

# Démarrage du noyau Linux

par Tim Jones ([Site perso de Tim Jones](#)) Traduction par David Côme

Date de publication : 22/06/2008

Cet article a pour but de vous présenter le processus de démarrage de Linux.

- I - Introduction
- II - Vue générale
- III - Démarrage du système
- IV - Phase 1: le chargeur d'amorçage
- V - Phase 2: un autre chargeur d'amorçage
- VI - Le noyau
- VII - Init
- VIII - Résumé
- IX - Ressource

## I - Introduction

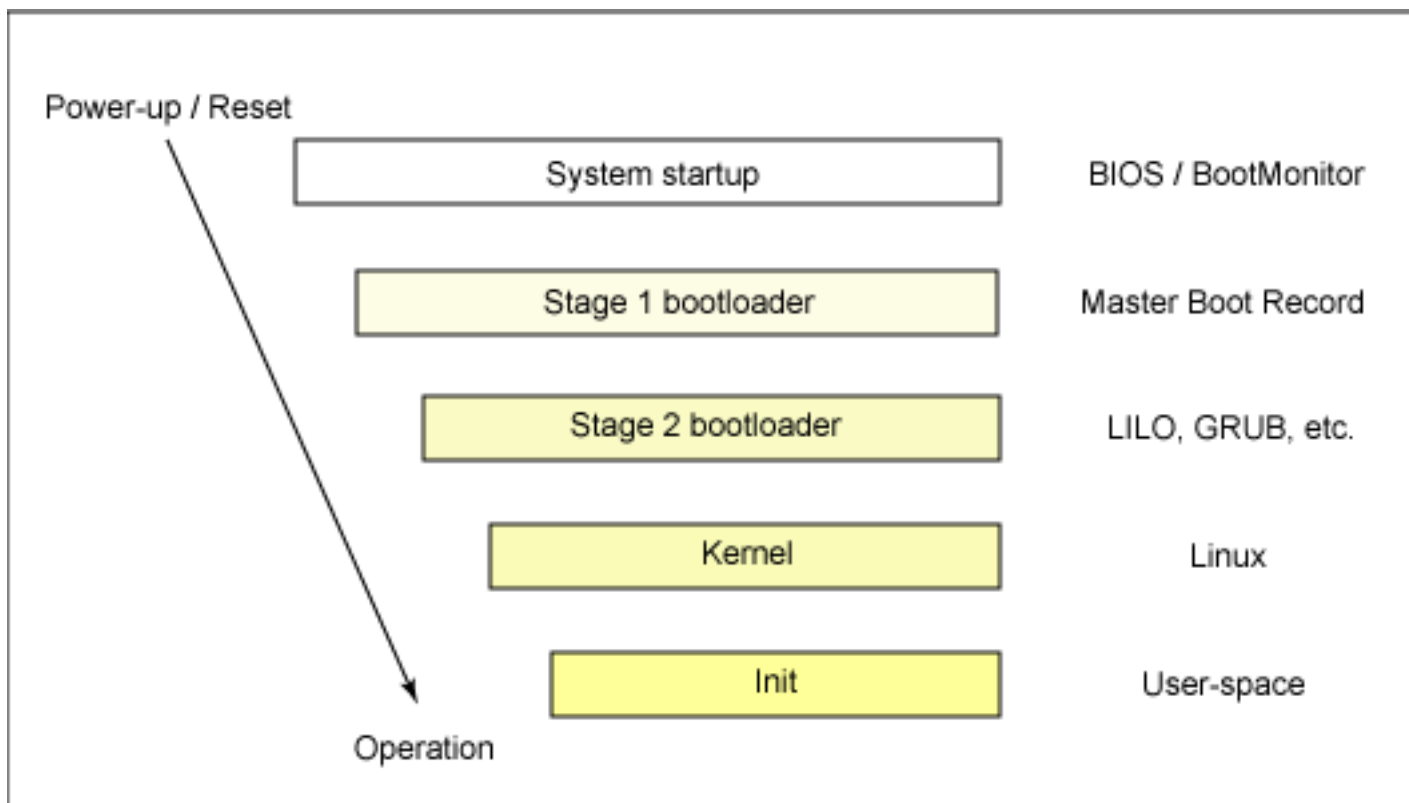
Le processus de mise en route d'un système d'exploitation consiste en l'exécution d'un certain nombre d'étapes. Mais, bien que vous soyez en train de démarrer un ordinateur personnel basé sur une architecture x86 ou un système embarqué avec un PowerPc, la plupart des opérations sont étonnamment semblables. Cet article explore le processus de démarrage de linux depuis le bootstrap initial jusqu'au lancement de la première application dans l'espace utilisateur. Tout au long, vous allez apprendre des choses à propos des nombreuses autres étapes liées au démarrage, tel que le chargeur d'amorçage, la décompression du noyau, l'initial RAM disk,...

