

Tema 5 - Gestión de procesos

ÍNDICE

1. Introducción

2. Gestión de Procesos

Procesos

Estados de un proceso

Threads o hilos

Multitarea vs multihilo

Planos de Ejecución de un proceso

3. Comandos Linux para Gestión de Procesos

3.1 Comandos generales

Comando `ps`

Comando `pstree`

Comando `top`

Comandos `kill` y `killall`

3.2 Comandos para gestionar los planos de ejecución de un proceso

Operador `&` (ejecución background)

Comando `bg`

Comando `fg`

`Ctrl-C`

`Ctrl-Z`

4. Ejecución de procesos y uso de comandos encadenados (`;` `&&` `||`)

Operador `;`

Operador `&&`

Operador `||`

5. Gestión de Memoria

6. Comandos Linux para Gestión de Memoria

Comando `free`

7. Algunos Procesos Importantes

`cron`

`init`

`systemd`

8. Ficheros de Arranque del sistema

1. Introducción

En este tema trataremos los conceptos básicos para la gestión de procesos en linux (planos de ejecución, estados,, etc) así como algunos aspectos básicos de la gestión de memoria.

2. Gestión de Procesos

Procesos

Un proceso es un concepto manejado por el sistema operativo y que **referencia un programa en ejecución**.

Cuando se ejecuta un programa en Linux, se crea al menos un proceso con todo el contexto de ejecución para dicho programa. Cada uno de esos procesos necesita que el SO le asigne tiempo de CPU, memoria, y acceso al sistema de archivos y a los dispositivos de E/S. *Por tanto, un programa y un proceso son entidades distintas.*

- En Linux, que es un SO multiusuario y multitarea, *podemos ejecutar varias veces un mismo programa, y cada una de estas instancias en ejecución es un proceso diferente*. Es decir, múltiples instancias de un mismo programa pueden ejecutarse simultáneamente.

Cada vez que un programa se convierte en proceso, es decir, cada vez que se ejecuta un programa, además de ubicar en memoria las instrucciones que lo componen y sus datos asociados, a dicho proceso se le asocia una estructura de datos. Esta es única para cada proceso, identifica el proceso respecto de los demás y sirve para controlar su correcta ejecución. Es gestionada por el sistema operativo. Es lo que se llama el **bloque de control del proceso o BCP (ó PCB en inglés)**.

Una de las tareas del sistema operativo es aislar a cada uno de los procesos entre sí, de tal forma que los procesos sean completamente independientes unos de otros; esto se hace con la finalidad prevenir que compartan información (memoria) entre ellos, lo cual pueda dar pie a errores. Es por ello que en esencia, los programas no pueden acceder a la información y datos de otros programas en ejecución.

Cuando un proceso finaliza, ya sea de forma natural o no, quizás por algún error, será el sistema operativo el encargado de liberar el espacio en memoria.

Normalmente los sistemas Linux son muy estables, pero alguna vez ocurrirá algo inesperado, y puede ocurrir que un proceso se quede colgado, o puede ocurrir que queramos enviar alguna señal a un proceso en ejecución. Veremos más adelante como hacerlo mediante el envío de señales.

Estados de un proceso

Básicamente los estados posibles de un proceso son:

- **En ejecución:** El procesador está ejecutando instrucciones del programa que lo compone y tiene concedido el tiempo de uso de la UCP en un instante concreto.
- **Preparado:** en espera o activo. Un proceso está preparado para ser ejecutado; es decir, está esperando turno para poder utilizar su intervalo de tiempo y poner en funcionamiento sus instrucciones accediendo a los recursos del sistema.
- **Bloqueado:** El proceso está retenido; es decir, está bloqueado debido a causas múltiples. Una de estas causas puede ser que dos procesos utilicen el mismo fichero de datos, otra causa puede ser que dos

procesos utilicen la unidad de DVD al mismo tiempo para cargar datos,...

NOTA: Los estados de los procesos en cada sistema operativo pueden cambiar para adaptarse a su gestión. Lo indicado anteriormente tiene como objetivo referencia didáctica.

Cada programa en ejecución, es decir, **cada proceso, tiene un identificador que lo diferencia de los demás**. Cada proceso tiene un número asignado por el sistema operativo que sirve precisamente para identificar el proceso, lanzarlo a ejecución, detenerlo, cancelarlo, reanudarlo, etc. **Este identificador de proceso se nombra con la abreviatura PID (Process ID)**. El comando en Linux para ver los procesos en el sistema es **"ps"**.

```
egg@UTADLR$ ps
```

```
# El resultado será algo similar a esto (hemos numerado las líneas):
```

PID	TTY	TIME	CMD
1978	pts/0	00:00:00	bash
6766	pts/0	00:00:00	ps

Siendo cada uno de los valores que se muestran:

- PID: Identificador del proceso.
- TTY: interfaz del terminal en el que se está ejecutando el proceso
- TIME: Tiempo de ejecución.
- CMD: Comando/proceso/programa que está en ejecución en ese proceso

Los procesos los lanzan normalmente otros procesos. Es decir, que **cada proceso que se lanza a ejecución depende, en la mayoría de los casos, de otro proceso denominado proceso padre PPID (Parent Process ID)**. Así, al nuevo proceso lanzado se le denomina proceso hijo.

Threads (hilos)

Un proceso tendrá siempre un hilo, en la que corre el propio programa, pero puede tener más hilos o threads. **Podemos definir un thread o hilo como una secuencia de instrucciones que el SO puede programar para su ejecución**. A diferencia de un proceso, los threads son entidades mucho más pequeñas, lo cual los hace fáciles de gestionar, tanto es así que **un thread es la unidad más pequeña a la cual un procesador puede asignar tiempo de ejecución**.

A diferencia de los procesos, los cuales podríamos decir que "viven" dentro del sistema operativo, **los threads "viven" dentro de los procesos**. Un thread se crea, ejecuta y finaliza dentro de un proceso, dicho en otras palabras: Un thread le pertenece a un proceso, y, a su vez, un proceso puede poseer múltiples threads.

Resumiendo:

- **Un proceso no es más que la instancia de un programa, el cual es creado y controlado por el sistema operativo**. Los procesos son entidades independientes y no pueden compartir información entre ellos. Para que un proceso pueda ser ejecutado es necesario que este posea por lo menos un thread.

