Exercice 1 - mise en route

Préparation : Ecrire un programme qui affiche bonjour, puis exécute un `fork` (sans `switch`, ne gérez pas son code de retour), puis affiche au revoir. Aviez-vous prévu le résultat ?

Faire un programme qui execute (avec `exec`) le programme `ls`, puis affichez le message "*ls est terminé*". Selon ce que vous ferez, ce message n'apparaitra pas (trouvez pourquoi). Ensuite, si il apparait, il pourrait être placé au milieu du ls, et vous aurez peut être l'impression que votre programme ne s'est pas terminé. Pourquoi ?

Exercice 2 - Hit the road, Jack

- Positionner des variables d'environnement `RUN_0`, `RUN_1`, `RUN_2` dans votre shell..., puis, écrire un programme les affichant les unes après les autres. Il faut utiliser la fonction `getenv()` pour récupérer la valeur d'une variable. Le programme devra afficher toutes les variables, jusqu'à ce qu'il y en ait une qui n'existe pas; par exemple s'il y a `RUN_0`, `RUN_1` et `RUN_3`, il ne faut afficher que la 0 et la 1 (comme la 2 n'existe pas, le programme doit s'arrêter).
- En utilisant `fork()` et `execvp()`, écrire un programme, qu'on appellera `mrun`, qui exécute les programmes présents dans `RUN_0`, `RUN_1`, etc., avec les arguments de la ligne de commande. Par exemple, si `RUN_0=ls` et `RUN_1=cat`, alors la commande `mrun toto.c titi.c` devra faire la même chose que `ls toto.c titi.c ; cat toto.c titi.c`. Il faut attendre l'exécution d'une commande avant de lancer la suivante, en utilisant `wait()`.

Exercice 3 - Accident de fourchette

- Question piège: écrivez un processus qui, grâce à l'appel système `fork()`, lance 20 processus affichant chacun "*je suis le numéro <1 à 20>, mon PID est <pid> et mon père est <pid>*" (utilisez `write()` sur ce coup-là). Réfléchissez bien avant de lancer votre programme (si vous ne faites pas attention, vous allez planter votre PC; c'est conseillé de tester avec 3 processus avant...).
- Une fois que le programme précédent marche correctement, modifiez-le de façon à ce que le processus père, après avoir lancé les 20 fils, boucle indéfiniment (`for(;;) pause();`). Avec la commande `ps`, listez vos processus: pourquoi les 20 fils sont-ils encore là, bien que leur exécution soit terminée ?
- Utiliser l'appel système `waitpid()` pour corriger le problème de la question précédente. Pour chaque processus fils terminant son exécution, afficher "*le processus n°<x> de PID <y> vient de terminer*".

Exercice 4 - Au coeur des ténèbres

Ecrire un programme qui manipule un entier i, par exemple en lui attribuant la valeur .

- Affichez la valeur de i, et son adresse en mémoire.
- Forkez ! Affichez la valeur et l'adresse de i dans les deux processus. Comparez.
- Maintenant, le père et le fils modifie la valeur de i : affichez les valeurs et les adresses de i. Que remarquez-vous ? Les i sont ils bien différents ? et les adresses ?

Exercice 5 - Faire-parts de décès

Lorsqu'un processus fils se termine, le système envoie un signal `SIGCHLD` au processus père. Utiliser un handler de signal pour notifier sur la sortie standard de la terminaison des fils.

Exercice 6 - Anne, ma soeur Anne, ne vois-tu rien venir ?

Écrire un programme effectuant les opérations suivantes, dans l'ordre :

1. afficher "*mon PID est* " avec `printf` (sans retour à la ligne); pour obtenir le PID: `getpid()`
2. afficher "*my PID is* " avec `write` (toujours sans retour à la ligne)
3. `fork()`
4. afficher "*je suis le <père|fils> et mon PID est* " avec `printf` (toujours pas de retour à la ligne)
5. afficher "*I am the <parent|child> process and my PID is*" en utilisant `write` (toujours pas de retour à la ligne)
6. forcer l'affichage de tous ces messages, par exemple avec un `puts("")`.