Dans les premiers temps de l'informatique, démarrer un ordinateur signifiait fournir une bande perforée contenant les instructions du programme de démarrage ou encore charger manuellement un programme en utilisant un panneau frontal. Aujourd'hui, les ordinateurs sont équipés avec un certain nombre de services pour simplifier le démarrage, mais ceci ne le rend pas pour autant simple.

Commençons avec une vue de haut niveau du démarrage d'un système linux pour que vous puissiez voir le processus dans sa globalité. Ensuite, nous nous attarderons individuellement sur chaque phase du processus. Les références vous aideront à naviguer à l'intérieur du noyau et à approfondir.

## II - Vue générale

La figure 1 vous donne une vision très globale du processus de démarrage.



*Vue très globale du processus de boot*

Quand un système démarre, ou est redémarré, le processeur exécute du code à une adresse fixe. Dans un PC, cette adresse fixe est celle du BIOS (*Basic Input/Output System*), qui est stocké dans une ROM sur les cartes mères. Dans un environnement embarqué, le processeur exécute le début du segment de code pour démarrer un programme à une adresse connue dans une mémoire flash/ROM. Dans tous les cas, le résultat est le même. Mais à cause de la flexibilité offerte par le PC, le BIOS doit aussi déterminer quels périphériques sont candidats pour démarrer dessus. Nous y reviendrons plus tard.

Quand un périphérique sur lequel on peut démarrer est trouvé, le premier programme du processus de démarrage est chargé en RAM puis exécuté. Ce chargeur de démarrage fait au plus 512 bits (un secteur) et son rôle est de charger le deuxième programme.

Quand le deuxième programme (le chargeur d'amorçage) est chargé en RAM et exécuté, un *splash-screen* est souvent affiché et le noyau linux ainsi que l'optionnel *initrd* (initial RAM disk: le système de fichier principal temporaire) sont chargés en mémoire. Après que les images ont été chargées, le deuxième programme passe la main à l'image du noyau et le noyau est alors décompressé. À ce stade, le deuxième programme vérifie aussi le matériel, énumère les périphériques attachés, monte le périphérique principal et charge les modules du noyau nécessaires. Quand tout cela est fini, le premier programme de l'espace utilisateur (*init*) démarre, et l'initialisation à haut niveau du système a lieu.

C'est ainsi que, de façon très synthétique, se déroule le démarrage d'un système linux. Maintenant creusons un peu plus loin et regardons dans les détails ce qu'il se passe.

### III - Démarrage du système

Le démarrage du système dépend du matériel sur lequel linux est démarré. Sur des plateformes embarquées, un bootstrap est utilisé quand le système est mis sous tension, ou re-démarré. On peut par exemple citer u-boot, RedBoot ou encore MicroMonitor de Lucent. Les plateformes embarquées disposent habituellement d'un programme principal de démarrage. Ce programme réside dans une zone spéciale de la mémoire flash sur la machine cible et fournit un moyen pour télécharger une image du noyau Linux dans la mémoire flash pour l'exécuter. En plus de pouvoir stocker et démarrer une image du noyau linux, ce programme principal de démarrage assure aussi plusieurs niveaux de test et d'initialisation du matériel. Dans une plateforme embarquée, il regroupe souvent le premier et le deuxième programme.

