



# Les processus Linux

Gestion des processus sur Linux

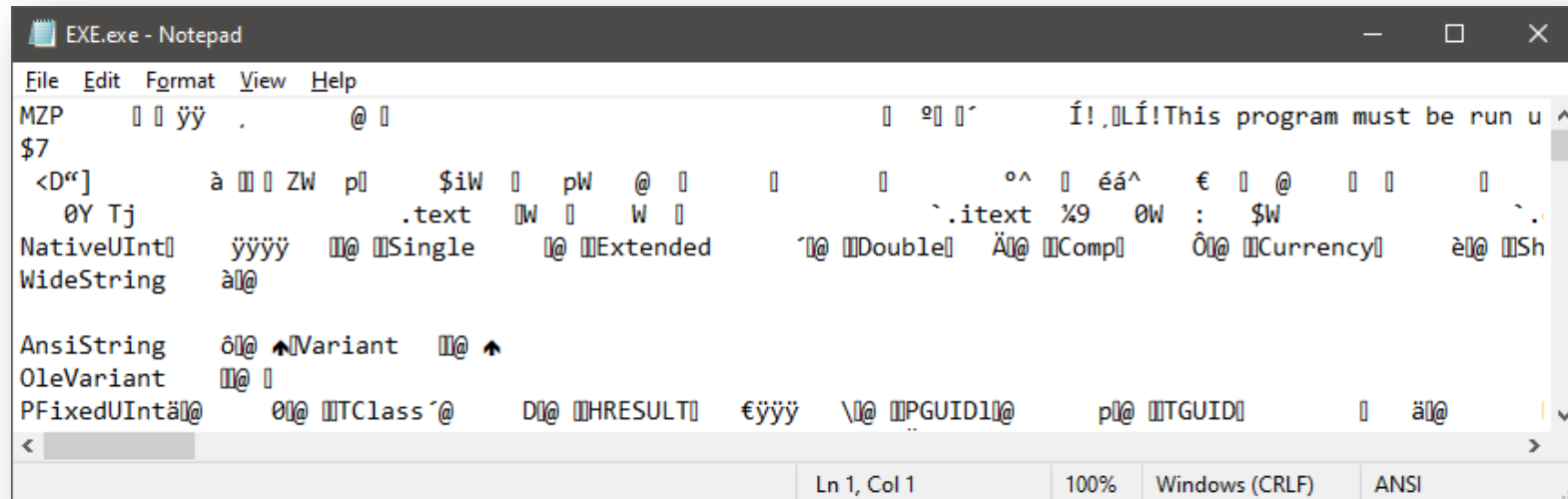
Automne 2022

Séance 06A



- ✓ Différence entre un fichier exécutable et un processus
- ✓ Qu'est-ce qu'un processus?
- ✓ Les processus sous Linux:
  - ✓ Liste des processus
  - ✓ Filtrer une liste dans un terminal
  - ✓ La commande top
  - ✓ Arrêter un processus par son PID
  - ✓ Arrêter un processus par son nom
  - ✓ Arrêter un processus d'un autre utilisateur
  - ✓ Processus en arrière-plan
  - ✓ Commandes de gestion de processus
  - ✓ Autres outils

Les fichiers exécutables contiennent des **instructions** pour le processeur, peu lisibles pour les humains.



