Processus & signaux

Gabriel Hondet
gabriel.hondet@lsv.fr

Exercise 1 :

Dans cet exercice on étudie l'utilisation et l'évaluation du code de sortie d'un processus. Normalement, le code de sortie est utilisé pour indiquer par exemple, 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).

Utilisez make pour compiler.

(a) 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`.

(b) Application plus complexe : télécharger l'archive `www.lsv.fr/~hondet/resources/archos/calc.zip` qui contient un calculateur simple qui évalue des expressions avec des entiers naturels, additions, multiplications et soustractions. Dans l'état actuel le programme convertit l'expression vers un arbre, puis il traverse les noeuds 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 communiquent le résultat au père par leurs codes de sortie. Le père attend les deux résultats et applique l'opération correspondante pour obtenir son propre résultat. Il (devrait) suffire de modifier la fonction `compute` dans `main`.

Exercise 2 :

Écrire un programme qui :

- crée un processus fils ;
- transmet une séquence de bits entrés au clavier, bit par bit à ce processus fils.

Le processus fils doit afficher la séquence dans le bon ordre. Pour cela, utilisez des signaux. Prenez bien soin de tester plusieurs motifs de séquences (alternances de 0/1, successions de 0, successions de 1).

Pour entrer les bits, vous pouvez utiliser

```
for (c = 0; c != 0 && c != 1 && c != EOF; c = getchar());
if (c == 0) // Do something
if (c == 1) // Do something
if (c == EOF) break;
```

(a) Implémentez dans un premier temps avec l'API ANSI C (fonctions `kill`, `pause`, `signal`).

(b) puis avec l'API Posix.

Exercise 3 :

Programmation système et λ calcul ne sont pas incompatibles : le dialecte Guile du Scheme implémente les API Posix. Écrivez un programme (en Guile) qui

- se fork
- le père attend un certain temps puis tue son fils (`SIGINT`),

- le fils imprime des citations sur la sortie standard avec la commande `fortune -e` (`-o` pour être insultant) et une pause entre chaque sortie.

Vous aurez besoin de la doc Guile sur Posix `https ://www.gnu.org/software/guile/manual/html_node/POSIX.html`.

Comme point de départ :

```
#!/usr/bin/guile \
-e main -s
!#
(define (main args)
  (let ((pid (primitive-fork)))
    (if (= pid 0)
        (...)
        (...))))
...
```

le fichier peut être rendu exécutable puis être interprété.

Vous pouvez bien entendu l'implémenter d'abord en C, pour apprécier le `execl` ou `execlp`.