- Por otro lado, **un thread es la unidad más pequeña a la cual un procesador puede asignar tiempo.** Los threads poseerán la secuencia más pequeña de instrucciones a ejecutar. Los threads se crean, ejecutan y mueren dentro de los procesos, siendo capaces de compartir información entre ellos.

Con los threads y los procesos seremos capaces de implementar la programación concurrente, y, dependiendo de la cantidad de procesadores la programación en paralelo.

Multitarea vs multihilo

Aclarar lo primero que es común usar el término **tarea** (*task*) para referirse a un *proceso*.

Un sistema multitarea permite ejecutar múltiples programas y tareas al mismo tiempo, si además es multihilo, el sistema ejecuta múltiples subprocesos (threads) de procesos iguales o diferentes, al mismo tiempo.

Para implementar ambos mecanismos **la CPU tiene que ir dedicando su tiempo entre los diferentes programas para que se ejecuten simultáneamente o entre los diferentes subprocesos.** Ya hemos dicho que los procesos no comparten los recursos que va asignando la CPU (memoria, registros, disco, etc) mientras que los subprocesos sí.

¿Cómo logra Linux simular la multitarea? ¿Cómo es posible, de esta forma, que podamos estar ejecutando varias aplicaciones «simultáneamente»?

Lo que ocurre en realidad es que el kernel del sistema operativo va repartiendo rodajas de tiempo de procesamiento (ciclo) a cada proceso, es decir, ****el S.O. asigna tiempo de CPU a un proceso, después retira la CPU de dicho proceso y se lo asigna a otro proceso y así sucesivamente.**** Este intercambio se lleva a cabo con una frecuencia altísima que permite simular una multitarea real a nivel de usuario. Eso conlleva el cambio de contexto de los procesos y que dichos procesos pasen por los estados que hemos indicado. Además, esto tiene relación directa con las prioridades de que tienen asignadas los distintos procesos y los algoritmos de planificación.

Planos de Ejecución de un proceso

Un proceso puede ejecutarse en primer plano (**foreground**) o en segundo plano (**background**).

Un programa lanzado en **foreground desde una terminal monopoliza dicha terminal, es decir, ya no podemos ejecutar ningún otro programa a la vez desde dicha terminal.** Tenemos que esperar a que finalice su ejecución para poder lanzar otro comando. Por el contrario, **la ejecución de un programa en background, deja libre la terminal desde la que se lanzó y el shell nos vuelve a mostrar el prompt.**

En foreground se lanzan los programas que no tardan mucho en ejecutarse y los programas que necesitan interacción con el usuario a través del teclado. Sin embargo, si necesitamos ejecutar un programa gráfico (por ej. gedit) desde la terminal, lo normal es lanzarlo en background, para que esta quede libre y podamos seguir ejecutando otros comandos; También se suelen ejecutar en segundo plano los programas cuya ejecución va a tardar bastante tiempo y además no necesitan ninguna interacción con el usuario.

Servicios o demonios

Hay procesos que están diseñados para ejecutarse en *background*, son los denominados **servicios** aunque es común llamarlos **demonios**. Los procesos que se ejecutan en segundo plano no pueden recibir ningún input o control por parte del usuario/terminal, es decir, carecen de interfaz para proporcionar mayor seguridad ya que algunos de estos servicios son críticos para el correcto funcionamiento del sistema.

Los **nombres de los servicios suelen terminar en "d"** (de ahí "daemons"), por ejemplo el servicio **sshd**.

Existen muchos servicios en un sistema Linux, aquellos que comprueban si existen actualizaciones pendientes de instalar o configurar, aquellos que monitorizan el sistema, los que controlan versiones de drivers con respecto a las versiones de su fabricante, los que controlan el reloj del sistema, los que gestionan la conectividad de la red, etc.

Aunque en un sistema Linux en funcionamiento existen multitud de servicios en ejecución, su administración es relativamente sencilla. Básicamente su administración, se centra en las operaciones de:

- Iniciar el servicio
- Comprobar su estado
- Reiniciar el servicio
- Parar el servicio.

Existen servicios que requieren otras operaciones, pero en general estas son las más usadas.

Indicar, que **para trabajar con los servicios se necesitan ciertos privilegios por lo que es necesario ejecutar operaciones con el comando sudo** previamente. El comando sudo permite a los usuarios no root ejecutar comandos que normalmente requerirían privilegios de superusuario.

Algunas consultas para conocer los servicios que están corriendo en el sistema son:

```
# Utilizaremos el comando systemctl

$sudo systemctl list-unit-files --type service --all
$dudo systemctl | grep running
```

Más adelante veremos más detalles y como manejar los planos de ejecución de un proceso.

3. Comandos Linux para Gestión de Procesos

3.1 Comandos generales

Comando ps

El comando **ps** **muestra una instantanea de todos los procesos que se están ejecutando en el sistema. Es una información estática**

Es habitual usarlo junto a *grep* para obtener información sobre un proceso en particular.

El comando **ps** **admite tres tipos de sintaxis en sus opciones, así como muchas otras opciones dentro de estas.** Los tres tipos de opciones son los siguientes:

- Opciones de Unix98 (**standard syntax**)- Estas opciones de un único carácter se pueden agrupar y van precedidas de un único guión (-).
- Opciones de BSD (Berkeley Software Distribution). (**BSD syntax**)- Estas opciones de un único carácter se pueden agrupar y no van precedidas por ningún guión.
- Opciones Proyecto GNU largas. (**GNU syntax**)- Estas son opciones multi-carácter y nunca van juntas. Están precedidas por dos guiones (--).