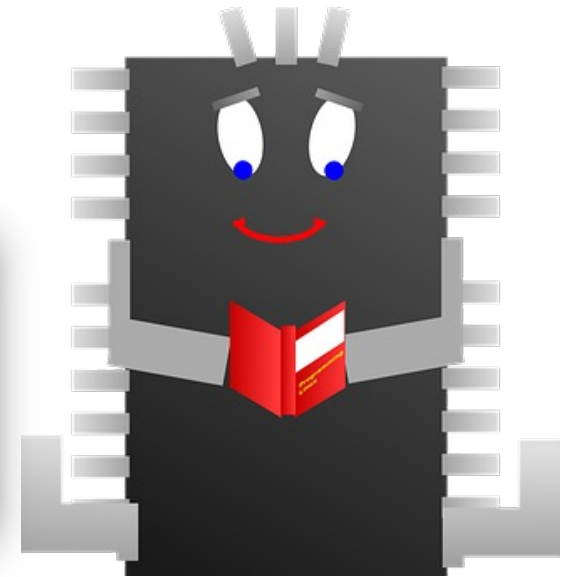
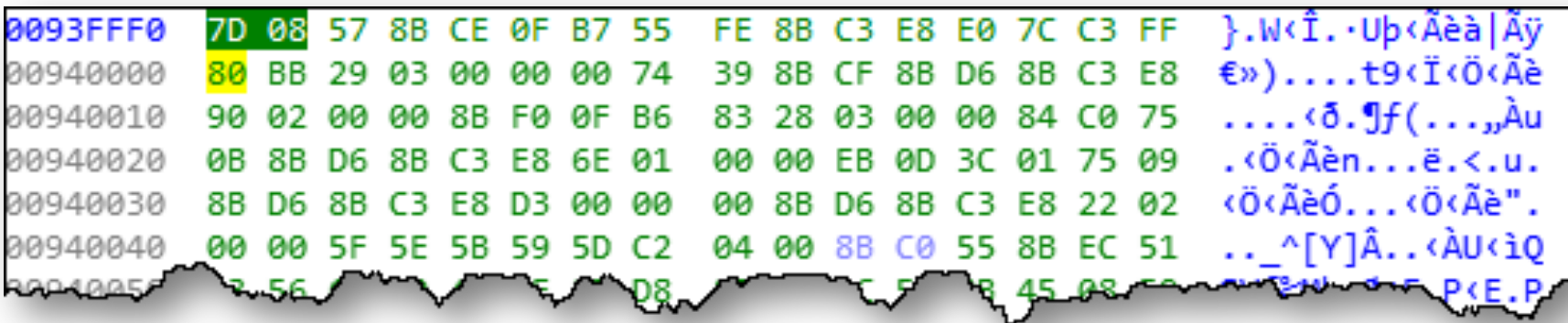


# Programmes exécutables

Lorsqu'un programme (par exemple, fait en C++) est **compilé**, il devient un ensemble de valeurs binaires parfaitement compréhensibles par la machine. On appelle cela le **langage machine** ou **programme exécutable** tout simplement.

Le langage machine est en fait un ensemble d'instruction que le processeur peut directement exécuter.

Certains éditeurs de fichiers peuvent améliorer la lisibilité du code binaire...



