Procés d'arrencada d'un sistema GNU/Linux

Índex

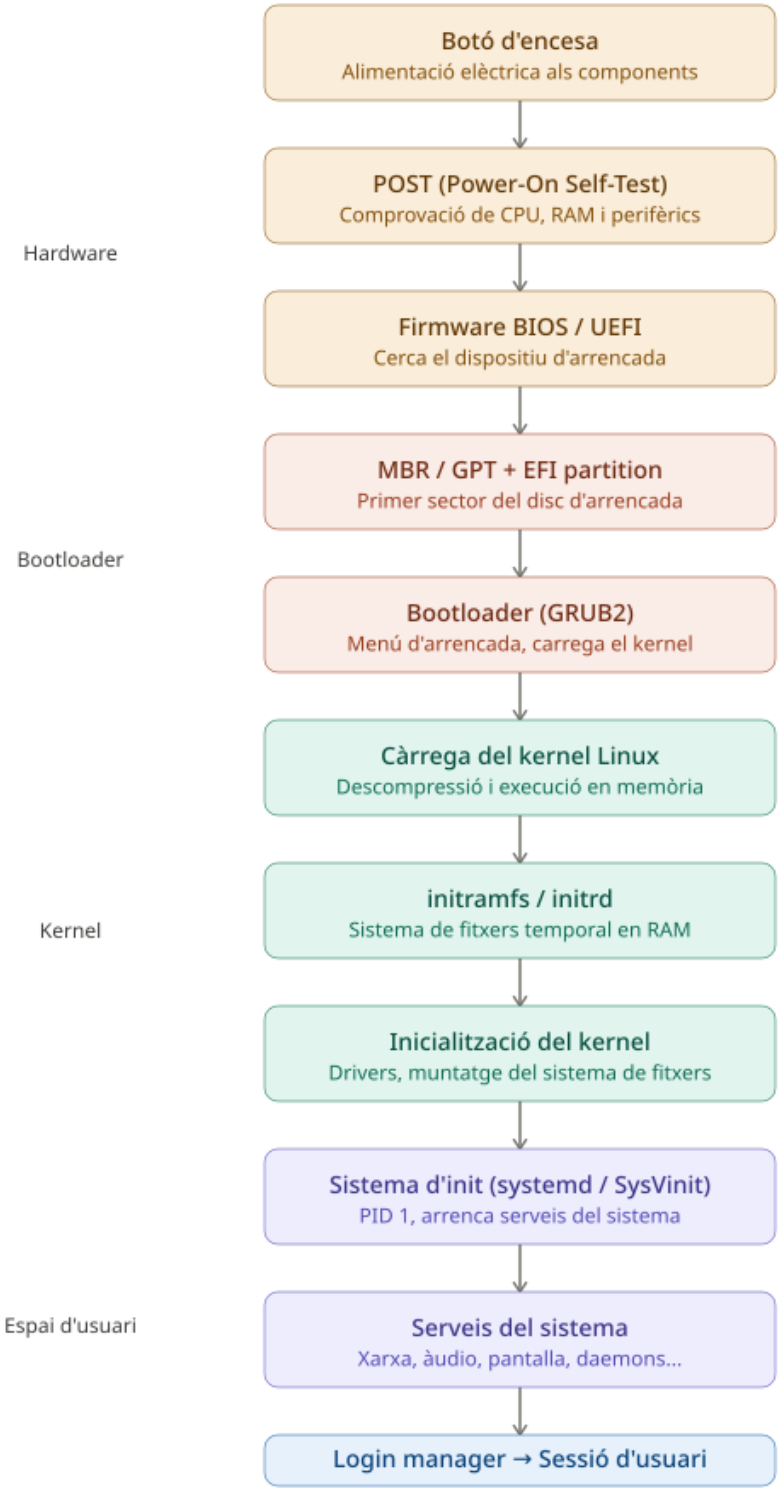

Figura 1: Arrencada d'un sistema GNU/Linux

1. Hardware i POST Quan prems el botó d'encesa, la placa base envia alimentació a tots els components. La CPU executa immediatament el codi emmagatzemat al xip del firmware. El POST (Power-On Self-Test) comprova que la memòria RAM, la CPU i els

perifèrics bàsics funcionen correctament. Si hi ha algun error crític, el sistema s'atura aquí (sovint amb bips o codis de llum LED).

2. Firmware BIOS / UEFI El firmware cerca el dispositiu d'arrencada (disc dur, SSD, USB…) seguint l'ordre configurat. En sistemes antics amb BIOS, llegeix el primer sector del disc (el MBR, de 512 bytes). En sistemes moderns amb UEFI, accedeix directament a la partició EFI (ESP) i carrega el bootloader des d'allà, sense passar pel MBR.
3. Bootloader (GRUB2) El bootloader (habitualment GRUB2 a GNU/Linux) és el primer programa de ple dret que s'executa. Mostra el menú d'arrencada (on pots triar el kernel o el sistema operatiu) i carrega el kernel de Linux i el fitxer initramfs a la memòria RAM.
4. Kernel de Linux El kernel es descomprimeix i comença a executar-se. Primer inicialitza el sistema de fitxers temporal initramfs, que conté els controladors mínims necessaris per poder accedir al disc real. Després munta el sistema de fitxers arrel definitiu, carrega els mòduls del kernel necessaris (xarxa, gràfics, so…) i traspassa el control al procés d'init.
5. Sistema d'init (systemd) El kernel llança el procés amb PID 1, que a la majoria de distribucions modernes és systemd. Aquest s'encarrega d'arrencar en paral·lel tots els serveis del sistema: la xarxa, el gestor de so (PipeWire/PulseAudio), el servidor gràfic (Wayland/X11), el Bluetooth, i molts altres daemons.
6. Login manager i sessió Finalment, systemd arrenca el display manager (com GDM, SDDM o LightDM), que mostra la pantalla gràfica de login. Quan introdueixes les credencials, s'inicia la teva sessió d'usuari: es carreguen les variables d'entorn, el gestor de finestres o l'escriptori (GNOME, KDE, etc.) i els programes d'inici de sessió. Des d'aquí ja tens el control total del sistema.

1. Què fa el firmware BIOS/UEFI durant l'arrencada?

El firmware és la peça més invisible del procés d'arrencada, però és on tot comença. Aquí tens el detall complet, amb una anàlisi comparativa visual entre BIOS i UEFI:

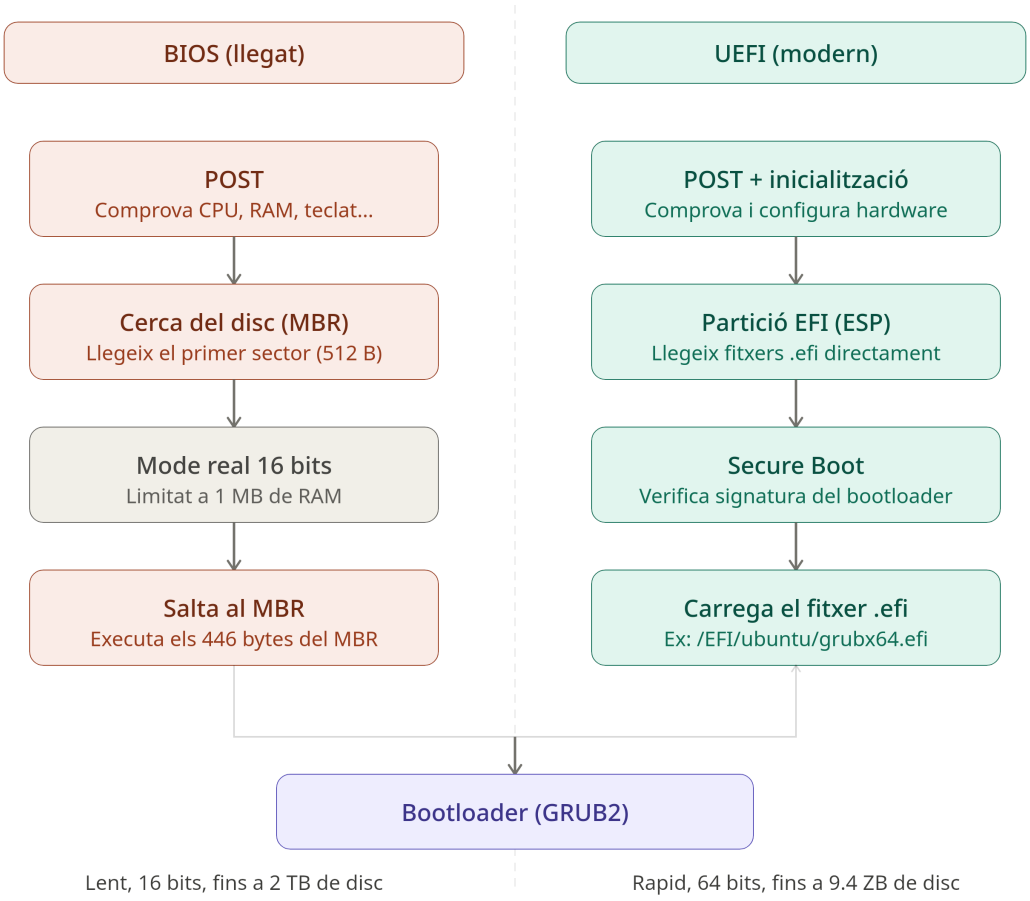

Figura 2: El firmware BIOS/UEFI durant l'arrencada

Quan s'activa el firmware

En el moment que la font d'alimentació estabilitza el voltatge (senyal Power Good), la CPU desperta i busca la primera instrucció a executar. Aquesta instrucció sempre apunta a la mateixa adreça de memòria fixa (a x86, és 0xFFFFFFF0), on hi ha el codi del firmware. És a dir, no és el sistema operatiu qui pren el control primer: és un programa gravat en un xip de la placa base.

El POST (Power-On Self-Test)

El firmware fa una inspecció ràpida del maquinari: comprova que la CPU funciona correctament, detecta i comprova la memòria RAM, inicialitza el pont nord i el pont sud de la placa base, i detecta els dispositius connectats (teclat, ratolí, targetes de xarxa, etc.). Si troba un error crític en aquesta fase, pot emetre bips (codis acústics) o mostrar missatges d'error i aturar el procés.

La gran diferència: com es troba el bootloader

Aquí és on BIOS i UEFI divergeixen radicalment. La BIOS llegat és molt primitiva: llegeix els primers 512 bytes del disc d'arrencada (el MBR, o Master Boot Record). D'aquests 512 bytes, 446 contenen codi executable, 64 bytes contenen la taula de particions, i els darrers 2 bytes són una signatura de validació. Amb només 446 bytes de codi, la BIOS no pot fer gaires coses; normalment aquest codi petit només sap trobar el bootloader real i saltar-hi. L'UEFI, en canvi, entén sistemes de fitxers. Busca una partició especial del disc anomenada EFI System Partition (ESP), formatada en FAT32, i hi llegeix fitxers .efi directament. Això li permet tenir un gestor d'arrencada propi, amb menú gràfic, suport de ratolí, i fins i tot connexió de xarxa abans de carregar cap sistema operatiu.