A continuación vemos algunas de las opciones de ps más importantes:

- a: muestra los procesos para **todos los usuarios** (excepto los que no están asociados al terminal en el que se ejecuta)
- u: muestra información **adicional**. Lista información del proceso como por ejemplo el usuario que lo está corriendo, la utilización de CPU y memoria, etc.
- x: muestra información sobre **procesos sin terminal** (típicamente demonios)
- f: muestra información sobre los **procesos y sus sub-procesos**
- e: muestra información **extendida**
- T: muestra los procesos lanzados desde **este terminal**

To see every process on the system using standard syntax (with -):

```
ps -e
ps -ef
```

To see every process on the system using BSD syntax (without -):

```
ps ax
ps axu
```

Veamos algunos ejemplos con las opciones más utilizadas:

```
# a: muestra los procesos para todos los usuarios (todos los terminales)
# u: lista información del proceso como por ejemplo el usuario que lo está corriendo, la utilización de CPU y memoria, etc.
# x: muestra información sobre procesos sin terminal (típicamente demonios)
ps aux
```

```
ps aux | more

# f: muestra información sobre los procesos y sus sub-procesos
ps faux

# e: muestra información extendida
ps e

# T: muestra los procesos lanzados desde este terminal
ps T

# Es común usar ps junto a grep o awk , para filtrar los resultados
ps Te | awk '/bash/ {print $1 "\t\t" $3 "\t\t" $4 "\t\t" $5}'

ps aux | grep cron

ps -ef | grep systemd
man ps
```

A continuación se detalla brevemente alguno de los campos útiles de la salida de *ps*:

Campo	Significado
UID	El Id del usuario propietario del proceso
PID	El Id. del proceso
PPID	El Id. del proceso Padre
C	Número de hijos del proceso
SZ	Tamaño de páginas de memoria de la imagen del proceso
RSS	Tamaño del conjunto residente, es la memoria física no intercambiada utilizada por el proceso.
TTY	El terminal desde el que fue lanzado el proceso, si no hay terminal aparece entonces un '?'
STAT	El estado del proceso
START (STIME)	Hora de inicio del proceso
TIME	El tiempo de CPU consumido por el proceso
COMMAND	El comando que fue ejecutado (y que generó al proceso)

En cuanto al campo *STAT*, los valores más comunes son:

Valor	Significado
D/U	Dormido, ininterrumpible. El proceso está en estado de espera (normalmente espera alguna operación de E/S), y no puede ser interrumpido (<i>dormant</i>)

Valor	Significado
I	En espera (o <i>idle</i>). El proceso está en ejecución, pero está esperando su turno (para que sea ejecutado por el planificador de la CPU o <i>scheduler</i>)
R	En ejecución (<i>running</i>)
S	Dormido, pero puede ser interrumpido (<i>sleeping</i>)
T	Detenido, debido a una señal de control
Z	Proceso <i>zombie</i> . El proceso ha terminado, pero no ha sido "limpiado" por su proceso padre porque no ha detectado que murió.

Por último, los valores del campo *STAT* suelen tener además algún indicador extra:

Indicador	Significado
<	Tarea de alta prioridad (relacionado con <i>nice</i>)
N	Tarea de baja prioridad (también relacionado con <i>nice</i>)
L	El proceso tiene páginas reservadas en memoria (típicamente usadas por procesos <i>real-time</i>)
s	El proceso es <i>líder de sesión</i> , es decir, es padre de otros procesos (un ejemplo de esto es una <i>shell</i>)
l	El proceso tiene varios hilos (<i>multi-thread</i>)
+	El proceso se ejecuta en primer plano (<i>foreground</i>)

Algunos ejemplos más:

```
# Procesos del usuario
$ ps -u egg
$ id
$ ps -u 1000

# Enumerar los procesos propiedad de PPID 2
$ ps -f --ppid 2
```

Comando pstree

Este comando muestra la estructura jerárquica de procesos en forma de árbol que están en ejecución. Con este se puede ver las dependencias entre procesos padre y procesos hijo.

```
$pstree | more
```

```
systemd+-ModemManager---2*[{ModemManager}]
| -NetworkManager---2*[{NetworkManager}]
| -VBoxClient---VBoxClient
| -VBoxService---8*[{VBoxService}]
| -accounts-daemon---2*[{accounts-daemon}]
| -acpid
| -avahi-daemon---avahi-daemon
| -colord---2*[{colord}]
| -cron
| -cups-browsed---2*[{cups-browsed}]
| -cupsd---6*[{dbus}]
| -dbus-daemon
| -fprintd---4*[{fprintd}]
| -gdm3+-gdm-session-wor+-gdm-wayland-ses+-gnome-session-b---2*[{gnome-session-b}]
| | | | | -2*[{gdm-wayland-ses}]
| | | | | -2*[{gdm-session-wor}]
| | | | | -2*[{gdm3}]
| -gnome-keyring-d---3*[{gnome-keyring-d}]
| -2*[{kerneloops}]
| -networkd-dispat
| -packagekitd---2*[{packagekitd}]
| -polkitd---2*[{polkitd}]
| -power-profiles----2*[{power-profiles-}]
--Más--
```

El elemento superior / raíz del árbol es el proceso padre de todos los procesos del sistema. En este caso, es `systemd`, que es el primer proceso que se inicia en el arranque.

```
# Muestra el árbol de los procesos del usuario
```

```
$pstree egg | more
```

```
# Muestra además los identificadores de cada proceso o subprocesso (PID)
```

```
$ pstree -p | more
```

Comando top

Para ver la tabla de procesos, se usa el comando `top` (**Table Of Processes**).

top proporciona un vistazo en tiempo real de todos los procesos que se ejecutan en el sistema, es por tanto una información dinámica (`ps` daba información estática).

El refresco de la lista de procesos se realiza cada 3 segundos.

```
top
```

```
# El resultado será algo similar a esto (hemos numerado las líneas):
```

```
# 1. top - 03:10:18 up 7:52, 3 users, load average: 0,01, 0,27, 0,42
```

```
# 2. Tasks: 141 total, 3 running, 89 sleeping, 0 stopped, 0 zombie
```

```
# 3. %Cpu(s): 0,3 us, 0,0 sy, 0,0 ni, 99,7 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
```

```
# 4. KiB Mem : 2038132 total, 339236 free, 1340556 used, 358340 buff/cache
```

```
# 5. KiB Swap: 839676 total, 555004 free, 284672 used. 518248 avail Mem
```

```
#
# 6.  PID USUARIO      PR  NI    VIRT    RES    SHR S  %CPU  %MEM    HORA+ ORDEN
# 7.  1350 elenagg      20   0 4213320 390096 120472 S   3,6   19,2   1:12.01 gnome-
shell
# 8.   992 elenagg      20   0  538292  65964  39836 S   3,3    3,2   0:24.75 Xorg
# 9.  2990 elenagg      20   0  974612  54812  40512 S   2,3    2,7   0:14.59 gnome-
terminal-
# 10. 4903 elenagg      20   0   20692   4144   3360 R   0,7    0,2   0:00.78 top
# 11. 1188 elenagg      20   0  166104   2684   2296 S   0,3    0,1   0:17.54
VBoxClient
# 12.    1 root         20   0  167668  11584   8376 S   0,0    0,6   0:02.49 systemd

man top
```

Veamos qué **información ofrece top**:

1. La primera línea nos informa de:

- **Hora del sistema.**
- **Uptime**, cuánto lleva encendida la máquina.
- **Número de usuarios** que han accedido al sistema.
- **Load Average**: Media de la carga de la CPU:
 - Carga media durante el último minuto.
 - Carga media durante los últimos 5 minutos.
 - Carga media durante los últimos 15 minutos.

