Chapitre 2

Système GNU/Linux

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

2021-2022

KERNEL LINUX

Historique

Composants

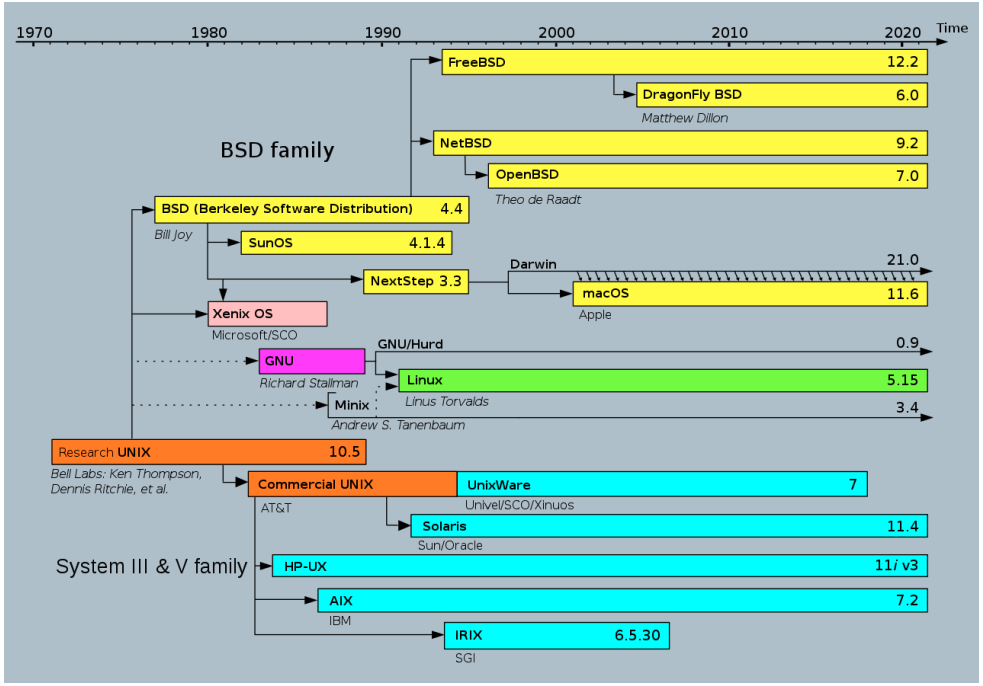
Filesystem

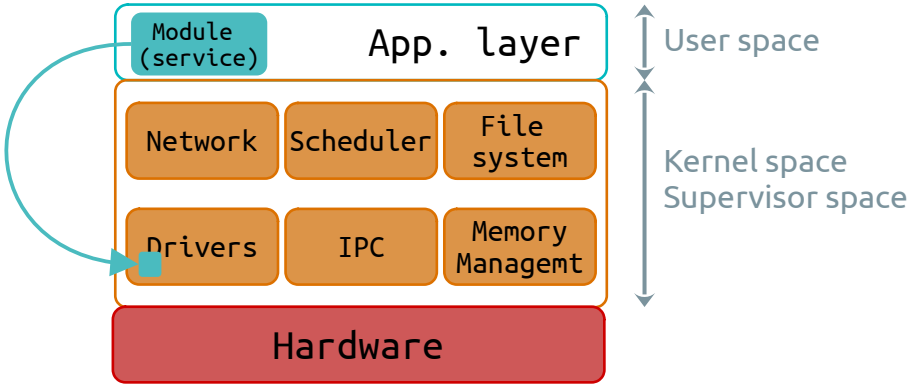
ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Le kernel Linux est un noyau monolithique modulaire.

Il est développé sous licence GPLv2 en suivant un modèle collaboratif décentralisé via internet. Les sources sont librement accessibles sur www.kernel.org .



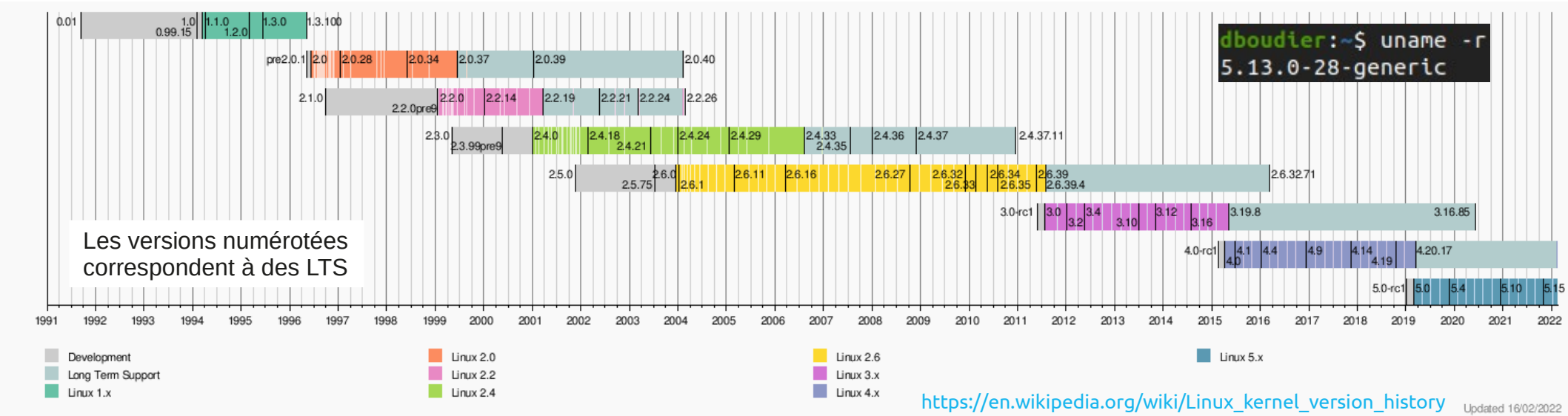


https://en.wikipedia.org/wiki/Linux#/media/File:Unix_timeline.en.svg

Le développement Linux a été lancé en 1991 par Linus Torvalds (8 ans après GNU).

En 2020, le projet comptait 27,8 millions de lignes de code. En 2020 on dénombrait en tout 887.925 commits avec près de 21.000 contributeurs au total.

Aujourd'hui (2022), Torvalds reste le *maintainer* des versions majeures (*-rc* et *mainline*).



Pour faciliter son développement, le projet Linux s'appuie dès 2002 sur Bitkeeper, un logiciel de gestion de version. Mais ce logiciel propriétaire devint payant trois ans après, ce qui contraint les contributeurs à lâcher le projet Linux.

Constatant le manque de solutions libres, Torvalds lance alors en 2005 le projet Git : l'objectif principal est de répondre aux besoins du développement du noyau Linux.

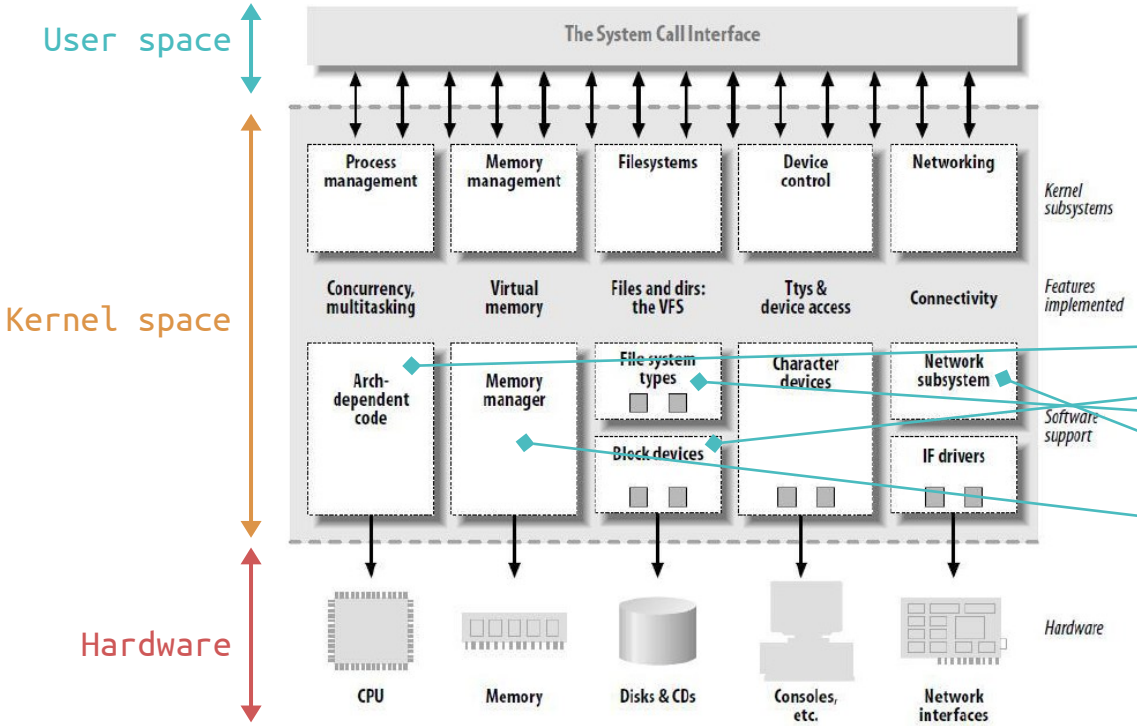
En deux semaines, Git fait accomplit ses premiers *merges*.
En deux mois, Git est finalisé et héberge le kernel Linux 2.6.12.

