Introducción a GNU/LINUX (apuntes)

Índice

Breve historia de GNU/LINUX. Definición de GNU, FSF y GPL.

En la década de los sesenta, principios de los setenta, los grandes fabricantes de ordenadores obtenían el beneficio del hardware regalando el Sistema Operativo.

Las universidades tenían permiso para estudiar y modificar el código fuente.

Los laboratorios Bell (AT&T) crean el sistema operativo UNIX caracterizado por:

- Una buena gestión de los recursos.
- ***Compatibilidad*** con el hardware de distintos fabricantes.

Las grandes compañias dejan, poco a poco, de "regalar" el Sistema Operativo y van poniendo cada vez más restricciones al acceso del código fuente.

Richard Stallman, que trabajaba para el MIT, necesitaba modificar los drivers de una impresora para que avisara de cuando se atascaba el papel. Para ello solicita las fuentes de dicho controlador a la empresa fabricante del dispositivo, recibiendo una negativa como respuesta.

Es entonces cuando decide iniciar un gran proyecto para intentar abrir otra vez el código fuente de los programas. Consciente de que no podría conseguir que las compañias cedieran en este punto, se propuso crear su propio sistema operativo y aplicaciones iniciando el proyecto ***GNU*** (GNU, Not UNIX).

Escribe un manifiesto en el que describe lo que es ***software libre*** (que no necesariamente gratuito), basándose en cuatro libertades:

- Libertad 0: Posibilidad de utilizar el programa para cualquier propósito.
- Libertad 1: Permiso para estudiar cómo funciona el programa y adaptarlo a las propias necesidades. Es necesario, por tanto, el acceso al código fuente.
- Libertad 2: Distribución libre.
- Libertad 3: Mejora del programa y publicación de las propias mejoras en beneficio de toda la comunidad. De nuevo, el acceso al código fuente es imprescindible.

A estas cuatro libertades forman la ***GPL*** (GENERAL PUBLIC LICENSE).

Para realizar este proyecto se crea la ***FSF*** (Free Software Foundation).

En 1984 se comienza la producción de software comenzando por el desarrollo de las herramientas necesarias para hacer el sistema operativo: editores, compiladores, etc.

Desde el primer momento se quiso realizar un sistema operativo parecido a Unix y siguiendo las normas POSIX (Portable Operating System Interface).

Se dejó para el final del proceso el kernel, denominado HURD, actualmente en fase de desarrollo.

El profesor de la Universidad de Holanda, Tanenbaum, desarrolló, con único objetivo la docencia, el sistema operativo microkernel MINIX en el año 1987. Si bien este sistema operativo es ideal para el aprendizaje de la teoría, no es posible su utilización como sistema operativo de trabajo.

Linus Torvalds, en 1991, estudiante de la universidad de Helsinki, realiza un sistema operativo, basándose en MINIX, para su proyecto fin de carrera.

"Cuelga" su proyecto en internet y da permiso a quien lo desee para modificar y mejorar su sistema operativo.

Gente de todo el mundo comenzó a interarse por este proyecto que, al utilizar el compilador (gcc) y el intérpete (bash) del proyecto GNU como piezas fundamentales, también tenía las características de software libre.

Al proyecto se le denomina GNU/LINUX.

Denominaciones de las versiones del kernel GNU/Linux

Los números de versión del kernel de GNU/Linux tienen más significado del que pudiera parecer a primera vista. Por ejemplo:

2.4.9

Este número de versión está dividido en tres partes:

-**número mayor:** (en el ejemplo: 2) rara vez se cambia. Un incremento aquí implicaría grandes cambios en el kernel.

-**número menor**: (en el ejemplo: 6) indica la estabilidad del kernel:

par: estable y de calidad.

Impar: inestable ya que se encuentra en fase de desarrollo.

-r**evisión**: (en el ejemplo: 9) nivel de actualización alcanzado. Soluciona bugs y errores varios.

Para saber la versión de kernel que está "corriendo" en tu máquina:

uname -r

A partir del mes de marzo de 2005 se decide un nuevo sistema de numeración que añade un número más a la versión: 2.6.x.y. Donde **x** es el número de versión e **y** la revisión.

La revisión son arreglos rápidos y sencillos que no necesitan ser realizados por "expertos".

Estructura del kernel de GNU/LINUX

En el siguiente código:

```
j=1;
for (i=0;i<=5;i++)
{
    j=j * (j-i);
}
printf ("%d", j);
```

¿Quién está escribiendo realmente el número en la pantalla?