2. La segunda línea muestra el número de procesos en ejecución y su estado.

- La mayoría serán procesos del sistema, que normalmente estarán en estado durmiente o de hibernación (**sleeping**). Esto quiere decir que son procesos que están esperando algún evento ante el cual reaccionar.
- Pueden estar también en estado **Running**, procesos ejecutándose actualmente o preparados para ejecutarse.
- Procesos en estado detenido o **Stopped**
- Proceso **Zombie** cuando el proceso no está corriendo, pero aún sigue en la tabla de procesos. Un proceso queda en este estado por algún error de programación y el padre no lo ha podido finalizar.

3. La tercera línea ofrece información sobre el tiempo de CPU usado por los procesos (en %) desde la última actualización. El tiempo de actualización es de unos 3 segundos :

- % de tiempo de CPU usada por procesos del usuario (**us**).
- % de tiempo de CPU usada por procesos del sistema (**sy**).
- % de tiempo de CPU usada por procesos que tienen un **nice** definido positivo (ni). El número **nice** define la prioridad relativa de un proceso con respecto a los demás que están en la cola de procesos esperando asignación de la cpu. Su valor va entre -20 para indicar la máxima prioridad y 19 para la menor.

- % de tiempo de CPU inactiva o libre (**id**).
 - % de tiempo de CPU esperando por alguna operación de E/S (**wa**).
 - % de tiempo de CPU usada por interrupciones hardware (**hi**).
 - % de tiempo de CPU usada por interrupciones software (**si**).
 - % de tiempo de CPU *Steal time* (CPUs virtuales) (**st**). Sólo aplica cuando la distro corre en una máquina virtual. Es el % de tiempo que la CPU virtual ha estado esperando la CPU real mientras el hypervisor eligió asignarlo a otro procesador virtual.
4. La cuarta línea ofrece un resumen sobre la RAM, memoria física del sistema,(total, libre, usada y memoria del buffer). La unidad de medida es el MiB (MebiByte, 2 elevado a 20 bytes).
5. La quinta línea ofrece un resumen sobre la memoria virtual del sistema (total, utilizada, libre y caché).
6. A partir de la 6ª línea se muestran los procesos en ejecución ordenados por los que consumen más recursos. Esta información se actualiza constantemente (3seg por defecto.), y ofrece una idea de qué procesos cargan el sistema. Normalmente se vigilan la memoria RAM y la CPU que están consumiendo los procesos. También aparece el usuario que ha ejecutado el proceso, y un *PID*, o identificador de proceso. Veamos los datos que aparecen:
1. **PID** del proceso (PID).
 2. **Usuario** que ejecuta el proceso (USER).
 3. **Prioridad del proceso** (PR). (RT-Real Time indicaría que se está ejecutando en tiempo real)
 4. **Valor de nice** (NI). asigna la prioridad. 20 valor más alto -20 valor mas bajo
 5. **Memoria virtual** usada por el proceso (*swap*) (VIRT).
 6. **Memoria física** memoria RAM usada por el proceso. Resident Task Size (RES).
 7. **Memoria compartida** Cantidad de memoria Virtual que ocupa un proceso y podría ser compartida con otros procesos (SHR).
 8. **Estado** actual de los procesos en ejecución (S).
 9. **% CPU** usada por el proceso (%CPU).
 10. **% RAM** usada por el proceso (%MEM).
 11. **Tiempo** que lleva el proceso **en ejecución** (TIME+).
 12. **Nombre** del proceso (COMMAND).

Se pueden ordenar los resultados de *top* pulsando **Shift + F**.

Otras teclas para manejar el uso del comando TOP son: (esto es mientras top se está ejecutando):

- M → Ordenar por uso de memoria
- P → Ordenar por uso de CPU
- T → Ordenar por tiempo de CPU utilizado
- N → Por número de proceso
- F → seleccionar campo por el cual ordenar la lista de procesos
- i → mostrar u ocultar procesos idle y zombie
- c → Ver la línea de ejecución completa
- 1 → Ver el número de cores o CPUs
- m, l, t → muestra u oculta las líneas del resumen (parte de arriba)
- f, o → agregar/quitar columnas, cambiar el orden
- s → cambia el tiempo de actualización de la pantalla (por defecto es de 3 segundos)
- z → Cambia de color

- k -> Matar un proceso (luego pregunta el PID)
- r -> "renice" cambia el nice value de un proceso (luego pregunta el PID)
- q -> Salir de top
- h -> ayuda

Si queremos resaltar la columna que estamos ordenando realizaremos lo siguiente.

- Inicialmente presionaremos la tecla x mientras se esté ejecutando top. De este modo, el texto de la columna ordenada se mostrará en negrita.
- Acto seguido presionaremos la tecla b para que se resalte la columna que estamos ordenando.

Además top cuenta con una serie de switches además de las opciones anteriores:

- top -u usuario -> Muestra los procesos que estan corriendo con ese usuario y sus valores
- top -p PID -> muestra el proceso seleccionado y sus valores
- top -n numero -> Numero es la cantidad de iteraciones que se va a ejecutar el comando y luego se cerrara
- top -d numero -> "Numero" es el tiempo en segundos que va a esperar el comando para refrescar la lista.
- top -b -> Batch mode, ideal para mandar resultados desde top a otros programas

Comandos kill y killall

El comando **kill** sirve para enviar señales (*signals*) a un proceso (*PID*).

Cuando se envía el comando **Kill**, se envia **por defecto la señal de terminación SIGTERM**, señal que se envía el proceso para comunicarle un apagado "amable" (cerrando conexiones, ficheros y limpiando sus propios búferes). También puede ser controlada o ignorada por un manejador de señales del proceso.