Dans un ordinateur, le démarrage de Linux commence dans le BIOS à l'adresse 0xFFFF0. La première chose que réalise le BIOS est le *power-on self test* (POST). Le rôle du POST est de vérifier le matériel. La seconde chose que fait le BIOS est d'énumérer puis d'initialiser les périphériques locaux.

Étant donné les différents usages des fonctions du BIOS, ce dernier est conçu en deux parties : le POST et les fonctions utilitaires. Après que le POST est fini, il est nettoyé de la mémoire mais les fonctions utilitaire du BIOS restent et sont toujours disponibles pour le système que l'on va démarrer.

Pour démarrer un système d'exploitation, la routine du BIOS cherche un périphérique qui est à la fois actif et démarrable selon l'ordre de préférence défini par les réglages du CMOS (*metal oxyde semiconductor*). Un périphérique démarrable peut être une disquette, un CD-ROM, une partition d'un disque, un périphérique situé sur le réseau ou encore une clé USB.

Le plus souvent, Linux est démarré depuis un disque dur, où le *master boot record* (MBR) contient le premier chargeur d'amorçage. Le MBR est un bloc de 512 octets, localisé dans le premier secteur du disque (secteur 1 du cylindre 0, tête 0). Après que le MBR a été chargé en RAM, le BIOS lui donne le contrôle.

#### **Extraire le MBR**

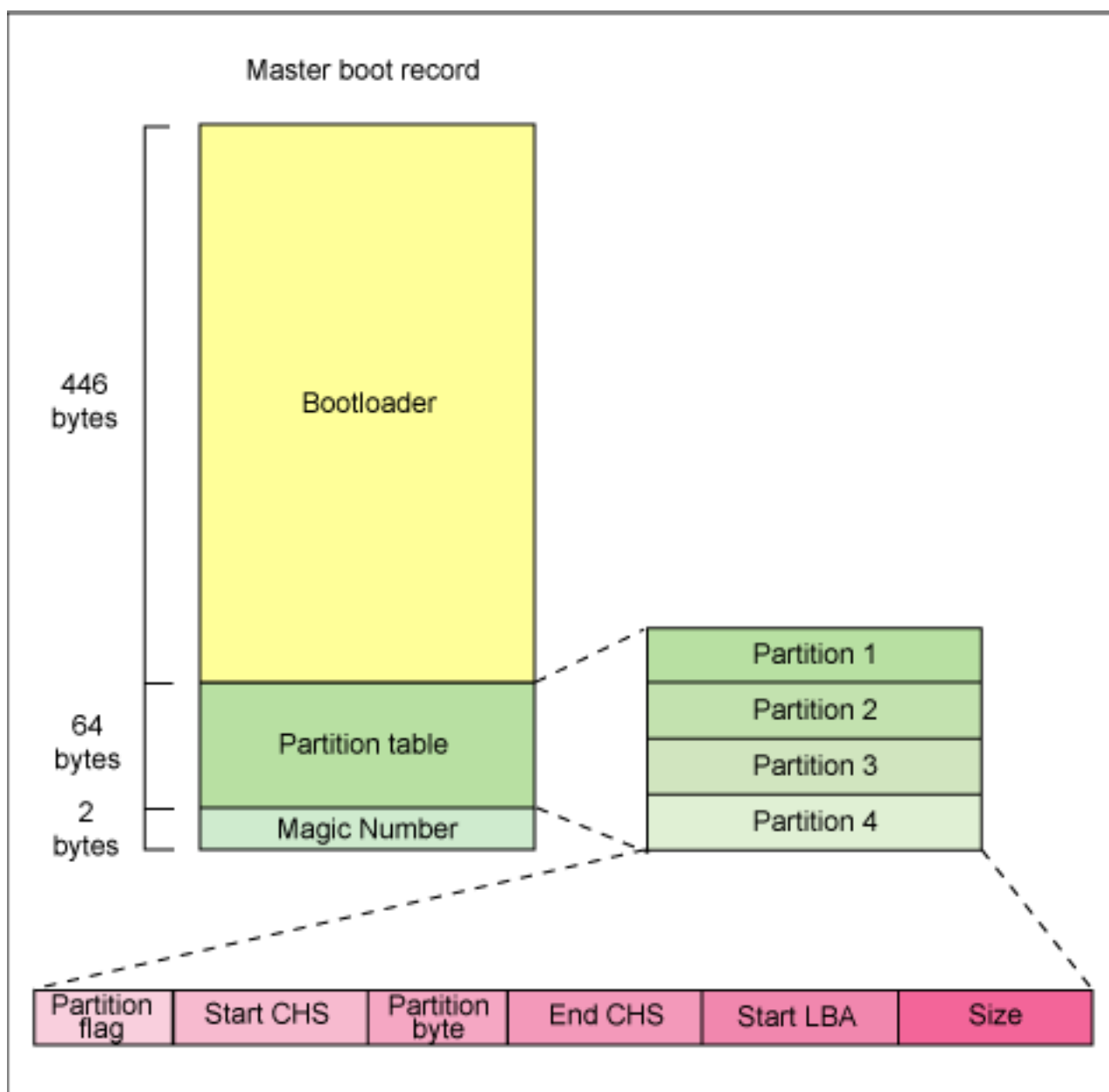
*Pour voir le contenu de votre MBR, utilisez les commandes suivantes:*

```
# dd if=/dev/hda of=mbr.bin bs=512 count=1
# od -xn mbr.bin
```