Secure Boot (exclusiu d'UEFI)

L'UEFI pot verificar criptogràficament que el bootloader que executarà té una signatura digital vàlida, signada per una clau de confiança. Això impedeix que programari maliciós s'instal·li en la cadena d'arrencada sense que te n'adonis. En distribucions com Ubuntu o Fedora, el bootloader ja ve signat. En Arch Linux o Gentoo, cal gestionar les claus manualment si es vol activar.

El lliurament al bootloader

Tant BIOS com UEFI acaben fent el mateix: passar el control al bootloader (habitualment GRUB2). La diferència és el camí: la BIOS hi arriba saltant a una adreça de disc molt limitada, mentre que l'UEFI hi arriba carregant un fitxer .efi com si fos un programa normal. A partir d'aquí, GRUB2 pren el relleu.

2. POST (Power-On Self-Test)

El POST és la primera tasca real que fa el firmware just després de rebre alimentació estable. És essencialment un diagnòstic automàtic del maquinari que es fa cada vegada que encens l'ordinador, abans que res més pugui passar.

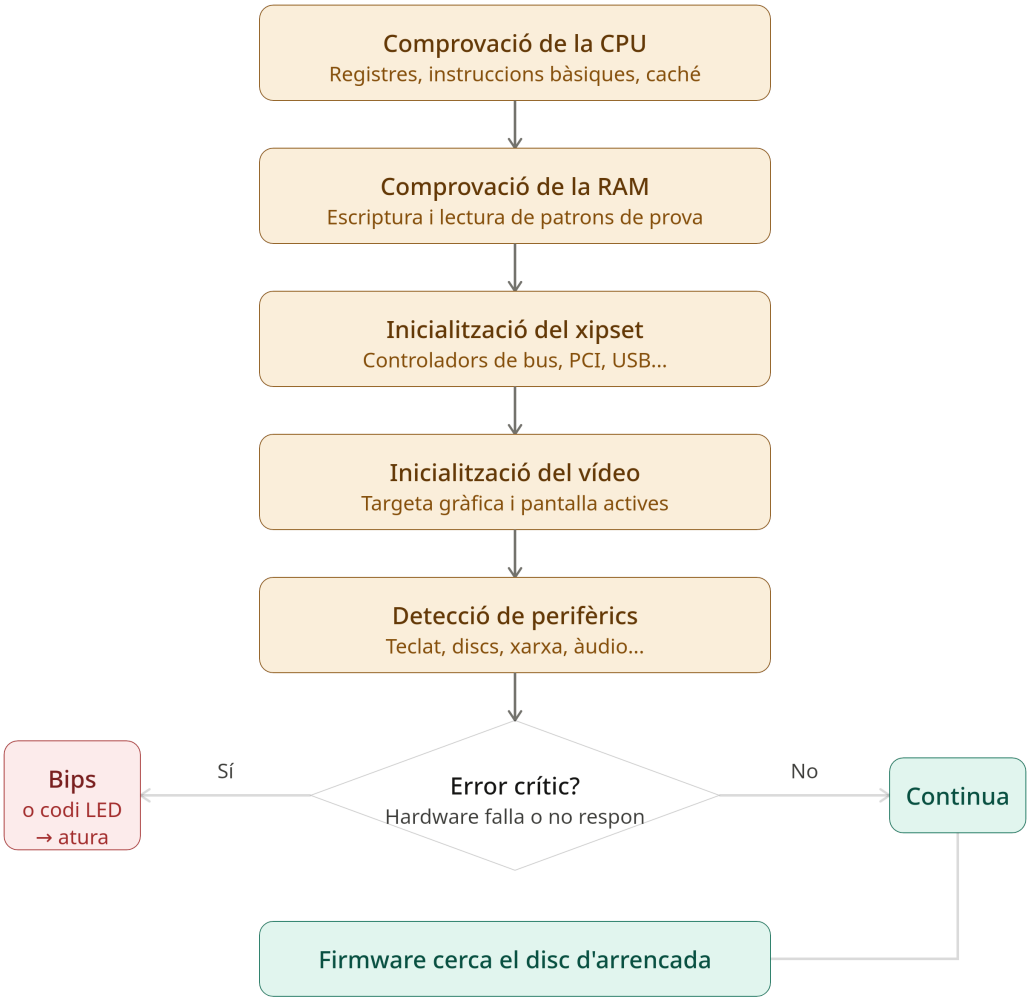

Figura 3: POST

Comprovació de la CPU

És el primer que fa el firmware, perquè sense CPU funcional no es pot fer res més. El processador s'autocomprova: verifica que els seus registres interns es poden llegir i escriure correctament, que les instruccions bàsiques produeixen els resultats esperats, i que la memòria cau (L1 i L2) funciona. Si la CPU és defectuosa, l'ordinador simplement no reacciona en absolut.

Comprovació de la RAM

El firmware escriu patrons de bits coneguts a diverses adreces de memòria i després els llegeix per verificar que coincideixen. D'aquesta manera detecta cel·les de memòria

defectuoses. En ordinadors moderns aquest procés és molt ràpid perquè es comprova una mostra representativa, no cada byte de tota la RAM. Aquí és on veus el recompte de memòria a la pantalla en sistemes antics (el “64MB… 128MB…”).

Inicialització del xipset i els busos

El xipset és el conjunt de circuits que connecta la CPU amb la resta del sistema. El firmware l'activa i configura els busos de comunicació: PCI Express (per a targetes gràfiques i NVMe), USB, SATA, etc. Sense aquesta fase, cap dispositiu perifèric existeix des del punt de vista del sistema.

Inicialització del vídeo

La targeta gràfica s'inicialitza relativament aviat perquè el firmware necessita la pantalla per poder mostrar missatges d'error o el menú de configuració. La BIOS/UEFI carrega un firmware mínim de la mateixa targeta (anomenat VBIOS o GOP) per activar una resolució bàsica. Per això veus el logo del fabricant de la placa base just en encendre l'ordinador.

Detecció de perifèrics

El firmware recorre tots els busos i registra els dispositius que hi troba: discs durs, SSDs, teclat, ratolí, targetes de xarxa, lectors de targetes, etc. Cada dispositiu s'identifica i s'assigna als recursos del sistema (adreces d'E/S, interrupcions IRQ). Aquesta informació s'emmagatzema en estructures de dades que el sistema operatiu consultarà més tard.

Codi d'errors si alguna cosa falla

Si una de les comprovacions detecta un problema crític, el sistema no pot continuar. Com que potser la pantalla encara no funciona en fases molt primerenques, el firmware comunica l'error amb bips (codis acústics), LEDs de diagnòstic a la placa base, o un display de 7 segments. Cada fabricant té la seva codificació: per exemple, tres bips curts en una placa ASUS indica un error de RAM. Si el problema no és crític (per exemple, un dispositiu USB que no respon), el firmware el registra, l'ignora i continua.

Quan acaba el POST

Si tot és correcte, el firmware passa el control a la fase de cerca del dispositiu d'arrencada, que és on entren en joc la BIOS/UEFI per trobar el bootloader. La durada total del POST en un sistema modern és d'entre 1 i 4 segons, tot i que molts sistemes l'amaguen darrere el logo del fabricant per fer una sensació de velocitat.

3. Quina diferència hi ha entre BIOS i UEFI?

La diferència és molt més profunda que simplement “un és més nou que l'altre”. Són dues arquitectures de firmware completament diferents, dissenyades en èpoques i amb limitacions molt distintes.

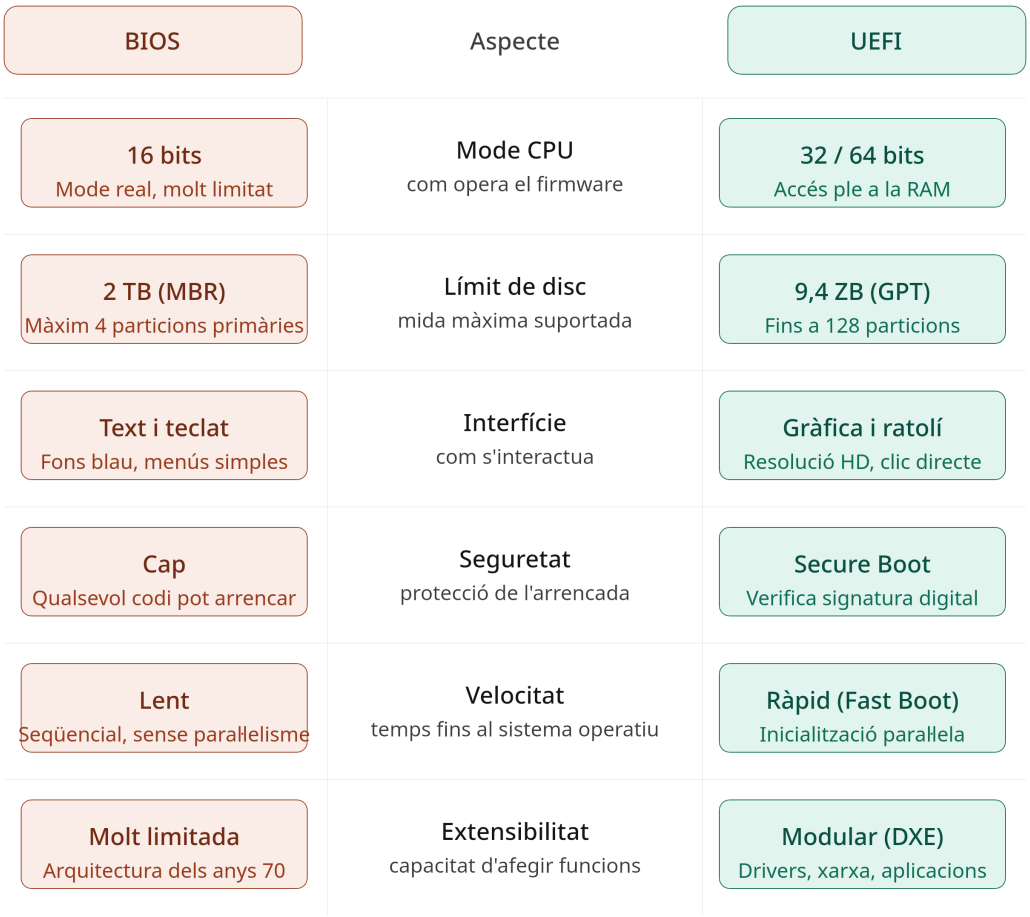

Figura 4: BIOS vs. UEFI

D'on venen i per què existeix la diferència

La BIOS (Basic Input/Output System) va nàixer als anys 70 amb el PC d'IBM i va ser dissenyada per a ordinadors amb processadors Intel 8088 de 16 bits. Durant dècades va ser l'estàndard, però carregava un pes enorme: la compatibilitat enrere. Cada nova versió havia de seguir funcionant exactament igual que l'original. El resultat és un firmware que en el 2020 continuava executant-se en mode real de 16 bits, exactament igual que el 1981. L'UEFI (Unified Extensible Firmware Interface) va nàixer a Intel als anys 90 per als servidors Itanium, que ja eren de 64 bits i no cabien en les limitacions de la BIOS. El 2005 es va crear el fòrum UEFI per estandarditzar-lo, i des del 2011 pràcticament tota la placa base nova incorpora UEFI.

