Cours « système d'exploitation »
2ème année
Département d'Informatique

Chapitre 5

Echange de données entre Processus :

les Tubes et les Verrous

Plan

5.1 Les tubes

La commande "**ps -a | wc -l**" entraîne la création de deux processus concurrents (allocation du processeur). Un tube est créé dans lequel les résultats du premier processus ("**ps -a**") sont écrits. Le second processus lit dans le tube.

Lorsque le processus écrivain se termine et que le processus lecteur dans le tube a fini d'y lire (le tube est donc vide et sans lecteur), ce processus détecte une fin de fichier sur son entrée standard et se termine.

Le système assure la synchronisation de l'ensemble dans le sens où :

- il bloque le processus lecteur du tube lorsque le tube est vide en attendant qu'il se remplisse (s'il y a encore des processus écrivains);
- il bloque (éventuellement) le processus écrivain lorsque le tube est plein (si le lecteur est plus lent que l'écrivain et que le volume des résultats à écrire dans le tube est important).

Le système assure l'implémentation des tubes. Il est chargé de leur création et de leur destruction.

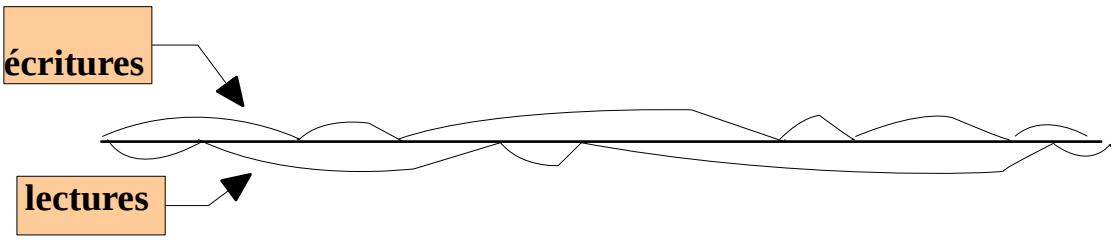
Un tube de communication (|) permet de mémoriser des informations. Il se comporte comme une file FIFO, d'où son aspect unidirectionnel.

Un tube est presque identique à un fichier ordinaire. Il est caractérisé par :

- ☛ son numéro d'i-noeud (sur le disque logique des tubes) ;
- ☛ aucune référence dans le système de fichier (fichier anonyme);
- ☛ deux descripteurs de fichiers (lecture et écriture);
- ☛ sa taille limitée (nombre d'adresses directes que contient un noeud du système de fichier) : d'où la notion de tube plein;
- ☛ deux extrémités, permettant chacune soit de lire dans le tube, soit d'y écrire;
- ☛ au plus deux entrées dans la table des fichiers ouverts (une pour la lecture et une pour l'écriture);
- ☛ l'opération de lecture dans un tube est destructrice : une information ne peut être lue

qu'une seule fois dans un tube;

☛ communication d'un flot continu de caractères (STREAM) : les envois successifs d'informations dans un tube apparaissent du point de vue de leur extraction comme une seule et même émission, d'où la possibilité de réaliser les opérations de lecture dans un tube sans relation avec les opérations d'écriture ;



☛ le **nombre de lecteurs** : c'est le nombre de descripteurs associés à l'entrée en lecture sur le tube (dans la table des fichiers ouverts). La nullité de ce nombre interdit toute écriture sur le tube ;

☛ le **nombre d'écrivains** : c'est le nombre de descripteurs associés à l'entrée en écriture sur le tube dans la table des fichiers ouverts. La nullité de ce nombre

détermine le comportement de la primitive **read** lorsque le tube est vide et permet en particulier de définir la notion de "fin de fichier" sur un tube.

L'impossibilité pour ce nombre de devenir nul (par suite d'une erreur de programmation) est susceptible de conduire à des situations d'interblocage (**deadlock**) d'un ensemble de processus.

Il existe deux sortes de tube :

- les tubes ordinaires ou non-nommés et
- les tubes nommés (FIFO).

Nous venons de décrire les tubes ordinaires. Les tubes nommés possèdent une référence dans le système de fichier afin de permettre à des processus sans lien de parenté particulier de communiquer en mode flot.

