De la mise sous tension à la ligne de commande de Bash

Greg O'Keefe, gcokeefe@postoffice.utas.edu.au v0.7, April 2000

Voici une description rapide de ce qui se passe dans un système Linux, depuis l'instant où vous mettez celui-ci sous tension, jusqu'au moment où vous vous loguez et obtenez la ligne de commande de bash (NDT : Bourne Again SHell). Ce document est organisé "par lots" pour faciliter la tâche des gens qui veulent mettre en place un système à partir du code source. Comprendre cela vous sera très utile lorsque vous aurez besoin de résoudre des problèmes ou de configurer votre système (version française par Dominique van den Broeck, Mai 2000, *dvandenbroeck@free.fr* <mailto:dvandenbroeck@free.fr>).

Table des matières

1 Introduction **3**

2 Partie matérielle (Hardware) **4**

2.1 Configuration 4

2.2 Exercices 4

2.3 Plus d'informations. 5

3 Lilo **5**

3.1 Configuration 5

3.2 Exercices 5

3.3 Plus d'informations 6

4 Le noyau Linux **6**

4.1 Configuration 7

4.2 Exercices 7

4.3 Plus d'informations 7

5 La bibliothèque C de GNU **8**

5.1 Configuration 8

5.2 Exercices 8

5.3 Plus d'informations 9

6 Init **9**

6.1 Configuration 10

6.2 Exercices 10

6.3 Plus d'informations. 10

7 Le système de fichiers (filesystem) 10

7.1 Configuration . . . 11

7.2 Exercices . . . 11

7.3 Plus d'informations . . . 11

8 Démons noyau 12

8.1 Configuration . . . 13

8.2 Exercices . . . 13

8.3 Plus d'informations . . . 13

9 Le journal système (System Logger) 13

9.1 Configuration . . . 13

9.2 Exercices . . . 14

9.3 Plus d'informations . . . 14

10 Getty et Login 14

10.1 Configuration . . . 14

10.2 Exercices . . . 14

11 Bash 14

11.1 Configuration . . . 15

11.2 Exercices . . . 15

11.3 Plus d'informations . . . 15

12 Les commandes 16

13 Construire un système Linux minimum à partir des sources. 16

13.1 Ce qu'il vous faut . . . 16

13.2 Le système de fichier (Filesystem) . . . 18

13.3 MAKEDEV . . . 18

13.4 Le noyau (kernel) . . . 19

13.5 Lilo . . . 19

13.6 Glibc . . . 20

13.7 SysVinit . . . 21

13.8 Ncurses . . . 21

13.9 Bash . . . 21

13.10Util-linux (getty et login) . . . 22

13.11Sh-utils . . . 22

13.12Vers l'utilisabilité . . . 22

13.13Astuces diverses 23
13.14Plus d'informations 23

14 Conclusion **23**

15 Section administrative **23**

15.1 Copyright 23
15.2 Page principale 24
15.3 Retours 24
15.4 Références et remerciements. 24
15.5 Historique des changements 24
15.5.1 0.6 -> 0.7 24
15.5.2 0.5 -> 0.6 25
15.6 A faire (TODO) 25

1 Introduction

Je trouve frustrant qu'il se passe dans ma machine Linux des choses que je ne comprends pas. Si, comme moi, vous souhaitez vraiment comprendre votre système plutôt que simplement savoir comment l'utiliser, ce document devrait être un bon point de départ. Ce genre de "connaissance de fond" est aussi requis si vous voulez devenir un as de la résolution de problèmes sous Linux.

Je pars du principe que vous avez une machine Linux en état de marche, et que vous maîtrisez les bases d'Unix et de l'architecture matérielle des PC. Si ce n'est pas le cas, *The Unix and Internet Fundamentals HOWTO* <http://www.linuxdoc.org/HOWTO/Unix-and-Internet-Fundamentals-HOWTO.html> ("Les notions fondamentales d'Unix et Internet", miroirs français en *VO* <http://www.linuxfr.org/LDP/HOWTO/Unix-and-Internet-Fundamentals-HOWTO.html> et *VF* <http://www.freenix.org/unix/linux/HOWTO/Unix-Internet-Fundamentals-HOWTO.html>) est un excellent endroit pour débuter. C'est un site concis, lisible, et qui couvre toutes les bases.

Le sujet principal de ce document est la façon dont Linux démarre. Mais il se veut également être une ressource d'apprentissage plus large. J'ai inclus des exercices dans chaque section. Si vous en faites vraiment quelques uns, vous apprendrez bien plus qu'en vous contentant de lire ce document.

Il y a aussi des liens vers du code source à télécharger, car j'espère que certains lecteurs s'attaqueront au meilleur exercice d'apprentissage de Linux que je connaisse : construire un système à partir du code source. Giambattista Vico, un philosophe italien (1668-1744) disait “verum ipsum factum” ce qui signifie "la compréhension découle de l'expérience" (NDT : Traduction libre). Merci à Alex (voir 15.4 (Remerciements)) pour cette citation.

