Université Côte d'Azur
Faculté des Sciences - Département d'Informatique

Licence d'Informatique L2
Introduction aux Systèmes et Réseaux

TP n ° 3 : Processus: Signaux et terminaison

1 Capture des signaux en Python

1. Recopier le programme `testsig.py` suivant :

```
import time

if __name__ == '__main__':
    while True:
        time.sleep(1)
        print("Alive!")
```

   Exécuter, puis au bout de quelques secondes interrompre en tapant `^C`.

2. Nous allons protéger le programme en captant le signal `SIGINT`. Rajouter dans le programme :

```
def capter_INT(sig_num, frame):
    print("Ouch!")
```

   puis au début du `main` rajouter : `signal.signal(SIGINT, capter_INT)`
   Exécuter, interrompre au bout de quelques secondes en tapant `^C` ; réessayer une seconde fois. Moralité ?

3. Faire en sorte de restaurer le comportement par défaut après le première frappe de `^C`. I.e. on veut qu'une seconde réception de `SIGINT` termine le programme.

   Note : La fonction `getsignal(signalnum)` du module `signal` retourne le gestionnaire de signal associé au signal `signalnum`.

4. Capter le signal `SIGALRM` avec une fonction qui affiche un message et termine le programme. Utiliser la fonction `alarm()` pour que le programme s'arrête au bout de 10 secondes.

   Note : la fonction `alarm(time)` du module `signal` provoque l'envoi d'un signal `SIGALRM` au processus en cours au bout de `time` secondes. Chaque nouvel appel à `alarm` annule et remplace le précédent.

5. Modifier le programme de façon à ce qu'il s'arrête au bout de 5 secondes après la frappe du dernier `^C`.

2 `sleep` vs `pause`

`time.sleep` et `signal.pause` peuvent avoir un rôle équivalent. En fonction de l'implémenation de votre système, une différence existe entre eux, eu-égard aux signaux. Implémenter un programme qui :

— installe un traitant pour *SIGINT*. Ce traitant affiche un simple message,

— le programme principal affiche son PID puis exécute un `time.sleep(30)`.

Tester en envoyant un `SIGINT` au processus.

Faire une seconde version du programme dans laquelle `time.sleep()` est remplacé par `signal.pause()`. Conclusion ?

3 `hup`

Lors de la fermeture d'un terminal, d'une connexion ssh ou d'un déconnexion utilisateur, ses processus sont tués. En effet, la fermeture du terminal, de la connexion ssh et/ou de la déconnexion envoie le signal `SIGHUP` aux processus fils. Il est possible d'immuniser un processus donné à la fermeture du terminal (cf commande UNIX `nohup`) en ignorant ce signal.

— Créer un processus immunisé à `SIGHUP`, à longue durée de vie,

— Fermer le terminal auquel il est rattaché et vérifier qu'il est toujours en vie. Le tuer.

Note : en python, il existe deux traitants prédéfinis : `signal.SIG_IGN` (pour ignorer le signal) et `signal.SIG_DFL` (traitant par défaut). Vous pouvez également utiliser la fonction `getsignal`.

4 `atexit`

Le module `atexit` de python, permet de définir une méthode qui sera appelée automatiquement lors d'un appel à `sys.exit()`. Il permet de fournir un moyen commode d'effectuer un traitement générique lors de la fermeture du programme (comme fermer des fichiers ouverts, libérer des ressources, etc.). À partir de la documentation (`https://docs.python.org/fr/3/library/atexit.html`), implémenter un programme qui :

— lors de la fermeture du programme, affiche un message de sortie par l'intermédiaire du module `atexit`. Ce programme doit proposer deux terminaisons différentes,

— faire en sorte que la même fonction effectuant l'affichage de sortie soit appelée lors de la réception des signaux `SIGQUIT`, `SIGINT`, `SIGABRT`, `SIGTERM`.

Tester.

5 La commande `mytime`

Écrire un programme Python `mytime.py` qui admet la ligne de commande suivante :

```
mytime.py [-n k] [-s] commande [arg ...]
```

Le programme exécute la `commande` avec ses arguments éventuels, attend sa terminaison puis affiche la durée d'exécution de la commande en secondes et microsecondes.

Si l'option `-n` est présente, la même chose est faite `k` fois (`k` > 0) ; de plus la durée moyenne est affichée. Si l'option `-s` est présente, le programme affiche également chaque fois le code de sortie de la commande.

Note : Pour mesurer la durée, on peut se servir de la fonction `time()` du module `time` qui fournit le nombre de secondes et microsecondes écoulées depuis l'Epoch (le 1^e janvier 1970 à 0h) ; il suffit de l'appeler avant et après l'exécution puis de calculer la différence.