Control de procesos

Los sistemas Slackware a menudo ejecutan cientos o miles de programas, cada uno de los cuales se conoce como un proceso. El manejo de estos procesos es una parte importante de la administración de sistemas. Entonces, ¿cómo manejamos exactamente todos estos procesos por separado?

ps

Los primeros pasos en manejar procesos es examinar qué procesos se están ejecutando actualmente. La herramienta más popular y poderosa para esto es ***ps***(1). Sin ningún argumento, ***ps*** no le brindará mucha información. Por defecto, solo le indica qué procesos se están ejecutando en su shell activa actualmente. Si queremos más información, tendremos que profundizar sobre este tema.

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

Aquí se puede ver los procesos que están corriendo actualmente en su shell o terminal activa y solo se incluye alguna información. El PID es el *“ID del proceso”*; a todos los procesos se les asigna un número único. El parámetro TTY le dice a qué dispositivo terminal se añade el proceso. Naturalmente, CMD es el comando que estaba corriendo. Sin embargo, usted podría estar un poco confundido con el parámetro TIME, ya que parece moverse muy lentamente. Esta no es la cantidad de tiempo real que el proceso ha estado ejecutándose, sino la cantidad de tiempo de CPU que el proceso ha consumido. Un proceso inactivo no usa virtualmente tiempo de CPU, por lo que este valor no puede aumentar rápidamente.

Ver solo nuestros propios procesos no es muy divertido, así que es posible ver todos los procesos con el argumento *-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
... muchas más líneas han sido omitidas ...
```

Los ejemplos anteriores usan la sintaxis estándar de ***ps***, pero mucha más información puede se descubierta si usamos la sintaxis BSD. Para hacerlo, debemos utilizar el argumento *aux*.

Esto es distinto del argumento *-aux*, pero en la mayoría de los casos los dos argumentos son equivalentes. Esto es una reliquia de décadas. Para más información, vea la página man para ***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]
... muchas más líneas han sido omitidas ....
```

Como puede ver, la sintaxis BSD ofrece mucha más información, incluyendo qué usuario controla los procesos y qué porcentaje de RAM y CPU está consumiendo cuando ***ps*** está corriendo.

Para lograr una parte de esto, en función del proceso, ***ps*** permite que se proporcionen una o más IDs de proceso (PIDs) en la línea de comandos, y posee el argumento '-o' para mostrar un atributo particular del PID.

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

Lo que se muestra es la PID del nombre del comando (cmd) y su tiempo transcurrido (etime). El PID en este ejemplo, es una variable shell para el PID de la shell actual. Como se puede observar en este ejemplo, el proceso de shell ha existido durante 12 minutos, 22 segundos.

Usando el comando ***pgrep***(1), esto puede ser más automatizable.

```
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
```

En este ejemplo, una ejecución de sub-shell usando ***pgrep*** devuelve los PIDs de los procesos cuyos

comandos incluyen 'httpd'. Entonces, ***ps*** muestra el nombre del comando, el tamaño de la memoria residente y de la memoria virtual.

Finalmente, ***ps*** puede también crear un árbol de procesos. Esto muestra qué procesos tienen procesos hijos. Terminar un proceso padre que posee hijos también termina los procesos hijos que posea. Se puede hacer esto empleando el argumento *-ejH*.

```
darkstar:~$ ps -ejH
... many lines omitted ...
 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
... muchas más líneas han sido omitidas ...
```

Como se observa, ***ps***(1) es una herramienta increíblemente poderosa para determinar no solo que procesos están actualmente activos en tu sistema, si no que también, permite obtener una gran cantidad de información importante acerca de los mismos.

Como es el caso con muchas de las aplicaciones, existen varias herramientas para el trabajo. Similar a la salida de ***ps -ejH***, pero de una forma más concisa, está el comando ***pstree***(1). Este comando muestra el árbol de procesos un poco más visual.

```
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 y killall

La gestión de los procesos no solo consiste en saber cuáles se están ejecutando, sino también en comunicarse con ellos para cambiar su comportamiento. La forma más común de administrar un

programa es terminarlo. Por lo tanto, la herramienta para el trabajo se llama ***kill***(1). A pesar del nombre, ***kill*** en realidad, no termina los procesos, sino que les envía señales. La señal más común es un SIGTERM, que le dice al proceso que termine lo que está haciendo y termine. Hay una variedad de otras señales que pueden enviarse, pero las tres más comunes son SIGTERM, SIGHUP y SIGKILL.

Lo que hace un proceso cuando recibe una señal varía. La mayoría de los programas terminarán (o intentarán terminar) cada vez que reciban una señal, pero hay algunas diferencias importantes.

Para empezar, la señal SIGTERM informa a el proceso que debe terminarse a sí mismo lo antes posible. Esto le da tiempo al proceso para finalizar cualquier actividad importante, como escribir información en el disco, antes de que se cierre. En contraste, la señal SIGKILL le dice al proceso que se termine de inmediato, sin preguntas. Esto es más útil para matar procesos que no responden y, en ocasiones, se denomina *"bala de plata"*. Algunos procesos (particularmente demonios) capturan la señal SIGHUP y re cargan su archivo de configuración cada vez que la reciben. Para señalar un proceso, primero necesitamos saber cual es el PID. Esto se puede obtener con ***ps*** como se discutió previamente. Para enviar diferentes señales a un proceso en ejecución, simplemente pase el número de señal y *-s* como un argumento. El argumento *-l* lista todas las señales que puede elegir y sus números. También puede enviar señales por su nombre con *-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
```

