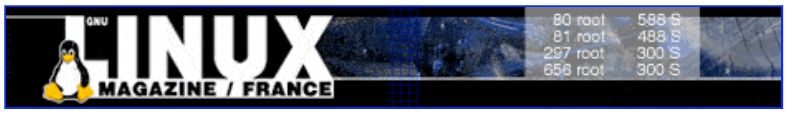


20 août 2008

Le boot Linux

Catégorie : Distribution Tags : lmhs

~~Retrouvez cet article dans :~~ Linux Magazine Hors série 17

Le noyau Linux est le système de base de toute distribution Linux. Il permet la gestion du matériel, de la mémoire, des entrées/sorties, la distribution du CPU pour l'exécution des processus et bien d'autres choses encore. Mais il reste souvent le côté obscur du système, ce qu'il ne faut surtout pas toucher, sous peine de ne plus rien voir fonctionner.

Cet article fournit quelques précisions sur le fonctionnement central de tout système Linux. Nous verrons la compilation d'un nouveau noyau pour les cas de mise à jour ou pour les besoins de noyaux taillés sur mesure. Après avoir compilé un noyau, nous montrerons comment on le fait "booter" par l'intermédiaire d'un boot loader, la notion d'initrd ou de ramdisks, les paramètres qu'il accepte et les différents périphériques disponibles pour le boot. Nous terminerons cet article en examinant les possibilités et les moyens de faire cohabiter plusieurs noyaux sur un même système.
Nous n'aborderons cependant pas le système des patches et la manière de mettre à jour les sources de votre noyau. Un article sur le sujet (cf réf. [1]) a déjà été écrit.
Signalons que tout au long de l'article, nous utiliserons la dernière version du noyau Linux, la version 2.4.22 sur plate-forme Intel.

Compilation et installation du noyau Linux

Les sources

Évidemment, pour compiler et installer, il nous faut d'abord les sources du noyau. La première solution est d'aller sur le site officiel, http://www.kernel.org.
Toutes les versions y sont présentes. Que ce soit les anciennes, les stables ou les futures versions du noyau (branche 2.6), vous les trouverez toutes sur ce site (cf réf. [2] pour le système de numérotage des versions de noyaux).
Actuellement, nous sommes à la version stable 2.4.22. Voici donc comment récupérer les sources et vérifier l'intégrité de l'archive récupérée (cf. http://www.kernel.org/signature.html) :

```
# pwd
/usr/src
# wget http://www.kernel.org/pub/linux/kernel/v2.4/linux-2.4.22.tar.bz2
[...]
# wget http://www.kernel.org/pub/linux/kernel/v2.4/linux-2.4.22.tar.bz2.sign
[...]
# gpg -- keyserver wwwkeys.pgp.net --recv-keys 0x517D0F0E
gpg: requesting key 517D0F0E from wwwkeys.pgp.net ...
gpg: key 517D0F0E: public key imported
gpg: Total number processed: 1
```

```
gpg:               imported: 1
# gpg --verify linux-2.4.22.tar.bz2.sign linux-2.4.22.tar.bz2
gpg: Signature made Mon Aug 25 13:54:52 2003 CEST using DSA key ID 517D0F0E
gpg: Good signature from "Linux Kernel Archives Verification Key <ftpadmin@kernel.org>"
Could not find a valid trust path to the key.  Let's see whether we
can assign some missing owner trust values.

No path leading to one of our keys found.

gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
gpg: Fingerprint: C75D C40A 11D7 AF88 9981  ED5B C86B A06A 517D 0F0E
```

Ne faites pas attention au warning, gpg nous indique bien que c'est la bonne signature.
Moins universelle comme méthode, les distributions proposent des packages de sources de noyaux ou de noyaux pré-compilés. Le problème généralement est qu'il n'y a pas la dernière version. Debian par exemple propose au maximum la version 2.4.18 (connue comme étant vulnérable à une faille de sécurité ptrace()). La meilleure solution est donc évidemment d'aller sur le site officiel ou sur un miroir.

La compilation du noyau Linux

Vous avez tout d'abord besoin d'être root et de vous positionner dans le répertoire ~~/usr/src~~ (par convention, c'est dans ce répertoire que nous mettons les sources) pour désarchiver le tarball que nous venons de récupérer.
Avec les anciennes versions de noyaux (versions 2.2.x et antérieures), l'installation des sources créait le répertoire ~~/usr/src/linux.~~ Il fallait alors prendre soin de le renommer par ~~/usr/src/linux-x.y.z~~ si vous décidiez d'installer de nouvelles sources. Avec les nouvelles versions du noyau (version 2.4.x et plus), vous n'avez plus ce souci puisque le répertoire ~~/usr/src/linux-x.y.z~~ est créé à chaque installation.
A partir de maintenant et tout au long de l'article, nous ferons référence au répertoire ~~/usr/src/linux-x.y.z~~ par la variable ~~$SRC_LINUX~~.
Procédons ensuite à la compilation des sources. La première chose à faire est d'utiliser la commande make ~~mrproper~~. Elle permet d'effacer les anciens fichiers objets ~~.o~~, les dépendances et le fichier de configuration si une configuration a déjà été faite précédemment ou si elle comporte des erreurs. Au contraire, elle n'est pas utile si vous souhaitez le garder ou si aucune configuration n'est déjà présente. Vous pouvez savoir exactement ce que fait cette commande en éditant le fichier ~~Makefile~~ à la racine des sources du noyau. Il en est de même pour les commandes suivantes.
Il s'agit ensuite de configurer votre noyau avec la commande ~~make config~~. Si vous trouvez l'aspect trop rudimentaire, vous pouvez utiliser les commandes ~~make menuconfig~~ qui offre une interface Ncurses en mode console ou make xconfig avec une interface Tcl/Tk sous XFree. La configuration consiste à choisir ce que supportera votre noyau, comme une carte graphique ou encore le système de fichiers NTFS. Nous ne détaillerons pas ici chaque option, une aide très bien faite est disponible pour chacune et nous vous conseillons de la lire avant d'activer ou désactiver une option. Chacune de ces commandes génère le fichier ~~$SRC_LINUX/.config~~ que vous pouvez éditer à la main. Une dernière commande concernant la configuration est ~~make oldconfig~~ qui modifie votre fichier de configuration pour qu'il contienne les paramètres par défaut.
Toujours pendant la configuration, vous avez la possibilité de choisir une option en tant que module. Les avantages sont d'avoir un noyau plus léger, et par conséquent moins d'espace mémoire utilisé, un démarrage du système plus rapide, et une plus grande facilité de mise à jour de votre système, notamment de vos drivers sans avoir besoin de recompiler le noyau.
Une fois la configuration terminée, vous pouvez la vérifier avec la commande ~~make checkconfig~~.
Après la configuration de votre noyau, vous devez construire les fichiers de dépendances ~~$SRC_LINUX/.hdepend~~ (à la racine des sources du kernel) et ~~$SRC_LINUX/.depend~~ (dans chaque répertoire des sources du kernel) à l'aide de la commande ~~make dep~~. Ces fichiers servent d'une part à indiquer de quels fichiers ~~include .h~~ les fichiers objets .o ont besoin, et d'autre part les informations sur les symboles que les modules exportent.
Un petit nettoyage avec la commande ~~make clean~~ efface les binaires créés lors d'une précédente compilation. Cette étape est quasi indispensable. Vient ensuite la compilation du noyau à l'aide de la commande ~~make bzImage~~ qui construit une image compressée du noyau. Nous pouvons aussi créer une image non compressée à l'aide de ~~make zImage~~, mais la taille des noyaux actuels est trop importante pour qu'un noyau non compressé soit supporté :

```
# make zImage
[...]
Root device is (3, 1)
Boot sector 512 bytes.
Setup is 4652 bytes.
System is 861 kB
System is too big. Try using bzImage or modules.
[...]
```

```
#
```

