Module 1 - Gestion de processus

Nous verrons comment gérer les processus qui roulent sur une machine Linux. La façon de faire est comparable à l'utilisation des commandes tasklist et taskkill de Windows, mais avec quelques nuances importantes.

Tâches en arrière-plan

Linux est évidemment un système d'exploitation multitâches. Il permet d'exécuter plusieurs processus de façon concurrente.

- Un **processus** est un programme en cours d'exécution, avec ses données.

Il est possible pour un utilisateur de lancer des processus à l'arrière plan tout en poursuivant son travail normal. Il suffit pour ce faire d'ajouter une esperluette (&) à la fin de la commande:

```
nomDuProgramme –switch paramètre1 paramètre2 &
[1] 1212
```

l'interpréteur de commandes identifie le processus avec un numéro de job et un PID (Process Identification Number). Il nous les affiche immédiatement sous notre commande avant de nous redonner notre prompt.

- Dans notre exemple (ci-dessus), le numéro de la job serait 1 et le PID serait 1212.

Visualiser les processus avec ps

La commande servant à visualiser les processus en cours d'exécution est ps. Par exemple, ps affichera la liste de vos processus présentement en exécution dans l'ordinateur où se déroule la session de travail (où s'exécute votre shell).

Si vous faites un man ps, vous vous rendrez compte qu'il y a une très grande quantité de switches possibles, la plupart d'entre elles plus ou moins incompréhensibles.

Les plus importantes sont les suivantes:

- -e: affiche tous les processus sur la machine, incluant ceux des autres et incluant ceux qui ne sont pas associés à un port tty (donc démarrés par le système) - finalement, tout.
- -a: n'affiche pas les processus du système, ni les "session leaders" (les processus de login et les init)
- -x: affiche les processus du système (sans tty) et les "session leaders", et ajoute l'état de chaque processus
- -f: affiche un format plus long, avec plus d'information sur chaque processus

En gros, ps –ax équivaut à ps –e, en ce sens qu'ils nous affichent la même liste de processus. Ils diffèrent par contre sur un point: ps –ax affiche l'état de chaque processus et ps –e ne le fait pas.

ps tout seul affiche uniquement les processus de l'usager en cours, ce qui est souvent suffisant. ps –f ajoutera de l'information sur chaque processus.

Un administrateur du système qui voudra voir tout ce qui roule sur son serveur, peu importe chez quel usager, utilisera ps –ef pour voir tous les processus possibles avec un maximum d'information.

Voici quelques exemples d'utilisations courantes, avec explication sur la sortie.

Exécution typique de ps (tout court, sans switches):

```
PID  TTY      TIME     CMD
2321 pts/0    00:00:00 bash
2332 pts/0    00:00:00 ps
```

À noter: on voit ici la liste de vos processus les plus clairement actifs au moment du lancement de la commande: votre shell (bash), et la commande ps en cours d'exécution.

- Le TTY est le port qui est attribué par le système à votre session de travail (c'est le même dans chaque cas, puisqu'on parle d'une seule et même session de travail).
- Le PID est le numéro d'identification du processus.
- Le TIME est le temps de traitement total utilisé par chaque processus, arrondi à la seconde (ce qui donne souvent des 0).

Exécution typique de ps -f (voir les processus lancés par l'usager, de façon plus détaillée et en ordre croissant de PID):

```
UID        PID   PPID  C STIME TTY      TIME     CMD
eforest    2321  2320  0 06:04 pts/0    00:00:00 bash
eforest    2366  2321  0 06:17 pts/0    00:00:00 ps –f
```

- Le UID identifie l'utilisateur propriétaire de chaque processus.
- Le PID est le numéro d'identification du processus.
- Le PPID identifie le processus parent de chaque processus. Tout processus a un parent (par exemple, ici, le parent de la commande ps -f est le shell d'où elle a été lancée).
- Le C est le pourcentage de temps d'utilisation du CPU sur le temps réel écoulé depuis le lancement du programme, arrondi à l'entier près.
- STIME est l'heure (ou la date, si ce n'est pas aujourd'hui) où le processus a été lancé.
- Il est possible, avec d'autres options, de modifier grandement l'affichage des données de ps.

À noter: un processus enfant hérite de la plupart des propriétés de son parent (si un processus root lance une commande, elle aura elle aussi par défaut des droits root).

