I2180 : Exercices de Programmation Système (4)

4.1. Appels système : `kill & sigaction`

a) Concevez un processus père qui engendre un fils exécutant une boucle infinie. Le père devra envoyer un signal SIGUSR1 à son fils. Que se passe-t-il ?

b) Modifiez votre programme pour que le fils affiche le message « signal SIGUSR1 reçu ! » à la réception du signal, pour reprendre ensuite sa boucle infinie. Le père doit effectuer `sleep(1)` après `fork` afin de s'assurer que son fils a le temps d'armer un handler pour SIGUSR1.

c) Processus chat (aux 7 vies) : modifiez votre code pour que le père envoie régulièrement un signal SIGUSR1 à son fils ; le fils ne devra se terminer que lorsqu'il aura reçu 7 fois le signal de son père.

Exemple d'exécution :

```
OK il me reste 7 vies
Le pere envoie le signal SIGUSR1 a son fils
--> Signal SIGUSR1 reçu! OK il me reste 6 vies
Le pere envoie le signal SIGUSR1 a son fils
--> Signal SIGUSR1 reçu! OK il me reste 5 vies
Le pere envoie le signal SIGUSR1 a son fils
--> Signal SIGUSR1 reçu! OK il me reste 4 vies
Le pere envoie le signal SIGUSR1 a son fils
--> Signal SIGUSR1 reçu! OK il me reste 3 vies
Le pere envoie le signal SIGUSR1 a son fils
--> Signal SIGUSR1 reçu! OK il me reste 2 vies
Le pere envoie le signal SIGUSR1 a son fils
--> Signal SIGUSR1 reçu! OK il me reste 1 vies
Le pere envoie le signal SIGUSR1 a son fils
--> Signal SIGUSR1 reçu! That's all, folks!
Fin du fils
Fin du pere
```

4.2. Appel système : `sigaction`

Ecrivez un programme qui, à la réception d'un signal, affichera simplement le signal reçu (utilisez le tableau *`sys_siglist[sig]`* défini dans <signal.h>) avant de reprendre son traitement. Pour ce faire, limitez-vous aux 15 premiers signaux (pour afficher la liste des signaux définis sur votre système, tapez la commande `kill -l`). Le traitement de votre programme consistera en une boucle infinie effectuant une lecture au clavier (`read`). Affichez un message avant d'exécuter un `read`, ainsi qu'un message indiquant si une erreur s'est produite pendant son exécution (`errno=INTR`).

Testez votre programme avec différents signaux (commande `kill` exécutée depuis un autre terminal). Essayez notamment d'arrêter le processus avec Ctrl-C.

Voici un exemple d'exécution :

```
I am process 4976
Signal 0 ((null)) non capturé
Signal 9 (Killed) non capturé

appel read()
^C
 Signal 2 (Interrupt) reçu
read() interrompu

appel read()
                                        $ kill -10 4976
 Signal 10 (User defined signal 1) reçu
read() interrompu

appel read()
                                        $ kill -9 4976
Killed
```

4.3. Appel système : `sigaction`

On veut écrire un programme immunisé contre les processus zombies.

Reprenez votre solution de l'exercice 4.1.c. Le processus père exécute une boucle infinie pour envoyer régulièrement des signaux à son fils et ne sait pas quand celui-ci mourra. Il ne peut donc se mettre en attente (`wait`) car cela bloquerait son traitement.

Adaptez votre code afin que le père supprime automatiquement son fils zombie. Pour ce faire, il doit dérouter le signal SIGCHLD reçu à la mort de son fils et supprimer celui-ci de la table des processus avant de se terminer.