Master 2 IAP
Année 2018 – 2019

Amorce :

Du néant à la lumière

Djamel CHEKROUN, Hugo POMPOUGNAC, Julien ROLLAND,
Slimane SIROUKANE, Vincent BONNEVALLE

université PARIS DIDEROT PARIS 7

Table des matières

Partie 1

Introduction

1.1 Démarrage de Linux

1.1.1 Introduction

A la mise sous tension, le processus de démarrage de Linux a la charge de transformer un ensemble de matériels inertes en un système cohérent au service de l'utilisateur, capable d'éxécuter des commandes, de lancer des applications sophistiquées (d'intelligence artificielle par exemple), finalement en un système quasi-vivant (!)... C'est dire l'importance de ce processus.

Le processus de démarrage se réalise en une succession d'étapes, qui s'appuient sur un principe simple : les modalités qui président à une étape donnée sont précisées par l'étape précédente, tout en s'appuyant sur des schémas convenus par avance (comment sont organisées les données par exemple). La toute première phase ne peut s'appuyer que sur des données inscrites en dur dans le matériel, ces données étant ensuite interprétées comme du code exécutable qui va orienter les phases suivantes, les différentes phases du processus se faisant ainsi la courte échelle.

Le processus de démarrage de Linux comprend dans les grandes lignes 6 étapes :

Le BIOS (Basic Input Output System) comprend du code exécutable en mémoire morte (ainsi qu'une petite zone de mémoire programmable) qui s'active dès qu'on appuie sur le bouton Marche.
Ce code charge ensuite à partir d'un media externe (un disque dur par exemple), d'un endroit convenu par avance, le MBR (Master Boot Record) qui contient à son tour un petit programme exécutable ainsi qu'une description de l'organisation du disque (les partitions).

Ce petit programme exécutable est en réalité la première partie d'un programme plus large, le bootloader (sa localisation précise est justement indiquée dans le code du MBR) qui va réaliser des tâches plus complexes, jusqu'au chargement d'une première partie du noyau linux (partie primitive qui va commencer à donner vie à la machine).

Ce noyau primitif va ensuite lancer le premier processus Linux (Init) qui se charge de construire tout l'édifice (lancer les processus de service de tous types nécessaires à une vie normale : gestionnaires de terminaux, interface graphique, réseau, etc.), en respectant des fichiers de configuration spécifiés à l'installation du système, ou modifiés ensuite, organisés en différents niveaux d'exécution (run levels).

Toutes ces étapes seront décrites en détail.

Pour appréhender comment ce miracle quotidien peut se produire, il faut comprendre quels dispositifs sont mis en oeuvre, et selon quels arrangements, à chaque étape. En gros, il faut comprendre comment sont organisés :

- les disques : disques durs mais aussi SSD (disque électronique sans partie mécanique), ensembles RAID (plusieurs disques pour gagner en capacité et fiabilité) et LV (logical volumes, "disques logiques").
- les types de systèmes de fichiers : c'est à dire la façon dont sont organisées les données sur le disque afin que le noyau puisse les lire
- les partitions : c'est-à-dire l'organisation logique du disque physique, chaque partition étant réservée à un ensemble de données cohérent
- les systèmes de fichiers proprement dits (ou file systems), c'est-à-dire les branches qui constituent l'arborescence globale des fichiers, abstraction simple manipulée par les plus hautes couches du système et par l'utilisateur, ainsi que leur localisation et organisation sur le disque
- le processus d'installation du système sur la machine, qui détermine la façon dont le système démarrera ensuite. Le processus de démarrage peut bien sûr être modifié à partir d'un système Linux opérationnel, les modifications étant prises en compte dès le démarrage suivant.

FIGURE 1.1 – Étapes du démarrage d'un ordinateur

1.1.2 Disques

1.1.2.1 disques HDD (Hard Disk Drive)

Un disque HDD est constitué d'une pile de plateaux que lisent des têtes magnétiques solidaires d'un bras mécanique commandé par un contrôleur. Chaque plateau (platter) est organisé en pistes (tracks) concentriques, découpées en secteurs. Une colonne de secteurs constitue un cylindre (cylinder), que peuvent alors lire en parallèle les différentes têtes de lecture. Le système, quant à lui, n'interagit avec le disque (à travers son contrôleur) qu'en termes de blocs de données. Un bloc est une unité logique constituée de plusieurs secteurs consécutifs. En général, un secteur correspond à 512 octets et un bloc à 4096 octets.

Nous préciserons, chaque fois que c'est pertinent, les principales possibilités offertes à l'utilisateur (commandes, fichiers de configuration) pour intervenir dans le champ des thèmes abordés.

commandes de gestion des disques :
`du`
indique l'occupation sur le disque d'un catalogue ou fichier en blocs ou octets
`df`
indique de façon détaillée l'occupation sur le disque d'un système de fichiers

1.1.2.2 disques SSD (Solid State Drive)

Un disque SSD contient des pages de mémoire flash. 128 pages constitue un bloc, qui constitue l'unité d'accès au disque. Aucune partie mécanique n'est en mouvement et l'accès aux données est beaucoup plus rapide. Attention cependant à la persistence des données. Les données peuvent en effet être détériorées après un ou deux ans sans alimentation. Linux doit signaler au SSD les blocs des données détruites afin que le SSD puisse les effacer (ce qui accélérera les écritures suivantes). C'est la commande TRIM (Linux prend en compte cette commande avec le système de fichiers ext4).

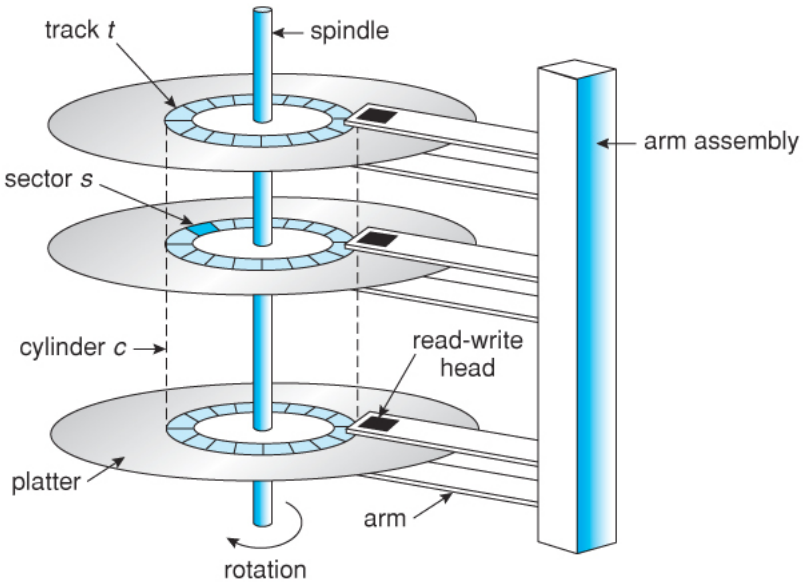


FIGURE 1.2 – Structure d'un disque dur

1.1.2.3 disques RAID (Redundant Array of Independant Disks)

Pour obtenir une fiabilité accrue et/ou un temps d'accès plus rapide, on rassemble des disques bon marché en une matrice (array). Les données sont alors réparties ou recopiées sur plusieurs disques (pour gagner en fiabilité) ou bien des accès peuvent être réalisés en parallèle (pour gagner en temps d'accès). Le contrôleur de l'ensemble peut être hardware ou software. Un array de disques RAID apparaît comme un seul disque à l'OS (sauf au démarrage ou un traitement particulier est exigé). Tous les loaders ne sont pas capable de booter sur un RAID (GRUB2 a cette capacité).

Voici les types d'array RAID les plus fréquents :

RAID 0 : chaque fichier est découpé en morceaux répartis sur les disques de l'array (striping). L'accès au fichier peut alors se faire en parallèle. Le temps d'accès est meilleur mais il n'y a pas de redondance.

RAID 1 : Les données sont dupliquées (mirorring) pour une meilleure fiabilité.

RAID 3 : On fait du Striping et un disque entier est réservé au contrôle de parité. Le temps d'accès est meilleur et le contrôle de parité offre une tolérance aux pannes (un disque en panne ne perturbe pas l'intégrité des données)

RAID 6 : Même principe que RAID 5 avec tolérance à la panne de deux disques.

1.1.2.4 LVM (Logical Volume Manager)

On peut regrouper des *volumes physiques* (qu'il faut d'abord créer à partir de partitions des disques physiques) en *groupes de volumes* puis créer des *volumes logiques*. Les informations des volumes logiques sont contenues dans les premiers blocs des volumes physiques. Un volume logique est alors vu comme une partition unique. Ce dispositif permet d'obtenir des partitions de taille arbitraire.

commande de gestion des volumes logiques
`pvcreate`
crée des volumes physiques à partir de partitions de disques physiques
`vgcreate`
crée des groupes de volumes logiques à partir de volumes physiques
`lvcreate`
crée des volumes logiques à partir des groupes de volumes

1.1.3 partitions

Une partition correspond à une partie d'un disque physique gérée et organisée de façon indépendante. Un problème survenant dans une partition n'affecte pas le reste du disque. L'ensemble des partitions est décrit

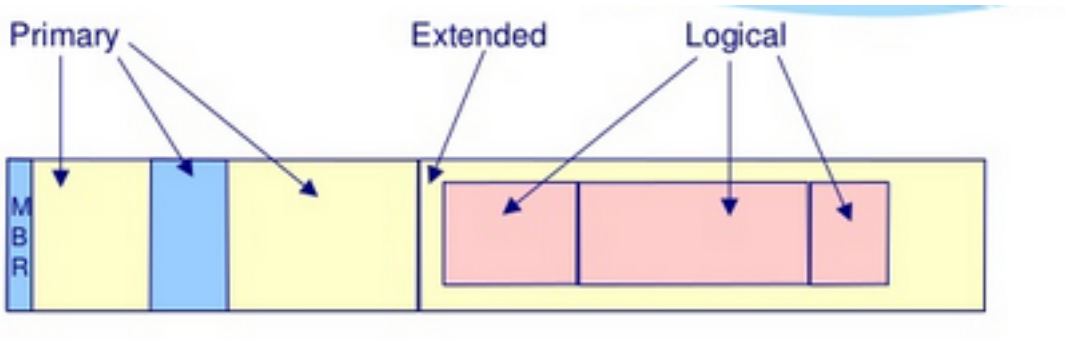


FIGURE 1.3 – Schéma des partitions d'un disque

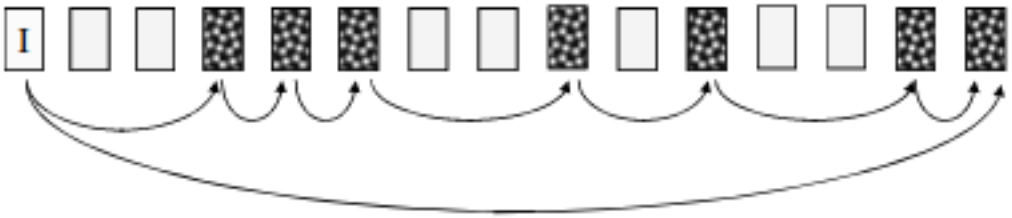


FIGURE 1.4 – Chaînage des blocs d'un système de fichier

dans le MBR. Le partitionnement MBR limite à 4 le nombre de partitions dites primaires, dont l'une peut être étendue (et peut alors contenir plusieurs partitions logiques). Nous verrons que ces limitations sont levées avec les dispositifs succédant au MBR. Chaque partition a un type (FAT32, NTFS, Linux...), lié au type de systèmes de fichiers qu'elle peut contenir.

commandes de gestion des partitions :
`fdisk`
crée une partition en un endroit précis du disque
`Ntfsresize`
pour redimensionner les partitions windows
`parted`
outil de création, destruction, modification des partitions (version graphique gparted)
`mkswap`
création d'une zone de swap

1.1.4 systèmes de fichiers

Dans la terminologie Linux, un système de fichiers désigne l'organisation des données sur le disque mais aussi la partie de l'arborescence des catalogues ainsi inscrite sur disque. Les méthodes d'organisation des fichiers sur disque sont nombreuses (FAT32, NTFS, ext2, ext3, ext4...).
Deux critères en particulier distinguent ces différentes systèmes de fichiers :

- la table qui décrit l'organisation des répertoires et des fichiers sur le disque
- la méthode de désignation des blocs appartenant à un même fichier (qui peuvent être directement pointés ou bien chaînés entre eux par exemple)

Windows utilise les systèmes de fichiers FAT32 et NTFS. Les systèmes de fichiers de Linux (ext2, ext3, ext4) sont basés sur la notion d'inode (*index node*). Un catalogue ne correspond qu'à la gestion d'une table de correspondance entre nom de fichier et numéro d'inode.
ext3 a apporté en particulier la fonction de journalisation qui permet la récupération des données en cas d'incident.