A veces es posible que desee terminar todos los procesos en ejecución con un nombre determinado. Puede matar procesos por nombre con ***killall***(1). Solo pasa los mismos argumentos a ***killall*** que pasaría a ***kill***.

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

top

Hasta ahora hemos aprendido cómo mirar los procesos activos por un momento, pero ¿y si queremos monitorearlos durante un periodo de tiempo prolongado? ***top***(1) nos permite hacer esto precisamente. ***top*** muestra una lista ordenada de los procesos en su sistema, junto con información vital sobre ellos y actualizaciones periódicas. De forma predeterminada, los procesos están ordenados por su porcentaje de CPU y las actualizaciones se producen cada tres segundos.

```
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 página del manual tiene detalles útiles sobre cómo interactuar con ***top***, como cambiar su intervalo de retardo, se muestran los procesos de orden, e incluso cómo terminar los procesos desde ***top*** directamente.

cron

Bien, hemos aprendido muchas formas diferentes de visualizar los procesos activos en nuestro sistema y los medio de señalizarlo, pero ¿Y si queremos ejecutar un proceso periódicamente? Afortunadamente, Slackware incluye ***crond***(8). cron corre procesos para todos los usuarios en la fecha prevista. Esto lo hace muy útil para los procesos que deben ejecutarse periódicamente, pero no requieren una demonización completa, como por ejemplo los scripts de copia de seguridad.

Todos los usuarios tienen su propia entrada en la base de datos de cron. Por lo tanto, los usuarios que no tienen privilegios de administrador también pueden correr procesos periódicamente.

Con el fin de correr programas desde el cron, necesitará usar el ***crontab***(1). La página del manual enumera una variedad de formas de hacer esto, pero el método más común es empleando el argumento *-e*. Esto bloqueará la entrada del usuario en la base de datos de cron (para evitar que otro programa lo sobrescriba), luego abra esa entrada con cualquier editor de texto especificado por la variable de entorno VISUAL. En sistemas Slackware, típicamente el editor es ***vi***. Antes de continuar, es posible que usted necesite consultar el capitulo de ***vi***. Las entradas de la base de datos de cron pueden parecer un poco arcaicas al principio, pero son muy flexibles. Cada línea descomentada es procesada por ***crond*** y el comando especificado se ejecuta si todas las condiciones temporales se cumplen.

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

Como se mencionó anteriormente, la sintaxis para las entradas cron son difíciles de entender al principio, así que revisemos cada parte de forma individual. De izquierda a derecha, las diferentes secciones son: Minutos, Hora, Día, Mes, día de la semana y comando. Cualquier entrada asterisco `*` coincide con cada minuto, hora, día y así sucesivamente. Así que, a partir del ejemplo anterior el comando es *"/usr/local/bin/rsync-slackware64.sh 1>/dev/null 2>&1"*, se ejecuta diariamente a las 2:30 a.m.

crond también enviará un correo electrónico al usuario local con cualquier salida que genere el comando. Por está razón, muchas tareas tienen su salidas re-dirigidas a `/dev/null`, un dispositivo especial que inmediatamente descarta todo lo que recibe. Para que le resulte más fácil recordar estas reglas, puede pegar el siguiente texto comentado en la parte superior de sus propias entradas cron.

```
# Redirect everything to /dev/null with:
#   1>/dev/null 2>&1
#
# MIN HOUR DAY MONTH WEEKDAY COMMAND
```

Por defecto, Slackware incluye un número de entradas y comentarios en el crontab del administrador. Estas entradas facilitan la configuración periódica de las tareas del sistema al crear una serie de directorios en `/etc` correspondientes a la frecuencia con la que se deben ejecutar las tareas. Cualquier script colocado dentro de estos directorios se ejecutará cada hora, diariamente, semanalmente o mensualmente. Los nombres deben ser autoexplicativos: `/etc/cron.hourly`, `/etc/cron.daily`, `/etc/cron.weekly`, y `/etc/cron.monthly`.

Navegación de capítulos

Capítulo previo: The Bourne Again Shell

Próximo capítulo: The X Window System

Fuentes

- Fuente original: http://www.slackbook.org/beta
- Escrito originalmente por Alan Hicks, Chris Lumens, David Cantrell, Logan Johnson

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