6. Arranque del sistema

Elena García-Morato, Felipe Ortega, Enrique Soriano, Gorka Guardiola

GSyC, ETSIT. URJC.

Laboratorio de Sistemas (LSIS)

9 marzo, 2023

Contenidos

6.1 Secuencia de arranque del sistema

La secuencia de arranque en Ubuntu

- A continuación describiremos el proceso de arranque de una máquina Ubuntu GNU/Linux en Intel 64-bit.
- La secuencia es:
  1. Firmware.
  2. Cargador primario.
  3. Cargadores secundarios.
  4. Kernel.
  5. Área de usuario (`init`).

Secuencia de arranque en Linux

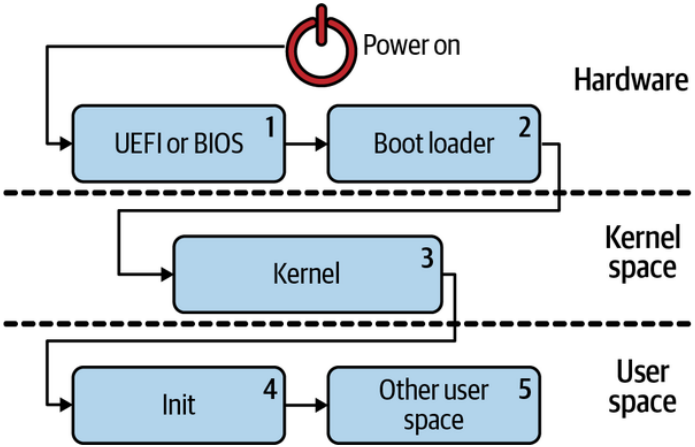

Figura 1: Esquema de la secuencia de arranque de un sistema GNU/Linux. Fuente: [1]

6.2 El firmware

Firmware

- Es el software que viene integrado en el hardware (por ejemplo, en una memoria en la placa madre). Hay distintos tipos.
- Sus funciones principales son:
  - Inicializar el hardware de la máquina: enumerar los dispositivos, inicializar el controlador de memoria DRAM, la gestión de energía, el sistema de gestión del sistema (SSM, Ring -2), etc.
  - Ofrecer una interfaz para interoperar con algunos componentes hardware.
  - Iniciar el arranque del sistema (de disco o de red).
- Normalmente se puede actualizar (**cuidado**: proceso delicado, aunque las placas modernas permiten guardar versiones y revertir si hay problemas).
- Los PCs antiguos usan **BIOS** (Basic I/O System). Los sistemas más modernos usan **UEFI** (puede emular una BIOS si se activa el modo *legacy*).

UEFI: Unified Extensible Firmware Interface

- Es un conjunto de especificaciones desarrolladas por el UEFI forum, que definen la arquitectura de la plataforma *firmware* para arranque del sistema y su interfaz con el S.O. (véase UEFI Forum FAQ).
- UEFI tiene su propio intérprete de comandos y se pueden desarrollar *aplicaciones EFI* (ficheros extensión `*.efi`).
- El cargador del S.O. es en realidad una aplicación EFI.
- UEFI también puede arrancar mediante el Preboot eXecution Environment (PXE), un entorno que permite cargar remotamente el S.O. a través de la red en lugar de leer el disco local.

UEFI: Unified Extensible Firmware Interface

- UEFI tiene configurado un orden de arranque (CD/DVD, discos internos, discos externos, red)... irá intentando arrancar de ellos.
- Todo disco con formato GPT (GUID Partition Table) debe tener una (y solo una) partición ESP (EFI System Partition), que es en realidad una partición FAT.
- El programa `parted` la llama partición *boot* y el instalador *biosgrub*. Todos estos términos designan el mismo elemento.
- En ese sistema de ficheros habrá directorios con los cargadores de los sistemas operativos instalados en ese disco.

