


Capítulo 8

ADMINISTRACIÓN DE PROCESOS

Procesos

- Los procesos en Linux son ni más ni menos que programas que están corriendo en un momento dado.
- En nuestro sistema operativo el árbol de procesos está representado en el directorio **/proc**
- Cuando un proceso **A** inicia otro proceso **B**, se dice que **A** es el padre de **B** (o lo que es lo mismo, que **B** es hijo de **A**).
- Los **procesos normales** generalmente son ejecutados en una terminal y corren en el sistema operativo a nombre de un usuario.
- Los **procesos daemon** también se ejecutan a nombre de un usuario pero no tienen salida directa por una terminal, sino que corren en segundo plano. Generalmente los conocemos como **servicios**, y en vez de utilizar una terminal para *escuchar* por un requerimiento, lo hacen a través de un *puerto*.
- Los **procesos zombies** son aquellos que han completado su ejecución pero aún tienen una entrada en la tabla de procesos debido a que no han enviado la señal de finalización a su proceso padre (o este último no ha podido leerla).
  - Usualmente la presencia de este tipo de procesos indica un error en el diseño del programa involucrado.

El comando pstree

Si la utilizamos con la opción **-p**, podremos ver el **P**rocess **ID**entifier (o **identificador de proceso**, comúnmente llamado **PID**) de cada proceso. En este contexto, decimos que si el proceso **A** es el padre de **B**, el **PID** de **A** recibe el nombre de **PPID** (**P**arent **P**rocess **ID**) de **B**.

Si a continuación de la opción **-p** indicamos un PID dado, el árbol de procesos se mostrará comenzando por el proceso al que le corresponde dicho PID.

```
gacanepa@debian:~$ pstree -p
systemd(1)─┬─agetty(896)
           ├─apache2(1079)─┬─apache2(1092)
           │               ├─apache2(1093)
           │               ├─apache2(1094)
           │               ├─apache2(1095)
           │               └─apache2(1096)
           ├─atd(751)
           ├─blkmapd(458)
           ├─cron(750)
           ├─dbus-daemon(753)
           ├─exim4(1387)
           ├─lvmetad(470)
           ├─mdadm(673)
           ├─monit(993)───{monit}(997)
           ├─mysqld(991)─┬─{mysqld}(1066)
           │             └─{mysqld}(1067)
```

```
gacanepa@debian:~$ pstree -p 1079
apache2(1079)─┬─apache2(1092)
              ├─apache2(1093)
              ├─apache2(1094)
              ├─apache2(1095)
              └─apache2(1096)
gacanepa@debian:~$
```

El comando ps

- El uso de las opciones **-ef** o **aux** (esta última sin guión medio al comienzo) hace que podamos visualizar rápidamente el listado de procesos que están corriendo en nuestro sistema.
- Una característica distintiva de **ps** es que no es interactivo, sino que saca una *foto* de los procesos que están corriendo en un determinado momento.
- Sin opciones, **ps** nos devuelve los procesos ligados a la terminal actual.
- *El uso de las opciones a, u, y x de manera separada devuelve una lista de 1) todos los procesos que se están ejecutando en una terminal, 2) muestra el estado de los procesos del usuario actual y qué cantidad de recursos está requiriendo cada uno, y 3) indica la información de los demonios y procesos sin terminal. Al combinar estas opciones como aux podemos acceder a todos esos datos simultáneamente.*

El comando ps (Cont.)

- **USER**: usuario dueño del proceso.
- **PID**.
- **%CPU**: porcentaje de tiempo de CPU utilizado sobre el tiempo que el proceso ha estado en ejecución.
- **%MEM**: porcentaje de memoria física utilizada.
- **VSZ**: memoria virtual del proceso medida en KiB.
- **RSS** (**R**esident **S**et **S**ize) es la cantidad de memoria física no *swappeada* que la tarea ha utilizado (en KiB)
- **TT**: terminal asociada al proceso. Si en este campo vemos un signo de pregunta (**?**) significa que el proceso en cuestión se trata de un servicio que no está utilizando ninguna terminal.
- **STAT**: código de estado. Una **R** en este campo indica que el proceso se está ejecutando. Una **S** nos dice que está *durmiendo* esperando a que suceda un evento.
- **STARTED**: fecha y hora de inicio del proceso.
- **TIME**: tiempo de CPU acumulado.
- **COMMAND**: comando relacionado con el proceso, incluyendo todos sus argumentos.

```
gacanepa@debian:~$ ps aux | head -n 2
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.2  1.3 139220  6892 ?        Ss   19:57   0:00 /sbin/init
gacanepa@debian:~$
gacanepa@debian:~$ ps a | head -n 2
  PID TTY      STAT   TIME COMMAND
  896 tty1     Ss+    0:00 /sbin/agetty --noclear tty1 linux
gacanepa@debian:~$
gacanepa@debian:~$ ps u | head -n 2
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
gacanepa  1402  0.0  0.9  21248  4952 pts/0    Ss   19:58   0:00 -bash
gacanepa@debian:~$
gacanepa@debian:~$ ps x | head -n 2
  PID TTY      STAT   TIME COMMAND
 1394 ?        Ss     0:00 /lib/systemd/systemd --user
gacanepa@debian:~$
gacanepa@debian:~$ ps -ef | head -n 2
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 19:57 ?        00:00:00 /sbin/init
gacanepa@debian:~$
```