La línea "printf" no pinta el número en la pantalla, sino que llama a una biblioteca que sabe cómo hacerlo.

La biblioteca es libc.

Pero libc tampoco escribe el número en la pantalla, convierte la cadena en un formato y realiza una llamada a write que no está implementada en libc ni en ninguna otra biblioteca instalada.

¿Quién "habla" con el hardware entonces? La respuesta es el kernel.

¿Cómo sabremos cuándo nos encontramos en modo usuario? Pues cuando se esté ejecutando el código que hemos tecleado nosotros o el que haya en alguna de las bibliotecas que hayamos enlazado.

¿Cuándo estaremos en modo “kernel”? Cuando se ejecuten las líneas de código desarrolladas por un programador de kernel.

La arquitectura 80x86 posee cuatro anillos de privilegio:

- Anillo 0: es el más privilegiado, en él se puede hacer todo (kernel)
- Anillo 3: es el menos privilegiado. Impide que un proceso pueda sobreescribir un área de kernel con código malicioso.
- Anillos 1 y 2: no son empleados por Linux.

El kernel tiene como función “VIRTUALIZAR LA MÁQUINA”. Cualquier proceso cree que está solo en el sistema.

El kernel da estabilidad, robustez, facilita la vida a la hora de programar e independiza los programas del hardware.

Programar en área de kernel tiene muchos privilegios, ya que podemos hacer cosas con la máquina que en área de usuario no es posible realizar. Pero el código es más difícil de programar y depurar. Cualquier error es catastrófico.

La estructura del kernel GNU/LINUX es ***monolítica***.

¿Qué ve un proceso?

Un microprocesador 80x86 (sin PAE[1]) puede direccionar hasta ***4 Gbytes*** de memoria.

Un proceso cree que se encuentra solo en los ***tres primeros Gigas*** de memoria.

No puede, por tanto, reservar memoria que no quepa en estos 3 Gb, y no estarán disponibles para el proceso hasta que el sistema operativo nos reserve memoria.

El mecanismo que permite al sistema operativo asignarle memoria hasta 3 Gb teniendo menos memoria RAM a un proceso se denomina ***swap***. El swap es la utilización del disco duro como memoria virtual.

Si el proceso analiza el último Gbyte encontrará un conjunto de páginas en las que puede leer, pero no escribir. En estas páginas podrá ver información del propio proceso y código del kernel (***llamadas del sistema***).

Las **llamadas al sistema** se realizan de la siguiente manera:

- El proceso lanza la interrupción $\boldsymbol{80_{(16}}$
- El procesador busca en la tabla de descriptores de interrupción adónde debe saltar, y encuentra una dirección en el 4Gb que está viendo el proceso.
- El código del cuarto Gigabyte se ejecutará en el anillo 0.
- La función invocante ha dejado un valor en un registro del procesador (***%EAX***) que identifica la función a realizar que se ejecutará también en el anillo 0.
- Cuando la rutina de servicio de la llamada termina, se devolverá a la intrucción en área de usuario inmediatamente posterior a la que lanzó la interrupción.

Las **interrupciones hardware** se gestionan de la siguiente manera:

- Siempre se utiliza una interrupción distinta a la $\boldsymbol{80_{(16.}}$
- Como hablar con el hardware es lento y costoso, Linux lo resuelve de la siguiente manera:
  - Cuando salta la interrupción, Linux da una atención mínima al hardware y se asegura que el resto de las operaciones relativas a la interrupción serán realizadas en algún otro momento (***Top half***). En este momento las otras interrupciones no se contestan por estar enmascaradas.

1 PAE (Physical Address Extension): permite a los nuevos procesadores 80x86 direccionar hasta 64 Gb de memoria.

- El resto de acciones a realizar con la interrupción se ejecutarán con las interrupciones no enmascaradas (***Bottom half***).

Linux y los virus:

- Linux no puede tener virus por su arquitectura de memoria virtual.
- Un proceso no tiene forma de acceder a la memoria de los otros procesos porque no la tiene en su espacio de direcciones.

  No ve, tampoco, todo el kernel, y el que ve no lo puede modificar.
- En Linux, un ejecutable no puede infectar a otros.
- Lo que sí se puede producir es un troyano, pero habría que lanzarlo como root (superusuario)

Gestión de procesos

Todo proceso se crea realizando una llamada al sistema (clone).

El nuevo proceso recién creado es un duplicado, en cuanto a páginas de memoria se refiere, del padre. Cree que tiene toda la memoria para él solo y que es el único proceso que utiliza el procesador.

