TUBES NOMMES

• Premier moyen d'échange de données entre processus sous UNIX

• Fichiers spéciaux gérés selon une méthodologie FIFO (First In First Out), c'est-à-dire que l'ordre des caractères en entrée est conservé en sortie (premier entré, premier sortie).

• Taille limitée finie.
• Deux types de tubes :
  - Tubes ordinaires, non visibles dans l'arborescence, entre processus d'une même filiation *fork()*
  - Tubes nommés, visibles dans l'arborescence, entre processus non forcément liés par filiation

• N'importe quel processus (avec les droits adéquats) peut ouvrir des tubes nommés pour communiquer

• Tube unidirectionnel : pour pouvoir communiquer dans les 2 sens avec 2 processus, il faut 2 tubes.

• L'opération de lecture y est destructive !

CREATION DU TUBE NOMME

• Création par la primitive *mknod* qui permet de créer des fichiers spéciaux
• En POSIX, création par la primitive simplifiée *mkfifo*

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *filename, mode_t mode);
...
// Exemple:
mkfifo(”myfifo”, 0666); // octal 0666 = permissions lecture + écriture
// ou equivalent
mknod(”myfifo”,0666|S_IFIFO,dev_t(0));
```

Crée un tube nommé de nom *filename* et de permissions *mode*
• Retourne 0 en cas de succès, -1 en cas d'échec
• Une fois crée, n'importe quel processus en ayant les droits peut ouvrir le tube nommé comme il le ferai avec un fichier ordinaire pour lecture et écriture.

OUVERTURE DU TUBE NOMME

• Ouvertures possibles par les processus connaissant le nom du tube

• Avant de lire ou d'écrire, chaque extrémité du tube doit être ouverte.
– *flags* spécifie les droits d'ouverture (lecture, écriture, ou les 2)
– par défaut, l'ouverture du tube en lecture est bloquante tant qu'il n'y a pas ouverture en écriture et vice versa (le producteur et le consommateur sont alors synchronisés)
– Avec le flag O_NONBLOCK (mode non bloquant), *open* se comporte de façon asymétrique

• En lecture seule, *open* retourne le descripteur immédiatement sans attendre l'ouverture en écriture par le producteur

• En écriture seule, *open* retourne le descripteur immédiatement ou échoue (errno = ENXIO) si il n'est pas ouvert en lecture en mode non bloquant par le producteur, ou si le lecteur est bloqué sur l'ouverture de celui-ci en mode bloquant.

Pourquoi ce comportement ?
Ceci pour éviter que le processus qui vient d'ouvrir le tube nommé, n'écrive dans le tube avant qu'il n'y ait de lecteur (qu'un processus ait ouvert le tube en lecture) et ce qui engendrerait un signal SIGPIPE (tube détruit), ce qui n'est pas vrai car le tube n'a pas encore été utilisé.

• Toutes les opérations de lecture écriture qui suivent sont alors non bloquantes

```
#include <sys/types.h>
#include <sys/stat.h>
int open(const char *filename, int flags);

// Exemples:
open("fifo", O_RDONLY); // ouverture en lecture seule
open("fifo", O_WRONLY); // ouverture en écriture seule
open("fifo", O_RDONLY|O_NONBLOCK); // Réussit immédiatement (tous les cas)
open("fifo", O_WRONLY|O_NONBLOCK); // Réussit ou échoue
```

OPERATIONS DE LECTURE-ECRITURE DEPUIS/VERS LES TUBES NOMMES

• Peuvent être lu ou écris comme des fichiers ordinaires
– en utilisant les primitives systèmes (bas niveau) : read, write
– en utilisant les fonctions de la librairie (haut niveau)