D'après Torvalds :

```
"I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'git'."
```

Jouant son rôle de kernel, Linux propose des services bas niveaux :

Ordonnanceur, gestion des systèmes de fichiers, pilotes de périphériques (LDD), gestion/protection mémoire, ...



Mode	Name
-rw-r--r--	.clang-format
-rw-r--r--	.cocciconfig
-rw-r--r--	.get_maintainer.ignore
-rw-r--r--	.gitattributes
-rw-r--r--	.gitignore
-rw-r--r--	.mailmap
-rw-r--r--	COPYING
-rw-r--r--	CREDITS
d---------	Documentation
-rw-r--r--	Kbuild
-rw-r--r--	Kconfig
d---------	LICENSES
-rw-r--r--	MAINTAINERS
-rw-r--r--	Makefile
-rw-r--r--	README
d---------	arch
d---------	block
d---------	certs
d---------	crypto
d---------	drivers
d---------	fs
d---------	include
d---------	init
d---------	ipc
d---------	kernel
d---------	lib
d---------	mm
d---------	net
d---------	samples
d---------	scripts
d---------	security
d---------	sound
d---------	tools
d---------	usr
d---------	virt

Sources du noyau

Les fichiers d'en-tête du kernel Linux sont généralement accessibles depuis une machine host, dans le répertoire `/usr/src/linux-headers-<version_no>/`. Ceci permet de compiler du code applicatif pouvant communiquer avec les interfaces logicielles du noyau (appels système)

(C'est Ubuntu qui récupère les sources du kernel et les place dans ce dossier)

```
dboudier:/usr/src$ ls
linux-headers-5.11.0-46-generic  linux-headers-5.8.0-44-generic   linux-hwe-5.8-headers-5.8.0-44
linux-headers-5.13.0-27-generic  linux-headers-5.8.0-63-generic   linux-hwe-5.8-headers-5.8.0-63
linux-headers-5.13.0-28-generic  linux-hwe-5.11-headers-5.11.0-46 tty0tty-1.3
linux-headers-5.4.0-59           linux-hwe-5.13-headers-5.13.0-27 virtualbox-6.1.26
linux-headers-5.4.0-59-generic   linux-hwe-5.13-headers-5.13.0-28
linux-headers-5.8.0-43-generic   linux-hwe-5.8-headers-5.8.0-43
dboudier:/usr/src$
dboudier:/usr/src$
dboudier:/usr/src$ cd linux-headers-5.13.0-28-generic/
dboudier:/usr/src/linux-headers-5.13.0-28-generic$ ls
arch    crypto          fs       ipc      kernel    mm              samples   sound   usr
block   Documentation   include  Kbuild   lib       Module.symvers  scripts   tools   virt
certs   drivers         init     Kconfig  Makefile  net             security  ubuntu
```

`linux-headers-<version_no>-generic` = version stable du kernel

`linux-hwe-<version_no>` = Hardware Enablement = version kernel spécifique au matériel utilisée par Ubuntu

Observons succintement les différents répertoires du kernel (www.kernel.org).

arch/ Architecture. Parties spécifiques aux architectures CPU et plateformes supportées ainsi que certains services propres aux coeurs (core DMA, cache, timer, vecteur d'interruptions, board support, device tree, …)

Il s'agit du second plus gros répertoire de l'arborescence.

```
dboudier:/usr/src/linux-headers-5.13.0-28-generic/arch$ ls
alpha  arm    csky   hexagon  Kconfig  microblaze  nds32  openrisc  powerpc  s390  sparc  x86
arc    arm64  h8300  ia64     m68k     mips        nios2  parisc    riscv    sh    um     xtensa
```

Le répertoire `arch/<arch>/boot/` contient les premiers fichiers (en assembleur) lancés au démarrage du kernel

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

`Documentation/` Documentation du kernel et sous-services proposés (drivers, ...).

`drivers/` Pilotes de périphériques, (GPIO, I²C, CAN, USB, … multiplateformes).

Rappelons que Linux reste un kernel monolithique, tout service matériel supporté par la mainline doit pouvoir être ajouté à la compilation du noyau.

Il s'agit du répertoire le plus volumineux de l'arborescence.

`fs/` *File system*.

Systèmes de fichiers réels et virtuels supportés (ext4, FAT, NTFS, …).

`include/` Principaux fichiers d'en-tête pour compilation du noyau.

`init/` Fichiers d'amorçage et d'initialisation système (`main.c`, `initramfs.c`, …) indépendants de l'architecture.

`ipc/` *Inter-Process Communication*.

Mécanismes de communication inter-processus (mémoire partagée, message queue, sémaphores, …).

`kernel/` Cœur du noyau (scheduler, signaux UNIX, …).

`lib/` *Libraries*.

Petite bibliothèque C interne au kernel (algorithmes de décompression, manipulation de string "`strcmp`, `strlen`, …", …).

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

`mm/` — *Memory management*.

Mécanismes de gestion et protection mémoire (segmentation, pagination, swap, …).

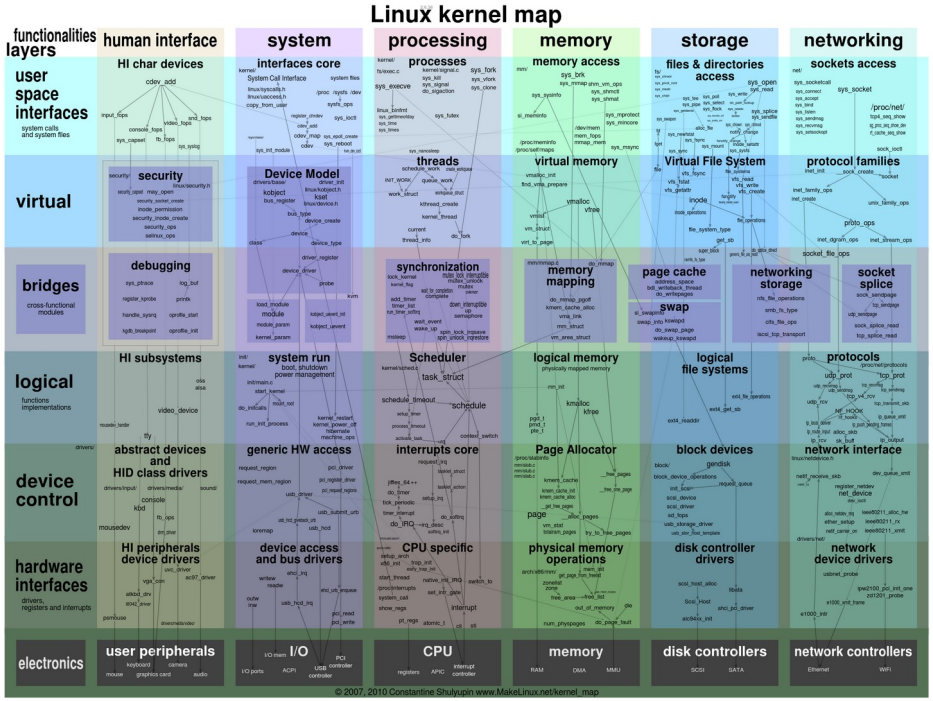
`net/` — *Network*.

Interfaces non-architecture-dépendantes et protocoles réseaux (Ethernet, IPv4, IPv6, CAN, …).

`scripts/` — Code sources des outils de configuration et de compilation du kernel.

`tools/` — Utilitaires de prototypage et d'analyse du kernel et de l'architecture cible (profilage, trace, consommation, échauffement thermique, …).

Afin d'assurer un bonne interopérabilité, maintenabilité et portabilité, le kernel Linux est découpé en couches proposant plusieurs niveaux d'abstraction, notamment au regard du matériel. Observons le *kernel map* du noyau :



https://makelinux.github.io/kernel/map/

GNU

Historique

Composants

Filesystem

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Naissance de GNU et du libre

GNU est un projet de système d'exploitation lancé en 1983 par Stallman, avec pour objectif principal de fournir un OS libre et collaboratif. GNU repose sur une implémentation libre de l'OS UNIX (1969), ce dernier étant à la fois populaire et modulaire (donc facile à réimplémenter morceau par morceau).

Notons que l'acronyme GNU signifie "GNU's Not UNIX".