Une autre manière de compiler le noyau est d'utiliser la commande make bzdisk, qui copie le noyau sur une disquette directement.
Pour savoir ce que font toutes ces commandes qui servent à créer une image du noyau, éditez le fichier ~~$SRC_LINUX/arch/i386/boot/Makefile~~.
La compilation terminée, vous trouverez l'image du noyau bzImage dans le répertoire ~~$SRC_LINUX/arch/i386/boot~~ (dans le cas évidemment d'un processeur i386). La commande file nous indique la nature du fichier :

```
$ file bzImage
bzImage: Linux kernel x86 boot executable RO-rootFS, root_dev=0x301, Normal VGA
```

Si vous avez configuré certains modules, il est nécessaire de les compiler à l'aide de la commande ~~make modules~~.
Voilà pour la configuration et la compilation de votre noyau. Le conseil que nous pouvons vous donner est d'abord d'éliminer au maximum ce dont vous n'avez pas besoin dans le noyau. Cela l'allégera davantage. Ensuite les fonctionnalités nécessaires continuellement sont configurées nativement dans le noyau et non en tant que module. Nos modules sont plutôt les composants peu utilisés.

L'installation du noyau Linux

Comme nous l'avons dit, nous partons du principe qu'il n'y a pas d'autres noyaux compilés sur la machine. Nous n'avons donc pas à prendre de précaution quant à la sauvegarde de l'ancien noyau et des anciens modules.
L'installation consiste d'abord à copier le fichier bzImage dans le répertoire /boot et à lui donner le nom que vous voulez. Ensuite, il faut copier le fichier $SRC_LINUX/System.map dans le même répertoire que l'image du kernel. Ce fichier est le fichier de symboles du noyau, c'est-à-dire qu'il contient tous les symboles du noyau et leurs adresses respectives. Un fichier manquant ou une mauvaise version de noyau peut vous rendre dans l'impossibilité de charger un module ou vous donner ce genre d'erreur :

```
$ ps fauxwwwwwww
Warning: /boot/System.map not parseable as a System.map
{seq_startplay} {sound_timer_init}
Warning: /System.map does not match kernel data.
[...]
$
```

Il est donc indispensable d'avoir la bonne version de ce fichier dans le répertoire /boot. Si par mégarde, vous avez perdu ou effacé ce fichier, vous pouvez le reconstruire à l'aide du script suivant :

```
#!/bin/sh

PATH_VMLINUX=$1

if [ -z "$PATH_VMLINUX" ];
then
    echo "[!] usage: ./create_system_map.sh [path of vmlinux].."
    exit 0
fi

nm $PATH_VMLINUX | grep -v '\(compiled\)\|\(\.o$\)\|\( [aU] \)\|\(\.\.ng$\)\|\(LASH[RL]DI\)' | sort
> System.map
```

Le fichier vmlinux se trouve généralement à la racine des sources du noyau.
Ensuite, pour que votre système démarre sur le noyau fraîchement compilé, vous avez deux solutions. La première est d'utiliser les commandes make zlilo ou make install qui feront tout pour vous : installation du noyau et configuration du boot loader LILO. Elles font même plus que cela. Regardez le fichier ~~$SRC_LINUX/arch/i386/boot/Makefile~~ pour en savoir davantage, mais nous ne sommes pas en fait partisans des procédures automatiques et nous préférons la deuxième solution qui est de tout faire manuellement. Vous verrez cela dans le chapitre suivant.
Si vous avez configuré des modules et que vous les avez compilés, il vous reste à les installer avec la commande ~~make modules_install~~. Les modules seront copiés dans le répertoire ~~/lib/modules/version_de_votre_noyau~~ (~~/lib/modules /2.4.22~~ pour nous). Ce répertoire contient aussi le fichier ~~modules.dep~~ qui contient les dépendances entre modules. Il est recréé à chaque boot de la machine avec la commande ~~depmod -a~~.

Le Boot du Kernel

Si vous avez bien suivi la section précédente, vous devriez avoir un noyau flambant neuf. C'est bien ! Mais un noyau que l'on boote et dont on se sert, c'est encore mieux. Pour cela, nous allons tenter de découvrir les secrets du boot d'un noyau Linux. Nous verrons tout d'abord les boot loaders et aborderons ensuite la notion de ~~initrd~~ ou ramdisk. Nous passerons ensuite aux paramètres qu'il est possible de passer au noyau, juste avant qu'il ne se lance. Nous terminerons par les différents supports qui permettent à un noyau de s'amorcer.

Boot loader ou le chargement de noyau en mémoire

Qu'est-ce qu'un boot loader ? C'est un programme qui est lancé sur les architectures i386, par le BIOS, pour charger un système d'exploitation. Lorsque le PC démarre, au niveau matériel (à la mise sous tension), le BIOS prend la main. Il vérifie que tout fonctionne correctement, grâce à des tests (mémoire, CPU...), et fait l'inventaire des ressources disponibles (disques durs, lecteurs de CD-ROM, lecteurs de disquette...). Lorsque le BIOS a terminé, il donne la main à un lecteur de boot, tel que défini dans la configuration du BIOS. Le plus souvent, c'est un disque dur. Mais il est possible de configurer un lecteur de disquette, de CD-ROM, un disque USB ou la carte réseau pour booter. Le BIOS définit une liste de périphériques pour le boot. Le BIOS essaiera les périphériques définis dans cette liste, dans l'ordre, jusqu'à ce qu'il en trouve un qui boote. Selon le périphérique, la notion de boot est différente, mais pour un disque dur, il s'agit du MBR (Master Boot Record) qui contient un bout de code assurant le lancement d'un boot loader. Si le MBR ne contient rien, mais qu'une partition est estampillée avec le flag "bootable", c'est cette partition qui porte le boot loader (c'est le cas, par exemple, des systèmes d'exploitation Windows).
Mais revenons à nos moutons, ou plutôt à notre boot loader. Le BIOS charge le MBR en mémoire et l'exécute. La première partie (ou stage 1) du boot loader est alors en mémoire et s'exécute. C'est lui qui va ensuite aller chercher sur le disque la seconde partie (ou stage 2) du boot loader et charger sa configuration. Selon cette configuration, il permettra de charger en mémoire tel ou tel système d'exploitation (Windows, Linux, FreeBSD, OpenBSD, BeOS...). En résumé, le boot loader assure le lancement d'un système d'exploitation.
Sous Linux, deux boot loaders (entre autres) existent. Ils sont largement utilisés, aussi bien l'un que l'autre. Il s'agit de LILO (LInux LOader) et GRUB (Grand Unified Boot Loader). Il en existe d'autres, mais nous nous concentrerons sur ces deux-là.

