Capítulo 8

Proceso de arranque en GNU/Linux

1. Al principio toma el control la **BIOS**.
2. En una segunda etapa, tomará el control **el cargador de arranque**.
3. En una tercera, etapa el control pasa al propio **Kernel**.
4. Ultima etapa tendremos en memoria los programas de usuario conviviendo junto con el propio sistema operativo, quienes tomarán el control del sistema.

Primera etapa: La BIOS

Al encender la PC, toma control la **BIOS**, que realiza una serie de operaciones básicas de hardware. Una vez que el hardware es reconocido y queda listo para usar, la BIOS carga en memoria el código del cargador de arranque y le pasa el control.

Segunda parte: El cargador de arranque

Existen diferentes cargadores de arranque, **GRUB**, **Lilo**, etc. Normalmente el cargador de arranque se guarda en el **MBR** (Master Boot Record) que tiene un tamaño reducido de 512 bytes del disco, obliga a dividir el arranque en varias etapas. De este modo, la **BIOS** carga la primer etapa del cargador de arranque. Y, después cargará el resto del cargador de arranque.

GRUB se carga y se ejecuta en 4 etapas:

- La **BIOS** carga la primera etapa del cargador, que se encuentra almacenada en el **MBR**.
- La primera etapa carga el resto del cargador. Si la segunda etapa está en un dispositivo grande, se carga una etapa intermedia (llamada etapa 1.5), que contiene código extra que permite leer cilindros mayores que 1024 o dispositivos tipo LBA.
- La segunda etapa ejecuta el cargador y muestra el menú de inicio de GRUB, permitiendo seleccionar el SO que se desea arrancar.
- Una vez seleccionado el sistema operativo que se quiere arrancar, se carga en memoria y se le pasa el control.

Tercera parte: El kernel

El proceso del kernel se realiza en dos etapas:

- **La etapa de carga.**
- **La etapa de ejecución.**

El **kernel** generalmente se almacena en un archivo comprimido. Este archivo comprimido se carga y se descomprime en memoria.

Por otra parte, también se cargan los drivers necesarios mediante el **initrd**. El **initrd** crea un sistema de archivos temporal usado en la fase de ejecución del kernel.

Una vez que el **kernel** se ha cargado en memoria y está listo, se lleva a cabo su ejecución.

Lo último que se lanza es el proceso **init**.

¿ Qué es un proceso ?

Un proceso es un programa que se ejecuta en un determinado momento en el sistema. Siempre hay una serie de procesos que se ejecuta sin que el usuario sepa de ellos y éstos hacen que el Sistema sea utilizable. Contiene un conjunto de estructuras de datos del Kernel y en una dirección de memoria. En la dirección de memoria se ha reservado espacio para el código del proceso, su área de datos, pila de proceso y otra información adicional usada por el sistema durante su ejecución.

Hay 2 tipos de procesos :

- **Los Procesos de usuarios** : son aquello procesos que el usuario utiliza.

- **Los Demonios** : son aquellos procesos que para su funcionamiento, no requiere de la intervención del usuario y en general se usan para programas que funcionan constantemente como servidores de red, programas administrativos, etc.

Cuarta parte: El proceso init

Todos los sistemas **Unix**, ejecutan el proceso **init**. El archivo de configuración de **init** es **/etc/inittab**, en el que se indica el primer script que se debe ejecutar que es el **/etc/init.d/rcS**

Este proceso **init**, es el responsable de la inicialización de nuevos procesos, todos los procesos del sistema, excepto el proceso **swapper**.

El proceso **init** es un proceso dispatcher (despachador) y produce, entre otros, los procesos para que los usuarios puedan conectarse al sistema.

En operaciones de multiusuario, el papel de **init** es crear un proceso por cada puerto del terminal en el cuál un usuario pueda conectarse.

