TP6

La page web du cours :

`http://www.lsv.ens-cachan.fr/~schwoon/enseignement/systemes/ws1415/`.

Rappel : Utilisez `man` pour obtenir la documentation des appels système.

1 Fork

Téléchargez le programme `toto.c`.

1. Sans exécuter le programme, quel est son comportement attendu ? Le programme, va-t-il terminer ? Dans quel ordre les processus seront-ils crées ?
2. En général, les processus forment un graphe défini par la relation entre père et fils. Quelle est la forme du graphe dans cet exemple ?
3. Téléchargez et compilez le programme `tototree.c` (qui sera mis à disponibilité pendant l'exercice). Ceci est une variant de `pstree`, limité aux programmes avec `toto` dans leur nom et qui permettra de visualiser l'arborescence de (b). Pour cela, faites `toto.c` attendre pendant 10 seconds avant de terminer, puis exécutez `tototree` dans une autre fenêtre.
4. Augmentez `BOUND` à 5 et rajoutez une boucle infinie à la fin de `main`. (Il est fortement conseillé de faire cela par `sleep` et non par attente active !)

   Utilisez `kill` pour tuer certains instances de votre programme et observez l'effet avec `tototree`. Qu'est-ce qui se passe dans des cas différents, p.ex. le cas d'un processus « feuille », le processus « racine » ou un processus à l'intérieur de l'arborescence ? Quel est l'effet de Ctrl+C lorsque la racine est en vie et lorsqu'elle n'est pas ? Quid des zombies ?

   Remarque : on peut utiliser `killall toto` pour se débarasser de toutes les instances.

2 Calculatrice

Dans cet exercice on étudie l'utilisation et évaluation du code de sortie d'un processus. Normalement, le code de sortie est utilisé pour indiquer p.ex. la résussite ou non d'un programme. Ici, on s'en sert pour communiquer les résultats d'un calcul. (Remarque : le code de sortie doit entre entre 0 et 255, donc ce n'est pas idéal pour ce genre de chose - c'est juste un exemple !)

Pages man utiles : fork(2), wait(2). Pour (ii), utilisez `make` pour compiler.

1. On commence avec un exemple simple (`simple.c`) qui calcule la somme de deux entiers. Complétez le programme tel que le fils utilise `exit` pour communiquer la somme au père et que le père reçoit cette valeur par `wait` / `WEXITSTATUS`.

2. Application plus complexe : La page web du cours contient un calculateur simple qui évalue des expressions avec des entiers naturels, addition, multiplication et subtraction.

   Dans l'état actuel le programme convertit l'expression vers un arbre, puis il traverse les nœuds de l'arbre un par un pour calculer les valeurs de toutes les sous-expressions. Votre tâche est de permettre au programme de calculer les sous-expressions *en parallèle*.

   Lorsque le programme évalue une sous-expression, il devrait lancer deux processus fils qui évaluent leurs sous-expressions et qui on communiquent le résultat au père par leurs codes de sortie. Le père attend les deux résultats et applique l'opération (+,*,-) pour obtenir son propre résultat. Il suffit de modifier la fonction `compute` dans `main`.

   Remarque : Il est légèrement plus facile de réaliser la multiplication et l'addition que la subtraction – pourquoi ?

3 Programmation d'un shell

L'objectif ce cette exercice est de créer une version simple d'un shell comme `bash`.

Sur la page web vous trouverez un code squelette qui connaît déjà le syntaxe pour écrire des commandes avec enchaînements et redirection des entrées/sorties. Pourtant, il ne sait que réaliser que les commandes simples (sans enchaînement ni redirection ...).

L'objectif sera de compléter ce programme peu à peu. Pour l'instant, rajoutez la possibilité d'enchaîner des commandes avec `;`, `&&` et `||`. Pour ce faire, il suffit de modifier la fonction `execute` dans `main.c` et de remplir les cas `C_SEQ`, `C_AND` et `C_OR`.