El kernel le ha virtualizado el sistema y, a partir de ahora, no verá al padre en el espacio de direcciones. Las modificaciones que se hagan en sus páginas de memoria no serán visibles para el padre.

Tampoco serán visibles las modificaciones que el padre haga en sus páginas de memoria.

Este mecanismo de creación de procesos basados en llamadas a clone, hace que en Linux los procesos generen un árbol de relaciones padre-hijo.

Copy on Write

- para crear un proceso se realiza una llamada a la función fork() que es un “wraper” de clone.
- Clone, cuando es invocado, crea un PCB para el hijo y avisa al sistema gestor de memoria para que, tanto padre como hijo, vean las mismas páginas de memoria.
- Estas páginas de memoria se marcan como de sólo lectura.
- Cuando padre o hijo intentan modificar alguna página se genera una excepción.
- El sistema gestor de memoria, entonces, recibe la excepción y duplica exclusivamente la página que se ha intentado modificar.

Estados de un proceso

- Planificado
- En ejecución: mientras está utilizando el procesador en modo usuario.
- En área de kernel: cuando el proceso hace una llamada al sistema el proceso cambia de

estado y pasa a ejecutar el código de área de kernel.

- Dormido: Cuando carece de un recurso para ejecutarse.
- Zombie: cuando un proceso muere debe notificárselo al padre enviándole una señal. Hasta que el padre no atiende la señal el proceso no muere. En este estado el proceso no consume recursos.

Planificador de procesos

- Expropiativo
- Round Robin multinivel dinámico:
    - varias colas con gestión Round Robin, pero con distinta prioridad cada una.
    - Se selecciona el proceso en la cola de mayor prioridad.
    - Sólo si esta cola está vacía pasamos a la de siguiente prioridad.
    - Es dinámico, ya que mientras va consumiendo Quantum su prioridad va cayendo. Si no utiliza el procesador va subiendo.
    - (-20) mayor prioridad, (19) menor prioridad.
- Es un plafinicador muy optimizado y su rendimiento no se ve prácticamente afectado cuando existe sobrecarga en el sistema.

PID

Número entero positivo que identifica unívocamente cada proceso.

Siempre que se referencia a un proceso se hace a través de su PID.

El PID se mantiene constante mientras el proceso viva.

Cuando el proceso muere se puede asignar ese PID a otro proceso que nazca en ese momento.

En Linux hay dos PID de asignación exclusiva:

0 ---> Tarea no existente.

1 ---> init.

(con ps afx comprobaremos que, en casi todas las máquinas, los primeros procesos tienen el mismo PID)

En Linux 2.4 los PID son short int, por lo que existe un máximo de 32767 procesos.

En Linux 2.6 los PID son enteros (32 bits), por lo que existe un máximo de 2147483647 procesos.

(cat /proc/sys/kernel/pid_max mostrará el número máximo de pid).

Gestión de memoria

La forma de entender la memoria en área de usuario y en área de kernel es totalmente diferente:

En área de usuario:

- Cuando programamos en área de usuario, habitualmente, hablamos de “solicitar una página de memoria” o de “reservar una página de memoria”.
- Desde el punto de vista estricto, esto no es así porque el proceso en área de usuario no entiende de páginas ya que cree que toda la memoria disponible es para él.
- Realmente, la razón por la que el proceso de usuario reserva memoria no tiene nada que ver con que exista memoria visible empleada por otros proceso, sino porque pregunta si, realmente, hay memoria disponible.

En área de kernel:

- El kernel puede ver tanto las direcciones físicas, como lógicas.
- La forma de trabajar en área de kernel con la memoria es compleja y, de momento, no veremos nada.

Módulos

Con los primeros kernel GNU/Linux la recompilación era casi imprescindible.

La memoria era muy escasa, por lo que no se podía permitir que el kernel soportara todo el hardware existente.

La recompilación del kernel permitía eliminar aquellos controladores que no eran necesarios.

A partir del kernel 2.0 se permite que algunas partes del kernel no estén enlazadas con el bloque principal del mismo en tiempo de compilación, sino que se puedan enlazar y desenlazar según sea necesario durante la ejecución del S.0.

En Linux prácticamente todo puede ser compilado en un módulo:

- soporte de protocolos de red.
- Sistema de ficheros.
- Características del procesador.
- Etc.

En Linux podríamos tener un módulo con funcionalidad mínima e ir cargando los soportes a dispositivos, protocolos, sistemas de fichero y otras características del kernel cuando sea necesario.