En GNU/Linux, los Procesos se manejan en forma jerárquica: cada proceso es lanzado desde un proceso “padre”: dicho proceso se le llama proceso “hijo”. Entonces podremos deducir que todos los procesos son hijos del proceso padre por excelencia: el proceso **init**.

```
# more /etc/inittab

        # /etc/inittab: init(8) configuration.
        # $Id: inittab,v 1.91 2002/01/25 13:35:21 miquels Exp $

        # The default runlevel.
        id:2:initdefault:

        # Boot-time system configuration/initialization script.
        # This is run first except when booting in emergency (-b) mode.
        si::sysinit:/etc/init.d/rcS

        # What to do in single-user mode.
        ~~:S:wait:/sbin/sulogin

        # /etc/init.d executes the S and K scripts upon change
        # of runlevel.
```

```
#
# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
# Normally not reached, but fallthrough in case of emergency.
z6:6:respawn:/sbin/sulogin

# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

# Action on special keypress (ALT-UpArrow).
#kb::kbrequest:/bin/echo "Keyboard Request--edit /etc/inittab to let this work."

# What to do when the power fails/returns.
pf::powerwait:/etc/init.d/powerfail start
pn::powerfailnow:/etc/init.d/powerfail now
po::powerokwait:/etc/init.d/powerfail stop

# /sbin/getty invocations for the runlevels.
#
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
#
# Format:
#  <id>:<runlevels>:<action>:<process>
#
# Note that on most Debian systems tty7 is used by the X Window System,
# so if you want to add more getty's go ahead but skip tty7 if you run X.
#
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6

# Example how to put a getty on a serial line (for a terminal)
#
#T0:23:respawn:/sbin/getty -L ttyS0 9600 vt100
#T1:23:respawn:/sbin/getty -L ttyS1 9600 vt100

# Example how to put a getty on a modem line.
#
```

```
#T3:23:respawn:/sbin/mgetty -x0 -s 57600 ttyS3
```

El archivo /**etc/init.d/rcS** ejecuta todos los scripts situados en **/etc/rcS.d/S***, los que empiezan con **S** y un número que es el orden, si tiene el mismo número de orden lo realiza alfabéticamente. Donde realiza la comprobación y montaje de los sistemas de archivos, la carga de módulos, la inicialización de los servicios de red, configuración del reloj, etc.

```
# ls /etc/rcS.d

README              S06hostname.sh      S08mtab.sh          S13networking
S15nfs-common       S20fuse             S01mountkernfs.sh   S06hwclockfirst.sh
S08pvenetcommit     S13pppd-dns         S16mountnfs.sh      S20lm-sensors
S02udev             S06lvm2             S09checkfs.sh       S13procps
S17mountnfs-bootclean.sh  S20open-iscsi  S03mountdevsubfs.sh  S07checkroot.sh
S10ifupdown         S13udev-mtab        S18kbd              S20quota
S04bootlogd         S08hwclock.sh       S10mountall.sh      S13urandom
S19console-setup    S21stop-bootlogd-single  S05keyboard-setup
S08ifupdown-clean   S11mountall-bootclean.sh  S13x11-common  S20alsa-utils
S06hdparm           S08module-init-tools  S12mountoverflowtmp
S14portmap     S20bootmisc.sh
```

Luego, y por compatibilidad, también ejecuta todos los archivos (excepto aquellos con un `.' en su nombre) situados en **/etc/rc.boot/**. Este último directorio está reservado para el administrador del sistema y **su utilización ha caído en desuso**.

Niveles de arranque

Hay dos modos de arranque :

- **System-v** = Utiliza **init** por defecto, debido a la simplicidad, mayor número de recursos y mayor flexibilidad que el tradicional **init BSD**, los archivos de configuración residen en el directorio **/etc/rc.d**. La mayoría de las distribuciones de GNU/Linux.

- **BSD** = Tiene un **init** diferente la principal diferencia se encuentra en los niveles, el sistema BSD contiene un archivo con los demonios que se ejecutaran. Sistemas operativos : *BSD, archlinux, slackware.

