ANNEXE 3 :

la famille

UNIX,

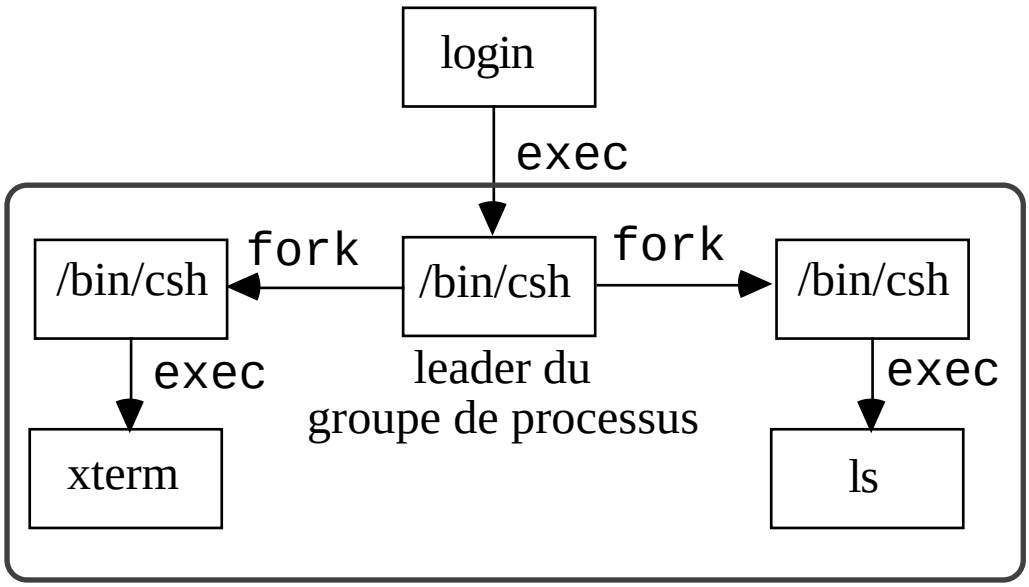
LINUX,

POSIX

présentation des processus

voir chapitre 1

Groupe de processus



groupe de processus

Le leader du groupe de processus a son pid égal à son pgid. Un processus fils crée un nouveau groupe en se retirant du groupe auquel il appartient par la primitive `setpgrp()`.

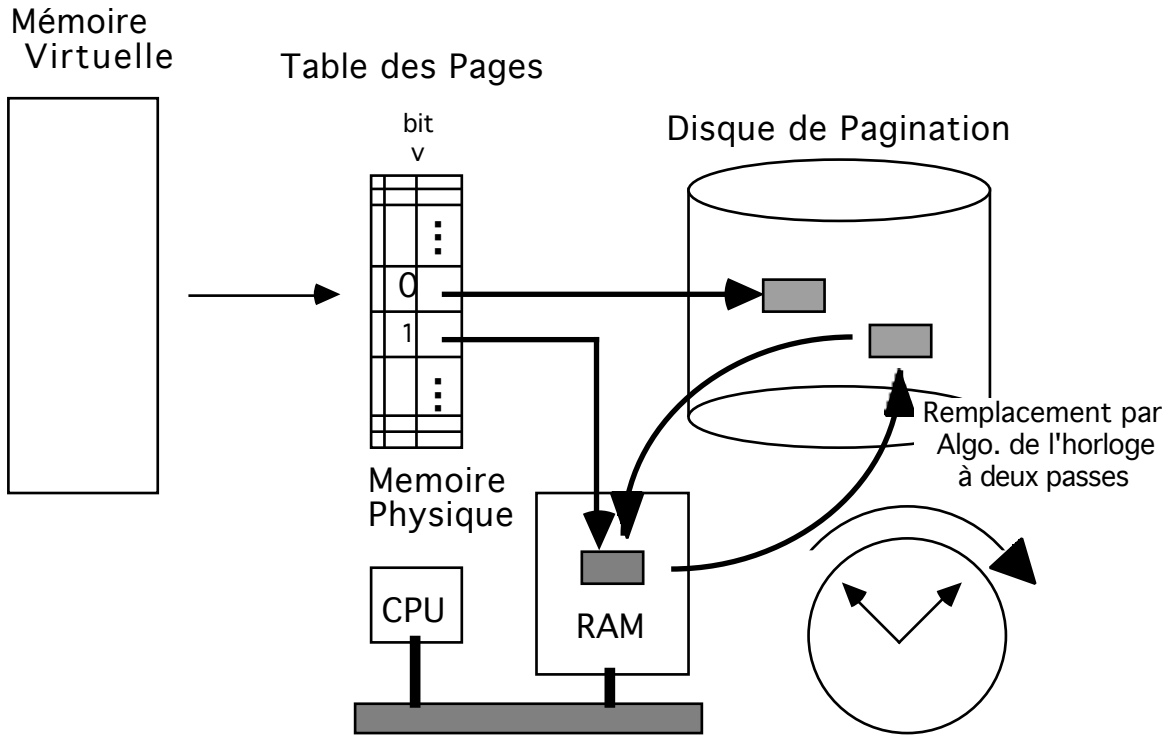
En Bourne shell, quand vous lancez une commande, un fils est créé, il appartient au même groupe de processus que son père. Dès que vous quittez votre session, le processus qui se termine a :
id de processus = id de groupe de processus = id de terminal attaché au groupe

Le signal SIGHUP est envoyé à tous les processus qui ont le même id de groupe de processus. Tous les processus fils, petits fils, ... dans le groupe sont tués automatiquement.
En Cshell, chaque commande lancée en tache de fond (avec & au bout) crée son propre groupe, et devient donc son propre "group leader". Par conséquent, la fin de la session utilisateur ne peut tuer les commandes lancées en tâches de fond.