# Systèmes numériques



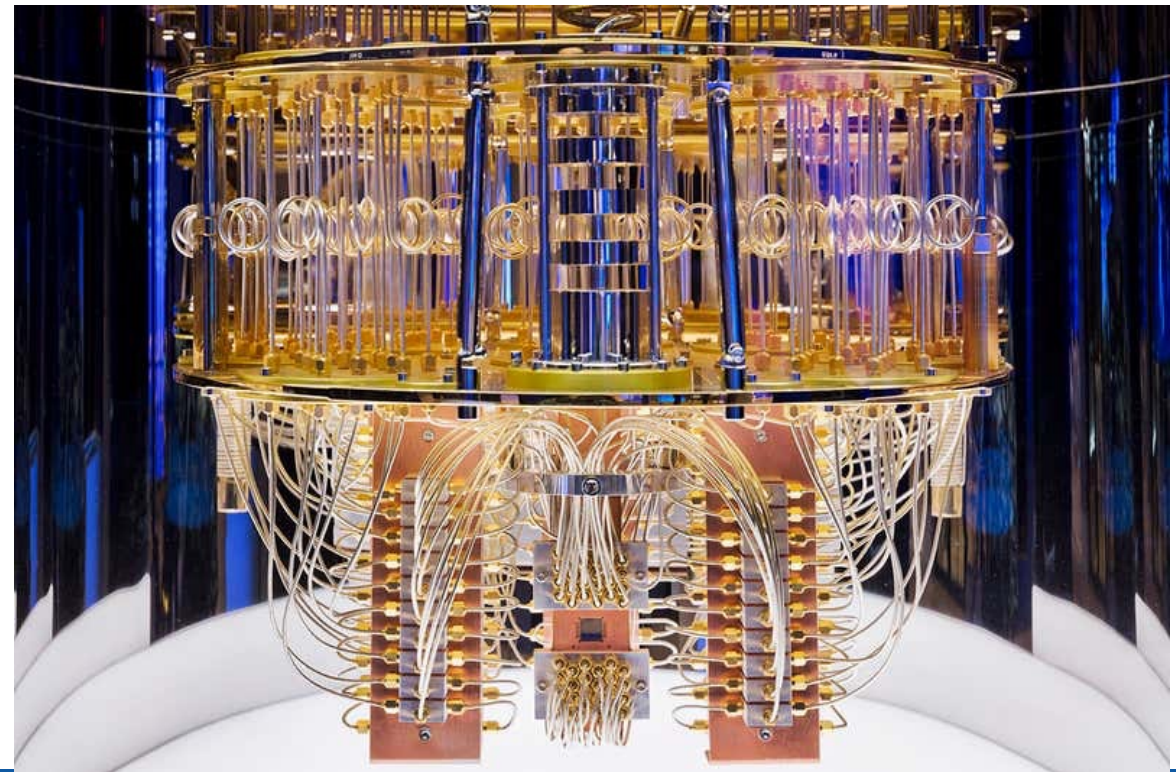
Le langage machine est en binaire car basé sur un signal électrique : le courant passe (1) ou ne passe pas (0). L'humain utilise le système décimal en base 10 (0-9), l'ordinateur utilise le binaire en base 2 (0-1).

Il existe aussi l'hexadécimal en base 16 (0-9, A-F).

L'ordinateur quantique promet une infinité de valeurs car basé sur des probabilités de valeurs entre 0 et 1 !

C'est un domaine de recherche de pointe très actif actuellement car les possibilités d'un tel ordinateur sont prodigieuses...

... ce sujet dépasse cependant les objectifs de ce cours !





# Programme exécutable vs. processus

Le programme exécutable n'est qu'un ensemble d'instructions contenues dans un **fichier**. Les instructions qui s'y trouvent n'ont aucun effet; c'est un fichier comme un autre.

Pour que le processeur puisse exécuter ces instructions, le programme doit être chargé dans la **mémoire vive**.

Quand l'utilisateur exécute ce fichier, un **processus** est créé par le système d'exploitation (Windows/Linux/autres), et alloue une partie des ressources du système (mémoire, temps de processeur).

L'organisation des programmes en processus est pratique pour le SE car nos besoins modernes exigent l'usage (exécution) de plusieurs applications (programmes) à la fois.

# Le processus



Le processus est un conteneur qui a pour but de fournir un **environnement** d'exécution et des **ressources** système pour qu'un programme puisse être exécuté par le processeur.

Tous les programmes ont besoin d'un processus, même les composants internes du système d'exploitation.

Chaque programme est encapsulé dans son processus, ce qui permet de l'isoler des autres programmes en cours d'exécution. Le système d'exploitation gère l'accès aux ressources du système de sorte qu'un programme ne puisse pas nuire aux autres.



