Bibliografía

Sarwar, S. M.; Koretsky, R.; Sarwar, S. A. El libro de LINUX. Addison Wesley, 2005.

Sobell, Mark G. Manual práctico de Linux. Comandos, editores y programación Shell. Anaya Multimedia, 2008

Sánchez Prieto, S. UNIX y LINUX. Guía práctica (Tercera edición). RaMa, D.L. 2004.

Pons, N. Linux. Principios básicos de uso del sistema. Ediciones ENI, 2011.

1. CICLO DE VIDA DE UN PROCESO EN LINUX

En Linux el PCB es una estructura de datos denominada *task_struct (*definida en el fichero *include/linux/sched.h).* Entre la información contenida en esta estructura cabe destacar la siguiente:

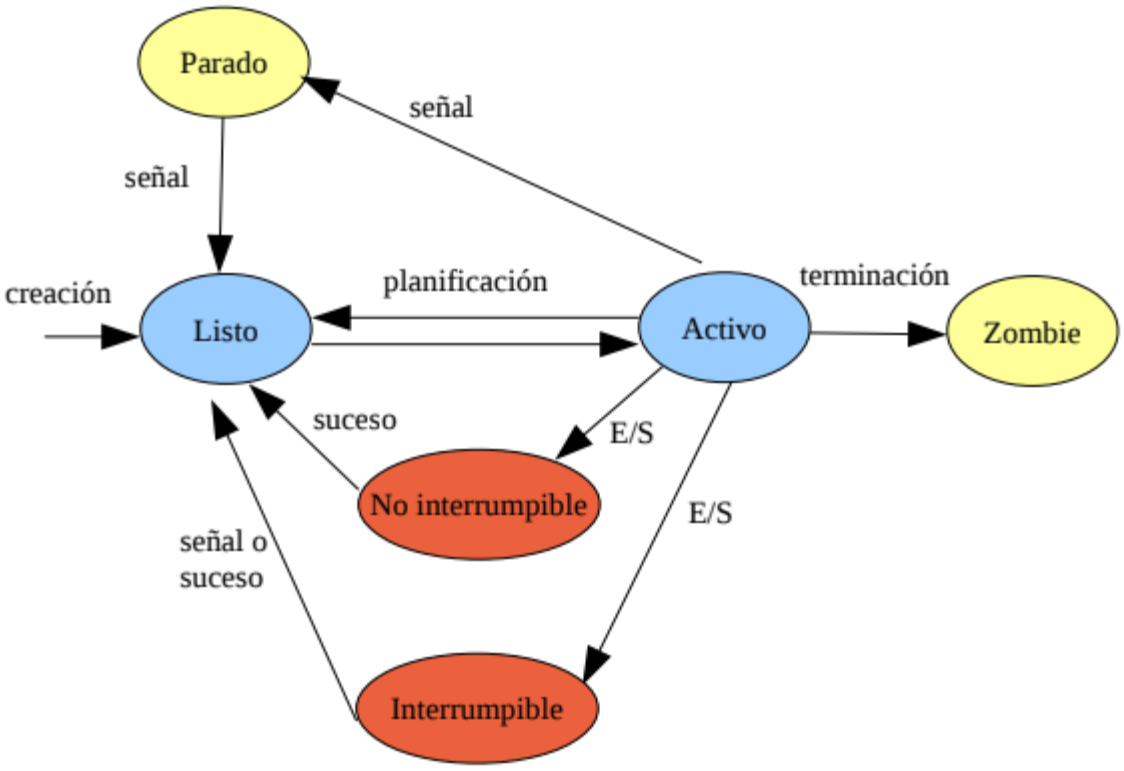
- **Estado** en el que se encuentra el proceso;
- **Información** necesaria para la **planificación** de procesos. Un proceso puede ser normal o de tiempo real. Los procesos de tiempo real se planifican antes que los procesos normales y se utilizan prioridades relativas dentro de cada categoría.
- **Identificadores**. Cada proceso tiene un único identificador de proceso (PID), un identificador de usuario y otro de grupo.
- **Vínculos**. Cada proceso incluye un vínculo con su proceso padre, vínculos con sus hermanos (procesos con el mismo padre) y vínculos con todos sus hijos.
- **Instante de creación** del proceso y cantidad de **tiempo del procesador** consumido hasta el momento por el proceso.
- **Punteros** a cualquier **fichero** abierto por el proceso.
- **Memoria virtual** asignada al proceso.
- **Contexto del proceso** que contiene la información de los registros y la pila específica del proceso.