Les tubes ordinaires

Un **tube ordinaire n'étant pas nommé**, il est impossible de l'ouvrir en utilisant la primitive **open**. Pour utiliser un tube il faut connaître un descripteur associé à son entrée dans la table des fichiers dans le mode correspondant (lecture ou écriture).

L'acquisition d'un tel descripteur est réalisée de deux manières :

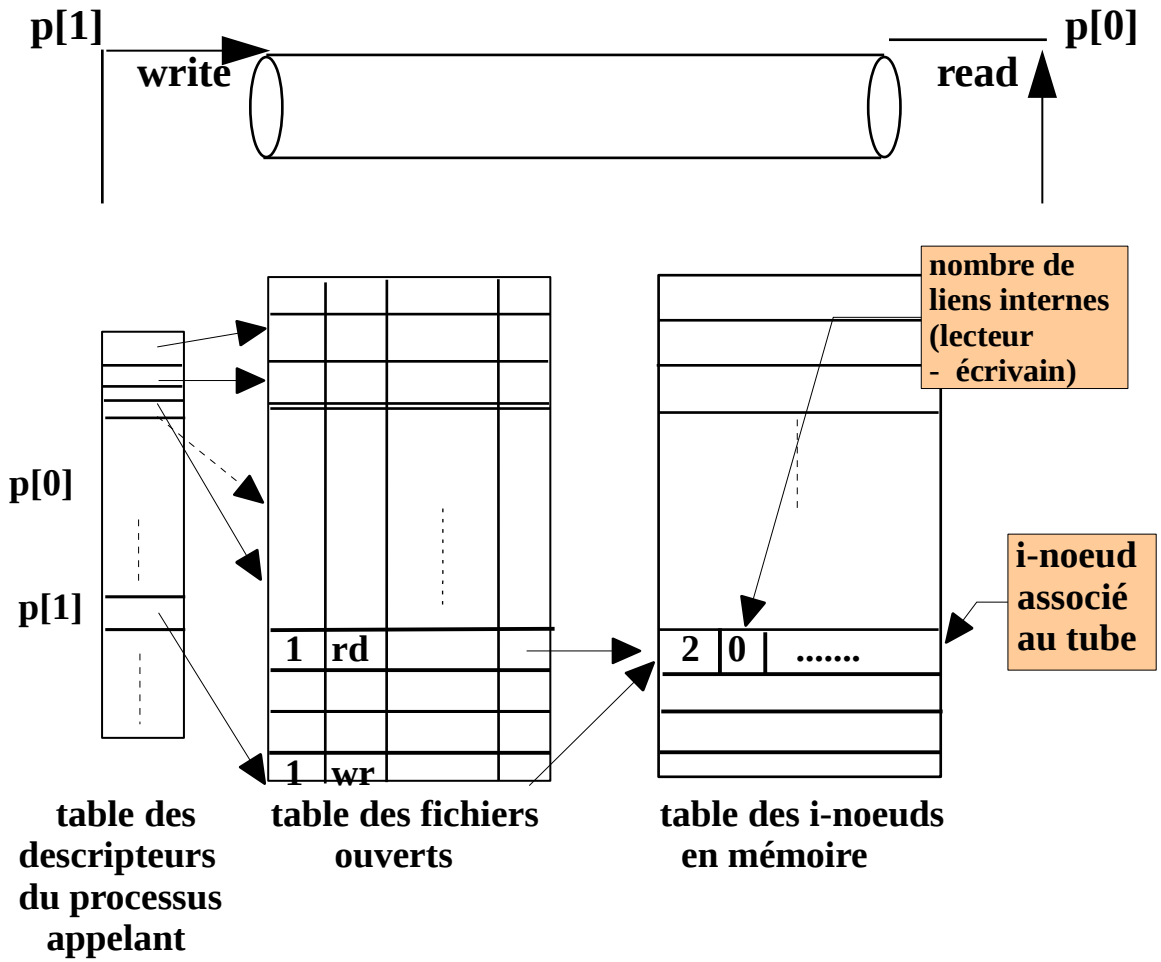
- en appelant la primitive de création de tube (**pipe**) ;
- par héritage : un processus héritant à sa création des descripteurs que possède son père, hérite en particulier des descripteurs de tubes.

