Contrôle des processus

Les systèmes Slackware exécutent souvent des centaines ou des milliers de programmes, et chacun d'entre eux est appelé processus (*process*). Gérer ces processus est une part importante de l'administration système. Comment pouvons-nous manipuler tous ces processus distincts ?

ps

La première étape dans la gestion des processus est de savoir quels sont en cours d'exécution. L'outil le plus populaire et le plus puissant pour cela est ***ps***(1). Lancé sans argument, ***ps*** ne vous donnera pas beaucoup d'information. Par défaut, il vous dira quels sont les processus en exécution dans votre shell actif. Si nous souhaitons plus d'information, il faudra regarder plus en détails.

```
darkstar:~$ ps
  PID TTY          TIME CMD
12220 pts/4    00:00:00 bash
12236 pts/4    00:00:00 ps
```

Vous pouvez voir ici quels processus sont exécutés dans votre shell courant ou votre terminal et peu d'informations sont données. Le PID est le "*Process ID*" (identifiant de processus) ; chaque processus reçoit un numéro unique. Le TTY vous indique à quel terminal le processus est attaché. Vous serez peut-être un peu désorienté par la colonne TIME, qui semble se déplacer lentement. Ce n'est pas le temps écoulé pendant lequel le processus s'est exécuté, mais la quantité de temps CPU que le processus a utilisé. Un processus au repos (*idle*) n'utilise virtuellement aucun temps CPU, c'est pourquoi la valeur n'augmente pas très vite.

Visualiser uniquement ses propres processus n'est pas très amusant, jetons donc un œil sur tous les processus du système avec l'option *-e*.

```
darkstar:~$ ps -e
  PID TTY          TIME CMD
    1 ?        00:00:00 init
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 migration/0
    4 ?        00:00:00 ksoftirqd/0
    7 ?        00:00:11 events/0
    9 ?        00:00:01 work_on_cpu/0
   11 ?        00:00:00 khelper
  102 ?        00:00:02 kblockd/0
  105 ?        00:01:19 kacpid
  106 ?        00:00:01 kacpi_notify
... beaucoup de lignes supprimées ...
```

L'exemple ci-dessus utilise la syntaxe standard de ***ps***, mais beaucoup plus d'informations peuvent être vues si nous utilisons la syntaxe BSD. Pour cela nous devons utiliser l'option *aux*.

Ceci est différent de l'option *-aux*, mais dans les plupart des cas, les deux options sont équivalentes. C'est une relique d'il y a quelques dizaines d'années. Pour plus d'information, lisez la page de manuel de ***ps***.

```
darkstar:~$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   3928   632 ?        Ss   Apr05   0:00 init [3]
root         2  0.0  0.0      0     0 ?        S    Apr05   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    Apr05   0:00
[migration/0]
root         4  0.0  0.0      0     0 ?        S    Apr05   0:00
[ksoftirqd/0]
root         7  0.0  0.0      0     0 ?        S    Apr05   0:11 [events/0]
root         9  0.0  0.0      0     0 ?        S    Apr05   0:01
[work_on_cpu/0]
root        11  0.0  0.0      0     0 ?        S    Apr05   0:00 [khelper]
... beaucoup de lignes supprimées ....
```

Comme vous pouvez le voir, la syntaxe BSD offre beaucoup plus d'information, incluant quel utilisateur contrôle le processus et quels pourcentages de RAM et de CPU le processus consomme lorsque ***ps*** est lancé.

Pour obtenir un résultat spécifique, basé sur chaque processus, ***ps*** accepte un ou plusieurs PID comme options en ligne de commande et dispose de l'option '-o' pour afficher un attribut particulier d'un PID.

```
darkstar:~$ ps -o cmd -o etime $$
CMD                          ELAPSED
/bin/bash                      12:22
```

Dans le résultat nous trouvons le nom de la commande (cmd) pour ce PID et le temps écoulé (etime). Le PID dans cet exemple est une variable du shell pour le PID du shell courant. Vous pouvez donc voir dans cet exemple que le processus de ce shell existe depuis 12 minutes et 22 secondes.

