# TP 2 – Les signaux

Les exercices sont à faire dans l'ordre.

L'exercice 1 sera **noté**. Il est à rendre pour la fin de la séance. Le reste du TP est à faire (pendant et en dehors de la séance) et doit être rendu avant la séance suivante. Il ne sera pas noté, mais vous travaillez pas que pour les notes. Non ?

**Important : Signaux et processus *init***

- ☞ Jusqu'à Ubuntu/Linux < 14.04, un processus fils n'ayant plus de père était rattaché au processus 1 (*init*). Si l'utilisateur envoyait un signal par inadvertance à ce processus, il n'y avait pas de problème car l'utilisateur ne peut envoyer de signaux qu'à ses propres processus.
- ☞ Sur les machines avec Ubuntu > 14.04, les processus "zombie" sont maintenant rattachés au processus *leader* de la session ! ! Résultat, un envoi de signal peut entrainer maintenant la fermeture immédiate de votre session ! **Il devient donc encore plus important de maitriser les fonctions `wait` et `waitpid` d'un processus père !**

## 1 Attrape signaux à la volée

Nous voulons qu'un processus PERE envoie N signaux à un processus fils FILS.

Soit le processus PERE ayant le comportement suivant :

- PERE crée le processus fils FILS
- PERE envoie au fils N signaux de type SIGUSR1 (N > 100000) :
- Lorsque l'envoi de tous les signaux est terminé, le père envoie SIGTERM au fils puis il attendra la fin de son fils.

Le programme fils a le comportement suivant :

- FILS compte le nombre de signaux SIGUSR1 reçus ;
- À la réception du signal SIGTERM, FILS affiche sur la sortie standard le nombre de signaux SIGUSR1 reçus et se termine.
- Lorsque FILS affiche quelque chose il affichera en même temps son pid.

Écrivez les **deux** programmes C permettant de simuler le comportement de PERE et de FILS en suivant les étapes suivantes. N'oubliez pas de commenter votre programme (comme tout programme normalement !).

Les fichiers contenant les programmes s'appelleront `exo1pere.c` et `exo1fils.c`. Le père recevra en argument (via `argv`) le nombre `N` de signaux à envoyer.

Attention : ces deux contraintes sont là pour nous permettre d'automatiser une partie de la correction. Merci de les respecter.

Avant de commencer à envoyer les N signaux, il ne faut pas oublier de prendre en compte plusieurs phénomènes

- Si aucun handler n'est prévu (ou n'a pas encore été effectivement mis en place par le processus) pour les signaux SIGUSR1 ou SIGTERM, le processus qui reçoit est alors tué (c'est le comportement par défaut).
- Le père peut (si on ne fait pas attention) commencer à envoyer ses signaux avant que le fils n'ait eu le temps mettre en place son mécanisme de handler pour le signal SIGUSR1.... Utilisez la fonction `sleep` (pour endormir le père 1 seconde).
- N'oubliez pas que l'on ne contrôle pas exactement l'instant d'envoi ou de réception d'un signal.
- Vous pouvez essayer d'envoyer des signaux du père vers le fils. A priori pour cet exercice en mettant simplement en place le mécanisme d'envoi d'un côté et la réception de l'autre, il est tout à fait normal que le fils ne puisse pas réussir à attraper tous les signaux. Nous réglerons ce problème dans l'exercice suivant. L'important ici est de bien mettre en place les choses de base.

## 2 Attrape signaux de précision

Copier/Coller les codes de l'exercice précédent dans de nouveaux fichiers. On va les compléter. Si vous envoyez l'ensemble des signaux avec une simple boucle avec la fonction `kill`, souvenez-vous que le fils n'aura sûrement pas le temps de prendre en compte tous les signaux envoyés massivement (il n'y a pas de mémorisation) ...

- Avant que PERE envoie le signal suivant, il faut s'assurer que le signal venant d'être envoyé a bien été reçu par le fils...Comment remédier à ce problème ? [Indice : le fils a le droit d'envoyer des signaux]. Du côté du père, il y a deux façons de mettre en place la solution :
  — Solution 1 : Dans le processus PERE l'utilisation de masque et de la fonction `sigsuspend` (au lieu de `pause` pour attendre un signal de réponse). La fonction `sigsuspend` rend atomique les actions de changement de masque et d'attente d'un signal (voir le cours et le schéma donné sur ARCHE).
  — Solution 2 : En gérant l'envoi des signaux dans le handler.
- Profitez-en pour éviter de passer par la fonction `sleep` dans le père (pour s'assurer que le fils a eu le temps de placer son handler). N'oubliez pas qu'il faut attendre que le fils soit prêt avant de commencer à envoyer le premier signal.

# 3 Interruptions et masques

Dans cet exercice, pour chaque question, vous proposerez un programme qui permette de répondre à la question et qui illustre l'explication que vous donnerez. L'explication/réponse à la question sera donnée sous forme d'un commentaire en début de programme et est aussi importante que le programme lui-même. [Vous appellerez les programmes question1, question2...]

## Question 1

Lorsqu'un processus est bloqué/endormi dans un "sleep", peut-il être interrompu par un signal ? (n'oubliez pas de faire un handler pour éviter que le processus recevant le signal ne soit tué directement)

## Question 2

Pendant l'exécution d'un handler, il est dit que le même signal est masqué automatiquement. Habituellement on initialise le champ $sa\_mask$ de la structure $sigaction$ à un masque vide. Est-ce qu'en faisant cela, le signal est tout de même masqué pendant l'exécution du handler ? Ensuite, initialisez le champ $sa\_flag$ à SA_NODEFER. Le signal est-il toujours masqué ?

## Question 3

Si l'on masque un signal dans le père et qu'ensuite on crée un fils, le même signal sera-t-il masqué chez le fils ? Examinez les deux cas possibles :

- il y a un seul programme (pas de recouvrement)
- il y a deux programmes : le fils est recouvert avec un "*execl*"

**Rassemblez tous les fichiers utilisés pour les exercices 2 et 3 dans un fichier zip (Nom_Prenom.zip) et déposez-le sur ARCHE .....**