La primitive de création d'un tube ordinaire (non nommé) : **pipe**

```
# include <unistd.h>
int pipe (int p[2]);
```

p[0] correspond au descripteur en mode lecture

p[1] correspond au descripteur en mode écriture


p[1]
write
p[0]
read
nombre de liens internes (lecteur - écrivain)
p[0]
p[1]
1 rd
1 wr
2 0 .....
i-noeud associé au tube
table des descripteurs du processus appelant
table des fichiers ouverts
table des i-noeuds en mémoire

Exemple d'utilisation des tubes ordinaires (fichier "prog.c") :

```
# include <stdio.h>
# include <unistd.h>
# include <errno.h>

int tube[2];
char buf[20];

main() {

      if(pipe(tube))
        { perror("pip");exit(1);}

        switch (fork()) {
              case -1 : perror("fork");exit(2) ;
              case 0: /* fils */
                    close (tube[1]);
                    read (tube[0], buf, 8);
                    printf ("%s bien recu \n",
                    buf);break ;
              default : /* père */
                    close (tube[0]);
                    write (tube[1], "bonjour", 8);
             }
        }
```

L'exécution de ce programme ("prog") donne le résultat suivant :

```
$ prog
bonjour bien recu
$
```

La duplication de descripteur

La duplication de descripteur permet à un processus d'acquérir un nouveau descripteur (dans sa table des descripteurs) synonyme d'un descripteur déjà existant.

Ce mécanisme est principalement utilisé pour réaliser des redirections des trois fichiers d'entrée-sorties standard.

Il repose sur le fait que **le descripteur synonyme renvoyé est toujours le plus petit descripteur disponible, dans la table des descripteurs du processus demandeur, qui satisfait une condition particulière**.

La duplication associe donc un descripteur supplémentaire à une entrée existante dans la table des fichiers ouverts.

La primitive

```
#include <unistd.h>
int dup (int desc);
```

associe le plus petit descripteur disponible du processus appelant, à la même entrée dans la table des descripteurs ouverts que *desc*.

En cas de réussite, la valeur retournée est le descripteur synonyme.

Si l'on veut rediriger la sortie standard (descripteur "**1**") du processus courant vers le tube (**tube[1]**) dont il a accès, il suffit de fermer cette sortie (**close (1)**), puis de faire une duplication de la sortie en écriture dans le tube : "**dup(tube[1])**".

Ensuite il ne restera plus qu'à fermer le descripteur "**tube[1]**" devenu inutile.

Enfin les écritures standard (**fwrite**, **printf**, ...) iront écrire directement dans le tube.

Les tubes nommés : FIFO

Ils permettent de transmettre des données entre des processus qui ne sont pas attachés par des liens de parenté.

Pour cela il faut nommer les tubes créés en les créant ; dans le shell on utilise la commande "**mknod**" ou plus généralement (X/Open) la commande "**mkfifo**" :

```
$ mkfifo  nomfichier
```

En "C" on utilisera l'interface suivante (**mknod** existe aussi mais n'est pas conseillée pour la portabilité du code) :