Si vous souhaitez vous "la rouler vous-même", je vous conseille d'aller voir *Linux From Scratch HOWTO* <http://www.linuxfromscratch.org> (LFS soit "Linux depuis zéro"). LFS fournit des instructions détaillées pour bâtir un système complet et exploitable à partir du code source. Sur le site web de LFS, vous trouverez aussi une mailing-list (liste de diffusion) à disposition des personnes qui construisent de tels systèmes. Ce que j'ai inclus dans ce document, ce sont des instructions pour construire un système "jouet", (voir 13 (Bâtir un système Linux minimal à partir des sources)) purement pour l'exercice.

Les lots (NDT : "packages" en anglais et repris ci-après) sont présentés dans l'ordre où ils apparaissent dans le processus de démarrage du système. Cela signifie que si vous installez les packages dans cet ordre vous

pouvez redémarrer après chaque installation, et voir à chaque fois le système se rapprocher petit à petit de l'état où il vous donnera la ligne de commande. Il y a une notion de progression rassurante dans cela.

Je vous recommande de commencer par lire le texte principal de chaque section, en ignorant les exercices et références, puis de décider du point jusqu'auquel vous souhaitez comprendre votre système. Reprenez alors depuis le début, en faisant les exercices et en relisant en détail.

2 Partie matérielle (Hardware)

Lorsque vous allumez votre ordinateur, celui-ci se teste lui-même pour s'assurer que tous ses composants sont en état de marche. Cela s'appelle le "Power On Self Test" (POST ou Auto-Test à l'Allumage). Ensuite, un programme nommé Bootstrap loader (Chargeur de boot), situé dans le BIOS en ROM, recherche un secteur de boot, ou secteur d'amorce. Un secteur d'amorce est le premier secteur d'un disque et contient un petit programme capable de charger un système d'exploitation. Les secteurs d'amorce sont marqués par un "nombre magique" (valeur fixe caractéristique) 0xAA55 = 43603 à l'octet 0x1FE = 510. Ce sont les deux derniers octets du secteur. C'est de cette façon que l'électronique peut déterminer s'il s'agit d'un secteur d'amorce ou pas.

Le bootstrap loader a une liste d'endroits où chercher un secteur d'amorce. Ma vieille machine regarde d'abord sur le lecteur de disquette, puis sur le disque dur. Les machines modernes peuvent aussi rechercher un secteur d'amorce sur un CD-ROM. S'il trouve un secteur amorçable ("bootable"), il le charge en mémoire et passe ainsi le contrôle au programme qui charge le système d'exploitation en mémoire. Sur un système Linux typique, ce programme sera la première étape du chargeur de LILO. Il existe malgré tout plusieurs manières différentes de faire démarrer ("booter") votre système. Voir le *Guide de l'utilisateur de LILO* pour plus de détails. Voir la section 3.3 (LILO) pour l'URL.

Evidement, il y a bien plus à dire sur ce que fait le hardware du PC. Mais ce n'est pas l'objet de ce document. Lisez un des nombreux livres traitant de l'architecture matérielle du PC.

2.1 Configuration

La machine stocke des informations sur son propre état dans son CMOS. Cela inclut la RAM et les types de disques installés dans le système. Le BIOS de la machine contient un programme, Setup, qui vous permet de modifier ces informations. Pour savoir comment y accéder, regardez attentivement les messages qui apparaissent sur votre écran lorsque vous mettez votre machine sous tension. Sur ma machine, il faut appuyer sur la touche DEL (Suppr) avant qu'elle ne commence à charger le système d'exploitation.

2.2 Exercices

Une bonne façon d'en apprendre plus sur le hardware d'un PC est de monter une machine à partir de composants d'occasion. Prenez au moins un 386 pour pouvoir y installer Linux facilement. Cela ne vous coûtera pas très cher. Posez la question autour de vous, quelqu'un pourrait bien vous donner une partie des pièces qu'il vous faut.

Allez voir *Unios* `<http://learning.taslug.org.au/resources>`, (Ils avaient une page sur `<http://www.unios.org>`, mais elle a disparu) et téléchargez, compilez et fabriquez votre disquette bootable. Ce n'est qu'un programme bootable affichant "Hello World !", contenant à peine plus de 100 lignes d'assembleur. Il serait intéressant de voir à le convertir en un format exploitable par l'assembleur GNU `as`.

Ouvrez l'image de la disquette bootable pour unios avec un éditeur hexadécimal. Cette image fait 512 octets de long. Exactement la longueur d'un secteur. Trouvez-y le nombre magique 0xAA55. Faites la même chose pour une disquette bootable de votre propre ordinateur.

Vous pouvez utiliser la commande `dd` pour la copier dans un fichier : `dd if=/dev/fd0 of=boot.sector`. Faites *très* attention à paramétrer `if` (fichier source) et `of` (fichier destination) comme il faut !