El problema del mode de 16 bits

Quan la BIOS s'executa, la CPU opera en mode real de 16 bits, cosa que limita l'accés a tan sols 1 MB de memòria RAM. Tota la lògica del firmware, la detecció de dispositius i la càrrega del bootloader ha de cabre en aquest espai minúscul. L'UEFI, en canvi, commuta la CPU a mode protegit de 32 o 64 bits des del primer moment, amb accés a tota la RAM del sistema.

MBR vs. GPT: el límit de 2 TB

La BIOS utilitza l'esquema de particionament MBR (Master Boot Record), que descriu les particions amb números de 32 bits. Matemàticament, 2^{32} sectors de 512 bytes = exactament 2 TB. Qualsevol disc més gran és invisible per a la BIOS. L'UEFI utilitza GPT (GUID Partition Table), que usa 64 bits per als sectors i admet discs pràcticament il·limitats en mida. A més, el MBR permet un màxim de 4 particions primàries, mentre que GPT en permet fins a 128.

Secure Boot

Aquesta és potser la diferència més important per a la seguretat. La BIOS executarà qualsevol codi que trobi al MBR, sigui el que sigui: un bootloader legítim, un virus de rootkit o qualsevol cosa. L'UEFI pot verificar criptogràficament que el fitxer .efi que executarà té una signatura digital vàlida. Sense la signatura correcta, no arrencarà. Això fa molt més difícil que programari maliciós s'instal·li a la cadena d'arrencada sense ser detectat.

L'UEFI és un sistema operatiu petit

Potser el canvi conceptual més gran és que l'UEFI no és simplement un programa de configuració: és un entorn d'execució complet. Pot tenir controladors de xarxa, llegir sistemes de fitxers FAT32, executar aplicacions EFI, fins i tot navegar per Internet en alguns models d'alt gamma. Té una arquitectura modular anomenada DXE (Driver eXecution Environment) que permet als fabricants afegir funcionalitats sense tocar el codi base. La BIOS, per comparació, era un bloc monolític i quasi impossible d'estendre.

4. Quin és el rol de l'MBR o GPT en l'arrencada de Linux?

El MBR i el GPT no són programes en si mateixos: són estructures de dades gravades al disc que descriuen com està organitzat i on comença el codi d'arrencada. Sense elles, el firmware no sabria ni per on començar.

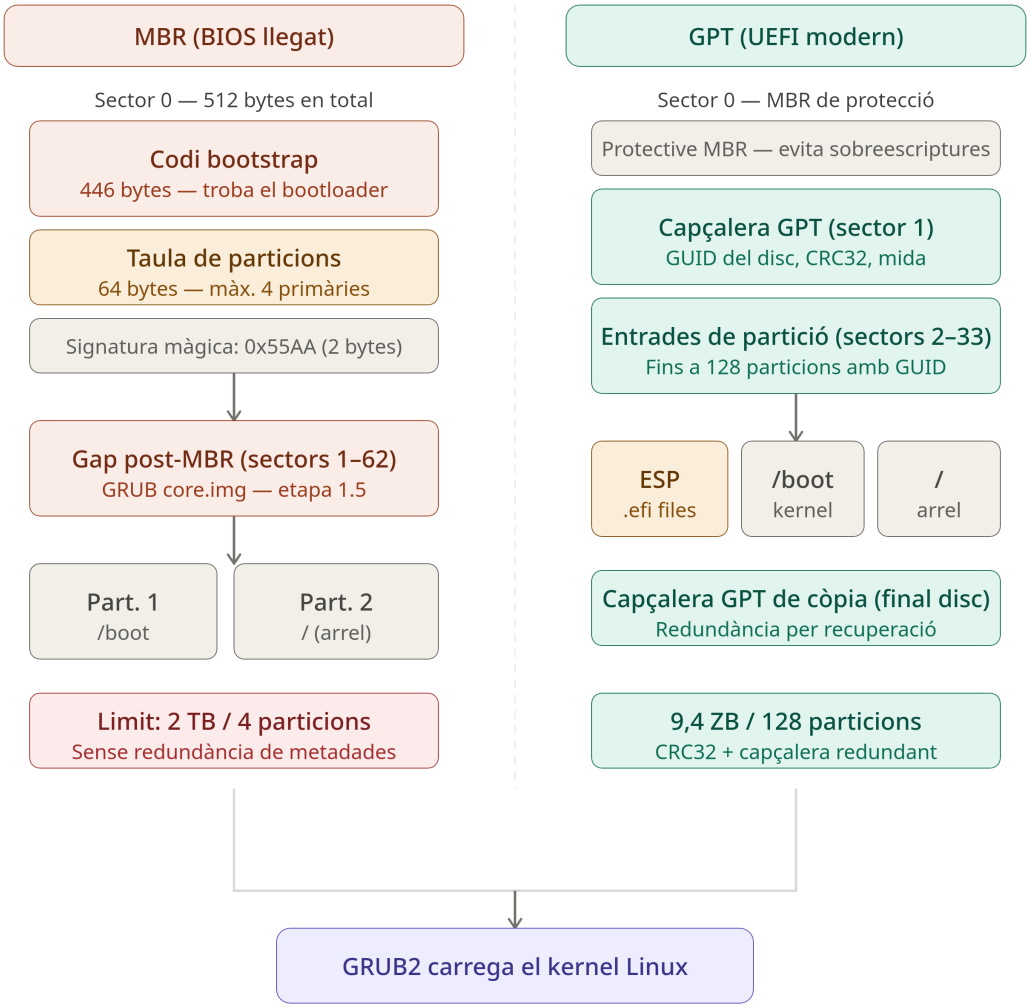

Figura 5: Rol de l'MBR o GPT en l'arrencada

El problema que resolent ambdós esquemes

Quan el firmware acaba el POST i sap quin disc és el d'arrencada, ha de respondre a dues preguntes: on és el codi que ha d'executar primer, i com està dividit el disc en particions. MBR i GPT són les dues respostes possibles a aquestes preguntes, cadascuna amb una arquitectura radicalment diferent.

L'estructura del MBR: 512 bytes que ho fan tot

El MBR ocupa exactament el primer sector del disc (512 bytes) i té tres parts ben diferenciades. Els primers 446 bytes contenen el codi bootstrap: un petit programa en assemblador que la BIOS carrega directament a la memòria i executa. Aquest codi és

massa petit per fer gran cosa; el seu únic objectiu és trobar la partició activa i saltar al bootloader real. Els 64 bytes següents contenen la taula de particions, que descriu fins a 4 particions primàries amb la seva posició i mida (en valors de 32 bits, d'aquí el límit de 2 TB). Els darrers 2 bytes són la signatura màgica 0x55AA, que el firmware verifica per confirmar que el sector és un MBR vàlid i no dades aleatòries. El problema és que GRUB2 no cap en 446 bytes. Per resoldre-ho s'usa el “gap post-MBR”: l'espai entre el primer sector i la primera partició (típicament els sectors 1 al 62) on GRUB instal·la el seu core.img, la part que ja sap llegir sistemes de fitxers i trobar el kernel.

L'estructura del GPT: dissenyada per durar

El GPT és molt més sofisticat. El sector 0 conté un “Protective MBR”, un MBR fals que declara que tot el disc és una única partició de tipus desconegut. Això evita que eines antigues que no entenen GPT sobreescriguin el disc per error. El sector 1 conté la capçalera GPT pròpiament dita: un identificador únic global (GUID) del disc, un checksum CRC32 per detectar corrupció, i metadades sobre la mida i la posició de les entrades de partició. Els sectors 2 al 33 contenen fins a 128 entrades de partició, cadascuna amb el seu propi GUID de tipus (que identifica si és una partició EFI, Linux, swap, etc.) i el seu GUID únic. Cada entrada usa 64 bits per a les adreces, eliminant el límit de 2 TB. L'element més elegant del GPT és la redundància: una còpia idèntica de la capçalera i les entrades de partició es guarda al final del disc. Si la capçalera principal es corromp, les eines de recuperació poden restaurar-la des de la còpia.

La partició ESP i el seu rol clau

En sistemes UEFI amb GPT, la peça central de l'arrencada és l'EFI System Partition (ESP). És una partició formatada en FAT32 que conté els fitxers .efi dels bootloaders instal·lats. Quan instal·les Ubuntu, per exemple, es crea el fitxer /EFI/ubuntu/grubx64.efi dins l'ESP. L'UEFI llegeix directament aquest fitxer i l'executa, sense necessitar cap codi al sector 0. Això permet tenir múltiples sistemes operatius, cadascun amb el seu propi .efi, sense que interfereixin entre si.

Per què Linux pot usar ambdós

El kernel de Linux és agnòstic respecte a l'esquema de particionament: pot arrencar des d'un disc MBR amb BIOS o des d'un disc GPT amb UEFI. GRUB2 s'adapta a ambdós casos. En un sistema BIOS+MBR, GRUB s'instal·la al gap post-MBR. En un sistema UEFI+GPT, GRUB s'instal·la com un fitxer .efi dins l'ESP. El resultat final és el mateix: GRUB2 pren el control i carrega el kernel de Linux.

5. Com funciona GRUB com a bootloader de Linux?

GRUB2 és molt més que un simple menú d'arrencada: és un petit sistema operatiu per si mateix, capaç de llegir sistemes de fitxers, desxifrar discs xifrats i carregar kernels de múltiples sistemes operatius. S'executa en tres etapes successives, cadascuna amb més capacitats que l'anterior.

