TP: Signaux entre processus

1 Envoi de signaux à une application

Q1.1 - Installer le programme `xeyes` et vérifier qu'il démarre.

Q1.2 - Le démarrer en arrière plan, et noter son PID.

Q1.3 - Lui envoyer le signal SIGTERM et vérifier qu'il s'arrête

Q1.4 - Le relancer en arrière plan et lui envoyer le signal SIGSTOP. Qu'observez-vous sur son état ?
Pour le remettre au premier plan, utilisez la commande `$ fg 1`
(fg : "*foreground*", pour mettre le job n° qui tourne en arrière plan au premier plan).

2 Signaux POSIX en Bash

Q2.1 - Créer dans son "home" un dossier `test_signaux` et y descendre.

Q2.2 - Ouvrir deux terminaux dans ce dossier. Dans le 1er, afficher son PID avec la variable `$$`
Dans l'autre, envoyer le signal `SIGINT` au terminal 1. Que pouvez-vous observer dans le terminal 1 ?
Qu'est-ce qui aurait pu provoquer ceci dans le terminal 1 ?

IL effectue un ctrl+C

Q2.3 - Depuis le terminal 2, envoyer le signal `SIGSTOP` à la première. Repasser dans le terminal 1. Que pouvez-vous observer ? (essayer d'exécuter une commande) :

le terminal ne répond plus

Q2.4 - Ouvrez un nouveau terminal 1 et y lancer la commande `sleep` en spécifiant une durée de 10mn. Depuis le terminal 2, essayer de tuer cette attente avec `$ kill -s SIGTERM <PID1>`. Pourquoi ceci ne fonctionne pas ?

car on essaye d'arreter le terminal et non la commande sleep

Q2.5 - Dans le terminal 2, chercher le PID du process dans lequel est lancé la commandee sleep du terminal 1 avec : `$ ps -ef`. Si vous avez du mal à le repérer, ajouter un "pipe" et "grepper" la commande que vous cherchez : (`| grep sleep`).

Puis l'arreter en lui envoyant le signal `SIGTERM`. Ceci fonctionne-t-il ? oui

Q2.6 - La commande `ps` affiche une sélection[1] des process en cours d'exécution ou en attente sur la machine. Quelle est la différence avec `top` (ou `htop`) ?

top affiche les processus linux

Q2.7 - Créer un script `t1.sh` contenant deux lignes : une qui affiche le PID (variable $$) et l'autre qui met le script en sommeil (commande `sleep`) pendant 10 minutes (voir le "man" de sleep pour voir comment spécifier cette durée).

Q2.8 - Ouvrir une autre console dans ce dossier et essayer tuer le 1er script avec la commande `kill`.
Arrivez-vous à l'arreter ? oui

3 Signaux en Python

Q3.1 - Créer un fichier texte `sig.py` dans ~/`test_signaux` contenant le code ci-dessous.

```
x=1
while True:
        print("pid=", os.getpid(), x)
        sleep(.5)
        x += 1
```

1. selon les options choisies

Q3.2 - Ajouter en première ligne un *shebang* pour l'identifier comme un programme Python3, et permettant de l'exécuter directement avec `$ ./sig.py`

Vous obtiendrez le chemin à donner dans le *shebang* avec : `$ which python3`

Q3.3 - Ajouter les imports suivants en tête du fichier :

```
import signal as sig
from time import sleep
import sys
```

Q3.4 - Donner au script les droits d'exécution, puis vérifier qu'il fonctionne bien.

Q3.5 - Que se passe-t-il si vous envoyez depuis le deuxième terminal la commande `$ kill <PID>` (avec "<PID>" le pid du programme affiché dans le terminal 1)

Q3.6 - Ajouter au début du programme l'enregistrement d'un *handler* d'interruption, qui sera appelé automatiquement en cas de réception du signal `SIGINT` :

```
sig.signal(sig.SIGINT, signal_handler)
```

Q3.7 - Ecrire ensuite la définition du *handler* (avant le programme lui-même) :

```
def signal_handler(s, frame):
        print( "réception du signal ", sig.Signals(s).name )
```

Q3.8 - Vérifiez ensuite le fonctionnement : Démarrer le programme et vérifiez que l'envoi d'un "kill" depuis le 2ème terminal est bien détecté par le programme.

Q3.9 - Modifier le programme pour qu'il réponde au cahier des charges suivants :

- La réception du signal `SIGINT` devra être détectée (affichée) et arreter le programme.
- La réception du signal `SIGQUIT` devra être détectée (affichée) mais ne pas arreter le programme.

Voir la page `https://docs.python.org/3/library/signal.html`
Vérifier le fonctionnement.