# TP 1 - Processus



## Création de la Machine virtuelle

Pour ce TP, tu vas utiliser une machine virtuelle en ligne tournant sous LINUX et disposant de PYTHON 3. Celle-ci est hébergée sur le site CoCalc.

Regarder la vidéo sur l'espace commun de travail pour mettre la séance en place.

## Affichage statique des processus

1. Dans le terminal de gauche, taper `ps -aux`.
   Que signifient les colonnes `PID`, `%CPU`, `%MEM` ?

2. La colonne `STAT` indique l'état du processus, il suffit de regarder la première lettre : `R` pour *Running*, `S` pour `Sleeping`, `T` pour `sTopped` par exemple. Dans le terminal de droite, tu vas créer un script PYTHON nommé `inp.py` à l'aide de la commande `nano` :

   - tape `nano inp.py`;
   - dans l'éditeur, tape la seule ligne `a = input('En attente de texte : ')`;
   - enregistre en pressant `CTRL` + `o` puis `Enter`;
   - quitte `nano` en pressant `CTRL` + `x`;

   Exécute ce script en tapant `python inp.py` puis laisse-le tourner.
   Dans le terminal de gauche, affiche la liste des processus.
   Dans quel état se trouve le processus PYTHON ? Pourquoi ?

3. Dans le terminal de droite, crée un script PYTHON appelé `bcl.py` dans lequel il y aura une boucle infinie (sans mettre d'`input`, mais avec un `print`).
   Ensuite exécute-le, puis dans le terminal de gauche, affiche la liste des processus.
   Que remarques-tu ?

4. Pour suspendre l'activité du processus (et arrêter le massacre), dans le terminal de droite presse `CTRL` + `z`.

5. Selon `ps`, dans quel état se trouve le processus ?

6. Pour réactiver le processus, tape `fg` dans le terminal de droite.
   Pour l'arrêter définitivement, tape `CTRL` + `c`.

7. Relance `bcl.py` à droite , puis, dans le terminal de gauche, affiche la liste des processus.
   Pour arrêter définitivement un processus, l'instruction est `kill` suivi du `PID` du processus.
   Arrête `bcl.py`.

## Affichage dynamique des processus et ressources

1. Dans le terminal de gauche, tape `top`.
   Ce que tu obtiens est l'équivalent du gestionnaire des tâches de WINDOWS.

2. Dans le terminal de droite, écris un petit script appelé `bcl2.py` qui calcule

   $$1^4 + 2^4 + \ldots + 20\,000\,000^4$$

   Exécute-le et regarde l'état du processus dans le terminal de gauche. Comment interpréter la consommation CPU ?

3. Modifie ce script

   - en ajoutant en première ligne `from time import sleep`;
   - en allant jusqu'à 10 000 au lieu de 20 000 000;
   - en rajoutant dans la boucle `sleep(0.001)` (dormir 1 milliseconde).

   Exécute ton script et regarde top. Quelles conclusions peux-tu en tirer ?

## Arbre des processus

Tu peux fermer un des deux terminaux.

Comme expliqué dans le cours, un premier processus est d'abord créé par l'OS, puis c'est à partir de celui-ci que d'autres sont créés et ainsi de suite.
Le premier processus est celui qui a le plus petit PID.

1. Quelle commande a lancé le premier processus ?

2. Lorsque `top` est exécuté, on peut, en appuyant sur `f` , ajouter des colonnes et en modifier l'ordre.
   Ajoute la colonne PPID de telle sorte qu'elle soit placée juste après PID.

3. En regardant ces deux colonnes, reconstruis l'arborescence des processus.

4. Tu peux quitter `top` et vérifier ton arborescence avec `pstree` (il manquera `top` mais il y aura un autre processus)...

## Observation de l'ordonnancement

1. Il est possible d'exécuter un processus en tâche de fond pour que le terminal « garde la main» en ajoutant une esperluette à la fin de la commande.
   Tapes `python bcl2.py &`.
   Tu peux exécuter `ps -aux` et constater que le processus est bien là et qu'il finit par se terminer.

2. Enregistre le script suivant sous le nom `proc.py`

   **Python**

   ```python
   from os import getpid

   pid = str(getpid()) # on récupère le n° de processus
   print(f'----------------------------DEBUT PROCESSUS {pid}')
   for i in range(1000):
       print(f'processus {pid} valeur {i}')
   print(f'_____________________________FIN PROCESSUS {pid}')
   ```

   Exécute-le pour constater qu'il fonctionne.

3. Exécute la commande `python proc.py & python proc.py & python proc.py &` pour en lancer 3 instances.
   Recommence plusieurs fois (tu peux effacer l'écran avec `clear`).

   Les 3 processus sont-ils exécutés à la suite les uns des autres ou bien y a-t-il chevauchement ?

   L'exécution simultanée des 3 processus se passe-t-elle de manière prévisible (on dit aussi déterministe) ?