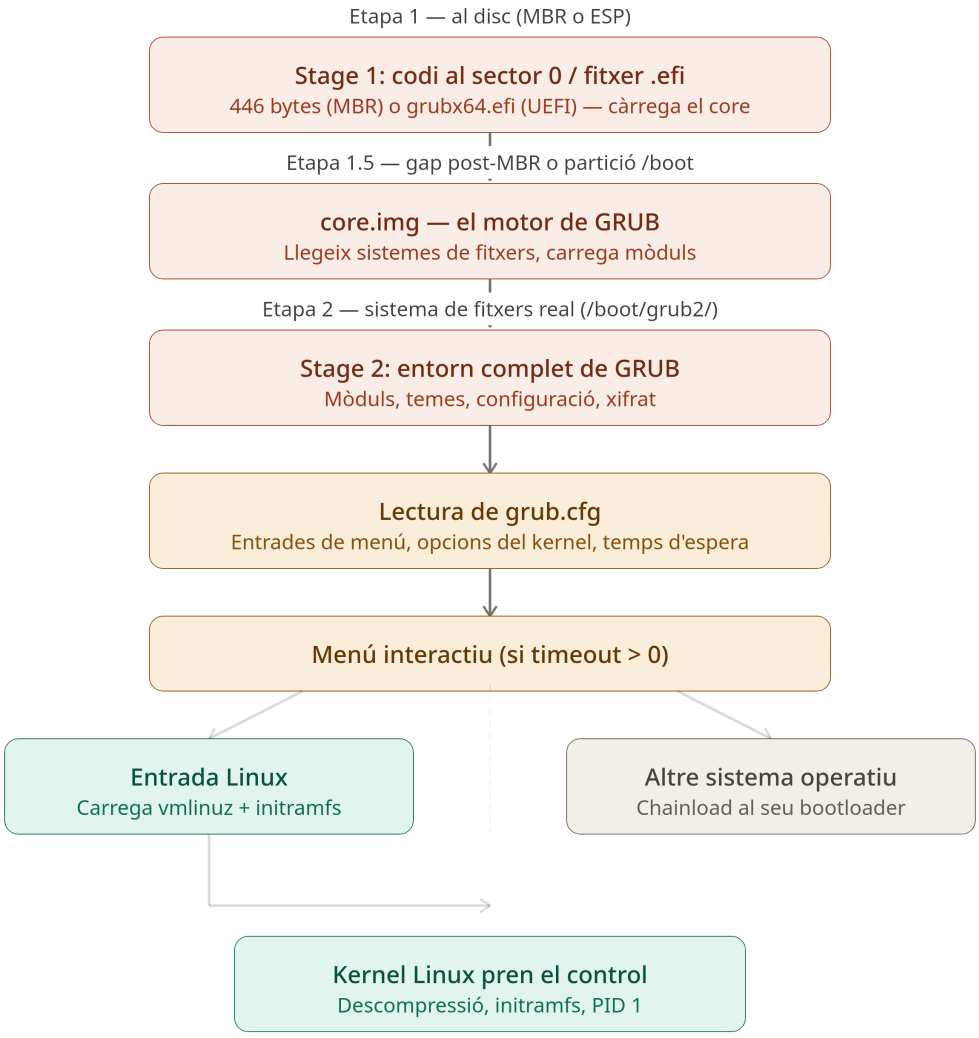

Figura 6: GRUB com a bootloader

El problema que GRUB ha de resoldre

El firmware (BIOS o UEFI) és molt limitat: pot llegir sectors bruts del disc, però no entén sistemes de fitxers com ext4 o Btrfs. El kernel de Linux, en canvi, viu dins un sistema de fitxers. Com es passa d'un món a l'altre? Amb una cadena de tres etapes progressivament més capaces.

Stage 1: el punt d'entrada mínim

En sistemes BIOS+MBR, la Stage 1 de GRUB és el codi dels 446 bytes del MBR. Fa una única cosa: sap on és el core.img al disc (té les adreces de blocs codificades directament) i hi salta. En sistemes UEFI+GPT, la Stage 1 és el fitxer grubx64.efi dins l'ESP, que l'UEFI carrega i executa directament com un programa.

Stage 1.5: el core.img, el motor real

El core.img és on GRUB guanya capacitat real. Conté el codi mínim per llegir el sistema de fitxers on viu /boot/grub2/, ja sigui ext4, XFS, Btrfs, ZFS o fins i tot un volum LUKS xifrat. Un cop el core.img pot llegir el sistema de fitxers, carrega tots els mòduls addicionals que necessita. A partir d'aquí, GRUB és un entorn complet.

Stage 2: l'entorn complet

El directori /boot/grub2/ conté tot l'ecosistema de GRUB: mòduls per a cada sistema de fitxers, suport per a xarxa (PXE boot), suport per a discs xifrats amb LUKS, temes gràfics, i el fitxer de configuració grub.cfg. Aquest fitxer és el cor de tot: defineix les entrades del menú, els paràmetres del kernel, el temps d'espera abans d'arrencar automàticament, i molt més. Normalment, es genera automàticament amb grub-mkconfig i no s'edita a mà.

El menú i la selecció

Si el timeout és més gran de zero, GRUB mostra el menú i espera. Pots triar entre diverses entrades: versions del kernel (útil si una actualització trenca alguna cosa), modes de recuperació, o altres sistemes operatius. Si no toques res, arrencarà l'entrada per defecte. Si el timeout és zero, GRUB arrencarà immediatament sense mostrar res, que és el comportament habitual en sistemes de producció.

La càrrega del kernel: vmlinuz i initramfs

Quan GRUB té l'entrada triada, fa dues càrregues a la RAM. Primer carrega el kernel comprimit, típicament el fitxer vmlinuz-X.Y.Z (la z final significa que està comprimit amb gzip o zstd). Després carrega l'initramfs, un arxiu comprimit que conté un sistema de fitxers mínim temporal. Finalment, GRUB passa el control al kernel, juntament amb una línia de paràmetres: coses com root=/dev/sda2 quiet splash o rd.luks.uuid=… si el disc arrel és xifrat.

El chainloading per a altres sistemes

Si tries Windows o un altre sistema operatiu, GRUB fa chainloading: carrega el bootloader d'aquell sistema (el bootmgr de Windows, per exemple) i li passa el control completament. GRUB no intenta carregar Windows directament; simplement cedeix el testimoni al seu propi gestor d'arrencada. Això és perquè GRUB pot gestionar arrencades múltiples sense necessitar entendre el format intern de cada sistema operatiu.

6. Quins passos segueix el kernel Linux en inicialitzar-se?

És el moment més crític de tota l'arrencada: GRUB ha passat el control, i ara el kernel ha de passar de ser un fitxer comprimit a la RAM a tenir el control absolut del maquinari. Ho fa en una seqüència molt precisa.

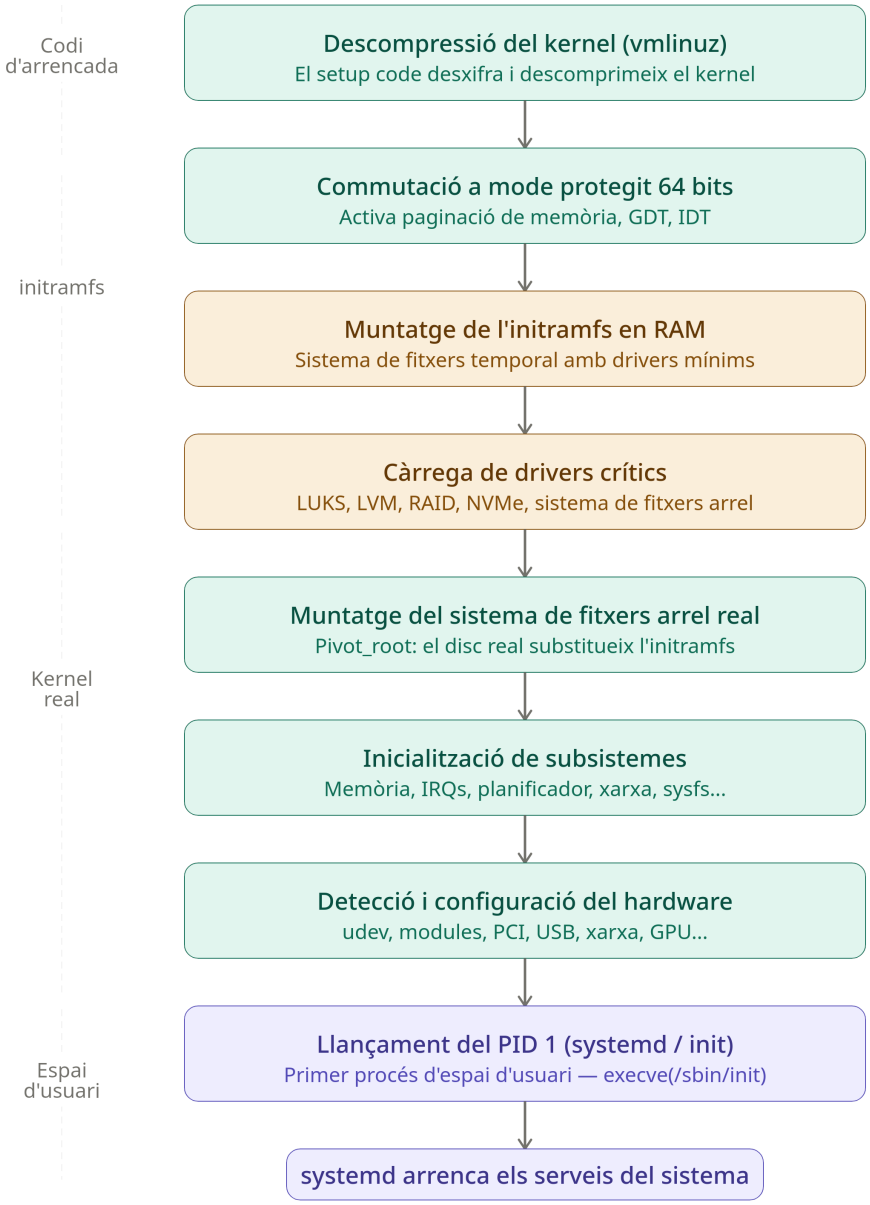

Figura 7: Passos del kernel Linux en inicialitzar-se

Descompressió i primer codi

El fitxer vmlinuz que GRUB carrega a la RAM no és el kernel directament: és el kernel comprimit envoltat per un petit programa en assemblador anomenat setup code. Aquest codi s'executa primer i té dues tasques: negociar amb el firmware alguns paràmetres bàsics

de memòria, i descomprimir el kernel real (amb gzip, lzma o zstd depenent de la distribució) a una adreça de memòria fixa. Un cop descomprimit, el control passa a la funció start_kernel() del codi C del kernel.

Commutació a mode protegit de 64 bits

En aquest punt la CPU encara pot estar en mode real de 16 bits (en sistemes BIOS antics). El kernel la commuta immediatament a mode protegit de 64 bits. Això implica configurar estructures de dades crítiques: la GDT (Global Descriptor Table, que defineix els segments de memòria), la IDT (Interrupt Descriptor Table, que mapeja cada interrupció amb el seu gestor), i activar la paginació de memòria, que és el mecanisme que permet a cada procés tenir el seu propi espai d'adreces virtual independent.

L'initramfs: el pont cap al disc real

Aquí és on entra en joc l'initramfs, que GRUB havia carregat a la RAM just al costat del kernel. El kernel el munta com a sistema de fitxers arrel temporal (/). Dins hi ha un conjunt mínim d'eines i drivers: els necessaris per poder accedir al disc on viu el sistema real. Sense l'initramfs, el kernel estaria encallat: no podria llegir el disc perquè els drivers per fer-ho viuen al disc, i no pot accedir al disc sense els drivers. L'initramfs trenca aquest cercle viciós.