# Processus et multitâche

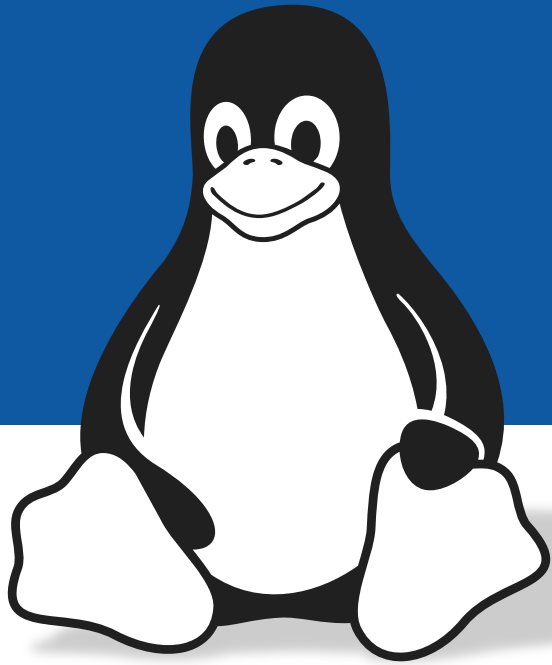


Les anciens ordinateurs ne pouvaient pas exécuter plusieurs programmes en même temps. Lorsqu'un programme était exécuté, il s'accaparait toutes les ressources du système. Lorsqu'il plantait, l'ordinateur au complet plantait.

Ensuite, les systèmes d'exploitation ont introduit le **multitâche coopératif**. Les processus permettaient de partager des ressources entre les programmes, mais si un programme devenait hors de contrôle et consommait toutes les ressources, les autres programmes étaient inutilisables.

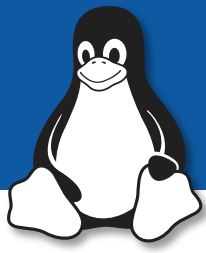
De nos jours, les systèmes d'exploitation offrent du **multitâche préemptif**. Le noyau du système est donc un chef d'orchestre qui attribue les ressources aux processus en les régulant pour les empêcher d'interférer sur les autres programmes, ce qui améliore grandement leur stabilité.





# Linux

# Liste des processus



La commande pour obtenir la liste des processus sous Linux est **ps**.

Plusieurs commutateurs sont offerts.  
Pour voir tous les processus, on peut utiliser la commande: **ps -aux**

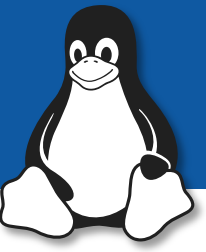
<b>a</b>	Montre les processus de tous les utilisateurs
<b>u</b>	Montre le propriétaire
<b>x</b>	Montre les processus d'arrière-plan

```
etudiant@etudiant-virtual-machine:~$ ps -aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.4  0.2 159836   9100 ?        Ss   09:13   0:02 /sbin/init splash
root         2  0.0  0.0      0      0 ?        S    09:13   0:00 [kthreadd]
root         3  0.0  0.0      0      0 ?        I<   09:13   0:00 [rcu_gp]
root         4  0.0  0.0      0      0 ?        I<   09:13   0:00 [rcu_par_gp]
...
etudiant  2039  0.2  0.1  51560  4284 pts/1    S+   09:18   0:00 top
root      2048  0.0  0.0  80964  3816 tty3     Ss   09:19   0:00 /bin/login -p --
root      2049  0.0  0.0      0      0 ?        R    09:19   0:00 [kworker/u256:0+]
etudiant  2140  0.0  0.1  29708  5028 tty3     S    09:19   0:00 -bash
etudiant  2157  0.0  0.2 239056 11032 tty3     Sl+  09:19   0:00 nmtui
etudiant  2185  0.0  0.0  46856  3660 pts/0    R+   09:21   0:00 ps -aux
etudiant@etudiant-virtual-machine:~$
```