Hay ocasiones que un proceso puede no responder a esta señal. **Si se desea terminar forzosamente dicho proceso** se puede enviar la señal **SIGKILL** (kill -9). Esta señal provoca un apagado forzoso del proceso y no puede ser ignorada ni manejada por un controlador de señales. Es la manera más segura de matar un programa si no se puede de otra forma.

La siguiente tabla muestra algunas de las señales más comunes:

Nombre	Número	Descripción
SIGHUP	1	Notifica a un proceso cuyo padre ha terminado su ejecución (muchas veces, el padre es un terminal)
SIGINT	2	Provoca la interrupción del programa. Notifica a un proceso cuando el usuario envía una señal de interrupción (<i>Ctrl + C</i>)
SIGQUIT	3	Notifica a un proceso cuando el usuario envía una señal de terminación, es simioar a SIGINT pero genera un fichero coredump justo antes de la termianción del programa. (<i>Ctrl + D</i>)
SIGFPE	8	Notifica a un proceso cuando se intenta realizar una operación matemática ilegal

Nombre	Número	Descripción
SIGKILL	9	Si un proceso recibe esta señal, debe terminar su ejecución de forma forzosa e inmediatamente, sin hacer ninguna limpieza
SIGALRM	14	Señal de alarma de reloj (usada con temporizadores)
SIGTERM	15	Señal de terminación de software. Se envía el proceso para comunicarle un apagado "amable" (cerrando conexiones, ficheros y limpiando sus propios búfer). Es la señal que kill envía por defecto
SIGSTOP	20	Notifica a un proceso para que detenga su ejecución y se mueva al segundo plano (<i>Ctrl + Z</i>)

```
# Envía la señal 15 (SIGTERM) a un proceso (ej:firefox) (ordena a firefox que se cierre ordenadamente)
```

```
$kill -s 15 1568 # Equivalente a kill 1568
```

```
$ kill 1568
```

```
# Envía la señal 9 (SIGKILL) a un proceso (fuerza el cierre del proceso)
```

```
$kill -s 9 1568
```

```
$kill -9 1568
```

```
# Se pueden ver todas las señales disponibles:
```

```
$kill -l
```

```
man kill
```

También se pueden enviar señales por el nombre del programa (sin conocer su *PID*). Esto se consigue con el comando `killall`.

```
# Arrancar un navegador
```

```
killall firefox
```

```
killall -9 firefox
```

```
man killall
```

3.2 Comandos para gestionar los planos de ejecución de un proceso

Ya sabemos que normalmente **los comandos que se introducen en el terminal sirven para ejecutar programas en primer plano (*foreground*)**. Cuando un programa se ejecuta en primer plano, su salida se visualiza por el terminal (que normalmente lo muestra por pantalla). El programa nos devuelve el control cuando se termina su ejecución, y volvemos a ver el *prompt*.

No obstante, **hay comandos que se ejecutan en segundo plano (*background*)**, y que no suelen mostrar su salida por el terminal. Por ejemplo, muchos procesos del sistema se ejecuten en segundo plano, y su salida se envía hacia algún fichero de *log* (bitácora).

Operador &

Para lanzar procesos en segundo plano, se usa el operador & (*ampersand*), que se escribe al final del comando

En cualquier caso, se pueden ejecutar comandos en primer y segundo plano como se muestra a continuación:

```
# Proceso en primer plano
sleep 10

# Proceso en segundo plano
sleep 15 &
```

Matar un proceso en rpimer plano: Ctrl -C

Antes se ha visto cómo se puede usar el comando *kill* para terminar un proceso. Si ese proceso se ha lanzado mediante un comando en la terminal, se puede matar **Ctrl + C**. Esto enviará la señal *SIGTERM* al programa en cuestión.

Parar un proceso (Ctrl + Z)

Cuando se ejecuta un proceso, éste puede ser detenido (*stopped*) por el usuario pulsando la combinación de teclas **Ctrl + Z**. Al hacer esto, el proceso se detiene y se mueve a segundo plano. Para iniciar su ejecución en segundo plano es necesario ejecutar el comando **bg**

Cuando ejecutamos un programa en segundo plano (mediante el **&**) o cuando lo pasamos de primer plano a segundo plano mediante **Ctrl-Z**, la shell asigna a dicho protrama un número de "trabajo o tarea" que aparece acompañado del níemro de proceso *PID*. Para conocer los "trabajos" que teenmos en segundo plano utilizamos el comando **jobs**.

```
sleep 10

# Pulsar Ctrl + Z: el prceso se detiene y se mueve a segundo plano
[1] Detenido sleep 10

jobs
```

bg

El comando **bg** sirve para continuar con la ejecución de un proceso que estaba detenido en segundo plano. Se puede usar especificando un número de *job* (que no es lo mismo que el *PID*). La ejecución del proceso se mantiene en sefundo plano.

Cuando se ejecuta *bg* sin un número de *job*, se continua con la ejecución del *job* por defecto [1]. Conviene resaltar que cuando un proceso se mueve a segundo plano, ya no acepta entradas desde el terminal, por lo que si se pulsa *Ctrl + C* o *Ctrl + Z* no se enviará ninguna señal al proceso.

Además, si un proceso se mueve del primer al segundo plano, el proceso seguirá mostrando su salida por el terminal, lo cual es un poco lioso, ya que cuesta más escribir en el terminal sin confundirse.

```
sleep 10
```

```
# Pulsar Ctrl + Z
```

```
[1] Detenido sleep 10
```

```
bg
```

```
jobs
```

```
# Otro ejemplo:
```

```
ping -c20 -a www.debian.org
```

```
# Pulsar Ctrl + Z # Se detiene y se envia a segundo plano
```

```
[1]+ Detenido ping -c20 -a www.debian.org
```

```
jobs
```

```
# Ejecutar bg
```

```
[1]+ Ejecutando ping -c20 -a www.debian.org &
```

```
# La salida de ping sigue mostrándose por el terminal, y será más costoso escribir nuevos comandos.
```

```
# Si se pulsa Ctrl + C, no se podrá terminar el comando ping.
```

```
# La salida del comando se puede redirigir a un fichero o bien a /dev/null si no es relevante.
```

```
ping -c20 -a www.debian.org > salida_ping.txt
```

```
# Pulsar Ctrl + Z
```

```
[1]+ Detenido ping -c20 -a www.debian.org > salida_ping.txt
```

```
jobs
```

```
man bg
```

fg

El comando *fg* sirve para traer un proceso desde el segundo plano al primer plano. Al igual que ocurría con *bg*, el comando *fg* puede ser invoacdo con o sin un número de *job*.


```
ping -c 20 -a www.debian.org & # Para enviar el proceso al segundo plano

# Pero ahora se mueve el proceso desde el segundo plano al primer plano, y se
continúa con su ejecución:
fg

# A partir de este momento, se puede detener el comando ping usando Ctrl + C.

man fg
```