La commande dd, qui doit être exécutée en tant que root, lit les 512 premiers octets depuis /dev/hda (le premier disque de type IDE) et écrit le tout dans le fichier mbr.bin. Puis la commande od affiche le contenu du fichier dans un format hexadécimal, ASCII si possible.

## IV - Phase 1: le chargeur d'amorçage

Le premier chargeur d'amorçage qui réside dans un unique secteur de 512 octets contient à la fois un programme de chargement et une petite table de partitions (regardez la figure 2). Les 446 premiers octets sont précisément le premier chargeur d'amorçage, qui contient en même temps du code exécutable et les messages d'erreurs. Les 64 prochains octets forment la table des partitions, qui contient un enregistrement sur 16 bits de chacune des partitions primaires ou étendues présentes sur le disque. Le MBR se termine avec avec 2 bits étant définis comme étant un nombre magique (0xAA55), nombre qui sert à vérifier que le MBR est correct.



*Schéma détaillé du MBR*

Le travail du premier chargeur d'amorçage est de trouver puis de charger le deuxième chargeur d'amorçage. Il réalise ceci en regardant à travers la table des partitions celles qui sont marquées comme actives. Quand il trouve une

partition active, il examine quand même les autres partitions pour vérifier qu'elles sont bien inactives. Quand cette vérification est terminée, l'enregistrement actif de la partition est chargé en RAM puis exécuté.

## V - Phase 2: un autre chargeur d'amorçage


Le deuxième chargeur d'amorçage pourrait être à juste titre appelé le *kernel loader* (ou encore chargeur de noyau). En effet, le devoir de ce deuxième programme est de charger le noyau Linux (une obligation) et *l'initial RAM disk* (qui est optionnel).

La combinaison du premier et du deuxième programme est appelée soit *Linux Loader* (LILO) ou *GRand Unified Bootloader* (GRUB) dans un environnement de type x86. À cause d'un certain nombre de problèmes de LILO qui ont été corrigés dans GRUB, nous regarderons seulement GRUB (Regardez les ressources additionnelles sur GRUB, LILO et les sujets apparentés dans la section ressource.)

