Preparación para el examen LPI 101

Tema 103.5

Creando, monitorizando y matando procesos

Créditos y licencia de uso

Coordinación:
Manuel Guillán (xLekOx) lpi@xlekox.org

Traducción:
Dani Donisa (kasei) kasei@flashmail.com
Manuel Guillán (xLekOx) lpi@xlekox.org
Kiefer Von Jammo (Kiefer) kiefer@khrooon.net

Maquetación:
Manuel Guillán (xLekOx) lpi@xlekox.org
Kiefer Von Jammo (Kiefer) kiefer@khrooon.net
Oscar Casal (ocs) oscar@glug.es

Versión 1.1 (09-08-2004 16:30)

ÍNDICE

Índice de contenido

Tema 103.5

Introducción

En éste capítulo veremos como administrar los procesos. Esto incluye saber ejecutar procesos en primer y segundo plano, cambiarlos de plano, monitorizar procesos actuales, ordenarlos según varios parámetros, enviar señales a los procesos y matar procesos innecesarios para el sistema/usuario.

Los comandos que se verán en este tema son:

- &
- bg
- fg
- jobs
- kill
- nohup
- ps
- top

Así mismo se harán ejercicios sobre los mismos al final del tema, que serán muy parecidos a los realizados en los exámenes.

Este tema tiene un peso (importancia) de 5 de cara al examen final de la certificación LPI 101. El total de la suma de pesos de todos los temas es de 106.

Que es un proceso

Un proceso es, en resumen, un programa en ejecución.

Todos los comandos ejecutados a lo largo del curso acaban de una forma u otra, por generar un proceso que realiza la tarea. La shell que usamos para interactuar con el sistema también es un proceso. De hecho, la shell es un buen ejemplo para entender las relaciones de parentesco entre procesos.

Cuando se ejecuta un comando en la shell, primero comprueba si es uno de sus comandos internos. Por ejemplo export:

```
$ export
```

export es un comando interno de la shell, no necesita la ejecución de ninguna utilidad externa a la misma shell para llevar a cabo la tarea.
Otro ejemplo, el comando que nos muestra la fecha:

```
$date
```

En caso de no ser un comando interno, la shell ejecuta el comando necesario para llevar a cabo la tarea, invocando un nuevo proceso. Este nuevo proceso se le conoce como proceso hijo (en relación a nuestra shell) y la propia shell es el proceso padre (de ese proceso hijo).

Cuando el proceso ha acabado de realizar su tarea, devuelve los resultados al proceso padre y finaliza su ejecución. El proceso hijo puede necesitar invocar otro proceso donde delegar parte de la tarea, convirtiéndose a su vez, en el proceso padre del proceso recién invocado. En los sistemas multitarea como GNU/Linux, los procesos pueden llegar a desarrollar auténticos árboles genealógicos, ya que los procesos pueden se padres e hijos y además, varios procesos pueden ser hijos del mismo proceso padre a la vez.

Los demonios

De los diferentes procesos que pueden estar ejecutándose en un sistema GNU/Linux, esto es, procesos invocados por el usuario, procesos invocados por otros usuarios y procesos invocados por el sistema operativo, los últimos son los más especiales y se les conoce como demonios (daemons).

Los demonios son procesos que el sistema operativo invoca para proporcionar servicios. Una característica de los demonios es que no suelen interactuar mucho. Proporcionan la funcionalidad para la que están programados de manera silenciosa, al menos en cierta forma, puesto que la gran mayoría utilizan ficheros de log para registrar sus transacciones. Entre los demonios más comunes están los de impresión, los de correo, los de periodicidad de tareas y monitorización. Pero hay muchos más.