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *nomfichier, mode_t mode);
```

"**nomfichier**" définit le chemin d'accès au tube nommé et "**mode**" les droits d'accès des différents utilisateurs à cet objet.

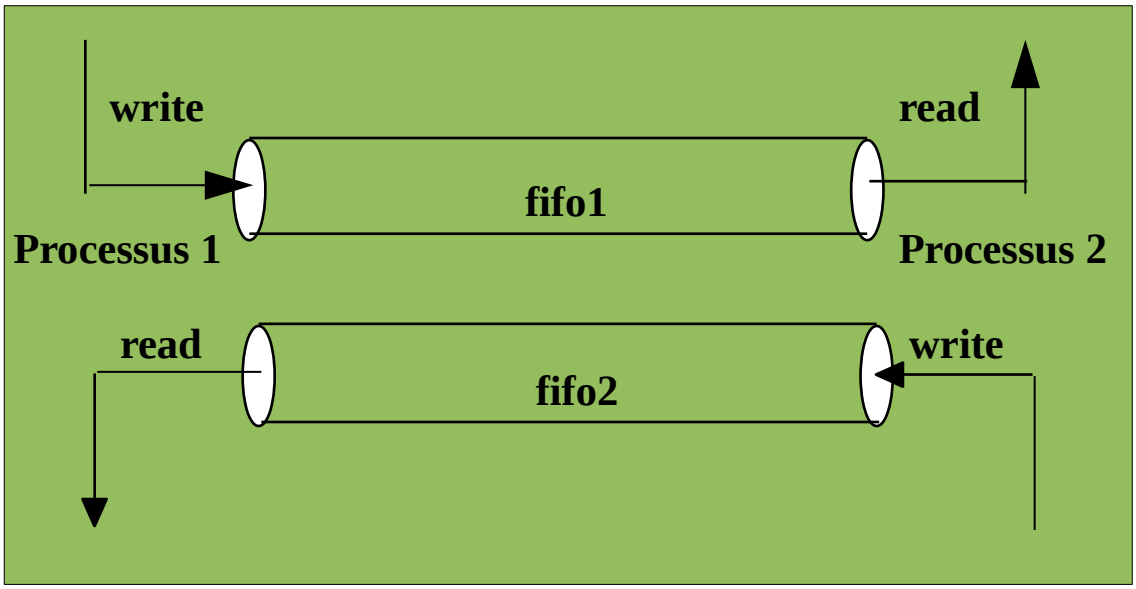
L'ouverture d'un tube nommé est bloquante par défaut (on utilise **O_NONBLOCK** sinon).

Pour accéder à un tube nommé un processus devra faire un "open" sur le fichier correspondant. Si cette ouverture est faite en lecture et qu'aucun processus n'a fait une ouverture en écriture, alors le processus courant est endormi et réciproquement avec l'ouverture en écriture.

C'est un moyen pour deux processus de **faire un point de rendez-vou**s. En effet il suffit que l'un demande l'ouverture en écriture et l'autre en lecture. Ils seront synchronisé par le système sur le deuxième "**open**".

Il faudra néanmoins, lors de dialogues dans les deux sens entre deux processus, s'assurer que les ordres d'ouverture sont faits dans le bon sens, sinon c'est l'**interblocage** ! Les séquences d'ouvertures correctes sont donc :

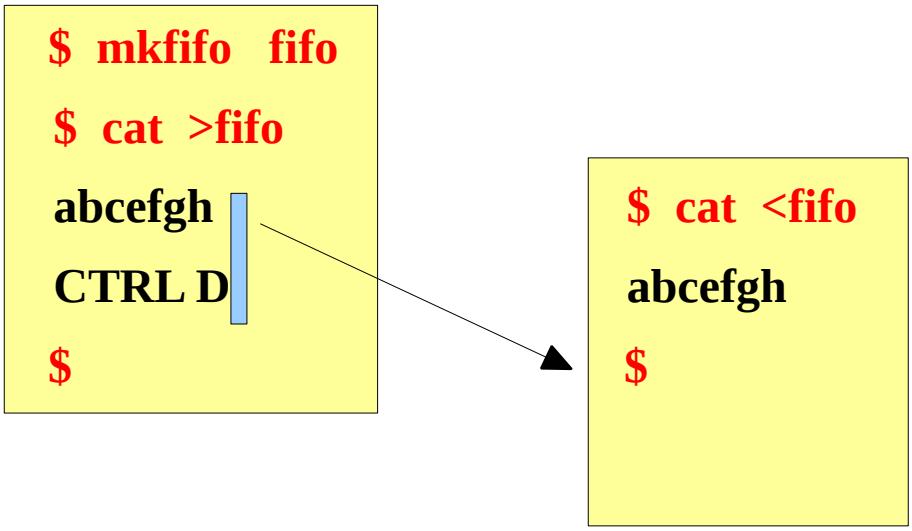
```
/* processus  1  */
int d_lecture, d_ecriture ;
...
d-ecriture=open("fifo1", O_WRONLY);
d_lecture=open("fifo2", O_RDONLY);
/* processus  2  */
int d_lecture, d_ecriture;
    ...
d-ecriture=open("fifo2", O_WRONLY) ;
d_lecture=open("fifo1", O_RDONLY);
```



Deux exemples d'utilisation des tubes nommés

a) Communication de processus dans deux fenêtres X-Windows

Deux processus sont lancés depuis les shells principaux de deux fenêtre X-Windows. Ils communiquent via un tube nommé ("fifo") en utilisant la commande "cat". Le tampon de la bibliothèque standard est vidé dans le shell du processus *lecteur*, dès que le processus *écrivain* écrit dans le tube (sortie standard vers "fifo").