El comando ps (Cont.)

- Es importante notar que ps nos permite adaptar la cantidad y el orden de las columnas a mostrar, e inclusive nos deja ordenar el resultado utilizando una de ellas de manera ascendente o descendente. Eso es posible mediante las opciones **-eo** y **--sort**, respectivamente. La primera debe ser seguida por la lista de campos a mostrar (tomados de la sección **STANDARD FORMAT SPECIFIERS** del man page)

  La segunda debemos colocar un signo igual y el criterio de ordenamiento (**-%mem** indica ordenar por uso de memoria en forma descendente):

```
ps -eo pid,ppid,cmd,%cpu,%mem --sort=-%mem
```

```
gacanepa@debian:~$ ps -eo pid,ppid,cmd,%cpu,%mem --sort=-%mem
  PID  PPID CMD                         %CPU %MEM
  991     1 /usr/sbin/mysqld             0.0 14.5
 1079     1 /usr/sbin/apache2 -k start   0.0  5.1
  846     1 php-fpm: master process (/e  0.0  4.8
 1106     1 /usr/sbin/smbd               0.0  3.0
 1423  1106 /usr/sbin/smbd               0.0  2.6
    1     0 /sbin/init                   0.0  1.3
 1392   874 sshd: gacanepa [priv]        0.0  1.3
 1092  1079 /usr/sbin/apache2 -k start   0.0  1.2
 1093  1079 /usr/sbin/apache2 -k start   0.0  1.2
 1094  1079 /usr/sbin/apache2 -k start   0.0  1.2
 1095  1079 /usr/sbin/apache2 -k start   0.0  1.2
```

kill y killall

- Matar un proceso significa interrumpir la normal ejecución del mismo
- Para evitar errores, antes de matar procesos es preferible identificar aquellos que se verían afectados por nuestra medida. Por ejemplo, antes de matar el proceso con PID 3092 es una buena idea identificarlo primero mediante **ps** y la opción **--pid**.
- Para identificar y matar varios procesos de una sola vez:
  - Identifiquemos aquellos procesos que comparten el padre cuyo PID es 1576:

    ```
    pgrep -l -P 1576
    ```

En el caso de que existan varios procesos que compartan el mismo nombre y deseemos detenerlos a todos, podemos hacer uso del comando **killall**

- Antes de reemplazar **pgrep** por **pkill**, preguntémonos si realmente deseamos matar todos esos procesos. Si la respuesta es sí, podemos hacer

  ```
  pkill -P 1576
  ```

- Si en vez de realizar la identificación por PPID quisiéramos hacerlo a partir del propietario o del grupo dueño del proceso, simplemente deberemos reemplazar la opción **-P** con **-u** o **-G**, respectivamente, seguido del nombre o del identificador del usuario o grupo.

Señales

Cuando empleamos kill para matar un proceso, se le envía al mismo una señal. En otras palabras, le estamos mandando una indicación para modificar su funcionamiento normal. Los distintos tipos de señales se enumeran a través de

```
kill -l
```

```
gacanepa@debian:~$ 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   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
gacanepa@debian:~$
```

Las señales más utilizadas son **1** (**SIGHUP**), **9** (**SIGKILL**), y **15** (**SIGTERM**):

- Cuando se cierra la terminal asociada a uno o más procesos, se envía a los mismos la señal **SIGHUP**, lo que hace que se detengan.
- **SIGKILL** le indica a un proceso que debe finalizar de inmediato, sin darle tiempo a liberar adecuadamente los recursos que esté utilizando. Esta señal no puede ser ignorada por el proceso al que es enviada.
- Finalmente, **SIGTERM** le permite al proceso terminar su ejecución normalmente dándole la oportunidad de liberar los recursos utilizados. Aunque esto suene bien, cabe aclarar que esta señal puede ser ignorada por un proceso. Por eso puede ser necesario (como último recurso) utilizar **SIGKILL** en algunas ocasiones.