Drivers crítics i casos complexos

L'initramfs és especialment important en configuracions avançades. Si el disc arrel és xifrat amb LUKS, l'initramfs conté el codi per demanar la contrasenya i desxifrar el volum abans de muntar-lo. Si s'usa LVM o RAID per programari, l'initramfs activa aquests volums. Si el sistema de fitxers arrel és Btrfs o ZFS, l'initramfs conté els seus drivers. Tota la complexitat de l'emmagatzematge es resol aquí, en un entorn controlat, abans de passar al sistema real.

El pivot_root: canvi d'arrel

Un cop l'initramfs ha fet la seva feina i el disc real és accessible, el kernel fa una operació anomenada pivot_root: desmunta l'initramfs de la RAM i munta el sistema de fitxers real del disc com a nova arrel /. A partir d'aquest moment, el que veus al sistema de fitxers és el disc real, no el sistema temporal.

Inicialització dels subsistemes del kernel

Amb el sistema de fitxers arrel disponible, el kernel inicialitza la resta dels seus subsistemes interns: el gestor de memòria virtual (amb les seves zones, slabs i caches), el planificador de processos (el CFS, Completely Fair Scheduler), la pila de xarxa, el sistema de fitxers virtual (VFS), el subsistema de blocs, sysfs i procfs (els sistemes de fitxers virtuals a /sys i /proc), i molts més. Cadascun d'aquests es registra i s'inicialitza en un ordre determinat per dependències.

Detecció del maquinari amb udev

El kernel explora els busos (PCI Express, USB, I^2C, etc.) i detecta tots els dispositius connectats. Per a cada dispositiu, busca el mòdul del kernel adequat i el carrega. Aquí és on apareixen les línies de dmesg que veuries si no tinguessis el logo de la distribució tapant-ho. udev rep notificacions de cada dispositiu nou i crea els fitxers corresponents a /dev (com /dev/sda, /dev/nvidia0, etc.).

El llançament del PID 1: el punt de no retorn

L'últim acte del kernel és executar el primer procés d'espai d'usuari amb execve(). Prova diverses ubicacions en ordre: /sbin/init, /etc/init, /bin/init, /bin/sh. En qualsevol distribució

moderna, /sbin/init és un enllaç simbòlic a systemd. Aquest procés rep el PID 1, un número especialíssim: el kernel mai no el pot matar, i quan qualsevol procés orfe queda sense pare, és adoptat pel PID 1. A partir d'aquí, el kernel ha acabat la seva feina d'inicialització i systemd pren el relleu per arrencar tots els serveis del sistema.

7. Quin és el paper de initramfs durant l'arrencada de Linux?

Ja hem tocat l'initramfs en diverses explicacions anteriors, però mereix un enfocament propi perquè és una de les peces més enginyoses de tot el sistema d'arrencada. La seva existència resol un problema filosòficament elegant: com accedeixes al disc quan el codi per accedir-hi és al disc?

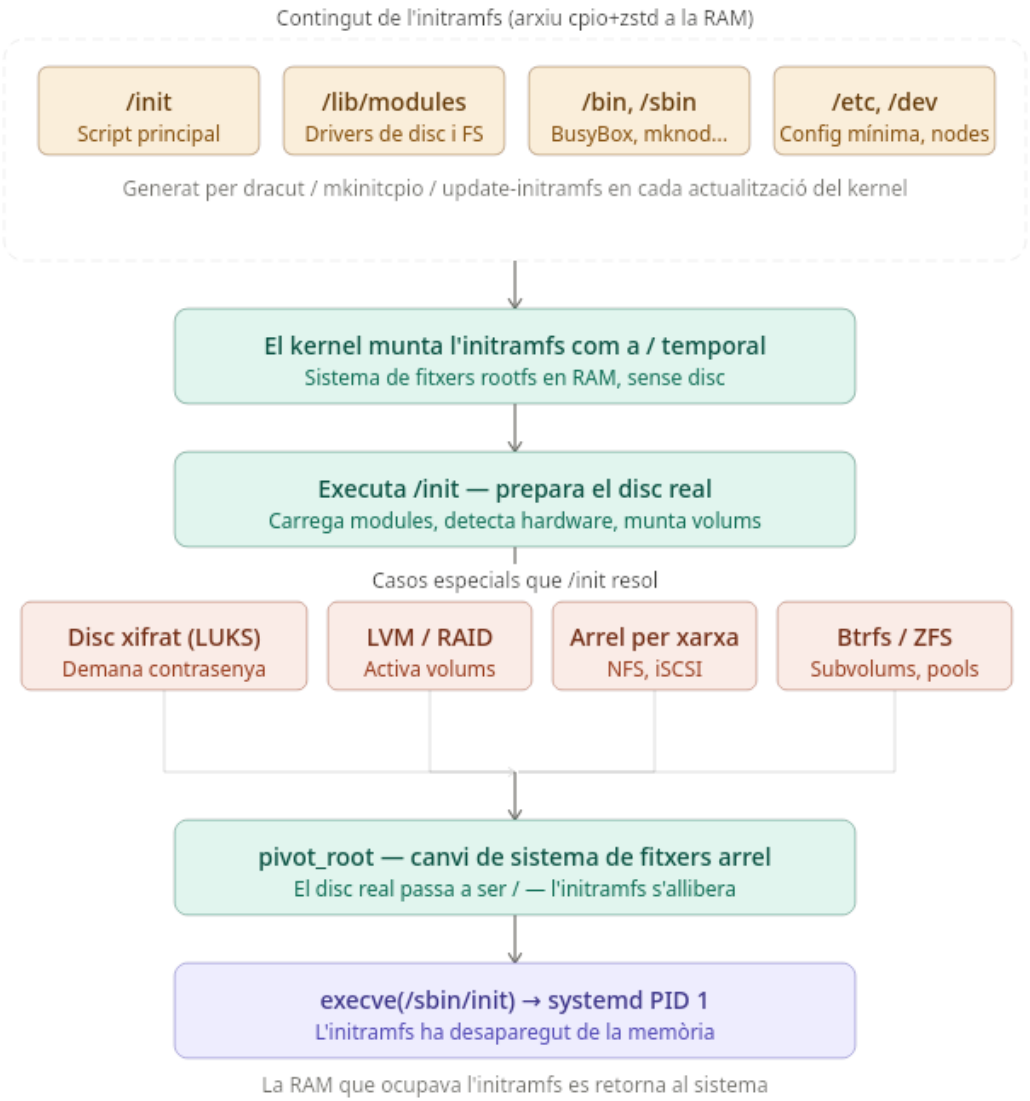

Figura 8: El paper de initramfs durant l'arrencada

El problema del pollastre i l'ou

Per muntar el sistema de fitxers arrel del disc (que pot ser ext4, Btrfs, XFS, ZFS…) el kernel necessita el driver d'aquell sistema de fitxers. Però el driver viu al disc. I per accedir al disc potser calen drivers addicionals: el controlador NVMe, els mòduls LUKS per desxifrar, les eines LVM per activar els volums lògics. Tot és al disc, i el disc no és accessible sense tot això. L'initramfs és la solució: un sistema de fitxers completament autònom que viu a la RAM des del primer instant, amb tot el necessari per resoldre qualsevol configuració d'emmagatzematge.

Què és físicament l'initramfs

No és una partició del disc ni un dispositiu: és un arxiu comprimit (normalment amb zstd o gzip) en format cpio, que el kernel sap desempaquetar directament a la RAM. El trobes a /boot/initramfs-6.x.y.img o /boot/initrd.img-6.x.y depenent de la distribució. Quan GRUB el carrega, simplement el posa a la RAM al costat del kernel, i el kernel ja sap que hi és. Dins, hi trobes una jerarquia de directoris completa, però mínima: /init (el script principal), /bin i /sbin amb eines bàsiques (normalment BusyBox, que empaqueta desenes d'utilitats en un sol binari), /lib/modules amb els mòduls del kernel necessaris, /dev amb nodes de dispositiu essencials, i /etc amb configuració mínima.

L'script /init: el cervell de l'operació

Quan el kernel munta l'initramfs com a sistema de fitxers arrel temporal, executa /init. Aquest script (o binari, en sistemes moderns amb systemd-initrd) és el que fa tota la feina real. Segueix una seqüència aproximadament així: munta els sistemes de fitxers virtuals /proc, /sys i /dev; carrega els mòduls del kernel necessaris per al maquinari detectat; si el disc arrel és xifrat, atura l'execució i demana la contrasenya a l'usuari; si s'usa LVM, executa lvm vgchange per activar els grups de volums; munta el sistema de fitxers arrel real en un directori temporal com /sysroot; i finalment fa el pivot_root.

Els casos on l'initramfs és imprescindible

En un sistema senzill amb un sol disc, un kernel monolític i ext4 sense xifratge, teòricament podries arrencar sense initramfs compilant tots els drivers directament al kernel. Però en la pràctica, les distribucions sempre l'inclouen perquè la majoria de sistemes reals usen alguna d'aquestes configuracions: disc xifrat amb LUKS (molt comú per privacitat), volums lògics LVM (estàndard en moltes instal·lacions empresarials), arrays RAID per programari, sistema de fitxers arrel en Btrfs o ZFS amb subvolums, o fins i tot el disc arrel per xarxa via NFS o iSCSI en servidors sense disc local. Per a tots aquests casos, l'initramfs és l'única solució viable.

El pivot_root: l'operació més elegant

Quan /init ha fet la seva feina i el disc real és accessible i muntat a /sysroot, executa pivot_root. Aquesta crida al sistema intercanvia els dos sistemes de fitxers: el que era /sysroot passa a ser el nou /, i l'antic / (l'initramfs) queda en un directori secundari que immediatament es desmunta i s'allibera. La memòria RAM que ocupava l'initramfs torna al sistema lliure per a altres usos. A partir d'aquest moment, l'initramfs ha deixat d'existir completament.

Com es genera i actualitza

L'initramfs no és estàtic: es regenera automàticament cada vegada que actualitzes el kernel o canvies la configuració del sistema. Les distribucions usen eines específiques per a això: Fedora i RHEL usen dracut, Arch Linux usa mkinitcpio, i Debian/Ubuntu usen

update-initramfs. Totes funcionen de manera similar: examinen el sistema actual, detecten quins drivers i eines cal incloure basant-se en el maquinari i la configuració, i generen el fitxer comprimit que acabarà a /boot. Per això, si afegeixes un disc xifrat o actives LVM, cal regenerar l'initramfs perquè el nou arrencada en tingui suport.

