Université de Rouen
Licence informatique
Module : Système II

T.P. 6 : Tubes et Tubes Nommés

1 Les tubes

1.1 Généralités

Un tube de communication *pipe* est un i-noeud spécial géré par le SGF. On utilisera les mêmes appels systèmes pour accéder à ce fichier spécial (read, write, dup et close) que pour accéder à un fichier ordinaire. Par contre, les tubes n'ont pas de nom externe. Par conséquent, le partage du tube n'est possible que pour des processus partageant le nom local du tube (c.a.d le descripteur).
La création, ainsi que l'ouverture se font par la même primitvive *pipe()* qui fournit un tableau de 2 descripteurs (entiers) :

- fd[0] descripteur en lecture.
- fd[1] descripteur en écriture.

Le tube possède une gestion de type FIFO qui implique que tout caractère écrit dans le tube ne peut être lu qu'une et une seule fois (pas d'utilisation possible de l'appel système *lseek()*).

1.2 Synchronisation

Pour un tube donné, on appelle lecteur (resp écrivain) tout processus possédant un descripteur en lecture (resp écriture) sur le tube.
Principes de synchronisation :

- Lecture : si un processus demande la lecture de n caractères dans un tube en contenant $m > 0$, il y a lecture de *inf(n,m)* caractères (retour immédiat du *read()* du nombre de caractères réellement lus).
- Lecture dans un tube vide : lecture bloquée juqu'à ce qu'une écriture ait lieu ou qu'il n'y ait **plus d'écrivain**. Dans ce cas, le *read()* rend 0 pour indiquer la fin de tube.
- Ecriture : si un processus demande l'écriture de n caractères ($n <$ PIPE_BUF) dans un tube pour lequel il existe un lecteur, le système garantit que ces n caractères seront écrits de façon consécutive dans le tube.
- Ecriture dans un tube plein : suspension de l'écriture jusqu'à ce qu'une lecture ait lieu.
- Ecriture sans lecteur : le système envoie au processus le signal SIGPIPE.

2 Les tubes nommés

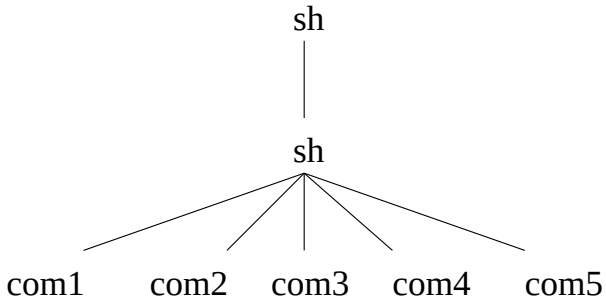
Un tube nommé est un tube qui possède un nom externe. Le fonctionnement est identique aux tubes à ces 2 différences près.

- Un tube nommé permet de faire communiquer des processus indépendants (accès par le nom externe).
- La création et l'ouverture se font de façon distincte et deux processus peuvent se synchroniser sur l'ouverture.
- **mknod("nom_FIFO", 0666 | S_IFIFO, 0)** : Création du tube nommé.
- **open("nom_FIFO", int mode, 0)** : Ouverture du tube nommé.

2.1 Synchronisation à l'ouverture

Lors d'une demande d'ouverture en lecture (resp écriture) d'un tube nommé sans écrivain (resp lecteur), le processus est endormi jusqu'à l'arrivée d'une ouverture en écriture (resp lecture).

3 Exercices



Le fonctionnement de l'enchainement de processus en c-shell est le suivant. Le shell courant lance un sous-shell. Ce sous-shell lance les commandes 1 à $n-1$ et se recouvre avec la n-ième. Pendant ce temps, le shell courant attend la fin de son fils. Chacune des commandes lancées devra rediriger son entrée standard ou sa sortie standard ou les deux en fonction de sa position dans la liste des processus. Dans l'exemple du schéma, quatre tubes devront être créé. *com*1 redirige sa sortie standard sur le tube 1, *com*2 redirige son entrée standard sur le tube 1 et sa sortie standard sur le 2, etc ....

Ecrire un programme *spipe* dont le but sera de simuler l'enchaînement de commandes cshell à travers des pipes :

```
Ex: spipe "ls -l" "wc" -> ls -l | wc
```

Le nombre d'arguments n'est pas limité