Essayez d'en extraire le code source du chargeur de LILO.

2.3 Plus d'informations.

- *Les notions fondamentales d'Unix et d'Internet* <http://www.linuxdoc.org/HOWTO/Unix-and-Internet-Fundamentals-HOWTO.html>, par Eric S. Raymond, miroirs français en *VO* <http://www.linuxfr.org/LDP/HOWTO/Unix-and-Internet-Fundamentals-HOWTO.html> et *VF* <http://www.freenix.org/unix/linux/HOWTO/Unix-Internet-Fundamentals-HOWTO.html>, et particulièrement la section 3, *Que se passe-t-il lorsque vous allumez un ordinateur ?*
- Le premier chapitre du *Guide de l'utilisateur de LILO* donne une excellente explication des partitions de disques sur PC et de l'amorçage (booting). Voir la section 3.3 (LILO) pour l'URL.
- Le NOUVEAU *Peter Norton Programmer's Guide to the IBM PC & PS/2* (Guide Peter Norton du programmeur pour l'IBM PC et PS/2), par Peter Norton et Richard Wilton, Microsoft Press, 1988. Il existe un nouveau livre Norton, qui a l'air bien, mais que je ne peux m'offrir pour le moment.
- Un des nombreux ouvrages disponibles sur la manière de faire évoluer (upgrader) son PC.

3 Lilo

Lorsque l'ordinateur charge le secteur d'amorce d'un système sous Linux normal, ce qu'il charge est en fait un partie de LILO, appelée "first stage boot loader" (chargeur de boot de premier niveau). Il s'agit d'un mini programme dont la seule tâche est de charger et d'exécuter le "second stage boot loader" (chargeur de boot de deuxième niveau).

Le "second stage loader" vous donne une invite de commande (s'il a été installé de cette manière) et charge le système d'exploitation de votre choix.

Lorsque votre système est monté et en état de marche, et que vous exécutez `lilo`, ce que vous exécutez en réalité est le "map installer". Celui-ci lit le fichier de configuration `/etc/lilo.conf` et écrit le chargeur de boot sur le disque dur, avec les informations concernant les systèmes d'exploitation qu'il peut charger.

Il y a de nombreuses manières de rendre votre système bootable. Celle que je viens de décrire est la manière la plus évidente et "normale", au moins pour une machine dont le système d'exploitation principal est Linux. Le Guide de l'Utilisateur Lilo explique plusieurs exemples de "boot concepts". Cela vaut la peine de les lire, et d'en essayer quelques uns.

3.1 Configuration

Le fichier de configuration de Lilo est `/etc/lilo.conf`. Il existe une page de manuel (man page) à son sujet : tapez `man lilo.conf` dans un shell pour l'afficher. La principale caractéristique de `lilo.conf` est qu'il existe une entrée pour chaque chose que Lilo doit booter. Pour une entrée Linux, cela inclut l'endroit où se trouve le kernel, et la partition à monter à la racine du système de fichier ("filesystem"). Pour les autres systèmes, la principale information est la partition sur laquelle booter.

3.2 Exercices

DANGER : soyez prudent avec ces exercices. Il est assez facile de faire une erreur quelque part et de bloquer votre "master boot record" (NDT : ou MBR, premier secteur du disque dur, contient le bootloader et la table des partitions) et de rendre ainsi votre système inutilisable. Assurez-vous que vous avez une disquette

de réparation qui fonctionne, et que vous savez comment vous en servir pour remettre les choses en état. Voir plus bas un lien vers tomsrtbt, la disquette de réparation que j'utilise et recommande. La meilleure des précautions est d'utiliser une machine qui ne contienne pas de données sensibles.

Installez Lilo sur une disquette. Peu importe s'il n'y a rien d'autre sur la disquette que le kernel - vous obtiendrez un "kernel panic" quand le kernel sera prêt a charger init, mais au moins vous saurez que Lilo fonctionne.

Si vous le souhaitez, vous pouvez essayer de voir jusqu'à quel point vous pouvez faire tenir un système sur une disquette. C'est sûrement la deuxième meilleure activité pour apprendre Linux. Voir le Bootdisk HOWTO (url plus bas), et tomsrtbt (url plus bas) pour avoir des pistes.

Configurez Lilo afin qu'il lance unios (voir section 2.2 (exercices hardware) pour une URL). Comme défi supplémentaire, voyez si vous pouvez le faire sur une disquette.

Faites une boucle de boots. Configurez le Lilo du Master Boot Record pour qu'il boote le Lilo du secteur de boot d'une des partitions principales, puis configurez ce Lilo pour qu'il relance celui du MBR. Ou alors utilisez le MBR et vos quatre partitions principales pour faire une boucle en cinq points ! Marrant !

