OS et processus activités

Ouvrer le terminal en ligne : https://bellard.org/jslinux/

CPU	OS	User Interface	VFsync access	Startup Link	TEMU Config	
x86	Alpine Linux 3.12.0	Console	Yes	click here	url	

0. Renseignement

Dans le terminal taper :

- Type de CPU "lscpu"

  localhost# lscpu

  Indiquer quel est le type de microprocesseur qui est utilisé.

- type d'OS

  localhost# cat /proc/version (cat sort sur le terminal le contenu d'un fichier, sous linux tout est fichier)

Un peu de python

Lancer le shel de python3 dans le terminal

localhost#python3

```
localhost:~# python3
Python 3.8.3 (default, May 15 2020, 01:53:50)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Derrière le prompt de python >>> on peut lancer du code
Taper

```
>>> import os
>>> os.name
```

Puis

```
>>> os.uname()
```

```
>>> list(os.environ.keys())
```

Pour sortir du Shell python taper ^d (^ :CTRL)

OS et processus activités

1. Les fichiers

1.1. Utilisation des commandes en lignes

- Création d'un nouveau répertoire "Travail" : Votre_login# mkdir Travail
- Listage des différents éléments du répertoire courant : Commande ls (lister)

Votre_login# ls

Votre_login# ls -al

- Passage au répertoire enfant

Votre_login# cd travail

- Retour au répertoire parent

Votre_login# cd ..

- Aller au répertoire utilsateur (personne qui est loguée)

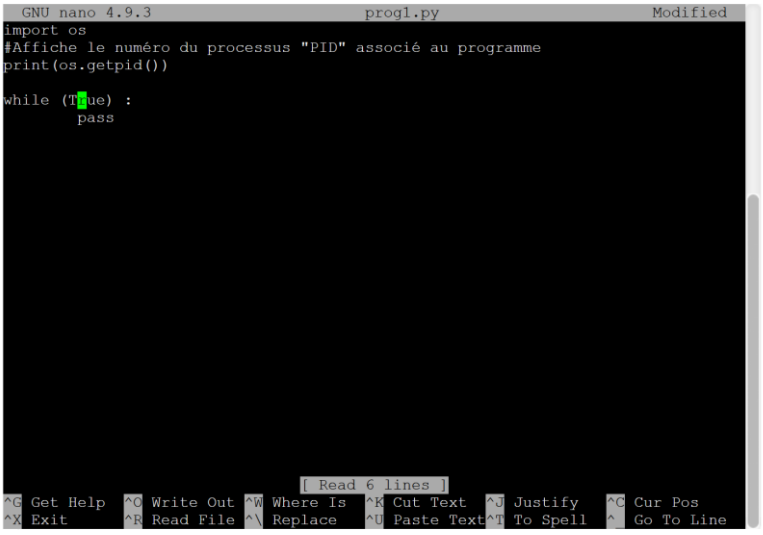
Votre_login# cd ~

1.2. Création des programmes

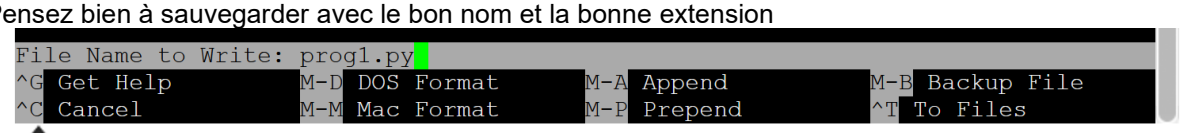
On utilise nano pour éditer des fichiers tapez :

```
localhost:~# nano
```

Puis



^X pour sortir.
Pensez bien à sauvegarder avec le bon nom et la bonne extension



Créer 4 suite 4 programmes identiques à prog1.y (prog1.py, prog2.py, prog3.py et prog 4.py) dans Travail.

OS et processus activités

Faites un "ls -al" à la racine

Indiquer les la signification des "d" "x" "r" "w"

https://www.linuxtricks.fr/wiki/droits-sous-linux-utilisateurs-groupes-permissions

Changement des droits pour les rendre accessibles en écriture, lecture et exécution à tous les utilisateurs https://www.leshirondellesdunet.com/chmod-et-chown

On désire :