Voic un schéma décrivant le principe des inodes

La structure de données sur disque correspondant à l'inode contient différentes informations concernant le fichier et pointe (directement ou indirectement) les blocs composant le fichier.

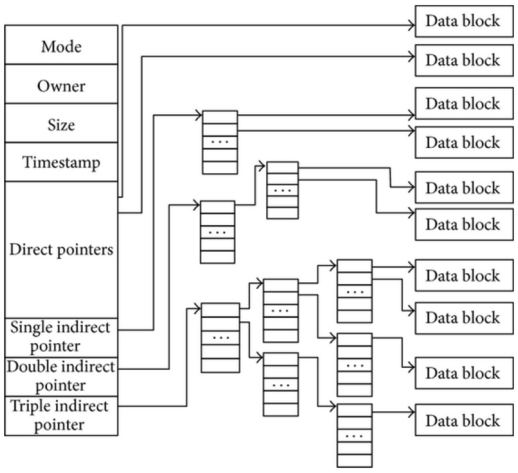


FIGURE 1.5 – Structure d'un *inode*

commandes de gestion des systèmes de fichiers :
`fsck`
vérifie et répare les systèmes de fichiers
`debugfs`
visualise et modifie le contenu d'un système de fichiers

1.1.5 hiérarchie des systemes de fichiers (file systems)

L'ensemble des fichiers Linux est rassemblé en une seule hiérachie de racine "/" ou "root". Un système de fichiers peut correspondre à tout sous-arbre de cette hiérarchie (/home, /var, /temp etc.). Une partition d'un disque ne peut correspondre qu'à un seul système de fichiers. C'est à l'installation du système qu'on affecte des systèmes de fichiers (en spécifiant leur type) à des partitions.

Monter un système de fichiers correspondant à son arrimage à l'arborescence principale. Au démarrage, les différents systèmes de fichiers seront montés à l'arborescence principale, à partir du sommet /root.
Comme dans Linux "tout est fichier", les "fichiers" correspondant aux périphériques sont rangés dans le catalogue /dev et ceux correspondant à la gestion des processus dans le catalogue /proc. Ces fichiers sont dits "fichiers spéciaux". Les périphériques (CDROM, clé USB) sont alors "montés" dans la hiérarchie principale comme les autres systèmes de fichiers.

Les couches hautes dans le noyau du système de gestion des systèmes de fichiers est désignée comme Virtual File System, offrant des fonctions génériques de manipulations des fichiers, indépendantes du type des systèmes de fichiers. C'est ce qui permet à Linux de lire facilement des systèmes de fichiers Windows.

Système de fichiers Linux

Le schéma ci-dessous représente l'organisation d'un système de fichier dans une partition. Le super bloc décrit le contenu du système de fichiers en donnant les localiations des tables d'inodes. Le superbloc est répété plusieurs fois pour pallier une détérioration éventuelle.

commandes pour l'utilisateur :
`mount`, `umount` (p140, p 601)
monte, démonte un système de fichiers
`mkfs`
crée un système de fichiers

1.1.6 Installation

L'installation du système sur une nouvelle machine se fait à partir d'un CD-ROM, clé USB ou encore à partir d'un accès réseau. Il faut programmer le BIOS pour choisir le media de démarrage. Une étape importante de la procédure d'installation est le partitionnement, c'est-à-dire définir les partitions ainsi que les données, correspondant éventuellement à plusieurs systèmes d'exploitation, que ces partitions contiendront.
S'agissant de Linux, une partition est affectée à un système de fichier. On inclut également une partition de

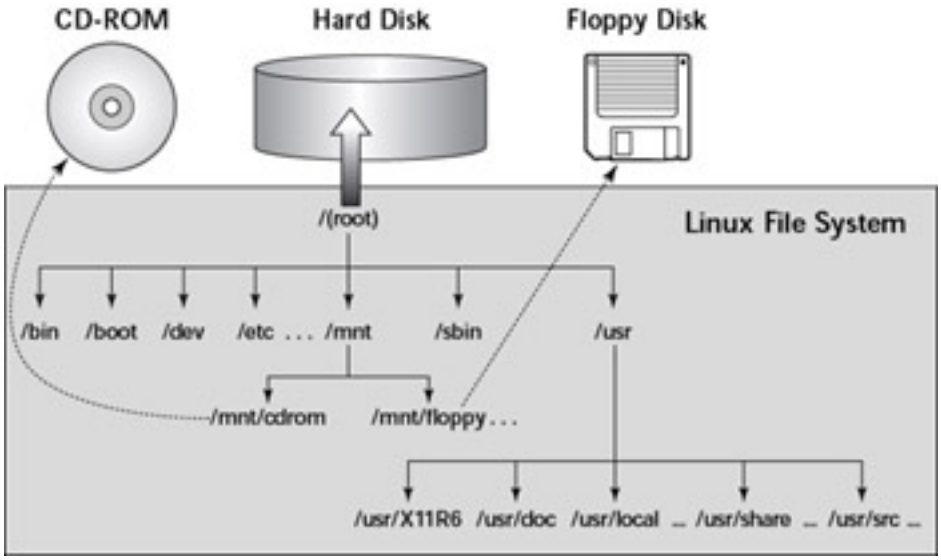


FIGURE 1.6 – Arborescence d'un système de fichier

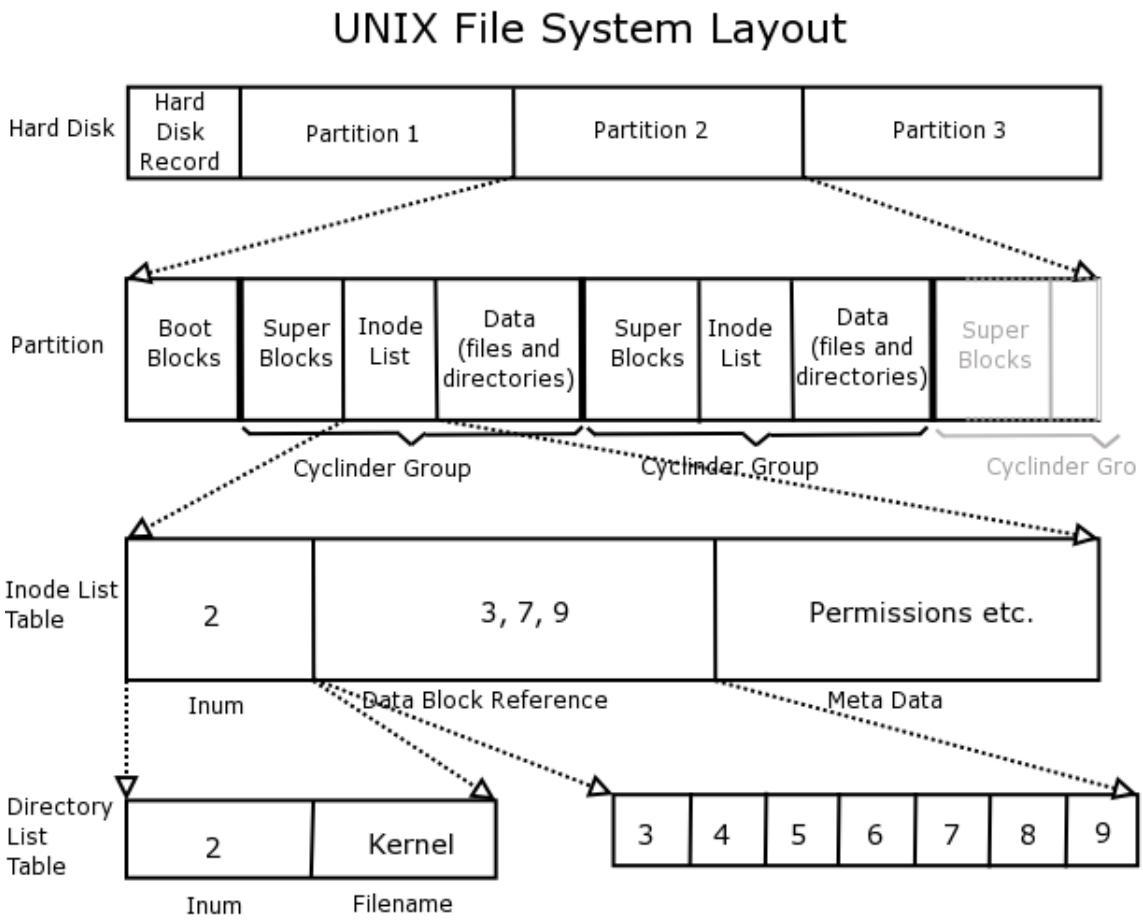


FIGURE 1.7 – Organisation d'un système de fichier dans une partition

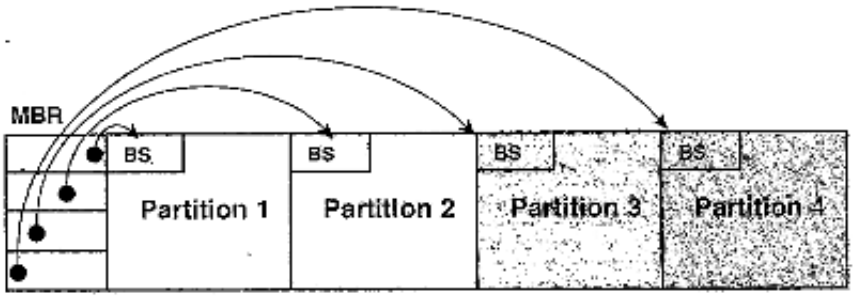


Figure 12.3 • MBR et boot sectors.

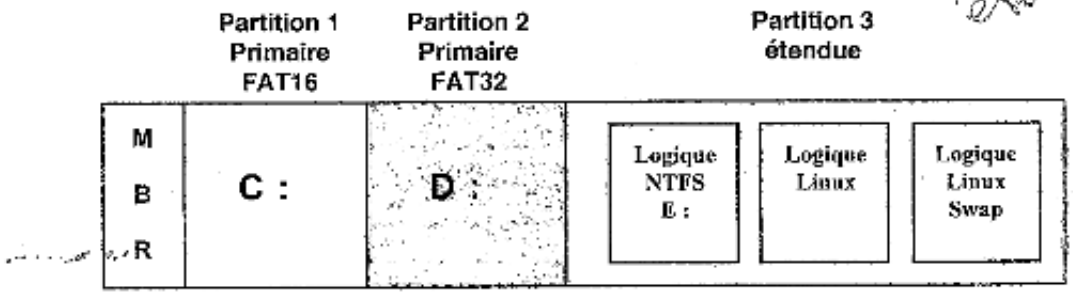


Figure 12.4 • Partitions primaires, étendues et logiques.

FIGURE 1.8 – Schéma du *MBR*

swap à la disposition du système pour la gestion de la mémoire utilisée par les processus.
Linux peut se contenter d'une seule partition pour un simple poste de travail. Dans le cas d'un serveur, on réserve en général en plus une partition pour /home et une partition pour /temp.
Dans une configuration multi-boot (Windows + Linux par exemple), des partitions sont créés pour les deux systèmes.

Exemple de partitionnement multi-boot :

Précisément, l'installation se passe en général ainsi :

— constituer un media d'installation
— configurer le BIOS pour booter à partir de ce media
— lancer la procédure de boot et procéder au partitionnement
— paramétrage du système (fuseau horaire, mot de passe, etc. . .)
— selection des paquets à installer
— installation (chargement et exécution d'une partie du noyau qui va recopier les fichiers de Linux sur disque). Dans le même temps, le système produit les fichiers de configuration adéquats pour diriger le démarrage ultérieur du système.
— création d'un media de démarrage de secours
— redémarrage du système

A titre d'exemple voici un menu correspondant à l'affectation des partitions pour une distribution de Linux, lors de la phase d'installation. On demande à l'utilisateur de choisir la taille de la partition, le type de système de fichiers, et le point de montage (c'est-à-dire le système de fichiers correspondant).