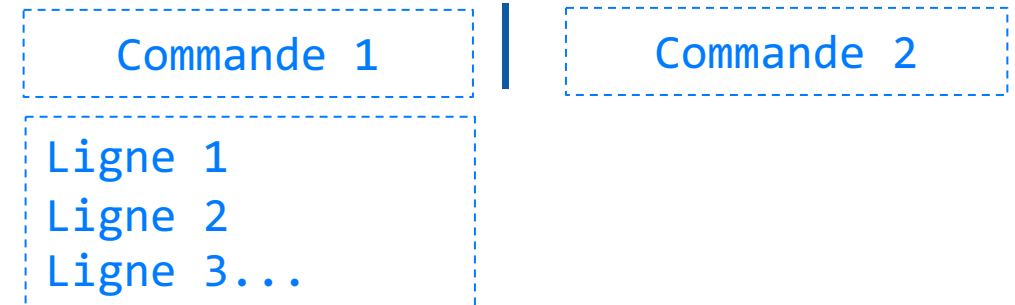
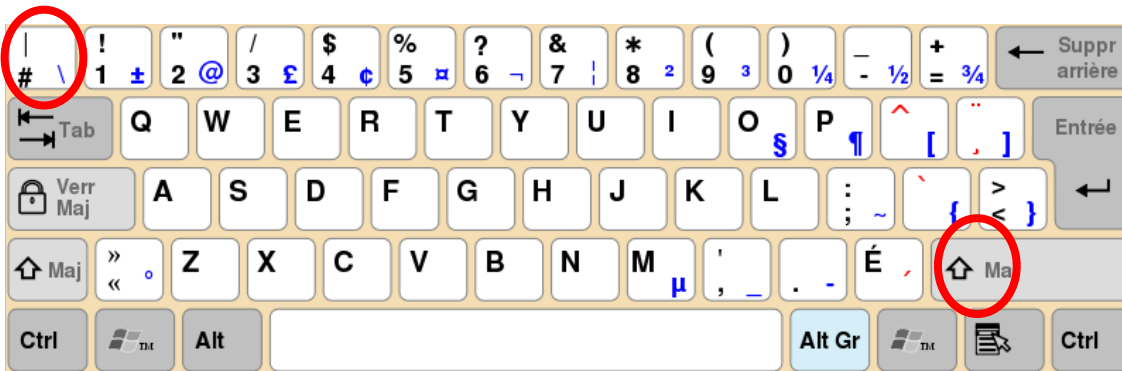
Le numéro unique  
du processus

La commande qui a  
démarré le processus

# Filtrer une liste dans un terminal

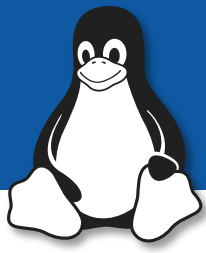


On peut utiliser la commande **grep** pour filtrer le texte retourné par une commande, ici **ps**. On utilise un procédé nommé **pipeline**, qui se caractérise par le caractère « **|** », appelé le *tube*, ou *pipe* en anglais.



```
etudiant@etudiant-virtual-machine:~$ ps -aux | grep nmtui
etudiant    2157  0.0  0.2 239056 11032 tty3      Sl+   09:19   0:00 nmtui
etudiant    2225  0.0  0.0  21532  1004 pts/0    R+    09:28   0:00 grep --color=auto nmtui
etudiant@etudiant-virtual-machine:~$
```

# La commande top



L'outil **top** est l'équivalent du gestionnaire de tâches de Windows.

Il affiche en temps réel les processus en cours, triés par leur utilisation du processeur.

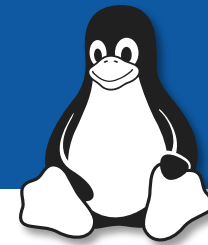
Raccourcis clavier:

- > Flèches du clavier pour se déplacer
- > « **h** » pour de l'aide
- > « **q** » pour quitter

```
top - 12:12:43 up 6 days, 1:29, 1 user, load average: 0.00, 0.00, 0.00
Tâches: 279 total, 3 en cours, 276 en veille, 0 arrêté, 0 zombie
%Cpu(s): 2.0 ut, 3.9 sy, 0.0 ni, 94.1 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 3908.4 total, 1182.9 libr, 1085.7 util, 1639.8 tamp/cache
MiB Éch: 2048.0 total, 2048.0 libr, 0.0 util. 2557.2 dispo Mem
```