Además, Linux mantiene una Tabla de Procesos implementada como una lista doblemente enlazada mediante punteros.

En un sistema Linux, durante la vida de los procesos éstos pueden pasar por diferentes estados. A continuación, se describe cada uno de ellos añadiendo entre paréntesis el carácter con que lo identifica algunos de los comandos de Linux con los que se gestionan los procesos:

- Activo o en ejecución (R). El proceso es ejecutado por el procesador.
- Interrumpible (S). Es un estado de bloqueo en el cual el proceso espera a que se complete un suceso, como la liberación de un recurso o la señal de otro proceso.
- No interrumpible (D). Es otro estado de bloqueo. La diferencia entre éste y el estado anterior es que en el estado No interrumpible un proceso espera directamente en una condición de hardware y por lo tanto no acepta señales.
- Parado (T). El proceso ha sido detenido por una intervención externa y sólo puede reanudarse por una acción positiva de otro proceso. Por ejemplo, un proceso puede estar en este estado durante la ejecución de un programa de depuración.
- Zombie (Z). El proceso ha terminado pero, por alguna razón, su PCB debe permanecer aún en la tabla de procesos.

Las transiciones entre los estados se muestran en la siguiente figura:



2. Jerarquía entre Procesos

En Linux, un proceso puede crear otro proceso. Se denomina **proceso hijo** al proceso creado y **proceso padre** al proceso creador.

Cuando se lanza una orden desde la línea de comandos, el shell (proceso padre) crea un nuevo proceso (proceso hijo) que se encarga de ejecutar la orden. Mientras que se ejecuta el proceso de la orden, el shell espera a que ést e termine.

La ejecución de un script (serie de órdenes almacenadas en un fichero) es ligeramente distinta a la ejecución de una orden desde la línea de comandos. En el caso de un script, el shell actual crea un shell hijo y hace que el shel l hijo ejecute las órdenes del script, una tras otra. Todas las órdenes del script se ejecutan de la misma forma que las órdenes procedentes de la línea de comandos, esto es, el shell hijo crea un proceso hijo por cada orden que se ejecuta. Mientras que el shell hijo está ejecutando órdenes del script, el shell padre espera a que el hijo finalice. Cuando el shell hijo concluye, el shell padre sale del estado de espera y reanuda su ejecución.

Al arrancar el sistema, el primer proceso que se ejecuta es el proceso de arranque, *init* (cuyo PID es 1), de quién depende el resto de los procesos. Cuando un usuario se conecta al sistema se lanza un proceso que ejecuta el shell de ese usuario (especificado en el fichero */etc/passwd*). Este proceso shell es hijo del proceso *int.* El shell hijo espera por un comando lanzando el proceso que ejecutará dicho comando. De esta forma, todos los procesos forman parte de un mismo árbol.

.

3. Comandos para el control de procesos

3.1. Obtención de información

Si se desea obtener información sobre los procesos existentes en el sistema, se pueden usar los siguientes comandos:

(a) *ps*. Este comando ofrece información sobre los procesos que existen en el momento de su ejecución.

La información dada es variada y depende del "shell" que se utilice, aunque básicamente ofrece el identificador del proceso (PID), el terminal desde el cual se inició el proceso (TTY), la cantidad de tiempo que el proceso lleva ejecutándose (TIME) y la línea de orden que se escribió para iniciar el proceso (COMMAND).