Breve nota sobre kill

Aunque antes se ha usado *kill* usando un *PID*, también se puede enviar una señal a un proceso usando su número de *job*. Para ello, se emplea un % antes del número de *job*.

```
sleep 10

# Pulsar Ctrl + Z

kill %1
```

4. Ejecución de procesos y uso de comandos encadenados (; && ||)

Operador ;

Sirve para **encadenar comandos y ejecutar uno detrás de otro. Si uno de los comandos falla se para la ejecución.**

```
echo "Empezamos..." ; sleep 5 ; echo "...han pasado 5 segundos"
```

En el ejemplo anterior, se ejecutarán todos los comandos (dos *echo* y un *sleep*) uno tras otro y en orden.

```
echo "Empezamos..." ; sleep 5 ; tlz ; echo "...han pasado 5 segundos"
```

Nota. *tlz* es una orden errónea, se para la ejecución

Operador &&

Permite la ejecución o no, de un comando en función del código de retorno desde otro comando.

Sintaxis: `comando1 && comando2`

- El comando `comando2` se ejecuta únicamente si el comando `comando1` devuelve el código verdadero.
- Si los dos expresiones son verdaderas entonces los dos comandos devuelven verdadero.

Es decir, sirve para ejecutar un comando y, si se ejecutó SIN errores, entonces se ejecuta el segundo comando.

```
cd
pwd
echo "Empezamos..." && sleep 5 && tlz && echo "...han pasado 5 segundos"

mkdir /etc/nosepuede && cd /etc/nosepuede && echo "Estoy en $(pwd)"
pwd

mkdir ~/sisepuede && cd ~/sisepuede && echo "Estoy en $(pwd)"
pwd

cd
rmdir ~/sisepuede
```

Operador ||

Permite la ejecución o no, de un comando en función del código de retorno desde otro comando.

Sintaxis: `comando1 || comando2`

- El comando `comando2` se ejecuta únicamente si el comando `comando1` devuelve un código falso.
- La expresión global es verdadera si al menos uno de los comandos devuelve verdadero.

Sirve para ejecutar un comando y, si se ejecutó CON errores, entonces se ejecuta el segundo comando.

```
cd
pwd
echo "Empezamos..." || sleep 5 || echo "...han pasado 5 segundos"

mkdir /etc/nosepuede || echo "No se ha podido crear /etc/nosepuede :("
pwd

mkdir ~/sisepuede || echo "No se ha podido crear ~/sisepuede..."
pwd
```

5. Gestión de Memoria

Cuando un programa inicia su ejecución es necesario cargarlo desde el soporte de almacenamiento en el que está almacenado, normalmente un disco, a la memoria física del ordenador.

Sin embargo, la memoria física (principal o RAM) es limitada y en muchas ocasiones es insuficiente para almacenar las diferentes aplicaciones que ejecutan los usuarios, por tanto, ha sido necesario evolucionar en como cargar dichos programas y como hacer la gestión de memoria adecuadamente. Actualmente se utiliza lo que denominamos memoria virtual, que en realidad permite cargar en la memoria física sólo la parte del programa que se está ejecutando mientras que el resto permanece en el disco (que funciona como memoria virtual) y que se cargará en memoria física cuando sea necesario.

La gestión de memoria es especialmente relevante cuando trabajamos en sistemas operativos multiproceso, y aún más en sistemas operativos multihilo, ya que se comparten zonas de memoria en las que se alojan las variables compartidas y a las que acceden varios procesos o hilos de un proceso.

Cuando cargan demasiadas aplicaciones a la vez, el sistema se ralentiza, ya que tiene que pasar información continuamente desde el disco/almacenamiento a la memoria física o viceversa.

Los sistemas operativos suelen traer una configuración por defecto de espacio en el disco para memoria virtual, si bien, esto puede ser modificado por los administradores del sistema. Si la configuración no es correcta, entonces puede generar un excesivo uso del disco, que es mucho más lento que la memoria física, y por tanto el rendimiento o performance del sistema se degradaría.

El valor por defecto en cualquier instalación de Linux es 60 y es posible modificarlo. Este valor está indicando al kernel que utilice la memoria virtual cuando la memoria RAM tenga el 40% o menos de su capacidad libre.

El intercambio entre la zona de memoria física y virtual se denomina swap. Dicha zona es una zona del disco o del almacenamiento utilizado para almacenar procesos que actualmente no están en ejecución y así dejar memoria RAM libre para los procesos que sí lo están. Esta gestión de procesos es llevada a cabo por el gestor de memoria del sistema operativo.

Así, el sistema operativo dispone de una cola de procesos que solicitan entrar en memoria y son el gestor de memoria y el planificador del sistema operativo los responsables de ir cargando los procesos para ejecución y lo hacen en función de sus prioridades y de las necesidades de memoria de cada uno. Esta información está almacenada en su BCP (Bloque de Control de Proceso).

| _____ min (0 - 59)

Para generar el archivo propio, cada usuario deberá hacer uso del comando `crontab` para crear su fichero de tareas `crontab -e`

Para listar las tareas existente en el corntab del usuario: `crontab -l`

Para remplazar el archivo existente por otro que defina el usuario se debe utilizar el siguiente comando: `crontab archivo`

Borrar el crontab que está configurado `crontab -d`

Definir el directorio en el que se almacenará el archivo de crontab. Para realizar esta operación se deben tener permisos de ejecución en dicho directorio: `crontab -c dir`

Instrucción para manejar el crontab de otros usuarios existentes en el sistema: `crontab -u usuario`

Ejmplos:

Ejecutar todos los días a las 7 de la tarde:

```
00 19 * * * usuario /ubicacion/del/script/consulta.sh
```

Ejecutar todos los domingos a las 7 de la tarde:

```
00 19 * * 0 usuario /ubicacion/del/script/consulta.sh
```

Ejecutar el script todos los 4 de febrero a las 7 de la tarde:

```
00 19 4 2 * usuario /ubicacion/del/script/consulta.sh
```

Procesos de Arranque

init

En sistemas operativos basados en Linux / Unix, **init** (abreviatura de initialization) **es el primer proceso que inicia el kernel durante el arranque del sistema.**

Mantiene un ID de proceso (PID) de 1. Se ejecutará en segundo plano continuamente hasta que el sistema se apague, es el proceso principal.

init es el nombre genérico del proceso de arranque. A continuación, describimod los tres sistemas init más ampliamente utilizados en Linux.