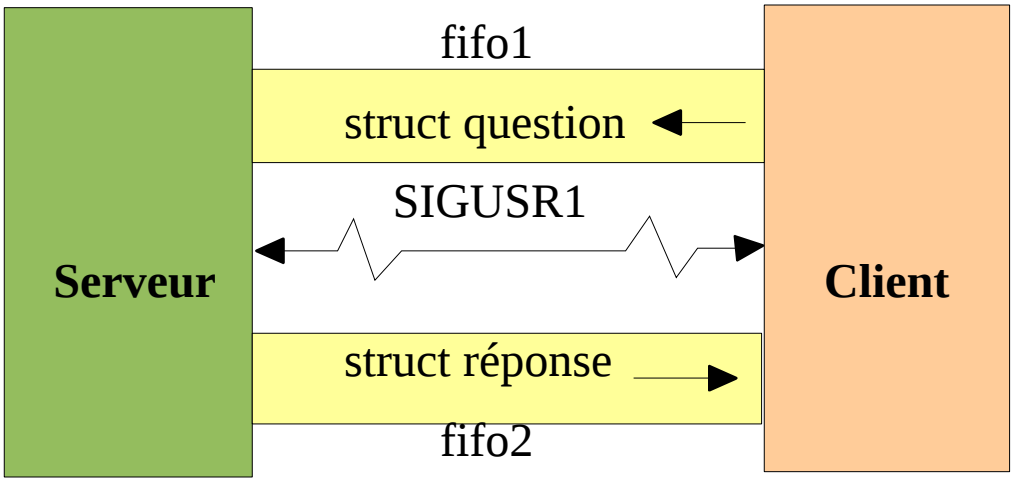
b) Communication client / serveur

Un serveur attend des questions de clients dans un tube *fifo1*. Une question correspond à la demande d'envoi de **n** nombres tirés au sort par le serveur (**n** est un nombre aléatoire compris entre **1** et **NMAX** tiré au sort par le client).

Dans sa question, le client envoie également son numéro (**PID**) de telle sorte que le serveur puisse le réveiller par l'intermédiaire du signal **SIGUSR1** quand il a écrit la réponse.

En effet, plusieurs clients pouvant être en attente de réponses dans le même tube, il est nécessaire de définir un protocole assurant que chaque client lit les réponses qui lui sont destinées.

Le client avertit par ce même signal le serveur quand il a lu les réponses.



5.2 Les verrous

Il s'agit d'un mécanisme général visant à assurer un contrôle de la concurrence des accès à un même fichier régulier. La norme POSIX se base sur les mécanismes issues des implantation SYSTEM V.

Les verrous sont attachés à un i-noeud, c'est-à-dire que l'effet d'un verrou sur un fichier est visible au travers de tous les descripteurs (et donc tous les fichiers ouverts) correspondant à ce noeud.

- **Un verrou est la propriété d'un processus** : seul le processus propriétaire d'un verrou peut le modifier et l'enlever.
- **La portée du verrou** : c'est l'ensemble des positions dans le fichier (ensemble des valeurs de l'offset) auxquelles il s'applique. Les verrous utilisables ici peuvent porter sur tout intervalle [entier1 : entier2] ou [entier1 : ∞] (-> fin de fichier).