En utilisant la commande ***pgrep***(1), c'est encore plus automatisable.

```
darkstar:~$ ps -o cmd -o rss -o vsz $(pgrep httpd)
CMD                          RSS    VSZ
/usr/sbin/httpd -k restart  33456  84816
/usr/sbin/httpd -k restart  33460  84716
/usr/sbin/httpd -k restart  33588  84472
/usr/sbin/httpd -k restart  30424  81608
/usr/sbin/httpd -k restart  33104  84900
/usr/sbin/httpd -k restart  33268  85112
/usr/sbin/httpd -k restart  30640  82724
/usr/sbin/httpd -k restart  15168  67396
/usr/sbin/httpd -k restart  33180  84416
/usr/sbin/httpd -k restart  33396  84592
/usr/sbin/httpd -k restart  32804  84232
```

Dans cet exemple, un shell interne est exécuté, utilisant ***pgrep***, pour retourner les PID de tous les processus dont le nom de la commande comporte 'httpd'. Puis, ***ps*** affiche le nom de la commande, la taille de la mémoire résidente et la taille de la mémoire virtuelle.

Et enfin, ***ps*** peut aussi créer une arborescence des processus. Ceci vous montre quels processus ont des processus enfants. Tuer le parent d'un processus enfant tuera aussi ce dernier. Nous pouvons faire cela avec l'option *-ejH*

```
darkstar:~$ ps -ejH
... beaucoup de lignes supprimées ...
 3660  3660  3660 tty1     00:00:00   bash
29947 29947  3660 tty1     00:00:00     startx
29963 29947  3660 tty1     00:00:00       xinit
29964 29964 29964 tty7     00:27:11         X
29972 29972  3660 tty1     00:00:00         sh
29977 29972  3660 tty1     00:00:05           xscreensaver
29988 29972  3660 tty1     00:00:04           xfce4-session
29997 29972  3660 tty1     00:00:16             xfwm4
29999 29972  3660 tty1     00:00:02             Thunar
... beaucoup de lignes supprimées ...
```

Comme vous pouvez le voir, ***ps***(1) est un outil incroyablement puissant non seulement pour déterminer quels sont les processus actifs sur votre système, mais également pour apprendre beaucoup de choses importantes à leur propos.

Comme c'est le cas avec beaucoup de programmes, il existe souvent plusieurs outils pour faire le travail. Avec un résultat proche de ***ps -ejH***, mais plus laconique, on trouve ***pstree***(1). Il affiche l'arborescence des processus, de manière un peu plus visuelle.

```
darkstar:~$ pstree
init-+-atd
     |-crond
     |-dbus-daemon
     |-httpd---10*[httpd]
     |-inetd
     |-klogd
     |-mysqld_safe---mysqld---8*[{mysqld}]
     |-screen-+-4*[bash]
     |        |-bash---pstree
     |        |-2*[bash---ssh]
     |        `-bash---irssi
     |-2*[sendmail]
     |-ssh-agent
     |-sshd---sshd---sshd---bash---screen
     `-syslogd
```

kill et killall

Gérer des processus ne limite pas à savoir quels sont actifs, mais également communiquer avec eux

pour changer leur comportement. La façon la plus courante de gérer un programme est de le tuer. De fait, l'outil pour ce travail est nommé ***kill***(1). Malgré son nom, ***kill*** ne tue pas les processus mais leur envoie un signal. Le signal le plus courant est SIGTERM, qui indique au processus de finir sa tâche en cours et de se terminer. Il existe une variété d'autres signaux qui peuvent être envoyés, mais les trois plus courant sont SIGTERM, SIGHUP et SIGKILL.

La manière dont un processus réagit en recevant un signal peut varier. La plupart des programmes s'arrêteront (ou tenteront de s'arrêter) chaque fois qu'ils reçoivent un signal, mais il y a peu de de différences importantes. Pour commencer, le signal SIGTERM informe le processus qu'il doit s'arrêter dès que possible. Cela donne le temps au processus de finir ses actions importantes, tel qu'écrire des données sur le disque, avant de se fermer. Au contraire, le signal SIGKILL dit au programme de se terminer immédiatement, sans autre question. Ceci est particulièrement utile pour tuer les processus qui ne répondent plus et est parfois appelé la *balle d'argent*. Certains processus (en particulier des daemons) capturent le signal SIGHUP et rechargent leurs fichiers de configuration à chaque fois qu'ils le reçoivent.