En quelques années, GNU fournissait un certain nombre de services (application ou librairie) :

- GCC (Initialement *GNU C Compiler*, maintenant *GNU Compiler Collection*)
- Emacs (éditeur de texte), un shell, des bibliothèques
- gdb, make, coreutils, binutils, libc, bash, grub, nano, grep, gimp, etc

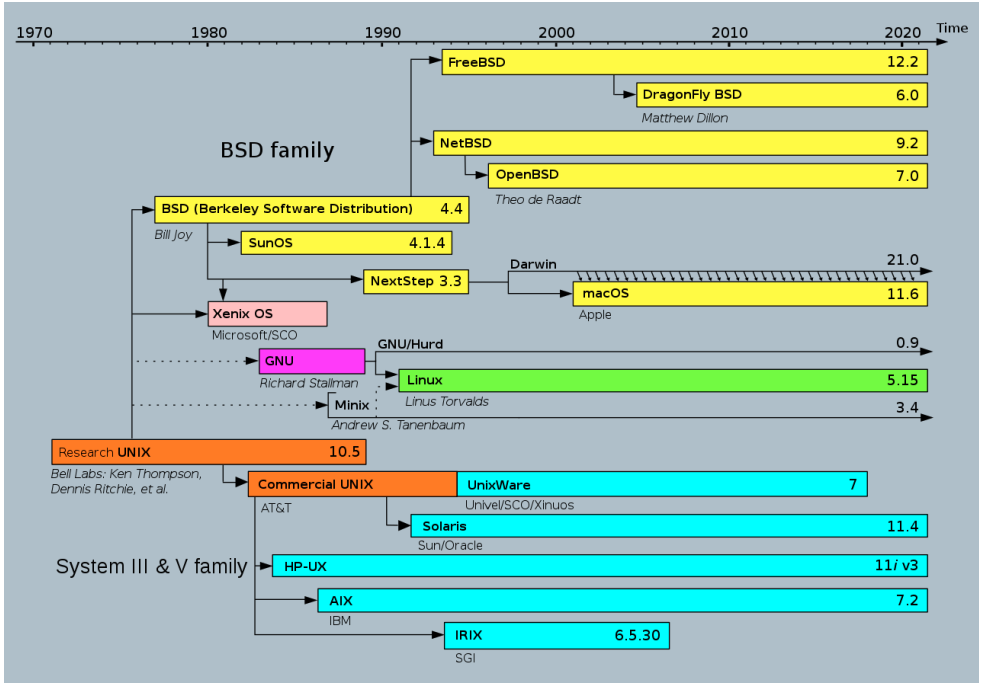
Mais il manquait toujours l'élément central de l'OS : **le noyau**

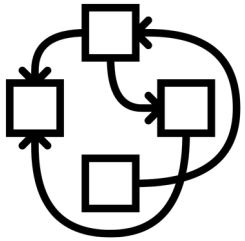
https://www.gnu.org/
https://www.gnu.org/software/software.html

Kernel GNU : Hurd, puis Linux-libre

En 1990, *The GNU Project* lance la production de son noyau nommé Hurd.

Basé à l'origine sur BSD 4.4, puis sur Mach, Hurd est toujours en cours de développement, soutenu par la *Free Software Fundation* et *The GNU Project*.





En 2012, *The GNU Project* décide d'intégrer officiellement un fork de Linux dans son projet : Linux-libre.

Sans code propriétaire, sans binary blob.

On parle alors de système **GNU/Linux**.

Un système GNU signifiant GNU (dont Hurd).

NB : les distributions utilisent majoritairement GNU/Linux, mais The GNU Project incite à utiliser GNU/Linux-libre.

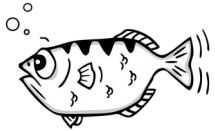
https://en.wikipedia.org/wiki/Linux#/media/File:Unix_timeline.en.svg

Composants de GNU

GNU est composé uniquement de logiciels libres (ce qui est le cas pour GNU/Linux-libre, mais pas GNU/Linux et distributions dérivées).

Les principaux outils proposés par GNU sont :

`GCC`	GNU Compiler Collection (initialement *GNU C Compiler*)
`GDB`	GNU Debugger
`binutils`	GNU Binary Utilities
`glibc`	GNU C library
`coreutils`	GNU Core Utilities (`cat`, `ls`, `mv`, `rm`, …)
`Bash`	Bourne Again Shell
`GRUB`	Grand Unified Bootloader



Liste des 373 packages GNU ici : https://directory.fsf.org/wiki/GNU

FHS – Filesystem Hierarchy Standard

Les systèmes GNU/Linux utilisent par convention le *Filesystem Hierarchy Standard* (FHS), mais d'autres variantes d'UNIX l'utilisent également.

`/` Le répertoire racine du système de fichiers (hiérarchie principale).

`/home` Répertoire des dossiers des utilisateurs (fichiers personnels).

`/home/user1`, `/home/user2`, …

`/root` Répertoire `home` spécifique au super-utilisateur (`root`).

FHS – Filesystem Hierarchy Standard

/boot Fichiers de démarrage (image des kernels, `System.map`, `initrd`).

/etc *Editable Text Configuration*. Fichiers de configuration du système.

/opt *Optional*. Répertoire pour l'installation de logiciels externes.

/var Stockage de données variables, i.e. fichiers dont le contenu est censé varier au fil de l'exécution du système (*logs*, *lock files*, *spool*, …).

FHS – Filesystem Hierarchy Standard

/proc Système de fichiers virtuel contenant sous forme de fichier les informations des processus et du kernel. Répertoire forcément dynamique.

/proc/<pid> Contient tous les paramètres du processus désigné.

```
dboudier:/proc$ ls
1     114   1240  14    1522  170   17354 181   1927  20056 20577 2091  220   27    313   394   47    55    6149  6766  78    904   98            filesystems  meminfo      sysvipc
10    1140  1242  1401  1543  171   17355 18116 1932  20095 20590 2094  222   28    3138  4     48    5512  62    68    780   906   acpi          fs           misc         thread-self
100   1145  1244  1402  1549  172   17356 182   1935  20122 20614 2097  2221  282   316   40    49    5582  6336  6965  783   908   asound        i8k          modules      timer_list
1002  1146  1247  1403  1556  1723  17357 183   1938  20124 20664 2099  2267  2830  32    402   50    56    636   6979  79    909   bootconfig    interrupts   mounts       tty
101   1147  1248  1426  1561  173   17358 18440 1940  2013  20689 2101  2274  29    320   41    52    5610  637   7     790   91    buddyinfo     iomem        mtrr         uptime
102   1148  1253  1435  1565  17339 17359 185   1953  20147 2069  2105  2293  298   34    416   521   5665  638   70    80    910   bus           ioports      net          version
103   1149  1257  146   1570  17340 17360 1852  1956  20149 20691 2107  23    299   340   42    522   5732  639   71    82    912   cgroups       irq          pagetypeinfo version_signature
104   115   1264  1466  1574  17341 17361 1867  1961  2015  20719 2108  2315  3     341   423   523   5736  64    7123  83    913   cmdline       kallsyms     partitions   vmallocinfo
106   1150  1268  1467  1581  17344 1741  1883  1967  20191 20747 2109  2393  30    346   426   524   58    640   72    84    914   consoles      kcore        pressure     vmstat
107   1151  127   1481  1590  17345 1746  18859 197   20199 20755 2113  24    300   35    427   526   59    641   724   85    917   cpuinfo       keys         schedstat    zoneinfo
108   1152  1279  1482  1596  17346 175   18882 198   20217 20788 2115  2401  302   357   43    527   5958  646   73    86    918   crypto        key-users    scsi
109   1153  1283  15    16    17347 1754  18905 19876 20228 20801 2117  2402  304   36    44    5273  6     6472  733   88    92    devices       kmsg         self
11    116   1284  1507  164   17348 177   18918 19946 2025  20816 212   2405  305   37    4415  5278  60    65    734   886   920   diskstats     kpagecgroup  slabinfo
110   1167  1285  1509  165   17349 1774  19    19950 20372 2086  2121  2413  306   38    4418  528   6014  6544  735   887   921   dma           kpagecount   softirqs
1103  1168  1286  1512  1656  17350 178   190   1998  20382 2087  2133  2443  307   382   46    53    607   66    739   888   94    driver        kpageflags   stat
111   12    13    1516  166   17351 179   1909  2     20388 2088  2137  25    308   385   467   5314  61    67    74    89    95    dynamic_debug loadavg      swaps
112   1235  1308  1519  169   17352 18    1911  20    2040  2089  2143  2571  309   386   468   5353  613   6712  76    9     96    execdomains   locks        sys
113   1236  1396  1521  17    17353 1807  1919  20024 205   209   22    26    31    391   469   54    614   6742  77    90    97    fb            mdstat       sysrq-trigger
```

En vrac : `ps -Af` `ls /proc` `ls -l /proc/<pid>` `cat /proc/<pid>/status` `cat /proc/cpuinfo` `cat /proc/modules` `cat /proc/filesystems` `cat /proc/meminfo`

Voir /Documentation/filesystems/proc.rst

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

`/dev` *Devices files*. Liste des périphériques sous forme de fichiers (*device nodes*).

`/media` Points de montage pour les supports amovibles (clé USB, CD, DD externe, …).

`/mnt` Points de montage temporaire pour les systèmes de fichiers.