Una vez completo el proceso de arranque, el proceso **init** iniciará todos los servicios que han sido configurados para ejecutarse en el nivel de ejecución predeterminado.

```
# The default runlevel.
id:2:initdefault:
```

Debian utiliza los siguiente niveles de arranque :

- 0 (apagar el sistema) .
- 1 (modo monousuario).
- 2 al 5 (modos multiusuario).
- 6 (reiniciar el sistema).

Lo que ejecuta se encuentra en **/etc/rcN.d** (**N** es el nivel de ejecución). Como se indico anteriormente ejecuta los que empieza con **S** y el orden de arranque. Vemos que el contenido de este directorio apunta a los script de **init.d**. Dentro de **/etc/init.d/** se encuentran los script de los

demonio de nuestros servicios.

```
# ls -l /etc/rc2.d/*mysql

        lrwxrwxrwx 1 root root 15 mar 21 10:39 S19mysql -> ../init.d/mysql
```

Como vemos en el ejemplo anterior la primera letra empieza con una **S** :

- Los que empiezan con **K** (estos scripts se ejecutan con el argumento **stop**).
- Los que empiezan con S (estos scripts se ejecutan con el argumento **start**).

Después de la primer letra continua con 2 dígitos que indica el orden de arranque, siempre son dígitos de **01** a **99**, y luego el nombre, si tiene una **S**, luego el mismo nivel de arranque, se ejecuta en orden alfabético.

Para ejecutar un demonio hacemos lo siguiente :

```
# cd /etc/init.d
# ./ssh stop
```

Tenemos los siguiente parámetros para pasarle :

- **start** = ejecuta el demonio.
- **stop** = mata al proceso.
- **status** = nos muestra el estado si se esta ejecutando o no.
- **reload** = relee el archivo de configuración.
- **restart** = realiza un **stop** y luego un **start**.

Esto lo realiza en el momento, pero cuando reiniciamos nuestra PC tomara la configuración de nuestro nivel de arranque.

Personalizar los niveles de ejecución

Para saber en que nivel de arranque estamos ejecutamos el comando **runlevel**.

```
# runlevel

 N 2
```

Dónde **N** nos indica el nivel anterior que estábamos ejecutando y **2** es el nivel corriente que estamos ejecutando.

Para pasarnos en el momento de un nivel a otro sin reiniciar podemos ejecutar tanto el comando **init** como **telinit**.

Opciones :

q o Q	*Relee el archivo **/etc/inittab**.*
s o S	*Cambia a single user.*
0 a 5	*Nivel de ejecución.*

Cuando me paso de un nivel a otro lo que hace es ir a dicho nivel bajar y subir los demonios.

Si queremos que al reiniciar no se ejecute un demonio de terminado tenemos 2 opcions :

```
# cd /etc/rc2.d

# mv S19mysql K19mysql
```

O bien utilizando la herramienta update-rc :

```
# update-rc.d mysql default
```

Si queremos remover un demonio en el proceso de arranque :

```
# update-rc.d -f mysql
```

Si queremos que solo se ejecute en el runlevel 2 y con orden 99 :

```
# update-rc.d mysql default 99 2
```

Manejo de procesos

Las estructuras de datos referidas a los procesos contienen información que permite el manejo de éstos. Algunos de los datos que contienen son : mapa de espacio del proceso, estado actual, prioridad de ejecución, máscara actual de la señal del proceso y propietario.

Salvo el proceso **init** que tiene el **PID 1**, todos los demás procesos son creados por otros procesos.

Atributos de un proceso

Algunos de los parámetros asociados a los procesos afectan de forma directa a su ejecución, por ejemplo, el tiempo de proceso, prioridad, ficheros a los que se pueden acceder, etc.

- **PID (Process IDentificacion)** = Es un número que identifica al proceso en el sistema. Se asignan de forma secuencial a cada nuevo proceso que se crea.

- **PPID (Parent Process IDentification)** = Es un número que se corresponde con el **PID** del proceso <padre>. El proceso <padre> es aquel que creó al proceso actual, llamado proceso <hijo> del anterior.