3.3 Plus d'informations

- La page de man de Lilo
- Le package Lilo (voir 13.1 (downloads)) contient le "LILO User's Guide" `lilo-u-21.ps.gz` (ou plus récent). Il se peut que vous ayez quand même déjà ce document. Regardez dans `/usr/doc/lilo` ou approché. La version postscript est meilleure que la version en texte brut, car elle contient des diagrammes et des tables.
- *tomsrtbt* `<http://www.toms.net/rb>` La disquette unique linux la plus cool ! Constitue une excellente disquette de secours.
- *Bootdisk-HOWTO* `<http://www.linuxdoc.org/HOWTO/Bootdisk-HOWTO/>` ("HOWTO Disquette-de-boot", miroirs français en *VO* `<http://www.linuxfr.org/LDP/HOWTO/Bootdisk-HOWTO>` et *VF* `<http://www.freenix.org/unix/linux/HOWTO/Bootdisk-HOWTO.html>`.

4 Le noyau Linux

Le noyau (kernel) fait vraiment beaucoup de choses. Je pense qu'une bonne manière de résumer tout cela est de dire qu'il fait faire au hardware ce que les programmes veulent, proprement et efficacement.

Le processeur ne peut exécuter qu'une seule instruction à la fois, mais Linux semble faire tourner beaucoup de choses simultanément. Le noyau accomplit cela en sautant de tâche en tâche très rapidement. Il fait le meilleur usage possible du processeur en gardant trace des processus qui sont prêts à être exécutés et de ceux qui attendent quelque chose comme un enregistrement en provenance d'un disque, ou une saisie clavier quelquonque. Cette tâche du noyau est appelée "scheduling" (planification).

Si un programme ne fait rien, alors il n'a pas besoin d'être en RAM. Même un programme qui fait quelque chose peut avoir certaines parties inactives, qui donc n'ont pas besoin d'être en RAM. L'espace adressable est divisé en pages. Le noyau garde une trace des pages les plus utilisées. Les pages qui ne sont pas autant utilisées peuvent être déplacées dans la partition d'échange (swap). Lorsqu'une page est à nouveau sollicitée, une autre page inutilisée est retirée de l'espace adressable pour lui faire de la place. Cela s'appelle la gestion de la mémoire virtuelle.

Si vous avez un jour compilé votre propre noyau, vous avez remarqué qu'il y a un grand nombre d'options pour des périphériques spécifiques. Le noyau contient beaucoup de code spécifique pour converser avec l'électronique de tout type, et les présente d'une façon propre et uniforme aux programmes applicatifs.

Le noyau prend aussi en charge la gestion des fichiers, les communications interprocessus, et beaucoup du travail concernant le réseau.

Une fois le noyau chargé, la première chose qu'il fait est de rechercher un programme appelé `init` et l'exécuter.

4.1 Configuration

La majorité de la configuration du noyau est effectuée quand vous le construisez, en utilisant `make menuconfig`, ou `make xconfig` dans le répertoire `/usr/src/linux/` (ou là ou se trouvent les sources de votre noyau Linux). Vous pouvez réinitialiser le mode vidéo par défaut, la racine du système de fichiers, le périphérique de swap et la taille du disque virtuel (RAM disk) en utilisant `rdev`. Ces paramètres ainsi que d'autres peuvent aussi être passés au noyau depuis Lilo. Vous pouvez indiquer à Lilo les paramètres à passer au noyau soit dans lilo.conf, soit à l'invite (prompt) de Lilo. Par exemple, si vous souhaitiez utiliser hda3 comme racine du système de fichiers plutôt que hda2, vous pourriez taper :

```
LILO: linux root=/dev/hda3
```

Si vous mettez en place un système à partir de ses sources, vous pouvez vous simplifier la vie en créant un noyau "monolithique", c'est-à-dire sans module. Vous n'aurez donc pas à copier ceux-ci sur le système cible.

NOTE : Le fichier `System.map` est utilisé par le logger (le journal système, qui enregistre les messages de service) du noyau pour déterminer les noms des modules générant des messages. Le programme `top` utilise également ces informations. Lorsque vous copiez le noyau vers un système cible, copiez aussi `System.map`.

4.2 Exercices

Réfléchissez à ceci : `/dev/hda3` est un type de fichier spécial qui décrit une partition d'un disque dur. Mais il vit sur le système de fichiers comme tous les autres fichiers. Le noyau veut savoir quelle partition monter à la racine - donc il n'a pas encore de système de fichiers. Alors comme peut-il lire `/dev/hda3` pour trouver la partition à monter ?

Si vous ne l'avez pas encore fait, compilez votre noyau. Lisez l'aide (Help) pour chaque option.

Essayez de voir jusqu'à quel point vous pouvez réduire la taille de votre noyau avant qu'il ne cesse de fonctionner. Vous pouvez apprendre beaucoup en écartant les parties qui ne sont pas nécessaires.