`/sys` Système de fichiers virtuel contenant sous forme de fichier les informations du matériel et des pilotes.

Par opposition à `/dev`, `/sys` est géré dynamiquement.

`/sys` est pour le matériel ce que `/proc` est pour le logiciel.

FHS – Filesystem Hierarchy Standard

/bin Binaires des commandes utilisateurs (`apt`, `cat`, `cd`, `cp`, `ls`, `mv`, …).
Le FHS impose une liste minimale de commandes.

/sbin Binaires des commandes systèmes (`init`, `route`, … réservées au *super-user*).
Certaines sont en accès restreint aux autres utilisateurs (`ifconfig`, …).

/lib Bibliothèques partagées pour les binaires de `/bin` et `/sbin`.

/lib32 La commande `ldd <binary>` indique les bibliothèques utilisées par le binaire.

/lib64

```
dboudier:/etc/init.d$ ldd /bin/true
        linux-vdso.so.1 (0x00007ffcf9122000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3342c45000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f3342e64000)
```

Glibc

Dynamic linker

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

`/usr` USR = **UNIX Shared Resources** (et non "~~Usef~~").
Hiérarchie secondaire (sous-répertoires) en lecture seule et accès partagé.

`/usr/bin` Binaires des commandes non-essentielles.

`/usr/sbin` Binaires des commandes systèmes non-essentielles .

`/usr/lib` Librairies pour les répertoires `/usr/bin` et `/usr/sbin` .

`/usr/src` Fichiers sources des kernels.

FHS – Filesystem Hierarchy Standard

```
dboudier:/$ ls -l
total 2097240
lrwxrwxrwx   1 root root          7 août  31  2020 bin -> usr/bin
drwxr-xr-x   4 root root       4096 févr. 18 11:08 boot
drwxr-xr-x   2 root root       4096 août  31  2020 cdrom
drwxr-xr-x  24 root root       5680 févr. 18 10:07 dev
drwxr-xr-x 150 root root      12288 févr. 18 11:08 etc
drwxr-xr-x   4 root root       4096 avril 15  2020 home
lrwxrwxrwx   1 root root          7 août  31  2020 lib -> usr/lib
lrwxrwxrwx   1 root root          9 août  31  2020 lib32 -> usr/lib32
lrwxrwxrwx   1 root root          9 août  31  2020 lib64 -> usr/lib64
lrwxrwxrwx   1 root root         10 août  31  2020 libx32 -> usr/libx32
drwx------   2 root root      16384 août  31  2020 lost+found
drwxr-xr-x   3 root root       4096 janv. 18  2021 media
drwxr-xr-x   2 root root       4096 juil. 31  2020 mnt
drwxr-xr-x  11 root root       4096 janv. 14 18:04 opt
dr-xr-xr-x 460 root root          0 févr. 18 10:07 proc
drwx------  18 root root       4096 janv. 14 18:05 root
drwxr-xr-x  39 root root       1100 févr. 18 11:14 run
lrwxrwxrwx   1 root root          8 août  31  2020 sbin -> usr/sbin
drwxr-xr-x  26 root root       4096 sept. 22 10:33 snap
drwxr-xr-x   2 root root       4096 juil. 31  2020 srv
-rw-------   1 root root 2147483648 janv. 11 10:26 swapfile
dr-xr-xr-x  13 root root          0 févr. 18 10:07 sys
drwxrwxrwt  26 root root      12288 févr. 18 11:28 tmp
drwxr-xr-x  14 root root       4096 oct.  13  2020 usr
drwxr-xr-x  14 root root       4096 juil. 31  2020 var
```

Notons que certains distributions (ici Ubuntu) prennent la liberté d'effectuer un lien symbolique (symlink) entre différents répertoires.

BOOTLOADER

Chargeur d'amorçage

Firmwares

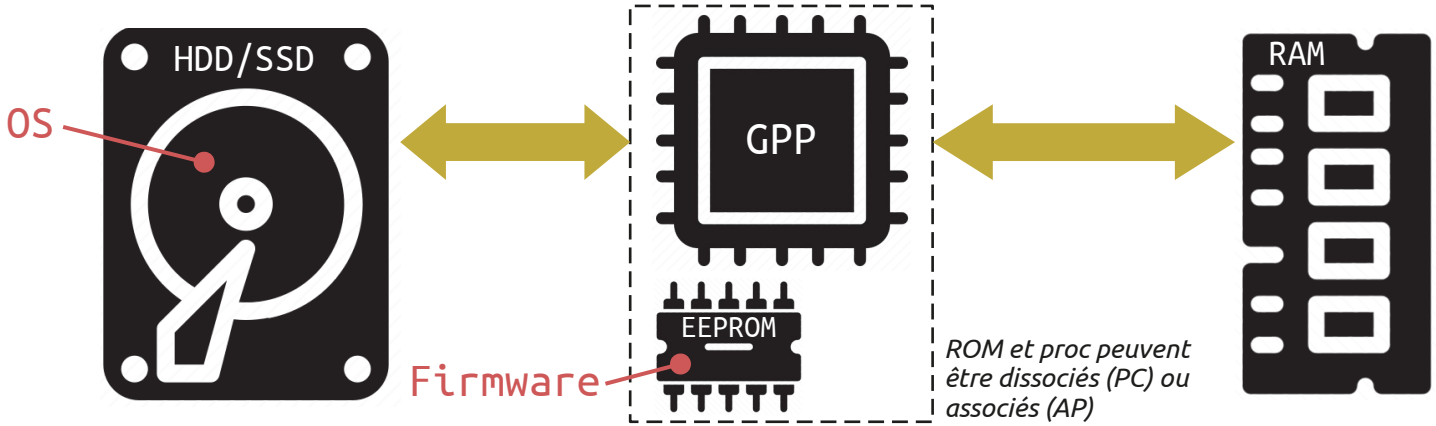
Bootloaders

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Un système d'exploitation a vocation a être modifié régulièrement, que ce soit par l'utilisateur ou par l'entreprise qui en assure la distribution et/ou le support. Il est donc judicieux de déposer l'OS sur une mémoire de masse externe au processeur (disque dur, Flash, clé USB, …).

Par opposition, un *firmware* est censé être figé pour la durée de vie du système. Il est intégré à la ROM interne (ou associée) du processeur.



Chargeur d'amorçage

Au démarrage d'un ordinateur, il faut alors charger le système d'exploitation depuis la mémoire de stockage de masse (HDD, SSD, SD, etc) vers la mémoire de travail (RAM).

C'est le rôle du *bootloader* ou chargeur d'amorçage.

Cette fonction ne peut pas être effectuée par un OS standard du marché, puisque la stratégie de boot (amorçage) peut évoluer dans le temps en fonction du besoin de l'utilisateur (multi-boot, multi-OS, média de masse HDD, SSD, USB ou réseau, …).

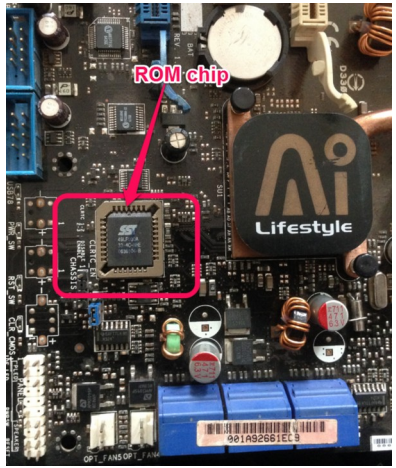
Le système d'amorçage peut être constitué de plusieurs étages de bootloader. Les premiers étages de boot sont très primitifs (peu de services inclus) et peuvent potentiellement se trouver sur des médias physiques séparés (BIOS, UEFI, sur ROM, Flash, MCU externes). Les niveaux de boot évolués (GRUB, U-Boot, Barebox, etc) se trouve en général sur le même média que l'OS.

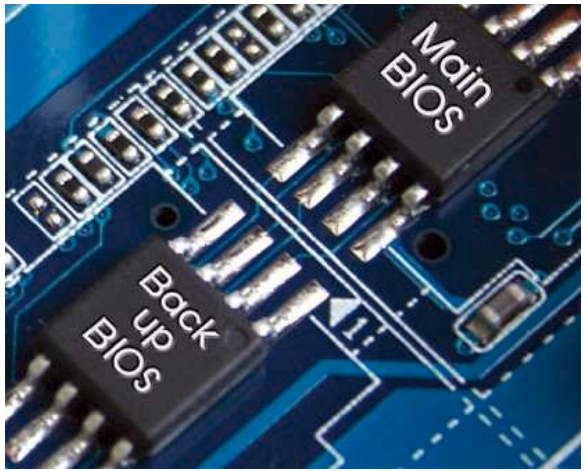
Le firmware charge le bootloader en RAM, puis ce dernier chargera l'OS en RAM.

Le BIOS (*Basic Input Output System*) est un firmware très répandu sur les cartes mères.

Outre l'initialisation des composants et l'identification des périphériques, il a pour mission la lecture du MBR (*Master Boot Record* ou secteur d'amorçage) d'un disque.