Trabajando con ps

El comando ps da información sobre los procesos corriendo sobre el sistema. Invocado de la manera más sencilla:

```
$ ps
```

Devuelve un listado con los procesos que se lancen con el usuario actual y que aún se están ejecutando. En la última línea aparecerá el proceso que representa a *ps*, ya que es el último proceso lanzado.

```
  PID TTY          TIME CMD
18621 pts/24   00:00:00 bash
18628 pts/24   00:00:00 ps
```

Echando un vistazo al resultado:

- La primera columna, marcada como PID, representa el identificador de proceso (Process ID, PID). Este identificador de proceso se asigna desde cero hasta un límite marcado por el propio sistema operativo. Cuando el identificador llega a ese límite, vuelven a asignarse los identificadores libres empezando otra vez desde cero.
- La segunda columna indica la terminal asociada al proceso. Hay procesos que no tienen asignada terminal (demonios...), estos aparecen marcados con un signo de interrogación (?) en esta columna.
- La tercera columna indica el porcentaje de tiempo de procesador que el proceso está usando. Normalmente, los procesos ejecutan sus tareas muy rápidamente, durante intervalos cortos. Luego se mantienen a la espera. Un proceso que muestre lecturas altas en esta columna puede repercutir en el rendimiento del resto del sistema.
- La cuarta columna es el nombre del proceso (el nombre del comando). La primera línea es siempre la shell sobre la que se ejecutan los procesos. A esta shell (de hecho al primer proceso del usuario) se le llama líder de sesión (session leader).

A ps se le llama frecuentemente de las siguientes formas:

```
$ ps -a
```

Devuelve el listado de todos los procesos sin incluir al líder de sesión ni los procesos sin terminal asociada.

```
$ ps -e
```

Devuelve el listado de todos los procesos. La opción -e es sinónimo de la opción -A.

```
$ ps -l
```

Devuelve un listado extendido. Con información sobre UID's (identificadores de usuario o User ID's, marcado como UID), los PID's del proceso padre (marcado como PPID), información sobre si el proceso entra en las expectativas de planificación de procesos del kernel (marcado como C), o la fecha de inicio del proceso (marcado como TIME). Hay que destacar que en este formato, el nombre del proceso se sustituye por la cadena entera entrada con la que se lanzó el proceso, con parámetros, opciones y argumentos (marcado como CMD).

Por último:

```
$ ps -u root
```

Devolverá todos los procesos que ha lanzado el usuario especificado tras el parámetro -u (en el ejemplo debe sacar información sobre los procesos invocados por root o por el sistema).

Trabajando con pstree y top

Existen dos comandos relacionados con el comando *ps* que ofrecen una vista de los procesos ligeramente diferente a como lo hace *ps*.

El primero de ellos es *pstree*, que ofrece una visualización gráfica (aunque en modo texto) de las relaciones que existen entre los procesos:

```
$ pstree
init-+-amd
     |-atd
     |-cron
     |-6*[getty]
     |-httpd.apache—2*[httpd.apache]
     |-inetd
     |-kdewizard
     |-kflushd
     |-klogd
     |-kpiod
     |-kswapd
     |-kwmsound
     |-lockd
     |-mdrecoveryd
     |-rpc.portmap
     |-rpc.rstatd |-rpciod
     |-sendmail
     |-sh—rc.gui—kdm-+-X
     |                '-kdm—kwm-+-kaudioserver
     |                          |-kbgndwm
     |                          |-kfm—kvt—bash-+-awk
     |                          |              |-cat
     |                          |              |-paste
     |                          |              |-pstree
     |                          |              '-vi
     |                          |-kpanel
     |                          |-krootwm
     |                          '-kwmsound
     |-syslogd
     '-update
```

Se puede observar a través del gráfico los cinco procesos bajo el shell del usuario, así como bajo que proceso se ejecutó el shell, y también que proceso es padre y que proceso es hijo.

El segundo comando relacionado con *ps* es *top*. Éste comando, no sólo muestra los procesos actuales, sino que automáticamente se va actualizando para mostrar los cambios acontecidos. Adicionalmente, en la parte superior se muestra información sobre el número de días que ha estado la maquina en marcha, el número de usuarios, la memoria, estadísticas de la memoria de intercambio, etc.