LILO

Chronologiquement, LILO est plus ancien que GRUB, mais il est encore très souvent installé. Il se décompose en deux éléments : l'exécutable ~~/sbin/lilo~~ et le fichier de configuration ~~/etc/lilo.conf~~. Le but de cet article n'est pas de commenter toutes les fonctions possibles de LILO (le man devrait suffire pour cela), mais plutôt de présenter un fichier de configuration standard, et de l'expliquer, pour que le lecteur puisse acquérir les connaissances de bases pour aller, ensuite, plus loin. Cependant, les références suivantes ([4] et [5]) apporteront plus d'informations à ceux qui le souhaitent. Prenons donc un fichier de configuration de base.

```
# LILO configuration file
# /etc/lilo.conf
# Start LILO global section
# Où doit-on installer LILO
boot = /dev/hda
# rapide, mais ne fonctionne pas partout
compact
# L'utilisateur est autorisé à entrer quelque chose
prompt
# Mais il n'a que 5 secondes pour cela
timeout = 50
# Configuration à lancer par défaut
default=Linux
# End LILO global section
# Linux bootable partition config begins
# Le noyau Linux à charger
image = /boot/vmlinuz
# La partition root de notre linux
  root = /dev/hda4
# Le nom LILO de cette configuration
  label = Linux
# Le noyau Linux à charger
image = /boot/vmlinuz
# Le fichier initrd à charger avec le noyau
  initrd = /boot/initrd.img
# La partition root de notre Linux
  root = /dev/hda4
```

```
# Le nom LILO de cette configuration
  label = Linux-initrd
# Linux bootable partition config ends
# Windows bootable partition config begins
# On désire booter sur cette partition
other = /dev/hda1
# Le nom LILO de cette configuration
label = WinXP
# Pour Windows, il suffit que la partition soit bootable
  table = /dev/hda
# Windows bootable partition config ends
```

Le fichier de configuration comporte deux sections : la section globale et la section des différents systèmes d'exploitation à booter. Dans la section globale, nous définissons où doit être installé LILO. Ici, il le sera sur ~~/dev/hda~~, donc, dans le MBR. Ensuite, nous définissons que le mode de lecture du noyau sur le disque doit être compact. Cela signifie que les requêtes de lecture sont regroupées pour que le chargement du noyau en mémoire soit plus rapide. Cette option est particulièrement utile sur les périphériques lents comme les lecteurs de disquettes. Elle augmente considérablement la vitesse de chargement, même sur un disque dur. Par contre, sur ce dernier type de périphérique, cette option risque d'être incompatible avec le mode d'accès LBA32 (accès sur 32 bits pour les disques durs récents). Le mode LBA32 est le mode d'accès par défaut pour LILO. Si une incompatibilité est trouvée, pour un disque dur, la première chose à faire est de désactiver l'option ~~compact~~. Les deux paramètres suivants définissent le comportement de LILO vis-à-vis de l'utilisateur. ~~prompt~~ permet d'interagir avec LILO (pour choisir une configuration ou ajouter des paramètres à une configuration) et ~~timeout~~ définit un ~~timer~~ (en dixième de seconde) au bout duquel la configuration par défaut sera lancée. Le dernier paramètre de la section globale pointe sur la configuration par défaut à lancer, lorsque le timer est expiré.
Après la section de configuration globale, trois configurations sont définies. La première intègre le fichier du noyau Linux (image), la partition ~~root~~ de cette configuration (partition racine de la distribution) et le nom que nous donnons à cette configuration, qui apparaîtra au boot. La seconde configuration est identique à la première, exceptée qu'elle intègre la définition d'un fichier ~~initrd~~. Enfin, la dernière configuration définit un système Windows. Elle détermine quelle partition doit être bootée, le label (le nom) de cette configuration, tel qu'il apparaîtra sur l'écran de LILO et quel disque dur contient la partition à booter.
Lorsque le fichier de configuration est terminé, il ne faut pas oublier d'installer LILO. Pour cela, il suffit de lancer la commande ~~lilo -v~~. Le fichier de configuration par défaut est ~~/etc/lilo.conf~~ (le connecteur -C permet de spécifier un autre fichier). LILO lit le fichier de configuration et s'installe. Le problème majeur de LILO est qu'il nécessite de le relancer à chaque fois que le fichier est modifié. Si, par exemple, vous modifiez une configuration ou ajoutez une nouvelle configuration de boot, il ne faut pas oublier de relancer LILO. Sinon, les modifications apportées au fichier de configuration ne seront pas prises en compte.

GRUB

GRUB est arrivé après LILO. Il ne souffre pas de la limitation énoncée à la fin de la partie consacrée à LILO. Avec GRUB, il est possible de modifier le fichier de configuration, sans avoir à relancer l'installation de GRUB. GRUB lit le fichier de configuration tel qu'il existe sur le système de fichiers, à la différence de LILO qui contient le fichier de configuration codé dans son installation sur le disque. Pour GRUB, il suffit donc d'installer GRUB sur le MBR ou sur la partition voulue, une seule fois, puis de mettre à jour le fichier de configuration comme on le souhaite. De plus, GRUB offre la possibilité de créer ou modifier, directement au boot, la configuration d'un système. Typiquement, si ce que l'on vient de définir dans le fichier de configuration ne marche pas, il est possible de le modifier. Ainsi, on réussira à booter son Linux. Bien sûr, il ne faudra pas oublier de modifier le fichier de configuration, sinon au prochain reboot, on se retrouvera dans la même situation (la modification au boot d'une configuration ne met pas à jour le fichier de configuration).
Comme pour LILO, le but de cet article n'est pas de présenter toutes les options de GRUB, mais plutôt un exemple standard, pour expliquer les concepts de base, et être opérationnel assez vite. Cependant, pour plus d'informations, la consultation des pages de ~~man~~ et d'~~info~~ correspondantes, ainsi que [6], sera appréciable.
GRUB s'utilise en deux étapes. La première étape installe GRUB sur le MBR du disque principal (ou un autre endroit, selon sa configuration) en lui spécifiant le fichier de configuration à lire. La seconde étape est la création du fichier de configuration qui sera lu par GRUB lors du démarrage.
L'installation de GRUB se fait de la manière suivante. Il faut lancer GRUB par la commande grub. On se retrouve sous GRUB dans ce qui ressemble à un shell. Il faut alors dire à GRUB de s'installer. Pour cela, on prendra la commande ~~install~~ de GRUB. Nous allons définir ce qu'il faut installer et où. la syntaxe de la commande est la suivante : ~~install <stage1> <où_mettre_le_stage1> <stage2> p <chemin_fichier_configuration>~~. ~~<stage1>~~ est le fichier qui contient le code exécutable de la première partie du boot loader. Il s'agit de /boot/grub/stage1. Mais il faut spécifier à GRUB sur

quel disque et quelle partition il doit trouver ce fichier. GRUB définit les disques par hd0, hd1, hd2 qui correspondent sous Linux, respectivement, à hda, hdb, hdc. Les partitions sont définies par un nombre, en commençant par 0. Donc hda1 s'écrit, sous GRUB, (hd0,0) ; hda2 s'écrit (hd0,1) ; hdb3 s'écrit (hd1,2). Si le fichier /boot/grub/stage1 se trouve sur la partition hda1, on écrira (hd0,0)/boot/grub/stage1. Si ma partition /boot est une partition spécifique qui réside sur hda1, on écrira (hd0,0)/grub/stage1, car le répertoire /boot n'existe qu'à partir de la partition racine /. Le paramètre <où_mettre_le_stage1> définit un disque ou une partition où le stage 1 doit être installé. Si l'on veut installer GRUB sur le MBR de hda, on écrira (hd0). Le paramètre <stage2> définit le chemin d'accès (comme pour le stage 1) du stage 2. Par défaut, celui-ci se trouve dans /boot/grub/stage2. Dans notre exemple précédent, on écrira (hd0,0)/boot/grub/stage2. Il ne nous reste plus qu'à spécifier le chemin du fichier de configuration GRUB, avec la même syntaxe que les deux premiers paramètres. Si le fichier de configuration se trouve sur hda1 et qu'il se nomme /boot/grub/menu.conf, on écrira (hd0,0)/boot/grub/menu.conf. Pour résumer, la commande totale est donc :

```
grub> install (hd0,0)/boot/grub/stage1 (hd0) (hd0,0)/boot/grub/stage2
p (hd0,0)/boot/grub/menu.conf
```