Parmi les informations contenues dans le MBR se trouve l'adresse du bootloader.





Le successeur du BIOS est l'UEFI (*Unified Extensible Firmware Interface*).

Il offre évidemment plus de fonctionnalité que le vieux BIOS écrit en assembleur, mais supporte ce standard pour satisfaire des OS anciens.

L'UEFI est capable de travailler avec le réseau, d'utiliser une IHM de meilleure qualité et de supporter des disques de taille supérieure à 2,2 To.

En ce qui nous intéresse (chargement du bootloader), l'UEFI supporte le système de partitionnement GPT (*Globally unique identifier Partition Table*) en plus du MBR.

Cependant le système GPT est encore peu répandu dans l'embarqué (pas de besoins).

Sur les ordinateurs personnels, les bootloaders sont par défaut transparents.

On n'y accède qu'en interrompant le BIOS/UEFI, ou en cas de dual-boot.

GNU Grub (*Grand Unified Bootloader*) est le bootloader par défaut des distributions GNU/Linux. Il est capable de démarrer différents OS à partir de tous les systèmes de fichiers connus (contrairement aux deux homologues ci-dessous).

Windows Boot Manager est le bootloader Microsoft utilisé depuis Windows Vista.

Boot Camp est le bootloader d'Apple, capable de lancer un OS Windows.

Das U-boot : bootloader pour l'embarqué

Avec **Barebox**, le bootloader le plus connu pour les OS GNU/Linux embarqués est **Das U-boot** (the Universal Bootloader) (https://www.denx.de/wiki/U-Boot/).

Il fonctionne sur x86, ARM, RISC-V, MicroBlaze, MIPS, …

Il supporte plusieurs filesystems et peut charger un kernel depuis une interface SATA, SD, I^2C, eMMC, USB, port série, réseau, ...

U-boot propose une CLI (console ou liaison série) pour régler ses paramètres en direct.

Il est décomposé en deux étages :

- Un SPL (*Secondary Program Loader*) très léger qui fera les initialisations nécessaires au bootloader complet
- U-boot lui-même, qui configurera les contrôleurs mémoire et chargera le kernel en RAM.

Das U-boot : Exemple de stage 3A SATE chez BOOTLIN en 2020

Bootlin contributes SquashFS support to U-Boot

SquashFS is a very popular read-only compressed root filesystem, widely used in embedded systems. It has been supported in the Linux kernel for many years, but so far the U-Boot bootloader did not have support for SquashFS, so it was not possible to load a kernel image or a Device Tree Blob from a SquashFS filesystem in U-Boot.

Between February 2020 and August 2020, João Marcos Costa from the ENSICAEN engineering school, has worked at Bootlin as an intern. João's internship goal was specifically to implement and contribute to U-Boot the support for the SquashFS filesystem. We are happy to announce that João's effort has now completed, as the support for SquashFS is now in upstream U-Boot. It can be found in fs/squashfs/ in the U-Boot source code.

https://bootlin.com/blog/bootlin-contributes-squashfs-support-to-u-boot/

BareBox : bootloader pour l'embarqué

Initialement u-boot-v2, BareBox est un fork de Das U-boot.

https://www.barebox.org/

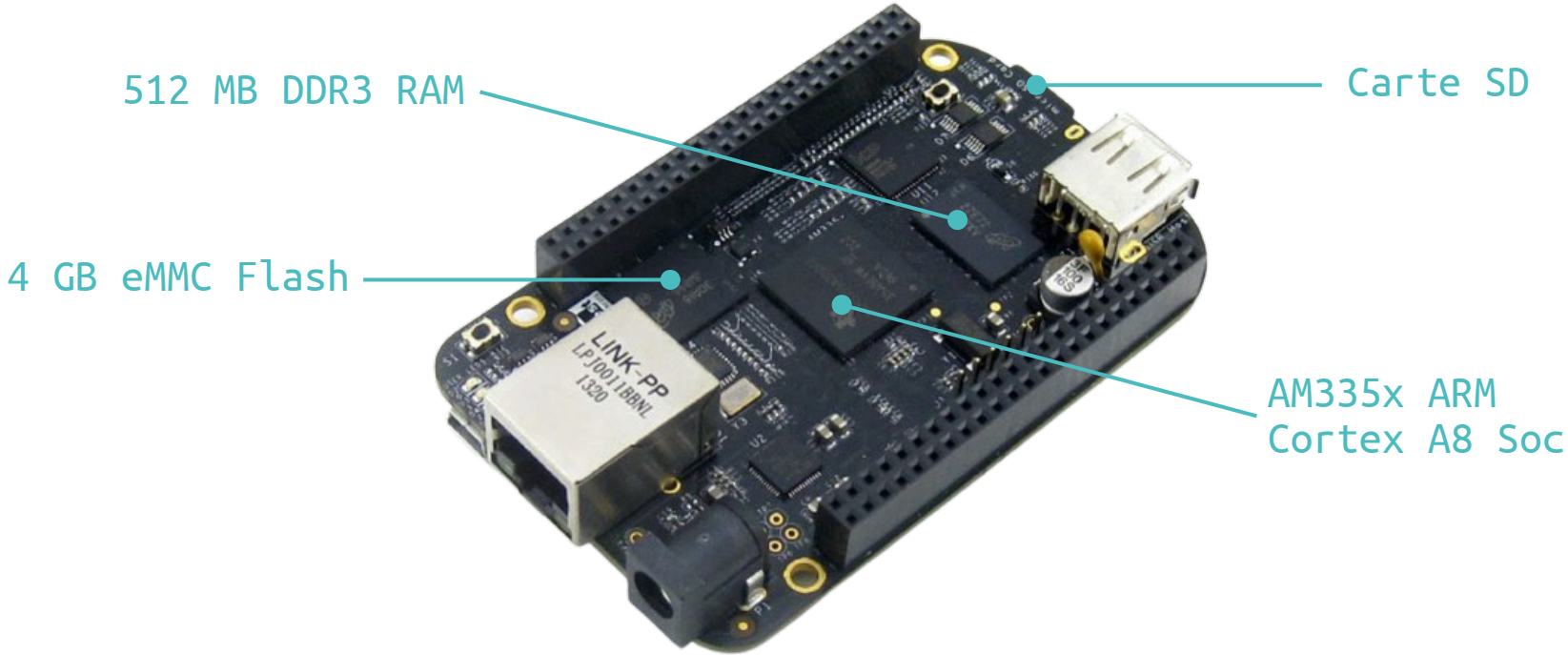
Il vise les systèmes embarqués, et support de nombreuses architectures (ARM, x86, MIPS, PowerPC, …) et filesystems.

SÉQUENCE D'AMORÇAGE

Cas de la BeagleBone Black

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Prenons comme exemple la BeagleBone Black sur laquelle nous travaillerons en TP.



https://beagleboard.org/black/

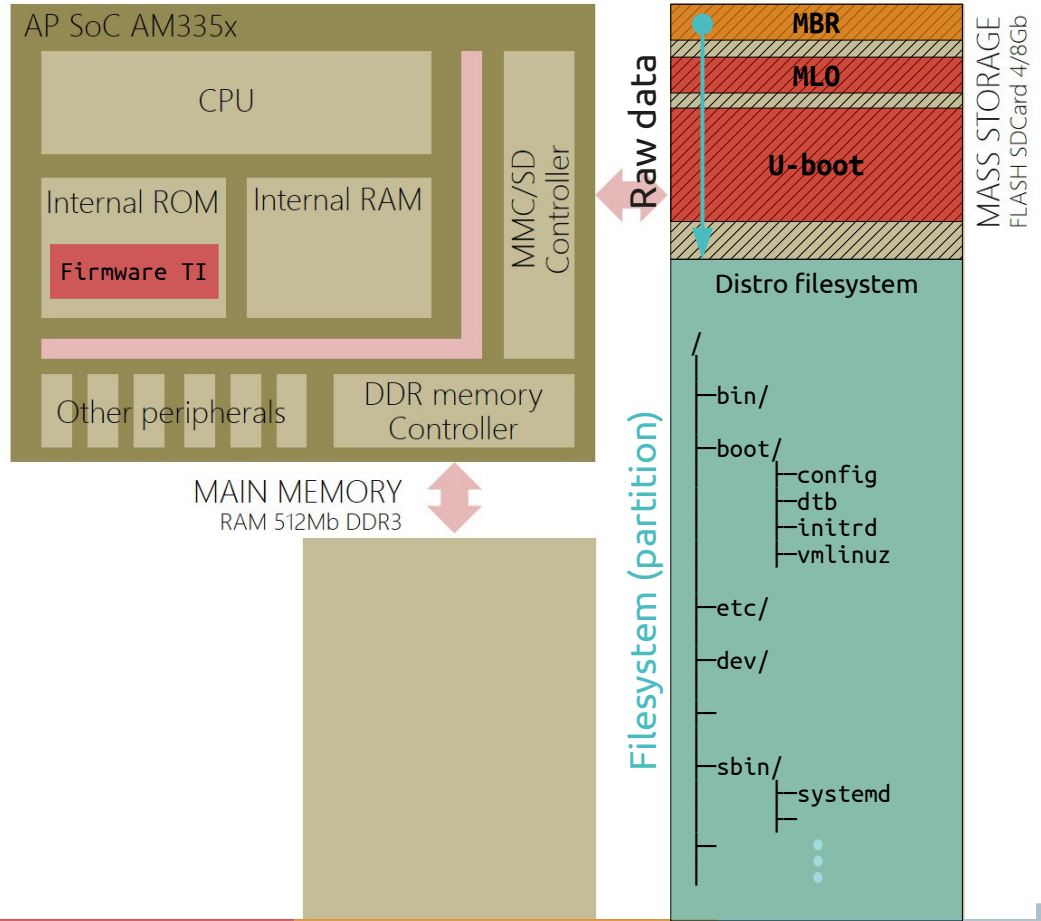
Système hors tension

Le firmware Boot Rom est contenu dans la ROM interne au SoC AM335x de TI.