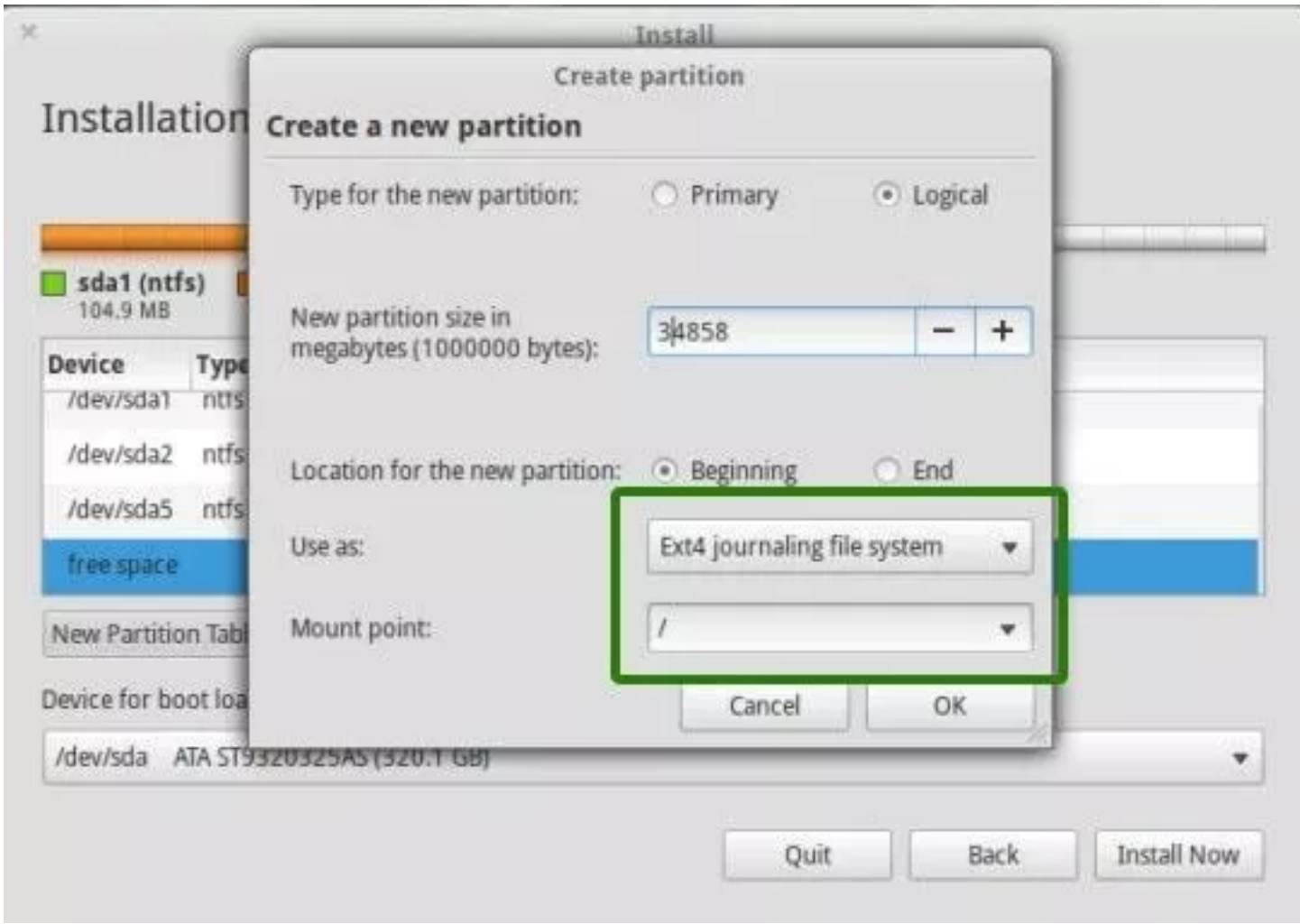


FIGURE 1.9 – Création d'une nouvelle partition

Partie 2

BIOS

2.1 BIOS

2.1.1 Definition de BIOS :

- Le BIOS est un très petit programme qui démarre lorsque vous appuyez sur le bouton de démarrage de votre PC / laptop. Il est stocké dans une puce mémoire (ROM) sur la carte mère[1].
- Le BIOS a traditionnellement 3 rôles principaux :
  - Initialisation et test du bon fonctionnement des matériels connectés à la carte mère / PC (processeur, RAM, clavier, ...). En cas de problème lors d'un test, le BIOS répond par une série de bips sonores s'il s'agit de l'absence d'un matériel nécessaire tel que la RAM ou alors simplement d'un message d'AVERTISSEMENT qui demande de régler le problème sans éteindre le PC comme par exemple l'absence de clavier[2].
  - Lancer le système d'exploitation (OS) ou le bootloader si on possède plusieures OSs dans le dispositif (flash disque, disque dure, cd-rom, ... etc)[2].
  - Créer une couche abstraite entre les matériaux (E / S) et le système d'exploitation. Cette dernière fonctionnalité n'est plus fréquente aujourd'hui. Pour éviterle temps perdu en passant par la couche BIOS, les nouveaux systèmes d'exploitation communiquent directement avec les matériels (E / S) sans passer par cette couche[2].

2.1.2 La configuration de BIOS :

Pour configure le BIOS, il existe un menu du BIOS auquel on accède par une touche du clavier réservée au démarrage du PC, généralement la touche Suppr ou F2. Une fois qu'on clique sur la touche de menu du BIOS apparaît un menu où on peut voir les différentes informations sur les matériels du PC (processeur, RAM, disques, ... etc.), changer l'ordre des périphériques analysés au lancement (cd-rom puis disque dur...).

2.1.2.1 La memoire de BIOS (CMOS) :

La mémoire de BIOS est une puce appelée CMOS (Complementary Metal Oxide Semiconductor). Cette puce stocke la configuration BIOS de l'utilisateur et le BIOS récupère cette configuration à chaque démarrage[1][3].

Pour garder la config CMOS hors alimentation, une pile longue durée alimente cette puce (CMOS) pour ne pas perdre la configuration d'utilisateur. Si la puce est endommagée, la configuration du BIOS retourne à la config par défaut, ce qui oblige à reconfigurer le BIOS a chaque démarrage (changer l'ordre des devices de boot par exemple)[3].

2.1.2.2 L'ordre de démarrage des dispositifs :

Dans le menu BIOS il existe une liste avec un ordre d'accès à différents dispositifs pouvant lancer un système d'exploitation (clé USB, CD-ROM, disque dur, ...etc.). Le BIOS recherche la séquence de démarrage (boot secteur) sur le premier dispositif et lance les instructions qui permettent de lancer le système d'exploitation dans ce dispositif. S'il ne détecte pas de boot secteur il passe au dispositif suivant[5].

FIGURE 2.1 – Puce ROM BIOS dans une carte mere

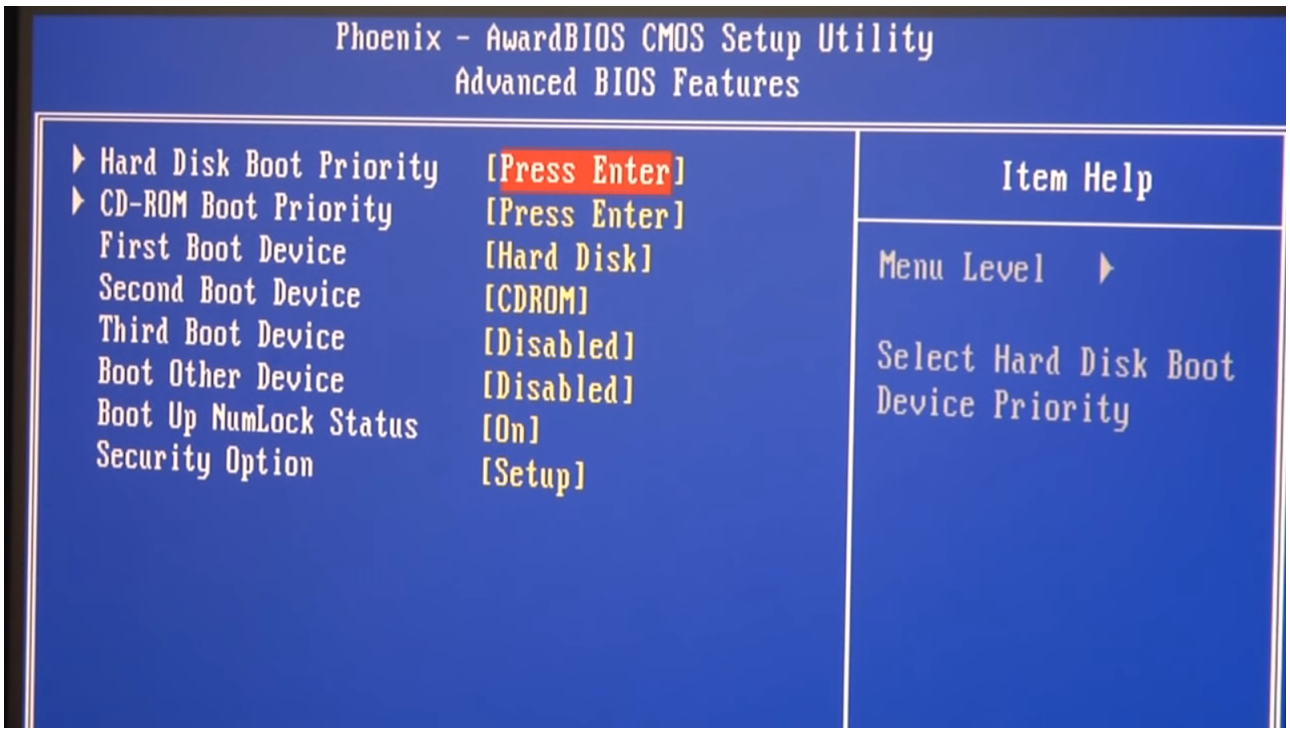


FIGURE 2.2 – La configuration d'ordre de dispositif au BIOS

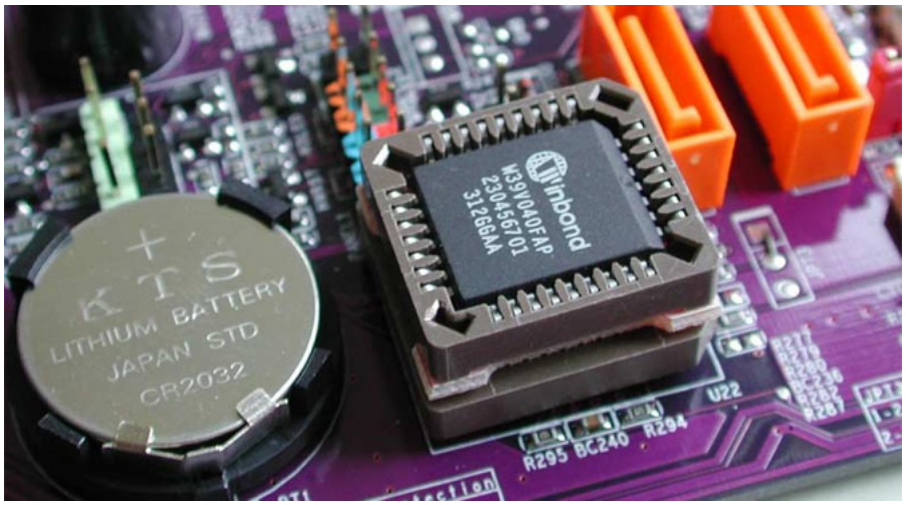


FIGURE 2.3 – Puce CMOS avec ça pile dans une carte mere

2.1.2.3 L'UEFI (BIOS++)

— L'UEFI est un BIOS mais amélioré avec plus d'avantages. Il permet un nouveau systeme de partition (la table de partition GPT) en remplacement de l'ancien systeme (la table de partition MBR)[4].

2.1.3 Les tables de partition :

— Une table de partition est une zone de données placée au début du disque où on peut trouver toutes les informations nécessaires sur les tranches de disque (partitions). Il existe deux types de tables de partition, MBR (Master Boot Record) et GPT (GUID Partition Table)[4].

2.1.3.1 Les tables de partition MBR et GPT :

— Les tables de partition MBR et GPT sont deux systèmes de partition de disque différents. On va étudier plus précisément la partition du boot sector où le BIOS / UEFI recherche les informations nécessaires pour démarrage le bootloader / OS [4].

MBR (Master Boot Record) Dans une table de partition MBR, on a le premiere secteur qui occupe 512 octets MBR. Apres, figurent la description des partitions. Voir la table présentée au dessous :

Adresse d'octet	Description	La taille (octets)
0x000	Code de boot	446
0x1BE	Partition n^{o}1	16
0x1CE	Partition n^{o}2	16
0x1DE	Partition n^{o}3	16
0x1EE	Partition n^{o}4	16
0x1FE	0x55	1
0x1FF	0xAA	1

Le code de boot se trouve dans les premiers 446 octets et permet de trouver la localisation du bootloader (NTLDR, GRUB, LILO, ... etc[6]) et de le lancer [7].

Quelques propriétés du systeme de partition MBR : - On ne peut pas avoir plus de 4 partitions principales. - Une partition ne peut pas depasser 2.2 Go - Le code de boot est limite en 446 octets.