Gestion Mémoire

relation

mémoire centrale et disque de pagination



-> mécanisme de “va et vient”, superposition de l’espace “swap” avec l’espace de pagination

commande vmstat

Statistiques sur la mémoire virtuelle, les processus, les disques, l'activité du processeur...

option -f pour avoir le nombre de fork et de vfork, option -v pour avoir des infos plus détaillées sur la mémoire, -S pour avoir des infos sur les pages soumises au va et vient (swap)

Sans options, la commande vmstat donne un résumé :

```
procs     faults     cpu      memory             page
disk
r b w in sy  cs us sy id avm fre re at pi po fr de sr s0 s1
0 0 0 98 571 75 16 12 72 21k 41k 2  4  4  1  1  0  0  0  0
```

procs: information sur les processus dans différents états.

- **r (run queue) processus prêts**
- **b bloqués en attente de ressources (E/S, demandes de pages)**
- **w prêts ou endormis (< 20 seconds)mais swappés**

faults: taux d'interruptions/d'appels système (trap) par seconde, la moyenne est calculée sur les 5 dernières secondes.

- **in interruption due à un contrôleur (sauf horloge) par seconde**
- **sy appels systemes par seconde**
- **cs taux de changement de contexte processuer (nb chgt par seconde)**

cpu: répartition de l'utilisation du processeur en pourcentage

- **us temps utilisateur pour les processus de priorité normale et basse**
- **sy proportion utilisée par le système**
- **id cpu inutilisé - libre**

La somme des 3 fait 100% !?!

memory: informations sur l'utilisation de la mémoire virtuelle et de la mémoire réelle, les pages virtuelles sont considérées actives si elles appartiennent à des processus qui sont exécutables ou qui se sont exécutés depuis moins de 20 secondes. Les pages sont indiquées en unité de 1 Ko, le suffixe k précise qu'il faut multiplié le chiffre affiché par 1000, et le suffixe m par 10^6.

- **avm pages virtuelles actives**
- **fre taille de la liste des pages libres**

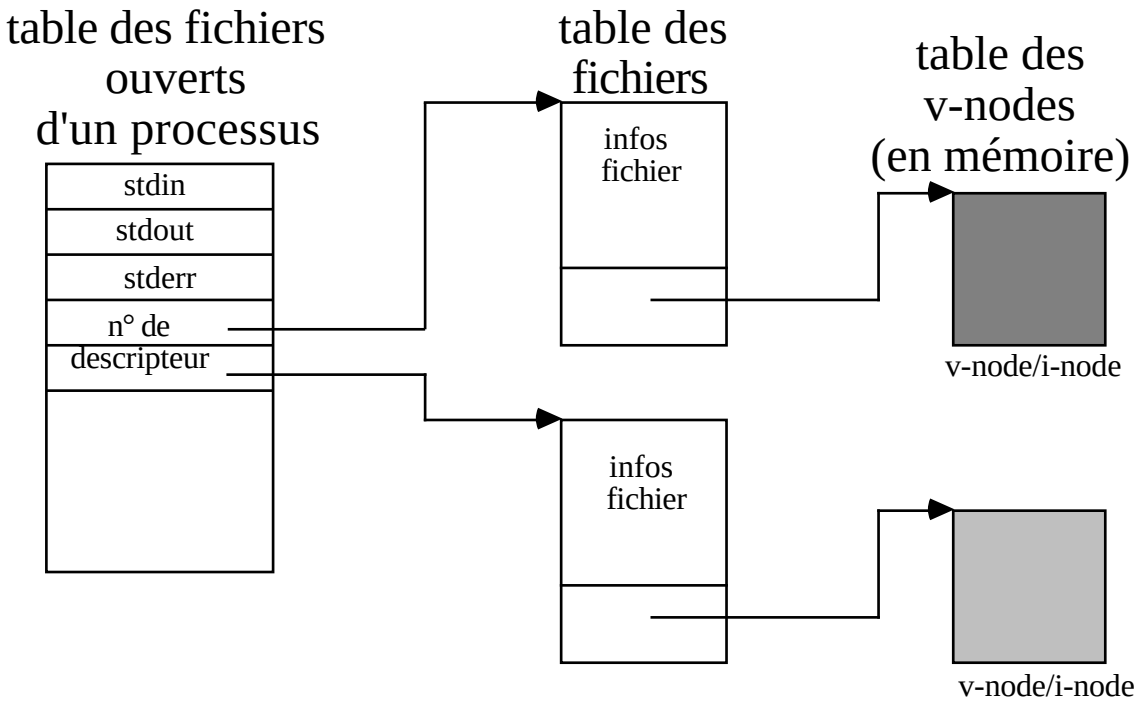
page: information les défauts de page et l'activité de pagination. La moyenne est calculée toutes les 5 secondes. L'unité est toujours 1Ko et est indépendante de la taille réeel des pages de la machine.