MLO (édité par TI) et U-boot (édité par DENX) sont les bootloaders externes, stockés en binaire dans le média de masse.

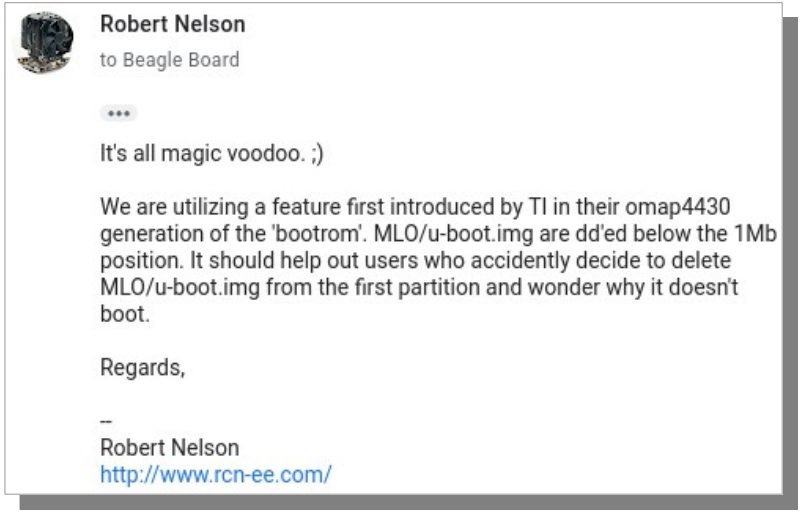
La distribution GNU/Linux (et donc le kernel) se trouvent sur une partition (système de fichier).

La partition est indiquée par le MBR, présent sur les 512 premiers octets de mémoire de masse.



Précisions sur les bootloaders.

Le firmware Boot Rom est capable d'aller chercher un bootloader sur un système de fichier FAT. Ici, le développeur Linux pour BBB (R.C. Nelson) a fait le choix de le placer dans un espace en donnée brute (hors fs).



Robert Nelson
to Beagle Board

•••

It's all magic voodoo. ;)

We are utilizing a feature first introduced by TI in their omap4430 generation of the 'bootrom'. MLO/u-boot.img are dd'ed below the 1Mb position. It should help out users who accidently decide to delete MLO/u-boot.img from the first partition and wonder why it doesn't boot.

Regards,

--
Robert Nelson
http://www.rcn-ee.com/

U-boot est un bootloader complexe, décomposé en deux étages.

Le premier étage est un SPL (*Secondary Program Loader*). Ce rôle est rempli par MLO (Minimal Bootloader), intégré au code source de U-boot par Texas Instruments.

TI fournit ce programme pour assurer la portabilité de U-boot (solution ultra-répandue) pour leur processeur.

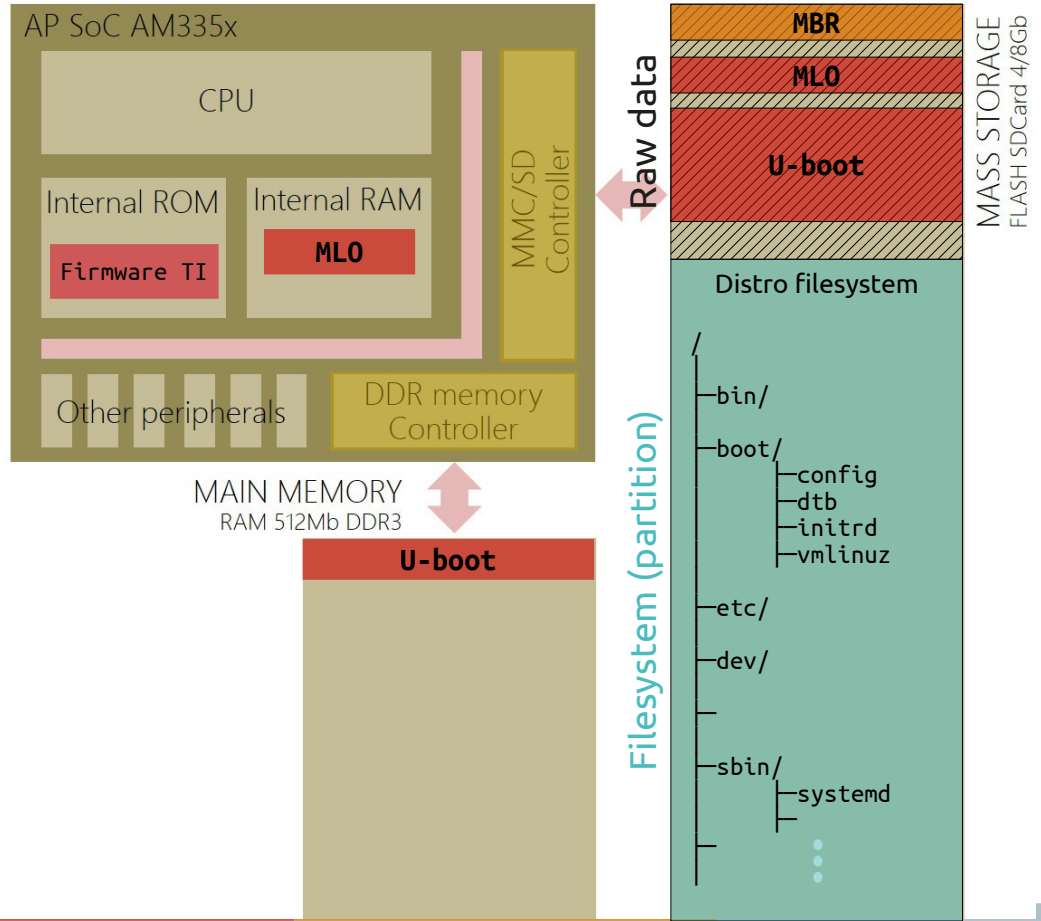
Mise sous tension

Le firmware TI active le contrôleur MMC/SD.

Il parcourt la partition binaire brute, identifie le MLO et le charge alors en RAM interne au SoC.

À son tour, le MLO active le contrôleur DDR puis localise U-boot.

Il transfère U-boot en RAM externe.



Comment Boot Rom détecte MLO dans un espace binaire brut (sans filesystem).

26.1.11 Table of Contents

The Table of Contents (TOC) is a header needed only in GP devices while using MMC RAW mode. This must not be confused with the TOC used in HS devices. The TOC is 512 bytes long and consists of a maximum of 2 TOC items (32 bytes long each), located one after the other. The second TOC item must be filled by FFh. Each TOC item contains information required by the ROM Code to find a valid image in RAW mode, as illustrated in Table 26-38. To detect RAW mode, the ROM also needs the magic values mentioned in Table 26-39. Other than the TOC item fields and magic values, all the other bytes in the 512-byte TOC must be zero.

Table 26-38. The TOC Item Fields

Offset	Field	Size (bytes)	Description
0000h	Start	4	0x00000040
0004h	Size	4	0x0000000C
0008h	Flags	4	Not used, should be zero.
000Ch	Align	4	Not used, should be zero.
0010h	Load Address	4	Not used, should be zero.
0014h	Filename	12	12 character long name of sub image, including the zero ('\0') terminator. The ASCII representation is "CHSETTINGS".

Table 26-39. Magic Values for MMC RAW Mode

Offset	Value
40h	0xC0C0C0C1
44h	0x00000100

The ROM Code recognizes the TOC based on the filename described in Table 26-40.

