TP4 : Processus, tubes et signaux

▸ **Exercice 1 :** Parfois, quand nous travaillons avec plusieurs processus produits par fork, nous avons la sensation que printf ne fait pas son travail. Ce comportement est dû au fait que printf attend une fin de ligne avant d'afficher le contenu du buffer. Considérons le code suivant :

```
# nclude <unistd.h>
#include <stdio.h>
int main() {
        printf("Salut!");
        fork();
        fork();
        sleep (5);
        printf("\n");
}
```

1. Testez-le. Que voyez-vous ?
2. Suggérez une correction (toujours avec printf).
3. Faites-le en remplaçant printf par un appel write approprié.

https ://stackoverflow.com/questions/2530663/printf-anomaly-after-fork

▸ **Exercice 2 :** En utilisant fork(), créer deux processus communiquant par un tube (lui-même créé avec l'appel système pipe()). Le fils lira depuis l'entrée standard et écrira dans le tube les caractères, il mettra un caractère sur deux en majuscules avec toupper(), et l'autre caractère en minuscules avec tolower(), Le père lira depuis le tube et écrira sur la sortie standard.

(Ce programme a été vu en cours en utilisant uniquement toupper)

▸ **Exercice 3 : Highlander**

Écrire un programme qui "résiste" au signal SIGINT (rappel : c'est ce signal qui est envoyé lorsque vous tapez Control-C dans le terminal). Pour cela utiliser l'appel système signal(). Compilez votre code avec et sans l'option -ansi, et comparez les comportements à l'exécution. Que se passe-t-il ?

▸ **Exercice 4 : Highlander 2** Réécrire le programme précédent avec l'appel système sigaction(). Pour manipuler des sigset t, consultez la manpage sigsetops (il faudra utiliser au moins sigemptyset()).

▸ **Exercice 5 : speed-O-meter** Écrire un speed-o-meter, un programme qui mesure la vitesse à laquelle on lui envoie des données (par l'entrée standard), en octets par seconde. Vous pouvez afficher la vitesse toutes les secondes en utilisant un signal SIGALRM périodique (avec alarm() par exemple).

▸ **Exercice 6 : Communication père-fils via un tube**

Écrire un programme en `C` qui :

- crée un tube puis un fils,
- le fils lit des lignes sur l'entrée standard, convertit chaque ligne en entier (avec `atoi()`) et les envoie au père via le tube,
- le père lit ces entiers depuis le tube, calcule le minimum et le maximum,
- lorsque le fils atteint la fin de l'entrée, il ferme le tube,
- le père affiche alors `min = ...` et `max = ...`.

Toutes les communications se font sous forme **textuelle**. Utiliser uniquement `read()` et `write()` pour les échanges entre père et fils.

Variante : faire un tube bidirectionnel, où le père calcule et envoie au fils une ligne contenant la moyenne courante (de tous les entiers saisis précédemment). Le père envoie cette moyenne à chaque fois qu'il reçoit un entier du fils, mais le résultat doit être affiché par le fils seulement toutes les 5 secondes (utiliser un signal).

▸ **Exercice 7 :**

Dans cet exercice nous aurons 2 processus, père et fils, qui se communiqueront aussi à travers un tube où le fils écrira au père des chiffres aléatoires (utiliser les fonctions rand et srand). Le fils écrira ses chiffres à des intervalles aléatoires (d'un à cinq secondes), tandis que le père regardera le contenu du tube une fois par seconde, en affichant sur la sortie standard s'il y a eu des changements.

1. Ecrire ce code, en rendant le côté du pipe du père non bloquant.
2. L'appel-système select (faire "man select") permet aux processus d'attendre jusqu'à ce qu'un file descriptor devienne "ready". L'appel prévoit aussi un "timeout", une limite de temps que l'on attendra sur select. Résoudre cet exercice en utilisant select au lieu de rendre le tube non bloquant.