

## Les processus

Si les ressources matérielles (mémoire, disques...) sont gérées par le kernel Linux tout le reste n'est que processus (\*).

Un processus est un concept clé de tous les systèmes d'exploitation. Un processus est un programme chargé en mémoire et en cours d'exécution. Les systèmes d'exploitation tel que GNU/Linux sont des systèmes multitâches préemptifs, c'est à dire que chaque programme ou processus tournent indépendamment. Lorsqu'un processus est bloqué, le système continue à tourner car les processus sont traités indépendamment. La destruction d'un processus n'a pas d'effet sur l'exécution des autres processus. GNU/Linux n'est pas constitué d'un seul exécutable, mais d'un ensemble de processus qu'on choisit ou non d'exécuter.

ex : avant connexion, le terminal est épié par le processus *getty* et meurt une fois que vous êtes connecté (il sera relancé à la déconnexion). Le shell prend en charge le terminal, qui est un processus distinct. Il crée alors un nouveau processus chaque fois que vous tapez une commande. Chaque processus peut créer lui-même des processus d'où la notion de processus parent. C'est le cas par exemple du serveur Apache : lors de son lancement, le processus père crée plusieurs processus fils afin de répondre indépendamment à plusieurs clients. La destruction du *processus parent* (parent process) entraîne la destruction de tous les *processus fils* (child process).

Pour chaque processus exécuté, le système d'exploitation stocke un certain nombre d'informations :

- indication d'état, précisant entre autre si le processus est exécuté en mémoire centrale ou s'il a été déporté sur la zone de spool ;
- numéro d'utilisateur UID, *User IDentification*, ayant lancé le processus ;
- numéro du groupe GID, *Group IDentification*, ayant lancé le processus ;
- numéro unique du processus PID, *Process IDentification* ;
- numéro du processus parent PPID, *Parent Process IDentification* ;
- facteur de priorité PRI, *PRiority* : plus la valeur est grande et plus le processus sera appelé rapidement pour être traité en mémoire centrale ;
- facteur NI, *NIce* : définit un délai de report pour le processus ;
- taille totale du programme dans la mémoire SIZE ;
- mémoire réservée RSS, *Resident Set Size* ;
- canal d'attente WCHAN, *Waiting CHANel* : par lequel se fait la coordination avec les autres processus ;
- état du processus STAT, *STATus* : S, *Sleeping* (endormi ) ou R, *Running* (actif) ;
- heure de lancement du processus STIME, *Start TIME* ;
- nom du terminal TTY ;
- durée de traitement utilisé (temps CPU) du processus TIME (en secondes).

(\*) Note : tout processus effectue ses entrées/sorties par des "descripteurs de fichiers" (0, 1, 2). Leur association à un fichier ou à un périphérique, fait partie de l'environnement d'un processus.

## Liste des processus : ps

ps permet d'obtenir des informations sur les processus exécutés par le système. Cette commande permet de connaître la liste des processus actifs à un moment donné.

La commande ps sans arguments ne fournit que la liste des processus associés au terminal utilisé :

ex : ps

PID	TTY	TIME	CMD	affiche : N°process	Terminal	Temps process	Commande
7915	tty1	00:00:00	bash	tty	n° : console virtuelle	n°	
8377	tty1	00:00:00	ps				

Pour connaître les processus exécutés par le système, il est nécessaire d'ajouter des arguments :

ex : ps T liste et affiche tous les processus lancés par l'utilisateur dans ce terminal

PID	TTY	STAT	TIME	COMMAND
7915	pts/2	S	0:00	bash pts/n° : pseudo terminal n°
8331	pts/2	R	0:00	ps T

ps x affiche tous les processus lancés par l'utilisateur

ps aux affiche tous les processus lancés par tous les utilisateurs

ps aux | grep httpd affiche tous les processus nommés httpd lancés

ps e liste de tous les processus

ps auxw | more affichage large (w) de la liste de tous les processus (ax) avec nom user (u)

ps axl | more liste longue (l)

ps ax | grep sendmail teste si le démon (daemon) sendmail tourne

ps auxr liste les processus actifs (running)

ps -lU toto liste détaillée des processus lancés par toto

ps axf | more affiche la liste avec les arbres généalogiques des processus : filiation (f)

ps axl | sort liste avec tri

ps rss | more liste par RSS, Resident Set Size ; KiloBytes of program in memory

ps -U root -u root -N liste tous les processus sauf ceux de root

Pour obtenir un affichage remis à jour régulièrement, utilisez la commande top.

