Programmation système

TP 4 – `fork()`/`exec*()`

Exercice 1: Mise en route

Écrivez un programme qui affiche `"Bonjour\n"`, puis exécute un `fork()` (*exceptionnellement* sans `switch`, ne gérez pas son code de retour), puis affiche `"Au revoir\n"`. Aviez-vous prévu le résultat ?

Exercice 2: Affichage bilingue

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

- Afficher "mon PID est `<pid>`" avec `printf()` (sans retour à la ligne) ; pour obtenir le PID, servez-vous de `getpid()`.
- Afficher "my PID is `<pid>`" avec `write()` (toujours sans retour à la ligne).
- `fork()` (à partir de maintenant, toujours avec un `switch`).
- Afficher "je suis le `<parent|enfant>` et mon PID est `<pid>`" avec `printf()` (toujours pas de retour à la ligne).
- Afficher "I am the `<parent|child>` and my PID is `<pid>`" en utilisant `write()` (toujours pas de retour à la ligne).
- Afficher finalement un retour à la ligne, par exemple avec un `puts("")`.

Comprenez-vous pourquoi l'affichage ne correspond pas tout à fait à ce qu'on pourrait attendre ?

Exercice 3: Lancer des programmes

1. Écrivez un programme qui lance l'exécution d'un `ls`.
2. Modifiez le programme afin qu'il lance un `ls`, puis un `ps`, puis un `free`.

Exercice 4: Variables d'environnement

1. Positionnez des variables d'environnement `RUN_0`, `RUN_1`, `RUN_2` …, puis, écrivez 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).
2. En utilisant `fork()` et `execvp()`, écrivez un programme, qu'on appellera `mrun`, qui exécute de manière séquentielle 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 la séquence de commandes `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 5: Accident de fourchette

1. Question piège : écrivez un programme 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 parent est `<pid>`" (utilisez `write()` sur ce coup-là). Réfléchissez bien avant de lancer votre programme (si vous ne faites pas attention, vous pouvez ralentir momentanément votre PC – vous êtes prévenus).
2. Une fois que le programme précédent marche correctement, modifiez-le de façon à ce que le processus parent, après avoir lancé les 20 enfants, boucle indéfiniment (par exemple, `while (1) pause();`). Avec la commande `ps`, listez vos processus. Pourquoi les 20 enfants sont-ils encore là, bien que leur exécution soit terminée ?
3. Utilisez l'appel système `waitpid()` pour corriger le problème de la question précédente. Pour chaque processus enfant terminant son exécution, affichez "le processus numéro `<i>` de PID `<pid>` vient de terminer".