- **re page "reclaims", pages référencées**
- **at pages "attached" ???**
- **pi pages "paged in"**
- **po pages "paged out"**
- **fr pages "freed", libérées par seconde**
- **de anticipation du manque de mémoire**
- **sr pages examinées par l'algorithme horloge à deux phases**

disk: s0, s1 ...sn: activité de pagination ou de va et vient en secteurs transférés par seconde, ce champ dépend de la configuration du système. Habituellement l'espace swap est réparti sur plusieurs disques, on trouve la liste des périphériques qui supportent cette fonction et qui est configurée dans le système.

Fichiers et Tubes

gestion des E/S : disque, réseau, terminal



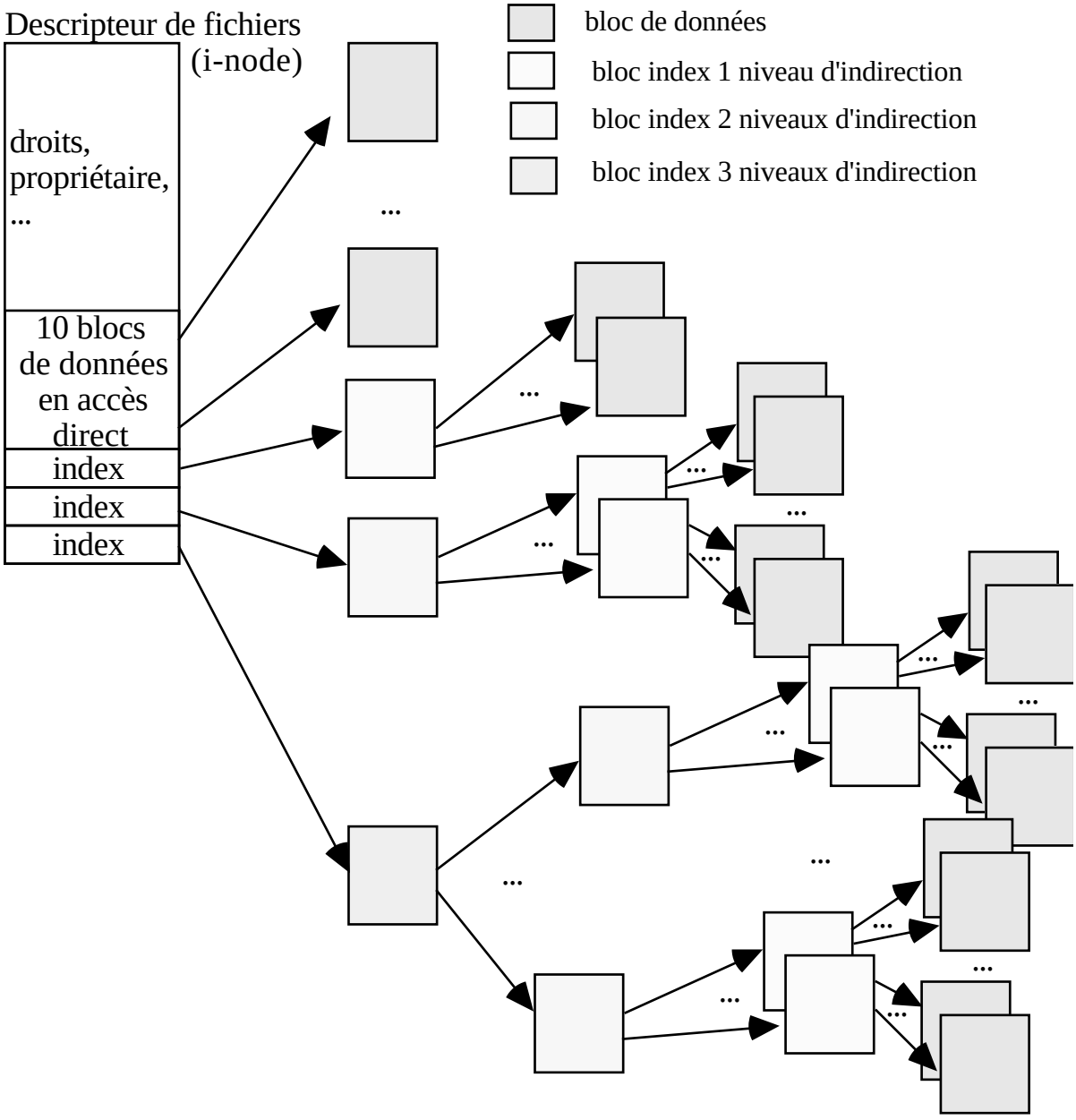
notion d'objet décrit par un descripteur et un ensemble d'opérations plutôt que par extension de l'espace d'adressage (Multics)