GPT (GUID Partition Table) Dans une table de partition GPT, on trouve les zones de données suivantes : le premier secteur qui occupe 512 octets MBR en mode protective (LBA 0), GPT header (LBA 1), Partition Table (LBA 2, LBA 3, ...,), Partition Area (LBA 3, LBA 4, ... , LBA n-2) et enfin la derniere partie Backup Area (LBA n-1 et LBA n).

Note : LBA (Logical Block Addressing)

Voir la table presentée ci-dessous :

FIGURE 2.4 – Une menu d'UEFI

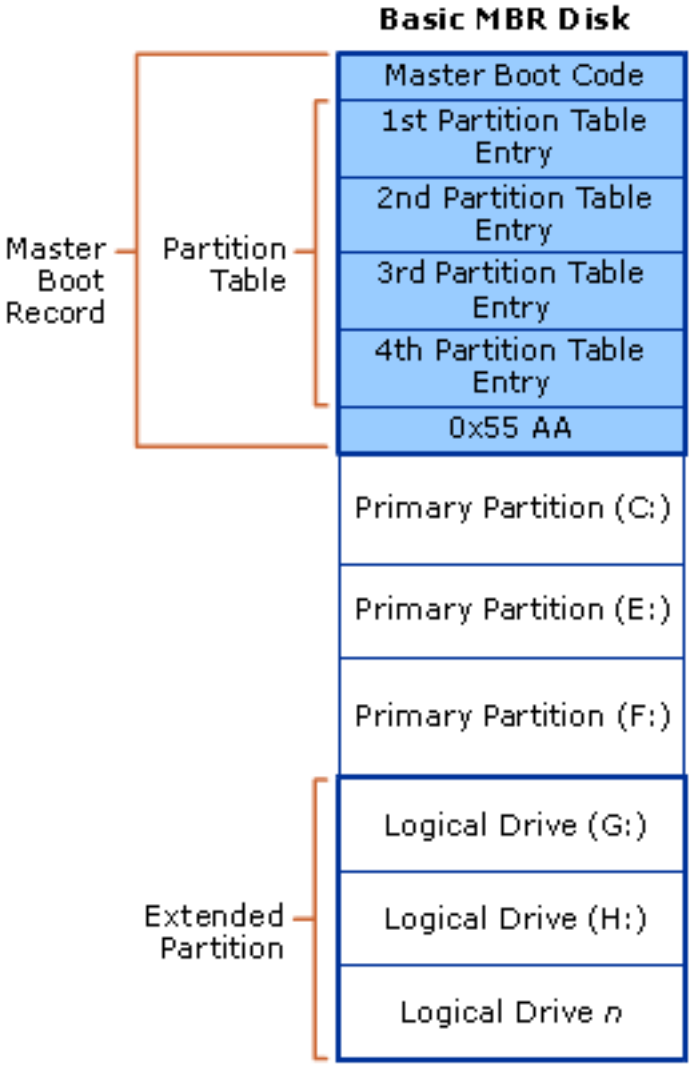


FIGURE 2.5 – Table de patition MBR

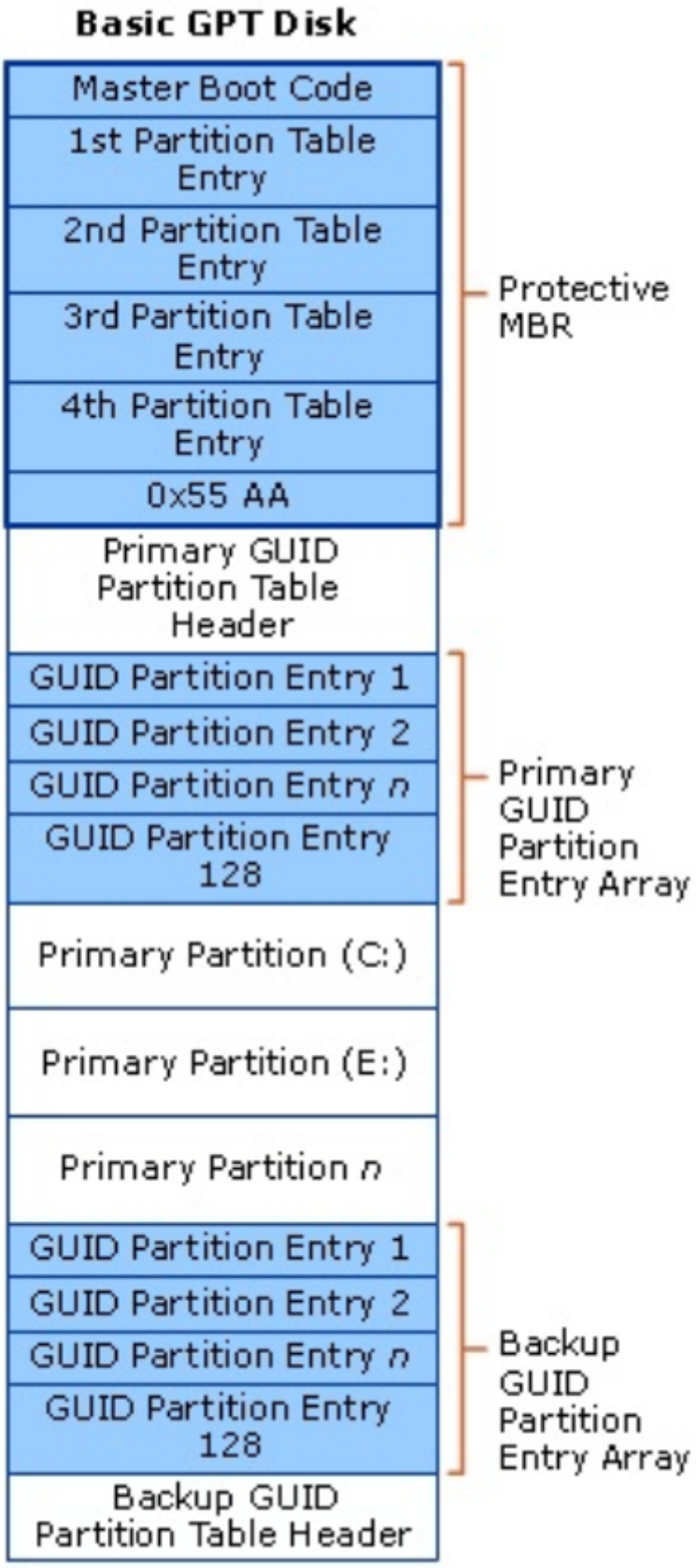


FIGURE 2.6 – Table de patition GPT

Adresse d'octet	Description	La taille (octets)
0x000	Non utilisé par l'UEFI	446
0x1EE	Une entrée pointe vers la partition EFI	64
0x1FE	0x55	1
0x1FF	0xAA	1

GUID Protective MBR :

GPT Header :

GPT Partition Entry :

La sécurité dans l'UEFI (GPT) : Pour chaque type de partition (EFI System partition, Microsoft Reserved Partition, Linux filesystem data, ... etc) on a un GUID unique qui sera verifié par l'UEFI[8].

Quelques propriétés du systeme de partition GPT : - On peut avoir jusqu'a 128 partitions. - Une paritition peut avoir jusqu'a 256 To (il suffit de créer une partition de type EFI (ESP)).

2.1.4 Le stage 1 de GRUB :

Le boot code cotenu dans le MBR pour le GRUB s'appele GRUB Stage1 MBR Code. On peut voir le code source assembleur dans le fichier boot.S qu'il etait appele stage1.S avant la version GRUB 2.

Voici une capture de code en hex.

Le code assembleur de GRUB est placé dans les 446 octets a partir de decalage *0000 :7C00*.

2.1.4.1 L'explication des instructions de stage 1 :

— Les 3 premiers octets représentent les instructions suivantes :

```
7C02 90              NOP                    ;  area to main body of code.
```

Un jump vers les instructions d'executions qui commence a partir de decalage *0000 :7C4A*.

— Les 59 octets suivants sont reserve ou BPB (BIOS Parameter Block) qui stockent des informations utilisées par la suite comme le mode de disque, la partition qui lance le stage 2 (/boot/grub), ... etc).

— On a la commande CLI (Clear Interrupt Flag) suiver par 2 NOP dans quelque version comme la version 0.94 et 0.95 comme une anticipation de changement de cette commande au future coder en 3 octets.

```
7C4A FA              CLI
```

— En suite, on un jump vers le decalage *0000 :7C00*.

```
7C4B EA507C0000      JMP      0000:7C50   ; Long Jump to the next instruction
                                          ; because some bogus BIOSes jump to
                                          ; 07C0:0000 instead of 0000:7C00.
```

— Cette partie de code a partir de décalage *0000 :7C50* jusqu'au décalage *0000 :7C83* pour afficher quelques informations sur le GRUB et faire des testes de mode pour faire un jump vers le code associe au mode trouver.

```
7C50 31C0          XOR     AX,AX
7C52 8ED8          MOV     DS,AX
7C54 8ED0          MOV     SS,AX
7C56 BC0020        MOV     SP,2000
7C59 FB            STI
7C5A A0407C        MOV     AL,[7C40]     ; <<<<<<<< Boot Drive
7C5D 3CFF          CMP     AL,FF
7C5F 7402          JZ      7C63
7C61 88C2          MOV     DL,AL
```

GPT Header				
Mnemonic	**Byte Offset**	**Byte Legnth**	**Bytes**	**Purpose**
Signature	0	8	0-7	Identifies EFI-compatible partition table header. This value must contain the ASCII string "EFI PART", encoded as the 64-bit constant 0x5452415020494645
Revision	8	4	8-11	The revision number for this header. This revision value is not related to the UEFI Specification version. This header is version 1.0, so the correct value is 0x00010000.
Headersize	12	4	12-15	Size in bytes of the GPT Header. The HeaderSize must be greater than 92 and must be less than or equal to the logical block size.
HeaderCRC32	16	4	16-19	CRC32 checksum for the GPT Header structure. This value is computed by setting this field to 0, and computing the 32-bit CRC for HeaderSize bytes.
Reserved	20	4	20-23	Must be Zero
MyLBA	24	8	24-31	The LBA that contains this data structure.
AlternateLBA	32	8	32-39	LBA address of the alternate GPT Header.
FirstUsableLBA	40	8	40-47	The first usable logical block that may be used by a partition described by a GUID Partition Entry.
LastUsableLBA	48	8	48-55	The last usable logical block that may be used by a partition described by a GUID Partition Entry.
DiskGUID	56	16	56-71	GUID that can be used to uniquely identify the disk.
PartitionEntryLBA	72	8	72-79	The starting LBA of the GUID Partition Entry array.
NumberOfPartitionEntries	80	4	80-83	The number of Partition Entries in the GUID Partition Entry array.
SizeOfPartitionEntry	84	4	84-87	The size, in bytes, of each the GUID Partition Entry structures in the GUID Partition Entry array. This field shall be set to a value of 128 x 2n where n is an integer greater than or equal to zero (e.g., 128, 256, 512, etc.). NOTE: Previous versions of this specification allowed any multiple of 8..
PartitionEntryArrayCRC32	88	4	88-92	The CRC32 of the GUID Partition Entry array. Starts at PartitionEntryLBA and is computed over a byte length of NumberOfPartitionEntries * SizeOfPartitionEntry.
Reserved	92	Blocksize-92	92-	The rest of the block is reserved by UEFI andmust be zero.

FIGURE 2.7 – GPT Header

GPT Partition Entry				
Mnemonic	**Byte Offset**	**Byte Legnth**	**Bytes**	**Purpose**
PartitionTypeGUID	0	16	0-15	Unique ID that defines the purpose and type of this Partition. A value of zero defines that this partition entry is not being used.
UniquePartitionGUID	16	16	16	GUID that is unique for every partition entry. Every partition ever created will have a unique GUID. This GUID must be assigned when the GPT Partition Entry is created. The GPT Partition Entry is created whenever the NumberOfPartitionEntries in the GPT Header is increased to include a larger rangeof addresses.
StartingLBA	32	8	32-39	Starting LBA of the partition defined by this entry.
EndingLBA	40	8	40-47	Ending LBA of the partition defined by this entry.
Attributes	48	8	48-55	Attribute bits, all bits reserved by UEFI
PartitionName	56	72	56-127	Null-terminated string containing a human-readable name of the partition.
Reserved	128	SizeOfPartition Entry - 128	32-39	The rest of the GPT Partition Entry, if any, is reserved by UEFI and must be zero.