UEFI: Unified Extensible Firmware Interface

- Es partición suele estar montada en `/boot/efi`.
- La aplicación EFI que se ejecuta por omisión es:

```
/boot/efi/EFI/BOOT/BOOTX64.EFI
```

- El comando Linux

```
efibootmgr -v
```

muestra las variables de UEFI. La variable `BootOrder` indica el orden de ejecución de las aplicaciones EFI (cada una con su dispositivo y ruta). El comando también permite modificar variables, borrar entradas, etc. (**mucha precaución**).

UEFI: Unified Extensible Firmware Interface

- Ejemplo de salida del comando `efibootmgr -v`.

```
$ efibootmgr -v
BootCurrent: 0001
Timeout: 1 seconds
BootOrder: 0001,0002,0003,0004
Boot0001* ubuntu    HD(1,GPT,a820e124-f6f4-4a6b-9500-b6eef1236f0c,0x800,0x100000)
                                                    /File(\EFI\ubuntu\shimx64.efi)
Boot0002* UEFI:CD/DVD Drive BBS(129,,0x0)
Boot0003* UEFI:Removable Device BBS(130,,0x0)
Boot0004* UEFI:Network Device BBS(131,,0x0)
```

6.3 Cargador

Cargador

- Existen distintos tipos de cargadores para Linux. GRUB es el más común, en su versión 2 (la versión 1 se llama GRUB *legacy*).
- Algunos son además gestores de arranque (Boot Managers): permiten seleccionar y lanzar otros cargadores para arrancar distintos sistemas operativos (p. ej. GRUB).
- El objetivo del cargador es seleccionar y encontrar la imagen del kernel que se quiere arrancar en una partición del disco, cargarla en memoria y saltar a ella.

Cargador

- Un cargador puede tener distintas fases (stages): cargador primario, secundario, etc.
- Con BIOS es necesario tener múltiples fases, el cargador primario tiene que ser muy pequeño. Con UEFI el cargador puede ser grande.
- En UEFI no es estrictamente necesario usar un cargador (puede ejecutar un kernel configurado para tal cosa), pero es lo habitual.
- GRUB dispone de:
  - Un intérprete de comandos propio.
  - Driver para entender particiones EXT.

GNU GRand Unified Bootloader (GRUB)

- GRUB estará configurado con la lista de imágenes del *kernel* que tenemos en las particiones de disco, para que pregunte cuál debe arrancar o arranque una directamente sin mostrar menú, etc.
- Pulsando `Esc` cuando carga GRUB podemos acceder al menú de GRUB (si estamos con BIOS, presionando `Shift`).
- Podremos activar el intérprete de comandos de GRUB para realizar operaciones avanzadas, pasar parámetros al *kernel*, etc.
- Al final, traerá la imagen del kernel a memoria, leerá y rellenará ciertas cabeceras (hdr) y saltará al punto de entrada del núcleo, denominado *linux setup*.

Linux setup

- Es el código en Real Mode de Linux. Cuando un Intel/AMD arranca, está en este modo de emulación del Intel 8086 16-bit, direcciones de 20 bits, menos de 1 MB de RAM disponible, sin paginación, etc.
- A estas alturas, puede que el resto del kernel esté comprimido.
- Configura los segmentos de memoria, pila, montón, etc.
- Recolecta información sobre el hardware.

6.4 Kernel

Kernel

[Continuación]

- Pasar la CPU a *Protected Mode* (32-bit). Este modo ya tiene memoria virtual.
- Inicializar consola, detectar memoria, inicializar teclado y video, etc.
- Pasar la CPU a *Long Mode* (64-bit).
- Descomprimir el resto del kernel y relocalizar el código, configurar tabla de interrupciones, ...
- Saltar a `start kernel`.

Kernel

[Continuación]

- Configura la CPU0, por ahora no hay más.
- Imprime el *Linux Banner* en la consola.
- Reserva memoria para `initrd`: que es el *RAM disk* de inicio.
- Inicialización de entrada/salida, PCI, SMP, DMA, etc.
- Inicialización del planificador.
- Inicializacion de cachés.
- Inicializacion del sistema de ficheros VFS.

