TP 4 – Tubes

☞ Lire entièrement l'exo 1 avant de le commencer - les indices sont à la fin.

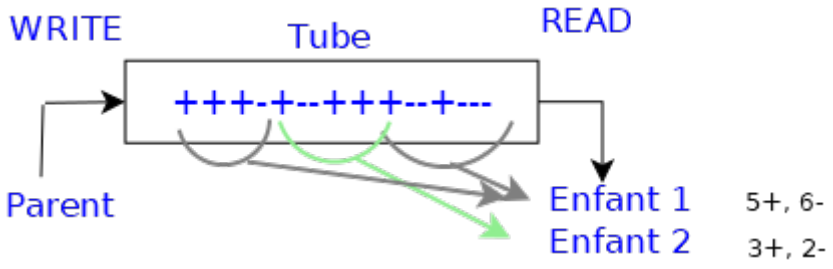
☞ L'important est de bien maitriser l'exercice 1 avant les TPs suivant.

1 Envoi par tubes

On souhaite réaliser une communication par tube (anonyme) entre un processus parent et deux processus enfants.

— Le processus parent envoie des + et des - dans le tube (de manière aléatoire).

— Les deux processus enfants lisent le tube (simultanément) et chacun comptabilise les jetons (+ et -) qu'il aura lus. [comme la lecture dans un tube est destructrice, les processus enfants ne lisent pas les mêmes jetons, mais à la fin, à eux deux les processus enfants auront donc lu tous les jetons.]



Mettez en place ce mécanisme. **On écrira 2 programmes**, l'un pour le parent et l'autre qui sera utilisé par les deux enfants (il y a aura donc deux `fork` dans le père et un `execl` dans les parties enfants appelant le deuxième programme.

Remarques :

- Le nombre (global) de jetons envoyés sera donné en argument au père.
- Une fois le tube et les enfants créés, le parent envoie aléatoirement des + et des - (voir encart).
- Un enfant lit caractère par caractère dans le tube (il lit un caractère puis met à jour ses compteurs et boucle).
- À la fin d'un processus enfant (quand il n'y a pas plus rien à lire dans le tube, ce dernier devra afficher le nombre de jetons + et le nombre de jetons - reçus (et son `pid`). Vous pourrez alors vérifier si tous les jetons ont été reçus en regardant les 2 résultats (on ne demande pas d'envoyer l'information au processus parent).

- Un enfant ne sait pas à l'avance le nombre de jetons qu'il pourra lire. La gestion de la terminaison d'un enfant, après la fin (d'écriture) du processus parent, utilise les propriétés d'un tube anonyme (voir les propriétés de la fonction *read* lorsqu'il n'y a plus d'écrivain : voir encart).
- N'hésitez pas à envoyer plus de 100000 jetons (en tout) pour que les deux enfants puissent en recevoir au moins un peu tous les deux.

Indices :

- Regarder le cours ;-) [transmission des descripteurs, *read* bloquant. . ....] Le tableau des descripteurs sont hérités automatiquement par l'enfant lors du fork. Cependant il faut transmettre par `execl` le numéro du descripteur qu'il devra utiliser (vu que l'enfant à la tableau en mémoire mais on ne connait dans le programme à quoi correspond les descripteurs, à part le 0, 1 et 2) - attention à bien transmettre une chaine de caractères et non directement le `int` (le numéro du descripteur)]
- Il faut penser à fermer les bouts de tube qui ne servent pas (pour éviter une attente infinie, notamment du lecteur qui croit qu'il y a encore un écrivain à l'autre bout alors que c'est lui-même... s'il a oublié de fermer son descripteur pour l'écriture).
- Le choix d'envoi d'un jeton + ou d'un jeton - se fera aléatoirement (soit avec l'opérateur *modulo* % pour commencer ou soit directement avec la fonction *rand*).

2 La redirection

Objectifs sous-jacents de l'exercice

- Comprendre le principe de descripteur de "fichiers" ouverts
- Illustrer l'utilisation des fonctions dup et dup2 (duplication de descripteurs) avec les tubes.
- Au final, réaliser ce que fait l'opérateur *pipe* | sous Linux : enchainer 2 programmes avec la sortie de du premier devient l'entrée du second.

Le but de cet exercice est d'écrire un programme *bypipe* qui réalise le mécanisme de redirection. Ce programme prend en argument deux chaînes de caractères contenant chacune les deux commandes à enchainer. La deuxième commande utilise la sortie de la première comme entrée.

ex : *bypipe* "ls -la" "wc -l" ⇒ compte le nombre "d'objets" contenus dans le répertoire courant (*ls* liste les objets et *wc* les compte).