El comando top

- A diferencia de **ps**, **top** actualiza la información cada cierto tiempo (por defecto, cada tres segundos) y además provee otros datos útiles sobre el estado del sistema.

```
top - 20:55:23 up 57 min,  1 user,  load average: 0,00, 0,00, 0,00          1
Tasks: 161 total,   1 running, 160 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,3 us,  0,3 sy,  0,0 ni, 99,3 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem :   504360 total,   177732 free,   130820 used,   195808 buff/cache
KiB Swap:  1044476 total,  1044476 free,        0 used.   340536 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 1709 gacanepa  20   0   42824   3728   3084 R  0,3  0,7   0:00.06 top          2
    1 root      20   0  139220   6892   5088 S  0,0  1,4   0:01.10 systemd
    2 root      20   0       0      0      0 S  0,0  0,0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S  0,0  0,0   0:00.05 ksoftirqd/0
    5 root       0 -20       0      0      0 S  0,0  0,0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S  0,0  0,0   0:00.15 rcu_sched
    8 root      20   0       0      0      0 S  0,0  0,0   0:00.00 rcu_bh
    9 root      rt   0       0      0      0 S  0,0  0,0   0:00.00 migration/0
   10 root       0 -20       0      0      0 S  0,0  0,0   0:00.00 lru-add-drain
   11 root      rt   0       0      0      0 S  0,0  0,0   0:00.01 watchdog/0
   12 root      20   0       0      0      0 S  0,0  0,0   0:00.00 cpuhp/0
   13 root      20   0       0      0      0 S  0,0  0,0   0:00.00 kdevtmpfs
```

El comando top (Cont.)

- **Fila 1:**
  - **top** es el nombre del programa.
  - **20:55:23** es la hora actual.
  - **up 57 min** indica que el equipo ha estado encendido por 57 minutos.
  - **load average: 0.00, 0.00, 0.00** muestra la cantidad de procesos (en promedio) que están utilizando (o han estado esperando para utilizar) el CPU durante los últimos 60 segundos, 5 minutos, y 15 minutos, respectivamente.
- **Fila 2:**
  - **Tasks: 161 total, 1 running, 160 sleeping, 0 stopped, 0 zombie** muestra la cantidad de procesos que están corriendo actualmente en el sistema y sus posibles estados:
    - *running* indica la cantidad de procesos que están corriendo. En la sección indicada con el 2 se los identifica con la letra **R** en la columna **S** (de **S**tatus).
    - *sleeping* representa el número de procesos que no están corriendo actualmente pero que se encuentran esperando que ocurra un evento para despertarse (por ejemplo, una consulta al servidor web). Se indican con la letra **S** en la misma columna mencionada anteriormente.
    - *stopped* son aquellos procesos que han sido detenidos. Se identifican con la letra **T**.
    - Los procesos zombie se indican con la letra **Z**

El comando top (Cont.)

- **PID**: Process ID.
- **USER**: Usuario dueño del proceso.
- **PR**: Prioridad de ejecución del proceso.
- **NI**: Es un valor que refleja la prioridad sobre el uso de los recursos del sistema otorgada a cada proceso en particular.
- **VIRT**: Cantidad de memoria virtual que el proceso está manejando.
- **RES**: Es la representación más cercana a la cantidad de memoria física que un proceso está utilizando.
- **SHR**: La cantidad de memoria compartida (potencialmente con otros procesos) disponible para este proceso en particular, expresada en KiB.
- **S**: Estado del proceso.
- **%CPU**: Es el porcentaje de tiempo de CPU utilizado por el proceso desde el último refresco de pantalla.
- **%MEM**: Indica el porcentaje correspondiente al uso de memoria física de un proceso en particular.
- **TIME+**: Tiempo de CPU que ha utilizado el proceso desde que inició, expresado en *minutos:segundos.centésimas* de segundo.
- **COMMAND**: Muestra el comando que se utilizó para iniciar el proceso o el nombre de este último.

El comando top (Cont.)

