

TP processus- Linux

Présentation

Vous allez utiliser une « émulation » de Linux fonctionnant dans votre navigateur.

Allez à l'adresse suivante : <https://bellard.org/jslinux/> et sélectionnez **Fedora (linux) X window** → [click here](#)

Cela va ouvrir un environnement Linux. Avant de démarrer le TP, un petit réglage est à faire.

Faites un clic droit dans la fenêtre, dans le menu sélectionnez **Keyboard mapping** puis **French**

Prise en main de l'environnement

Pour ouvrir un terminal, faites un clic droit puis cliquez sur **Terminal**. Vous pouvez ouvrir plusieurs terminaux.

Dans le terminal, entrez la commande : `ps`

Décrivez l'affichage obtenu

PID, PPID

Un processus est caractérisé par un identifiant unique : son **PID** (Process Identifier). Lorsqu'un processus engendre un fils, l'OS génère un nouveau numéro de processus pour le fils. Le fils connaît aussi le numéro de son père : le **PPID** (Parent Process Identifier).

Dans le terminal, entrez la commande : `ps -ef`

1. Quel est le PID du processus **init** ?
2. Quel est le PPID de **init** ?
3. **init** possède t-il un frère ?
4. Citer quelques descendants directs de **init**

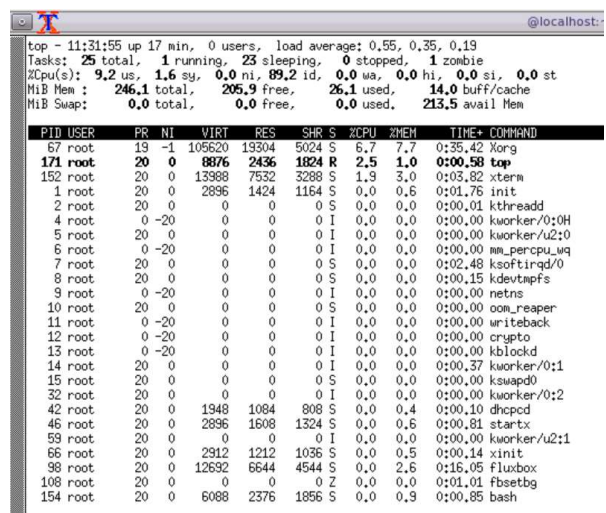
Une commande indispensable à connaître sous Linux pour inspecter les processus est la commande `top`.

Lancez cette commande dans un terminal. Vous devriez avoir quelque chose de ce style

L'affichage se rafraîchit en temps réel contrairement à `ps` qui fait un instantané. L'application est plus riche qu'il n'y paraît. Il faut passer un peu de temps à explorer toutes les options.

Celles-ci s'activent par des raccourcis clavier. En voici quelques uns :