FIGURE 2.8 – Partition Entry

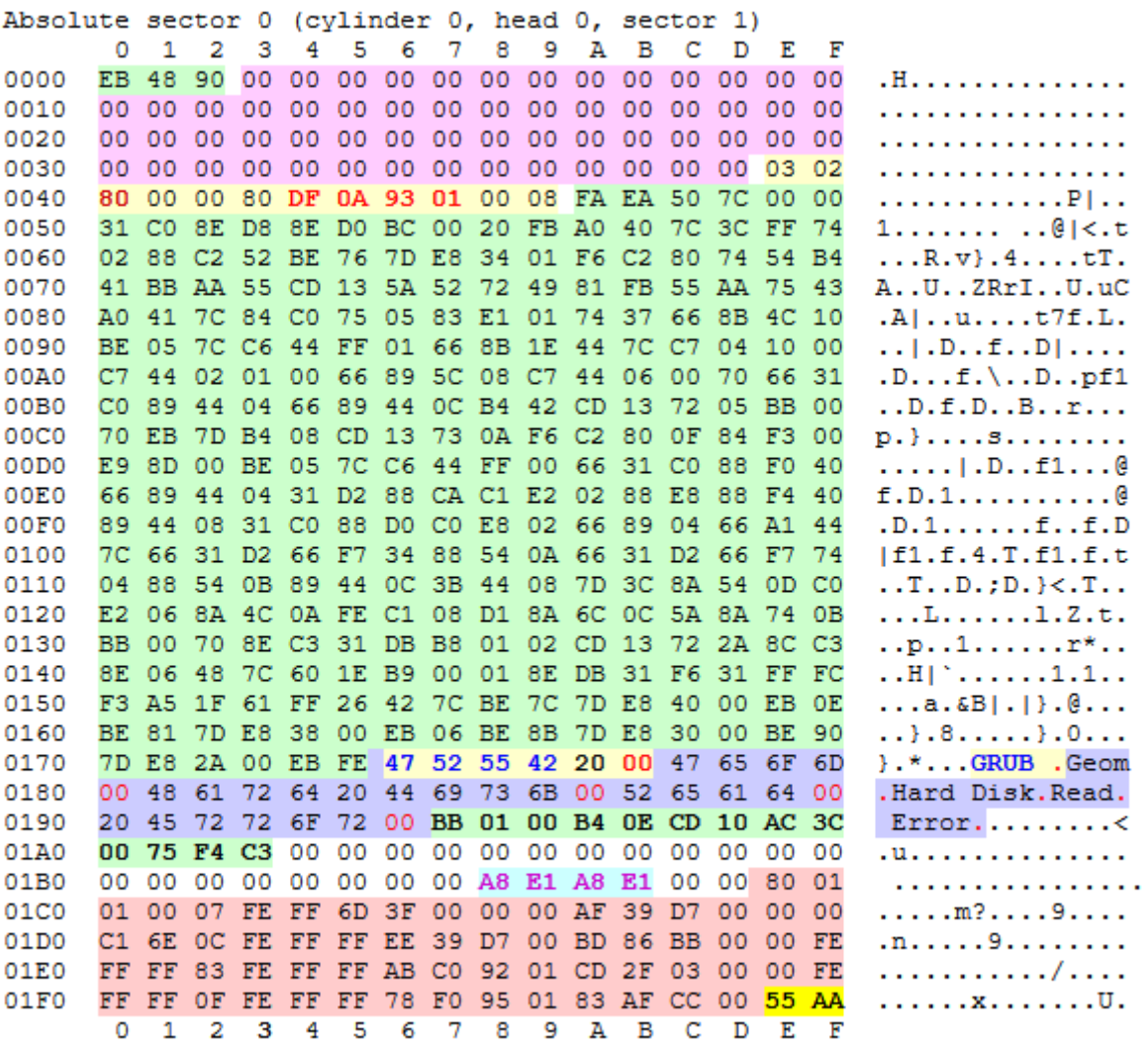


FIGURE 2.9 – Capture de boot code GRUB 1 v0.93[6]

```
7C63 52           PUSH    DX
7C64 BE767D       MOV     SI,7D76       ; --> "GRUB "
7C67 E83401       CALL    7D9E          ; Display GRUB ID on screen.

7C6A F6C280       TEST    DL,80
7C6D 7454         JZ      7CC3
7C6F B441         MOV     AH,41         ; Function 41h of INT13
7C71 BBAA55       MOV     BX,55AA
7C74 CD13         INT     13            ; Test for INT13 Extensions
7C76 5A           POP     DX
7C77 52           PUSH    DX
7C78 7249         JC      7CC3          ; If CF = 1, something wrong
                                        ; with test, so use CHS Mode.
7C7A 81FB55AA     CMP     BX,AA55
7C7E 7543         JNE     7CC3          ; If ZF = 0, INT13 Extensions
                                        ;   failed, so use CHS Mode.

7C80 A0417C       MOV     AL,[7C41]     ; <<<< Force LBA mode byte
7C83 84C0         TEST    AL,AL

7C85 7505         JNZ     7C8C
7C87 83E101       AND     CX,+01
7C8A 7437         JZ      7CC3
```

— Les instructions suivante sont pour le mode LBA (accès linéaire à tous les secteurs du disque dur, en partant de zéro)[9].

```
7C8C 668B4C10   * MOV     ECX,[SI+10]

7C90 BE057C       MOV     SI,7C05         ; <<<<<< Setup "Disk Packet"
                                                   for Extended Read.
7C93 C644FF01     MOV     BYTE PTR [SI-01],01

7C97 668B1E447C * MOV     EBX,[7C44]      ; <<<<<<< Location of Stage2
                                                code from the beginning of
                                                the partition (the offset
                                                is in number of sectors).

7C9C C7041000     MOV     WORD PTR [SI],0010
7CA0 C744020100   MOV     WORD PTR [SI+02],0001

7CA5 66895C08   * MOV     [SI+08],EBX

7CA9 C744060070   MOV     WORD PTR [SI+06],7000

7CAE 6631C0     * XOR     EAX,EAX

7CB1 894404       MOV     [SI+04],AX

7CB4 6689440C   * MOV     [SI+0C],EAX

7CB8 B442         MOV     AH,42           ; Function 42h of INT13
7CBA CD13         INT     13              ; Extended Read (using
                                          ; Disk Address Packet).
7CBC 7205         JC      7CC3            ; If LBA not supported,
                                          ; go to CHS mode only.
```

```
7CBE BB0070          MOV     BX,7000
7CC1 EB7D            JMP     7D40
```

— Les instructions suivante sont pour le mode CHS (Cylindres/Têtes/Secteur, anciens mode d'accès aux disques durs)[9].

```
7CC3 B408            MOV     AH,08              ; Function 08 of INT13
7CC5 CD13            INT     13                 ; Get Drive Parameters

7CC7 730A            JNB     7CD3

7CC9 F6C280          TEST    DL,80              ; Tests if HDD exists.
7CCC 0F84F300    *   JZ      7DC3               ; Therefore, this jump is
; never taken unless grub was installed on and running from a floppy
; disk.  And only then will you find more executable code at 7DC3.
; In the source code file (stage1.S), you'll find this short comment
; about the extra code: "Kinda sneaky, huh?"

7CD0 E98D00          JMP     7D60               ; There was an HDD Error!

7CD3 BE057C          MOV     SI,7C05              <<<<<< "Disk Packet"
7CD6 C644FF00        MOV     BYTE PTR [SI-01],00
7CDA 6631C0      *   XOR     EAX,EAX

; Save number of heads:
7CDD 88F0            MOV     AL,DH
7CDF 40              INC     AX
7CE0 66894404    *   MOV     [SI+04],EAX
7CE4 31D2            XOR     DX,DX
7CE6 88CA            MOV     DL,CL
7CE8 C1E202      *   SHL     DX,02
7CEB 88E8            MOV     AL,CH
7CED 88F4            MOV     AH,DH

; Save number of cylinders:
7CEF 40              INC     AX
7CF0 894408          MOV     [SI+08],AX
7CF3 31C0            XOR     AX,AX
7CF5 88D0            MOV     AL,DL
7CF7 C0E802      *   SHR     AL,02

; Save number of sectors:
7CFA 668904      *   MOV     [SI],EAX

7CFD 66A1447C    *   MOV     EAX,[7C44]       <<<<<< Location of Stage2 code
                                                     from the beginning of
                                                 the partition (the offset
                                                 is in number of sectors).
7D01 6631D2      *   XOR     EDX,EDX
7D04 66F734      *   DIV     WORD PTR [SI]          ; Double word here.

7D07 88540A          MOV     [SI+0A],DL

7D0A 6631D2      *   XOR     EDX,EDX
7D0D 66F77404    *   DIV     WORD PTR [SI+04]       ; Double word here.

7D11 88540B          MOV     [SI+0B],DL
7D14 89440C          MOV     [SI+0C],AX
```

```
7D17 3B4408          CMP     AX,[SI+08]
7D1A 7D3C            JGE     7D58            ; There was a Geometry Error!
7D1C 8A540D          MOV     DL,[SI+0D]

7D1F C0E206      *   SHL     DL,06

7D22 8A4C0A          MOV     CL,[SI+0A]
7D25 FEC1            INC     CL
7D27 08D1            OR      CL,DL
7D29 8A6C0C          MOV     CH,[SI+0C]
7D2C 5A              POP     DX
7D2D 8A740B          MOV     DH,[SI+0B]
7D30 BB0070          MOV     BX,7000
7D33 8EC3            MOV     ES,BX
7D35 31DB            XOR     BX,BX
7D37 B80102          MOV     AX,0201          ; Function 02 of INT13
7D3A CD13            INT     13               ; Read 1 sector into Memory

7D3C 722A            JB      7D68             ; There was a Read Error!

7D3E 8CC3            MOV     BX,ES
```

— Ici les instructions qui permettent a passe au stage 2 de GRUB (*0000 :7D53*).

```
7D40 8E06487C        MOV     ES,[7C48]        ; <<<<<<<< WORD [0800 hex]
                                              ; Note: 800:0000 = 0000:8000
7D44 60          *   PUSHA

7D45 1E              PUSH    DS
7D46 B90001          MOV     CX,0100
7D49 8EDB            MOV     DS,BX
7D4B 31F6            XOR     SI,SI
7D4D 31FF            XOR     DI,DI
7D4F FC              CLD
7D50 F3A5            REP     MOVSW

7D52 1F              POP     DS
7D53 61          *   POPA

7D54 FF26427C        JMP     [7C42]           ; WORD    <<< 8000 hex.
                                              ; "stage2_address"
```

— Cette partie et pour les fonctions qui affichent les differents erreurs rencontrer durant l'execution s'il y en a des erreurs.

```
7D58 BE7C7D          MOV     SI,7D7C          ; --> "Geom Error"
7D5B E84000          CALL    7D9E             ; Display it on screen.
7D5E EB0E            JMP     7D6E             ; Finish it and 'lock-up'

7D60 BE817D          MOV     SI,7D81          ; --> "Hard Disk Error"
7D63 E83800          CALL    7D9E             ; Display it on screen.
7D66 EB06            JMP     7D6E             ; Finish it and 'lock-up'

7D68 BE8B7D          MOV     SI,7D8B          ; --> "Read Error"
7D6B E83000          CALL    7D9E             ; Display it on screen.
7D6E BE907D          MOV     SI,7D90          ; (For displaying " Error")
7D71 E82A00          CALL    7D9E             ; Finish it and 'lock-up'
7D74 EBFE            JMP     7D74             ; Locks-up execution in an
```

```
                                        ; infinite loop!  You must
                                        ;   reboot your computer!
```

— L'emplacement de messages d'erreurs.

```
                      6  7  8  9  A  B  C  D  E  F
7D76                 47 52 55 42 20 00 47 65 6F 6D         GRUB .Geom
7D80  00 48 61 72 64 20 44 69 73 6B 00 52 65 61 64 00   .Hard Disk.Read.
7D90  20 45 72 72 6F 72 00                               Error.
       0  1  2  3  4  5  6
```

— Une boucle qui parcourt les caractères pour afficher un message.

```
7D97 BB0100        MOV     BX,0001
7D9A B40E          MOV     AH,0E          ; Function 0Eh of INT 10h
7D9C CD10          INT     10             ; Display the character

7D9E AC            LODSB
7D9F 3C00          CMP     AL,00
7DA1 75F4          JNZ     7D97           ; Loop until finding a zero byte.
7DA3 C3            RET
```

— L'emplacement des partitions physiques de MBR et la signature de la fin de MBR 0x55AA.

```
                                                    E  F
7DBE                                               80 01   ................
7DC0   01 00 07 FE FF 6D 3F 00 00 00 AF 39 D7 00 00 00   .....m?....9....
7DD0   C1 6E 0C FE FF FF EE 39 D7 00 BD 86 BB 00 00 FE   .n.....9........
7DE0   FF FF 83 FE FF FF AB C0 92 01 CD 2F 03 00 00 FE   ............./....
7DF0   FF FF 0F FE FF FF 78 F0 95 01 83 AF CC 00 55 AA   ......x.......U.
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
```

2.1.5 Références :

[1] Le BIOS (Basic Input Output System)
[2] Why Do Computer Devices Need a BIOS?
[3] C'est quoi les BIOS, CMOS, ROM ?
[4] Les tables de partitionnement MBR et GPT
[5] How BIOS works
[6] All the Details of many versions of both MBR (Master Boot Records) and OS Boot Sectors (also called: Volume Boot Records)
[7] CEIC 2013 UEFI, MBR and GPT oh my!
[8] GUID Partition Table
[9] Structure du MBR et des tables des partitions sur le disque dur

Partie 3

GRUB

3.1 Intro

GRUB (*GRand Unified Bootloader*) est un bootloader. Son rôle est de charger le système d'exploitation à démarrer.