Si no se añade ningún parámetro, el comando *ps* muestra los procesos asociados al usuario conectado. Por otra parte, las opciones más básicas son las siguientes:
◦ Si se emplea la opción *l* se obtiene más información como, el estado del proceso (S), el identificador del usuario que ha inicializado el proceso (UID, User Identification), el identificador del proceso superior (PPID) y la prioridad de ejecución del proceso (PRI) (Esta prioridad está asignada por el sistema operativo.).

◦ Con la opción *u* se lista la información de los procesos indicando el usuario al que pertenece así como porcentajes sobre la utilización de CPU y memoria que realiza cada uno de ellos.
◦ La opción *x* hace que se listen los procesos de todos los terminales y de todos los usuarios.

(b) *pstree*. Este comando muestra la relación entre los procesos (padre e hijo), al mostrar el árbol de procesos.

Usando el parámetro *p* se obtiene el PID de cada proceso y con el parámetro *a* los argumentos de las líneas de comandos. Si sólo interesa información de un determinado proceso será necesario ejecutar el comando *pstree* seguido del PID asociado a dicho proceso.

(c) *top*. Este comando permite controlar a intervalos regulares la evolución de los procesos y el consumo de recursos del sistema (memoria, procesador, E/S). El comando es interactivo, basta con teclear *q* para salir y se puede acceder a la ayuda pulsando *h*.

(d) *htop*. Este comando es una versión mejorada del comando *top*, ya que añade funcionalidades, colores, soporte para ratón, etc. No obstante, es posible que este comando no esté instalado en el sistema. Para instalarlo será necesario teclear en la línea de comandos la siguiente orden:

sudo aptget install htop

al introducirla pedirá la password.

Al ejecutar este comando aparece una ventana dividida en tres partes. La parte superior izquierda resume, mediante barras indicadoras horizontales, el consumo total de CPU, memoria RAM y memoria de intercambio (swap). Arriba, a la derecha, se muestra una breve estadística del sistema y sus procesos.

La segunda parte corresponde a un listado de todos los procesos actualmente en ejecución en el sistema, ordenados, por defecto, de mayor a menor por su consumo de CPU. Existen más de diez columnas disponibles de información. Haciendo click en cualquiera de los títulos de las distintas columnas, la información que muestra se organiza por el criterio seleccionado.

La barra turquesa de esta segunda parte permite navegar entre los procesos, tanto vertical como horizontalmente mediante las flechas del teclado. Si se pulsa la barra espaciadora se colorea de amarillo el proceso seleccionado, facilitando su seguimiento. Si se pulsa la tecla t se despliega un árbol de procesos. Por último, si se teclea el carácter h, se obtiene la ayuda y los atajos asociados a este comando.

La tercera parte corresponde a un menú que incluye opciones para pedir ayuda, realizar búsquedas, enviar señales a los procesos, desplegar la lista en forma de árbol y la opción de salida, entre otros.

3.2. Procesos en Primer y Segundo Plano

En Linux, existen dos formas de ejecutar un trabajo:
1. De **forma interactiva** (***foreground*** o en **primer plano**). El proceso lanzado desde el terminal monopoliza el terminal, por lo que en principio, no se podrá ejecutar ningún otro programa a la vez. En definitiva, funciona casi como un diálogo de preguntas y respuestas: se introduce una orden y se espera a obtener el resultado de su ejecución antes de introducir la siguiente.
2.Con **prioridad subordinada** también denominada **tarea de fondo** (***background*** o en **segundo plano**). Al lanzar un programa como tarea de fondo, éste no monopoliza el terminal desde el que se lanzó, permitiendo al usuario interactuar con el shell al mostrar de nuevo el prompt sin esperar a obtener el resultado de la ejecución de dicho programa.
Normalmente se lanzan en primer plano aquellos programas que necesitan interacción con el usuario y que dicha interacción se realiza a través del terminal. Cuando se requiere bastante tiempo de CPU y poca interacción con el usuario, se lanzan como tarea subordinada.
Además, los procesos en segundo plano se ejecutan con menor prioridad que los procesos en primer plano.