Exécution typique de ps -ef (voir tous les processus, en ordre croissant de PID):

```
UID        PID   PPID  C STIME TTY          TIME CMD
root         1      0  0 Jan30 ?        00:00:04 init [3]
root         2      1  0 Jan30 ?        00:00:00 [kflushd]
root         3      1  0 Jan30 ?        00:00:00 [kupdate]
root         4      1  0 Jan30 ?        00:00:00 [kpiod]
root         5      1  0 Jan30 ?        00:00:01 [kswapd]
bin         72      1  0 Jan30 ?        00:00:00 /sbin/rpc.portmap
root        76      1  0 Jan30 ?        00:00:03 /usr/sbin/syslogd
root        79      1  0 Jan30 ?        00:00:00 [klogd]
root        81      1  0 Jan30 ?        00:00:00 /usr/sbin/inetd
root        83      1  0 Jan30 ?        00:00:00 [lpd]
root        86      1  0 Jan30 ?        00:00:00 /usr/sbin/rpc.mountd
root        88      1  0 Jan30 ?        00:00:00 /usr/sbin/rpc.nfsd
root        90      1  0 Jan30 ?        00:00:00 /usr/sbin/crond -l10
daemon      92      1  0 Jan30 ?        00:00:00 /usr/sbin/atd -b 15 -l 1
root        99      1  0 Jan30 ?        00:00:02 sendmail: accepting connections
root       104      1  0 Jan30 ?        00:00:00 /var/lib/apache/sbin/httpd
nobody     107    104  0 Jan30 ?        00:00:06 /var/lib/apache/sbin/httpd
nobody     108    104  0 Jan30 ?        00:00:07 /var/lib/apache/sbin/httpd
nobody     109    104  0 Jan30 ?        00:00:10 /var/lib/apache/sbin/httpd
nobody     110    104  0 Jan30 ?        00:00:01 /var/lib/apache/sbin/httpd
nobody     111    104  0 Jan30 ?        00:00:23 /var/lib/apache/sbin/httpd
root       114      1  0 Jan30 ?        00:00:06 /usr/sbin/xntpd
root       116      1  0 Jan30 tty2     00:00:00 [agetty]
root       117      1  0 Jan30 tty3     00:00:00 [agetty]
root       118      1  0 Jan30 tty4     00:00:00 [agetty]
root       119      1  0 Jan30 tty5     00:00:00 [agetty]
root       120      1  0 Jan30 tty6     00:00:00 [agetty]
nobody     121    104  0 Jan30 ?        00:00:18 /var/lib/apache/sbin/httpd
nobody     129    104  0 Jan30 ?        00:00:03 /var/lib/apache/sbin/httpd
nobody     307    104  0 Jan31 ?        00:00:11 /var/lib/apache/sbin/httpd
nobody     308    104  0 Jan31 ?        00:00:05 /var/lib/apache/sbin/httpd
nobody     309    104  0 Jan31 ?        00:00:08 /var/lib/apache/sbin/httpd
robin      938      1 99 Feb02 ?        6-19:55:54 mprime -B
root       983      1  0 Feb02 tty1     00:00:00 [agetty]
root      2320     81  0 06:04 ?        00:00:00 in.telnetd: 64.228.235.171
eforest   2321   2320  0 06:04 pts/0    00:00:00 bash
eforest   2336   2321  0 06:06 pts/0    00:00:00 ps -ef
eforest   2337   2321  0 06:06 pts/0    00:00:00 bash
```