Lisez "The Linux Kernel" (URL plus bas) et ce faisant, trouvez les parties du source auxquelles il se réfère. Le livre (au moment où j'écris ces lignes) se réfère au noyau version 2.0.33, qui devient franchement dépassé. Il pourrait être plus facile de suivre si vous téléchargiez cette ancienne version et y lisiez le source. Il est très excitant de trouver des morceaux de code C appelés "process" et "page".

Programmez ! Faites des essais ! Voyez si vous pouvez faire cracher au noyau des messages supplémentaires ou quelque chose du même genre.

4.3 Plus d'informations

- `/usr/src/linux/README` et le contenu de `/usr/src/linux/Documentation/` (Ces emplacements peuvent varier sur votre système)
- Le *Kernel-HOWTO* <`http://mirror.aarnet.edu.au/linux/LDP/HOWTO/Kernel-HOWTO.html`> (miroirs français en *VO* <`http://www.linuxfr.org/LDP/HOWTO/Kernel-HOWTO.html`> et *VF* <`http://www.freenix.org/unix/linux/HOWTO/Kernel-HOWTO.html`>).
- L'aide disponible quand vous configurez un noyau en utilisant `make menuconfig` ou `make xconfig`
- *The Linux Kernel (et autres guides du LDP)* <`http://mirror.aarnet.edu.au/linux/LDP/LDP/`> (miroir français en *VO* <`http://www.linuxfr.org/LDP/LDP`>)

– Téléchargement des sources voir 13.1 (téléchargements)

5 La bibliothèque C de GNU

L'étape suivante qui se produit au démarrage de votre ordinateur est le chargement d'init et son exécution. Cependant, init, comme la plupart des programmes, utilise des fonctions issues de bibliothèques.

Vous avez peut-être déjà vu un exemple de programme C comme celui-ci :

```
main() {
        printf("Hello World!\n");
}
```

Le programme ne définit nullement `printf`, alors d'où vient-il? Il provient des bibliothèques C standard ("standard C libraries"), sur un système GNU/Linux, glibc. Si vous les compilez sous Visual C++, alors il provient d'une implémentation Microsoft de ces mêmes fonctions standard. Il existe des milliers de ces fonctions standard, pour les mathématiques (math), la gestion des chaînes de caractères (string), l'heure et la date, l'allocation de mémoire et ainsi de suite. Tout, dans Unix (y compris Linux) est soit écrit en C, soit tente de le simuler, de sorte que tous les programmes utilisent ces fonctions.

Si vous jetez un oeil dans `/lib` sur votre système Linux, vous verrez un grand nombre de fichiers appelés `libquelquechose.so` ou `libquelquechose.a` etc. Ce sont les bibliothèques de ces fonctions. Glibc est simplement l'implémentation GNU de ces fonctions.

Les programmes peuvent utiliser ces fonctions de deux manières. si vous "linkez" (NDT : "éditez les liens", opération qui consiste à établir les relations avec les différents objets sollicités par le programme, puis à créer l'exécutable) *statiquement*, ces fonctions seront copiées à l'intérieur de l'exécutable généré. C'est à ça que servent les bibliothèques `libquelquechose.a`. Si vous linkez votre programme *dynamiquement* (effectué par défaut), lorsque le programme aura besoin du code d'une bibliothèque, il ira l'extraire directement du fichier `libquelquechose.so`