Après le lancement de cette commande, on sort de GRUB, et il ne reste plus qu'à construire le fichier de configuration. Le fichier de configuration /boot/grub/menu.conf est composé de deux parties. La première partie comporte les paramètres globaux à GRUB, puis viennent les différentes configurations pour chaque système bootable. Un fichier standard de configuration GRUB ressemble à ceci (à titre de comparaison, nous allons définir la même configuration que dans le paragraphe consacrée à LILO).

```
# Timer fixé à 5 secondes
timeout 5
# Ce fichier est à afficher à l'écran de GRUB
i18n (hd0,3)/boot/grub/messages
# On définit une map clavier azerty
keytable (hd0,3)/boot/fr-latin1.klt
# La configuration par défaut est la première définie (Linux)
default 0
# Nom de la configuration
title Linux
# Définition du fichier noyau et
# de la partition racine
  kernel (hd0,3)/boot/vmlinuz root=/dev/hda4
# Nom de la configuration
title Linux-initrd
# Définition du fichier noyau et
#  de la partition racine et
#  du fichier de ramdisk
  kernel (hd0,3)/boot/vmlinuz root=/dev/hda4 initrd=/boot/initrd.img
# Nom de la configuration
title WinXP
# Partition contenant l'OS
  root (hd0,0)
# Cette partition est à activer
  makeactive
# Et à booter
  chainloader +1
```

Comme dans le fichier de configuration de LILO, le fichier de configuration de GRUB se compose de deux parties : une partie globale et une partie regroupant les configurations bootables. La partie globale définit le timer d'attente pendant lequel l'utilisateur a l'opportunité d'intervenir. Ensuite, on trouve le fichier texte comportant le message à afficher lors du choix par l'utilisateur de la configuration à booter. Le mapping du clavier à prendre en compte est ensuite donné. Comme nous avons un clavier français, on préférera un mapping Azerty français. Enfin, on a la configuration bootable par défaut, celle qui sera chargée si l'utilisateur n'intervient pas pendant le timer défini plus haut. Il s'agit de la configuration baptisée "Linux".
Après la configuration globale, on trouve les trois configurations définies dans le paragraphe LILO. La configuration "Linux" contient un nom, une référence au fichier du noyau et le paramètre passé au noyau concernant la partition root de cette distribution. Dans la configuration "Linux-initrd", on reprend la configuration précédente, à laquelle on ajoute un fichier initrd. Pour la troisième et dernière configuration, appelée "WinXP", on trouve la partition à booter et à activer. Avec ce fichier, au reboot, l'utilisateur devrait voir une fenêtre contenant ces trois configurations. Il lui est possible de se déplacer entre ces trois configurations (avec les flèches du clavier), pour choisir celle qu'il souhaite voir lancer.
Juste avant de terminer avec les boot loaders, nous souhaitons vous donner une mise en garde en ce qui concerne la sécurité. En effet, le boot loader est une partie critique de la sécurité d'un système. Dans le cas où un utilisateur est autorisé à modifier la configuration du chargement d'un système Linux, celui-ci peut être booté avec des paramètres n'assurant plus un niveau de sécurité acceptable. Le cas le plus flagrant consiste à faire booter la machine sur un

CD-ROM (sans modifier la configuration du BIOS) avec le noyau défini dans GRUB (le noyau sur le disque dur), mais en modifiant le paramètre root. Dans ce cas, on obtient un système Linux sur CD-ROM, contrôlé par l'utilisateur. Il ne lui reste plus qu'à monter la partition root du système Linux et modifier le mot de passe root. Il reboote la station et passe root sur le système grâce au mot de passe qu'il vient de modifier. Il faut donc verrouiller la configuration du boot loader d'une station, ce qui est possible, aussi bien avec LILO qu'avec GRUB, en définissant des mots de passe pour pouvoir modifier, voire même choisir une configuration à booter. Moins on donne de latitude à l'utilisateur lors du boot, plus la sécurité sera assurée. Ceci est également valable au niveau du BIOS et de sa possible modification par un utilisateur.

Initrd ou l'art des ramdisks

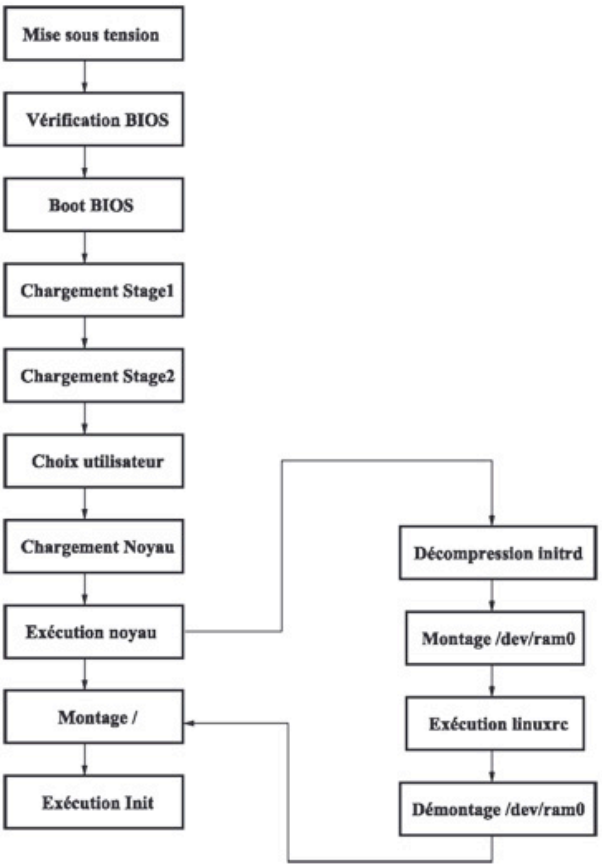
Derrière ce mot un peu barbare se cache un concept tout simple : comment faire exécuter quelque chose au noyau Linux avant qu'il lance le processus init ? Mais reprenons depuis le début : que se passe-t-il lorsque l'on boote sous Linux ?

Les différentes phases du boot d'un système Linux

Tout d'abord, le boot loader est chargé en mémoire (pour le boot loader, c'était le paragraphe précédent). Celui-ci permet de choisir un noyau à lancer et de lui passer des paramètres. Ceci fait, le noyau s'exécute, détecte le matériel, charge les différents supports matériels, charge les couches réseaux, etc. Lorsque tout est bon au niveau du matériel, il monte ensuite une partition (la partition root ou /), puis donne la main à /sbin/init. Si le noyau ne trouve pas de partition / ou de init, celui-ci nous insulte avec son célèbre cri, le fameux "kernel panic". C'est /sbin/init qui va terminer le processus de boot, en lançant les différents processus (les consoles virtuelles en mode texte), en configurant les adresses IP, en chargeant une configuration de pare-feu, en lançant les services réseaux (http, nfs, inetd...). init est le père de tous les autres processus et gère le démarrage et l'arrêt de la machine grâce aux run levels définis dans /etc/inittab et aux scripts dans /etc/rc.d et /etc/init.d.