Pour pouvoir démarrer correctement, les systèmes d'exploitations ont besoin de connaître certaines informations sur l'environnement sur lequel ils s'exécutent. Avant de passer la main au système d'exploitation, *GRUB* doit donc mettre en place des données.

Ce mécanisme s'appelle la *relocation*. Son fonctionnement est spécifique à un système d'exploitation. *GRUB* possède donc des fonctions de *relocation* spécifique pour chaque système qu'il supporte

3.2 Relocation

3.2.1 Définition

La *relocation* est le processus qui est chargé de déplacer du code d'une zone mémoire à une autre afin que celui ci soit accessible par un autre programme.

3.2.2 Dans *GRUB*

Dans *GRUB*, la *relocation* désigne plus spécifiquement le fait de mettre dans des zones mémoires spécifique des données nécessaires pour que le système d'exploitation puisse démarrer correctement. Par exemple, lors du démarrage de *Linux*, *GRUB* déplacera, entre autres, des informations sur le *firmware* (*BIOS* ou *UEFI*), le *bootloader* (*GRUB*) et sur les périphériques vidéos. Ce déplacement permettra à *Linux* d'accéder à ces données. Le reste des données chargées en mémoire par *GRUB* seront écrasées par *Linux*

3.2.3 Le code

Le processus de *relocation* gère des blocs mémoire. Il est indépendant de la mémoire à déplacer. Les informations sur la mémoire à déplacer est stocké dans des structures.

3.2.3.1 `struct grub_relocator`

Cette structure contient les informations générales sur les données à déplacer. La mémoire est séparé en *chunk*. Ces *chunks* possèdent les informations précises sur les déplacements à effectuer. La structure `grub_relocator` contient une liste chaînée (`chunk`) de *chunk* à déplacer. De plus elle contient des informations générales comme la taille d'un *chunk* (`relocators_size`) et des adresses, comme par exemple l'adresse la plus haut à laquelle on peut déplacer de la mémoire (`highestaddr`).

```
struct grub_relocator
{
  struct grub_relocator_chunk *chunks;
  grub_phys_addr_t postchunks;
  grub_phys_addr_t highestaddr;
```

```
  grub_phys_addr_t highestnonpostaddr;
  grub_size_t relocators_size;
};
```

3.2.3.2 chunk

Un *chunk* (`grub_relocator_chunk`) contient la mémoire à déplacer (`srcv`), l'adresse de départ (`src`) et d'arrivée (`target`) de la mémoire et la taille de la mémoire (`size`).

```
struct grub_relocator_chunk
{
  struct grub_relocator_chunk *next;
  grub_phys_addr_t src;
  void *srcv;
  grub_phys_addr_t target;
  grub_size_t size;
  struct grub_relocator_subchunk *subchunks;
  unsigned nsubchunks;
};
```

3.3 grub_linux_boot

La fonction `grub_linux_boot` met en place les données à transmettre à *Linux*. Ensuite, elle initialise le *relocator*. Enfin, elle appelle la fonction `grub_relocator32_boot` afin de déplacer la mémoire là où *Linux* sait comment y accéder.

3.4 grub_relocator*_boot

Le processus spécifique de *relocation* dépend de l'architecture. Le processus général consiste en parcours des *chunk* de mémoire. Chaque *chunk* est déplacé à l'adresse `target`. Cette adresse est prédéfini. En effet, le système d'exploitation va lire ces données et doit donc savoir où elles sont positionnées.

En plus des données utilisées par l'OS, *GRUB* mets en mémoire le noyaux sur lequel démarrer.

Une fois le processus de *relocation* fini, *GRUB* va sauter au début du noyau pour lui donner la main. Pour cela, *GRUB* génère les instructions *CPU* à exécuter sous forme d'un tableau d'entier :

```
void
grub_cpu_relocator_jumper (void *rels, grub_addr_t addr)
{
  grub_uint8_t *ptr;
  ptr = rels;
#ifdef __x86_64__
  /* movq imm64, %rax (for relocator) */
  *(grub_uint8_t *) ptr = 0x48;
  ptr++;
  *(grub_uint8_t *) ptr = 0xb8;
  ptr++;
  *(grub_uint64_t *) ptr = addr;
  ptr += sizeof (grub_uint64_t);
#else
  /* movl imm32, %eax (for relocator) */
  *(grub_uint8_t *) ptr = 0xb8;
  ptr++;
  *(grub_uint32_t *) ptr = addr;
  ptr += sizeof (grub_uint32_t);
#endif
  /* jmp $eax/$rax */
  *(grub_uint8_t *) ptr = 0xff;
```

```
  ptr++;
  *(grub_uint8_t *) ptr = 0xe0;
  ptr++;
}
```

Puis il exécute ce tableau :

```
  // relst is an array of opcode (represented as an array of u8)
  ((void (*) (void)) relst) ();
```

3.5 Conclusion

Le mécanisme de *relocation* utilisé par *GRUB* en lui même est simple. Il déplace une partie de la mémoire à une adresse donnée. Les données à déplacer dépendent du système d'exploitation à lancer. Par exemple, *Linux* suis une spécification (*Linux multiboot specification*) pour le processus de démarrage. Cette spécification définie les données que le *bootloader* doit transmettre et où il doit les positionner. *Windows* quand à lui n'a pas de spécification libre d'accès équivalente. De plus, le code source du système n'est pas accessible. *GRUB* ne peut donc pas démarrer directement *Windows*. C'est pour ça qu'il doit passer par le *bootloader* de l'OS.

Partie 4

Setup Part

4.1 From bootloader to kernel

Dans un premier temps, on "allume la lumière" : * On aligne les *segment registers* (code segment, stack segment, data segment...) de manière à pouvoir les initialiser la pile, la section .bss... * On installe la pile, en commençant par vérifier que le *stack segment* est correct (ie pointe sur 0x1000) puis en récupérant le *stack pointer* fourni par le bootloader. * On vérifie qu'il n'y a pas eu de problème dans l'installation des *segment registers* et de la pile, puis on crée la section .bss dans la pile (où sont stockées les données allouées mais non initialisées du code C). * On saute vers le *main* de l'installeur du noyau.

4.2 Environnement d'exécution

Dans un second temps, on met en place un environnement minimal pour l'exécution de code C. * On enregistre les paramètres de boot passés par le bootloader sur la "zero page", les adresses mémoires au tout début de l'espace d'adressage. * On initialise une première console série fournissant des fonctions élémentaires : putchar, puts, gettime... Voir le code dans arch/X/boot/tty.c * Disposant déjà de la pile et de la section .bss, on initialise le tas, qui récupère autant de mémoire que la pile en laisse. * Vérifie le *privilege level* du CPU (0 pour exécuter le code du noyau) et vérifie les flags du CPU (interruptions, etc). * Détecte la mémoire (récupère un pointeur sur la mémoire disponible). * Initialisation du clavier. * Lancement du mode vidéo du kernel : on vérifie qu'il y a la place sur le tas pour le charger, et si oui on s'interface avec le matériel. Il ne s'agit pas pour le moment d'interfaces graphiques, mais d'afficher du texte.

4.3 Transition vers le mode protégé

Dans un troisième temps, on passe du *real mode* (où tous les programmes peuvent accéder à toute la mémoire) au *protected mode* (où le noyau... "segmente" les programmes, causant une *segmentation fault* en cas de violation). Cette transition se déroule en plusieurs moments : * On désactive les interruptions (pour pouvoir installer la table d'interruptions). * On active l'*A20 gate*, qui permet au processeur de communiquer avec la mémoire additionnelle (les barettes de RAM). * On désactive les coprocesseurs. * On installe la table d'interruptions. * On installe la GDT (global descriptor table), la mémoire virtuelle. * On "saute" en *protected mode*. * On chaîne les changements de mode (par exemple vers le *64 bits mode*).

4.4 Décompression du kernel

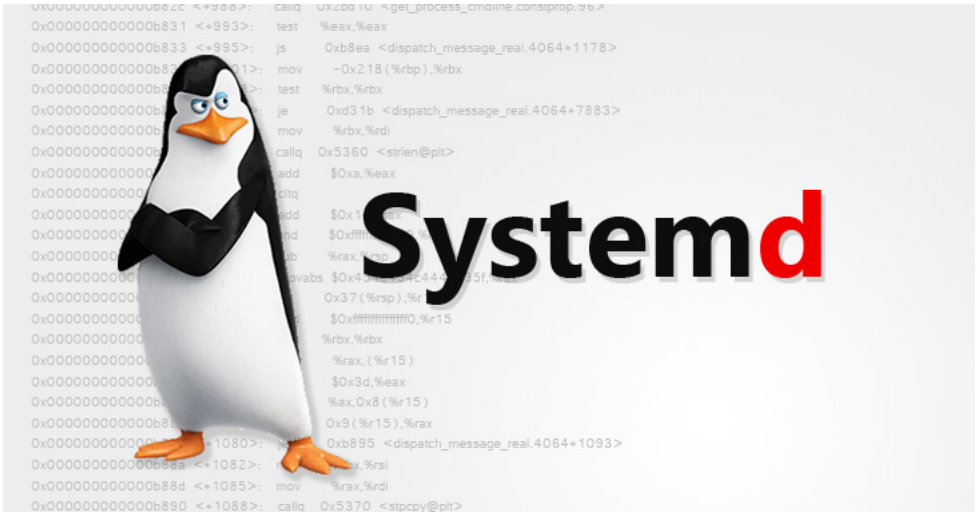
Dans un quatrième temps, on organise la décompression du Kernel : * On calcule l'adresse où le kernel sera déplacé pour être décompressé. * On installe la section .text et on y charge le code de décompression. * On installe la section .rodata à l'adresse calculée et on y déplace le kernel compressé. * On nettoie la section .bss. * On récupère les paramètres de boot. * On réinitialise la console pour le *protected mode*. * On récupère des pointeurs vers la fin et vers le début du tas. * On choisit (aléatoirement, pour des raisons de sécurité) l'adresse où le kernel sera décompressé. * On décompresse le kernel. * Une fois que le kernel est décompressé, on calcule son emplacement définitif en mémoire et on l'y déplace. * On saute à cette adresse et le kernel prend la main...

4.5 Résumé

— Environnement d'éxecution C (pile, console minimale)
— Reconnaissance du matériel et mémoire virtuelle
— Transition vers le mode protégé
— Relocalisation et décompression du noyau
https ://0xax.gitbooks.io/linux-insides/content/Booting/

Partie 5

Init



Dans ce chapitre est introduit la gestion des services lors du démarrage du système , avec pour objectif d'améliorer la compréhension du fonctionnement de cette phase d'initialisation sous *GNU/Linux* dans un système utilisant **systemd**.

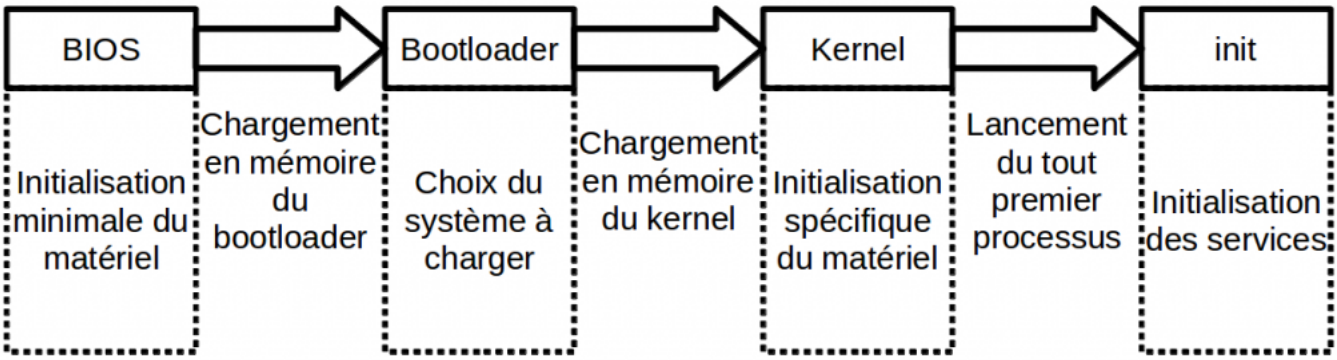
5.1 Systemd

5.1.1 Introduction

Systemd est le processus qui gère tous les services (on parle de processus init).