Mientras el comando *top* está en marcha, se pueden usar las siguientes teclas para interactuar con él:

h	Ayuda
q	Salir
s	Cambia el tiempo entre actualizaciones (por defecto, 5 segundos)
espacio	Actualizar ahora en lugar de esperar al siguiente intervalo de actualización
u	Muestra un único usuario

Finalizando un proceso

Bajo circunstancias normales, un proceso hijo actúa bajo el padre que lo ha creado. Cuando el proceso hijo ya no es necesario, desaparece. Algunas veces, sin embargo, los procesos se convierten en procesos 'fugitivos', y aunque no sea necesario que se sigan ejecutando, continúan su ejecución consumiendo recursos innecesarios.

Un proceso padre no puede (y no debe) finalizar su ejecución mientras tenga procesos hijos asociados a él que estén en funcionamiento. Teniendo ésto en cuenta, cuando un proceso hijo no puede finalizar correctamente su ejecución, origina que el proceso padre se quede en un estado inconsistente, y que no pueda, a su vez, terminar su ejecución, quedando el proceso padre (y el o los hijos 'colgados') en un estado conocido como 'zombie', haciendo uso de recursos innecesarios del sistema.

Un ejemplo para entender todo esto: el shell de un usuario ejecuta un proceso (A), que no puede hacer todo por si mismo, así que ejecuta otro proceso (B), que a su vez ejecuta otro proceso (C).
Pueden suceder entonces varias cosas:

- Bajo condiciones normales, cuando el proceso C termina su ejecución, se lo notifica al proceso B, y desaparece (C). El proceso B trata la información, notifica los datos al proceso A, y muere (B). El proceso A, hace lo propio con los datos recibidos, y retorna la información al shell del usuario, y entonces muere (A).

- En condiciones anormales, supongamos que el proceso C, después de pasar la información al proceso padre (el proceso B), no muere. Continua ejecutándose, lo que impide que el proceso B finalice, dado que tiene un proceso hijo (C) en marcha. El proceso B trata la información y la reporta hacia el proceso padre (A), que a su vez, devuelve la información hacia el shell que lo originó.
  Tanto el proceso A como el proceso B, no pueden finalizar su ejecución dado que tienen procesos hijos en marcha. Así pues, un error en el proceso C, que hace que se quede en ejecución cuando no debería, origina que haya tres procesos en marcha en el sistema, consumiendo recursos de forma innecesaria.

- Otro tipo de problema, podría darse de la siguiente manera: el proceso C, como antes, entra en un estado inestable, y no finaliza su ejecución. Aun así, el proceso B, acaba su ejecución y desaparece. El proceso A, también finaliza dado que su hijo, el proceso B, ha finalizado. Así pues, se queda únicamente el proceso C en marcha (en estado inestable), pero ahora, no tiene procesos padre a los que reportar.

Para resolver los problemas que pueden ocasionar estos procesos extraños, se puede usar el comando *kill*.

La sintaxis del comando *kill* es la siguiente:

kill {opcion} PID

Así por ejemplo, para acabar con el proceso *cat* la sentencia es:

```
$ ps -f
UID PID PPID C STIME TTY TIME CMD
root 19605 19603 0 Aug10 pts/0 00:00:34 bash
root 30089 19605 0 Aug20 pts/0 00:00:00 vi fileone
root 30425 19605 0 Aug20 pts/0 00:00:00paste -d fileone filetwo?
root 32040 19605 0 Aug22 pts/0 00:00:00 cat
root 1183 19605 0 Aug23 pts/0 00:00:00 awk -F: questions
root 30900 19605 0 14:25 pts/0 00:00:00 ps -f

$ kill 32040
```

Se pide 'amablemente' que el proceso acabe su ejecución. El termino 'amablemente' se usa dado que hay 32 posibles señales que se pueden enviar a un proceso para que acabe su ejecución, y ésta es la manera más elegante y más segura de acabar con un proceso, se pide que finalice de forma ordenada..
En muchas ocasiones, el proceso ignorará la petición de finalizar su ejecución. Cuando ésto suceda, se puede usar cualquiera de las otras 32 señales para acabar con el proceso.

