TP 5 : signaux 

Finissez le TP 4 avant de commencer celui-ci.

Exercice 1. signal

- Ecrire un programme *signaux.c* qui compte tous (enfin presque) les signaux qu'il reçoit et affiche ces compteurs.

Exemple d'utilisation :

```
$ signaux &
[2] 4640

Je ne peux pas capturer le signal no 9
Je ne peux pas capturer le signal no 19
$ kill -USR1 4640
Signal 10 recu 1 fois

$ kill -USR1 4640
Signal 10 recu 2 fois

$ kill -CONT 4640
Signal 18 recu 1 fois
```

Fonctions utiles :

includes	fonctions
`#include <signal.h>`	`void (*signal(int sig, void (*func)(int)))(int);`
`#include <unistd.h>`	`int pause(void);`

Le nombre de signaux disponibles dans le système est donné par la constante NSIG.

Exercice 2. SIGCHLD

- Ecrire un programme *sig_chld.c* qui créé un processus fils par un `fork()` puis :
  - Le père place un gestionnaire de signal sur le signal SIGCHLD puis se met en `pause()`.
  - Le gestionnaire affiche simplement le numéro de signal reçu.

- Le fils affiche son pid, attend une seconde et se termine.

Fonctions utiles :

includes	fonctions
`#include <signal.h>`	`void (*signal(int sig, void (*func)(int)))(int);`
`#include <unistd.h>`	`int pause(void);`
`#include <unistd.h>`	`unsigned int sleep(unsigned int seconds);`

Qu'en concluez-vous ?

- A partir de cette conclusion, proposez un mécanisme simple qui évite la création de processus zombies et testez-le dans un programme *sig_chld1.c*

Bonus : il existe une façon encore plus simple d'empêcher les processus zombies : il suffit d'ignorer le signal SIGCHLD dans le père. Testez-le dans un programme *sig_chld2.c*.

Exercice 3. SIGHUP

- Ecrivez un programme *sig_hup.c* qui place un gestionnaire sur le signal SIG_HUP puis se met en pause. Le gestionnaire écrira "Signal SIGHUP reçu" dans le fichier *sighup.txt* puis se terminera. Vous utiliserez les fonctions *standard* (`fopen`, `fprint`...) pour manipuler le fichier.

Pour utiliser ce programme vous ouvrirez un nouveau terminal, exécuterez le programme et fermerez la fenêtre du terminal.

Vérifiez que le fichier sighup.txt est bien créé.

On attend souvent dire que quand un processus père est tué, tous ses fils sont tués. L'exemple qui est souvent pris consiste à ouvrir un terminal, à lancer un éditeur de texte en tâche de fond (par exemple : gedit &) puis à fermer le terminal. On peut alors constater que l'éditeur de texte se termine.

Qu'en pensez-vous ?

Exercice 4. Signaux et synchronisation

- Ecrire un programme qui crée deux processus. Le père affichera les entiers pairs compris entre 1 et 100, le fils affichera les entiers impairs compris dans le même intervalle. Synchroniser les processus à l'aide des signaux pour que l'affichage soit **1 2 3 ... 100**.

Cette solution est-elle sûre ?

Question annexe : comment se comportent les signaux vis-à-vis de l'héritage père/fils ?