- **System V (Sys V):** System V (Sys V) es uno de los primeros y tradicionales sistemas init para Unix como sistema operativo.
- **Upstart:** Upstart es un reemplazo del init de SysV, basado en eventos. Maneja el inicio de tareas y servicios durante el arranque, los detiene durante el apagado y los supervisa mientras el sistema se está ejecutando. Fue desarrollado originalmente para la distribución de Ubuntu, pero está diseñado para ser

adecuado para la implementación en todas las distribuciones de Linux como un reemplazo para el init System-V.

- **systemd**: Systemd es un nuevo sistema de inicio y administrador de sistema que se implementó / adaptó en todas las principales distribuciones de Linux sobre los sistemas de inicio tradicionales SysV. **systemctl** es la utilidad de línea de comandos y la herramienta principal para administrar los demonios / servicios de systemd como start, restart, stop, enable, disable, reload y status.

Systemd

Systemd es actualmente el sistema más extendido como herramienta init y de administración de servicios del sistema, que usan como estándar las distribuciones linux.

Principales características de Systemd

- Systemd ofrece una paralelización muy potente.
- Hace uso de la activación de socket y D-Bus para iniciar los servicios.
- Soporta el inicio de los demonios bajo demanda.
- Controla los procesos que utilizan el cgroups de Linux. Los grupos de control (cgroups) permiten definir jerarquías en las que se agrupan los procesos de manera que un administrador puede definir con gran detalle la manera en la que se asignan los recursos (no solo tiempo de atención de CPU, sino también I/O y memoria principal)
- Permite crear instantáneas y posterior restauración del estado del sistema.
- Maneja los puntos de montaje y desmontaje automáticamente.
- Implanta una lógica de control del servicio basada en dependencias transaccionales.

Systemd es el primer proceso iniciado por el kernel y siempre tendrá el pid 1. (init será un enlace simbólico a systemd)

```
egg@UTADLR$ ps -ef | grep init

# Devolverá como resultado algo parecido a:
# root      1      0      0      18:29 ?        00:00:03 sbin/init splash
# ...

# Lo que demuestra que init es el proceso de PID número 1 (primero en arrancar)

# Si ejecutamos ahora
egg@UTAD-R$ ls -l /sbin/init

# Podemos ver que init es un enlace simbólico a systemd
#lrwxrwxrwx 1 root 20 ago 21 23:11 /sbin/init -> /lib/systemd/systemd
```

systemctl

systemctl es la utilidad de línea de comandos y la herramienta principal para administrar los servicios (demonios) de systemd. Nos permitirá ejecutar las siguientes operaciones sobre dichos servicios: **start, restart, stop, enable, disable, reload y status.**

Descripción de comandos systemd para gestión de servicios del sistema:

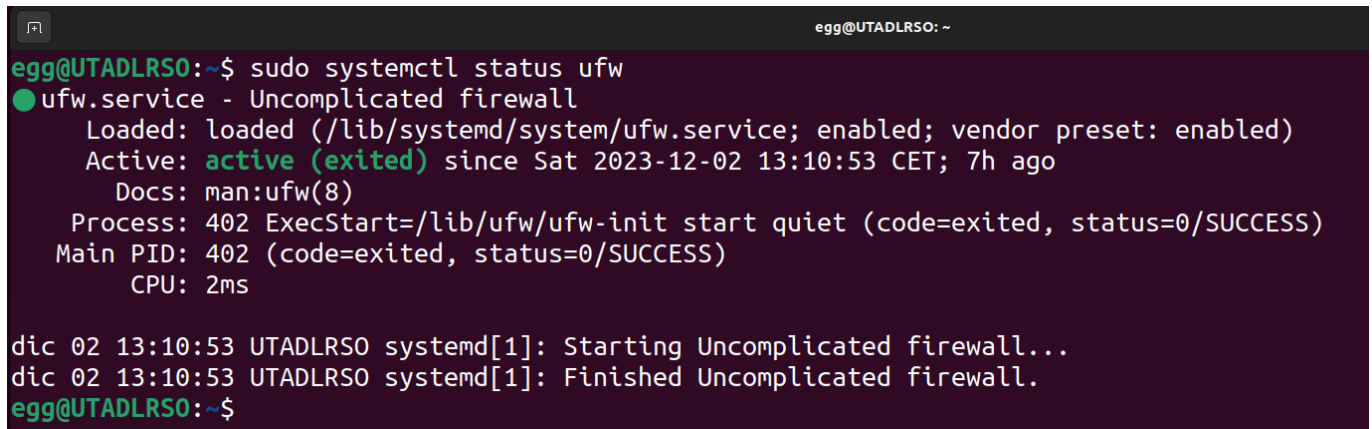
- Iniciar un servicio: `systemctl start <nombre_servicio>`
- Detener un servicio `systemctl stop <nombre_servicio>`
- Reiniciar un servicio `systemctl restart <nombre_servicio>`
- Recarga la configuración de un servicio `systemctl reload <nombre_servicio>`
- Reinicia el servicio sólo si el servicio se está ejecutando: `systemctl condrestart <nombre_servicio>`
- Verificar si el servicio se está ejecutando: `systemctl status <nombre_servicio>`

Veamos algunos ejemplos de uso:

```
# El servicio ufw permite habilitar y gestionar las reglas del cortafuegos del kernel
# Comprobamos su estado:

egg@UTAD-RS0$ sudo systemctl status ufw
```