Exécution typique de ps -ax (voir tous les processus du terminal courant, incluant ceux des autres usagers, et incluant ceux qui ne sont pas contrôlés directement par un terminal):

```
PID  TTY      STAT   TIME COMMAND
1    ?        S      0:04 init [3]
2    ?        SW     0:00 [kflushd]
3    ?        SW     0:00 [kupdate]
4    ?        SW     0:00 [kpiod]
5    ?        SW     0:01 [kswapd]
72   ?        S      0:00 /sbin/rpc.portmap
76   ?        S      0:03 /usr/sbin/syslogd
79   ?        SW     0:00 [klogd]
```

```
81   ?        S      0:00 /usr/sbin/inetd
83   ?        SW     0:00 [lpd]
86   ?        S      0:00 /usr/sbin/rpc.mountd
88   ?        S      0:00 /usr/sbin/rpc.nfsd
90   ?        S      0:00 /usr/sbin/crond -l10
92   ?        S      0:00 /usr/sbin/atd -b 15 -l 1
99   ?        S      0:02 sendmail: accepting connections
104  ?        S      0:00 /var/lib/apache/sbin/httpd
107  ?        S      0:06 /var/lib/apache/sbin/httpd
108  ?        S      0:07 /var/lib/apache/sbin/httpd
109  ?        S      0:10 /var/lib/apache/sbin/httpd
110  ?        S      0:01 /var/lib/apache/sbin/httpd
111  ?        S      0:23 /var/lib/apache/sbin/httpd
114  ?        SL     0:06 /usr/sbin/xntpd
116  tty2     SW     0:00 [agetty]
117  tty3     SW     0:00 [agetty]
118  tty4     SW     0:00 [agetty]
119  tty5     SW     0:00 [agetty]
120  tty6     SW     0:00 [agetty]
121  ?        S      0:18 /var/lib/apache/sbin/httpd
129  ?        S      0:03 /var/lib/apache/sbin/httpd
307  ?        S      0:11 /var/lib/apache/sbin/httpd
308  ?        S      0:05 /var/lib/apache/sbin/httpd
309  ?        S      0:08 /var/lib/apache/sbin/httpd
938  ?        RN     9836:02 mprime -B
983  tty1     SW     0:00 [agetty]
2320 ?        S      0:00  in.telnetd: 64.228.235.171
2321 pts/0    S      0:00 -bash
2339 pts/0    R      0:00 ps ax
```

- Le STAT identifie l'état d'exécution de chaque processus. On verra S pour dormant (sleeping), R pour en cours d'exécution (running or runnable) et T pour suspendu (traced); pour les définitions de Z, L, W, etc (beaucoup moins courantes), référez-vous aux pages de manuel (man ps).

On peut aussi visualiser les processus en ordre décroissant d'utilisation de l'unité centrale de traitement avec la commande top. L'affichage est également rafraîchi régulièrement :

```
6:13am  up 9  days, 10:34,  1 user,  load average: 1.00, 1.00, 1.00
37 processes: 35 sleeping, 2 running, 0 zombie, 0  stopped
CPU states:   0.0% user,  0.0% system,  3.8% nice,   1.2% idle
Mem:    62904K  av,   55076K used,    7828K free,   12056K shrd,   33548K  buff
Swap:   64508K  av,    3584K used,   60924K free                    7072K cached

PID USER      PRI   NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
938  robin     20   19  4568 4548   372 R N  96.8  7.2  9843m mprime
2358 eforest    2    0   892  892   688 R     2.8  1.4   0:00 top
1    root       0    0    72   60    40 S     0.0  0.0   0:04 init
2    root       0    0     0    0     0 SW    0.0  0.0   0:00 kflushd
3    root       0    0     0    0     0 SW    0.0  0.0   0:00 kupdate
4    root       0    0     0    0     0 SW    0.0  0.0   0:00 kpiod
5    root       0    0     0    0     0 SW    0.0  0.0   0:01 kswapd
72   bin        0    0   348  328   280 S     0.0  0.5   0:00 rpc.portmap
76   root       0    0   256  208   136 S     0.0  0.3   0:03 syslogd
```

```
79   root       0    0   480    0     0 SW    0.0  0.0   0:00 klogd
81   root       0    0   268  228   188 S     0.0  0.3   0:00 inetd
83   root       0    0   112    0     0 SW    0.0  0.0   0:00 lpd
86   root       0    0   180   76    48 S     0.0  0.1   0:00 rpc.mountd
88   root       0    0   188   76    44 S     0.0  0.1   0:00 rpc.nfsd
```

Il existe plusieurs commandes qui peuvent être tapées directement dans top, pour modifier l'affichage ou gérer les processus à partir de là. La plus importante à connaître: q pour quitter! Pour le reste, faites un man top pour plus de détails!

Jobs versus processus

Il est possible d'arrêter ou de suspendre temporairement un processus, de même que de le faire passer de l'exécution en arrière-plan à une exécution en avant-plan (et inversement). Dans ce contexte, on nommera les processus des jobs. En fait, **on dira qu'un processus devient une job aussitôt qu'on s'est mêlé de son exécution**, aussitôt qu'on la contrôlera. Du coup, toute job est un processus mais tout processus n'est pas une job.