Afin d'envoyer un signal à un processus, la première chose dont nous avons besoin est son PID. Vous pouvez obtenir cela facilement avec ***ps*** comme cela à déjà présenté. Pour envoyer différents signaux à un processus actif, vous devez simplement passer le numéro de signal et *-s* en tant qu'option. L'option *-l* liste tous les signaux disponibles et leurs numéros. Vous pouvez également envoyer des signaux par leur nom avec *-s*.

```
darkstar:~$ kill -l
 1) SIGHUP   2) SIGINT     3) SIGQUIT     4) SIGILL
 5) SIGTRAP  6) SIGABRT     7) SIGBUS     8) SIGFPE
 9) SIGKILL 10) SIGUSR1    11) SIGSEGV    12) SIGUSR2
13) SIGPIPE 14) SIGALRM    15) SIGTERM    16) SIGSTKFLT
... many more lines omitted ...
darkstar:~$ kill 1234 # SIGTERM
darkstar:~$ kill -s 9 1234 # SIGKILL
darkstar:~$ kill -s 1 1234 # SIGHUP
darkstar:~$ kill -s HUP 1234 # SIGHUP
```

Parfois, vous souhaiterez terminer tous les processus portant un certain nom. Vous pouvez tuer les processus par leur nom avec ***killall***(1). Utilisez simplement les même options pour ***killall*** que celles que vous utiliseriez avec ***kill***.

```
darkstar:~$ killall bash # SIGTERM
darkstar:~$ killall -s 9 bash # SIGKILL
darkstar:~$ killall -s 1 bash # SIGHUP
darkstar:~$ killall -s HUP bash # SIGHUP
```

top

Jusqu'à maintenant nous avons appris comment voir quels sont les processus actifs à un moment donné, mais comment faire si nous voulons les surveiller pendant une période plus longue ? ***top***(1) nous permet de faire cela. Il affiche une liste ordonnée des processus du système, avec leur informations essentielles, et la met à jour périodiquement. Par défaut, les processus sont triés par leur taux d'utilisation du CPU et les mises à jour se font toutes les trois secondes.

```
darkstar:~$ top
top - 16:44:15 up 26 days,  5:53,  5 users,  load average: 0.08, 0.03, 0.03
Tasks: 122 total,   1 running, 119 sleeping,   0 stopped,   2 zombie
Cpu(s):  3.4%us,  0.7%sy,  0.0%ni, 95.5%id,  0.1%wa,  0.0%hi,  0.2%si,
0.0%st
Mem:   3058360k total,  2853780k used,   204580k free,   154956k buffers
Swap:        0k total,        0k used,        0k free,  2082652k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    1 root      20   0  3928  632  544 S    0  0.0   0:00.99 init
    2 root      15  -5     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.82 migration/0
    4 root      15  -5     0    0    0 S    0  0.0   0:00.01 ksoftirqd/0
    7 root      15  -5     0    0    0 S    0  0.0   0:11.22 events/0
    9 root      15  -5     0    0    0 S    0  0.0   0:01.19 work_on_cpu/0
   11 root      15  -5     0    0    0 S    0  0.0   0:00.01 khelper
  102 root      15  -5     0    0    0 S    0  0.0   0:02.04 kblockd/0
  105 root      15  -5     0    0    0 S    0  0.0   1:20.08 kacpid
  106 root      15  -5     0    0    0 S    0  0.0   0:01.92 kacpi_notify
  175 root      15  -5     0    0    0 S    0  0.0   0:00.00 ata/0
  177 root      15  -5     0    0    0 S    0  0.0   0:00.00 ata_aux
  178 root      15  -5     0    0    0 S    0  0.0   0:00.00 ksuspend_usbd
  184 root      15  -5     0    0    0 S    0  0.0   0:00.02 khubd
  187 root      15  -5     0    0    0 S    0  0.0   0:00.00 kseriod
  242 root      20   0     0    0    0 S    0  0.0   0:03.37 pdflush
  243 root      15  -5     0    0    0 S    0  0.0   0:02.65 kswapd0
```