PID	UTIL.	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TEMPS+	COM.
15721	etudiant	20	0	4016624	251836	100080	R	6.6	6.3	2:34.33	gnome-s+
15483	etudiant	20	0	276016	60040	34944	S	2.7	1.5	0:16.47	Xorg
16027	etudiant	20	0	973840	54476	41060	S	0.7	1.4	0:04.95	gnome-t+
43085	etudiant	20	0	20740	4436	3620	R	0.7	0.1	0:00.04	top
390	root	-51	0	0	0	0	S	0.3	0.0	0:01.74	irq/16+
15420	etudiant	9	-11	2015296	19360	15316	S	0.3	0.5	0:00.68	pulseau+
15650	etudiant	20	0	471692	11016	9744	S	0.3	0.3	0:08.24	xdg-des+
43044	root	20	0	0	0	0	I	0.3	0.0	0:00.03	kworker+
1	root	20	0	168984	12912	8336	S	0.0	0.3	0:19.03	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.18	kthreadd

# Arrêter un processus par son PID



Pour mettre fin à un processus, on peut utiliser la commande:

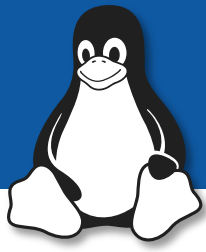
**kill** *[pid]* (où *[pid]* désigne le numéro de processus)

```
etudiant@etudiant-virtual-machine:~$ ps -aux | grep nmtui
etudiant  2157  0.0  0.2 239056 11032 tty3      Sl+   09:19   0:00 nmtui
etudiant  2225  0.0  0.0  21532  1004 pts/0     R+    09:28   0:00 grep --color=auto nmtui
etudiant@etudiant-virtual-machine:~$ kill 2157
etudiant@etudiant-virtual-machine:~$
```

Par défaut, la commande `kill` envoie un signal de terminaison au processus (*SIGTERM*). Si un processus ne répond plus et qu'un signal de terminaison ne suffit pas, l'option **-9** effectuera un arrêt forcé (*SIGKILL*).

```
etudiant@etudiant-virtual-machine:~$ kill -9 1234
```

# Arrêter un processus par son nom



On peut aussi arrêter tous les processus d'un nom précis. La commande est alors:

**killall** *[nom]* (où *[nom]* désigne le nom du processus)

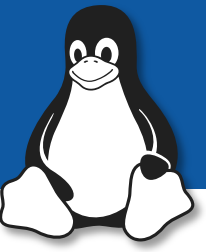
```
etudiant@etudiant-virtual-machine:~$ ps -aux | grep nmtui
etudiant  2196  0.1  0.2 238924 10672 pts/2    Sl+  19:39   0:00 nmtui
etudiant  2200  0.0  0.0  21532  1012 pts/0    S+   19:40   0:00 grep --color=auto nmtui
etudiant@etudiant-virtual-machine:~$ killall nmtui
etudiant@etudiant-virtual-machine:~$ ps -aux | grep nmtui
etudiant  2203  0.0  0.0  21532  1076 pts/0    S+   19:40   0:00 grep --color=auto nmtui
etudiant@etudiant-virtual-machine:~$
```



**Attention!** La commande killall envoie le signal de terminaison à TOUS les processus portant le nom demandé !

S'il y a plusieurs processus portant le même nom, ils recevront tous le signal de terminaison.

# Arrêter un processus d'un autre utilisateur



Chaque processus appartient à l'utilisateur qui l'a lancé, et seul cet utilisateur est autorisé à y mettre fin... ainsi que l'utilisateur **root**.

La commande **sudo** permet de lancer des commandes en tant que **root**, donc permet de mettre fin à tout processus, peu importe à qui il appartient.