- **UID, GID (User Identification, Group IDentification)** = Estos números ya aparecieron en la unidad anterior. Son el número de identificación del usuario que creó el proceso **UID**, y el número de identificación del grupo de usuario, **GID**. Sólo el superusuario y el usuario que creó el proceso, llamado propietario del proceso, pueden modificar el estado de operación de los procesos.

- **EUID, EGID (Effective User IDentification, Effective Group IDentification)** = Estos números identifican al usuario que ejecuta el proceso; es atráves de estos números y no del **UID** y **GID** cómo el sistema determina para qué ficheros el proceso tiene acceso.

- **Prioridad** = La prioridad de un proceso afecta al tiempo y al orden de ejecución de éste por parte del procesador. Un proceso de mayor prioridad que otro significa que en la ejecución de los dos procesos el sistema asignará un período de ejecución mayor para el de mayor prioridad y, además, lo elegirá más veces para ejecutarlo. Si no se indica nada al crear un proceso, éste toma la misma prioridad que la del proceso <padre>. El propietario y el proceso mismo pueden modificar la prioridad, pero siempre en sentido decrecimiento sólo el superusuario no tiene restricciones para cambiar la prioridad de un proceso.

- **Control de terminal** = Son enlaces que determinan de dónde toman la entrada y la salida de datos y el canal de error los procesos durante su ejecución. Si no se usan redirecciones, se utilizan por defecto la entrada y salida estándar.

Creación y ejecución de procesos

Cuando un proceso quiere crear un nuevo proceso, el primer paso consiste en realizar una copia de sí mismo mediante la llamada del sistema **fork**. La operación **fork** crea una copia idéntica del proceso padre, salvo en los siguientes casos :

- El nuevo proceso tiene un **PID** distinto y único.

- El **PPID** del nuevo proceso es el **PID** del proceso padre.

- Se asume que el nuevo proceso no ha usado recursos.

- El nuevo proceso tiene su propia copia de los ficheros descriptores del proceso padre.

Estados de los procesos

Un proceso tiene cinco estados de ejecución :

- **Ejecutandose (running R)**. El proceso se está ejecutando.
- **Durmiendo (sleeping S)**. El proceso está en espera de algún recurso del sistema.
- **Intercambiado (swapped)**. El proceso no está en memoria.
- **Zombi (Zombie Z)**. El proceso trata de finalizar su ejecución.
- **Parado (stopped)**. El proceso no puede ser ejecutado.

Cuando un proceso se ejecuta es porque tiene los recursos del sistema necesarios y dispone de tiempo de procesador. El sistema <duerme> un proceso en el momento que necesita recursos del sistema que no se pueden obtener de manera inmediata.

- Los procesos <**dormidos**> esperan a que ocurra un determinado evento. Por ejemplo entrada de datos, una conexión de la red, etc. Los procesos <**dormidos**> no consumen tiempo del procesador.

- Los procesos <**intercambiados**> no se encuentran en la memoria principal del sistema, se vuelcan a la memoria de intercambio. Esto sucede cuando la memoria principal no dispone de suficiente espacio libre y los datos del proceso pasan continuamente de la memoria de intercambio a la principal como consecuencia, la ejecución del proceso se realiza de forma poco efectiva.

- Cuando un proceso está <**zombi**> no se puede volver a ejecutar, el espacio de memoria que utilizaba se ha liberado y sólo mantiene algunas de las estructuras de datos que le dan entidad a los procesos.

- Un proceso está <**parado**> cuando no se puede ejecutar. La diferencia entre un proceso <**dormido**> y otro <**parado**> es que este último no puede continuar ejecutándose hasta que reciba una señal ***CONT*** de otro proceso. Las siguientes situaciones llevan un proceso al estado de parada:

    - Desde una shell **csh** se pulsa **Ctrl-Z**.

    - A petición de un usuario o proceso.