Initrd et les phases de boot

Et où trouve-t-on initrd dans ce processus de boot ? On le trouve au moment où le noyau monte la partition root. C'est à ce moment que le noyau prend en compte le paramètre initrd passé par le boot loader. Ce paramètre est inirtd= <nom_de_fichier>. S'il est présent, le noyau doit trouver, en mémoire, un fichier chargé par le boot loader (en même temps que le noyau lui-même). Pour le noyau, ce fichier est la partition racine qu'il essaye de monter. Il décompresse le fichier dans le device /dev/ram0. Puis, le noyau essaie de monter /dev/ram0 comme partition. initrd est donc un fichier qui contient un système de fichiers compressé et qui est monté par le noyau au boot. Cette partition un peu spéciale montée, le noyau cherche le fichier /linuxrc. S'il le trouve, il l'exécute. A la fin de l'exécution de /linuxrc, le noyau démonte la partition initrd et essaie de monter la partition root définie dans le boot loader comme un des paramètres du noyau. A partir de ce moment, on revient dans la procédure de boot du noyau, au niveau du montage de la partition root et du lancement de /sbin/init. Si, dans la partition root (celle du disque dur), le répertoire /initrd existe, le système de fichiers contenu dans /dev/ram0 (donc notre initrd) est monté sous ce répertoire. Le reste de la procédure de boot se déroule normalement. Tout ceci est résumé dans la figure 1.



initrd et mkinitrd

~~initrd~~ est l'abréviation de INITial RamDisk. Un fichier ~~initrd~~ est donc un système de fichiers stocké en RAM, monté par le noyau au boot et dans lequel ont été placés des outils (commandes, bibliothèques, fichiers de configuration...) afin de réaliser des actions spécifiques au démarrage du noyau.
Mais à quoi cela sert-il ? Considérons la commande ~~mkinitrd~~ (un ~~man mkinitrd~~ donnera toutes les informations, sinon la réf. [3] vous permettra de lire le man si la commande n'est pas installée sur votre machine). Cette commande crée un fichier ~~initrd~~ à mettre en mémoire lors du boot pour pouvoir charger des modules du noyau.
Mais prenons plutôt un exemple pour éclaircir les idées. Supposons que vous possédiez un PC sous Linux, avec un disque dur SCSI. Malheureusement, votre noyau ne supporte votre carte SCSI que s'il a chargé des modules nécessaires. Cela signifie que votre noyau n'est pas capable de booter sur votre disque SCSI. Il ne pourra pas trouver votre disque pour monter la partition ~~root~~ et lancer ~~/sbin/init~~. Alors comment faire ? ~~initrd~~ est votre ami. Il suffit de créer un système de fichiers qui contient les modules de votre noyau afin de supporter votre disque SCSI et de les charger dans le noyau.
Pour reprendre la procédure de boot du système, le noyau monte le fichier initrd en tant que root, le fichier /linuxrc est un script shell qui insère les modules du noyau dans le but de gérer la carte SCSI de votre système et le disque dur qui y est attaché. Le fichier ~~/linuxrc~~ est une liste de ~~insmod <nom_du_module>~~ pour insérer les modules du noyau (modules présents dans le système de fichiers ~~initrd~~). A la fin de l'exécution de ~~/linuxrc~~, le noyau sait gérer votre disque SCSI grâce aux modules qui viennent d'être chargés. Il est alors capable de monter le système de fichiers racine présent sur votre disque dur SCSI et de continuer la procédure de démarrage sans problème.

Comment créer un ramdisk

Un fichier ~~initrd~~ est utile pour réaliser beaucoup d'autres choses. Par exemple, il est possible d'y stocker le système de fichiers nécessaire à l'installation d'un distribution Linux, ou pour toute configuration qui a besoin d'un système de fichiers résidant en mémoire.
Si on ne veut pas lancer ~~mkinitrd~~ pour créer un fichier initrd, comment faire ? Voici le principe de base pour créer un tel système de fichier. Tout d'abord, il faut s'assurer que le noyau est capable de gérer les ramdisks. Sinon il faudra le

recompiler en ajoutant cette gestion. Lors de la configuration du noyau, il faut aller dans "Block devices" et mettre "Ram disk support" à "yes", définir la taille initiale d'un disque RAM (4096 octets par défaut) selon la taille du fichier initrd (lorsqu'il est décompressé) et ajouter le support initrd en mettant "Initial RAM disk (initrd) support" à "yes".
Lorsque l'on est sûr que le noyau gère le système de fichiers initrd, il faut créer un fichier qui contiendra le système de fichiers par la commande dd if=/dev/zero of=<fichier> bs=512 count=<taille_du_fichier>. Cette commande crée le fichier <fichier> dont la taille sera <taille_du_fichier>*512 octets.
Lorsque le fichier est créé, il faut créer un système de fichiers dans celui-ci grâce à la commande mkfs -t ext2 <fichier>. Cette commande crée un système de fichiers de type Ext2 dans le fichier <fichier>.
Ensuite, on monte le système de fichiers, fraîchement créé, dans un répertoire local avec la commande mount -t ext2 -o loop <fichier> /mnt/initrd. Cette commande monte le système de fichiers stocké dans <fichier>, de type Ext2, dans le répertorie /mnt/initrd.
A partir de là, il suffit de mettre dans le répertoire /mnt/initrd tous les outils, bibliothèques... que vous souhaitez voir dans votre initrd. Pour commencer, il faut recréer une arborescence standard avec /bin, /sbin, /lib, /etc, /usr... Puis il faut y copier les différents composants systèmes nécessaires à ce que vous voulez faire. La commande /usr/bin/ldd est votre amie, puisqu'elle permet de trouver les bibliothèques nécessaires à un exécutable. Il ne faut pas oublier le fichier /linuxrc qui initiera toute la procédure.
Il suffit alors de démonter le système de fichiers (umount /mnt/hd) et de compresser le fichier par la commande gzip -9 <fichier>. Votre fichier initrd <fichier> est prêt à l'emploi.
Il faut ajouter une entrée dans le fichier de configuration /etc/lilo.conf, par exemple, en spécifiant la ligne initrd= <fichier> pour le bloc d'un noyau (ne pas oublier de lancer lilo -v, pour prendre en compte vos modifications). Et voilà, au prochain démarrage, votre noyau bootera avec un fichier initrd.