Table 26-40. Filenames in TOC for GP Device

Filename	Description
CHSETTINGS	Magic string used by ROM

C'est défini dans le *Technical Reference Manual* du processeur, encore faut-il le trouver !

```
1 00000000: 4000 0000 0c00 0000 0000 0000 0000 0000  @...............
2 00000010: 0000 0000 4348 5345 5454 494e 4753 0000  ....CHSETTINGS..
3 00000020: ffff ffff ffff ffff ffff ffff ffff ffff  ................
4 00000030: ffff ffff ffff ffff ffff ffff ffff ffff  ................
5 00000040: c1c0 c0c0 0001 0000 0000 0000 0000 0000  ................
6 00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
7 00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Extrait du fichier MLO (après conversion binaire → texte)

MMC Raw mode, p5077

AM335x and AMIC110 Sitara™ Processors Technical Reference Manual

https://www.ti.com/lit/pdf/spruh73

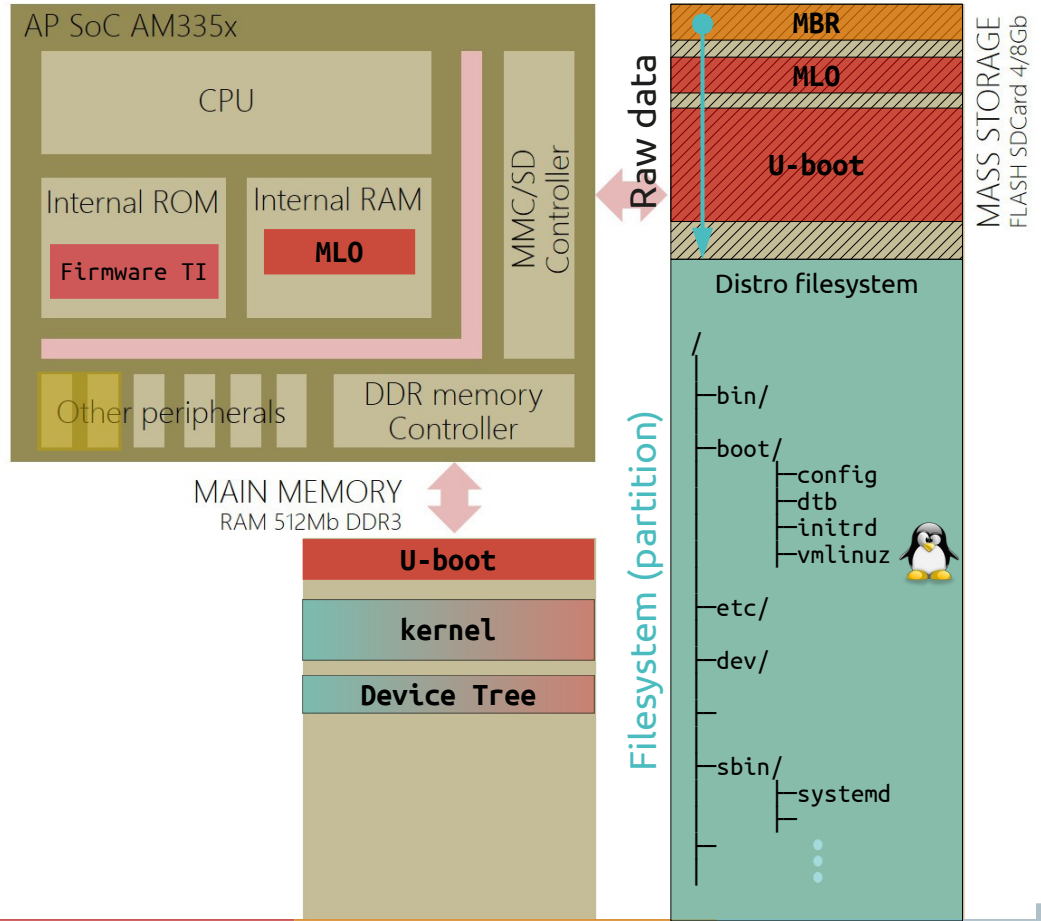
Mise sous tension (la suite)

La première tâche d'U-boot est de configurer certains périphériques (notamment la liaison série pour IHM).

Une fois prêt, U-boot lit le MBR pour identifier les partitions de fichiers dans la mémoire de masse.

Il parcourt le système de fichier de chaque partition jusqu'à trouver l'image compressée du kernel (vmlinuz) et le device tree.

Il les charge en RAM, puis donne la main au kernel quand sa mission est terminée.



Lancement du kernel

Le point d'entrée dans le kernel dépend de l'architecture. C'est le fichier assembleur :

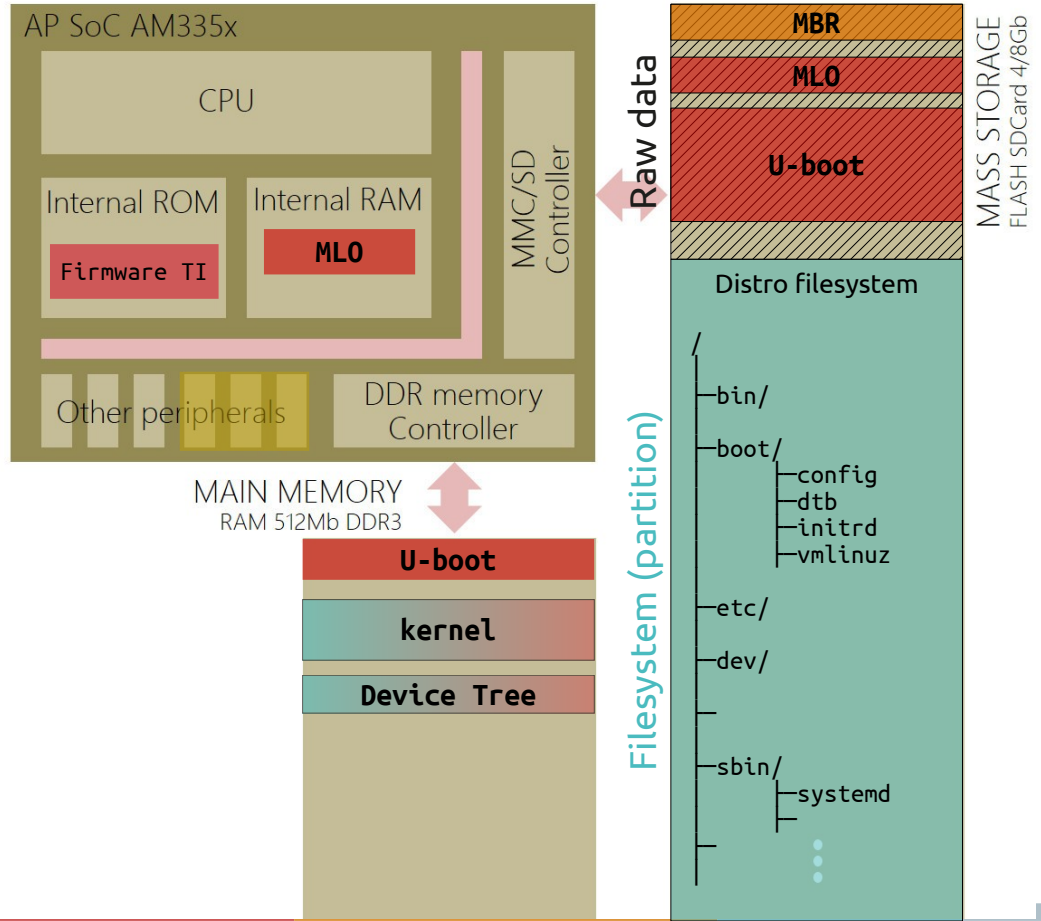
`arch/arm/boot/compressed/head.S`
label « `start` »

Le kernel initie sa décompression.

Toujours en lien avec le matériel (donc asm), le kernel configure le cache et la MMU.

L'exécution du kernel se poursuit, entrant maintenant dans la phase indépendante de l'architecture :

fichier `init/main.c` , fonction `start_kernel()`



Lancement du kernel

Le fichier `head.S` (à gauche) est le premier point d'entrée dans le kernel et dépend de l'architecture.

Le fichier `init/main.c` (à droite) est le premier point d'entrée architecture-agnostique du kernel.