Kernel

[Continuación]

- En este punto se empiezan a crear **procesos**:
  - PID 0: **`cpu_idle`**. Su función es ejecutar cuando no hay nada que hacer.
  - PID 1: **`init`**. Lo vemos más adelante.
  - PID 2: **`kthreadd`**. Gestionará la creación de hilos del kernel.
- A continuación, se activan el resto de las CPUs.

Kernel: `initrd`

Se monta **`initrd`**.

- Initrd (*Initial RAM Disk*) es un sistema de ficheros que se usa para poder arrancar el sistema completo. Actualmente, se usa initramfs, que usa el driver `tmpfs` [1].
- Se monta en el raíz de forma temporal. ¿Por qué?
  - Hay muchos drivers distintos para todos los tipos de hardware de almacenamiento que existen. En algunos casos requieren comandos.
  - Son módulos del *kernel* que se necesitarán cargar para poder montar el sistema de ficheros raíz verdadero.

[1] Pero se sigue llamando `initrd`

Kernel: `initrd`

- Todo ese sistema de ficheros está en un fichero `/boot/initrd.img-*`. Si quieres extraer a mano uno y ver qué lleva, usa el comando `unmkinitramfs`.
- Una vez montado el `initrd` en `/` , el proceso con PID 1:
  1. Ejecuta el script `/init` para montar `sysfs` y `procfs`, crear distintas variables de entorno, ejecutar otros *scripts* de arranque, cargar módulos necesarios, gestor de volúmenes lógicos (LVM), etc.
  2. Ejecuta `/sbin/run-init` para montar el raíz de verdad y eliminar de la memoria el `intird`.
  3. Ejecuta `/sbin/init` (del raíz de verdad).

6.5 init

Init

- El proceso con PID 1 quedará ejecutando el programa `/sbin/init` hasta que se apague la máquina.
- En la mayoría de distribuciones Linux actuales, `init` es `systemd`. Nos centraremos en este.
- En otras, es `System V init` (`sysvinit`). Hay muchos disponibles [2] .
- Todos los procesos de usuario serán hijos de `init` (ejecuta `pstree` para ver el árbol genealógico).

[2] http://without-systemd.org/wiki/index.php/Alternativestosystemd

Init

Los propósitos de **`init`** son:

- Montar los sistemas de ficheros configurados.
- Crear los procesos que terminan ejecutando la interfaz gráfica, programas de login, etc.
- Iniciar y parar los servicios (demonios).
- Liberar recursos cuando mueren procesos.
- Esperar a que se ordene el apagado o reinicio del sistema, la activación/desactivación de un servicio, etc.

En este punto el sistema ya ha arrancado por completo.

6.6 systemd

systemd

`systemd` es mucho más que un `init`. Es una colección de *demonios*, bibliotecas, programas y componentes del núcleo. Hace muchas cosas extra que antes hacían otros *demonios*:

- Ejecución periódica de programas.
- Ejecución de servicios en demanda (activación perezosa).
- Controla los *logins*.
- Maneja la configuración de dispositivos que se conectan.
- Controla las bitácoras.
- Información de localización (locales).
- Controlar conexiones de red.
- ...

systemd

Comandos (hay que ejectuarlos como `root`):

- `systemctl`: Permite controlar y dar órdenes a `systemd`.
- `systemd-analyze`: Permite analizar y depurar `systemd`.

systemd

- Maneja el concepto de *unit*.
- Un tipo de *unit* es cualquier cosa que puede manejar `systemd`.
- Cada cosa concreta es una *unit*.
- Una *unit* se puede activar o desactivar. Activar significa arrancar, realizar, montar, vincular, etc. Depende del tipo de *unit*.
- Existen muchos tipos, p. ej.:
  - `Service units`: controlan los servicios (demonios).
  - `Mount units`: controlan los sistemas de ficheros.
  - `Target units`: controlan otras units.
  - `Path units`: vigilan rutas para reaccionar si cambian ficheros/directorios.