8. Quins controladors i subsistemes inicialitza el kernel Linux?

Aquesta és la fase més llarga i silenciosa de l'arrencada: mentre veus el logo de la distribució, el kernel està inicialitzant desenes de subsistemes en paral·lel. Tot el que dmesg mostra passa aquí.

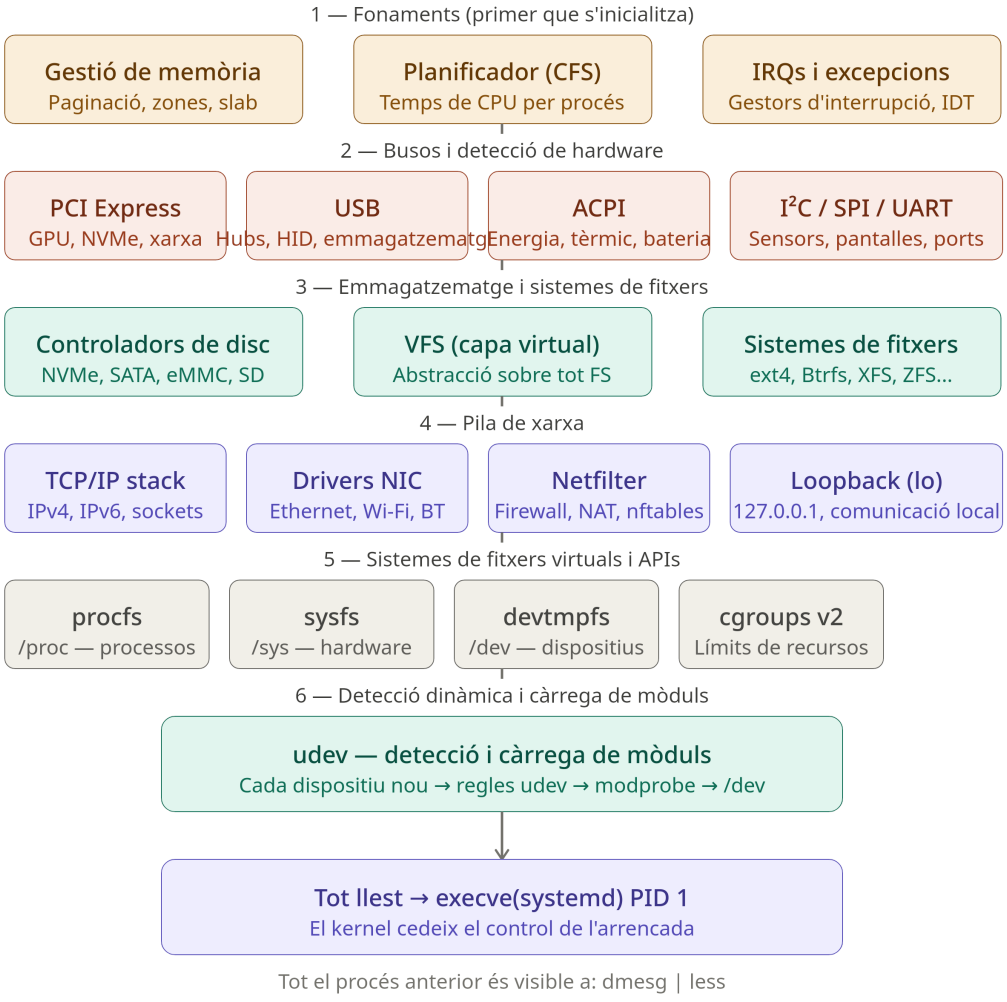

Figura 9: Controladors i subsistemes que inicialitza el kernel Linux

Capa 1: els fonaments que ho fan possible tot

El primer que inicialitza el kernel, abans de qualsevol driver o dispositiu, és la tríada que fa possible l'execució de qualsevol cosa. El gestor de memòria configura la paginació (el mecanisme que dona a cada procés el seu propi espai d'adreces virtual independent), les zones de memòria (DMA, Normal, HighMem), i l'assignador slab per a objectes de kernel petits. El planificador CFS (Completely Fair Scheduler) s'inicialitza per gestionar com es reparteix el temps de CPU entre processos. I el subsistema d'IRQs configura la taula de descriptors d'interrupcions (IDT), de manera que quan una targeta de xarxa o un disc volen dir alguna cosa al sistema, el kernel ho pot sentir.

Capa 2: exploració dels busos

Amb els fonaments al lloc, el kernel explora els busos de comunicació. PCI Express és el més important: hi viu la GPU, els discs NVMe i les targetes de xarxa. El kernel llegeix la configuració de cada dispositiu PCI, n'anota el vendor ID i el device ID, i els emmagatzema per poder carregar-hi el driver adequat. ACPI és la interfície amb el firmware per gestionar l'energia, la temperatura i els estats de suspensió: el kernel interpreta les taules ACPI que el firmware va deixar a la memòria. Els busos interns com I^2C i SPI connecten sensors de temperatura, sensors d'acceleròmetre en portàtils, controladors de retroil·luminació, i molts altres components de placa.

Capa 3: emmagatzematge i sistemes de fitxers

El VFS (Virtual File System) és una de les abstraccions més brillants del kernel: és una capa intermèdia que permet que qualsevol programa llegeixi fitxers sense saber si estan en ext4, Btrfs, NFS o una memòria USB. El kernel registra tots els sistemes de fitxers disponibles en el VFS, i quan muntes un disc, el VFS redirigeix les crides al driver concret. Els controladors de disc (NVMe, AHCI per a SATA, l'stack USB per a memòries) s'inicialitzen aquí i registren els seus dispositius de bloc a /dev.

Capa 4: la pila de xarxa

La pila TCP/IP de Linux és una de les més sofisticades i optimitzades del món. S'inicialitza de manera completament independent del maquinari de xarxa: primer existeix l'abstracció (sockets, protocols, taules d'encaminament), i després els drivers de les targetes concretes s'hi registren. El dispositiu loopback lo (127.0.0.1) es crea automàticament i és el primer dispositiu de xarxa funcional, útil fins i tot sense cap targeta física. Netfilter, el subsistema que dona suport a iptables i nftables, s'inicialitza aquí i permet filtrar paquets des del primer moment.

Capa 5: els sistemes de fitxers virtuals

Tres sistemes de fitxers que no corresponen a cap disc físic i que el kernel crea a la RAM. procfs (muntat a /proc) exposa informació de cada procés en execució com si fossin fitxers: /proc/1/maps et mostra el mapa de memòria del PID 1, /proc/cpuinfo descriu els processadors. sysfs (muntat a /sys) exposa l'arbre de dispositius del kernel: cada driver, cada bus, cada dispositiu té entrades aquí que els programes d'espai d'usuari poden llegir i escriure per configurar-los. devtmpfs (muntat a /dev) crea automàticament els nodes de dispositiu com /dev/sda o /dev/nvidia0 quan els drivers els registren.

Capa 6: udev i la càrrega dinàmica de mòduls

udev és el pont entre el kernel i l'espai d'usuari per a la gestió de dispositius. Quan el kernel detecta un dispositiu nou (un USB que acabes d'endollar, o durant l'arrencada), envia un esdeveniment de uevent a udev. Aquest consulta les seves regles (a /etc/udev/rules.d/ i /lib/udev/rules.d/) i executa les accions corresponents: normalment cridar modprobe per

carregar el driver adequat, crear l'entrada a /dev amb els permisos correctes, i potser executar scripts addicionals. Gràcies a udev, el kernel no necessita portar tots els drivers compilats dins: els carrega dinàmicament en forma de mòduls .ko quan els necessita.

Com veure tot aquest procés

Si vols veure exactament el que ha passat a l'última arrencada, dmesg | less et mostrarà totes les línies en ordre cronològic, amb timestamps si afegeixes -T. Cada línia correspon a un subsistema o driver que s'ha inicialitzat, ha detectat un dispositiu, o ha reportat un error. És literalment el diari complet de tot el que hem descrit aquí.

9. Quina diferència hi ha entre systemd, SysVinit i OpenRC com a sistemes d'init?

Tots tres fan la mateixa feina fonamental ---ser el PID 1 i arrencar els serveis del sistema--- però amb filosofies radicalment diferents sobre com s'ha de fer.

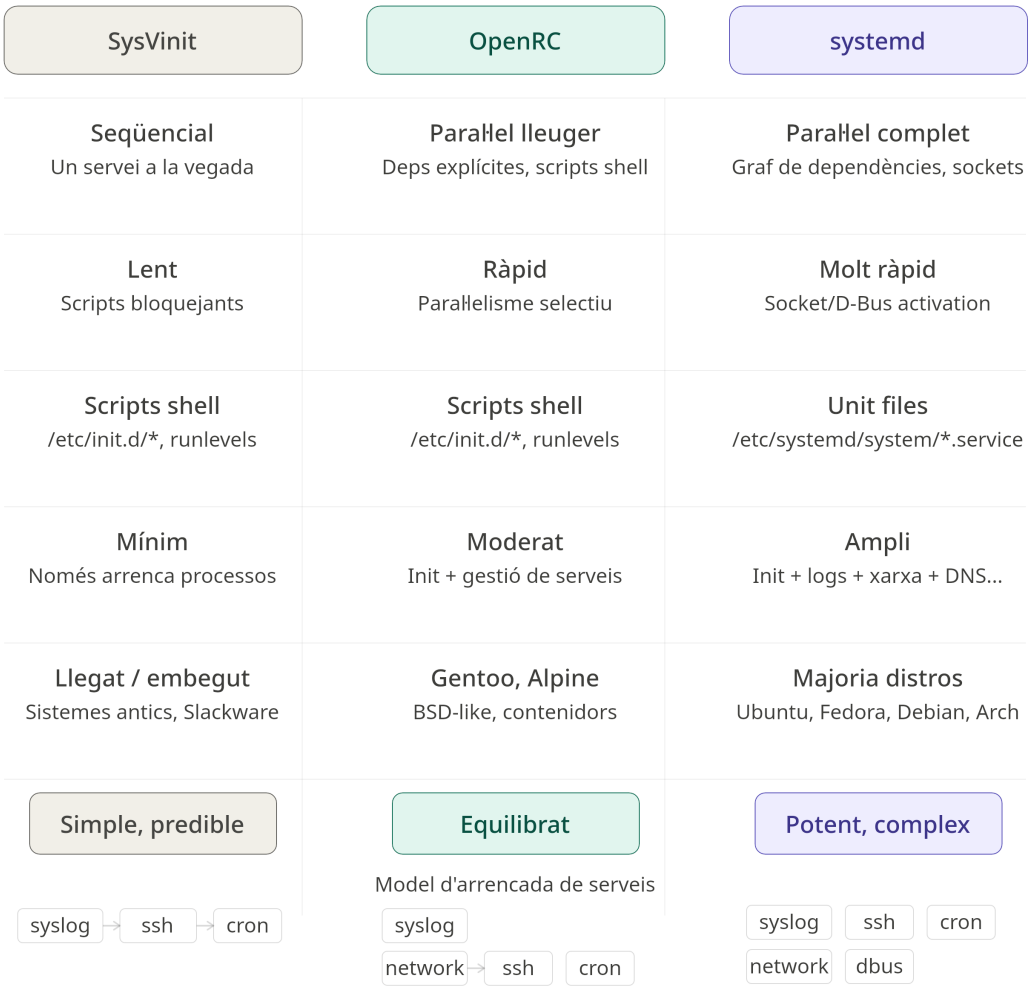

Figura 10: Comparativa de systemd, SysVinit i OpenRC com a sistemes d'init

SysVinit: el model original, senzill fins a l'extrem

SysVinit ve dels anys 80, inspirat en el System V d'AT&T. El seu model és radical en la seva simplicitat: quan el kernel li passa el control, llegeix /etc/inittab per saber a quin runlevel ha d'arrencar (del 0 al 6, on el 3 és multiusuari sense gràfics i el 5 és amb entorn gràfic), i executa en ordre numèric estrictament seqüencial tots els scripts de /etc/rc.d/ corresponents a aquell runlevel. El servei S01syslog s'executa, espera que acabi, llavors S02network, espera, llavors S03ssh, espera. Un darrere l'altre, sense excepcions. La bellesa d'aquest model és la seva predictibilitat absoluta: saps exactament en quin ordre ho arrencarà tot, i