Structure d'un i-node

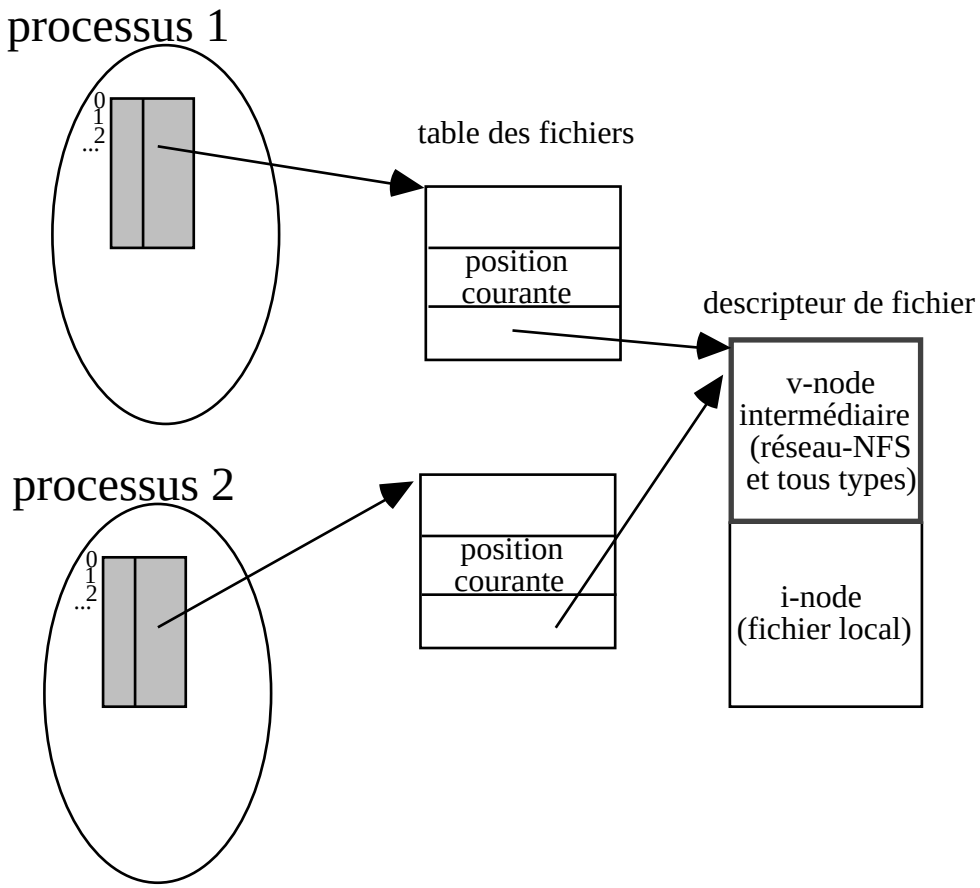
Vue utilisateur : Fichier non structuré

Structure d'un i-node

Vue Système

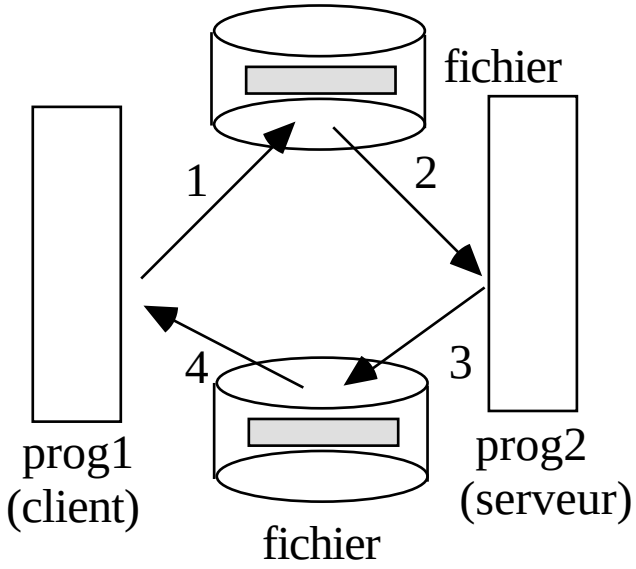


Partage de fichier



Communication par fichier

Les programmes peuvent communiquer par fichier :



Il peut être nécessaire de procéder à un verrouillage de la ressource fichier :
`flock(), lockf(), fcntl()`

ATTENTION :
ces mécanismes ne fonctionnent pas nécessairement à travers NFS !!!

Verrouillage de fichier (1)

verrouillage de tout le fichier (Unix souche BSD) :

`flock()`

Le type de verrouillage doit être précisé :
`LOCK_SH` : verrouillage en mode partagé (opération bloquante)
`LOCK_EX` : verrouillage en mode exclusif (opération bloquante)
`LOCK_UN` : déverrouillage
`LOCK_NB` : demande d'opération non bloquante

Une opération est dite **bloquante** si le **demandeur est bloqué jusqu'à ce que l'appel système correspondant soit terminé** ... ce qui peut arrivé ... l'appellant restant dans ce cas indéfiniment en attente... d'où l'utilisation par exemple de la combinaison `LOCK_SH|LOCK_NB`.

Un fichier peut être verrouillé "`LOCK_SH`" par plusieurs processus en même temps, mais ne peut être verrouilé "`LOCK_EX`" que par un seul à la fois.

Verrouiller une partie du fichier (Unix souche system V) :

`lockf()`

dans ce cas, il faut utiliser `lseek()` pour se positionner au début de la zone du fichier ciblée, puis spécifier dans l'appel de `lockf()` la longueur de la zone à verrouiller.