```
drwxr-xr-x    2 root     root            141 Feb  1 11:54 .
drwxr-xr-x    6 root     root            315 Jan  9  2021 ..
-rwxrwxrwx    1 root     root            109 Feb  1 11:58 prog_1.py
-rwxrwxrwx    1 root     root            110 Feb  1 12:12 prog_2.py
-rwxrwxrwx    1 root     root            110 Feb  1 12:12 prog_3.py
-rwxrwxrwx    1 root     root            110 Feb  1 12:12 prog_4.py
localhost:~/Travail#
```

2. OS et processus

On lance prog_1.py mais en arrière-plan pour garder la main (& en fin)

```
localhost:~/Travail# python3 prog_1.py &
localhost:~/Travail# 204
                                  entrée

localhost:~/Travail#
```

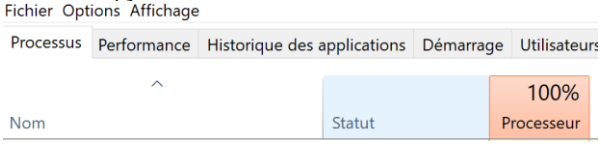
Si on regarde les processus tournant sur la machine :

Commande top (The top program provides a dynamic real-time view of a running system. It can display system summary information as well as a list of processes or threads currently being managed by the Linux kernel.)

Exemple sur un terminal Linux installé sur W10

```
boss@DESKTOP-GC7TQ2B:~/Travail$ top -n 1 -b
top - 11:40:05 up 42 min,  0 users,  load average: 0.52, 0.58, 0.59
Tasks:   8 total,   5 running,   3 sleeping,   0 stopped,   0 zombie
%Cpu(s): 98.3 us,  1.7 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   8127.3 total,   3085.2 free,   4818.1 used,    224.0 buff/cache
MiB Swap:  24576.0 total,  24551.1 free,     24.9 used.   3178.6 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  291 boss      20   0   23864   6452   3076 R 100.0   0.1   2:25.38 python3
  295 boss      20   0   23864   6452   3076 R 100.0   0.1   0:11.17 python3
  294 boss      20   0   23864   6452   3076 R 100.0   0.1   0:17.18 python3
  296 boss      20   0   23864   6452   3076 R 100.0   0.1   0:06.75 python3
    1 root      20   0    8936    316    272 S   0.0   0.0
    7 root      20   0    8944    232    180 S   0.0   0.0
    8 boss      20   0   18360   4004   3884 S   0.0   0.0
  297 boss      20   0   18788   2012   1452 R   0.0   0.0
```



OS et processus activités

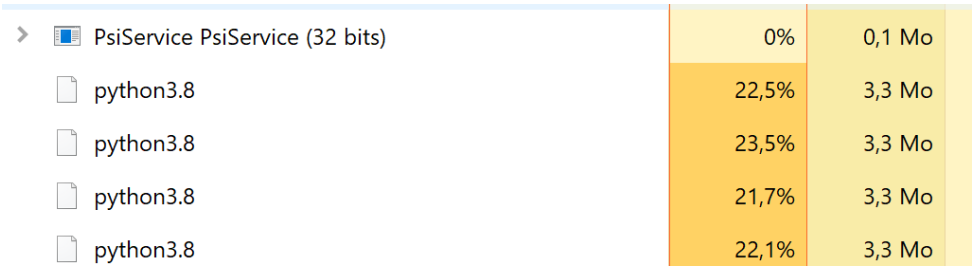
Sur la machine virtuelle en ligne

```
Mem: 16100K used, 171144K free, 8K shrd, 0K buff, 2616K cached
CPU:  94% usr   0% sys   0% nic   0% idle   0% io   0% irq   5% sirq
Load average: 1.56 0.64 0.25 5/35 226
  PID  PPID USER     STAT   VSZ %VSZ CPU %CPU COMMAND
  224    61 root     R     5472   3%   0  24% python3 prog_3.py
  225    61 root     R     5180   3%   0  24% python3 prog_4.py
  204    61 root     R     5472   3%   0  18% python3 prog_1.py
  223    61 root     R     5472   3%   0  18% python3 prog_2.py
  226    61 root     R     1516   1%   0  12% top -n 1 -b
   61     1 root     S     1552   1%   0   0% sh -l
    1     0 root     S     1512   1%   0   0% {init} /bin/sh /sbin/init
   56     1 root     S     1260   1%   0   0% dhcpcd -q
   55     1 root     S      744   0%   0   0% settime -d /
    7     2 root     SW       0   0%   0   0% [ksoftirqd/0]
```

Repérer sur votre machine les PID des programmes python

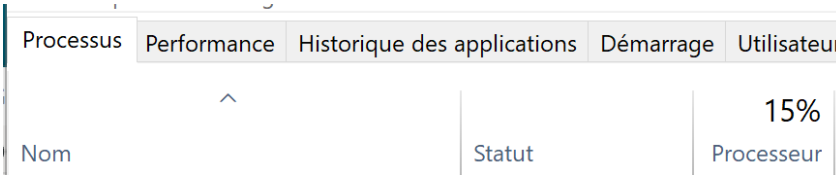
Commande ps (ps displays information about a selection of the active processes.)