5.1.1.1 Contexte

Le démarrage de *Linux* sur un PC peut se résumer en 4 étapes :



Le processus init se charge de lancer les services (daemons) et de mettre en place un environnement pour l'utilisateur (graphique, réseau, etc). Sa dernière action consiste généralement à présenter un écran d'authentification, laissant la main à l'utilisateur.

Qu'est-ce qu'un service ?

Un service est un programme qui est exécuté en tache de fond. C'est-à-dire qu'il effectue son travail sans interaction directe avec l'utilisateur. Certains services sont des composants essentiels du système. Un service peut être nommé démon ou daemon.

Qu'est ce qu'un daemon ?

Un daemon est un processus qui tourne en arrière plan et qui supervise le système ou bien fournit des fonctionnalités à d'autres processus.

Le processus init a un status particulier ;
— Il est le tout premier processus appelé par le kernel (PID = 1).
— Il est le parent de tous les processus qui deviennent orphelins.
— Il reste vivant jusqu'à l'arrêt du système.
— À l'arrêt du système, il est chargé de fermer tous les services et de quitter proprement.

5.1.1.2 Les différents processus init

Il existe différents processus init, dont les plus connus sont :
— *SysVinit* : processus init historique, hérité de *UNIX System V*.
— *Upstart* : sorti en 2006, à l'origine pour *Ubuntu*.
— **Systemd** : sorti en 2010.

SysVinit et *Upstart* ont une notion de niveau d'exécution (*RunLevel*) tandis que **systemd** a une notion de cible. À un niveau d'exécution ou une cible correspond un mode de fonctionnement du système dans lequel certains services sont arrêtés tandis que d'autres sont lancés.

Le tableau ci-dessous décrit les différents modes de fonctionnement standard sous *Debian* :

Niveaux d'exécution *SysVinit* / *Upstart*	Cibles **systemd**	Commentaires
0	runlevel0.target, poweroff.target	Arrêt du système
1,s,single	runlevel1.target, rescue.target	Mode utilisateur unique / mode maintenance (Single User Mode) en ligne de commande
2-4	runlevel2.target, runlevel3.target, runlevel4.target, multi-user.target	Mode multi-utilisateur non graphique. Peut être adapté aux besoins
5	runlevel5.target, graphical.target	Mode multi-utilisateur graphique
6	runlevel6.target, reboot.target	Redémarrage du système
emergency	emergency.target	Shell d'urgence

Obtenir la cible d'exécution par défaut `$ systemctl get-default`

Changer la cible d'exécution par défaut `$ systemctl set-default CIBLE.target`

Changer la cible d'exécution immédiatement `$ systemctl isolate CIBLE.target`

Exemple : `$ systemctl isolate multi-user.target` ou `$ systemctl isolate runlevel3.target` pour passer en mode multi-utilisateur non graphique.

Systemd est destiné à remplacer SysVinit et Upstart sur toutes les grandes distributions (Debian, Ubuntu, Fedora, ...).

5.1.1.3 Les services

Certains services sont lancés par défaut au démarrage du système. Il s'agit d'un choix des concepteurs de la distribution. Ce choix concerne des services essentiels et d'autres qui le sont moins.

Vous pouvez choisir de désactiver certains services qui ne sont pas essentiels pour vous. Vous pouvez aussi choisir d'arrêter momentanément des services. Par exemple, pour voir son impact, avant de les désactiver complétement. Vous pouvez activer des services qui ne sont pas actifs par défaut, mais dont, vous, vous avez besoin. Et enfin, vous pouvez lancer certains services momentanément sur besoin ponctuel.

Intuitivement, plus il y a de services qui sont lancés au démarrage, plus cela prend du temps pour que le système devienne fonctionnel. Il s'agit d'ailleurs d'une des principales motivations de la naissance de *Upstart* et **systemd**.

Systemd dispose d'une notion d'unité. Les services sont un type d'unité. Sans entrer dans les détails, systemd contrôle tout grâce aux unités.

5.1.2 Les unités de systemd

5.1.2.1 Qu'est-ce qu'une unité ?

Les unités sont les objets que **systemd** utilise pour initialiser le système. Il s'agit d'une représentation normalisée sous forme de fichier des ressources système pouvant être gérées par un ensemble de processus démons et manipulées par les utilitaires fournis par **systemd**, notamment la commande `systemctl`.

D'une certaine manière, les unités peuvent être assimilées à des services ou à des tâches d'autres systèmes d'initialisation. Cependant, une unité a une définition beaucoup plus large, car elle peut être utilisée pour résumer des services, des ressources réseau, des périphériques, des montages de système de fichiers et des pools de ressources isolés.

Des idées gérées dans d'autres systèmes init avec une définition de service unifiée sont de fait divisées grâce à ces unités. Cela organise et sépare distinctement certaines tâches et permet ainsi d'activer, de désactiver ou d'étendre facilement des fonctionnalités sans modifier le comportement de base d'une unité.

5.1.2.2 L'apport des unités

Certaines fonctionnalités que les unités peuvent implémenter facilement sont :

- Activation basée sur les sockets : les sockets associées à un service sont dissociées du processus démon lui-même afin d'être traités séparément. Cela offre un certain nombre d'avantages, tels que le fait de retarder le début d'un service jusqu'à ce que la socket associée soit d'abord utilisée. Cela permet également au système de créer toutes les sockets au début du processus de démarrage, ce qui permet de démarrer les services associés en parallèle.
- Activation basée sur le bus : les unités peuvent également être activées sur l'interface de bus fournie par D-Bus. Une unité peut être démarrée lorsqu'un bus associé est lancé.
- Activation basée sur un chemin : une unité peut être démarrée en fonction de l'activité ou de la disponibilité de certains chemins du système de fichiers.
- Activation basée sur un périphérique : les unités peuvent également être démarrées dès la première disponibilité du matériel associé en exploitant les événements udev.
- Instances et modèles : les fichiers d'unité de modèle peuvent être utilisés pour créer plusieurs instances de la même unité générale. Cela permet de légères variations ou des unités sœurs qui remplissent toutes la même fonction générale.
- Gestion facile de la sécurité : Les unités peuvent implémenter des fonctionnalités de sécurité relativement fiables en ajoutant des directives simples. Par exemple, vous pouvez spécifier un accès non lu ou en lecture seule à une partie du système de fichiers, limiter les capacités du noyau et attribuer un accès réseau privé.
- Extraits : les unités peuvent facilement être étendues en fournissant des extraits qui remplaceront des parties du fichier unité du système. Cela facilite le basculement entre les implémentations vanilla et les unités personnalisées.

Les unités **systemd** présentent de nombreux autres avantages par rapport aux fonctionnalités des autres systèmes d'initialisation, mais ce tour d'horizon donne une idée de la puissance de ces fichiers de configurations.

5.1.3 L'emplacement des fichiers unités

Les fichiers qui définissent la manière dont systemd gérera une unité peuvent être trouvés dans plusieurs emplacements différents, chacun ayant une priorité et un rôle particulier.

La copie système des fichiers d'unité est généralement conservée dans le répertoire '/lib/systemd/system". Lorsqu'un logiciel installe des fichiers d'unité sur le système, il s'agit de l'emplacement où ils sont placés par défaut.

Les fichiers d'unités stockés ici peuvent être démarrés et arrêtés à la demande pendant une session. Ce sera le fichier d'unité générique vanilla, souvent écrit par les responsables du projet en amont, qui devrait fonctionner sur tout système qui déploie systemd dans son implémentation standard. Vous ne devriez pas éditer les fichiers dans ce répertoire.

Si vous souhaitez modifier le fonctionnement d'une unité, le meilleur emplacement pour le faire est situé dans le répertoire `/etc/systemd/system`. Les fichiers d'unité trouvés dans ce répertoire ont priorité sur les autres emplacements du système de fichiers. Si vous souhaiter modifier le comportement d'un fichier d'unité, placer votre version dans ce répertoire est le moyen le plus sûr et le plus flexible de procéder.

Si vous souhaitez remplacer uniquement des directives spécifiques du fichier unité du système, vous pouvez d'ailleurs fournir des extraits de fichier unité dans un sous-répertoire. Ceux-ci étendront ou modifieront dynamiquement les directives de la copie du système, vous permettant de spécifier uniquement les options que vous souhaitez apporter dans des fichiers séparés. La bonne façon de faire est de créer un répertoire nommé d'après le fichier d'unité correspondant avec .d ajouté à la fin. Ainsi, pour une unité appelée `exemple.service`, un sous-répertoire appelé `exemple.service.d` devra être créé. Dans ce répertoire, un fichier se terminant par .conf peut être utilisé pour remplacer ou étendre les attributs du fichier unité du système.

Il existe également un emplacement pour les définitions d'unités à l'exécution dans `/run/systemd/system`. Les fichiers d'unité trouvés dans ce répertoire ont une priorité moindre que ceux de `/etc/systemd/system` mais plus importante que ceux de `/lib/systemd/system`. Le processus **systemd** lui-même utilise cet emplacement pour les fichiers d'unités créés de manière dynamique et créés au moment de l'exécution. Ce répertoire peut être utilisé pour modifier le comportement de l'unité du système pendant la durée de la session. Toutes les modifications effectuées dans ce répertoire seront perdues lors du redémarrage du système.

Liste des unités en cours d'exécution ou exécutés depuis le démarrage du système `$ systemctl list-units`

Liste des unités installés sur le système `$ systemctl list-unit-files`

toutes les unités situées dans `/lib/systemd/system/`

Liste des unités de type service `$ systemctl list-units -t service`

Temps de démarrage des différents services `$ systemd-analyze blame` >Il est ainsi possible de repérer les services très ou trop long à se lancer. Et vous obtenez la liste des services lancés au démarrage classés de celui qui a pris le plus de temps à celui en a pris le moins.

5.1.4 Les types d'unités

Systemd classe les unités en fonction du type de ressource qu'elles décrivent. Le moyen le plus simple de déterminer le type d'une unité est d'utiliser l'extention du fichier associé. La liste suivante décrit les types d'unités disponibles pour **systemd** :

Nom du type d'unité	Description
service	décrit comment gérer un service ou une application sur le serveur. Cela inclura comment démarrer ou arrêter le service, dans quelles circonstances il devrait être démarré automatiquement, ainsi que les informations de dépendance et de commande des logiciels associés
socket	décrit une socket réseau ou IPC, ou un tampon FIFO utilisé par systemd pour l'activation basée sur les sockets. Celles-ci ont toujours un fichier .service associé qui sera démarré lorsqu'une activité sera vue sur la socket définie par cette unité. Par exemple, un accès à la socket nscd.socket va démarrer le service nscd.service
device	décrit un périphérique désigné comme nécessitant une gestion systemd par udev ou le système de fichiers sysfs. Tous les appareils n'auront pas obligatoirement de fichiers .device
mount	définit un point de montage sur le système à gérer par systemd. Ceux-ci sont nommés d'après le chemin de montage, les barres obliques étant modifiées en tirets
automount	configure un point de montage qui sera automatiquement monté. Ceux-ci doivent être nommés d'après le point de montage auquel ils font référence et doivent avoir une unité .mount correspondante pour définir les spécificités du montage
path	définit un chemin pouvant être utilisé pour une activation basée sur le chemin. Par défaut, une unité .service du même nom sera démarrée lorsque le chemin d'accès atteindra l'état spécifié. Ceci utilise inotify pour surveiller le chemin des modifications
target	utilisée pour fournir des points de synchronisation aux autres unités lors du démarrage ou de la modification des états en regroupant des unités logiquement. Ils peuvent également être utilisés pour amener le système dans un nouvel état
timer	définit une minuterie qui sera gérée par systemd, similaire à une tâche cron pour une activation retardée ou planifiée. Une unité .service correspondante démarrera lorsque le temps sera écoulé

Nom du type d'unité	Description
snapshot	recontruit l'état actuel du système après les modifications. Les instantanés ne survivent pas d'une session à l'autre et sont utilisés pour restaurer des états temporaires pendant des changements de configuration de service
swap	active des zones de swap sur le système. Le nom de ces unités doit refléter le périphérique ou le chemin d'accès de l'espace à utiliser
slice	associée aux nœuds de groupe de contrôle Linux (cgroup), ce qui permet de restreindre ou d'affecter des ressources à tout processus associé à cette unité. Le nom reflète sa position hiérarchique dans l'arborescence du groupe de contrôle.
scope	créée automatiquement par systemd à partir des informations reçues de ses interfaces de bus. Ces unités permettent de gérer des ensembles de processus système créés en externe.