Les verrous sont soit **partagés** (shared), soit **exclusifs** (exclusive). Cela porte sur la cohabitation ou non de plusieurs verrous sur des intervalles non-disjoints.

Les modes opératoires des verrous sont soit le mode **consultatif** (advisory), soit le mode **impératif** (mandatory).

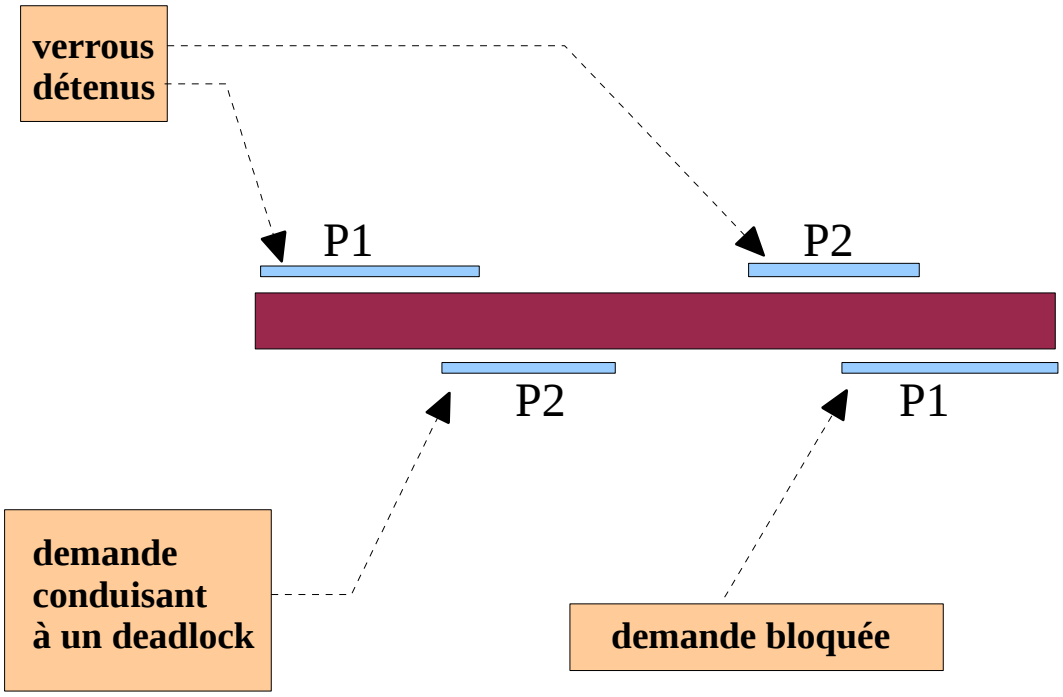
Dans le mode consultatif un verrou n'a pas d'influence sur le comportement des autres primitives d'entrées-sorties (open/read/write).

En mode **impératif**, un verrou peut empêcher la réalisation d'une opération de lecture ou d'écriture :

- **un verrou partagé** utilisé dans ce mode, tout autre processus faisant une écriture dans la portée de ce verrou est bloqué ;
- **un verrou exclusif** utilisé dans ce mode, tout autre processus faisant une lecture ou une écriture dans la portée de ce verrou est bloqué.

Il existe des situations de blocage (deadlock) comme dans le cas suivant mettant en compétition deux processus

P1 et P2 pour la pose de verrous exclusifs sur deux fichiers F1 et F2. Après que P1 ait posé avec succès un verrou exclusif sur F1 et P2 un verrou exclusif sur F2, le processus P1 est bloqué par une demande de pose de verrou sur F2. La demande de pose de verrou par le processus P2 sur le fichier F1 conduit à une situation typique de blocage : le processus P1 est bloqué par le processus P2 lui-même bloqué par P1.



Pour poser un verrou sur un fichier, les processus disposent des primitives suivantes :

- **flock()** : acquisition/libération de verrous sur un fichier complet ;
- **fcntl()** : acquisition/libération de verrous sur une section d'un fichier ;
- **lockf()** : idem "*fcntl*", mais avec une interface simplifiée.

L'interface de la fonction "**flock**" est la suivante :

