TD/TP de Système n° 13

Vous aurez besoin de la fonction suivante :
`int pipe(int fildes[2]);` crée un tube avec fildes[0], le descripteur de fichier en lecture sur le tube et fildes[1], le descripteur de fichier en écriture sur le tube.

Exercice 1 : tubes

Écrire une fonction `int tube(const char *cmd1[], const char *cmd2[])` qui exécute l'équivalent de « `cmd1 | cmd2` » et renvoie la valeur de retour de « `cmd2` ».

Exercice 2 :

Écrire un programme qui réalise la commande « `ls -l | wc` ».

Exercice 3 :

Écrire un programme qui évalue une expression en notation polonaise passée en ligne de commande. On suppose que les opérandes sont des entiers et que les opérations de base donnent des résultats entiers. On utilisera les notations '+', '-', 'x' (et non '*') et '/' pour les opérateurs.

Une expression en notation polonaise se présente récursivement de la façon suivante : 'opérateur expression1 expression2' et signifie 'expression1 opérateur expression2'. On supposera définie la fonction `int parseur (int argc, char **argv)` prenant en argument un tableau de chaînes de caractères, représentant une expression en notation polonaise, et renvoyant l'indice du début du tableau correspondant à l'expression expression2.

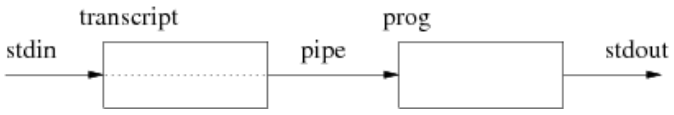
Le programme devra récursivement faire évaluer les expressions expression1 et expression2 par deux processus fils.

1. Pour communiquer avec les fils, utiliser des fichiers temporaires dans lesquels le processus père lira les résultats intermédiaires.
2. Que se passe-t-il si l'on omet l'appel à `wait()` avant de lire dans le fichier ?
3. Remplacer les fichiers temporaires par des tubes.

Exercice 4 : Enregistrement des entrées/sorties d'un programme

1. Dans un premier temps, on ne s'intéresse qu'à communiquer l'entrée clavier à un programme `prog`. Écrivez un programme `transcript` qui lance le programme `prog` passé en argument et communique avec lui grâce à un tube. L'entrée standard de `prog` sera branchée sur la sortie du tube (à l'aide de dup2). `transcript` lit ensuite en boucle les lignes entrées au clavier. Chaque ligne lue est simplement envoyée à `prog` à travers le tube. La sortie standard de `prog` n'est pas (encore) redirigée.

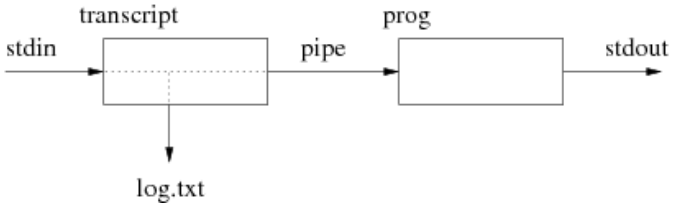
   Voici le schéma de fonctionnement :



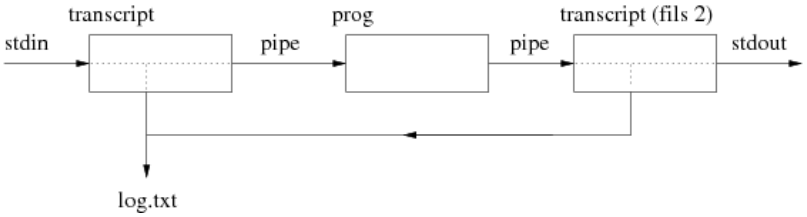
   Testez transcript sur :
   – un programme non interactif sans entrées : `./transcript echo toto`,

– un programme non interactif avec entrées : `./transcript grep toto`,
– un programme interactif : `./transcript bash` ou `./transcript bc`,
– un programme inexistant : `./transcript foobar`.

2. Modifiez `transcript` pour qu'il consigne chaque ligne entrée au clavier dans un fichier `log.txt` avant de l'envoyer à `prog`.
   Le schéma de fonctionnement devient alors :

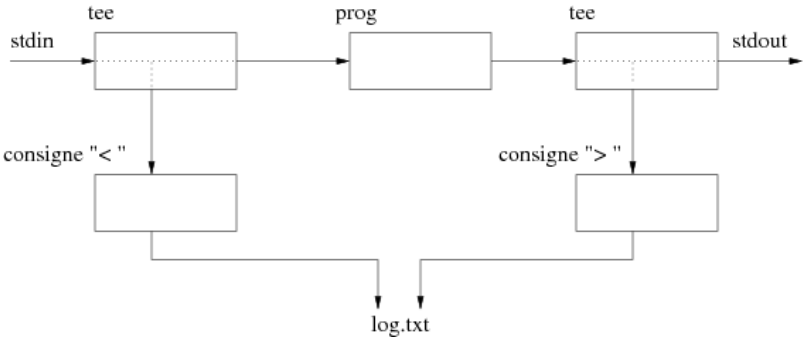


3. On souhaite maintenant consigner la sortie de `prog` en plus de son entrée. On propose le schéma de fonctionnement suivant :



   Dans ce schéma, un appel à transcript génère trois processus qui communiquent à travers deux tubes : le processus de gauche (père) consigne l'entrée clavier, le processus du milieu (fils 1) lance `prog`, enfin le processus de droite consigne la sortie écran.
   Que peut-il se passer si les deux processus accèdent en même temps au fichier `log.txt` ?

4. On souhaite obtenir le même résultat qu'à la question précédente mais en combinant plusieurs petits programmes génériques grâce au shell (ce qui correspond assez bien à la philosophie UNIX) plutôt qu'en utilisant un seul gros programme dédié. On utilisera pour cela le programme `tee` qui recopie son entrée standard à la fois sur sa sortie standard et sur le fichier passé en paramètre. On aura besoin de deux instances de `tee` : une pour dupliquer l'entrée de `prog`, et l'autre pour dupliquer sa sortie. Il ne nous reste plus qu'à réaliser un utilitaire `consigne` qui recopie son entrée standard dans `log.txt` après avoir préfixé chaque ligne du caractère (< ou >) passé en argument.
   On obtient le schéma de fonctionnement suivant :



   Programmez `consigne` en C. Programmez également un script shell qui lance tous les processus et réalise les interconnexions indiquées sur le schéma. Pour cela, les redirections <, > et | du shell seront utiles mais pas su□santes. Il faudra utiliser judicieusement des tubes nommés...