Par exemple:

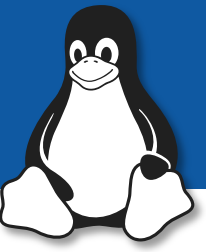
```
sudo kill [PID]
```

```
sudo killall [nomduprocessus]
```

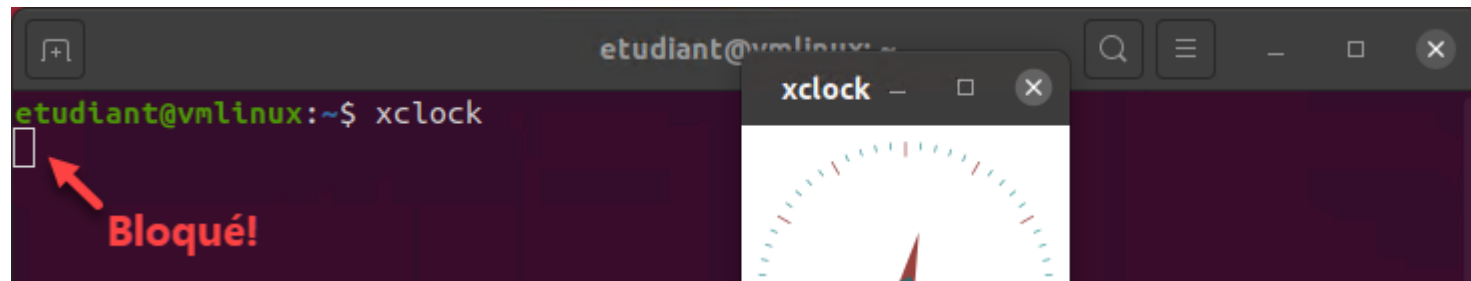
*etc.*



# Processus en arrière-plan

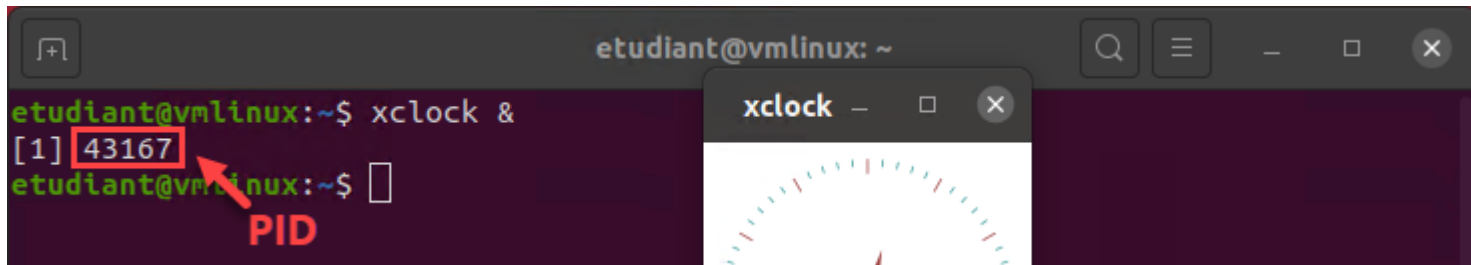


Lorsqu'on lance une commande dans le terminal, celui-ci est bloqué jusqu'à ce que la commande ait terminé son exécution.



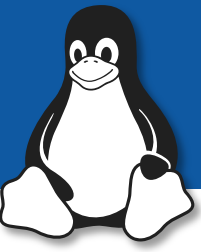
A terminal window titled 'etudiant@vmlinux: ~' shows the command 'xclock' being entered. A red arrow points to the prompt, and the word 'Bloqué!' is written in red. An 'xclock' window is visible in the background.

Pour lancer le programme en arrière-plan et le **détacher** du terminal, on utilise le caractère « & » à la fin de la commande



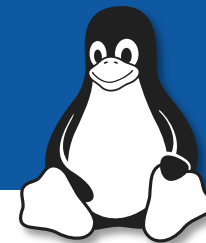
A terminal window titled 'etudiant@vmlinux: ~' shows the command 'xclock &' being entered. The output '[1] 43167' is shown, with '43167' highlighted in a red box. A red arrow points to the box, and the word 'PID' is written in red. An 'xclock' window is visible in the background.