    - Cuando un proceso que se ejecuta en segundo plano trata de acceder a un terminal.

Señales

Las señales se utilizan para que un proceso suspenda la tarea que está realizando y se ocupe de otra. Cuando se envía una señal a un proceso, éste puede actuar de dos maneras: si el proceso dispone de una rutina específica para esa señal, llamada <**manejador**> o ***handler***, la utiliza, en caso contrario, el **Kernel** utiliza el manejador por defecto para esa señal. Utilizar un manejador específico para una señal en un proceso se denomina capturar la señal.

Hay dos señales que no pueden ser ni capturadas ni ignoradas, **KILL** y **STOP**. La señal de **KILL** hace que el proceso que la recibe sea destruido. Un proceso que recibe la señal de **STOP** suspende su ejecución hasta que reciba una señal **CONT**.

Tabla de señales :

Número	Nombre	Descripción	Por defecto	¿ Capturada ?	¿ Bloqueada ?
1	SIGHUP	Hangup.	Terminar	SI	SI
2	SIGINT	Interrumpir.	Terminar	SI	SI
3	SIGQUIT	Salir.	Terminar	SI	SI
4	SIGILL	Instrucción ilegal.	Terminar	SI	SI
5	SIGTRAP	Trazar.	Terminar	SI	SI
6	SIGIOT	IOT.	Terminar	SI	SI
7	SIGBUS	Error de de Bus.	Terminar	SI	SI
8	SIGFPE	Excepción aritmética.	Terminar	SI	SI
9	**SIGKILL**	**Destruir**	**Terminar**	**NO**	**NO**
10	SIGUSR1	Primera señal definida por el usuario.	Terminar	SI	SI
11	SIGSEGV	Violación de segmentación.	Terminar	SI	SI
12	SIGUSR2	Segunda señal definida por el usuario.	Terminar	SI	SI
13	SIGPIPE	Escribir en un pipe.	Terminar	SI	SI

14	SIGALRM	Alarma del reloj.	Terminar	SI	SI
15	**SIGTERM**	**Terminación del programa.**	**Terminar**	**SI**	**SI**
16	SIGSTKFLT			SI	SI
17	SIGCHLD	El estado del hijo ha cambiado.	Ignorar	SI	SI
18	**SIGCONT**	**Continuar después de parar.**	**Ignorar**	**SI**	**NO**
19	SIGSTOP	Parar.	Parar	NO	NO
20	SIGSTP	Parada desde el teclado.	Parar	SI	SI
21	SIGTTIN	Lectura en segundo plano.	Parar	SI	SI
22	SIGTTOU	Escritura en segundo plano.	Parar	SI	SI
23	SIGURG	Condición urgente de socket.	Ignorar	SI	SI
24	SIGXCPU	Excedido el tiempo de CPU.	Terminar	SI	SI
25	SIGXFSZ	Excedido el tamaño de fichero.	Terminar	SI	SI
26	SIGVTALRM	Alarma de tiempo virtual.	Terminar	SI	SI
27	SIGPROF	Alarma.	Terminar	SI	SI
28	SIGWINCH	Cambia de ventana.	Ignorar	SI	SI
29	SIGLOST	Recurso perdido.	Terminar	SI	SI
30	SIGPWR			SI	SI
31	SIGUNUSED			SI	SI

Esta tabla la podemos ver con el comando :

```
# kill -l
```

Enviando señales

Sólo el propietario del proceso y el superusuario (**root**) pueden mandar señal **KILL** a un proceso, por defecto la señal es **15** (**TERM**).

```
kill [-señal] PID
```

Ejemplo :

```
# kill -9 204
```

También tenemos el comando : **killall** que nos permite todos los procesos por el nombre del demonio.