Paramètres du noyau

Le noyau Linux comporte des paramètres. Ces paramètres sont passés par le boot loader lors du chargement du noyau. C'est au moment de l'exécution du noyau que celui-ci regarde ces paramètres et modifie son comportement selon eux. Il suffit de regarder les logs du noyau lors du boot (par la commande /bin/dmesg par exemple) pour vérifier la liste des paramètres. On aura une ligne comme celle-ci : Kernel command line: BOOT_IMAGE=Linux ro root=306. On y trouve le nom de la configuration du boot loader (Linux), la définition de la partition racine (root=306) et le fait que cette partition doit être montée en lecture seule (c'est la procédure de démarrage gérée par init qui la remontera en lecture/écriture).
La liste des paramètres qu'il est possible de passer au noyau est assez grande et est définie dans le fichier /usr/src/linux/Documentation/kernel-parameters.txt. Les paramètres intéressants sont les suivants (cette liste n'est pas exhaustive) :

- hdc=ide-scsi : définit que le périphérique IDE hdc doit être traité comme un périphérique SCSI (permet l'utilisation de cdrecord sur un graveur IDE) ;
- init : donne l'exécutable à lancer après avoir monter la partition racine ;
- initrd : définit le fichier contenant le système de fichiers ramdisk ;
- mem : donne la taille de la mémoire physique lorsque le noyau n'est pas capable de la détecter entièrement ;
- nfsroot : donne le système de fichiers NFS root que le noyau doit monter avant de lancer init ;
- noinitrd : définit qu'il ne faut pas prendre en compte de ramdisk, même si un ramdisk a été fourni ;
- panic : définit ce que doit faire le noyau en cas de kernel panic ;
- ramdisk_size : donne la taille d'un ramdisk ;
- ro : définit que la partition racine doit être montée en lecture seule ;
- root : donne la partition racine à monter ;
- rootflags : définit les paramètres pour monter la partition racine ;
- rootfstype : donne le type (Ext3, Ext2, Minix...) de partition racine ;
- rw : définit que la partition racine doit être montée en lecture/écriture ;
- S : définit que init doit être lancé en mode single user ;
- vga : donne le mode vidéo à configurer.

Les autres paramètres du noyau concernent principalement la configuration de drivers matériels tels que les cartes sonores, les cartes SCSI... D'autres paramètres concernent la gestion du bus PCI ou ISA et la configuration de carte ISA plug and play.
Mais comment passer des paramètres au noyau ? Il suffit de penser aux boot loaders. Pour LILO, le paramètre append (à insérer dans la configuration d'un système bootable) définit une chaîne de caractères qui sera passée au noyau. Cette chaîne de caractères sera considérée par le noyau comme une liste de paramètres. Pour GRUB, on rajoutera simplement la liste des paramètres à passer au noyau sur la ligne kernel. Par exemple, la ligne suivante kernel (hd0,3)/boot/vmlinuz root=/dev/hda4 ro initrd=/boot/initrd.img S vga=711 passera les paramètres root, ro, initrd, S et vga au noyau,

directement.

Supports de Boot

Désormais, nous savons réaliser un ramdisk, configurer deux boot loaders et passer des paramètres au noyau. Et si on essayait de booter un Linux à partir d'autres choses qu'un disque dur ! Quels sont les supports physiques qui permettent de booter un système Linux ? En fait, il y en a beaucoup. Nous n'en prendrons que quelques-uns pour illustrer cet article : une disquette, un CD-ROM, une clé USB de type DiskOnKey et une carte réseau compatible PXE.

Booter depuis une disquette

Booter un système Linux depuis une disquette peut ne pas être très différent par rapport à un disque dur, excepté au niveau de l'espace disponible. Cependant, trois méthodes existent : la méthode classique du boot loader de type LILO ou GRUB, la méthode plus spécifique pour la disquette du SYSLINUX sur un système de fichiers FAT et enfin la méthode un peu brutale de la copie du noyau sur la disquette.
Il est possible de voir une disquette comme un disque dur de petite taille. Un boot loader pourra donc répondre à ce besoin. La configuration reprend ce que nous avons montré dans les paragraphes précédents. La disquette doit posséder un système de fichiers (Ext2, Minix, FAT...), sur lequel on copie les fichiers nécessaires au boot (noyau, ~~initrd~~, messages...). Cette disquette sera montée dans un répertoire. On crée un fichier de configuration, par exemple pour LILO. Ce fichier de configuration comportera une section globale spécifiant l'installation de LILO sur la disquette, puis un (ou des) configuration(s) de boot pointant sur les fichiers de la disquette. On installe LILO avec la commande ~~lilo -v~~ (ne pas oublier le ~~-C~~ pour définir le bon fichier de configuration). On vérifie avec les messages de sortie que tout est bon et on reboote sur cette disquette.
La deuxième méthode se sert de SYSLINUX. Des informations complémentaires sont disponibles sur le site Internet dédié [7]. SYSLINUX est un boot loader pour Linux, fonctionnant avec des systèmes de fichiers FAT. L'intérêt de SYSLINUX réside dans le fait qu'il suffit de créer une disquette au format FAT, d'y installer SYSLINUX, de copier les fichiers de boot et de créer un fichier de configuration définissant les systèmes à booter. Ceci fait, la disquette devient un support de boot pour ces systèmes. La philosophie ressemble un peu à GRUB, sans la partie interactive. Par contre, la disquette reste une disquette FAT. De plus, les options de configuration de SYSLINUX sont assez nombreuses et permettent de faire ce que l'on veut.
La troisième et dernière méthode n'est disponible que si le noyau Linux à booter le permet. Cette méthode consiste à copier directement le noyau sur la disquette par une commande de type cp bzImage /dev/fd0 (on notera que l'on n'a absolument pas monté la disquette dans un répertoire local). Lors de la création du noyau, un message précise si le noyau n'est pas utilisable avec cette méthode. On trouve ce message à la fin de la compilation du noyau (par un make bzImage). Pour un noyau non compatible avec cette méthode, on aura (c'est l'avant-dernière ligne qui est intéressante) :

```
[...]
Root device is (3, 6)
Boot sector 512 bytes.
Setup is 4764 bytes.
System is 998 kB
warning: kernel is too big for standalone boot from floppy
make[1]: Leaving directory `/usr/src/linux-2.4.22/arch/i386/boot'
```

Si le ~~warning~~ n'est pas présent, le noyau est utilisable avec cette méthode. Il est impossible, par cette méthode, de définir des configurations particulières, des paramètres, un ramdisk. Elle servira seulement à booter un noyau, point final. Les paramètres possibles sont ceux stockés dans le noyau, modifiables par la commande ~~rdev~~. Cette méthode est peu paramétrable, mais sera utile pour tester rapidement si un noyau répond aux besoins.

Booter depuis un CD-ROM

Le boot depuis un CD-ROM est possible grâce à deux techniques différentes. La première technique reprend la notion de CD-ROM bootable de type El-Torito. La seconde technique se sert d'un outil spécifique, nommé ISOLINUX, qui fait partie de la suite SYSLINUX dont nous venons de parler. Des informations complémentaires sur les CD-ROM El-Torito et ISOLINUX sont disponibles en [8], [9] et [10].
Un CD-ROM El-Torito est un CD bootable. Un CD-ROM bootable émule une disquette, dont le contenu est stocké dans un fichier. Deux fichiers spécifiques sont nécessaires, le fichier de boot et le fichier catalogue qui pointe sur le premier fichier. Le principe de création d'un CD-ROM bootable est simple. Il faut tout d'abord créer une disquette bootable (les trois méthodes définies dans le paragraphe précédent sont possibles). On transfère le contenu de cette disquette dans un fichier. Une commande de type ~~dd if=/dev/fd0 of=<fichier_boot>~~ est suffisante. Ensuite, il ne reste

plus qu'à créer une image de CD-ROM de type El-Torito. Pour cela, on lance la commande mkisofs avec les connecteurs -b <fichier_boot> -c <fichier_catalogue>. Le fichier de catalogue est créé par la commande mkisofs. Il ne reste plus qu'à graver l'image de CD-ROM avec la commande cdrecord et on obtient un CD-ROM El-Torito avec lequel on bootera sa machine.
La seconde technique impose que l'on se serve de ISOLINUX. Cet outil est un boot loader pour Linux sur architecture i386 qui sert à booter depuis des systèmes de fichiers ISO9660 (norme des CD-ROM), sans recourir au mode émulation. Il n'est plus nécessaire de devoir créer une disquette de boot, comme dans la technique précédente, et à la stocker dans un fichier d'émulation d'image disque. En fait, l'exécutable de ISOLINUX remplace le fichier d'émulation d'image disque. Puis, c'est cet exécutable qui lira le fichier de configuration ISOLINUX contenu sur le CD-ROM et affichera les messages à l'écran pour que l'utilisateur puisse choisir la configuration à booter. On utilise toujours les connecteurs -b et -c lors de la création de l'image du CD-ROM par mkisofs, mais, dans ce cas, ce sera toujours -b isolinux/isolinux.bin. Le fichier de configuration isolinux.cfg sera lu au boot du CD-ROM. Cette méthode est très répandue pour les distributions Linux, lors de l'installation par CD-ROM.