# Commandes de gestion de processus



Commande	Description
<code>ps</code>	Affiche un état instantané des processus en cours
<code>ps -aux   grep nom_processus</code>	Affiche uniquement les lignes contenant <code>nom_processus</code>
<code>pstree</code>	<b>Affiche la hiérarchie des processus</b>
<code>kill</code>	Permet d'envoyer un signal au processus en utilisant son numéro de PID
<code>killall</code>	Permet d'envoyer un signal au processus en utilisant son nom.
<code>kill -9 PID_processus</code>	tue le processus ayant pour numéro de PID <code>PID_processus</code>
<code>killall -1 nom_processus</code>	Oblige <code>nom_processus</code> à relire son fichier de configuration
<code>kill -CONT PID_processus</code>	Réactive un processus ( <code>PID_processus</code> ) se trouvant en arrière plan
<code>kill -STOP PID_processus</code>	Suspend un processus ( <code>PID_processus</code> ) en arrière plan
<code>kill -l</code>	affiche tous les signaux disponibles
<code>nohup</code>	Exécute la commande désirée en ignorant le signal de déconnexion (HANGUP) de l'utilisateur qui l'a lancé. Le programme continue donc de fonctionner lorsque vous vous déconnectez.
<code>jobs</code>	donne la liste des programmes fonctionnant en tâche de fond

# Autres outils



Plusieurs outils plus avancés permettent de manipuler les processus. Essayez-les!

- > htop
- > glances
- > xkill

On peut aussi utiliser l'outil graphique appelé Moniteur système.

```
etudiant@etudiant-virtual-machine:
Fichier  Édition  Affichage  Rechercher  Terminal  Aide

1  [ |                                     1.3%]   Tasks: 149, 434
2  [ |                                     0.0%]   Load average: 0
Mem[|||||1.73G/3.83G]                   Uptime: 01:07:0
Swp[|||||0K/947M]

  PID USER      PRI  NI  VIRT   RES   SHR  S  CPU%  MEM%   TIME+  Command
  3741 etudiant   20    0 40796  4740  3864  R   0.7   0.1   0:00.17 htop
  1580 etudiant   20    0 3411M  338M  84808  S   0.0   8.6   0:45.94 /usr/bin/g
  1883 etudiant   20    0  790M  43352 29960  S   0.0   1.1   0:08.59 /usr/lib/g
  1453 etudiant   20    0  420M  85500 46440  S   0.0   2.1   0:25.17 /usr/lib/x
  1598 etudiant   20    0 3411M  338M  84808  S   0.0   8.6   0:00.10 /usr/bin/g
  1459 etudiant   20    0  420M  85500 46440  S   0.0   2.1   0:02.57 /usr/lib/x
  2517 etudiant   20    0 2515M  104M  83048  S   0.0   2.7   0:01.10 /usr/lib/f
  1772 etudiant   20    0  519M  36748 29596  S   0.0   0.9   0:02.84 /usr/bin/v
  1335 gdm         20    0  788M  49408 37484  S   0.0   1.2   0:00.44 /usr/lib/g
   976 gdm         20    0 3164M  138M  80444  S   0.0   3.5   0:04.21 /usr/bin/g
    1 root        20    0  156M   9100  6724  S   0.0   0.2   0:02.36 /sbin/init
```

Processus							Ressources		Systèmes de fichiers			
Nom du processus	Utilisateur	% CPU ▲	ID	Mémoire	Total lecture d	Total écr						
gnome-shell	etudiant	1	1580	277,8 MiB	119,3 MiB	1,1						
gnome-system-monitor	etudiant	0	6053	13,1 MiB	16,0 KiB	4,						
Xorg	etudiant	0	1453	37,1 MiB	5,3 MiB	164,						
systemd	etudiant	0	1433	1,6 MiB	20,0 MiB	8,						
(sd-pam)	etudiant	0	1434	2,5 MiB	N/D							
gnome-keyring-daemon	etudiant	0	1447	0,76 KiB	N/D							