Université de Nice Sophia Antipolis
UFR Sciences - Département d'Informatique

Licence d'Informatique L2
Introduction aux Systèmes et Réseaux

TP n° 2 : Processus: Signaux et terminaison

(Feuille d'exercice adaptée du cours de S. Krakowiak, U. Grenoble.)

1 Traitement des signaux d'interruption

1. On rappelle que la primitive `unsigned time.sleep(t)` suspend le processus appelant pendant `t` secondes ou jusqu'à l'arrivée d'un signal.
   Écrire un programme qui suspend le processus pendant un nombre de secondes passé en paramètre.
2. Écrire un programme qui lorsqu'il reçoit le signal `signal.SIGINT` affiche le numéro de ce signal, et ne fait rien le reste du temps (on rappelle que la primitive `signal.pause()` permet à un processus de se bloquer en attendant l'arrivée d'un signal). Faire de même avec `signal.SIGKILL` (primitive `os.kill(pid,sig)`) et `signal.SIGSTOP` (*control-Z*). Que constatez-vous ?
3. Exécuter le programme de la question 2.3 du TD2. En ajoutant un `time.sleep()` dans le programme des fils (après l'instruction `os.kill()`), voir que l'on peut obtenir différentes valeurs (entre 1 et 5) pour la valeur finale de `counter`.

2 Traitement de la terminaison d'un fils

1. Tester le programme présenté au debut de la section 2 du TD1 qui attend la mort d'un fils et affiche le résultat. Ce programme est disponible sur la page wiki sous le nom `waitpid1.py` (voir en bas de la page wiki).
2. Reprendre le programme `waitpid1.py` et modifier ce programme pour qu'il ait l'effet suivant :
   - chaque processus fils s'endort 20 secondes avant de se terminer normalement par sys.exit.
   - après la création de chaque fils, le père affiche le numéro de pid du fils créé
   - le père indique la fin anormale des fils en imprimant le numéro de chaque fils et la cause de sa terminaison (numéro de signal).
   - tester ce qui se passe quand un processus fils est tué par la commande `kill` depuis une nouvelle fenêtre shell, avant qu'il ne se réveille.

   On rappelle (cf TD 1) que `os.waitpid` renvoie un tuple `(pid,statut)` et que l'on peut tester le `statut` pour connaître l'état du processus qui s'est terminé. En particulier :
   - `os.WIFSIGNALED(statut)` renvoie vrai si la fin du processus est due à un signal non traité.

– si `os.WIFSIGNALED(statut)` renvoie vrai, alors `os.WTERMSIG(statut)` renvoie le numéro du signal qui a causé la terminaison du processus.

3. (**non prioritaire, à faire hors TP**)

   Lorsqu'un processus crée des processus fils, il peut attendre leur fin avec la primitive `os.wait` ou `os.waitpid`. Néanmoins, il ne peut pas faire de travail utile pendant cette attente. C'est pourquoi on souhaite que le processus père traite la fin de ses fils uniquement au moment où cette fin est signalée par le signal `signal.SIGCHLD`.

   Sur la page wiki, on trouvera le texte d'un programme appelé `signal1.py.txt`, dans lequel le père crée dix fils et traite le signal `signal.SIGCHLD` envoyé lors de leur terminaison. Puis le père lit quelque chose au clavier et se met en boucle.

   Exécuter ce programme, attendre que le père se mette en attente d'une saisie, le suspendre par *control-Z* et vérifier, par `ps -x`, que tous les fils n'ont pas été collectés (il reste des zombis). Cela résulte du fait que les signaux ne sont pas mémorisés (un seul signal d'un type donné peut être dans l'état pendant). Modifier le programme pour corriger cette erreur.