```
boss@DESKTOP-GC7TQ2B:~/Travail$ ps -a -u -x
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   8936   316 ?        Ssl  10:57   0:00 /init
root         7  0.0  0.0   8944   232 tty1     Ss   10:57   0:00 /init
boss         8  0.0  0.0  18360  4004 tty1     S    10:57   0:00 -bash
boss       291 93.5  0.0  23864  6452 tty1     R    11:37   5:57 python3 prog1.py
boss       294 90.7  0.0  23864  6452 tty1     R    11:39   3:48 python3 prog2.py
boss       295 90.4  0.0  23864  6452 tty1     R    11:39   3:42 python3 prog3.py
boss       296 90.5  0.0  23864  6452 tty1     R    11:39   3:38 python3 prog4.py
boss       342  0.0  0.0  18904  2044 tty1     R    11:43   0:00 ps -a -u -x
```



Commande kill (pour tuer les processus)

```
boss@DESKTOP-GC7TQ2B:~/Travail$ kill 384
boss@DESKTOP-GC7TQ2B:~/Travail$ kill 291
[8]   Terminated              python3 prog5.py
boss@DESKTOP-GC7TQ2B:~/Travail$ kill 294
boss@DESKTOP-GC7TQ2B:~/Travail$ kill295
kill295: command not found
[1]   Terminated              python3 prog1.py
[2]   Terminated              python3 prog2.py
boss@DESKTOP-GC7TQ2B:~/Travail$ kill 295
[3]   Terminated              python3 prog3.py
boss@DESKTOP-GC7TQ2B:~/Travail$ kill 296
[4]   Terminated              python3 prog4.py
boss@DESKTOP-GC7TQ2B:~/Travail$ ps
  PID TTY          TIME CMD
    8 tty1     00:00:00 bash
  345 tty1     00:00:01 geany
  361 tty1     00:00:00 dbus-launch
  399 tty1     00:00:00 ps
```



Sur votre machine virtuelle en ligne "killer" les processus liés aux programmes python et comparer la charge du processeur

Résumer des instructions :

sudo	
chmod	
ls	
ps	
top	
python3 nom_programme. py &	
kill processus_sa	
lscpu	

3. Les Threads – processus légers programmation concurrente

Les Thread en Python s'utilisent « très facilement ». Pour cela, il suffit de déclarer une instance par le constructeur **threading.Thread(group=None, target=None, name=None, args=(), kwargs={})** où :

- **group** doit rester à None, en attendant que la classe **ThreadGroup** soit implantée.
- **target** est la fonction appelée par le Thread.
- **name** est le nom du Thread.
- **args** est un tuple d'arguments pour l'invocation de la fonction target
- **kwargs** est un dictionnaire d'arguments pour l'invocation de la fonction target

il suffit ensuite pour exécuter le Thread d'appliquer la méthode **start()**

3.1. Les Threads sans coordination

```
import threading
import time
import random
import os

panier = 50

def Prendre_une_pomme(n) :
    """
    Prendre_une_pomme sera appelé par un thread
    n sert à numéroter les Threads
    """
    print(f"Début du Thread {n}")
    global panier
    temp = panier
    time.sleep(random.randint(1,100)/10)
    pommes = random.randint(1,4)
    panier = temp-pommes
    print(f"Je suis le Thread  {n} j'appartient au processus {os.getpid()}  \
        je prends {pommes} pommes, il reste { panier } pommes")
    print(f"Fin de la prise du Thread {n}")

if __name__ == '__main__' :
    """
    On lance trois threads.
    """
    t1 = threading.Thread(target=Prendre_une_pomme, args=[1])
    t2 = threading.Thread(target=Prendre_une_pomme, args=[2])
    t3 = threading.Thread(target=Prendre_une_pomme, args=[3])
    t1.start()
    t2.start()
    t3.start()
```

- Faites tourner le programme plusieurs fois.
- Remarquer l'ordre de fin d'exécution des threads
- L'identifiant du processus.
- Afficher la valeur de 'panier" en fin de programme et comparer avec la somme des nombres de pommes retirée par les threads

3.2. Les Threads avec coordination

Pour éviter le mélange des traitements on va utiliser un système de verrou (mutex)

La zone qui peut être fragmentée par l'ordre anarchique de déroulement des threads est l'exécution de la fonction « target ». Cette portion de code est appelée **zône critique dans la littérature.**