un script d'init és literalment un script de shell que pots llegir i entendre completament. El problema és la velocitat: si un servei tarda 5 segons a inicialitzar-se, el següent espera 5 segons, tot i que no hi tingui cap dependència real. En sistemes moderns amb desenes de serveis, l'arrencada pot trigar minuts.

OpenRC: el compromís pragmàtic

OpenRC, creat per Gentoo i adoptat per Alpine Linux i altres distribucions BSD-like, manté la filosofia dels scripts de shell, però afegeix un graf de dependències explícit. Cada script d'init pot declarar depend() { need network; use logger; }, i OpenRC calcula quins serveis es poden iniciar en paral·lel i quins han d'esperar. Si ssh i cron ambdós necessiten network, però no es necessiten l'un a l'altre, s'inicien simultàniament un cop network és actiu. OpenRC segueix sent scripts de shell purs, cosa que el fa molt transparent i fàcil de depurar. És extremadament lleuger i funciona bé en contenidors i sistemes embeguts on systemd seria excessiu. La seva limitació és que el paral·lelisme és més limitat que el de systemd, i no té mecanismes avançats com l'activació per socket.

systemd: un canvi de paradigma complet

systemd, creat per Lennart Poettering a Red Hat el 2010, va trencar amb tota la tradició anterior. En comptes d'scripts de shell, usa fitxers de configuració declaratius anomenats "unit files" (.service, .socket, .timer, .mount, etc.) que descriuen què és un servei i les seves dependències, però no com s'inicia, ja que systemd ja sap com fer-ho. La innovació clau de systemd és l'activació per socket: en comptes d'esperar que el servei A estigui completament llest per iniciar el servei B que en depèn, systemd crea el socket de comunicació d'A immediatament i inicia B en paral·lel. Si B intenta parlar amb A abans que A estigui llest, les peticions s'encuen al socket. Quan A acaba d'inicialitzar-se, processa la cua. Això permet iniciar pràcticament tots els serveis en paral·lel des del primer instant. A més, systemd ha anat absorbint funcionalitats que abans gestionaven eines separades: journald per als logs del sistema (en binari, amb indexació i filtres potents), networkd per a la configuració de xarxa, resolved per a DNS, logind per a la gestió de sessions d'usuari, timedatectl per al temps, i molts més. Això és exactament el que genera controvèrsia.

La polèmica filosòfica

La crítica principal a systemd és que viola el principi Unix de "fes una cosa i fes-la bé". Els defensors de SysVinit i OpenRC argumenten que un sistema d'init hauria de ser únicament responsable d'arrencar processos i gestionar-ne el cicle de vida, no de gestionar logs, xarxa, DNS i sessions. Quan journald falla, perds els logs del sistema. Quan networkd falla, perds la xarxa. Tot forma part del mateix monòlit. Els defensors de systemd responen que la integració no és un defecte sinó un avantatge: quan journald i networkd formen part del mateix sistema, poden coordinar-se de manera que eines separades no podrien. La depuració és molt més senzilla perquè journalctl -u nginx.service et mostra tots els logs d'nginx amb context del sistema, i systemctl status et mostra en temps real si un servei funciona, quants recursos consumeix i les darreres línies del log. En la pràctica, la discussió s'ha resolt per adopció: Ubuntu, Fedora, Debian, Arch Linux, openSUSE i pràcticament totes les distribucions principals usen systemd. OpenRC manté la seva presència a Gentoo, Alpine i Devuan (un fork de Debian sense systemd). SysVinit pur gairebé ha desaparegut dels sistemes de propòsit general, tot i que sobreviu en sistemes embeguts i de recursos molt limitats.

10. Quins serveis arrenquen típicament durant el boot de Linux?

Quan systemd pren el control com a PID 1, el seu primer treball és construir un graf de dependències entre centenars de serveis potencials i iniciar-los en l'ordre correcte, tant en paral·lel com sigui possible. Aquí tens una visió completa de quins serveis s'inicien i per què.

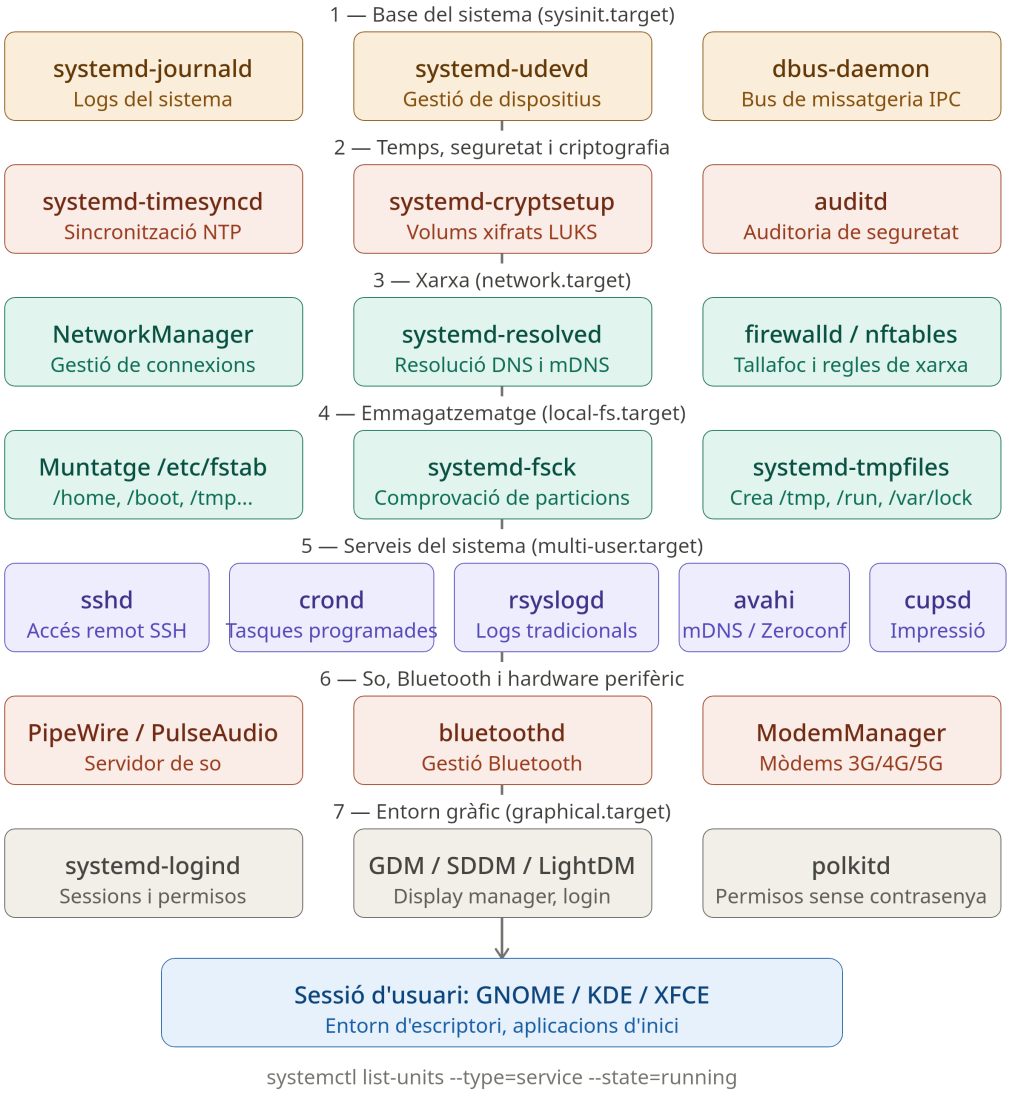

Figura 11: Serveis que arrenquen durant el boot de Linux

Capa 1: la base sense la qual res funciona

El primer que fa systemd és inicialitzar tres serveis fonamentals que la resta necessita. systemd-journald arrenca gairebé immediatament perquè tots els serveis posteriors necessiten poder escriure logs. En binari i indexat, recull tot el que passa al sistema des del primer instant. systemd-udevd és el dimoni que escolta esdeveniments del kernel quan detecta maquinari i carrega els drivers corresponents: sense ell, els dispositius existirien al kernel, però cap programa d'espai d'usuari els podria usar. dbus-daemon és el bus de

missatgeria que permet que els processos es comuniquin entre si sense saber res l'un de l'altre: NetworkManager parla amb el sistema gràfic a través de D-Bus, l'escriptori parla amb PipeWire, PolicyKit parla amb tot. És la columna vertebral de la comunicació entre processos a Linux modern.

Capa 2: temps, seguretat i criptografia

systemd-timesyncd sincronitza el rellotge del sistema amb servidors NTP a la xarxa. Això és més important del que sembla: certificats SSL, logs coherents, Kerberos i molts protocols depenen de tenir l'hora correcta. Si el disc arrel és xifrat amb LUKS, systemd-cryptsetup gestiona el desxifrat (o bé ho ha fet l'initramfs en fases anteriors). auditd registra esdeveniments de seguretat del kernel com intents d'accés, canvis de permisos i execucions de programes: imprescindible en entorns empresarials i servidors.

Capa 3: la xarxa

NetworkManager és el servei que gestiona les connexions de xarxa en la majoria de distribucions d'escriptori. Configura les interfícies, gestiona Wi-Fi, VPN, i notifica els canvis al sistema gràfic. En servidors s'usa més sovint systemd-networkd, més lleuger i configurable via fitxers. systemd-resolved proporciona resolució DNS local i admet DNS sobre TLS i mDNS (per descobrir dispositius a la xarxa local sense servidor DNS). El tallafoc, sigui firewalld o regles nftables directes, s'activa aquí per protegir el sistema abans que cap servei extern estigui disponible.

