Travaux Dirigés Programmation Système: Feuille 4

Informatique 2ème année. ENSEIRB 2018/2019

—*Mathieu Faverge* - `mfaverge@enseirb.fr` —

Tubes et entrées/sorties sur plusieurs descripteurs

Attention, vérifiez bien la valeur de retour de `fork()` et quittez vos processus au moindre problème. La création de processus dans une boucle est un exercice dangereux qui peut aboutir au plantage d'une machine. Utilisez la commande `ulimit -u 200` pour limiter le nombre de processus auxquels vous avez droit à 200 par exemple. Pour éviter de mettre en difficulté les serveurs, **exécutez vos programmes sur les stations de travail**.

▸**Exercice 1.** *Création d'un tube :*
Écrire un programme qui :

- *crée un tube en utilisant `pipe(2)`.*
- *se duplique (`fork(2)`).*
- *le père écrit 10 fois* **"Hello World"** *dans le tube*
- *le fils lit au plus 500 octets depuis le tube (en une fois), les affiche sur sa sortie standard et quitte.*

note : comme tout bon programmeur, vous utiliserez des macros pour les constantes.

▸**Exercice 2.** *Règles pour les tubes :*
Reprendre le premier exercice et tester dans le cas où le père écrit 50 fois son message, 100 fois, 500 fois, 1000 fois, 5000 fois, 10000 fois, ...

Si votre programme semble bloqué, vérifier le statut des processus (c.f. exo précédent). Sur une feuille blanche, faire une représentation des processus (père et fils) ainsi que de leurs tables des descripteurs. Trouver le problème et corriger votre programme.

Une fois que tout marche bien, lancer votre programme comme suit : `./exo2 &`. Lorsqu'on lance un processus en tache de fond dans le shell, celui-ci nous affiche des informations sur le statut de complétion du processus. Quel est le statut de complétion de votre programme ?

▸**Exercice 3.** *Mise en majuscules :*
Reprendre l'exercice précédent et modifier le fils pour qu'il lise les données en continu depuis le tube et les passe en majuscules avant de les afficher sur sa sortie standard.

Dans le père, faire en sorte que la sortie standard soit associée au tube (`dup2`) et exécuter le programme `ps` en utilisant `execlp`.

▸**Exercice 4.** *Fork, exec et pipe = commandes composées :*
Écrire un programme qui exécute un pipeline de commandes. Le programme aura une variable `commandes` qui contiendra la liste des commandes à executer, chaque commande consiste en un tableau de chaînes de caractères terminé par NULL :

```
char *ls[]={"ls","-l",NULL};
char *tr[]={"tr","-d","[:blank:]",NULL};
char **commandes[]={ls,tr};
int nb_commandes=sizeof commandes/sizeof(char*);
```

La commande se terminera juste après la terminaison du dernier processus du pipeline (`tr` dans l'exemple). Si ce dernier processus s'est terminé normalement, sa valeur de retour devra être propagée dans celle du processus `pipeline`. Autrement un message devra donner la raison de la terminaison du processus et retourner la valeur `EXIT_FAILURE`.

▸ **Exercice 5.** *Reprendre l'exercice* `commandes composées` *et faites en sorte que la liste des commandes qui constituent le pipeline soit fournie sur la ligne de commande. On utilisera la chaîne* `"---"` *comme séparateur de commandes. Par exemple,*

```
$ pipeline ls -l --- grep toto --- less
```

devra produire le même effet que

```
$ ls -l | grep toto | less
```