Procesos

Arquitectura y Sistemas Operativos.
Tecnicatura Superior en Programación.
UTN-FRA

Autores: *Prof. Martín Isusi Seff*

Revisores: *Prof. Marcos Pablo Russo*

Versión: 1

Esta obra está bajo una Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional.

Proceso de arranque en GNU/Linux

Para poder empezar a hablar de procesos en GNU/Linux, y del manejo de los mismos, es necesario conocer cómo es el proceso de arranque del sistema.

El proceso de arranque consta de cuatro etapas:

1. BIOS
2. Bootloader
3. Kernel
4. Init

Primera etapa: BIOS

Esta etapa inicia en el momento en que se enciende la PC. Es allí cuando el BIOS (Basic Input Output System) toma el control del sistema para poder realizar operaciones básicas de hardware (reconocimiento, prueba de la memoria, etc.). Una vez que el BIOS completa las operaciones, se encargará de cargar en memoria el bootloader que inicia la segunda etapa.

Es importante mencionar que esta etapa es independiente del sistema operativo instalado. El BIOS se encuentra en una memoria propia de la placa base de la PC y, si bien este sistema puede ser actualizado desde el sistema operativo, no forma parte del mismo.

Segunda etapa: Bootloader

El bootloader o cargador de arranque, es un programa encargado de iniciar el sistema operativo instalado. Este programa se guarda en una porción del disco llamada MBR (Master Boot Record). El MBR ocupa solo 512 bytes en el disco, de los cuales 2 bytes corresponden al "magic number", 64 bytes a la tabla de particiones y 446 bytes al bootloader. El "magic number" corresponde a un número que sirve de "bandera" y representa el inicio del MBR. La tabla de particiones contiene cada partición del disco y la información de cada una de ellas. Por último, el bootloader será el encargado de iniciar el resto del sistema. El bootloader puede contener la información de distintos sistemas operativos, por ejemplo, si una PC tiene instalado Microsoft Windos y Ubuntu, será el bootloader el encargado de mostrar una lista de los sistemas operativos instalados y luego iniciar el sistema deseado.

En GNU/Linux, existen dos bootloaders principales: LILO y GRUB. Si bien LILO prácticamente ya no se usa, es interesante explicar sus características. LILO sólo soporta hasta 16 sistemas operativos instalados y no tiene la posibilidad de iniciar alguno desde la red. No contiene una consola interactiva que permita modificar los parámetros de inicio de un determinado sistema operativo y, por último, si luego de instalado nuestro sistema, realizamos un cambio que requiera modificaciones del bootloader, será necesario modificar los archivos de configuración de LILO y reescribir el booloader en el MBR.

Por otro lado, GRUB es uno de los bootloaders más utilizados para sistemas operativos GNU/Linux. Las características más importantes son la posibilidad de manejar una cantidad ilimitada de sistemas operativos y de bootear por red, contiene una interfaz de línea de

comandos interactiva que permite establecer parámetros al iniciar un sistema operativo y, por último, si se realiza un cambio que requiera modificaciones en la configuración de GRUB, solamente es necesario modificar cambiar los archivos de configuración sin reescribir el MBR.

Tercera etapa: Kernel

Antes de comenzar de lleno en cómo se ejecuta esta etapa, es necesario recordar algunos aspectos generales del kernel.

El kernel es el componente fundamental de cualquier sistema operativo, ya que es el encargado de comunicar los programas que solicitan recursos y el hardware. Asignará los tiempos de ejecución para cada programa, gestionará cada una de las tareas que realiza el sistema operativo, el hardware del equipo y el sistema de archivos utilizado.

El kernel puede presentarse desarrollado en distintas arquitecturas, monolítico, modular o híbrido. La arquitectura de kernel monolítico implica que todos los módulos necesarios para controlar el hardware del equipo se encuentran cargados en un único archivo comprimido, mientras que los kernel desarrollados bajo una arquitectura modular se construyen con cada módulo compilado como un objeto que puede ser cargado o descargado según sea necesario.