Le verrouillage peut s'effectuer de différentes façons, qu'il faut spécifier :

`F_TEST` : test si une zone est déjà verrouillée
`F_LOCK` : vérrouiller une zone (opération bloquante)
`F_TLOCK` : test si la zone est verrouillée, verrouille sinon
`F_ULOCK` : déverrouille une zone déjà verrouillée

Primitive `mmap()` (1)

`mmap()`

Permet à un processus de projeter le contenu d'un fichier déjà ouvert dans son espace d'adressage. La zone mappée s'appelle une région. Au lieu de faire des lectures et des écritures sur le fichier, le processus y accède comme si les variables qu'il contient étaient en mémoire.

Plusieurs processus peuvent "mapper" le contenu d'un même fichier et ainsi le partager de façon efficace.

Un fichier peut être soit un fichier sur disque, soit un périphérique.

`mmap()` s'utilise pour un fichier déjà créé, il est impossible d'étendre un fichier "mappé"

suivant les implantations, il semble qu'un `munmap()` soit nécessaire pour "démapper" une région avant de fermer le fichier par `close()`.

pas de client/serveur !:-)

Primitive mmap() (2)

```
#include <sys/types.h>
#include <sys/mman.h>

caddr_t mmap(addr,len,prot,flags,fd,off)
caddr_t  addr;
size_t   len;
int      prot, flags, fd;
off_t       off;
```

C'est une mise en correspondance à partir de l'adresse "pa" de l'espace d'adressage d'un processus, d'une zone de "len" octets prise à l'intérieur d'un objet de descripteur "fd" à partir de "off". La valeur de "pa" dépend du paramètre "addr", de la machine, et de la valeur de "flags".

Si l'appel système réussit, mmap() retourne la valeur pa en résultat. Il y a correspondance entre les régions [pa, pa+len] de l'espace d'adressage du processus et [off, off+len] de l'objet. Une mise en correspondance remplace toute mise en correspondance précédente sur la zone [pa, pa+len] .

prot détermine le mode d'accès de la région : "read", "write", "execute", "none" (aucun accès) ou une combinaison de ces modes

flags donne des informations sur la gestion des pages mappées :

MAP_SHARED	page partagée, les modifications seront visibles par les autres processus quand ils accèderont au fichier
MAP_PRIVATE	les modifications ne seront pas visibles
MAP_FIXED	l'adresse addr donnée par l'utilisateur est prise exactement, habituellement le système effectue un arrondi sur une frontière de page

Exemple mmap()

```
int fd, *p, lg;

fd = open("fichier",2);

p =
  mmap((caddr_t) 0,lg,
                PROT_READ|PROT_WRITE,

  MAP_SHARED,fd,0);

*p = *p +1;

close(fd);
```

Tube ou "pipe" - "|" du niveau shell

`pipe()`

Un tube est un **canal unidirectionnel** qui fonctionne en mode flot d'octets. Mécanisme d'échange bien adapté à l'envoi de caractères.

La suite d'octets postée dans un tube n'est pas obligatoirement retirée en une seule fois par l'entité à l'autre bout du tube. Lors de la lecture, il peut n'y avoir qu'une partie des octets retirés, le reste est laissé pour une lecture ultérieure. Ceci amène les utilisateurs à délimiter les messages envoyés en les entrecoupant de "\n".

Sous certaines conditions, une écriture ou une lecture peut provoquer la terminaison du processus qui effectue l'appel système.

Les écritures dans un tube sont atomiques normalement. Deux processus qui écrivent en même temps ne peuvent mélanger leurs données. Mais si on écrit plus d'un certain volume max (4096 octets souvent), l'écriture dans le tube n'est pas atomique, dans ce cas les données écrites par deux processus concurrents peuvent s'entrelacer.

L'utilisation d'un tube se fait entre processus de même "famille"/"descendance". On peut dire que le tube est privé.

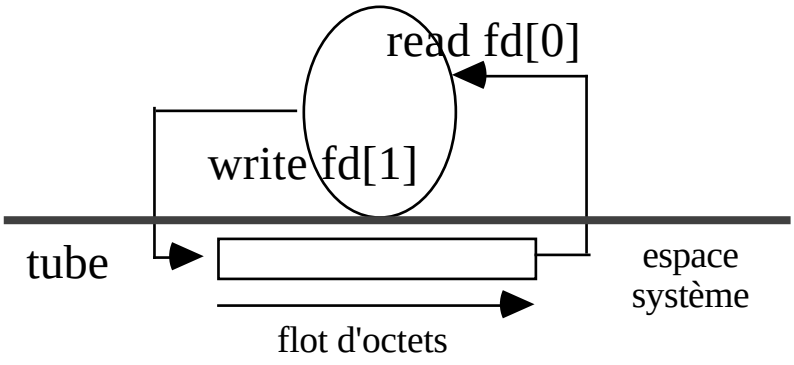
L'usage courant est :

```
int pipefd[2];
pipe(pipefd);
write (pipefd[1], ...);
read (pipefd[0], ...);
```

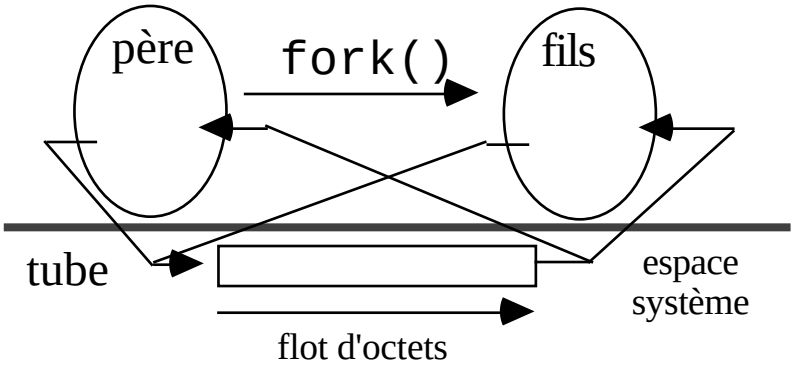
Se souvenir de stdin (0), on lit ce qui vient du clavier, et, stdout(1), on écrit sur l'écran.
Un tube existe tant qu'un processus le référence. Quand le dernier processus référençant un tube est terminé, le tube est détruit.