```
1   /* SPDX-License-Identifier: GPL-2.0-only */
2   /*
3    *  linux/arch/arm/boot/compressed/head.S
4    *
5    *  Copyright (C) 1996-2002 Russell King
6    *  Copyright (C) 2004 Hyok S. Choi (MPU support)
7    */
8   #include <linux/linkage.h>
9   #include <asm/assembler.h>
10  #include <asm/v7m.h>
11
12  #include "efi-header.S"
```

```
185                 .align
186                 /*
187                  * Always enter in ARM state for CPUs that support the ARM ISA.
188                  * As of today (2014) that's exactly the members of the A and R
189                  * classes.
190                  */
191  AR_CLASS(      .arm    )
192 start:
193                 .type   start,#function
194                 /*
195                  * These 7 nops along with the 1 nop immediately below for
196                  * !THUMB2 form 8 nops that make the compressed kernel bootable
197                  * on legacy ARM systems that were assuming the kernel in a.out
198                  * binary format. The boot loaders on these systems would
199                  * jump 32 bytes into the image to skip the a.out header.
200                  * with these 8 nops filling exactly 32 bytes, things still
201                  * work as expected on these legacy systems. Thumb2 mode keeps
202                  * 7 of the nops as it turns out that some boot loaders
203                  * were patching the initial instructions of the kernel, i.e
204                  * had started to exploit this "patch area".
205                  */
206                 __initial_nops
207                 .rept   5
208                 __nop
```

```
927 asmlinkage __visible void __init __no_sanitize_address start_kernel(void)
928 {
929         char *command_line;
930         char *after_dashes;
931
932         set_task_stack_end_magic(&init_task);
933         smp_setup_processor_id();
934         debug_objects_early_init();
935         init_vmlinux_build_id();
936
937         cgroup_init_early();
938
939         local_irq_disable();
940         early_boot_irqs_disabled = true;
941
942         /*
943          * Interrupts are still disabled. Do necessary setups, then
944          * enable them.
945          */
946         boot_cpu_init();
947         page_address_init();
948         pr_notice("%s", linux_banner);
949         early_security_init();
950         setup_arch(&command_line);
951         setup_boot_config();
952         setup_command_line(command_line);
953         setup_nr_cpu_ids();
954         setup_per_cpu_areas();
955         smp_prepare_boot_cpu(); /* arch-specific boot-cpu hooks */
956         boot_cpu_hotplug_init();
957
958         build_all_zonelists(NULL);
959         page_alloc_init();
```

Initialisation des composants matériels et logiciels

Lancement de l'OS et applications

Le kernel est capable de s'auto-décompressé !

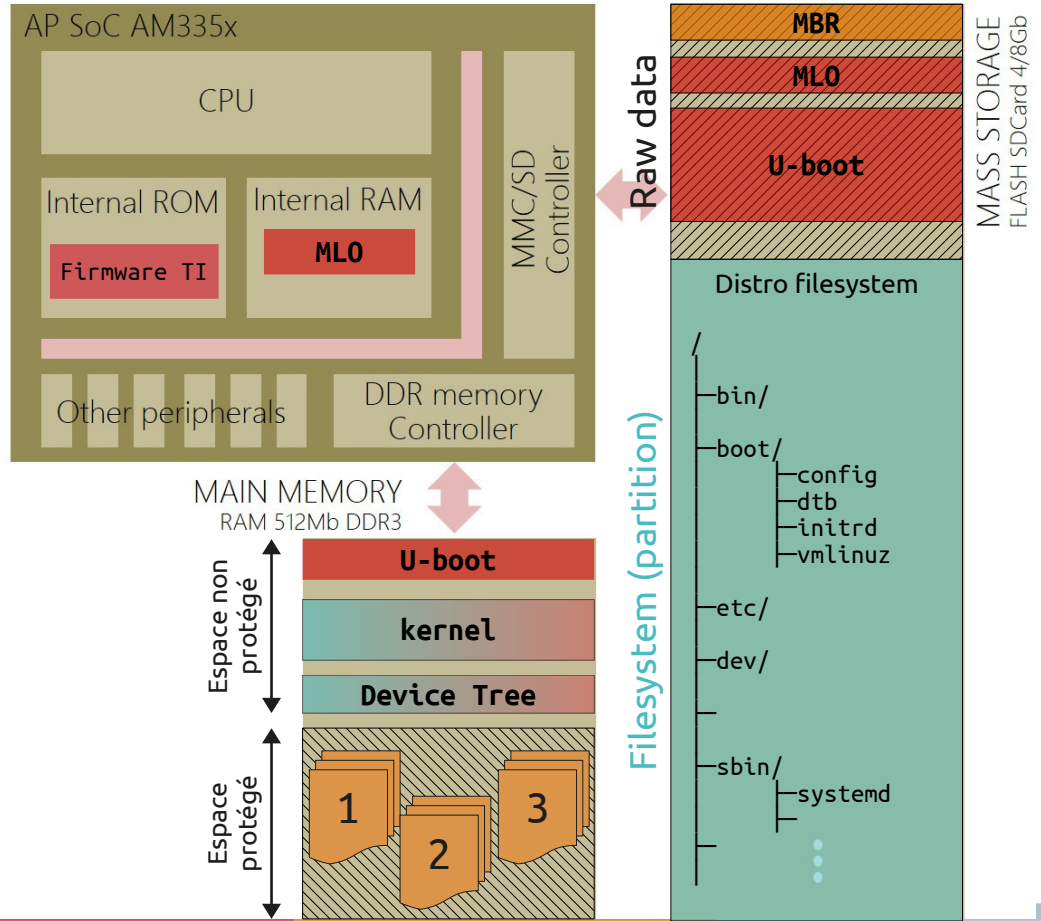
De même, il a configure et exploite la **MMU**. Cela implique que le kernel travaille en d'adressage réel non protégé (lien direct CPU vers RAM avec bypass de la MMU), voir `boot/system.map`.

Tandis que les processus (applications chargées puis exécutées depuis la RAM) tourneront en adressage virtualisé (via la MMU). On appelle cet espace mémoire « espace protégé ».

Après son démarrage, le kernel lance le premier exécutable (PID = 1) :

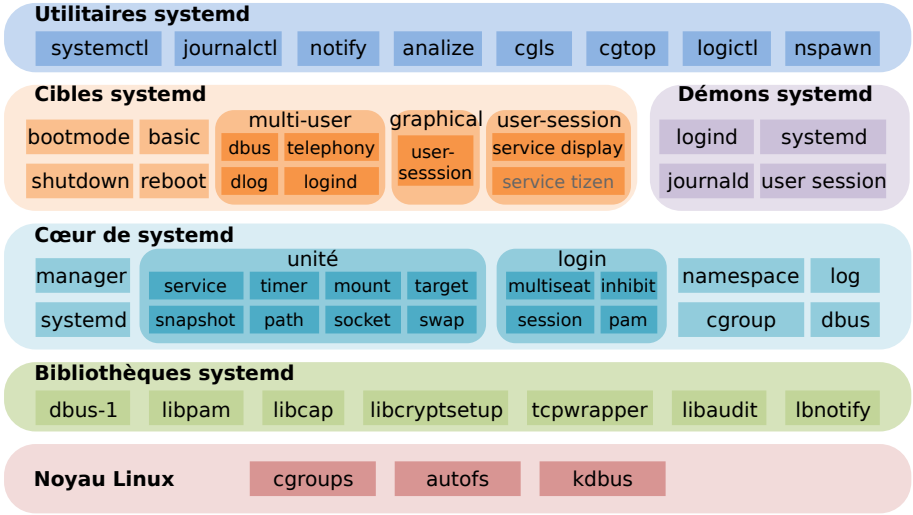
`/sbin/systemd`

Les segments de *code*, *stack* et *heap* arrivent en RAM. La machine est lancée, d'autre processus seront chargés puis exécutés automatiquement en parallèle par systemd.

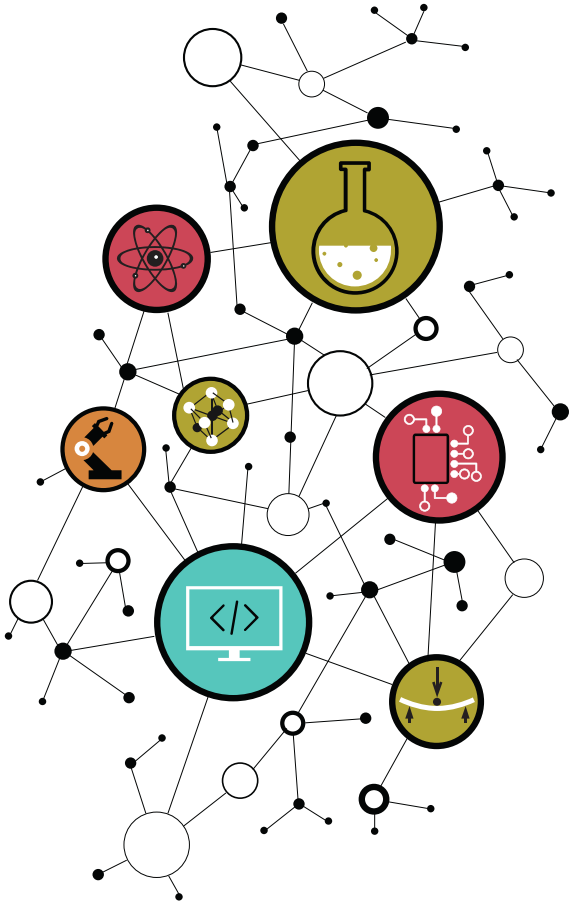


systemd est une suite logicielle qui fournit une gamme de composants système pour les systèmes d'exploitation Linux. Il est une alternative à SysV Init. Systemd a notamment pour but d'offrir un meilleur cadre pour la gestion des dépendances entre services, de permettre le chargement en parallèle des services au démarrage et de réduire les appels aux scripts shell.

En 2020, systemd représente à lui seul 1.273.896 lignes de code et plus de 300 développeurs ont contribué à son développement cette même année.



CONTACT



Dimitri Boudier – PRAG ENSICAEN

dimitri.boudier@ensicaen.fr

Avec l'aide précieuse de :

- Hugo Descoubes (PRAG ENSICAEN)