Pour réaliser les commandes, il est possible de le faire par la création de 2 processus enfants [1] et d'un tube de communication :

1. Il n'y a pas à écrire les programmes enants puisque l'on recouvre tout simplement le code de l'enfant par les programmes passés en paramètres avec `execl`. Il faut juste les créer et les recouvrir.

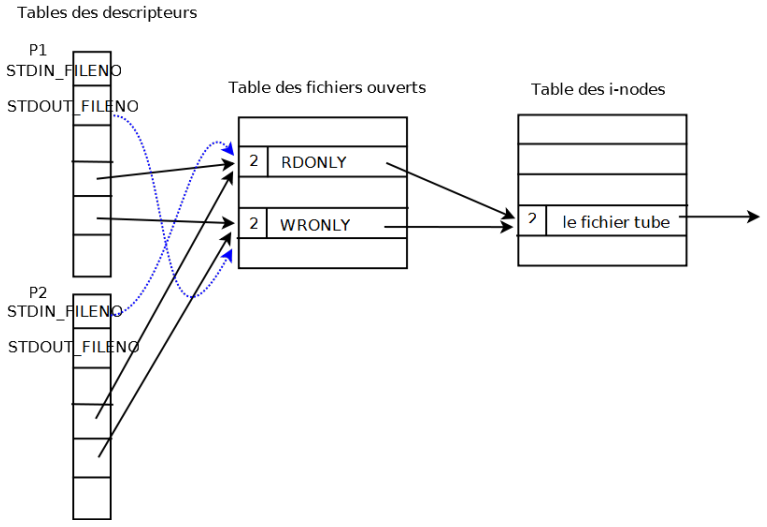
- Il faut donc rediriger la sortie (standard) du premier processus (devant réaliser la première commande) dans un tube.
- Et remplacer l'entrée (standard) du deuxième processus (devant réaliser la deuxième commande) par le tube.

Le principe repose sur la duplication des descripteurs pour associer la sortie et l'entrée standard à respectivement l'entrée et la sortie d'un tube : on utilisera la fonction *dup2*. Rappelez-vous le principe des descripteurs et de la table de fichiers ouverts. La sortie standard a pour descripteur STDOUT_FILENO et l'entrée standard STDIN_FILENO. La fonction **dup2(descr1, descr2)** permet de recopier/dupliquer le descripteur descr1 à l'emplacement du descripteur descr2. Vous pouvez vous aider du schéma (de la page suivante) pour comprendre ce qu'il faut faire. Le schéma représente les fichiers ouverts juste après la création du tube et des deux enfants.

Il faut ainsi faire croire au processus 1 qu'il écrit comme d'habitude sur le descripteur de la sortie standard. Mais en fait vous avez modifié le descripteur de la sortie standard pour qu'il pointe maintenant sur l'entrée du tube... oh c'est malin !

fermeture des descripteurs inutiles

Il ne faut pas oublier de fermer les descripteurs qui pointent entrées/sorties du tube qui ne servent pas (comme pour l'exercice précédent).



Aide en C :

- Dans un premier temps les commandes seront inscrites directement dans les *execl*. Ex : `execl("/bin/ls", "ls", -la", null);)`
- Dans un second temps, pour récupérer les paramètres des arguments de manière automatique, on pourra utiliser la fonction *char *strsep (char **stringp, const char *delim)*. La fonction *strsep()* renvoie l'élément lexical (token) suivant contenu dans la chaîne *stringp*, délimitée par *delim*. Le mot renvoyé est terminé par un caractère nul '\0', et le

pointeur *stringp* est mis à jour pour pointer après le mot. La fonction *strsep()* renvoie un pointeur sur l'élément lexical extrait, ou NULL si le séparateur *delim* n'est pas trouvé dans *stringp*.

```
char *lgcom, *comm1, *comm2 ;
char *args1[30], *args2[30] ;
/* découpage de la ligne de commande donnée en paramètre */
lgcom = argv[1] ;

i = 0 ;
while ((args1[i] = (char *)strsep(&lgcom, " ")) != NULL) {
        i++ ;
}
comm1 = args1[0] ;
```

On pourra ainsi utiliser la fonction *execvp(commande, pointeur_tableau_arguments)* pour le recouvrement : `execvp(comm1, args1)` ;