systemd

- Las *units* están definidas por ficheros, cuyo nombre es `nombre-de-unidad.tipo`.
- Son ficheros de texto plano.
- Este comando nos da la ruta donde se instalan. Esos ficheros no se deben modificar:

```
pkg-config systemd --variable=systemdsystemunitdir
```

- Por omisión es:

```
/lib/systemd/system
```

systemd

- También hay otras rutas con *units*, por ejemplo:

```
/etc/systemd/system
```

- Esas sí se pueden modificar si es necesario y tienen más precedencia.
- Las de usuario deben estar en el directorio que muestre este comando:

```
pkg-config systemd --variable=systemduserunitdir
```

systemd

Unas *units* dependen de otras. Los tipos básicos de dependencia son:

- `Requires`: es estricta, se activará y si falla la activación de la dependencia, no se puede activar esta *unit*.
- `Wants`: se intentará activar la dependencia, pero si falla no importa.
- `Requisite`: debe estar activa ya, si no lo está, falla la activación de esta *unit*.
- `Conflicts`: si está activa, se desactivará, no es compatible con esta *unit*.

systemd

Para indicar orden, se pueden usar estos modificadores:

- `Before`: Esta *unit* se activará antes de las *units* indicadas.
- `After`: Se activará después de las indicadas.

También se pueden poner condiciones. Por ejemplo:

- `ConditionPathExists=p`: Si el fichero p existe, se activa.
- `ConditionPathIsDirectory=p`: Si existe y es directorio.

systemd: comandos

```
systemctl start nombre-de-unit
```

- Así se arranca un servicio.
- Si en lugar de `start` ponemos `stop`, se para el servicio.
- Si ponemos `restart`, se está *reiniciando* el servicio (por ejemplo, para que recargue su configuración). Algunos servicios entienden también `reload` para recargar su configuración.

systemd: comandos

`systemctl enable nombre-de-unit`

- Así se habilita un servicio: se arrancará automáticamente cuando se inicie el sistema. No se arrancará ahora.
- Si en lugar de `enable` ponemos `disable`, se *deshabilita* el servicio.
- Se puede arrancar manualmente una unit que está deshabilitada.

systemd: comandos

```
systemctl status nombre-de-unit
```

- Forma de comprobar el estado de un servicio.
- Se mostrará un resumen del servicio, desde cuándo lleva arrancado, cuál es su fichero, información sobre sus procesos, las últimas líneas de su bitácora...

systemd: comandos

`systemctl list-units`

- Muestra todas las *units* activas.
- Se mostrarán columnas con el nombre de la unidad, si su configuración está cargada en su memoria, si está activa (se arrancó bien) y una descripción.
- Con el modificador `--all` nos muestra todas las *units* (activas e inactivas).
- Con el modificador `--state=estado` nos muestra las *units* con ese estado.

systemd: comandos

`systemctl list-unit-files`

- Muestra todas los ficheros de las *units*.
- `static` indica que dicha *unit* no puede ser habilitada, porque es una *unit* que simplemente realiza una acción, normalmente como dependencia de otra *unit*.
- `masked` indica que no se permite arrancarla.

`systemctl show nombre-de-unit`

- Muestra las propiedades de la *unit*.

systemd: comandos

`systemctl list-dependencies nombre-de-unit`

- Muestra las dependencias de la *unit* (*Required* o *Wanted*).
- Con el modificador `--all` lo hace recursivamente.
- Con el modificador `--reverse` muestra las *units* que dependen de esta *unit*.

systemd: comandos

`systemctl edit --full nombre-de-unit`

- Sirve para editar el fichero de la *unit*.
- Cuando se termina la edición, se escribe el fichero de la *unit* en `/etc/systemd/system`, que es el directorio con más precedencia para las *units*.
- **Cuidado**: debemos saber bien lo que hacemos.

`systemctl daemon-reload`

- Recarga `systemd`.

systemd: comandos

`systemctl halt`

- Apaga el sistema.
- Con `poweroff` hace un apagado total.
- Con `rescue` pone el sistema en modo *single-user* para recuperación.