```
# include <file.h>
int flock ( int fd, int operation );
```

Les valeurs du paramètre "**operation**" permettent de verrouiller l'accès au fichier défini par le descripteur "**fd**".

On trouve :

- **LOCK_SH** (pose de verrous partagés),
- **LOCK_EX** (pose de verrous exclusifs) et
- **LOCK_UN** (relâche des verrous).

La fonction "**lockf**" permet de poser/libérer un verrou sur une zone donnée d'un fichier ; son interface est la suivante :

```
# include <unistd.h>
# include <fcntl.h>
int lockf ( int fd, int cmd, off_t len );
```

Les valeurs du paramètre "**cmd**" permettent de verrouiller l'accès au fichier défini par le descripteur "**fd**".

On trouve :

- **F_ULOCK** (déverrouille la section verrouillée),
- **F_TLOCK** (verrouille une section),
- **F_LOCK** (verrouille une section avec suspension éventuelle du processus appelant) et
- **F_TEST** (teste la présence d'un verrou).

Le paramètre "**len**" représente la taille du verrou par rapport à la position courante.

L'interface de la fonction "**fcntl**" est la suivante :

```
# include <unistd.h>
# include <fcntl.h>
int fcntl ( int fd, int cmd );
int fcntl ( int fd, int cmd, long arg );
```

Le paramètre "**cmd**" permet de lancer une commande sur le fichier défini par le descripteur "**fd**". Parmi ces commandes on trouve celles qui sont propres aux verrous : **F_SETLK** et **F_SETLKW** (pose de verrous bloquant ou non-bloquant).

Par défaut, le mode de traitement utilisé de tous les verrous sur un fichier donné, est le mode **consultatif**. La demande du traitement en mode **impératif** se fait en positionnant le **set-gid-bit** du fichier :

```
$ chmod g+s f
$ ls -l f
-rw-r-Sr--      1     jmr   ens   27 Oct 13:48 f
```

Pour poser un verrou, il suffit de définir le verrou avec la structure "**flock**" et utiliser la fonction "**fcntl**" :

struct ***flock {***

short ***l_type;*** */* valeurs possibles :*

F_RDLCK, F_WRLCK, F_UNLCK */

short ***l_whence;*** */* valeurs possibles :*

SEEK_SET, SEEK_CUR, SEEK_END */

off_t ***l_start;*** */* position relative de debut*

par rapport a ***l_whence */***

off_t ***l_len;*** */* longueur : si valeur nulle*

*=> ⇔ jusqu'à la fin du fichier */*

pid_t ***l_pid;*** */** ***pid*** *du processus auquel*

*appartient le verrou */*

};

F_RDLCK : type partagé

F_WRLCK : type exclusif

F_UNLCK : relâchement de la zone réservée

SEEK_SET : début du fichier

SEEK_CUR : position courante de l'offset

SEEK_END : fin du fichier

L'intervalle verrouillé commence à "**l_whence + l_start**".

La manipulation :

```
struct flock verrou;
int rep,desc,commande;
...
rep = fcntl (desc, commande, &verrou);
...
```

où **fcntl** est une fonction qui permet en fonction du paramètre "**commande**" la réalisation d'un certain nombre d'opérations à différents niveaux dans les tables du système.

En utilisant la commande **F_SETLK** ou **F_SETLKW** **fcntl** pose un verrou respectivement non-bloquant et bloquant sur le fichier pointé par le descripteur **desc**.

Exemple :

```
$ cat verrou.c
#include <unistd.h>
#include <fcntl.h>
main() {
        int desc;
        struct flock verrou;
        if((desc=open("f",O_RDWR))==-1){
                perror("open");
                exit(0);}
        /* verrou exclusif sur le 15eme caractere de "f" */
        verrou.l_type=F_WRLCK; // exclusif
        verrou.l_whence=SEEK_SET; // début du fichier
        verrou.l_len=1;
        verrou.l_start=15;
        /* pose bloquante */
        if(fcntl(desc, F_SETLKW, &verrou)==-1){
                perror("pose verrou");
                exit(0);}
        printf("Verrou pose par %d\n",getpid());
        sleep(30);
        printf("Fin du processus %d\n", getpid());
}
$
```

```
$ cat lire.c
#include <fcntl.h>
main() {
   int desc, nb_lu;
   static char ch[10];
   desc = open ("f", 0_RDWR);
   nb_lu = read (desc, ch, 9);
   printf ("read1: %d caracteres lus → %s\n", nb_lu, ch);
   nb_lu = read (desc, ch, 9);
   printf ("read2: %d caracteres lus → %s\n", nb_lu, ch);
}
$ ls -l f
-rw-r-r--   1      jmr   ens   27 Oct 13:43    f
$ cat f
0123456789abcdefghijklmnop
$
```

```
$ verrou&  verrou&
[1]   879
[2]   880
$ Verrou pose par 879
cat f
0123456789abcdefghijklmnop
$ Fin du processus 879
Verrou pose par 880
Fin du processus 880
$
```

le processus 880 est bloqué sur la pose de son verrou par celui posé par le 879

en mode consultatif les lectures restent possibles

ceci libère le processus 880

Passage au mode impératif sur le fichier "f" :

```
$ chmod  g+s  f
$ ls -l f
-rw-r-Sr--        1      jmr   ens   27 Oct 13:48 f
$ verrou&  lire
Verrou pose par 885
[1]   885
read1: 9 caracteres lus - -> 012345678
Fin du processus 885
Read2: 9 caracteres lus - -> 9abcdefgh
$
```

libération du 15ème caractère