La commande `ldd` vous apporte une aide précieuse lorsque vous cherchez à retrouver les bibliothèques utilisées par un programme particulier. Par exemple, voici les bibliothèques utilisées par `bash` :

```
[greg@Curry power2bash]$ ldd /bin/bash
        libtermcap.so.2 => /lib/libtermcap.so.2 (0x40019000)
        libc.so.6 => /lib/libc.so.6 (0x4001d000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

5.1 Configuration

Certaines fonctions, dans leur bibliothèque, dépendent du lieu où vous vous trouvez. Par exemple, en français, on écrit les dates sous la forme dd/mm/yy, mais les américains les écrivent mm/dd/yy. Il existe un programme livré avec `glibc` appelé `localdef` qui vous donne la possibilité de définir cela.

5.2 Exercices

Utilisez `ldd` pour déterminer les bibliothèques qu'utilise votre application préférée.

Utilisez `ldd` pour déterminer les bibliothèques utilisées par `init`.

Créez une bibliothèque gadget, avec seulement une ou deux fonctions dedans. On utilise le programme `ar` pour les créer. La page de manuel d'`ar` pourrait être un bon point de départ pour commencer à enquêter

sur la manière dont cette opération est effectuée. Ecrivez, compilez, et linkez un programme utilisant cette bibliothèque.

5.3 Plus d'informations

– Code source, voir la section 13.1 (téléchargements)

6 Init

Je ne parlerai que du style d'initialisation "System V" que les systèmes Linux utilisent le plus souvent. Il existe des alternatives. En fait, vous pouvez mettre n'importe quel programme dans `/sbin/init`, que le noyau exécutera lorsqu'il aura fini de se charger.

Le travail d'`init` est de faire en sorte que tout se lance comme il faut. Il vérifie que les systèmes de fichier sont en bon état et les monte. Il démarre les démons ("daemons") qui enregistrent les messages système, gèrent le réseau, distribuent les pages web, écoutent les signaux de la souris, etc. Init démarre aussi les processus `getty` qui vous donnent l'invite de login sur vos terminaux virtuels.

Il y a un processus compliqué concernant le changement de niveau d'exécution ("run-levels"), mais je vais sauter tout ça, et ne parler que du démarrage du système.

Init lit le fichier `/etc/inittab`, qui lui dit quoi faire. Typiquement, la première chose demandée est l'exécution d'un script d'initialisation. Le programme qui exécute (ou interprète) ce script est `bash`, le même programme qui vous donne la ligne de commande. Sur les systèmes Debian, le script d'initialisation est `/etc/init.d/rcS`, sur Red Hat, `/etc/rc.d/rc.sysinit`. C'est là que les systèmes de fichiers sont vérifiés puis montés, l'horloge mise à l'heure, le fichier ou la partition d'échange (swap) activés, les noms de machines définis, etc.

Ensuite, un autre script est invoqué pour nous placer dans le niveau d'exécution par défaut. Cela implique simplement le démarrage d'un ensemble de sous-systèmes. Il existe un ensemble de sous-répertoires `/etc/rc.d/rc0.d`, `/etc/rc.d/rc1.d`, ..., `/etc/rc.d/rc6.d` sous Red Hat, ou `/etc/rc0.d`, `/etc/rc1.d`, ..., `/etc/rc6.d` sous Debian, correspondant aux run-levels. Si nous entrons dans le niveau d'exécution 3 sur un système Debian, le script exécute tous les scripts de `/etc/rc3.d` commençant par 'S' (pour Start). Ces scripts sont en réalité des liens vers un autre répertoire appelé généralement `init.d`.

Donc, le script de notre niveau d'exécution est appelé par `init`, et recherche dans un répertoire des scripts dont le nom débute par la lettre 'S'. Il se peut qu'il tombe sur `S10syslog` en premier. Les chiffres indiquent au script du niveau l'ordre dans lequel il doit les lancer. En l'occurence, `S10syslog` est lancé en premier parce qu'il n'y pas de script commençant par S00 ... S09. Mais S10syslog est en fait un lien vers `/etc/init.d/syslog` qui est un script chargé du démarrage et de l'arrêt du system logger (enregistreur de messages systèmes). Parce que le lien commence par un 'S', le script du run-level sait qu'il doit exécuter le script `syslog` avec le paramètre "start". Il y a aussi des liens dont le nom débute par 'K' (pour Kill), qui spécifient ce qu'il faut arrêter, et dans quel ordre, lorsque l'on entre dans le niveau d'exécution.

Pour changer ce que le sous-système lance par défaut, vous devez configurer ces liens dans le répertoire `rcN.d`, ou N est le niveau d'exécution par défaut défini dans votre fichier `inittab`.

La dernière chose importante qu'effectue init est de démarrer les `gettys` [NdRel :en pratique, souvent des `mingettys`)]. Ceux-ci sont ressuscités ("respawned"), ce qui signifie qu'ils sont automatiquement relancés par `init` s'ils viennent à se terminer. La plupart des distributions fournissent six terminaux virtuels. Il se peut que vous souhaitiez en enlever pour économiser de la mémoire, ou en ajouter pour pouvoir lancer beaucoup de choses à la fois, et passer rapidement de l'un à l'autre. Vous pourriez aussi avoir besoin de lancer un `getty` vers un terminal texte ou vers un modem. Vous devez alors éditer `inittab`.

6.1 Configuration

`/etc/inittab`, qui est le fichier de configuration au sommet de la "hiérarchie" des fichiers de configuration.

Les répertoires `rcN.d`, où N = 0, 1,..., 6 détermine les sous-systèmes à lancer.

Quelque part dans les scripts invoqués par init, se trouve la commande `mount -a`. Cela signifie : "Monte tous les systèmes de fichiers censés être montés". Le fichier `/etc/fstab` définit ce qui est censé être monté. Si vous souhaitez changer ce qui est monté par défaut au démarrage, c'est le fichier que vous devez modifier. Il existe une page de manuel pour `fstab`.

6.2 Exercices

Trouvez le répertoire `rcN.d` du niveau d'exécution par défaut de votre système puis faites un `ls -l` pour voir les fichiers pointés par les liens.

Changez le nombre de gettys tournant sur votre système.

Retirez tous les sous-systèmes dont vous n'avez pas besoin de votre niveau d'exécution par défaut.

Essayez de déterminer le minimum nécessaire pour démarrer.

Fabriquez une disquette avec Lilo, un noyau et un programme statique affichant "Bonjour tout le monde!" nommé `/sbin/init`, puis regardez-la démarrer et dire bonjour.

Regardez attentivement votre système démarrer, et notez les événements signalés. Ou imprimez une section de votre journal système `/var/log/messages` à partir du moment où votre système a démarré. Ensuite, en partant d'`inittab`, explorez tous les scripts et essayez de voir quel code fait quoi. Vous pouvez également ajouter des messages, comme