El resultado sería:



```
egg@UTADLRSO: ~
egg@UTADLRSO:~$ sudo systemctl status ufw
● ufw.service - Uncomplicated firewall
   Loaded: loaded (/lib/systemd/system/ufw.service; enabled; vendor preset: enabled)
   Active: active (exited) since Sat 2023-12-02 13:10:53 CET; 7h ago
     Docs: man:ufw(8)
   Process: 402 ExecStart=/lib/ufw/ufw-init start quiet (code=exited, status=0/SUCCESS)
  Main PID: 402 (code=exited, status=0/SUCCESS)
    CPU: 2ms

dic 02 13:10:53 UTADLRSO systemd[1]: Starting Uncomplicated firewall...
dic 02 13:10:53 UTADLRSO systemd[1]: Finished Uncomplicated firewall.
egg@UTADLRSO:~$
```

```
# Paramos el servicio

egg@UTAD-RS0$ sudo systemctl stop ufw
```

El resultado sería:


```
egg@UTADLRSO:~$ sudo systemctl stop ufw
egg@UTADLRSO:~$ sudo systemctl status ufw
○ ufw.service - Uncomplicated firewall
   Loaded: loaded (/lib/systemd/system/ufw.service; enabled; vendor preset: enabled)
   Active: inactive (dead) since Sat 2023-12-02 20:53:09 CET; 3s ago
     Docs: man:ufw(8)
   Process: 402 ExecStart=/lib/ufw/ufw-init start quiet (code=exited, status=0/SUCCESS)
   Process: 3415 ExecStop=/lib/ufw/ufw-init stop (code=exited, status=0/SUCCESS)
  Main PID: 402 (code=exited, status=0/SUCCESS)
    CPU: 2ms

dic 02 13:10:53 UTADLRSO systemd[1]: Starting Uncomplicated firewall...
dic 02 13:10:53 UTADLRSO systemd[1]: Finished Uncomplicated firewall.
dic 02 20:53:09 UTADLRSO systemd[1]: Stopping Uncomplicated firewall...
dic 02 20:53:09 UTADLRSO ufw-init[3415]: Skip stopping firewall: ufw (not enabled)
dic 02 20:53:09 UTADLRSO systemd[1]: ufw.service: Deactivated successfully.
dic 02 20:53:09 UTADLRSO systemd[1]: Stopped Uncomplicated firewall.
egg@UTADLRSO:~$
```

```
# iniciamos el servicio
```

```
egg@UTAD-RSO$ sudo systemctl start ufw
```

El resultado sería:

```
egg@UTADLRSO:~$ sudo systemctl start ufw
egg@UTADLRSO:~$ sudo systemctl status ufw
● ufw.service - Uncomplicated firewall
   Loaded: loaded (/lib/systemd/system/ufw.service; enabled; vendor preset: enabled)
   Active: active (exited) since Sat 2023-12-02 20:54:57 CET; 2s ago
     Docs: man:ufw(8)
   Process: 3432 ExecStart=/lib/ufw/ufw-init start quiet (code=exited, status=0/SUCCESS)
  Main PID: 3432 (code=exited, status=0/SUCCESS)
    CPU: 1ms

dic 02 20:54:57 UTADLRSO systemd[1]: Starting Uncomplicated firewall...
dic 02 20:54:57 UTADLRSO systemd[1]: Finished Uncomplicated firewall.
egg@UTADLRSO:~$
```

8. Ficheros de Arranque del sistema

El shell, cuando se inicia, lo primero que hace es ejecutar una serie de comandos que se encuentran agrupados en unos ficheros denominados ficheros de arranque o ficheros de inicio. Con estos ficheros lo que se pretende es configurar el entorno del shell en el modo que nos interese, para lo cual colocaremos en su interior unos comandos u otros.

Sirven para personalizar el entorno de las sesiones, bien a nivel global (usuario root), bien a nivel de usuario.

Según sea el tipo del shell, se utilizan unos ficheros de arranque u otros. Un shell puede ser:

- **Interactivo:** Son los shell que nos permiten interactuar con ellos, ejecutando comandos introducidos desde el teclado. Un shell no interactivo es aquel que ejecuta comandos desde un fichero o script.
- **De inicio o de login :** Son los shells que se ejecutan después del arranque del sistema e inmediatamente después de que el usuario se haya identificado. Por ejemplo cuando abrimos una terminal de texto (CTRL+ALT+F1, CTR+ALT+F2, etc.), introducimos usuario y clave, y a continuación

aparece un shell interactuando con nosotros, este shell, concretamente, es de inicio, y también es interactivo. Los shell que abrimos en el entorno gráfico sobre una ventana no es de inicio, aunque sí interactivo.

Cuando el shell bash es de inicio e interactivo, se ejecutan por orden los siguientes ficheros de arranque:

Dominio de root:

- **/etc/profile** y **/etc/profile.d/***: Estos ficheros los modifica root y afectan a todos los usuarios del sistema cuando abren una terminal de texto.
- **/etc/bash.bashrc**: Este fichero es llamado por el anterior (/etc/profile). También pertenece al dominio del root y sirve para centralizar la personalización de aspectos que afecten a todos los usuarios.

Dominio de usuario:

- **~/.profile**: Este fichero es del dominio del usuario, y con él se pretende que cada usuario configure el inicio de su shell. Puede sustituirse por **~/.bash_profile** o por **~/.bash_login**. El fichero **~/.profile** está creado por defecto. Solo se ejecutará uno de estos tres ficheros, aunque existan más de uno, y será el primero que se encuentre siguiendo este orden: **~/.bash_profile**, **~/.bash_login**, **~/.profile**.
- **~/.bashrc**: Este fichero es llamado por el anterior y tiene el mismo objetivo.

En todos los tipos de shell, cuando alguno de los ficheros de arranque no existe, no se detiene la ejecución del shell, sino que se sigue adelante con el siguiente paso.

Cuando el shell bash es interactivo pero no de inicio (los shells que abrimos desde el escritorio), **los ficheros de arranque que se ejecutan son por orden:**

Dominio de root:

- **/etc/bash.bashrc**

Dominio de usuario:

- **~/.bashrc**

Para ampliar

Este tema es sólo una introducción a la gestión de procesos en Linux. Para ampliar, se puede buscar información sobre:

- **jobs**
- **trap**
- **stty**
- **nice**
- **nohup**
- **pgrep**