On visualisera les jobs à l'aide de la commande jobs. Les jobs seront listées de par leur numéro de job, pas leur PID.

Tuer les processus

On tue un processus pour en interrompre de force l'exécution et l'éliminer de la mémoire. La commande servant à tuer un processus est kill <PID>

Notez que:

- Seul l'usager root peut tuer les processus des autres usagers.
- Parfois, un processus refusera le signal émis par un kill. Il sera quand même possible de le tuer avec kill -SIGKILL <PID>
- Vous verrez aussi, dans le registre des meurtres brutaux, la commande kill -9 <PID> qui est équivalente à kill -SIGKILL <PID>
- En réalité, kill n'est pas une commande qui sert uniquement à tuer des processus. C'est plutôt une commande qui sert à envoyer des signaux aux processus. Certains signaux tuent, d'autres demandent au processus de se terminer, d'autres encore peuvent avoir tout un tas de fonctionnalité.
- Chaque signal porte un nom qui commence par SIG et un numéro. On peut déterminer quel signal envoyer au processus en utilisant l'un ou l'autre, comme une switch. C'est pourquoi on peut faire kill -9 ou kill -SIGKILL pour obtenir exactement le même effet.
- Si on omet de spécifier un signal, kill envoie par défaut SIGTERM (ou 15), qui demande au processus de se terminer "gracieusement".
- Il est possible d'énumérer plusieurs PID dans une même commande: kill 123 145 1203 2345

Il existe également une commande killall qui fonctionne exactement comme kill mais qui accepte un nom de processus plutôt qu'un PID. killall tuera d'un seul coup tous les processus portant ce nom.

Gestion des jobs et des processus

On interrompra un processus s'exécutant en avant-plan à l'aide des touches CTRL-C (attention: ceci ne fonctionnera pas dans tous les cas). Ceci terminera l'exécution du processus en question pour de bon – équivalent à faire kill <PID>.

On suspendra un processus s'exécutant en avant-plan à l'aide des touches CTRL-Z (attention: ceci ne fonctionnera pas dans tous les cas). L'exécution du processus en question pourra être reprise ultérieurement – pour le moment elle sera suspendue.

On enverra en arrière-plan un processus dont l'exécution a été suspendue (donc une job) à l'aide de la commande bg <#job>. Ce processus reprendra donc son exécution, mais en arrière-plan.

On enverra en avant-plan une job (qu'elle soit en ce moment en arrière-plan ou suspendue) à l'aide de la commande fg <#job>. Si la job à envoyer en avant-plan est la toute dernière à avoir été envoyée en arrière-plan, on pourra même simplement taper fg.

On pourra tuer une job à l'aide de la commande kill <%job>. Ne pas oublier le % dans ce cas-là, c'est ce qui dit à la commande kill que le numéro qu'on lui passe est un numéro de job et non un PID... Il est bien entendu également possible d'utiliser son PID de façon normale, puisque toute job est un processus.

Démonstration – l'exemple de la commande fort positive yes

La commande yes a l'air bien anodine mais elle peut (parfois) être utile. Essayez-la d'abord pour voir ce qu'elle fait :

```
yes
```

Miracle! La commande yes vous dit « y » à répétition et ce jusqu'à l'infini :

```
y
y
y
y
y
y
y
y
y
y
y
```

À quoi ça sert? Simplement à répondre « oui » pour nous à une commande qui demande un tas de confirmations. Essayons-la dans un contexte (un peu plus) utile :

Peu importe où, créez un nouveau répertoire, disons « allo ».	mkdir allo
Allez dedans.	cd allo

Créez tout un tas de fichiers.	touch 1; touch 2; touch 3; touch 4; touch 5
Vérifiez qu'ils sont bien là.	ls
Créez l'alias rm pour qu'il vous demande une confirmation à chaque fois (s'il n'existe pas déjà).	alias rm='rm –i'
Tentez d'effacer tous les fichiers du répertoire (mais répondez non pour l'instant).	rm * n n n n n
Utilisez la commande yes pour envoyer une suite de « y » à rm.	yes | rm *
Regardez le contenu du répertoire.	ls

Voilà son utilité. Remarquez que dans un cas comme ça on aurait pu simplement faire \rm * pour contourner l'alias, mais certains programmes peuvent demander des confirmations sans être eux-mêmes des alias...