Para ejecutar un trabajo (mandato) como tarea de fondo, sólo hay que concluir dicho mandato con el carácter especial *&*.Al solicitar que se ejecute un mandato de esta forma, inmediatamente después se presentan dos números: uno entre corchetes que indica el número de tarea que ocupa ese trabajo entre los que se ejecutan como tarea de fondo, y un segundo número que corresponde al identificador (PID) del proceso que ejecuta dicho trabajo.

Este identificador interesa tenerlo en cuenta, porque es el número que permite conocer después cómo se desarrolla el proceso en plena actividad. Se puede observar que el shell visualiza a renglón seguido el prompt de espera de nueva orden.

Por consiguiente, es posible ejecutar varios mandatos sucesivos en tarea de fondo, sin dejar el modo interactivo del shell. Pero no es recomendable excederse, puesto que se debilitaría la eficiencia del sistema y se perdería las ventajas de ejecutar de este modo.

Sólo el trabajo en primer plano puede utilizar la entrada del teclado. Para conectar el teclado a una tarea que se e stá ejecutando en segundo plano, es necesario usar el comando *fg* seguido del número de la tarea o el comando empleado para lanzar el trabajo. Si se omite dicho número o comando, la orden *fg* conecta el teclado al último trabajo lanzado como tarea de fondo. En vez de usar este comando se puede usar el signo de porcentaje (%) que siempre conecta el teclado con el último trabajo subordinado. En ambos casos, el shell mostrará el comando utilizado para comenzar la tarea y se podrá introducir cualquier entrada que el programa necesite para seguir.

Cuando se ejecuta un programa en primer plano, es posible, que se necesite suspender para volver al shell, hacer algo en él y volver después al proceso suspendido. Esta detención del proceso en primer plano se obtiene mediante la combinación de teclas *Control+Z*. Automáticamente, el sistema mostrará un identificador de trabajo (número entero) con el que se podrá gestionar el proceso posteriormente.

Concretamente, para poner de nuevo en primer plano un proceso suspendido se empleará el comando *fg* seguido del identificador de trabajo.

En el caso de que se omita el identificador de trabajo, se reanudará el último proceso suspendido.

Si lo que se desea es ejecutar en segundo plano el proceso detenido (mediante Control+Z), se usa el comando *bg* seguido del identificador del trabajo. De igual forma que con el comando *fg*, se omite el identificador de trabajo,l a orden *bg* pasará a ejecutar en segundo plano el último proceso suspendido.

El comando *jobs* permite conocer el estado de los procesos suspendidos y en segundo plano. Si no se especifica u na lista con los identificadores de trabajo, el comando *jobs* muestra el estado de todas las tareas en curso. Con la opción l, también se muestra el PID de los procesos asociados a esas tareas.

La tecla de interrupción (normalmente *Control*+*C*) no puede abortar la ejecución de un proceso que se esté ejecutando en segundo plano.Será necesario usar el comando *kill* seguido del PID del proceso a abortar o seguido de un signo de porcentaje con el número de trabajo correspondiente. El resultado de ejecutar este coman do no proporciona ningún mensaje indicando la finalización de un proceso.

Realmente el comando *kill* manda señales a los procesos. La señal por defecto (cuando no se especifica ninguna señal) es la señal SIGTERM (15) que termina con el proceso que recibe dicha señal. Si se desea enviar otra señal distinta es necesario especificar, como parámetro del comando *kill*, el número que representa esa señal o la constante entera que la identifica. Es posible que un proceso pueda ignorar determinadas señales, en concreto la señal SIGINT, por lo que el proceso no finalizará aunque se le envíe esa señal. No obstante, existen señales que n o se pueden ignorar, por ejemplo la señal SIGKILL (9) que elimina drásticamente al proceso que la recibe.
La opción *l* del comando *kill* muestra todas las señales aceptadas por el sistema.