Capa 4: muntatge i verificació de particions

systemd llegeix /etc/fstab i genera automàticament units de muntatge per a cada entrada. Abans de muntar, systemd-fsck verifica la integritat del sistema de fitxers si ha passat massa temps des de l'última comprovació o si l'última desmuntada no va ser neta. systemd-tmpfiles crea i neteja directoris temporals: tmp.mount muntant /tmp com a tmpfs en RAM, i assegurant que /run, /var/lock i /var/run existeixen amb els permisos correctes.

Capa 5: els serveis del sistema pròpiament dits

Aquí és on viu la major part del que reconeixeries com a “serveis”. sshd obre el port 22 i espera connexions remotes. crond (o els systemd timers moderns) gestiona les tasques programades de /etc/crontab i /etc/cron.d/. rsyslogd recull logs en format tradicional de text per compatibilitat amb eines antigues, en paral·lel amb journald. avahi-daemon implementa mDNS (el protocol que fa que el teu ordinador aparegui com nom-ordinador.local a la xarxa local sense configuració). cupsd gestiona la impressió.

Capa 6: so i maquinari perifèric

PipeWire (el substitut modern de PulseAudio) és el servidor de so que intermedia entre les aplicacions i el maquinari d'àudio. Una aplicació no parla directament amb la targeta de so: demana a PipeWire que reprodueixi àudio, i PipeWire gestiona la barreja, els efectes i el routing. bluetoothd inicialitza l'adaptador Bluetooth i gestiona emparellaments i connexions. ModemManager gestiona mòdems de dades mòbils, necessari en portàtils amb SIM integrada.

Capa 7: l'entorn gràfic

systemd-logind gestiona les sessions d'usuari: controla qui té accés als dispositius físics (pantalla, àudio, teclat) i quan. El display manager (GDM a GNOME, SDDM a KDE, LightDM en d'altres) mostra la pantalla d'inici de sessió i espera les credencials. polkitd gestiona les autoritzacions elevades: és el que permet que un usuari normal pugui muntar un USB o

instal·lar paquets sense ser root directament, mostrant un diàleg de confirmació en comptes d'un prompt de contrasenya.

Com veure exactament quins serveis tens actius

L'ordre systemctl list-units --type=service --state=running et mostra tots els serveis en execució en el teu sistema en aquest moment. systemd-analyze blame et mostra quant de temps ha trigat cada servei a inicialitzar-se, ordenat de més lent a més ràpid. I systemd-analyze critical-chain et mostra el camí crític de dependències que ha determinat el temps total d'arrencada del teu sistema.

11. Com funciona el login manager (display manager) a GNU/Linux?

El display manager és l'última peça de l'arrencada abans que l'usuari tingui el control: és el programa que mostra la pantalla d'inici de sessió, verifica les credencials, i construeix la sessió gràfica des de zero.

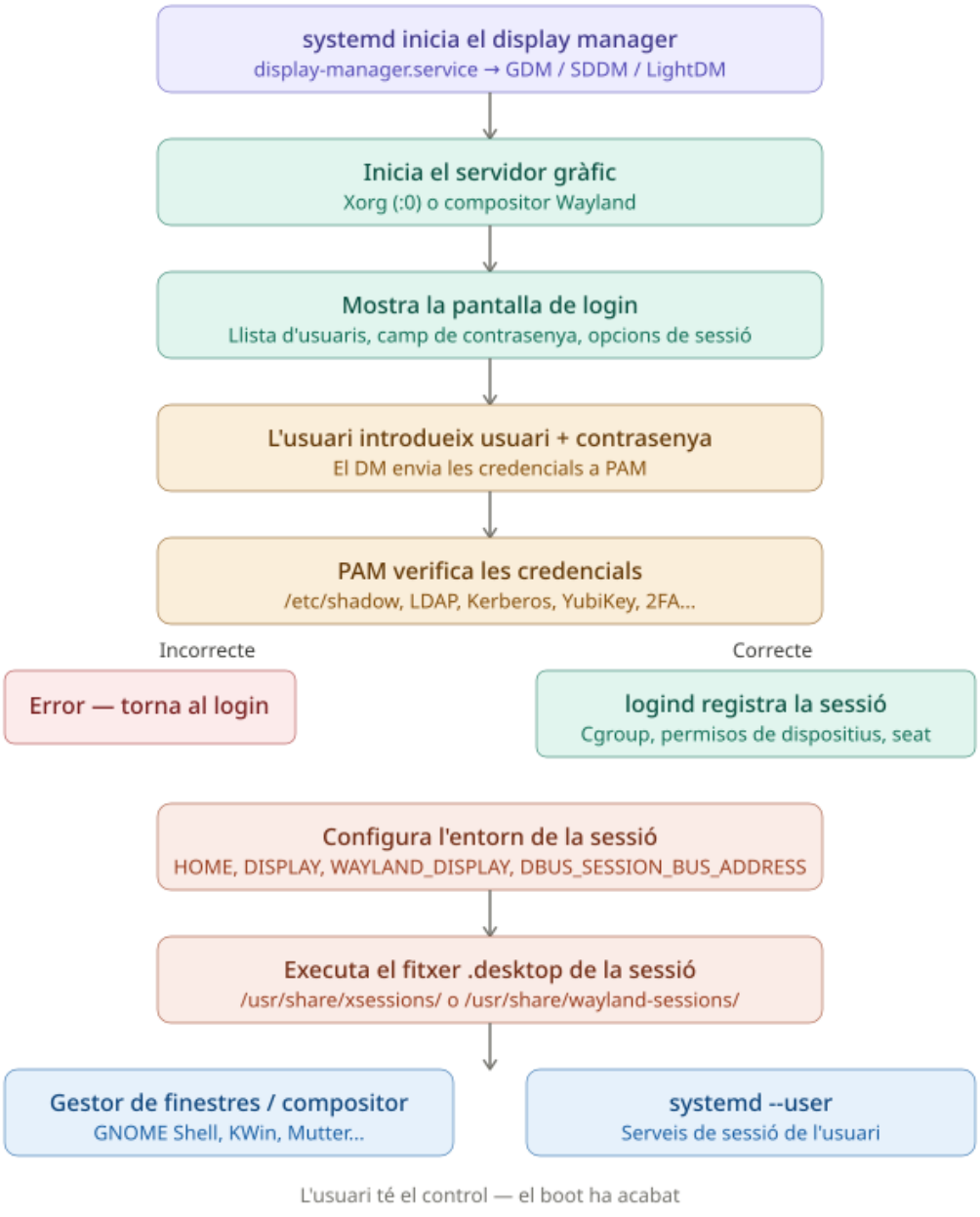

Figura 12: Funcionament del login manager (display manager)

Qui és qui: GDM, SDDM i LightDM

Tots tres fan la mateixa feina, però estan lligats a ecosistemes diferents. GDM (GNOME

Display Manager) és el que usa Ubuntu, Fedora i qualsevol distribució que porti GNOME per defecte: és un programa complet basat en GTK que executa la pantalla d'inici de sessió com una sessió de GNOME Shell reduïda. SDDM (Simple Desktop Display Manager) és el que usa KDE Plasma: està escrit en QML i C++, i és molt personalitzable amb temes. LightDM és el més lleuger i agnòstic: el fan servir XFCE, LXDE i moltes distribucions minimalistes. Tots tres es configuren com a display-manager.service a systemd, i /etc/systemd/system/display-manager.service és un enllaç simbòlic que apunta al que has triat.

El servidor gràfic: Xorg o Wayland

El display manager ha d'iniciar el servidor gràfic abans de poder mostrar res. En sistemes amb Xorg, el DM inicia un procés /usr/bin/X :0 que pren control del framebuffer i gestiona l'entrada de teclat i ratolí. En sistemes Wayland (el present a la majoria de distribucions modernes), el display manager en si actua de compositor o inicia un compositor separat que fa les funcions del servidor gràfic i el gestor de finestres alhora.

PAM: el veritable verificador de credencials

Quan introdueixes la contrasenya, el display manager no la compara ell mateix: la passa a PAM (Pluggable Authentication Modules), un sistema modular que decideix com verificar la identitat. La configuració de PAM per al display manager viu a /etc/pam.d/gdm-password o /etc/pam.d/sddm. PAM consulta /etc/shadow per a comptes locals, però si el sistema està unit a un domini Active Directory consultarà LDAP o Kerberos, si tens una YubiKey configurada la verificarà, i si tens 2FA activat demanarà el segon factor. Aquesta abstracció és el que fa que canviar el mètode d'autenticació d'un sistema Linux sigui tan senzill: no cal tocar el display manager, només reconfigurar PAM.

logind i la sessió

Un cop PAM confirma les credencials, systemd-logind registra la nova sessió. Això no és un tràmit: logind assigna la sessió a un “seat” (el conjunt de dispositius físics: pantalla, teclat, ratolí), crea un cgroup dedicat per a tots els processos d'aquella sessió, i atorga permisos d'accés als dispositius pertinents. Gràcies a logind, el teu usuari pot accedir a /dev/dri/card0 (la GPU), al dispositiu d'àudio, i als dispositius d'entrada sense ser root, perquè logind els cedeix temporalment mentre dura la sessió.

Les variables d'entorn i el fitxer de sessió

Abans de llançar l'escriptori, el DM configura les variables d'entorn que tots els programes de la sessió heretaran: HOME apunta al directori de l'usuari, DISPLAY=:0 o WAYLAND_DISPLAY=wayland-0 indica on és el servidor gràfic, DBUS_SESSION_BUS_ADDRESS indica on és el bus D-Bus de sessió (diferent del bus de sistema), i XDG_SESSION_TYPE indica si és X11 o Wayland. Llavors llegeix el fitxer .desktop de la sessió triada a /usr/share/xsessions/ (per a X11) o /usr/share/wayland-sessions/ (per a Wayland), que li diu quin binari ha d'executar. Per a GNOME seria gnome-session, per a KDE seria startplasma-wayland.

systemd --user: la instància de sessió

Un dels canvis més importants dels últims anys és que systemd ara gestiona també els serveis de sessió de l'usuari, completament separats dels serveis del sistema. Quan la sessió comença, s'inicia una instància de systemd per a l'usuari (systemd --user) que gestiona serveis com pulseaudio.service o pipewire.service de l'usuari, l'agent SSH, el dimoni de sincronització de fitxers, o qualsevol cosa que hagis configurat a ~/.config/systemd/user/. Amb systemctl --user status pots veure i gestionar tots aquests serveis exactament igual

que els serveis del sistema, però sense necessitar privilegis de root. Amb tot això actiu, el boot ha acabat completament: tens el control, i el sistema és teu.

Versions d'aquest document

- HTML - arrencada.html
- PDF - arrencada.pdf
- ODT - arrencada.odt
- MD - arrencada.md