Booter depuis une clé USB

Pour les machines qui supportent le boot depuis un périphérique USB, le fait de booter depuis ce type de périphérique ou un autre, ne change rien. L'accès à ce périphérique (que ce soit un disque dur, un lecteur de disquette, un lecteur de CD-ROM ou une clé USB) est transparent, la couche USB étant gérée par le BIOS. Ceci signifie que si l'on boote depuis un lecteur de CD-ROM USB ou IDE, il n'y a aucune différence, et ce que nous venons de dire dans le paragraphe précédent, pour les CD-ROM, est toujours valable. En revanche, la clé USB de type DiskOnKey est spécifique à cette interface. Y a-t-il des spécificités à connaître pour mettre en œuvre ce type de périphérique ?
La réponse est simple : non. Il n'y a aucune différence. Voici le principe dont nous nous sommes servis pour faire booter un portable IBM X31 sur une clé USB. Nous avons connecté le périphérique sur un port USB. Ce périphérique a été reconnu comme un dispositif SCSI en tant que /dev/sda. Nous avons créé une partition et un système de fichiers sur cette partition (fdisk /dev/sda et mkfs -t ext3 /dev/sda1). Nous avons monté cette partition (mount -t ext3 /dev/sda1 /mnt/usb) et copié les fichiers nécessaires sur cette partition. Nous avons créé une configuration LILO pour ce périphérique avec l'installation dans le MBR (/dev/sda), mais avec les paramètres suivants, dans la configuration globale :

```
disk=/dev/sda
bios=0x80
```

Lors du boot, le périphérique USB sera vu comme le premier disque de boot du BIOS (0x80), alors que ce n'est pas le cas actuellement, au moment de l'installation de LILO. On lance LILO (lilo -v -C /mnt/usb/etc/lilo.conf), on démonte le périphérique (umount /mnt/usb) et on reboote. On configure dans le BIOS du portable que l'on souhaite booter sur un périphérique USB, et le portable boote sur le périphérique USB, afin, par exemple, d'installer une distribution Linux à partir de ce périphérique.

Booter depuis une carte réseau

De plus en plus de PC possèdent une carte réseau et parmi ces cartes réseau, de plus en plus sont compatibles avec la spécification PXE (Pre-Execution Environment) d'Intel. Mais qu'est-ce que c'est qu'une carte réseau PXE ? C'est une carte réseau qui possède une ROM conforme à la spécification PXE afin de booter la machine par le réseau. Comment tout cela fonctionne-t-il ?
Le BIOS définit la carte réseau comme un périphérique de boot ; celle-ci émet une requête BOOTP sur le réseau. Un serveur BOOTP répond à cette requête (selon l'adresse MAC de la carte) en donnant les informations de configuration IP de la carte réseau (adresse IP, masque de réseau, passerelle, DNS...). Le serveur BOOTP transmet aussi un nom de fichier à télécharger via le protocole TFTP. Lorsque le client a configuré sa carte réseau, il récupère le fichier via TFTP sur le serveur BOOTP. Ce fichier correspond à un stage 2 au niveau d'un boot loader.
Ce fichier fait partie du projet SYSLINUX, dont nous avons parlé dans les paragraphes sur la disquette et le CD-ROM, et plus particulièrement de PXELINUX ([11]). Le fichier se nomme pxelinux.0 et il est placé dans le répertoire /tftpboot du serveur. Dans ce répertoire, un répertoire doit être créé. Il s'agit de pxelinux.cfg. Il contient le fichier default définissant une configuration de boot, comme on a pu en voir avec LILO.
Lorsque pxelinux.0 est chargé par le client, celui-ci charge le fichier de configuration de boot /tftpboot/pxelinux.cfg/default. Ce fichier contient le nom du fichier de noyau Linux, les paramètres de boot et, le cas échéant, un fichier initrd. Selon cette configuration, le client télécharge, toujours par TFTP, ces fichiers (qui doivent se trouver dans le répertoire /tftpboot du serveur) et lance le noyau comme le ferait LILO. A partir de ce moment, la procédure de boot redevient la même que pour un amorçage à partir d'un disque dur.

Cette méthode de boot offre la possibilité d'installer des PC sous Linux sur un réseau à grande échelle. Il suffit de les bancher sur le réseau, de booter via la carte réseau et, l'installation et la configuration de la station peuvent être entièrement automatisées. La station boot lance un script qui crée les partitions sur le disque dur et les systèmes de fichiers, installe les packages qu'elle récupère sur le réseau, configure la station, met en place un boot loader et reboote la station à partir du disque dur.

Comment faire cohabiter plusieurs versions de noyau

Nous avons vu jusqu'à présent la manière de compiler un noyau, de l'installer, de le faire booter sur divers supports physiques, etc. Le problème qui se pose maintenant est de pouvoir compiler et installer un nouveau noyau (dans le cas d'un test, ou parce que vous avez besoin d'une fonctionnalité bien spécifique), tout en gardant votre ancien noyau. Il est même conseillé de garder l'ancien noyau au cas où le nouveau ne fonctionnerait pas au boot du système pour une raison quelconque.
Nous allons donc reprendre pratiquement chaque étape de la compilation et de l'installation, et nous verrons à chaque fois les précautions à prendre.

Au moment de la configuration

Deux cas de figure se présentent. Le premier est que vous modifiez quelques options dans la version actuelle du noyau. Dans ce cas-là, nous vous conseillons de sauvegarder le fichier de configuration ~~.config~~ pour pouvoir éventuellement revenir en arrière. Le second cas de figure est que vous vous voulez installer un nouveau noyau. Dans ce cas-là, aucune sauvegarde n'est à faire.
Un autre conseil est de configurer la variable ~~EXTRAVERSION~~ dans le fichier ~~$SRC_LINUX/Makefile~~ :

```
# cat /usr/src/linux-2.4.22/Makefile
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 22
EXTRAVERSION =

KERNELRELEASE=$(VERSION).$(PATCHLEVEL).$(SUBLEVEL)$(EXTRAVERSION)
[...]
```

L'idée par exemple est de mettre comme valeur la date de compilation de votre noyau. Nous verrons plus tard pourquoi il est utile de configurer cette variable.