Runlevels

- El *runlevel* es un concepto de System V. Describe una configuración del sistema para un modo de operación.
- Cuando se activa un *runlevel,* se paran todos los servicios que no hacen falta y se activan los que si hacen falta.
- `systemd` lo hace con sus *targets*.

Runlevels

Comandos equivalentes:

- `runlevel 0`: apagar → `systemctl poweroff`
- `runlevel 1`: *single-user* → `systemctl rescue`
- `runlevels 2,3,4`: *multi-user* → `systemctl isolate multi-user.target`
- `runlevel 5`: *multi-user* con entorno gráfico → `systemctl isolate grafical.target`
- `runlevel 6`: reinicio → `systemctl reboot`

6.7 Entorno gráfico

El entorno grafico

- En un sistema de escritorio, lo último que se arranca es el entorno gráfico.
- Suele arancarse un programa que es la pantalla gráfica de login (como **gdm**, **sddm** o **lightdm**) y este arranca el sistema de ventanas X *windows* (**xorg**) que a su vez ejecuta una sesión con el manejador de ventanas, menús, etc. (**unity-session**, **gnome-session** o **lxsession**).
- Las X (Xorg) se pueden arrancar también a mano con `startx` desde una consola de texto, y puede haber varias arrancadas (`Ctrl-Alt-F1`, `Ctrl-Alt-F2`... conmuta entre consolas de texto o gráficas).
- Consulta las páginas de manual de los programas en negrita en esta diapositiva, en particular **xorg**.

El entorno grafico: Wayland

- Una de las ventajas de Xorg es que está tan integrado con todos los elementos del kernel Linux y el *userspace*. Se ha convertido en el servidor gráfico por defecto en Linux durante mucho tiempo.
- Sin embargo, su crecimiento ha sido muy caótico en muchos aspectos. Actualmente, es muy complicado realizar el mantenimiento o contribuciones al código de Xorg.
- Además, es poco eficiente porque los gráficos se pintan mediante una API en la que tenemos que dibujar mediante arrays de pixeles.
- El *kernel* Linux dispone ahora de un subsistema llamado Direct Rendering Manager (DRM) que implementa la interfaz con la GPU.
- Desde la versión 22.04, Ubuntu utiliza por defecto un gestor gráfico alternativo a Xorg llamado Wayland. Este gestor comenzó a desarrollarse en 2008 y utiliza DRM.

El entorno grafico: Wayland

- Wayland gestiona directamente los permisos de compartición de recursos gráficos (por ejemplo, cuando queremos compartir una pantalla o la ventana de una aplicación en una sesión de videoconferencia).
- Tiene un diseño mucho más ligero y mantenible que Xorg.
- A cambio, se ha probado mucho menos en entornos en producción y, por tanto, está sujeto a cambios más drásticos y a que surjan fallos.
- Para más información puedes consultar el capítulo 14 de [3].

6.8 Referencias

Para saber más

- El capítulo 6 de [1] explica con detalle el proceso de arranque de un sistema GNU/Linux, así como otros sistemas alternativos de ejecución de procesos (como contenedores virtuales).
- El libro de referencia sobre administración en Unix/Linux [2] incluye mucha información adicional y excelentes explicaciones:
  - Capítulo 3: El proceso de arranque y parada del sistema.
  - Capítulo 11: Archivos de registro de eventos del sistema (*syslog*).
- El capítulo 5 y el capítulo 6 de [3] también describen extensamente tanto el proceso de arranque del *kernel* como el inicio del espacio de usuario.

Referencias I

[1] M. Hausenblas. *Learning Modern Linux*. O'Reilly Media, abr. de 2022.

[2] E. Nemeth y col. *Unix and Linux System Administration Handbook*. 4ª ed. Pearson, 2010.

[3] B. Ward. *How Linux Works: What Every Superuser Should Know*. 3ª ed. No Starch Press, abr. de 2021.