Nous allons maintenant utiliser la commande yes pour démontrer le contrôle des jobs – c'est un outil idéal puisqu'il s'exécute à l'infini, demande du temps de processeur, mais ne fait pas grand chose.

Exécutez yes, mais pour ne pas s'embêter avec la sortie nous allons la rediriger vers /dev/null (le fameux trou noir de Unix où tout ce qui y entre disparaît sans laisser de trace).	yes > /dev/null
Vous remarquez que vous ne recevez plus votre prompt, le système semble « gelé ». Pourquoi? Simplement parce que la commande yes est en exécution et que le prompt vous reviendra lorsqu'elle aura terminé (dans ce cas-ci, jamais). Comme on a redirigé la sortie dans /dev/null, on ne voit rien mais en fait une longue série de "y" s'engouffre en ce moment dans un trou noir. Ayez une petite pensée pour eux et passez au point suivant.	
On pourrait arrêter l'exécution de yes en faisant CTRL-C, ce qui termine le processus présentement en avant-plan pour toujours.	CTRL-C
Mais qu'est-ce qu'on fait si on veut qu'il continue son exécution pendant que nous on fait autre chose? Relançons-la de nouveau.	yes > /dev/null
Et stoppons-la plutôt que de la terminer.	CTRL-Z
Vous retrouvez votre prompt! Victoire! Bash vous informe également que le processus est stoppé et lui assigne un numéro de job : 1.	
Faites un ps -x pour voir ce qui se passe sur le système. On y voit encore le processus yes mais son statut est à T (traced) plutôt qu'à R (running) ou même S (sleeping). En d'autres termes, notre prompt est revenu mais notre job s'est arrêtée et ne s'exécute plus. Elle est dans un état suspendu - comme si on avait mis le film à pause. Elle est donc prête à continuer à l'endroit exact où elle a été interrompue, pour peu qu'on lui en donne l'ordre.	ps –x
On peut aussi, à ce stade-ci, utiliser la commande jobs qui nous montre l'état de nos jobs. Une job est un processus dont on contrôle l'exécution de façon particulière, comme par exemple en le suspendant.	jobs
Si on souhaite qu'elle continue à rouler, on doit lui dire de rouler en background (arrière-plan) ou en foreground (avant-plan). Allons-y avec l'arrière-plan. On utilisera la commande bg et on lui passera le numéro de job qui a été assigné à notre processus. Ce n'est pas nécessaire d'utiliser le % puisque bg n'accepte que des numéros de jobs, donc aucune confusion possible avec des PIDs.	bg 1

Un autre ps -x nous confirmera que yes envoie encore des "y" dans le trou noir – son statut est à R. La commande a repris son exécution au point exact où elle avait arrêté, rien n'a été perturbé.	ps –x
Laissons cette job-là rouler et essayons maintenant quelque chose de plus rapide : exécuter yes directement en background, sans d'abord le suspendre. Bash assigne à ce nouveau yes le job id 2.	yes > /dev/null &
Voyons maintenant l'état de nos jobs.	jobs
Il y en a deux qui roulent. On peut en ramener une en avant-plan avec la commande fg. On y perd alors notre prompt.	fg 1
Pour ravoir notre prompt, on n'a pas d'autre choix que de la suspendre puis de la renvoyer en arrière-plan.	CTRL-Z bg 1
Comme ces jobs ne s'arrêteront jamais, on devra les tuer sans pitié éventuellement. Assurez-vous que Brigitte Bardot n'est pas dans le coin en train de faire un discours sur les pauvres jobs qu'on tue inutilement et laissez aller votre agressivité. Notez que comme kill peut prendre en paramètre un numéro de job ou un PID, on utilisera le % pour lui dire qu'on parle de jobs.	kill %1
Voyons l'état des choses. On voit que la première a été « terminated » (on ne l'a pas entendu puisque son output est redirigé à /dev/null, mais elle a certainement dit « I'll be back » avec un accent allemand avant de disparaître).	jobs
Si on le refait, il ne nous reste plus qu'une job. Remarquez qu'elle garde son numéro 2 même si la 1 n'existe plus, ce qui est fort cohérent.	jobs
Juste pour faire les choses différemment allons trouver le PID de cette job.	ps –x ou: jobs -l
Tuons-la maintenant en utilisant le PID plutôt que le numéro de job. À ce moment on omettra le %.	kill 27240 (le vôtre sera différent)