Entre otras, algunas posibilidades son:
- 1 Colgar/desconectar (hangup/disconnect)
- 2 Usando la secuencia de interrupción (Ctrl+C)
- 3 Salir (quit)
- 9 Sin esperar, salir inmediatamente, matar (kill)
- 15 Por defecto, terminar

Para ver la lista de las posibles señales en el sistema, se usará el comando *kill -l*, y para conocer mejor cuales son las acciones que toman cada una de las opciones se hará uso del comando *man*:

```
$man kill
```

Suponiendo que el proceso *cat* no finaliza su ejecución, la secuencia de operaciones será:

```
  $ ps -f
UID PID PPID C STIME TTY TIME CMD
root 19605 19603 0 Aug10 pts/0 00:00:34 bash
root 30425 19605 0 Aug20 pts/0 00:00:00 paste -d fileone filetwo?
root 32040 19605 0 Aug22 pts/0 00:00:00 cat
root 1183 19605 0 Aug23 pts/0 00:00:00 awk -F: questions
root 30996 19605 0 14:25 pts/0 00:00:00 ps -f

$ kill 32040
$ ps -f
UID PID PPID C STIME TTY TIME CMD
root 19605 19603 0 Aug10 pts/0 00:00:34 bash
root 30425 19605 0 Aug20 pts/0 00:00:00 paste-d fileone filetwo?
root 32040 19605 0 Aug22 pts/0 00:00:00 cat
root 1183 19605 0 Aug23 pts/0 00:00:00 awk -F: questions
root 30998 19605 0 14:25 pts/0 00:00:00 ps -f
$ kill -9 32040
[3]- Killed
$ ps -f
UID PID PPID C STIME TTY TIME CMD
root 19605 19603 0 Aug10 pts/0 00:00:34 bash
root 30425 19605 0 Aug20 pts/0 00:00:00 paste-d fileone filetwo?
root 1183 19605 0 Aug23 pts/0 00:00:00 awk -F: questions
root 31000 19605 0 14:25 pts/0 00:00:00 ps -f
```

Se recomienda siempre para finalizar un proceso, intentar primero con la señal 15 (terminar) antes de usar la señal 9 (matar). También se recomienda que se compruebe que un proceso no tenga hijos antes de matarlo. Si existen procesos hijos, primero se deberá finalizar los procesos hijos antes de finalizar el proceso padre.

Segundo plano (background) y primer plano (foreground)

Cuando se ejecuta un proceso, por defecto se ejecuta en primer plano. Cuando un proceso se ejecuta en primer plano, se convierte en el único trabajo en el que puede trabajar el usuario (con el que puede interactuar), la interacción se basa entonces, en que se acabe éste trabajo.

Por ejemplo, cuando un usuario ejecuta el comando *ls -l*, se mostrará por pantalla el resultado, y hasta que no acabe el comando, no se podrá ejecutar ningún otro comando.

Para ejecutar un proceso en segundo plano (background), simplemente se ha de añadir al final del comando el signo ampersand (&). Esta opción permitirá ejecutar más de un comando a la vez:

```
$ sleep 90 &
[5] 31168
```

El número que aparece entre corchetes es igual al número de trabajos que se tienen actualmente ejecutándose en segundo plano. El número que le sigue (en éste caso, el 31168), es el número de proceso de éste trabajo.

El número de proceso del último trabajo puesto en segundo plano, también puede ser referenciado como $!.

Al poner el trabajo a ejecutarse en segundo plano, se permite al usuario poder seguir trabajando e iniciar otros procesos.

Si se quiere esperar a que acabe un proceso para iniciar otro, el comando *wait*, usado junto al numero de proceso (el que se quiere esperar), esperará a que acabe para poder seguir trabajando:

```
$ sleep 120 &
[5] 31175

$ wait 31175
```

El prompt no aparecerá hasta que el proceso 31175 termine su ejecución.