Etapes d'un échange client/serveur avec tube

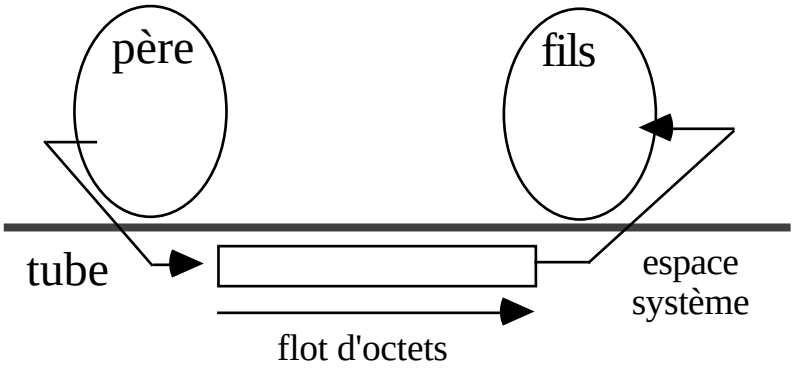
1. création du tube :



2. `fork()` du père :



3. fermeture des extrêmités non utilisées

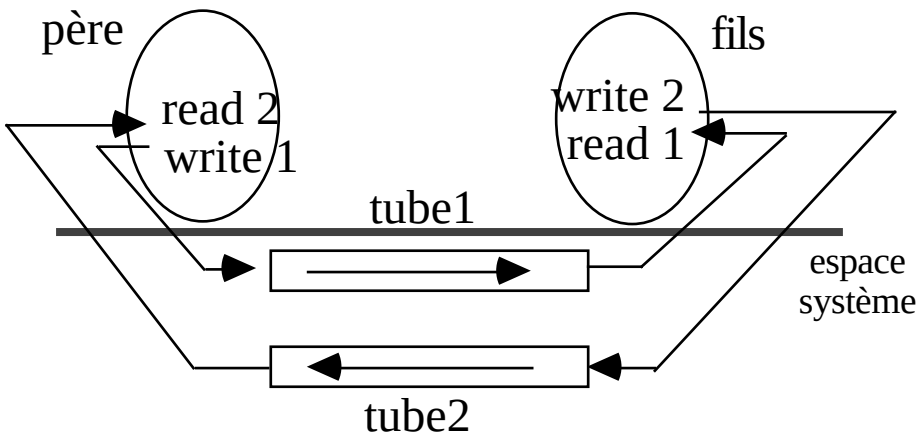


situation équivalente à celle de `"ls -l|more"`

Schéma Client/Serveur avec tubes

Etapes :

1. création de tube1 et de tube2
2. `fork()`
3. le père ferme l'entrée 0 (in) de tube1 et l'entrée 1 (out) de tube2
4. le fils ferme l'entrée 1 (out) de tube1 et l'entrée 0 (in) de tube2



Tubes Nommés ou FIFOs

Les tubes sont privés et connus de leur seule descendance par le mécanisme d'héritage. Les tubes nommés sont publics au contraire.