On constate ainsi que **systemd** sait gérer de nombreuses unités différentes. De nombreux types d'unités fonctionnent ensemble pour ajouter des fonctionnalités. Par exemple, certaines unités sont utilisées pour déclencher d'autres unités et ce qui offre une fonctionnalité d'activation.

Si un service est requis sous un certain nom mais qu'aucune unité correspondante n'est trouvée, **systemd** cherche un script *SysV* init du même nom (sans le suffix .service) and créer dynamiquement une unité .service à partir de ce script. Cette fonctionnalité permet une compatibilité forte (mais pas totale) avec *SysV*.

5.1.5 Anatomie d'une fichier unité

Les fichiers d'unité consistent habituellement de trois sections :

— [Unit] — contient des options génériques qui ne sont pas dépendantes du type de l'unité. Ces options fournissent une description de l'unité, spécifient le comportement de l'unité, et définissent les dépendances avec d'autres unités. Pour une liste des options [Unit] les plus fréquemment utilisées, veuillez consulter la Tableau « Options importantes de la section [Unit] ».

— [unit type] — si une unité possède des directives spécifiques au type, celles-ci seront regroupées dans une section nommée par le type d'unité (par exemple, les fichiers de l'unité de service contiennent la section [Service]). Le Tableau « Options importantes de la section [Service] » donne un aperçu des options [Service] les plus fréquemment utilisées.

— [Install] — contient des informations sur l'installation de l'unité utilisée par les commandes systemctl enable et disable. Le Tableau « Options importantes de la section [Install] » fournit une liste non exhaustive de différentes options de la section [Install].

Option	Description
`Description=`	Description significative de l'unité. Ce texte est de plus affiché dans la sortie de la commande `systemctl status`.
`Documentation=`	Fournit une liste des URI référençant la documentation de l'unité.

Option	Description
`After=`[1]	Définit l'ordre dans lequel les unités sont lancées. L'unité est lancée uniquement après l'activation des unités spécifiées dans `After=`. Contrairement à `Requires=`, `After=` n'active pas explicitement les unités spécifiées. L'option `Before=` offre une fonctionnalité contraire à `After=`.
`Requires=`	Configure les dépendances sur d'autres unités. Les unités répertoriées dans `Requires=` sont activées ensembles avec l'unité. Si le lancement de l'une des unités requises échoue, l'unité n'est pas activée.
`Wants=`	Configure les dépendances plus faibles que `Requires=`. Si l'une des unités répertoriées ne démarre pas, cela n'aura pas d'impact sur l'activation de l'unité. C'est la méthode recommandée pour établir des dépendances d'unité personnalisées.
`Conflicts=`	Configure des dépendances négatives, à l'opposé de `Requires=`.

Tableau « Options importantes de la section [Unit] » [1] Dans la plupart des cas, il est suffisant de ne déterminer que les dépendances d'ordonnancement avec les options de fichier `After=` et `Before=`. Cependant, si vous définissez aussi une ou plusieurs exigences de dépendance avec `Wants=` (conseillé) ou `Requires=`, la dépendance d'ordonnancement devra toujours être spécifiée car l'ordonnancement et les exigences de dépendance fonctionnent indépendamment.

Option	Description
`Type=`	Configure le type de démarrage de processus d'unité qui affecte la fonctionnalité d'ExecStart et des options connexes. L'une des options suivantes :

Tableau « Options importantes de la section [Service] »

- `simple` – valeur par défaut. Le processus lancé par `ExecStart=` est le processus principal du service.
- `forking` – le processus lancé par `ExecStart=` engendre un processus enfant qui devient le processus principal du service. Le processus parent s'arrête lorsque le startup est terminé.
- `oneshot` – ce type est similaire à `simple`, mais le processus s'arrête avant de lancer les unités suivantes.
- `dbus` – ce type est similaire à `simple`, mais les unités suivantes sont lancées uniquement après que le processus principal ait obtenu un nom D-Bus.
- `notify` – ce type est similaire à `simple`, mais les unités suivante sont lancées uniquement après l'envoi d'un message de notification via la fonction sd_notify().
- `idle` – similaire à `simple`, l'exécution du binaire du service est retardée jusqu'à ce que toutes les tâches soient terminées, ce qui permet d'éviter de mélanger la sortie du statut avec la sortie shell des services.

Option	Description
`ExecStart=`	Spécifie les commandes ou scripts à exécuter lorsque l'unité est lancée. `ExecStartPre=` et `ExecStartPost=` spécifient des commandes personnalisées à exécuter avant et après `ExecStart=`. `Type=oneshot` permet de spécifier des commandes multiples personnalisées exécutées de manière séquentielle par la suite.
`ExecStop=`	Spécifie les commandes ou scripts à exécuter lorsque l'unité est arrêtée.
`ExecReload=`	Spécifie les commandes ou scripts à exécuter lorsque l'unité est rechargée.
`Restart=`	Avec cette option activée, le service est redémarré après que son processus se soit arrêté, à l'exception d'un arrêt gracieux avec la commande `systemctl`.

Option	Description
`RemainAfterExit=`	Si défini sur True, le service est considéré comme actif, même lorsque tous ses processus sont arrêtés. La valeur par défaut est False. Cette option est particulièrement utile si `Type=oneshot` est configuré.

5.1.5.1 Tableau « Options importantes de la section [Install] »

Option	Description
`Alias=`	Fournit une liste de noms supplémentaires de l'unité séparés par des espaces. La plupart des commandes `systemctl`, sauf `systemctl enable`, peuvent utiliser des alias à la place du nom de l'unité.
`RequiredBy=`	Une liste des unités qui dépendent de l'unité. Lorsque cette unité est activée, les unités répertoriées dans `RequiredBy=` obtiennent une dépendance Require de l'unité.
`WantedBy=`	Une liste des unités qui dépendent faiblement de l'unité. Lorsque cette unité est activée, les unités répertoriées dans `WantedBy=` obtiennent une dépendance `Want=` de l'unité.
`Also=`	Indique une liste des unités à installer ou désinstaller avec l'unité.

Une gamme entière d'options qui peuvent être utilisées pour régler de manière détaillée la configuration de l'unité.

5.1.6 Dépendances et ordonnancement

Certains utilisateurs occasionnels ont peut-être entendu dire que systemd commence tout en même temps et donc que le système démarre ainsi plus rapidement. Mais la réalité n'est pas si simple. Examinons un peu plus en profondeur la façon dont systemd comprend les relations entre unités.

5.1.6.1 Dépendances de l'unité

Les fichiers d'unité incluent un système de dépendances. Toute unité peut souhaiter ou nécessiter le lancement d'une ou plusieurs autres unités avant de pouvoir fonctionner. Ces dépendances sont définies dans des fichiers d'unité avec les directives `Wants=` et `Requires=`. La différence entre celles-ci est simple.

- Si unit1 contient `Wants=unit2` comme dépendance, lorsque unit1 est exécutée, unit2 sera également exécutée. Mais le bon lancement d'unit2 ou son échec n'affecte pas l'unité1 en cours d'exécution.
- Cependant, lorsque l'unité1 a la valeur de `Requires=unit2`, les deux unités seront exécutées, mais si l'unit2 n'aboutit pas, l'unit1 est également désactivée. Cela se produit indépendamment du fait que les processus de unit1 auraient autrement fonctionné correctement.

Avez-vous remarqué quelque chose de subtil dans cette description ? Il n'est jamais question d'ordre. Lorsque systemd démarre le système, il charge tous les fichiers unité et les lit pour déterminer les différentes dépendances. Lorsque unit1 s'exécute dans ces exemples, unit2 est exécuté en même temps. Il est important de savoir que les dépendances et l'ordre sont deux choses différentes pour systemd.

Voici un exemple de dépendance dans une partie du fichier d'unité sshd.service :

```
Wants=sshd-keygen.service
```

Ainsi, lorsque sshd.service est exécuté, sshd-keygen.service est également exécuté. Cependant, sshd-keygen.service n'a pas besoin de réussir pour que sshd.service s'exécute correctement.

Pourquoi une telle dépendance devrait-elle exister ? Dans ce cas, sshd-keygen.service crée de nouvelles clés SSH pour un serveur si elles n'existent pas. Nous voulons toujours vérifier si ces clés existent avant le démarrage du serveur SSH. S'ils existent déjà, l'unité de service se termine avec une erreur l'indiquant. Mais dans ce cas, nous souhaitons toujours que le serveur SSH fonctionne comme d'habitude.

5.1.6.2 Ordonnancement des unités

Cela ne signifie pourtant pas que **systemd** est incapable d'ordonner le lancement des unités. En l'absence d'autres instructions, **systemd** exécuterait un groupe d'unités en même temps. C'est probablement pourquoi certaines personnes croient que **systemd** lance tout en même temps (ou «en parallèle»). Bien entendu, il est parfois nécessaire que les processus s'exécutent dans un certain ordre.

Heureusement, **systemd** a également des directives d'unité pour ce problème, `Before=` et `After=`. Ces directives fonctionnent à peu près comme leur nommage le laisse entendre :

— Si unit1 contient la directive `Before=unit2` et si les deux unités sont exécutées, unit1 sera exécutée complètement avant le démarrage de unit2.

— Si unit1 a pour directive `After=unit2` et si les deux unités sont exécutées, unit2 sera exécuté intégralement avant le démarrage de unit1.

Encore une fois, notez que cet ordre n'affecte pas les dépendances. Ni l'un ni l'autre cas ne provoque le lancement de unit2. Regardons à nouveau l'unité sshd.service :

```
Wants=sshd-keygen.service After=network.target sshd-keygen.service
```

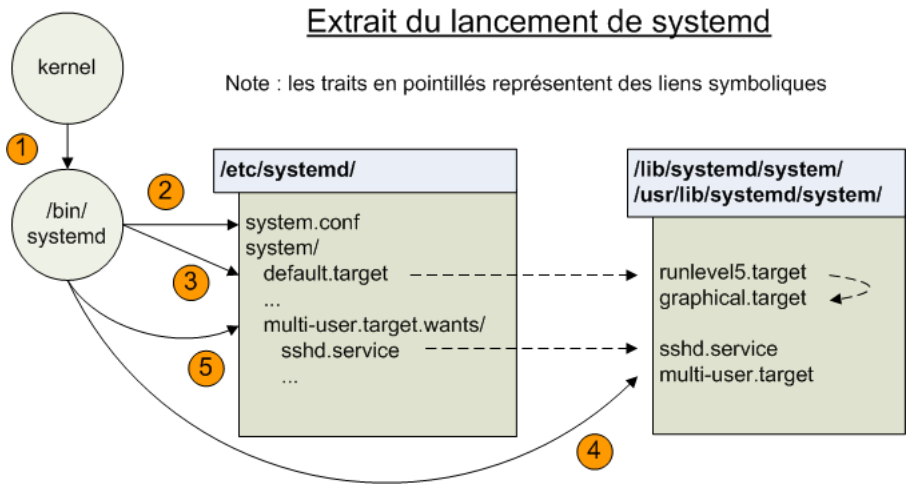
La directive de commande `After=` permet de s'assurer que le serveur SSH ne fonctionnera pas après l'unité de génération de clé d'hôte et après la mise en place du réseau. (L'unité network.target veille à ce que différentes unités mettent en place le réseau.) Les dépendances, telles que `Wants=` et `Requires=`, sont souvent utilisées avec `After=` pour préserver les dépendances et l'ordre approprié

5.1.6.3 Temps de démarrage plus rapide

Ainsi, **systemd** trie automatiquement les fichiers unité pour lire les informations de dépendance et d'ordonnancement. Il utilise ces informations pour permettre à de nombreux services de démarrer presque simultanément tout en préservant l'ordre le cas échéant et cette manipulation est l'une des raisons pour lesquelles le démarrage peut être plus rapide sous **systemd**.

5.1.7 Aperçu

5.1.7.1 Petit aperçu du déroulement d'init lors du démarrage du système



Lorsque systemd est lancé par le noyau (1), il lit son fichier /etc/systemd/system.conf (2), puis il examine la première unité à lancer : default.target (3) qui est en réalité un lien symbolique vers runlevel5.target (lui-même lien symbolique vers graphical.target).

Ce fichier contient une directive After=multi-user.target, donc systemd cherche la configuration de cette unité (4) et l'interprète. Mais il existe un répertoire multi-user.target.wants (5), donc systemd parcourt ce répertoire pour activer les unités qui y sont décrites. On y trouve par exemple sshd.service, etc.