## Arrêter une commande : kill

Les processus en tâche de fond ne peuvent pas être arrêtés par les touches ordinaires : kill permet d'expédier un signal à un processus en cours. Un signal est un chiffre ayant une signification particulière. Lorsqu'on envoie un signal à un processus, (un numéro) il se doit de réagir en fonction de la valeur de ce signal. La réaction normale à l'arrivée d'un signal est l'arrêt du processus.

ex : kill [signal] PID PID étant toujours le numéro de de processus

kill -l affiche la liste des principaux signaux de processus

kill 2 PID signal d'interruption (mnémonique : SIGINT) envoyé par [Ctrl-C]

kill 9 PID le signal 9 (mnémonique : SIGKILL) ne peut être ignoré par aucun processus

killall tue un processus mais au lieu d'indiquer le PID vous indiquerez le nom du processus

## Lancement d'un processus en arrière plan : & et nohup

Les processus peuvent être soit en avant-plan, soit en arrière-plan ; appelé aussi parfois tâche de fond. Le processus en avant-plan est celui avec lequel vous dialoguez, il reçoit des données de votre clavier et envoie des messages sur votre écran. (Sauf bien sûr si vous avez redirigé ces entrées/sorties comme expliqué dans la section ci-dessous). A l'opposé, un processus en arrière-plan ne reçoit rien de votre terminal ; en général ils tournent tranquillement sans jamais rien demander à personne.

Le `shell` offre la possibilité de lancer des processus en ligne de commande et l'*invite* (prompt) du shell peut réapparaître dès que le traitement de cette commande est terminé et que le processus est clos. Pour ne pas attendre la fin d'exécution synchrone des processus, le `shell` dispose d'une technique simple mais très efficace pour lancer un ou plusieurs processus en arrière plan. Ce processus sera également un process enfant du `shell`. Pour ce faire nous devons rajouter à la fin de la commande le signe `&`, *ET commercial* (ampersand) :

`&` cet opérateur permet de lancer plusieurs processus en parallèle alors que l'opérateur `;` lance les processus en série (les utilisateurs d'UNIX profitaient ainsi des capacités multitâches du système).

```
ex : cat > mypro.c fichier : #include <stdio.h> [↵] main() {
        printf ("Hello World !\n"); [↵] }
gcc -o mypro mypro.c & compilation gcc du programme .c exécutée en arrière plan.
[1] 4880 résultat : le chiffre [1] représente le numéro de job (tâche) associé à votre
commande, et 4880 le numéro de processus (PID).
```

```
gcc -o mypro mypro.c fichier : //mypro.c [↵] #include <stdio.h>
int main() { [↵] int i; [↵] int j = 0;
for (i = 0; i < 1000000000; i++) {
while (j < 100000000) { [↵] j++; } [↵] } [↵] return 0; [↵] }
./mypro & exécute le programme en tâche de fond
[1] 5241
ps aux affiche les processus des utilisateurs
toto 5241 84.0 0.0 2348 232 pts/2 R 15:42 0:03 ./mypro
```

`nohup`, *no hang up*, permet de lancer une commande qui ne sera pas interrompue par un signal de déconnexion ou de `quit` (ignore les signaux dans le cadre d'une commande). Cette commande (\*) entreprend d'office une redirection des entrées/sorties, si celle-ci n'a pas été spécifiée dans la commande elle-même. La commande redirigera la canal de la sortie standard et le canal d'erreur standard vers le fichier `nohup.out`. Le canal d'entrée standard est lié au fichier `/dev/null` :

ex : `nohup make &` conserve l'exécution de la commande `make` (compile de gros programmes) même si l'utilisateur se déconnecte. La commande s'exécutera en tant que tâche de fond sauf si il est suivi d'un `&`. La sortie de la compilation est dirigée par défaut vers un fichier `nohup.out`.

(\*) Note : si vous utilisez `nohup` dans un script, placez un `wait` (rend la main que lorsque tous les processus en tâche de fond sont terminés) pour éviter la création d'un *processus orphelin* (zombie).