Le tube nommé est créé par :

```
mknod("chemindaccès", mode d'accès)
```

ou par

```
/etc/mknod chemindaccès p
```

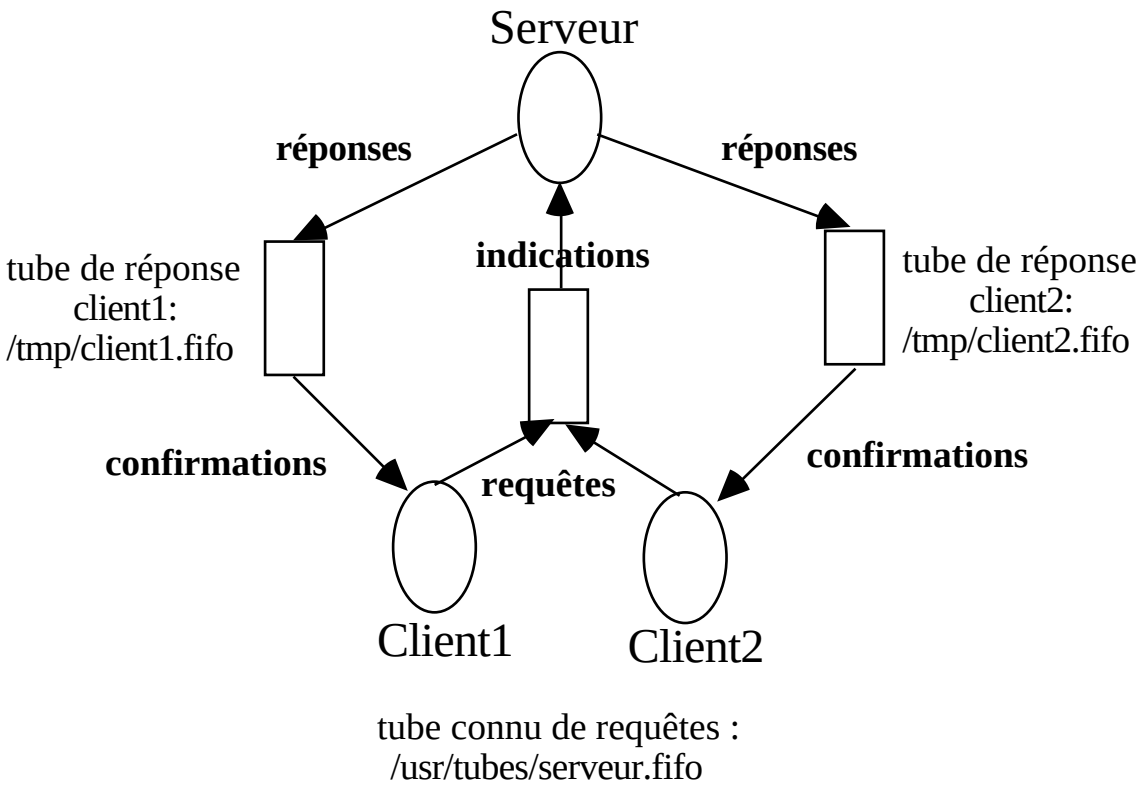
C'est presque l'équivalent d'un fichier pour le système.

Le tube est connu par son chemin d'accès et accessible comme un fichier par

`open()`, `read()`, `write()`, `close()`.

Les tubes nommés fonctionnent comme les tubes.

Client/Serveur avec des tubes nommés



Ne pas oublier de détruire les tubes nommés après utilisation !!!

Mécanismes de communication inter-processus

IPC distants :

- sockets (Unix BSD) ou streams (Unix System V)

IPC locaux :

- fichiers avec mécanisme de verrouillage (lock)
- tubes (pipe), tubes nommés,
- interruptions logicielles (signaux)
- sockets,
- sémaphores,
- files de messages,
- mémoire partagée,

IPC System V

Identification des objets IPC system V

Les objets IPC system V :

Files de messages
Sémaphores,
Segments de mémoire partagée

Ils sont désignés par des identificateurs uniques. La première étape consiste donc a créer un identificateur.

1. On peut fabriquer un identificateur grace à la fonction suivante :

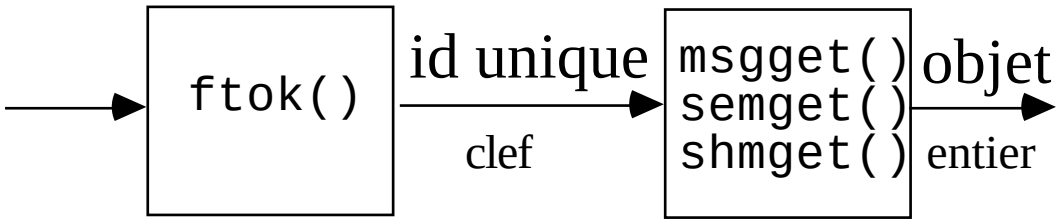
```
#include <sys/types.h>
#include <sys/ipc.h>

key_t ftok(char *chemindaccès, char proj)
```

L'identificateur rendu en résultat est garanti unique. "chemindaccès" correspond à un fichier, attention à ce qu'il ne soit pas détruit de façon inopportune... sinon, la fonction `ftok()` ne pourrait retourner l'identificateur qui sert à tous les processus pour repérer l'objet IPC.

2. On se le fixe soi même

Création d'objets IPC System V

On attaché à un objet IPC un certain nombre d'informations : l'id utilisateur du propriétaire, l'id du groupe du propriétaire, l'id utilisateur du créateur, l'id du groupe du créateur, le mode d'accès, son identificateur.



Mode de création :

IPC_PRIVATE	création de l'objet demandé et association de celui-ci à l'identificateur fourni
IPC_CREAT	création si l'objet n'existe pas déjà, sinon aucune erreur
IPC_CREAT|IPC_EXCL	création si l'objet n'existe pas déjà, sinon erreur
aucun mode précisé	erreur si l'objet n'existe pas

Pas comparable à un fichier dans son mode de gestion et d'héritage. L'identificateur d'objet se passe

Files de messages

L'objet "Message Queue" fonctionne suivant le modèle d'une file donc en FIFO, la politique FIFO peut être appliquée à l'ensemble des messages dans la file ou par seulement en fonction du type de message.

création : `msgget()`

Lors de la création, on spécifie les droits d'accès : read/write pour propriétaire/groupe/autres en combinant avec IPC_XXX.