Trabajos en segundo plano

Para ver que trabajos se ejecutan en segundo plano, se usará el comando *jobs*:

```
$ jobs
[1] Stopped vi fileone (wd: ~)
[2]- Stopped paste -d' fileone filetwo ' (wd: ~)
[4]+ Stopped awk -F: questions (wd: ~)
[5] Done sleep 120
```

Los trabajos que han finalizado (3) no aparecerán, y los trabajos que acaban de finalizar (5) aparecerán una única vez (la próxima vez que se ejecute el comando *jobs*, 5 no aparecerá).
El signo más (+) seguido del número de trabajo entre corchetes, indica el trabajo (proceso) más reciente que se puede ejecutar o que se está ejecutando. El siguiente trabajo más reciente se indica con el signo menos (-). La información 'wd: ~' indica el directorio de trabajo del proceso.

La opción **-** añade el número de proceso (PID) a la salida del comando jobs, poniendo **-**p mostrará sólo el número de proceso (PID) de cada proceso y con la opción **-**n mostrará sólo los procesos que están suspendidos.

De ejecución en segundo plano a primero

Se puede mover un trabajo (proceso) que se esté ejecutando en segundo plano (background) a primer plano a través del comando *fg*.

La sintaxis del comando *fg* permite referenciar a un trabajo usando el signo de porcentaje (%) y el número de trabajo.
Por ejemplo, la siguiente secuencia de comandos lanza una espera (*sleep*) de 120 segundos, el comando se lanza en segundo plano, y luego se pasa a primer plano con el comando *fg*:

```
$ sleep 120 &
[5] 31206

$ fg %5
sleep 120
```

Tener en cuenta que el comando que se está ejecutando se muestra en pantalla al pasar a ejecutarse en primer plano.
Al igual que se usa %5, también se puede referenciar a los dos trabajos más recientes utilizando %+ y %- respectivamente.
Si no se sabe el número de trabajo (y no se recuerda el uso del comando jobs), se puede hacer referencia a un trabajo usando una parte de su nombre, usándolo después del signo de porcentaje y el signo de pregunta (?):

```
$ fg %?v
vi fileone
```

De ejecución en primer plano a segundo

El comando opuesto al usado para pasar trabajos a primer plano, es el comando *bg*, que permite mover un trabajo desde primer plano (foreground) a segundo plano (background).

Antes de usar este comando, se deberá suspender la ejecución del trabajo (para volver a obtener el prompt del sistema).
Para suspender la ejecución, se pulsará la secuencia de teclas que corresponde a la señal de suspender un proceso, por defecto CTRL+Z.

Cuando se suspende la ejecución, el trabajo se para y no reanudará su ejecución hasta que se mueva a primer o a segundo plano:

```
$ sleep 180 {presionado de Ctrl+Z }
[5]+ Stopped sleep 180
```

Ejecutando el comando *bg*, se moverá el trabajo a segundo plano, y se cambiará el estado del trabajo a 'en ejecución'.

Manteniendo procesos al cerrar una sesión

Por defecto la mayoría de las shell's al cerrar una sesión envían la señal SIGHUP a los procesos que se ejecutan en segundo plano, provocando que se terminen. Para evitar este comportamiento existe el comando *nohup*, el cual funcionaría del siguiente modo:

```
$ nohup proceso &
```

De esta manera, al cerrar la sesión el proceso se seguiría ejecutando.

Preguntas TEST

1.¿Cual será la salida del comando *ps -ae*?

a. Sólo se muestran los procesos para el usuario actual, menos la sesión principal.
b. Sólo se muestran los procesos para el usuario actual, incluida la sesión principal.
c. Se muestran todos los procesos de todos los usuarios.
d. Se muestran todos los procesos de todos los usuarios, menos cualquier sesión principal.