## Redirection des E/S (Entrées/Sorties) : <, >, <<, >> et |

Toutes les commandes (du noyau, du shell et créées par le programmeur) sont dotées par le système de 3 canaux de communication :

- l'*entrée standard* (stdin : standard input) pour lire des données sur l'entrée : descripteur 0,
- la *sortie standard* (stdout : standard output) pour envoyer des résultats sur la sortie : descripteur 1,
- la *sortie des erreurs* (stderr : standard error) pour retourner des erreurs sur la sortie : descripteur 2.

Par défaut les canaux d'entrées et de sorties communiquent avec le clavier et l'écran : les commandes et les programmes qui ont besoin de données les attendent en provenance du clavier et expédient leurs résultats pour affichage sur le moniteur. Il est possible de les détourner pour les rediriger vers des fichiers ou même vers les entrées-sorties d'autres commandes. Les symboles utilisées sont :

> la sortie standard est redirigée dans un fichier (RAZ du fichier, écrase le fichier existant) : redirection stdout en direction d'un fichier et non vers l'écran/clavier :

```
ex: cat > lettre.txt redirection des textes ci-après dans le fichier
      une conférence.[Return]           ligne de texte
      tu pouvais venir faire [Return]   ligne de texte
      [Ctrl-D]                           termine cat
```

< l'entrée standard est lu à partir d'un fichier : redirection stdin depuis le fichier et non du clavier :  
ex: sort < lettre.txt le contenu du fichier est trié sur stdout

>> la sortie standard est redirigée dans un fichier (concaténation du fichier) : redirection stdout sur un fichier existant sans RAZ :

```
ex: cat >> lettre.txt [Return]
      Je serais très heureux si [Return]   ligne de texte
      [Ctrl-D]
```

<< l'entrée standard est lue jusqu'à ce que la chaîne située à droite soit rencontrée : l'exemple ci-dessous va lire stdin jusqu'à ce que le mot STOP soit rencontré, puis va afficher le résultat :

```
ex: cat >> lettre.txt << STOP
      Cher Linus, [Return]
      STOP [Return] dans ce cas pas besoin d'utiliser [Ctrl-D]
```

```
sort < lettre.txt
sort < lettre.txt > lettre-tri.txt
cat lettre-tri.txt
cat lettre.txt lettre-tri.txt
```

## Enchaînement de commandes : |, ;, `...`, && et ||

Le Shell BASH permet d'enchaîner plusieurs commandes :

| enchaînement de commandes grâce à un *tube* (pipe) : la sortie de la commande gauche est envoyée en entrée de la commande droite :

ex : `ls -l > liste.txt | cat >> liste.txt << STOP | more liste.txt` permet d'insérer un texte dans le fichier de sortie de la commande `ls` et lister son contenu page par page

`ls | wc -l` compte le nombre de fichiers dans le répertoire courant

; permet d'exécuter des commandes séquentiellement : la commande placée à gauche du ";" est déjà exécutée puis s'enchaîne l'exécution de la seconde commande placée après ";" :

ex : `date ; pwd` affiche la date puis le répertoire courant

`...` argument donné et placé entre quotes (\*) sur la sortie standard d'une commande :

ex : `tar cvf arch.tar `ls *.c`` archive les fichiers \*.c en plaçant l'argument de la commande `ls` entre quotes (`.`) sur la sortie de `tar`.

&& symétrique au tube (|) mais la commande suivante s'exécute si et seulement si la précédente commande s'est déroulée correctement (code de retour == 0) :

ex : `test -f fichier && more fichier` si et seulement si `test` retourne 0 l'enchaînement avec `more` s'exécute et affiche le fichier

|| symétrique au tube (|) mais n'exécute la commande suivante (après ||) que si la précédente commande (avant ||) échoue (code retour différent de 0) :

ex : `test -f fichier || "fichier inconnu"` le texte "fichier inconnu" s'affiche que si `test` retourne une valeur différente de 0

On peut enchaîner plusieurs commandes :

ex : `test -f fichier && more fichier || "Ce fichier n'existe pas dans le répertoire de travail courant $PWD"`

(\*) Notes : ne pas confondre le caractère simple quote (`) avec l'apostrophe (') utilisé pour la protection forte de chaînes de caractères ('chaîne\_caractères'). Il est courant, lors de la lecture ou de l'écriture d'un script de discerner clairement (visuellement parlant) ces deux caractères. Dans la plupart des cas un message d'erreur sera retourné.