TP5 Gestion de signaux - Système d'exploitation

Sereysethy Touch

5 novembre 2018

Nota :
— Que les exercices marqués * sont obligatoires, vous devez rendre vos travaux au plus tard le **16/11/2018**.
— Pour chaque exercice, il faut toujours un fichier Makefile qui permet de compiler vos programmes et éventuellement un fichier `README.md` indiquant comment exécuter votre programme.
— S'il s'agit de documenter un programme, il faut simplement un fichier `README.md`.

Exercice 1*

1. Écrivez un programme qui ignore tous les signaux et affiche son *pid*. A partir du shell, envoyez les différents signaux. Qu'observez-vous lorsque vous envoyez un signal `SIGKILL` ?
2. Écrivez un programme qui traite les signaux avec un comportement par défaut. A partir du shell, envoyez les signaux suivantes : `SIGINT`, `SIGQUIT`, `SIGUSR1`, `SIGUSR2`. Qu'observez-vous ?

Exercice 2*

Écrivez un programme qui crée un processus fils et attend la fin de processus fils pour se terminer.

Le processus fils envoie lui-même un signal `SIGALRM` tous les 3 secondes et se met en attente indéfiniment. Lors de la réception du signal, il affiche un message. À la réception du 3ième signal `SIGALRM`, il sort de sa boucle indéfinie et se termine. Le processus fils ignore tous les autres signaux.

Exercice 3*

Écrivez un programme en C qui crée un processus fils, attend 5 secondes, lui envoie ensuite un signal `SIGUSR1`, attend de nouveau 5 secondes avant de lui envoyer un signal `SIGUSR2`, et enfin, attend la fin du processus fils pour se terminer.

Le fils, quant à lui, attend indéfiniment les signaux ; à la réception d'un signal, il affiche le signal reçu, et si ce signal est `SIGUSR2`, il sort de sa boucle d'attente et se termine.

Question subsidiaire : que se passe-t-il si le processus père envoie le signal `SIGSTOP` au lieu de SIGUSR1 ? La situation est-elle différente avec le signal `SIGINT` ?

Question subsidiaire : comment améliorer le programme pour que le processus père ne soit pas interrompu lorsque l'on tape un `CTRL-C` ?

Attention : il est interdit d'implémenter une attente active.

Primitives : `kill()`, `sigaction()`, `fork()`, `sleep()`, `pause()`

Exercice 4*

Écrivez un programme qui crée un fils qui fait un calcul sans fin. Le processus père propose alors un menu :

- Lorsque l'utilisateur appuie sur la touche `"s"`, le processus père endort son fils.
- Lorsque l'utilisateur appuie sur la touche `"r"`, le processus père redémarre son fils.
- Lorsque l'utilisateur appuie sur la touche `"q"`, le processus près tue son fils avant de se terminer.

Exercice 5*

Écrivez un programme qui prend un nombre entier n et un nombre de processus m. Le programme va créer m processus fils qui partagent le travail de manière égale pour calculer la somme d'entiers de 1 jusqu'à n de manière égale.

Primitives : `sigqueue()`, `sigaction()` avec `SIG_INFO` comme flag.

Exercice 6* - producteur/consommateur

Écrivez un programme qui crée un processus fils. Le père et le fils s'échangent des données à travers d'un fichier commun. Le processus père se comporte comme un producteur qui écrit un nombre entier partant de 0 jusqu'à n sur ce fichier. Les nombres écrits sont stockés comme une liste des nombres séparés par un retour à la ligne `\n`. Le père ne peut pas écrire plus de 5 numéros dans le fichier.

Le processus fils, quant à lui, lit un numéro, fait la somme et puis recommence les mêmes opérations. Il affiche le résultat de la somme avant de terminer son exécution.

Le programme s'arrête quand le père et le fils écrit et lit n fois. n est un nombre prédéfini.

Vous devez utiliser les signaux pour synchroniser le processus père et le processus fils. D'après vous, est-ce que la solution que vous proposez peut effectivement résoudre le problème de synchronisation.

Attention : il est interdit d'implémenter une attente active.

Primitives : `kill()`, `sigaction()`, `fork()`, `sleep()`, `lseek()`