2.¿Cual de los siguientes comandos, produce el mismo resultado que el comando *ps -e*?

a. ps -f
b. ps -A
c. ps -l
d. ps -u

3.¿Cual de los siguientes comandos ejecuta un proceso en segundo plano?

a. -
b. +
c. %
d. &

4.Estando a punto de dejar la oficina por hoy, hace un minuto que se lanzó un trabajo de compilación que puede durar horas en segundo plano. Cuando ésta compilación acabe, se necesita que otro proceso se ejecute para imprimir unos resultados.
¿Cual de los siguientes comandos se usará para realizar estas tareas?

a. bg {proceso}
b. fg {proceso}
c. wait $! ; {proceso}
d. sleep ; {proceso}

5.¿Cual de los siguientes, representa al trabajo más reciente, lanzado en segundo plano, tal como muestra el comando *jobs*?

a. [1] Running {proceso}
b. [1]- Running {proceso}
c. [1]% Running {proceso}
d. [1]+ Running {proceso}

6.Se quiere ver un listado completo de todos los procesos que hay actualmente en ejecucion, y guardar una copia en un fichero llamado 'procesos'. ¿Que comando, o conjunto de comandos se usaría para esto?

a. ps -f ; ps -f > procesos
b. ps -ef >> procesos
c. ps -ef | tee procesos
d. ps -ef ; tee procesos

Respuestas TEST

1. La respuesta correcta es la c. La opción *-a*, cuando se usa a solas, no muestra la sesión principal, pero con la opción *-e* se muestra todo. Es equivalente a la opción 'mostrar selectivamente' y 'mostrar todo'. Se pueden poner las dos opciones, pero la opción *-e* gana.

2. La respuesta correcta es la b. La opción *-e* se usa para mostrar cualquier cosa, y la opción *-A* se usa para mostrar todo. La opción *-f* (respuesta a) mostrará un listado completo, pero sólo para el usuario actual. La opción *-l* (respuesta c) mostrara un listado completo, pero de nuevo, sólo para el usuario actual. La opción *-u* (respuesta d), mostrara la información relativa al usuario.

3. La respuesta correcta es la d. El ampersand (&) lanza un trabajo en segundo plano. Las otras opciones se ignoran al ser usadas como carácter final en la linea de comandos.

4. La respuesta correcta es la c. El signo de dolar y el de admiración ($!) representan al último proceso ejecutado en segundo plano. Cuando se usan con el comando wait (espera), el proceso en segundo plano deberá acabar antes que el próximo proceso comience a ejecutarse. Todas las otras opciones son incorrectas para especificar que un proceso debe esperar a otro para ejecutarse.

5. La respuesta correcta es la d. El signo más (+) a la derecha del numero de trabajo indica el trabajo más reciente puesto en segundo plano. Las otras opciones son invalidas dado que no tienen nada que ver con el trabajo más reciente puesto en segundo plano.

6. La respuesta correcta es la c. El comando *tee* coge los datos que recibe y lo muestra por pantalla, y a la vez, lo guarda en un fichero. La respuesta a no muestra todos los procesos, y requiere dos procesos, en contra de uno sólo de la opción c. La respuesta b guarda los procesos en el fichero pero no lo muestra por pantalla. La opción d tiene una sintaxis incorrecta.

Bibliografia y enlaces recomendados

LPIC 1 Certification Bible (Bible) by Angie Nash, Jason Nash
John Wiley & Sons; Bk&CD-Rom edition (July 1, 2001) ISBN: 0764547720

LPI Linux Certification in a Nutshell by Jeffrey Dean
O'Reilly & Associates; 1st ed edition (May 15, 2001) ISBN: 1565927486

CramSession's LPI General Linux Part 1 : Certification Study Guide
CramSession.com; ISBN: B000079Y0V; (August 17, 2000)

Referencias Unix Reviews
http://www.unixreview.com/documents/s=7459/uni1038932969999/

Página LPI: www.lpi.org

Apuntes IBM: http://www-106.ibm.com/developerworks/edu/l-dw-linux-lpir21-i.html

Manuales GPL: http://www.nongnu.org/lpi-manuals/