Todo esto no significa que Linux sea microkernel, ya que todos los módulos corren, una vez “activados” en el mismo nivel que el resto del kernel.

¿Qué es un módulo?

Un módulo es un trozo de kernel compilado, pero no enlazado con el bloque principal del mismo.

El enlace con el kernel se puede realizar en cualquier momento con:

$ depmod <nombre_módulo>

(se busca el m´doulo en el directorio /lib/modules/nombreversionkernel)

Se pueden listar los módulos cargados con:

$ lsmod

Descargamos el módulo de memoria con:

$ rmmod <nombre_módulo>

(aunque sólo podremos descargar un módulo que no esté siendo utilizado)

El directorio donde se encuentran los comandos anteriores es el /sbin.

Más comandos GNU/Linux

find: Muestra una lista con los archivos que coinciden con un criterio específico:

$ find [ruta] [opciones]

opciones:

-name: localiza los nombres de los archivos, directorios o enlaces simbólicos que coinciden con el modelo.

-type x, donde x:

f: localiza archivos

d: localiza directorios

l: localiza enlaces simbólicos.

-maxdepth n

restringe la búsqueda a n niveles de directorios.

Ejemplo

find /proc -name “pid_max” -type f

(Buscará el fichero pid_max a partir del directorio /proc)

enlaces simbólicos

Se crean con el comando ln -s nombre_fichero nombre_enlace.

Al realizar un “ls-l” aparece una l en la primera columna.

A diferencia del enlace duro (comando ln), crean únicamente una entrada en el directorio, por lo que su tamaño, aunque varíe el del fichero al que referencian, no cambia.

Inicio del sistema GNU/Linux

El programa init asume el control después de la carga del núcleo, pruebas de dispositivos, etc.

Este programa es es el padre de todos los procesos, la salida de ps siempre listará init como PID 1.

Sus acciones están dirigidas por el fichero /etc/inittab

Fichero inittab

Formato del fichero:

id:nivel de ejecución:acción:comando

Siendo:

-**id**: identificador de 1 a cuatro caracteres alfanuméricos que representa una acción concreta que se realiza mientras el sistema se inicia.

-**nivel de ejecución** (no son estándard, varían con cada distribución, aunque estos son los más comunes):

0.- Para el sistema.

1.- Monousuario

2-5.- Multiusuario.

6.- Reinicia el sistema

-**acción**: que realiza el proceso init. Las más comunes:

once: el comando se ejecuta una vez al entrar en el nivel de ejecución especificado

wait: lo mismo que once, pero init espera a que termine el comando antes de continuar con otras entradas de inittab

-**respawn**: se supervisa el proceso y se inicia una nueva instancia si el proceso original se finaliza.

-**sysinit**: el comando se ejecuta durante la fase de arranque del sistema. Se ignora el campo nivel de ejecución.

-**initdefault**: el nivel de ejecución que hay que ejecutar después de completar las acciones boot y sysinit.

-**ctrlaltdel**: atrapa la secuencia Ctrl-Alt-Del que reinicia el sistema.

Directorio /etc/rc<nivel de ejecución>.d

En estos directorios aparecen enlaces simbólicos a scripts situados en el directorio /etc/init.d

El nombre de estos enlaces tiene el siguiente formato:

- Todos comienzan por K o S.
- Detrás de la K o de la S aparece un número de dos dígitos.
- En la cuarta posición (en algunos casos la quinta si se han utilizado 3 números en lugar de dos) aparece un nombre de script que debe encontrarse exactamente igual en el directorio /etc/init.d

Directorio /etc/init.d

En este directorio se encuentran los scripts que inician, paran, restablecen o indican el estado de ejecución de los diferentes servicios del sistema.

Todos los scripts tienen la estructura siguiente:

- Un bloque start

- Un bloque restart
- Un bloque stop
- Un bloque status

En cada uno de estos bloques aparecen las acciones que permitirán, como ya se ha comentado, la parada, inicio, restauración o muestra del estado de dicho servicio.

Por ejemplo: si quisiéramos lanzar el servicio smb (samba) bastaría con teclear en modo root lo siguiente:

$ /etc/init.d/smb start

Si lo que quisiéramos fuera parar el servicio:

$ /etc/init.d/smb stop.

Script /etc/init.d/rc

Este script se ejecuta pasándole como parámetro el nivel de ejecución.