```
echo "Hello, moi c'est rc.sysinit"
```

C'est aussi un bon exercice pour apprendre le langage de script de Bash, certains scripts étant assez compliqués. Ayez un bon guide de bash à portée de main (NDT : "man bash" devrait suffire, faute de mieux).

6.3 Plus d'informations.

- Voir 13.1 (téléchargements) pour télécharger le code source.
- Il y a des pages de manuel pour les fichiers `inittab` et `fstab`. Tapez (par exemple) `man inittab` dans un shell pour l'afficher.
- Le Guide de L'Administrateur Système Linux contient une *section* <http://mirror.aarnet.edu.au/linux/LDP/LDP/> intéressante concernant init (miroir français en *VO* <http://www.linuxfr.org/LDP/LDP>).

7 Le système de fichiers (filesystem)

Dans cette section, j'emploierai l'expression "système de fichiers" pour deux notions différentes. Il y a les systèmes de fichiers installés sur des partitions de disque ou d'autres périphériques, et il y a le système de fichier (NdRel : la hiérarchie) tel qu'il vous est présenté par un système Linux en état de marche. Sous Linux, vous "montez" le système de fichiers d'un disque sur le système de fichiers du système.

Dans la section précédente, j'ai mentionné le fait que des scripts d'initialisation vérifiaient et montaient les systèmes de fichiers. Les commandes qui effectuent ces opérations sont respectivement `fsck` et `mount`.

Un disque dur n'est qu'un grand espace dans lequel vous pouvez écrire des zéros et des uns. Un système de fichiers impose une structure à tout cela, et le présente sous la forme de fichiers, à l'intérieur de sous-répertoires, à l'intérieur de répertoires... Chaque fichier est répresenté par un inode, indiquant le fichier dont il s'agit, la date de sa création, et où trouver son contenu. Les répertoires sont aussi représentés par des inodes, mais ceux-ci indiquent où trouver les inodes des fichiers que les repértoires contiennent. Si le système veut lire `/home/greg/bigboobs.jpeg`, il commence par lire l'inode du répertoire racine `/` dans le "superblock", puis trouve l'inode du répertoire `home` dans le contenu de `/`, puis trouve l'inode du répertoire `greg` dans le contenu de `home`, et enfin l'inode de `bigboobs.jpeg` qui lui dira quel bloc du disque il faut lire.

Si nous ajoutons des données à la fin d'un fichier, il peut arriver que les données soient écrites avant que l'inode ne soit mis à jour (indiquant que le nouveau bloc appartient désormais au fichier), ou vice-versa. Si le courant est coupé à cet instant précis, le système de fichiers sera "cassé". C'est ce genre de chose que `fsck` essaie de détecter et de réparer.

La commande mount prend le système de fichiers d'un périphérique, et l'ajoute à la hiérarchie de fichiers de votre système. En général le noyau monte son système de fichiers racine en lecture seule (read-only). La commande mount est ensuite utilisée pour le remonter en lecture-écriture (read-write) après que fsck ait vérifié que tout soit en ordre.

Linux prend aussi en charge d'autres types de systèmes de fichiers : msdos, vfat, minix, etc. Les détails d'un système de fichiers spécifique sont masqués par le Système de Fichier Virtuel (Virtual File System ou VFS) qui est une couche d'abstraction. Je ne rentrerai pas dans ces détails. Il existe une discussion sur ce sujet dans "Le Kernel Linux" (voir la section 4.3 (Le Noyau Linux) pour l'URL)

7.1 Configuration

Il existe des paramètres à la commande `mke2fs`, chargée de la création d'un système de fichiers de type ext2. Ils contrôlent la taille des blocs, le nombre d'inodes, etc. Voir la man page de `mke2fs` pour plus de détails.

Ce qui doit être monté sur votre système de fichiers est contrôlé par le fichier `/etc/fstab`, qui a lui aussi sa page de manuel.

7.2 Exercices

Fabriquez un tout petit système de fichiers, et visualisez-le avec un éditeur hexadécimal. Identifiez les inodes, les superblocs, et le contenu des fichiers.

Je crois qu'il existe des outils qui vous donnent une vue graphique d'un système de fichiers. Trouvez-en un, essayez-le, et envoyez moi l'url par email avec vos appréciations !

Explorez le code du système de fichiers ext2 dans le noyau.

7.3 Plus d'informations

- Le chapitre 9 du livre "Le noyau linux" du LDP donne une excellente description des systèmes de fichiers. Vous pouvez le trouver sur le site LDP : miroir *français* <`http://www.linuxfr.org/LDP/LDP/`> ou *australien* <`http://mirror.aarnet.edu.au/linux/LDP/LDP/`>).
- La commande `mount` fait partie du package util-linux, il y a un lien vers celui-ci dans 13.1 (téléchargements).
- Les pages de manuel de `mount`, `fstab`, `fsck` et `mke2fs`
- La home page des EXT2 File System Utilities *ext2fsprogs* <`http://web.mit.edu/tytso/www/linux/e2fsprogs.html`> et son miroir australien *ext2fsprogs* <`ftp://mirror.aarnet.edu.au/pub/linux/metalab/system/filesystems/ext2/`>. On y trouve aussi une vue d'ensemble d'Ext2fs, bien qu'elle ne soit plus à jour, et moins lisible que le chapitre 9 du "Noyau Linux"

– *Unix File System Standard* <ftp://tsx-11.mit.edu/pub/linux/docs/linux-standards/fsstnd/> Un autre *lien* <http://www.pathname.com/fhs/> vers le standard des systèmes de fichiers d'Unix. Ce document décrit où doit se trouver quoi, dans un système Unix, et pourquoi. Il indique aussi le minimum nécessaire à placer dans `/bin`, `/sbin`, etc. C'est une bonne référence si votre objectif est un système minimal mais complet.

8 Démons noyau

Malheureusement, cette section contient plus de questions et de conjectures que de faits. Peut-être pouvez-vous apporter votre pierre ?

Si vous saisissez la commande `ps aux`, vous verrez quelque chose ressemblant à ce qui suit :