La page de manuel comporte des détails pratiques sur comment interagir avec ***top*** tel que modifier l'intervalle de rafraîchissement, la manière dont les processus sont triés ou même comment tuer un processus directement depuis ***top***.

cron

Bien, nous avons étudié différents moyen de voir les processus actifs de notre système et comment leur envoyer des signaux, mais comment faire pour lancer un processus de manière régulière ? Heureusement, Slackware comporte ce qu'il faut, ***crond***(8). cron lance des processus pour chaque utilisateur selon le programme qu'il aura établi. Cela rend les choses très pratique pour les processus qui doivent s'exécuter de manière périodique, mais qui ne demandent pas à être “démonisés”, comme pour les scripts de sauvegarde. Chaque utilisateur peut gérér sa propre entrée dans la base de données de cron, les utilisateurs non-root peuvent dont lancer leurs processus planifiés également.

Pour pouvoir lancer un programme depuis cron, vous aurez besoin de ***crontab***(1). La page de manuel liste un large choix de façon de faire cela, mais la méthode la plus courante est d'utiliser l'option *-e*. Cela verrouillera l'entrée de l'utilisateur dans la base de données de cron (pour éviter d'être accédés en écriture par un autre programme) puis ouvrira cette entrée avec l'éditeur de texte indiqué par la variable d'environnement VISUAL. Sur les systèmes Slackware, c'est typiquement l'éditeur ***vi***. Vous pouvez avoir besoin de consulter le chapitre sur ***vi*** avant de continuer.

Les entrées de la base de données de cron peuvent paraître un peu archaïques au premier regard,

mais elles sont très souples. Chaque ligne non commentée est traitée par ***crond*** et la commande indiquée est lancée si les horaires correspondent.

```
darkstar:~$ crontab -e
# Keep current with slackware
30 02 * * * /usr/local/bin/rsync-slackware64.sh 1>/dev/null 2>&1
```

Comme mentionné précédemment, la syntaxe des entrées de cron est un peu difficile à comprendre au début, regardons chaque partie indépendamment. De la gauche vers la droite, les différentes sections sont : minutes, heures, jour, mois, jour de la semaine et commande. Une astérisque `*` indique toutes les minutes, heures, jours et ainsi de suite. Dans l'exemple ci-dessus, la commande est *"/usr/local/bin/rsync-slackware64.sh 1>/dev/null 2>&1"* et elle s'exécute tous les jours de la semaine, chaque semaine et chaque mois à 2 h 30 du matin.

crond enverra également un courrier électronique à l'utilisateur local avec la sortie générée par la commande. Pour cette raison, beaucoup de tâches ont leur sortie redirigée vers `/dev/null`, un périphérique spécial qui supprime directement ce qu'il reçoit. Afin de vous souvenir plus facilement de ces règles, vous souhaiterez peut-être coller le texte suivant en commentaire dans votre propre entrée de cron.

```
# Rediriger la sortie vers /dev/null :
#   1>/dev/null 2>&1
#
# MINUTE HEURE JOUR MOIS JOUR DE LA SEMAINE COMMANDE
```

Par défaut, Slackware comporte un certain nombre d'entrée et de commentaires dans la crontab de root. Ces entrée facilitent la mise en place de tâches d'administration système périodiques selon les répertoires correspondants dans `/etc`. Tout script placé dans ces répertoires sera exécuté chaque heure, jour, semaine ou mois. Les noms sont assez significatifs [1)] : `/etc/cron.hourly`, `/etc/cron.daily`, `/etc/cron.weekly` et `/etc/cron.monthly`.

Navigation

Chapitre précédent : BASH ou le Bourne Again Shell

Chapitre suivant : Le système X Window

Sources

- Source originale : http://www.slackbook.org/beta

- Publication initiale d'Alan Hicks, Chris Lumens, David Cantrell, Logan Johnson
- Traduction initiale de escaflown
- Traduction de Ellendhel

slackbook, process control, ps, kill, killall, top, cron

1)

en anglais *hourly* : horaire, *daily* : quotidien, *weekly* : hebdomadaire, *monthly* : mensuel

From:
https://docs.slackware.com/ - **SlackDocs**

Permanent link:
https://docs.slackware.com/fr:slackbook:process_control

Last update: **2012/12/29 15:23 (UTC)**