Linux 2 : Appels Systèmes - BINV2181 (tp03- pipe)

Tous les appels système doivent être testés, en cas d'erreur, un message adéquat doit être affiché et le programme arrêté.

3.A. Exercice préliminaire sur les appels système : `pipe`

Écrivez un programme qui transforme les minuscules en majuscules.

Le programme sera composé d'un processus père lisant à l'entrée standard (clavier) des lignes tant qu'il ne reçoit pas une fin de fichier (Ctrl-D). Les caractères lus seront transmis via un pipe à un processus fils.

Le fils transformera les caractères en majuscules et les affichera sur la sortie standard (écran).

3.B. Exercice sur les appels système : `fork, swaitpid, pipe, dup2` et `execl`

Simulation d'un Pipe Bash en C

Objectif : L'objectif de cet exercice est de simuler le fonctionnement d'un pipe (**|**) dans un environnement Bash. Vous devrez créer un programme qui prend deux commandes en argument et les exécute en redirigeant la sortie standard de la première commande vers l'entrée standard de la seconde, simulant ainsi le comportement de **cmd1 | cmd2**.

Instructions :

1. **Création du pipe :** Commencez par créer un pipe. Ce pipe servira à transférer les données de sortie de la première commande vers l'entrée de la seconde.
2. **Fork du processus :** Utilisez **fork()** pour créer deux processus enfants. Chaque processus enfant exécutera l'une des commandes passées en argument au programme.
3. **Redirection des flux :** Dans le premier processus enfant, redirigez la sortie standard (**stdout**) vers l'entrée du pipe à l'aide de **dup2()**. Dans le second processus enfant, redirigez l'entrée standard (**stdin**) pour lire depuis le pipe.
4. **Exécution des commandes :** Utilisez **execl()** pour exécuter les commandes dans chaque processus enfant. Assurez-vous de fermer les extrémités inutilisées du pipe dans chaque processus pour éviter les blocages.
5. **Synchronisation :** Attendez la terminaison des deux processus enfants avant de terminer le programme parent.

Format de l'entrée : Votre programme doit être exécuté avec la syntaxe suivante :

```
./pipe cmd1 cmd2 arg1 arg2 arg3 ...
```

où **cmd1** est une commande sans argument et **cmd2 arg1 arg2 arg3 …** est une autre commande.

Exemple d'utilisation :

```
./pipe ls grep ".c"
```

Cet exemple liste les fichiers du répertoire courant avec **ls** et utilise **grep** pour filtrer ceux qui se terminent par **.c**.

Consignes supplémentaires :

Gérez les erreurs potentielles pour chaque appel système.

[Pour réviser] 3.C. Exercice de révision sur les appels systèmes.

Écrivez un programme `pgmInscription.c` traitant des demandes d'inscription. Le programme lira un fichier binaire nommé "DemandesInscriptions.bin", contenant des demandes d'inscriptions. La lecture de ce fichier se fera en utilisant la redirection de l'entrée standard présente dans le shell de la manière suivante :

```
./pgmInscription < DemandesInscriptions.bin
```

Le programme principal validera une demande d'inscription sur deux. L'autre demande sera chaque fois envoyée à un processus fils pour être validée par ce dernier. Cette validation sera simplement constituée d'une vérification du nombre d'années du passé de l'étudiant. L'étudiant est accepté si son nombre d'années passées dans l'enseignement est < 3, et refusé sinon. Le père enverra les demandes d'inscriptions via un pipe au processus fils. Le fils répondra au père par 0 (inscription refusée) ou 1 (inscription acceptée) via un deuxième pipe. Le processus père centralisera et affichera le nombre total d'inscriptions acceptées par lui et son fils.

Voici une liste d'étapes à réaliser :

1. Récupérez et exécutez le programme « `createInscriptionRequests.c` ». Celui-ci créera un fichier nommé « `DemandesInscriptions.bin` ». Il s'agit d'un fichier binaire contenant des enregistrements de demandes d'inscriptions.
2. Créez un programme « `pgmInscription.c` » pour effectuer le travail de validation.
3. Mettez en place la communication entre le père et le fils : création et configuration des pipes nécessaires.
4. Prenez connaissance de la structure des enregistrements à traiter (fichier `inscriptionRequest.h`). La lecture des enregistrements et leur transmission au fils doit se faire proprement à l'aide d'appels système `read` et `write`.
5. Réalisez les affichages dans le père suivant le cas :
    1. « Trt par le père : <nom de l'étudiant> <nombre années du passé> »
    2. « Attente trt fils : <nom de l'étudiant> <nombre années du passé> »

6. Implémentez la validation de chaque inscription (simple test sur nombre d'années du passé), ainsi que l'obtention au niveau du père du total des inscriptions validées.
7. Programmez proprement, en vérifiant l'absence d'erreur lors de chaque appel système, et en veillant à ne pas laisser de ressources pendantes inutiles.

[Pour réviser] 3.D. Suite de l'exercice 3.C.

1. Si vous ne l'avez pas encore fait, utilisez une fonction de type *fork_and_run()* du module *utils* pour la création et l'exécution du code du processus fils. On ne veut pas que des informations soient placées dans des variables globales, donc les file descriptors des pipes devront être passés en paramètres de cette fonction *fork_and_run()*, puis de son handler.
2. À l'aide d'un appel système `dup2`, redirigez dans le fils le fd de la sortie standard pour qu'il serve à l'écriture d'informations vers le père.
3. Lorsque le fils réalise qu'il a fini son travail, il libère proprement ses ressources, puis affiche "Moi, le fils, j'ai fini mon boulot" à la console avant de se terminer. Zut, la sortie standard n'est plus disponible à cause du point 2 ! Modifiez votre code pour pouvoir à nouveau utiliser la sortie standard dans le fils lorsqu'il ne doit plus communiquer au père.
4. Le père vérifie que le fils a bien fermé le pipe qu'il utilisait pour communiquer vers le père, puis il affiche le message "OK, mon fils n'a plus rien à me transmettre" à la sortie standard. Le père attend enfin d'être sûr que le fils se termine avant de se terminer lui-même.