envoi de message : `msgsnd()`
réception de message : `msgrcv()`

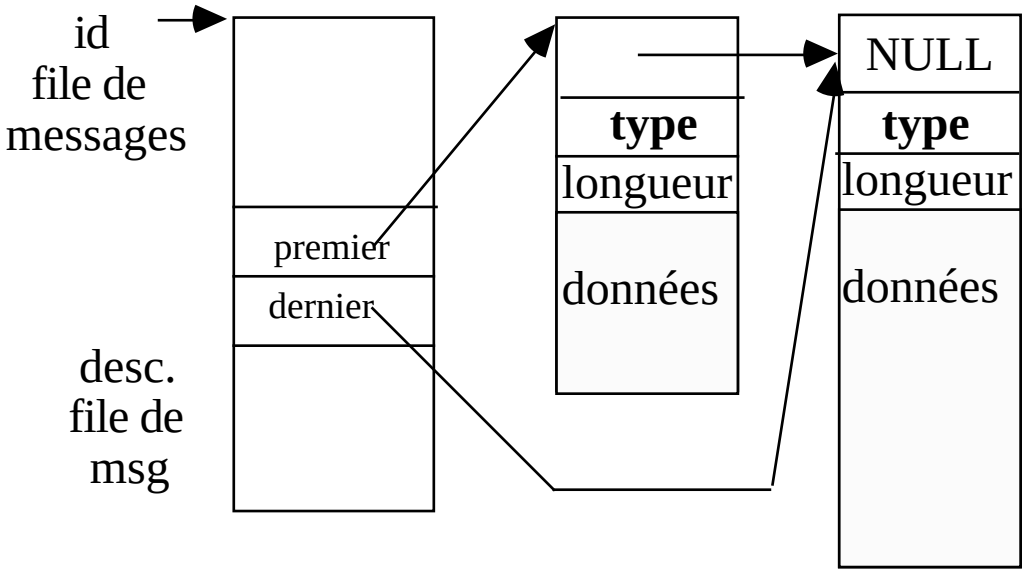
Les messages peuvent être retirés en fonction de leur type.

opérations de contrôle : `msgctl()`

Permet en particulier de détruire une file de messages, sinon il faut utiliser la commande `ipcrm`.

La commande `ipcrm` est applicable à tout objet IPC system V.

Objet File de Messages



Sémaphores

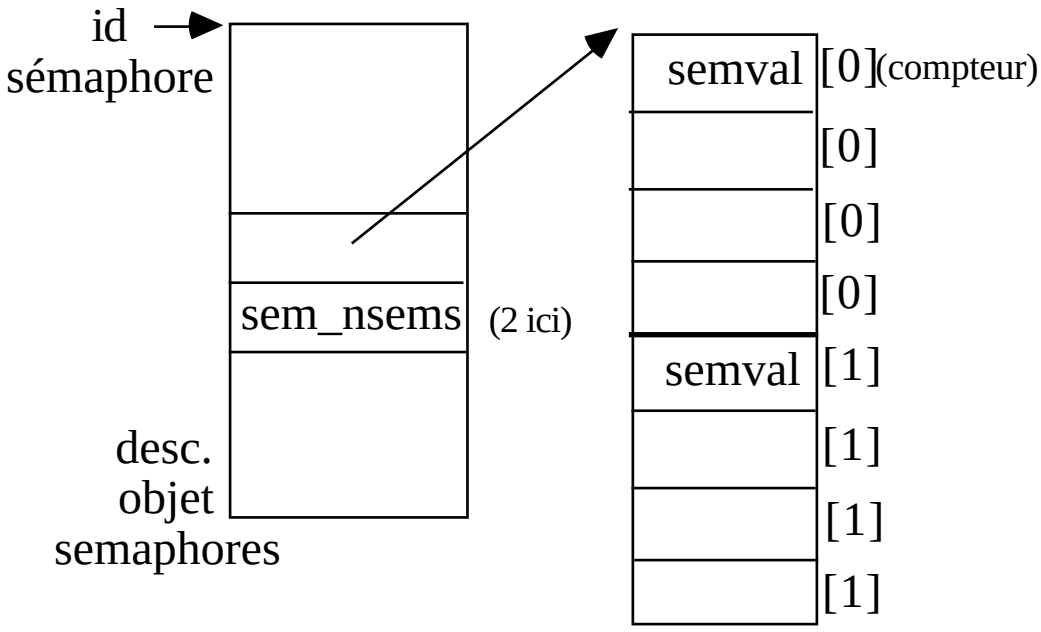
Les objets sémaphores permettent de gérer un groupe de sémaphores par objet.

création : **`semget()`**

manipulation du sémaphore : **`semop()`**

gestion du sémaphore : **`semctl()`**

Objet sémaphore :



Quand plusieurs processus attendent le relachement d'une ressource, on a aucun moyen de déterminer à l'avance celui qui l'obtiendra. En particulier, pas d'ordre FIFO d'attente.

Segments de Mémoire partagée

Communication par variable partagée à travers l'espace d'adressage des processus.

création : `shmget()`

Le segment de mémoire partagée est créé mais non accessible. Là encore, il faut spécifier le mode d'accès.

attachement à l'espace d'adressage d'un processus : `shmat()`
Le segment est inclu dans l'espace d'adressage du processus demandeur à l'adresse indiquée suivant le cas par *shsmaddr.

détachement du segment de mémoire partagée de l'espace d'adressage d'un processus : `shmdt()`

gestion du segment de mémoire partagée : `shmctl()`

Utilisation de segments de mémoire partagée

En général on fait du client/serveur (producteur/consommateur) à travers un segment de mémoire partagée avec une signalisation par sémaphore.