El proceso de carga del kernel se realiza en dos etapas: etapa de carga y etapa de ejecución.

El kernel se encuentra almacenado de forma comprimida en la memoria secundaria del sistema (generalmente un disco rígido, pero puede ser un pendrive, etc.), por lo que la etapa de carga del kernel se encargará de descomprimir el kernel y copiarlo entero en la memoria principal (RAM). Además de la carga del kernel en la memoria principal, también se cargarán los drivers necesarios mediante un proceso llamado initrd. Este proceso creará un sistema de archivos temporal que sólo es utilizado durante la fase de carga.

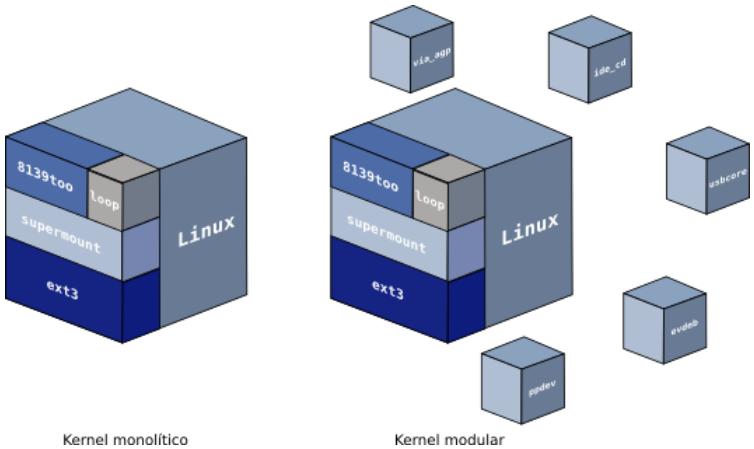

Figura 1

Cuarta etapa: init

Antes de comenzar a hablar del proceso init, debemos definir 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 ejecutan sin la intervención del usuario y que hacen que el sistema

sea utilizable. Cada proceso contiene un conjunto de estructuras de datos y una dirección en la memoria principal. En esa dirección de memoria se reserva el espacio para que se realice la copia del código del proceso, el área para los datos del mismo, pila del proceso e información adicional utilizada por el sistema durante la ejecución. Existen dos tipos de procesos, los **procesos de usuario**, que son aquellos procesos ejecutados directamente por el usuario, y los **procesos demonio**. Los procesos demonio son aquellos que no requieren la intervención del usuario y se ejecutan en un segundo plano.

Una vez que la etapa de carga y ejecución del kernel se completa, se iniciará el proceso **init**. Este proceso es ejecutado por todos los sistemas basados en Unix y es el responsable de la inicialización de todos los nuevos procesos excepto el proceso swapper. Init se conoce como un proceso *dispatcher* o planificador, encargado de decir qué proceso se ejecutará y cuáles serán copiados/borrados de la memoria principal. A partir del momento de ejecución de init, éste es conocido como el proceso 1. Dentro del directorio /etc se encuentra el archivo inittab que corresponde al archivo de configuración del proceso init.

En GNU/Linux, los procesos se ejecutan de forma jerárquica: cada proceso es lanzado desde un proceso "padre", por lo que el proceso lanzado se llama "hijo". Podemos decir entonces, que todos los procesos ejecutados luego del inicio del sistema son procesos "hijo" de init.

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 un proceso

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.

Para poder ver los procesos en ejecución en tiempo real, se puede utilizar el comando *top.*

Estado de los procesos

Existen cinco estados posibles que puede adoptar un proceso:

- *Ejecutándose (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 les 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.

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

Para poder imprimir esta tabla desde la terminal, se debe correr el comando *kill -l*

Enviar señales a un proceso

Sólo el propietario del proceso y el superusuario (root) pueden mandar señal KILL a un proceso. Por defecto la señal es 15 (TERM). Para enviar una señal a un proceso, se usa el siguiente comando:

```
kill [-señal] PID
```