Une bonne chose de GRUB est sa connaissance du système de fichiers utilisé par Linux. En effet, au lieu d'utiliser des secteurs bruts comme LILO le fait, GRUB est capable de charger un noyau linux depuis un système de fichiers de type ext2 ou ext3. Il y arrive en utilisant deux niveaux de chargement parmi les 3 possibles. En effet, le premier niveau (le MBR) démarre un niveau intermédiaire (noté 1.5) qui peut utiliser un système de fichier particulier, ce système étant celui sur lequel le noyau se situe. Par exemple, le niveau 1 va démarrer le `reiserfs_stage1_5` pour charger le noyau linux si celui-ci se trouve sur une partition de type reiserfs. Il chargera `e2fs_stage1_5` si le noyau se trouve sur une partition formatée en ext2 ou 3. Quand le niveau intermédiaire est chargé et a démarré, le deuxième programme peut enfin être chargé.

Avec le deuxième programme, GRUB peut, sur demande, afficher une liste des noyaux disponibles (cette liste étant définie dans `/etc/grub.conf` avec un lien symbolique depuis `/etc/grub/menu.lst` et `/etc/grub.conf`). Vous pouvez sélectionner un noyau et même le modifier avec des paramètres additionnels. Vous pouvez même utiliser la ligne de commande fournie par GRUB pour avoir un plus grand contrôle sur le processus de démarrage.

Avec le deuxième programme en mémoire, le système de fichiers est consulté, et l'image du noyau par défaut ainsi que `initrd` sont chargés en mémoire. Quand les images sont prêtes, le deuxième programme invoque l'image du noyau.

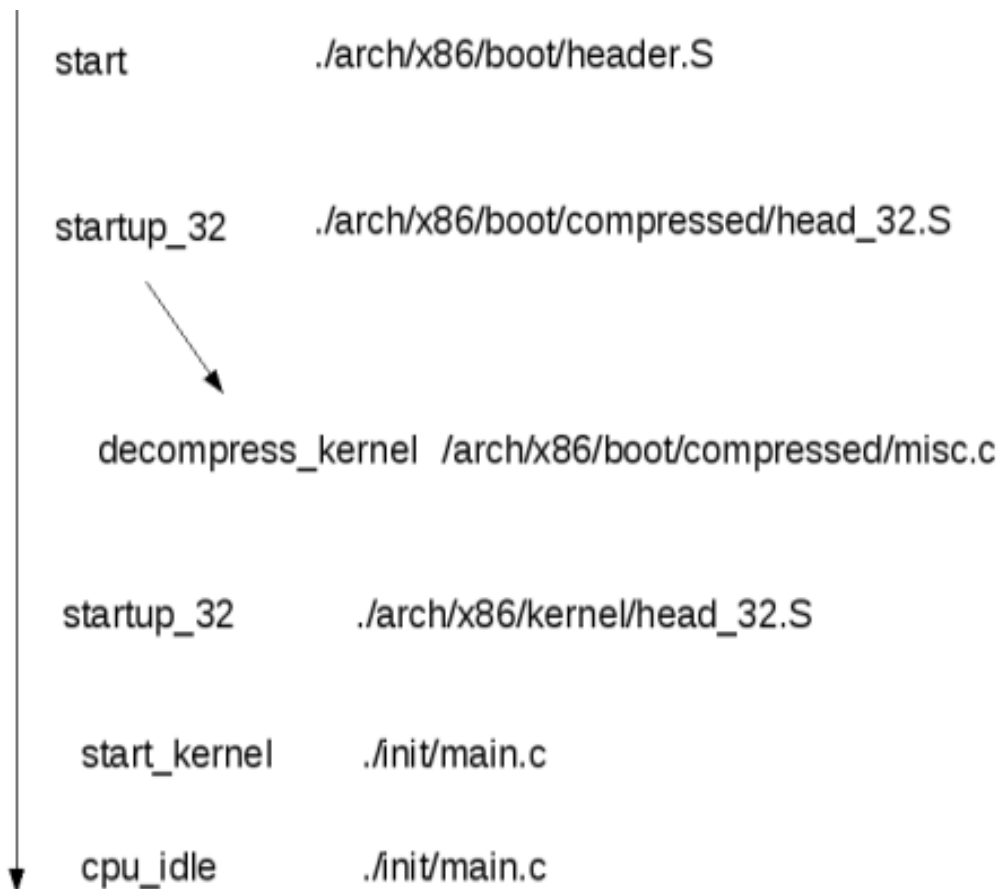
 *Le répertoire `/boot/grub` contient le `stage1`, `stage1_5` et `stage2` ainsi qu'un bon nombre d'autres chargeurs d'amorçage (par exemple, les CD-ROM utilisent `iso9660_stage1_5`)*