Ejemplo :

```
# killall apache2
```

Monitorización de los procesos

Disponemos de una serie de comandos para obtener información de los procesos que se están ejecutando, como poder conocer su PID, estado, propietario del proceso, etc. También podemos cambiar su prioridad de ejecución.

```
# ps [-opciones] [PID ...]
```

Opciones :

l	*Formato grande.*
u	*De usuario, con nombre y hora de comienzo.*
j	*Trabajos.*
s	*Señal.*
v	*Memoria Virtual.*
m	*Información acerca de la memoria.*
f	*Arbol.*
a	*Procesos de otros usuarios.*
x	*Procesos sin terminal de control.*
S	*Cpu hijo y posible fallo de la página.*
c	*Muestra el nombre del comando según la tarea.*
e	*Muestra el entorno.*
h	*No muestra la cabecera.*
r	*Muestra procesos activos.*

Ejemplo :

```
# ps aux | more
```

También tenemos el comando : **pstree** que me muestra en forma de arbol los nombre de los procesos y de quien cuelga.

Ejemplo :

```
# pstree | more
```

Con el comando **pidof** podemos obtener el ID de un nombre de proceso.

Ejemplo :

```
# pidof apache2

 6215 2828 2827 1721
```

Otro dos comandos interesantes para ver en tiempo real es : **top** y **htop**. En ambos nos da en forma detallada y en tiempo real.

```
# top
```

```
# apt-get install htop
# htop
```

También como se menciono en otros capítulos dijimos que podemos tener información de cada proceso que se ejecuta en el filesystem **/proc**.

Ejemplo :

```
# pidof apache2

  6215 2828 2827 1721

# cat /proc/6215/cmdline

  /usr/sbin/apache2-kstart
```

Priorizando procesos

La prioridad de los procesos se especifica mediante un valor numérico entero en el rango de **-20**, **prioridad máxima**, hasta **20**, **prioridad mínima**. Si no se indica nada, los procesos son lanzados con una prioridad por defecto. Un proceso de prioridad mínima sólo se ejecuta si no hay más procesos en el sistema en espera de ser ejecutados.

Para lanzar un proceso con una prioridad específica se utiliza el comando :

```
nice [-n ajuste] [-ajuste] [--adjustment=ajuste] [proceso]
```

Ejemplo :

Disminuye su prioridad en 5. Si no se indica un valor numérico, por defecto, la prioridad se decrementa en 10 unidades. El superusuario puede aumentar la prioridad de ejecución de los procesos.

```
# nice -5 apache2
# nice --10 apache2
```

Para modificar la prioridad de un proceso que se está ejecutando, se utiliza el comando **renice**.

```
renice prioridad [[-p] PID …] [[-g] grupo …] [[-u] usuario ...] [proceso]
```

Ejemplo:

```
# renice 16 -p 2069
```

Procesos en background

Con el comando **nohup** y la opción **&** podemos ejecutar procesos en segundo plano para que nos quede la terminal libre para poder seguir utilizandola.

Ejemplo:

```
# nohup yes >/dev/null &
```

El **nohup** lo que hace es generar un archivo con la salida y aparte me permite desconectarme de la terminal y se queda ejecutando, el **&** es utilizarlo en segundo plano.

Si no ejecuto **nohup** y solo **&**.

```
# yes >/dev/null &
```

Veremos que al salir de la terminal muere el proceso.

Ver procesos en background

Aquellos procesos que lance en background los puedo ver con el comando **jobs**.

```
# jobs

  [1]-  Ejecutando              nohup yes > /dev/null &
  [2]+  Ejecutando               yes > /dev/null &
```

El que aparece con el símbolo de + es el ultimo proceso que se esta ejecutando.

Traer los procesos

Con los comandos **fg** (foreground) y **bg** (background) es posible manipular los procesos que estén suspendidos temporalmente, ya sea porque se les envió una señal de suspensión como **STOP** (**20**) o porque al estarlo ejecutando se presionó **ctrl-Z**. Entonces para reanudar su ejecución en primer plano usaríamos :

```
# fg 1

  nohup yes > /dev/null &
```

Si solo ejecuto **fg** me trae el que tiene el simbolo +.