Université de Rouen	Année 2008-2009
Faculté des sciences	Système d'exploitation
Licence Informatique 3ème année	Fiche de TD n° 10

Les tubes

1 Présentation des tubes

1.1 Les tubes standards

1.1.1 Généralités

Un tube de communication (*pipe* en anglais) est un i-noeud spécial géré par le système de fichiers. 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 standrads 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 d'un tube standard se font par la même primitvive `pipe()` qui fournit un tableau de 2 descripteurs (entiers) :

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

Alors
- `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()`).

1.1.2 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.
- Écriture : 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.
- Écriture dans un tube plein : suspension de l'écriture jusqu'à ce qu'une lecture ait lieu.
- Écriture sans lecteur : le système envoie au processus le signal `SIGPIPE`.

1.2 Les tubes nommés

1.2.1 Présentation

Un tube nommé est un tube qui possède un nom externe. Le fonctionnement est identique aux tubes standards 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.
- Création du tube nommé :

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

- Ouverture du tube nommé (comme pour un fichier ordinaire) :

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

1.2.2 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).

2 Exercices

Exercice 1

1) Écrire un programme qui à l'aide de deux processus et d'un tube réalise la commande :

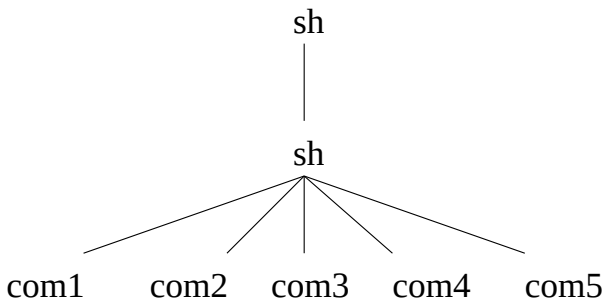
who | wc -l

2) Écrire un programme C qui affiche le i-ème mot de la j-ème ligne dans un fichier dont le nom est passé en argument au programme. Le programme aura le comportement du script shell suivant :

```
#!/bin/sh
# $1 : numéro de ligne
# $2 : numero de mot

if [ $# -ne 2 ]
then
    echo usage: $0 num_ligne num_mot
    exit 1
fi
head -n $1|tail -n 1|cut -f$2 -d\
```

Exercice 2 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, ...



1) 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

Exercice 3 On souhaite réaliser un serveur d'information du système. Un client émet une requête via un tube nommé pour demander :

1. le nom d'un utilisateur en fonction de son **uid**
2. le nom d'un groupe en fonction de son **gid**
3. les informations sur un processus en fonction de son **pid**

Le fonctionnement pourrait être le suivant : on utilise un tube nommé pour les questions et un autre pour les réponses. Le serveur est en attente passive. Lorsqu'une question arrive, le serveur se duplique. Le père reprend l'attente alors que le fils traite la question et envoie sa réponse dans le tube dédié.

Le fils pourra utiliser les primitives **getpwuid**, **getgrgid**, et les informations récupérées dans /**proc** (voir TD ordonnancement).

Lors de la terminaison du serveur, tous les tubes devront être supprimés. On captera les signaux **SIGINT** et **SIGQUIT**

1) Écrire les algorithmes pour le serveur puis pour le client.

2) Écrire les programmes.

Exercice 4 Soit T un tableau de $n = 2^k$ entiers. On souhaite calculer la valeur S donnée par la formule suivante :

$$S = \bigodot_{i=1}^{n} T(i) \tag{1}$$

où $\odot$ est une opération associative (par exemple : $+$, min, max, pgcd, $\times$, ...).

Le but de cet exercice est de répartir le calcul sur plusieurs processus. Chaque processus sera responsable de l'exécution d'un nombre constant d'opérations élémentaires.

En utilisant des tubes ordinaires pour la communication inter-processus et leurs synchronisations, proposer un programme qui calcule S.