## VI - Le noyau

Avec l'image du noyau en mémoire et l'abandon du contrôle par le deuxième programme, la phase du noyau peut commencer. L'image du noyau n'est pas plus qu'un noyau exécutable, mais compressé dans une image. Typiquement, c'est soit une zImage (une image compressée faisant moins de 512 koctets) ou une bzImage (une grosse image compressée, faisant plus de 512 koctets), qui a été précédemment compressée avec zlib. Au début du noyau se trouve une routine qui dresse une liste minimale de la configuration du matériel puis qui décompresse le noyau contenu dans l'image pour ensuite le placer en "haute" mémoire. Si initrd est présent, la routine déplace le noyau dans la RAM et note cette action pour plus tard. Ensuite la routine appelle le noyau et le démarrage du noyau peut commencer.

Quand la bzImage (pour une image destinée au i386) est appelée, vous commencez à `./arch/x86/boot/header.S` dans la routine assembleur nommée `start` (regardez la figure pour les grandes lignes). La routine effectue plusieurs initialisations basiques sur le matériel puis appelle la routine `startup_32` dans `./arch/x86/boot/compressed/head_32.S`. Cette routine met en place un environnement basique (pile,...) et nettoie le *Bloc Started by Symbol* (BSS). Le noyau est ensuite décompressé à travers l'appel d'une fonction C nommée `decompress_kernel` localisée dans `./arch/x86/boot/compressed/misc.c`. Quand le noyau est décompressé en mémoire, il est appelé. A ce moment, il y a appel d'une autre fonction `startup_32` mais cette fois dans `./arch/x86/kernel/header_32.S`

Dans la nouvelle fonction `startup_32` (aussi appelée *swapper* ou encore processus 0), les tables de pages sont initialisées et la pagination de la mémoire est activée. le type de CPU ainsi que le coprocesseur arithmétique (FPU) sont détectés durant cet appel, puis mis de côté pour être réutilisé plus tard. La fonction `start_kernel` est ensuite appelée (`init/main.c`) pour utiliser la partie du noyau non dépendante de l'architecture. Cette dernière fonction peut être considérée comme la fonction principale du noyau.



*Chaîne d'appel des principales fonctions dans le noyau*

Avec l'appel à `start_kernel`, une liste d'appel à des fonctions d'initialisations à lieu pour configurer les interruptions, procéder à la configuration de la mémoire et charger `initrd`. À la fin, un appel à `kernel_thread` est réalisé (dans `./arch/x86/kernel/process.c`) pour démarrer la fonction `init` (après l'appel à `cpu_idle`). Avec les interruptions activées, l'ordonnanceur pré-emptif prend périodiquement le contrôle pour fournir le multi-tâche.

Durant le démarrage du noyau, `initrd` a été chargé en mémoire par le deuxième programme puis copié en RAM et monté. `initrd` sert en tant que système de fichier principal temporaire en RAM et permet au noyau de démarrer sans avoir monté un seul disque physique. Depuis que les modules nécessaires pour interfacer le noyau avec les périphériques peuvent être une part de `initrd`, le noyau peut être très petit mais pour autant supporter un grand nombre de diverses configurations. Après que le noyau a démarré, le système de fichiers est changé via un appel à `pivot_root`, ce qui provoque le démontage de `initrd` et le montage du vrai système de fichiers principal.