- Como toda herramienta de la línea de comandos, top posee numerosas opciones. Entre las más útiles podemos destacar las siguientes:
- Refrescar la pantalla cada X segundos:

```
top -d X
```

- Ordenar la salida por uso de RAM (descendente):

```
top -o %MEM
```

- Monitorear sólo los procesos cuyos PIDs son 324, 697, y 25216:

```
top -p 324,697,25216
```

Los comandos nice y renice

- Las columnas **PR** y **NI** en la salida de top indican la prioridad que se le ha asignado a un proceso en particular -tal como lo ve el kernel en un momento dado- y el valor de *niceness* del mismo.
- **PR** y **NI** están relacionados entre sí: mientras **mayor** sea la prioridad de un proceso, es **menos *nice*** (del inglés ***bueno***) ya que consumirá más recursos del sistema.
- El valor de *niceness* puede ubicarse en algún lugar del intervalo comprendido entre -20 y 19. Estos dos límites representan la mayor y la menor prioridad posibles, respectivamente.
- Para modificar la prioridad de un proceso en ejecución, utilizaremos el comando **renice**. Los usuarios con privilegios limitados solamente pueden *aumentar* el valor de niceness correspondientes a un proceso del cual son dueños, mientras que root puede modificar este valor para cualquier proceso sin importar quién sea el usuario dueño del mismo.
- Cambiar la prioridad del proceso con PID 324 (opción -p) a 10: `renice -n 10 -p 324`
- Cambiar la prioridad de todos los procesos del usuario (opción -u) **alumno** a -10: `renice -n -10 -u alumno`
- Por defecto, cualquier nuevo proceso se ejecuta con una prioridad igual a 0. Si deseáramos iniciarlo con una prioridad diferente, podemos hacer uso del comando **nice** seguido de la opción **-n**, del nuevo valor de *niceness* deseado, y del comando a ejecutar.
- Por ejemplo, iniciemos top con un valor de *niceness* igual a 10: `nice -n 10 top`

Administración de procesos

systemctl es una herramienta provista como parte de **systemd**:

- **Iniciar** el servicio: `systemctl start miservicio.service`
- Configurarlo para que **inicie al arrancar el equipo**: `systemctl enable miservicio.service`
- **Detenerlo**: `systemctl stop miservicio.service`
- **Impedir** que inicie al arrancar el equipo: `systemctl disable miservicio.service`
- **Reiniciar** el servicio: `systemctl restart miservicio.service`
- Averiguar si está configurado para arrancar al inicio: `systemctl is-enabled miservicio.service`
- Averiguar si está corriendo actualmente: `systemctl is-active miservicio.service` o `systemctl -l status miservicio.service` (esta última variante provee más información sobre el estado y la operación del servicio).

Redirecciones y tuberías

Todos los procesos para poder lanzarse necesitan tener lo que se conoce como *entrada estandar* (más comúnmente llamado por su nombre en inglés **stdin**) y devuelven como resultado dos archivos que son capturados por la terminal en la cual estamos trabajando: la *salida estándar* (**stdout**) y el *error estándar* (**stderr**).

Como las salidas **stdout** y **stderr** son archivos, al fin y al cabo, los podemos trabajar como tales e incluso capturarlos por la terminal o (y esto es lo más interesante) también pueden direccionarse a otros archivos en disco para su posterior inspección.

Existe además otra forma de trabajar con las salidas, y es transformar la salida **stdout** de un comando en **stdin** de otro. Para eso utilizamos el símbolo **|**, conocido como **pipe** o **tubería**.

Procesos en primer y Segundo plano

- En algunos casos, un script puede tomar un tiempo considerable en completar su ejecución, o un programa puede ocupar la línea de comandos mientras se encuentre corriendo. Para permitirnos volver a tomar el control de la terminal en cuestión, la shell nos permite colocar procesos en segundo plano. También podemos iniciar directamente un proceso en segundo plano, de manera que no ocupe la terminal mientras corre.
- Para que un proceso se inicie en segundo plano, colocaremos el símbolo & al fin del mismo.
- Al iniciar un proceso en segundo plano, la terminal queda libre para seguir trabajando en la misma y se nos provee la identificación de dicho proceso y su PID.
- Si deseamos traer un proceso a primer plano, nos valdremos el comando **fg** seguido del identificador de este. Si sucediera que no supiéramos el identificador del proceso en cuestión, podemos valernos del comando **jobs** para averiguarlo.

El comando nohup

- La ejecución de un proceso se verá interrumpida si cerramos la terminal desde la que lo iniciamos, o si cerramos la sesión actual. Para evitar que esto suceda, podamos usar nohup, una herramienta que permite iniciar un proceso que sea inmune a la situación que describimos anteriormente.
- *El comando **nohup** es especialmente útil cuando estamos conectados a un equipo de manera remota y deseamos ejecutar un script que continúe su ejecución luego de que cerremos la terminal asociada. De esta manera, no tenemos que estar logueados durante todo el tiempo que corra el script.*
- La sintaxis del uso de esta herramienta es la siguiente:

```
nohup [programa a iniciar] &
```

- Como podemos ver, generalmente **nohup** se utiliza en conjunto con el símbolo **&** para enviar simultáneamente el programa en cuestión a segundo plano, aunque no es estrictamente necesario que lo hagamos de esa manera.