```
FILE *fout= fdopen( tube ,"w"); fprintf(fout, "Coucou le monde\n");
```

• Pour les tubes ouverts en mode bloquant (mode par défaut):
– la lecture depuis un tube vide est bloquante (attente de données)
– l'écriture dans un tube plein est bloquante (attente d'une possibilité d'écriture)

• Taille du buffer du tube : PIPE_BUF peut être définit dans <*limits.h*>
– PIPE_BUF = 4096 (LINUX)
– PIPE_BUF = 512 minimum si le système est conforme POSIX
– Cette taille est la taille maximum qui garantie l'atomicité de l'écriture

• Lecture `nb_lu = read(fd, buffer, TAILLE_READ);`

Si le tube n'est pas vide et contient *taille* caractères :
Lecture de *nb_lu* = min (*taille*, *TAILLE READ*) caractères.
Si le tube est vide
Si le nombre d'écrivains est nul
Alors c'est la fin de fichier et *nb_lu* est nul.
Si le nombre d'écrivains est non nul
Si lecture bloquante alors sommeil
Si lecture non bloquante alors en fonction de l'indicateur
O_NONBLOCK : nb lu= -1 et errno=EAGAIN.
O_NDELAY : nb lu = 0.

• Ecriture `nb_ecrit = write(fd, buf, n);`

L'écriture est atomique si le nombre de caractères à écrire est inférieur à PIPE_BUF, la taille du tube sur le système. (cf <*limits.h*>).

Si le nombre de lecteurs est nul
Envoi du signal *SIGPIPE* à l'écrivain et errno = EPIPE.
Sinon
Si l'écriture est bloquante, il n'y a retour que quand
Les n caractères ont été écrits dans le tube.
Si écriture non bloquante
Si n > PIPE BUF, retour avec un nombre inférieur a n éventuellement -1 !
Si n = PIPE BUF
Et si n emplacements libres, écriture *nb_ecrit* = n
Sinon retour -1 ou 0.

• Pour les tubes ouverts en mode non bloquant (O_NONBLOCK)
– la lecture depuis un tube vide réussit et retourne 0 (octets)
– l'écriture vers un tube plein (ou presque plein) peut échouer (écriture de moins de PIPE_BUF octets) ou écrire seulement une partie des données (écriture de plus de PIPE_BUF octets)

La limitation de la taille atomique d'écriture n'est pas si importante dans le cas d'un tube avec un seul producteur et un seul consommateur. Dans le cas de plusieurs processus qui écrivent dans le tube au même moment, il est important que les blocs de données des différents processus n'interfèrent pas entre eux. Si vous pouvez assurer que toutes vos écritures sont faites dans un tube ouvert en mode bloquant, et dont la taille en octets est inférieure ou égale à PIPE_BUF, alors l'atomicité des écritures dans ce tube sera garantie.

Attention !

L'appel à *read* retourne toujours le maximum de données possibles, même si les appels à *write* sont issus de processus différents. Il est donc nécessaire que les paquets envoyés soit toujours de la même taille.

Note

Quand on écrit ou lit dans/depuis un tube nommé, `read()` retournera EOF quand tous les producteurs seront fermés, et `write()` enverra le signal `SIGPIPE` quand il n'y aura plus de consommateur. Si `SIGPIPE` est bloqué ou ignoré, on peut tester errno = `EPIPE`.

SUPRESSION DU TUBE

• Suppression du tube lorsque explicitement demandé et pas d'ouverture en cours
• Si suppression alors qu'ils existent des lecteurs et écrivain, fonctionnement comme un fichier ordinaire

UTILISATION DE TUBES NOMMES DANS DES APPLICATIONS

Comment implémenter une communication bidirectionnelle entre un serveur et plusieurs clients ?

Il est possible que plus d'un client communique avec votre serveur en même temps. Tant que les commandes envoyées par les clients sont inférieures à PIPE_BUF (voir plus haut), ils peuvent tous utiliser le même tube nommé pour envoyer des données au serveur.

Par contre, le serveur ne peut pas utiliser un seul tube pour communiquer avec les clients. Si plus d'un client lit le même tube, il n'est pas possible de garantir que le bon client reçoive les données envoyées par le serveur.

Une solution est que chaque client crée son propre tube en mode lecture avant d'envoyer des données au serveur.

Utiliser le PID (Processus IDentificator) du client dans le nom du tube est une bonne façon de les identifier. Le client, à chaque fois qu'il communique avec le serveur, peut alors inclure son PID dans la commande envoyée. Toute donnée retournée par le serveur sera alors renvoyé par le tube approprié.