Vamos a observar las siguientes líneas (se han modificado para una mayor claridad) de este script:

```
runlevel=$1
for i in /etc/rc$runlevel.d/K[0-9][0-9]*
        do
                $i stop
        done
for i in /etc/rc$runlevel.d/S[0-9][0-9]*
        do
                $i start
        done
```

El script recorre el directorio rc<nivel de ejecución>.d y para todos los servicios cuyo enlace simbólico comience con una K utilizando como orden el número que aparece detrás de la K..

A continuación, recorre el mismo directorio para “lanzar” los servicios cuyo enlace simbólico comience con una S utilizando como orden el número que aparece detrás de la S.

Nota: podría ocurrir que dos Scripts K o S tuvieran el mismo número de orden.

Ejemplo de inicio con un fichero inittab concreto

```
# Nivel de inicio por defecto
id:2:initdefault:

si::sysinit:/etc/init.d/rcS


#  0 para el sistema.
#  1 monousuario.
#  2-5 multiusuario.
#  6 reinicia el sistema

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
```

ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3

¿Cómo es la secuencia de inicio con este fichero inittab?

1º.-Con este inittab el proceso init comprueba que el nivel de inicio por defecto es el 2 (***id:2:initdefault:***)

2º.-Realiza, sin que importe el nivel de inicio, una llamada al script de /etc/init.d/rcS (***si::sysinit:/etc/init.d/rcS***) que estudiaremos en siguientes apartado.

3º.-Ejecuta el script /etc/init.d/rc con el parámetro 2, nivel de inicio que se indicó en el paso primero (***l2:2:wait:/etc/init.d/rc 2***)

4º.- El script rc parará todos los servicios que tengan un enlace simbólico que empiece por K en el directorio /etc/rc2.d.

5º.- El script rc iniciará todos los servicios que tengan un enlace simbólico que empiece por S en el directorio /etc/rc2.d.

6º.- El proceso init capturará la combinación de teclas Ctr-alt-del.

7º.- el proceso init lanzará los procesos getty 38400 tty1, getty 38400 tty2 y getty 38400 tty3 y los "vigilará", es decir, si se cerrara una instancia de estos procesos volvería a lanzarlos como le indica la acción respawn.

Modificación del inicio del sistema

Añadir nuevos servicios:

1º.- Instalaremos el script de ejecución en el directorio /etc/init.d
2º.- Crearemos los enlaces simbólicos K o S en los directorios pertinentes.

Por ejemplo, si quisiéramos añadir el servicio mi_servicio para que se ejecutara en el nivel de inicio 3 bastaría con teclear lo siguiente:

```
$ ln -s /etc/init.d/mi_servicio /etc/rc3.d/S99mi_servicio
$ ln- s /etc/init.d/mi_servicio /etc/rc6.d/K99mi_servicio
$ ln -s /etc/init.d/mi_servicio /etc/rc0.d/K99mi_servicio
```

Es una buena política de gestión de sistemas el añadir un script de parada de servicio en las niveles que el sistema va a pararse (0 y 6).

Eliminar servicios:

Aquí bastará con eliminar los enlaces simbólicos que hagan referencia al servicio.
Es una buena política cambiar el nombre del enlace simbólico ya que si, en un futuro, se quisiera volver a lanzar el servicio no tendremos nada más que volver a renombrar el script.

Nota: En algunas distribuciones, como Fedora y Mandrake, existe una utilidad para modificar, en modo terminal, el inicio del sistema. Esta utilidad se llama "chkconfig".
También es posible modificar el inicio del sistema mediante utilidades gráficas.

Script lanzado en la acción sysinit
Estos scripts (/etc/init.d/rcS en el ejemplo, /etc/init.d/rc.sysinit en Fedora) no tienen en cuenta el nivel de

ejecución ya que realizan acciones válidas para todos ellos.

Estas acciones son: elegir un fichero de disposición de teclado, comprobar y mnontar los sistemas de archivos raíz y proc, establecer el reloj y el nombre de la máquina, configurar el espacio de intercambio, eliminar los ficheros temporales y cargar los módulos del sistema.

El directorio en el que leen los scripts K o S suele ser el /etc/rcS.d.

Comandos relacionados con el inicio del sistema

init:

Lanza el proceso init con el nivel de inicio como parámetro. Por ejemplo:

$ init 0

Apagaría la máquina

runlevel

Muestra el nivel de ejecución actual del equipo. Muestra una N si no hubo nivel de inicio anterior y una S si sí lo hubo.
Ejemplo:

$ runlevel
N 2

Indica que el nivel de ejecución actual es el 2 y que no hubo inicio anterior.