Au moment de l'installation

Rappelez-vous, l'installation consiste à copier les fichiers ~~$SRC_LINUX/System.map~~ et ~~$SRC_LINUX/arch/i386/bzImage~~ dans le répertoire ~~/boot~~ et à faire un make modules_install pour installer les modules compilés.
En ce qui concerne les deux fichiers à copier, il est évident qu'il faut les renommer pour chaque version différente de noyau. Parfois, certains utilitaires ont besoin du fichier ~~/boot/System.map~~ ; le mieux est de créer un lien symbolique avec le fichier du noyau le plus utilisé. Voici à quoi peut alors ressembler votre répertoire ~~/boot~~ au final :

```
# cat /etc/lilo.conf
[...]
image=/boot/bzImage-2.4.22-wlan
        label=main
        read-only

image=/boot/bzImage-2.4.22-sspax
        label=sspax
        read-only

image=/boot/bzImage-2.4.22-test
        label=test
        read-only
[...]
```

Il reste à installer les modules. C'est maintenant que la variable $EXTRAVERSION est utile. D'ordinaire, la commande make modules_install installe les modules dans le répertoire ~~/lib/modules/x.y.z~~. C'est gênant si vous avez décidé de modifier certaines options de la même version du noyau, car ils seront copiés au même endroit que ceux de votre précédente installation. Certes, vous pouvez faire une sauvegarde du répertoire avant. Mais à chaque fois que vous

voudrez démarrer sur une version différente de noyau, il faudra auparavant faire une restauration du répertoire correspondant à la version de votre noyau. Avouez que cela est assez contraignant. Par contre, le fait d'avoir configuré la variable ~~$EXTRAVERSION~~ fera que les modules seront copiés dans le répertoire ~~/lib/modules /x.y.z$EXTRAVERSION~~, et le noyau aura pour nom ~~bzImage-2.4.22$EXTRAVERSION~~ si vous faites un uname -a (évidemment si le fichier ~~/boot/bzImage-2.4.22~~ est présent). Au démarrage du système, le noyau saura à quel endroit aller chercher les modules.
Au moment de la configuration du boot et de initrd
Gérer plusieurs noyaux avec les boot loaders est assez facile. Nous avons vu, que ce soit avec GRUB ou avec LILO, que leur configuration contenait une partie concernant le boot du système avec le nom d'image du noyau, la racine root de cette configuration, et le nom de la configuration. Donc, si nous voulons gérer plusieurs noyaux et avoir le choix de choisir celui que nous voulons au démarrage du système, il suffit de faire un copier-coller de ces lignes et de modifier les paramètres nécessaires. Nous relançons ensuite /sbin/lilo et le tour est joué. Voici un exemple concret avec le fichier lilo.conf :

```
# cat /etc/lilo.conf
[...]
image=/boot/bzImage-2.4.22-wlan
        label=main
        read-only

image=/boot/bzImage-2.4.22-sspax
        label=sspax
        read-only

image=/boot/bzImage-2.4.22-test
        label=test
        rea
```

Si, pour chaque configuration, nous voulons aussi charger un fichier initrd, il suffit de lui donner un nom différent pour chaque version de noyau (il est préférable de toujours utiliser la convention ~~nom_de_fichier-x.y.z~~) et d'inclure, comme nous avons vu dans le paragraphe précédent, la ligne adéquate dans les fichiers de configuration des boot loaders.

Conclusion

Au terme de cet article (et plus généralement, des deux derniers hors séries de Linux Magazine France), nous espérons avoir quelque peu démystifié le noyau Linux, cette couche basse qui supporte tout le reste. La compilation du noyau, son lancement, son paramétrage, l'utilisation des ramdisks et des périphériques de boot ne devraient plus vous poser de problèmes. Nous espérons également vous avoir montré que les capacités du noyau Linux, en termes de paramétrage et d'adaptation à des besoins spécifiques, sont assez incroyables. Il est, en effet, possible de se construire un noyau Linux sur mesure, pour la ou les fonction(s) dont on a besoin et seulement celles-ci. De plus, il est possible de disposer d'un grand nombre de périphériques pour toute une palette de systèmes Linux différents. Bienvenue dans le monde de la "customization" de noyau Linux !!!

Bibliographie

[1] "Patcher les sources de son noyau" - Cédric Blancher, Linux Magazine HS Kernel Linux n°1
[2] "Linux: histoire d'un noyau" - Christophe Blaess, Linux Magazine HS Kernel Linux n°1
[3] http://www.rt.com/man/mkinitrd.8.html
[4] http://www.freenix.fr/unix/linux/HOWTO/mini/LILO.html
[5] http://gd.tuwien.ac.at/opsys/linux/lilo/Version21_docs/tech.pdf
[6] http://www.linuxgazette.com/issue64/kohli.html
[7] http://syslinux.zytor.com/index.php
[8] http://www.phoenix.com/resources/specs-cdrom.pdf
[9] http://www.tldp.org/HOWTO/Bootdisk-HOWTO/cd-roms.html
[10] http://syslinux.zytor.com/iso.php
[11] http://syslinux.zytor.com/pxe.php

~~Retrouvez cet article dans :~~ Linux Magazine Hors série 17

Posté par (La rédaction) | Signature : Frédéric Combeau, Samuel "zorgon" Dralet | Article paru dans 

Laissez une réponse

Vous devez avoir ouvert une session pour écrire un commentaire.

« Précédent Aller au contenu »

Identifiez-vous
Inscription
S'abonner à UNIX Garden

Articles de 1ère page

- La légalité du spam : une question ouverte
- Pear et les librairies PHP
- FreeDOS
- Mise en œuvre d'une passerelle Internet sous Linux
- Les pseudo-classes en CSS
- GNU/Linux Magazine N°108 - Septembre 2008 - Chez votre marchand de journaux
- Linux Pratique N°49 - Septembre/Octobre 2008 - Chez votre marchand de journaux
- Donner du style à son texte : utiliser les lettrines
- Installer un serveur Syslog
- Quatre serveurs FTP hyper sécurisés avec vsftpd

Actuellement en kiosque :

Il y a actuellement

- 719 articles/billets en ligne.

Recherche

Catégories

- Administration réseau
- Administration système

- Agenda-Interview
- Audio-vidéo
- Bureautique
- Comprendre
- Distribution
- Embarqué
- Environnement de bureau
- Graphisme
- Jeux
- Matériel
- News
- Programmation
- Réfléchir
- Sécurité
- Utilitaires
- Web

Archives

- août 2008
- juillet 2008
- juin 2008
- mai 2008
- avril 2008
- mars 2008
- février 2008
- janvier 2008
- décembre 2007
- novembre 2007
- février 2007

GNU/Linux Magazine

- Ce billet est hors ligne montrer/masquer GNU/Linux Magazine 108 - Septembre 2008 - Chez votre marchand de journaux
- Edito : GNU/Linux Magazine 108
- GNU/Linux Magazine HS 38 - Septembre/Octobre 2008 - Chez votre marchand de journaux
- Edito : GNU/Linux Magazine HS 38
- GNU/Linux Magazine 107 - Juillet/Août 2008 - Chez votre marchand de journaux

GNU/Linux Pratique

- Linux Pratique N°49 -Septembre/Octobre 2008 - Chez votre marchand de journaux
- Edito : Linux Pratique N°49
- À télécharger : Les fichiers du Cahier Web de Linux Pratique n°49
- Linux Pratique Essentiel N°3 - Août/Septembre 2008 - Chez votre marchand de journaux
- Edito : Linux Pratique Essentiel N°3

MISC Magazine

- Misc 38 : Codes Malicieux, quoi de neuf ? - Juillet/Août 2008 - Chez votre marchand de journaux
- Edito : Misc 38
- Références de l'article « Détection de malware par analyse système » d'Arnaud Pilon paru dans MISC 38
- Références de l'article « La sécurité des communications vocales (3) : techniques numériques » d'Éric Filiol paru dans MISC 38
- Misc 37 : Déni de service - Mai/Juin 2008 - Chez votre marchand de journaux