Le service offert par `initrd` permet de créer de tout petits noyaux linux avec des drivers compilés en tant que modules chargeables sur demande. Ces modules donnent au noyau les moyens d'accéder aux disques et aux systèmes de fichiers présents sur ces disques, aussi bien qu'au reste du matériel. Étant donné que la racine du système de fichier (`/`) pointe obligatoirement sur une partition du disque dur, le programme `initrd` fournit de quoi accéder au disque physique et monter la vraie racine du système de fichiers. Dans une plateforme embarquée qui ne possède pas de disque dur, `initrd` peut être le système de fichiers principal final, ou ce dernier peut aussi être monté via le réseau et la technologie NFS (Network File System)

 **Sortie de la fonction `decompress_kernel`**

*La fonction `decompress_kernel` correspond au fameux message de décompression du noyau : "Uncompressing Linux... Ok, booting the kernel."*

## VII - Init

Après que le noyau a démarré et qu'il se soit initialisé, il démarre le premier programme de l'espace utilisateur. C'est le premier programme appelé à être écrit avec la bibliothèque C standard. En effet, avant ce programme, aucun de ceux qui ont été appelés n'avaient été écrits en C standard.

Dans un ordinateur personnel, la première application utilisateur est souvent `/sbin/init`, même si ce n'est pas une obligation. Les systèmes embarqués requièrent rarement une initialisation aussi intensive que ce que fait `init` (à travers sa configuration dans `/etc/inittab`). Dans beaucoup de cas, on peut appeler un simple script shell qui va démarrer les applications nécessaires.

## VIII - Résumé

Plus que linux lui même, le processus de démarrage de linux est hautement flexible, supportant un grand nombre de processeurs et d'architectures. Dans les débuts, les chargeurs d'amorçages fournissaient un moyen de démarrer le noyau sans aucune fioriture supplémentaire. LILO a étendu les capacités de démarrage mais manquait du support des systèmes de fichiers. La dernière génération de chargeurs d'amorçages, tel GRUB, permet à Linux de démarrer depuis un grand nombre de systèmes de fichier (de Minix à ReiserFs).

## IX - Ressource

### Apprendre

- **Boot Records Revealed** est une source importante d'information sur les MBR et sur les divers boot-loader. Le site ne fait pas qu'étudier des MBR mais discute aussi de LILO, GRUB ainsi que des divers boot-loader de Windows.
- Regarder la page sur la **géométrie des disques** pour comprendre leur architecture. Vous trouverez un résumé des attributs des disques.
- Un **live-cd** est un système d'exploitation qui démarre depuis un CD ou un DVD sans l'utilisation d'un disque dur.
- **Boot loader showdown: Getting to know LILO and GRUB" (developerWorks, August 2005)** vous permettra d'examiner en détail LILO et GRUB.
- **LILO** a été le précurseur de GRUB, mais vous pouvez toujours le trouver dans les distributions.
- la commande **mkinitrd** sert à créer *l'initial RAM disk*. Cette commande est très utile pour créer un premier système de fichier principal pour configurer le démarrage de façon à permettre l'accès aux blocs et accéder au vrai système de fichier principal.
- Sur le site du projet **Debian Linux Kernel**, vous pourrez trouver plus d'informations sur le noyau linux, le démarrage et le développement sur des architectures embarquées.

### Obtenir les logiciels

- **GNU GRUB** est un shell pour démarrer rempli d'options et très flexible.
- **coreboot** est un BIOS de remplacement. Il ne permet pas seulement de démarrer linux, mais coreboot est lui même un noyau linux.
- **OpenBios** est un autre BIOS portable qui peut s'installer sur un grand nombre d'architectures comme x86, ou encore AMD64.
- Sur **kernel.org**, obtenez le dernier noyau stable.

### A propos de l'auteur.

M. Tim Jones est un ingénieur architecte pour les systèmes embarqués et l'auteur de *GNU/Linux Application Programming, AI Application Programming, and BSD Sockets Programming from a Multilanguage Perspective*. Sa formation d'ingénieur va du développement de noyau pour la géo-synchronisation de systèmes spatiaux à l'architecture des systèmes embarqués en passant par le développement réseau. Tim est employé en tant qu'ingénieur consultant chez Emulex Corp. à Longmont, Colorado, USA.

Merci à ovh, gorgonite ainsi qu'à troumad et diogene pour les relectures.

