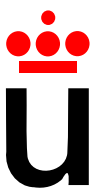
Arranque del sistema y servicios

Laboratorio de Sistemas

Enrique Soriano, Gorka Guardiola

GSYC

1 de marzo de 2021



Universidad
Rey Juan Carlos

Arranque del sistema

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)

Firmware

- Es el software que viene incrustado en el hardware (P. ej. en una ROM 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).

Firmware

- El firmware suele ser actualizable (cuidado!).
- Los PC actuales usan **UEFI**. Los PCs antiguos usan **BIOS** (UEFI puede emular una BIOS si se activa el modo *legacy*).

Unified Extensible Firmware Interface

- UEFI tiene su propio intérprete de comandos y se pueden desarrollar *aplicaciones* EFI (ficheros `*.efi`).
- En realidad el cargador del sistema operativo es una *aplicación* EFI.
- UEFI también puede arrancar con Preboot eXecution Environment (PXE) para poder traer el sistema por red en lugar del disco.

Unified Extensible Firmware Interface

- UEFI tendrá 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*, son todas lo mismo.
- En ese sistema de ficheros habrá directorios con los cargadores de los sistemas operativos instalados en ese disco.

Unified Extensible Firmware Interface

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

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

- El comando de Linux

  `efibootmgr -v`

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

Cargador

- Hay distintos cargadores para Linux. GRUB es el más común (la versión 2, la primera 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.

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...
- A estas alturas, puede que el resto del kernel esté comprimido.
- Configura los segmentos de memoria, pila, montón, etc.
- Recolecta información sobre el hardware.

Kernel

Seguimos...

- Pasar la CPU a Protected Mode (32-bit). Este modo ya tiene memoria virtual.
- Inicializar consola, detectar memoria, inicializar teclado y video, ...
- 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

- 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 caches
- Inicializacion del sistema de ficheros VFS

Kernel

Se empiezan a crear **procesos**:

- PID 0: **cpu_idle**. Su función es ejecutar cuando no hay nada que hacer.
- PID 1: **init**. Lo veremos más tarde.
- PID 2: **kthreadd**. Gestionará la creación de hilos del kernel.

Y se activan el resto de las CPUs.

Kernel: initrd

Montar **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, 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).

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/Alternatives_to_systemd

Init

Los objetivos 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.

¡El sistema ya ha arrancado!

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 ejecutarlos como root):

- `systemctl`: comando que permite controlar y dar órdenes a systemd.
- `systemd-analyze`: comando que 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 está parando 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 está deshabilitando el servicio.
- Se puede arrancar manualmente una unit que está deshabilitada.

systemd: comandos

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

- Así se comprueba 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.

systemd: comandos

`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.
- Evidentemente, debemos saber bien lo que hacemos.

systemd: comandos

```
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

Los comandos equivalentes son:

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`

Entorno gráfico

- 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** o **sddm**) 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).
- Mira las páginas de manual de los programas en negrita en esta transparencia, en particular **xorg**.