El comando ***killall*** es parecido al comando *kill*, con la diferencia de que en vez de indicar el PID o el número de trabajo, se usa el nombre del programa, lo que afectará a todos los procesos que estén asociados a ese programa. Se aconseja usar la opción *i* de este comando para que pida confirmación con cada uno de los proceso s seleccionados.

3.3. Prioridad de los Procesos

Cuando se lanzan trabajos como tareas de fondo, a menudo algunos de ellos no urgen. Para tratar este tipo de procesos utilizaremos el comando *nice*.

El comando *nice* se encarga de decirle al sistema cuáles son los trabajos no prioritarios.

Estos tardarán en ejecutarse, pero no molestarán a los restantes trabajos en curso, especialmente aquellos que se realizan en modo interactivo y que deben favorecerse siempre.

El comando *nice* se antepone al comando al que queremos "quitarle urgencia".

Este comando también permite asignar a un proceso una prioridad NICE concreta. Para conocer la prioridad que tiene asignado un proceso es necesario usar la opción *l* del comando *ps* y observar la columna *NI*.

El rango de asignación de prioridad disponible es de 20 a 19, siendo 20 la prioridad más alta y 19 la más baja. La sintaxis de este comando es:

nice n niceness comando

Si *niceness* es positivo el comando se ejecutará con menor prioridad y si es negativo (sólo puede hacerlo el administrador) se ejecutará con mayor prioridad.

El comando *nice* sin argumentos devuelve la prioridad por defecto (normalmente 0).
Así como el comando *nice* establece la prioridad de un proceso cuando se inicia su ejecución, el comando *renice* permite alterarla en tiempo real, sin necesidad de detener el proceso. Su sintaxis es la siguien

te: renice niceness PID

Cuando un usuario se desconecta del sistema (usando el comando exit), sus tareas son abortadas hayan o no finalizado estás.
No obstante, es posible que el usuario desee que una tarea determinada continúe con su ejecución independientemente de si se desconecta o no.
Si la tarea ha sido lanzada en primer plano, será necesario previamente suspenderla (*Control + Z*) y pasarla a segundo plano mediante el comando *bg*.

Si la tarea ya ha sido lanzada en segundo plano, estos pasos previos no serán necesarios.

Una vez que la tarea se está ejecutando en segundo plano basta con usar el comando *dosown* para que el proceso continúe sin depender del shell, por lo tanto, pudiendo desconectar el usuario sin perder su ejecución.

En definitiva, el comando disown desasocia un proceso en segundo plano del proceso shell. Su sintaxis es:

disown %identificador_tarea

Como resultado de ejecutar este comando se obtendrá un mensaje avisando de la desasociación realizada.
Pero el usuario abra otro terminal el comando jobs no le proporcionará información sobre la ejecución .
Pero, si se desea disponer información de la ejecución de la tarea desde otro terminal será necesario usar la opción *h* del comando *disown*.

Las tareas de fondo que se hayan creado, desaparecerán cuando el usuario se desconecte del sistema. Sería intere sante que dichos trabajos pudieran continuar. Esto es bastante útil, especialmente si un trabajo requiere considerable tiempo.

Para ello se utiliza el mandato *nohup*, que permite la ejecución de tareas después de desconectarse del sistema. Se debe utilizar con trabajos subordinados, ya que si no, no podríamos finalizar la sesión. Su sintaxis es:

nohup comando &

La principal diferencia entre los comandos *nohup* y *disown* es que el primero se debe usar a la hora de lanzar la tarea de fondo, mientras que el segundo permite. una vez lanzada una tarea de fondo, solicitar que se siga ejecutando independientemente de la desconexión del terminal.