```
#On crée un verrou en instanciant la class Lock() de threadin
verrou = threading.Lock()

def Prendre_une_pomme(n) :
    """
    Prendre_une_pomme sera appelé par un thread
    n sert à numéroter les Threads
    """
    #Quand un thread veut agir il demande l'autorisation en vérifiant l'état
du verrou
    verrou.acquire()
    #Acces à la zône critique
    global panier
    temp = panier
    print(f"Début du Thread {n}")
    time.sleep(random.randint(1,100)/10)
    pommes = random.randint(1,4)
    panier = temp-pommes
    print(f"Je suis le Thread  {n} j'appartient au processus {os.getpid()} \
     je prends {pommes} pommes, il reste { panier } pommes")
    print(f"Fin de la prise du Thread {n}")
    #Fin de la zône critique on relâche le verrou
    verrou.release()
    #ce qui permet le traitement qui sera effectué par le prochain Thread
```

- Faites tourner le programme plusieurs fois.
- Remarquer l'ordre de fin d'exécution des threads
- L'identifiant du processus.
- Afficher la valeur de 'panier" en fin de programme et comparer avec la somme des nombres de pommes retirée par les threads

A retenir pour plus tard

La coordination des threads demande l'utilisation de verrous (mutex en C++) ou de sémaphore (le nombre de threads pouvant accéder simultanément à la même ressource) .

3.3. Les Threads situation d'interblocage

La situation d'interblocage ne peut pas arriver dans le scénario précédent car nous n'avons qu'un verrou donc une seule ressource convoitée par 3 threads : celle-ci va passer de threads en threads sans conflits. Pour faire apparaître l'interblocage, nous avons besoin de plusieurs verrous afin de créer une attente circulaire.
Nous allons donc créer 3 (nb_thread) verrous : V0 V1 V2 et V3. La fonction **Prendre_une_pomme2** va utiliser deux verrous.
Le Thread 0 va donc réserver V0 puis V1
Le Thread 1 va donc réserver V1 puis V2
Le Thread 2 va donc réserver V2 puis V0

```
import threading
import time
import random
import os

panier = 50
#On crée des  verrous en instanciant la class Lock() de thread
nb_thread = 3
verrous = [threading.Lock() for i in range(nb_thread)]

def Prendre_une_pomme2(n,verrou1,verrou2) :
    """
    Prendre_une_pomme sera appelé par un thread
    n sert à numéroter les Threads
    """
    global panier

    #Quand un thread veut agir il demande l'autorisation en vérifiant l'état
du verrou
    #Acces à la zône critique1
    verrou1.acquire()
    temp = panier
    #Acces à la zône critique2
    verrou2.acquire()
    print(f"Début du Thread {n}")
    time.sleep(0.000000001)
    # time.sleep(random.randint(1,100)/10)
    pommes = random.randint(1,4)
    panier = temp-pommes
    print(f"Je suis le Thread  {n} j'appartient au processus {os.getpid()} \
         je prends {pommes} pommes, il reste { panier } pommes")
    print(f"Fin de la prise du Thread {n}")
    #Fin de la zône critique2 on relâche le verrou
    verrou2.release()
    #Fin de la zône critique1 on relâche le verrou
    verrou1.release()
    #ce qui permet le traitement qui sera effectué par le prochain Thread

if __name__ == '__main__' :
    """
    On lance n threads.
    """
    mes_threads = []
    for i in range(nb_thread):
         t= threading.Thread(target=Prendre_une_pomme2, \
             args=[i,verrous[i],verrous[(i+1)%nb_thread]])
         t.start()
         mes_threads.append(t)
```

- Faites tourner le programme plusieurs fois.
- Remarquer l'ordre de fin d'exécution des threads
- L'identifiant du processus.
- Afficher la valeur de 'panier" en fin de programme et comparer avec la somme des nombres de pommes retirée par les threads

Que pouvez vous dire par rapport aux deux situations précédentes ?

4. Les Threads – et les serveurs

Allez sur le site :
https://www.geeksforgeeks.org/socket-programming-multi-threading-python/

essayer de faire tourner le serveur et le client sur votre machine

Bibliographie :
livre TNSI

Olivier Lecluse

http://www.xavierdupre.fr/app/teachpyx/helpsphinx/c_parallelisation/thread.html