteurs partagent toujours le verrou. Quand on libère le verrou sur un descripteur le verrou est automatiquement libéré sur l'autre descripteur.

```
flock( fd, LOCK_EX );
```

```
newfd = dup( fd );
```

```
flock( newfd, LOCK_UN); // libère le verrou sur newdf  
                        // mais aussi sur fd
```

Si on ouvre le même fichier deux fois avec open() les deux descripteurs sont traités de façon indépendante par les verrous flock.

limitation de flock

- pas de possibilité d'obtenir le verrou sur une partie du fichier, ce qui limite le parallélisme,
- seulement les verrous consultatifs
- le système de fichier NFS souvent ne prend pas en compte les verrous flock

**Les verrous sur une partie
de fichier (record locking)
fcntl**

verrous fcntl

Ce sont les verrous POSIX. Permettent de verrouiller une partie du fichier. Au lieu de parler de verrous de fichiers il serait plus approprié de parler de verrous sur un intervalle de fichier.

```
struct flock flockstr;
```

```
/* remplir la structure flockstr */
```

```
fcntl( fd, cmd,  &flockstr);
```

fd - descripteur de fichier dont nous voulons verrouiller ou déverrouiller une partie.

cmd - commande

verrous fcntl - struct flock

```
struct flock{  
    short    l_type; //type de verrou : F_RDLCK, F_WRLCK, F_UNLCK  
    short    l_whence ; // valeurs SEEK_SET, SEEK_CUR,  
                        // SEEK_END  
    off_t     l_start; //offset du début de verrou relatif à l_whence  
    off_t     l_len; // longueur de l'intervalle en octets, peut-être < 0  
    pid_t     l_pid; // pid du processus qui empêche la pose de verrou  
                //retourné si cmd == F_GETLK  
}
```

verrous fcntl - paramètre cmd

le paramètre cmd de fcntl prend une de trois valeurs:

F_SETLK - obtenir ou libérer un verrou. Si l'acquisition de verrou impossible parce qu'un autre processus détient un verrou incompatible alors fcntl() retourne tout de suite -1 et errno prend la valeur EAGAIN ou EACCES

F_SETLKW - comme précédent, mais l'appel est bloquant jusqu'à ce que l'acquisition de verrou devienne possible (W à la fin == wait). Un signal peut interrompre l'attente (errno == EINTR). Utile pour l'alarme ou timeout.

F_GETFLK - permet de vérifier si l'acquisition de verrou est possible (sans le demander). Le type de verrou *l_type* de struct flock doit être **F_RDLCK** ou **F_WRLCK**.

Si l'acquisition de verrou est possible *l_type* prendra la valeur **F_UNLCK**.

Si l'acquisition de verrou est impossible struct flock retournera l'information sur un verrou incompatible détenu par un autre processus avec l'identité de ce processus dans le champ *l_pid*.

jusqu'à la fin de fichier

`l_len == 0`

indique qu'on demande de verrouiller tous les octets à partir d'une position donnée jusqu'à la fin de fichier et que ***`l'intervalle verrouillé s'étend automatiquement quand la taille de fichier augmente.`***

```
struct flock fl;
```

```
fl.l_type = F_WRLCK; fl.l_start = 0;
```

```
fl.l_whence = SEEK_END; fl.l_fl.l_len = 0;
```

```
fcntl( fd, F_SETLK, &fl ); /* le verrou s'étend sur tous  
le nouveau octets qui seront ajoutés à la fin de fichier */
```

verrous fcntl

- libérer un verrou qu'on ne possède pas ne pose pas de problème
- changer le verrou sur certain segment de fichier est atomique (contrairement à flock)
- un processus ne peut jamais avoir le conflit de verrou avec le verrou qu'il a posé lui-même (contrairement à flock)
- placer un verrou exclusif au milieu de segment déjà verrouillé par un verrous partagé peut diviser le segment en trois segments avec de verrous distincts.
- avec F_SETLKW un deadlock (inter-blocage) est possible. Le noyau détecte quand le deadlock survient, l'appel à fcntl termine avec échec et `errno == EDEADLK` indique que l'échec est du à un deadlock.

deadlock

Le fichier "toto" contient deux octets : ab

- processus A pose le verrou exclusif sur le premier octet
- processus B pose le verrou exclusif sur le deuxième octet
- processus A tente de poser le verrou sur le deuxième octet
- processus B tente de poser le verrou sur le premier octet