- `h` : affiche l'aide



```
top - 11:31:55 up 17 min, 0 users, load average: 0.55, 0.35, 0.19
Tasks: 25 total, 1 running, 23 sleeping, 0 stopped, 1 zombie
%Cpu(s): 9.2 us, 1.6 sy, 0.0 ni, 89.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 246.1 total, 205.9 free, 26.1 used, 14.0 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used, 213.5 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
67	root	19	-1	105620	19304	5024	S	6.7	7.7	0:35.42	Xorg
171	root	20	0	8876	2436	1824	R	2.5	1.0	0:00.58	top
152	root	20	0	13988	7532	3288	S	1.9	3.0	0:03.82	xterm
1	root	20	0	2896	1424	1164	S	0.0	0.6	0:01.76	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthreadd
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworke/0:0H
5	root	20	0	0	0	0	I	0.0	0.0	0:00.00	kworke/u2:0
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
7	root	20	0	0	0	0	S	0.0	0.0	0:02.48	ksoftirqd/0
8	root	20	0	0	0	0	S	0.0	0.0	0:00.15	kdevtmpfs
9	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	netns
10	root	20	0	0	0	0	S	0.0	0.0	0:00.00	oom_reaper
11	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	writeback
12	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	crypto
13	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kblockd
14	root	20	0	0	0	0	I	0.0	0.0	0:00.37	kworke/0:1
15	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kswapd0
32	root	20	0	0	0	0	I	0.0	0.0	0:00.00	kworke/0:2
42	root	20	0	1948	1084	808	S	0.0	0.4	0:00.10	dhcpcd
46	root	20	0	2896	1608	1324	S	0.0	0.6	0:00.81	startx
59	root	20	0	0	0	0	I	0.0	0.0	0:00.00	kworke/u2:1
66	root	20	0	2912	1212	1036	S	0.0	0.5	0:00.14	xinit
98	root	20	0	12632	6644	4544	S	0.0	2.6	0:16.05	fluxbox
108	root	20	0	0	0	0	Z	0.0	0.0	0:01.01	fbsetbg
154	root	20	0	6088	2376	1856	S	0.0	0.9	0:00.85	bash

TP processus- Linux

- M : trie la liste par ordre décroissant d'occupation mémoire. Pratique pour repérer les processus trop gourmands
- P : trie la liste par ordre décroissant d'occupation processeur
- i : filtre les processus inactifs. Cela ne montre que ceux qui travaillent réellement.
- k : permet de tuer un processus - à condition d'en être le propriétaire. Essayez de tuer init ...
- V : permet d'avoir la vue arborescente sur les processus.
- q : permet de quitter `top`

Repérez le *PID* de `top` puis tuez-le à l'aide de la commande

`kill`. Que se passe-t-il après que vous ayez arrêté `top` ?

Tuer un processus

Pour tuer un processus, on lui envoie un *signal* de terminaison. On en utilise principalement 2 :

- SIGTERM (15) : demande la terminaison d'un processus. Cela permet au processus de se terminer proprement en libérant les ressources allouées.
- SIGKILL (9) : demande la terminaison immédiate et inconditionnelle d'un processus. C'est une terminaison violente à n'appliquer que sur les processus récalcitrants qui ne répondent pas au signal SIGTERM.

Pour terminer `top` proprement, vous lui enverrez donc un signal SIGTERM en tapant le numéro 15. Cela est équivalent à la commande shell `kill -15` où *PID* désigne le numéro du processus à quitter proprement. Si ce dernier est planté et ne réagit pas à ce signal, alors vous pouvez vous en débarrasser en tapant `kill -9 PID`.

1. Lancez le navigateur Dillo (clic droit sur le bureau)
2. Repérez son *PID* à l'aide de la commande `ps` ou `top`, notez le ci-dessous
3. Depuis un terminal, terminez l'application en utilisant la commande `kill`. Notez la commande utilisée ci-dessous.

Priorités

Sous Linux, on peut passer des consignes à l'ordonnanceur en fixant des priorités aux processus dont on est propriétaire : Cette priorité est un nombre entre -20 (plus prioritaire) et +20 (moins prioritaire).

On peut agir à 2 niveaux :

- fixer une priorité à une nouvelle tâche **dès son démarrage** avec la commande `nice`
- modifier la priorité d'un processus **en cours d'exécution** grâce à la commande `renice`

les colonne **PR** et **NI** de la commande `top` montrent le niveau de priorité de chaque processus

Le lien entre **PR** et **NI** est simple : **PR = NI + 20** ce qui fait qu'une priorité **PR** de 0 équivaut à un niveau de priorité maximal.

Exemple : Pour baisser la priorité du process `terminator` dont le *PID* est 21523, il suffit de taper `renice +10 21523`

TP processus- Linux

A vous de jouer

Nous allons tester l'efficacité du paramètre *nice* de l'ordonnanceur sur le temps d'exécution d'un programme python. Pour cela, nous allons charger le processeur de la machine à fond et chronométrer le temps d'exécution d'un script python pour plusieurs valeur du paramètre *nice*.

Pour cet exercice, n'hésitez pas à ouvrir plusieurs fenêtres de terminal côte à côte.

1. Ouvrez une console et lancez la commande `top`.
2. créer un programme python nommé **infini.py** contenant une boucle infinie .
Pour cela `nano infini.py` ouvre un éditeur de texte dans la console.
Une fois votre code terminé, appuyez sur CTRL+X et sauvegardez votre travail.
L'indentation se fait à la main (4 espaces).

3. créer un second programme **test.py** contenant

```
def bidon():  
    a = 0  
    for i in range(1000):  
        a += a**3
```

4. lancer un interpréteur python3 et noter son numéro de processus

Depuis une nouvelle console : `python3`

5. dans l'interpréteur python, tapez les commandes

```
>>> from timeit import timeit  
>>> import test  
>>> timeit(test.bidon, number = 100)
```

cette commande va lancer 100 fois la fonction `bidon` et renvoyer le temps d'exécution moyen.

Noter ici le temps d'exécution obtenu :

6. Depuis une nouvelle console, taper la commande `python3 infini.py &` le symbole `&` indique au shell de lancer le programme en arrière plan. Nous allons donc monopoliser l'ensemble des ressources processeurs de la machine avec des boucles infinies. Le travail de l'ordonnanceur sera donc bien visible car les ressources processeur vont se raréfier.
7. Relancer `timeit(test.bidon, number = 100)` dans le shell python. Vous devriez noter un ralentissement par rapport à la première exécution. En effet, le processeur a moins de temps à consacrer à l'exécution de la fonction `bidon`. Noter ici le temps d'exécution obtenu :
8. Changer la priorité de l'interpréteur python en mettant un *nice* à **+10**.
9. Relancer `timeit(test.bidon, number = 100)` dans le shell python. Que constatez-vous ?