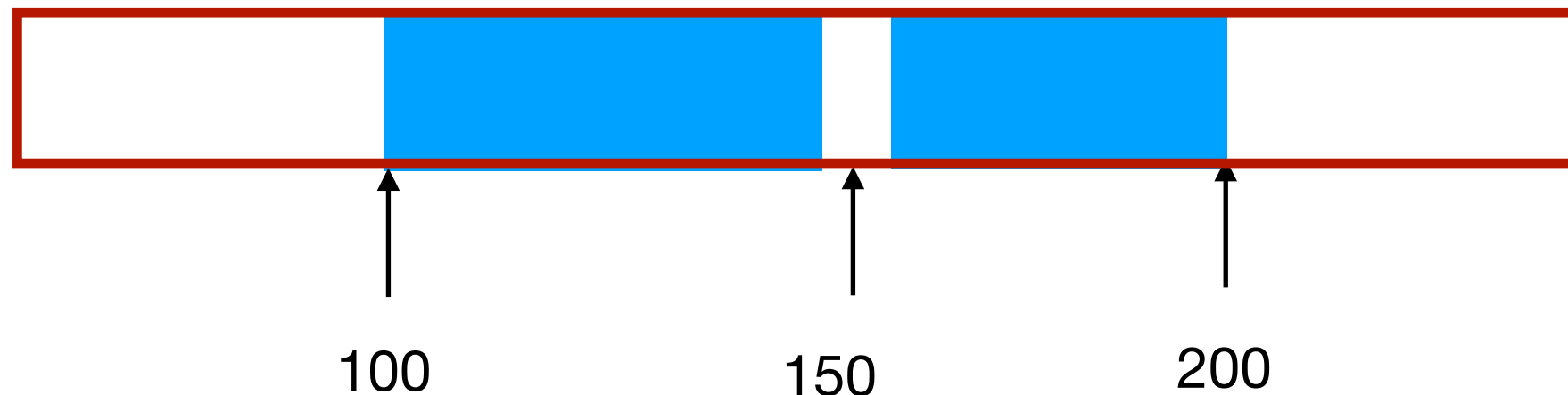
La deuxième tentative échoue pour un des deux processus et
`errno == E_DEADLK`

intervalles de verrou

processus A possède un verrou sur l'intervalle entre 100 et 200 :



processus A libère le verrou sur l'octets 150



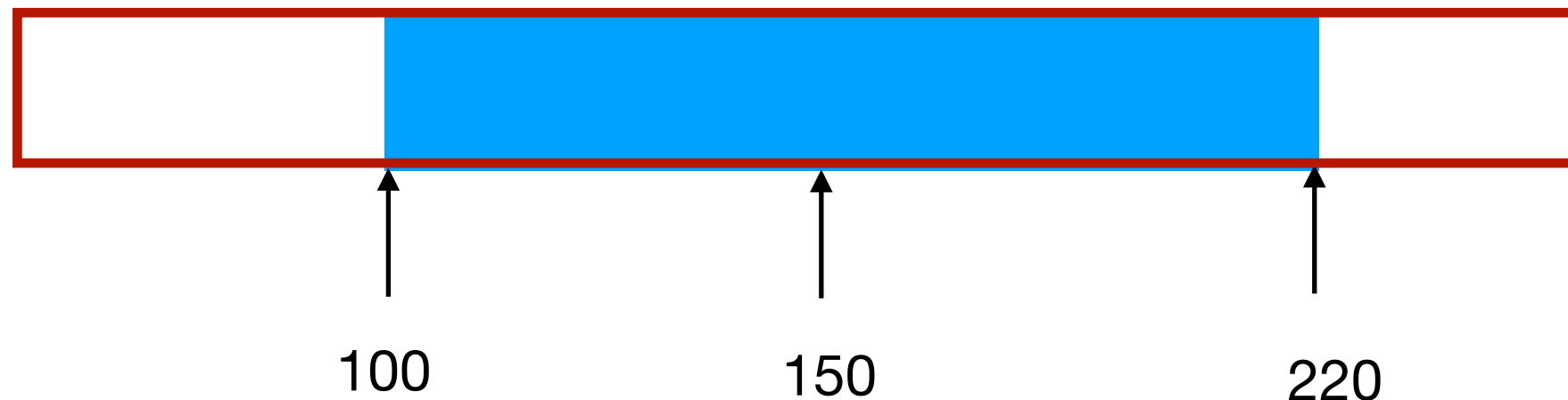
l'intervalle est divisé en deux

intervalles de verrou

processus A possède une verrou sur l'intervalle entre 100 et 170 :



processus A demande le même type de verrous sur l'intervalle 150 - 220



processus A possède maintenant le verrou sur l'intervalle 100 - 220

la famine

POSIX ne spécifie pas ce qui se passe dans la situation suivante :

- processus A pose un verrou partagé sur un intervalle
- processus B tente de poser un verrou exclusif en conflit avec le verrou précédent (et il bloque)
- processus C tente de poser un verrou partagé sur le même intervalle.
- processus A lève le verrou

Il est possible que processus C obtienne le verrou et B continue à être bloqué. Les processus qui posent les verrous partagés peuvent bloquer de façon permanente le processus B

les règles

Les verrous sont associés au couple (processus, fichier).

Cela implique plusieurs choses :

1. quand le processus termine tous les verrous qu'il possède sont levés.

les règles

2. si le processus ferme un descripteur alors tous les verrous qu'il a posé sur le même fichier sont levés, peu importe si c'est le même descripteur ou non.

```
fd1 = open( chemin, .... );  
read_lock(fd1, ... ); /* fonction qui pose le verrou fcntl */  
fd2 = open(chemin, ... )  
close(fd2); /* cela enlève le verrou sur fd1 */
```

cela peut poser des problèmes : le processus utilise des fonctions d'une librairie qui pose les verrous sur les fichiers (mais nous ne les savons pas, peut-être nous n'avons pas de code de la librairie). Notre programme ouvre le même fichier et le ferme en supprimant les verrous posés par la librairie.

les règles

3. Les verrous fcntl posés par un parent ne sont pas hérités par le processus enfant
4. les verrous fcntl sont préservés par un exec..()

Implementation FreeBSD