```
USER       PID %CPU %MEM  SIZE   RSS TTY STAT START   TIME COMMAND
root         1  0.1  8.0  1284   536   ? S    07:37   0:04 init [2]
root         2  0.0  0.0     0     0   ? SW   07:37   0:00 (kflushd)
root         3  0.0  0.0     0     0   ? SW   07:37   0:00 (kupdate)
root         4  0.0  0.0     0     0   ? SW   07:37   0:00 (kpiod)
root         5  0.0  0.0     0     0   ? SW   07:37   0:00 (kswapd)
root        52  0.0 10.7  1552   716   ? S    07:38   0:01 syslogd -m 0
root        54  0.0  7.1  1276   480   ? S    07:38   0:00 klogd
root        56  0.3 17.3  2232  1156   1 S    07:38   0:13 -bash
root        57  0.0  7.1  1272   480   2 S    07:38   0:01 /sbin/agetty 38400 tt
root        64  0.1  7.2  1272   484  S1 S    08:16   0:01 /sbin/agetty -L ttyS1
root        70  0.0 10.6  1472   708   1 R   Sep 11   0:01 ps aux
```

C'est une liste de processus en cours d'exécution sur le système. Remarquez que `init` est le processus numéro un. Les processus 2, 3,4 et 5 sont kflushd, kupdate, kpiod et kswapd. Il y a quand même quelque chose d'étrange : dans les deux colonnes de la taille virtuelle de stockage (SIZE) et la taille réelle de stockage (Real Storage Size, RSS), ces processus renvoient zéro. Comment un processus peut-il ne pas utiliser de mémoire ? En réalité, ces processus font partie du noyau. Le noyau, lui, n'apparaît pas du tout sur la liste des processus, et vous ne pouvez définir la mémoire utilisée qu'en soustrayant la mémoire disponible du total installé sur votre système. Les parenthèses autour du nom de commande pourraient signifier qu'il s'agit d'un processus noyau (?)

`kswapd` déplace des parties d'un programme non utilisées à un instant donné de l'espace de stockage réel (c'est-à-dire la RAM) vers l'espace de swap (c'est-à-dire le disque dur). `kflushd` écrit les données des buffers vers le disque. Ceci permet aux choses d'aller plus vite. Ce que les programmes écrivent peut être conservé en mémoire, dans un buffer, puis écrit sur le disque par morceaux plus larges et de façon plus efficace (NDT : principe du cache). Je ne sais pas à quoi servent `kupdate` et `kpiod`.

C'est ici que finissent mes connaissances. Que font ces deux derniers démons ? Pourquoi les démons noyau ont-ils des numéros de processus explicites plutôt que d'être simplement des bouts de code anonymes ? Est-ce qu'init les lance effectivement, ou tournent-ils déjà lorsque init apparaît ?

J'ai mis dans `/sbin/init` un script pour monter `/proc` et faire un `ps aux`. Le processus 1 était le script lui-même, et les processus 2, 3, 4 et 5 étaient les démons noyau, juste au dessous du vrai init. Le noyau les a forcément mis là, car ce n'est sûrement pas mon script qui l'a fait.

Les hypothèses suivantes viennent de David Leadbeater :

Ces processus semblent s'occuper des accès disque, ils semblent être lancés par le noyau, mais après avoir lancé le processus init. Il semble qu'en étant lancés comme des processus noyau plutôt que comme des

processus séparés, ils soient protégés et ne peuvent être tués (kill -9 ne les arrête pas), je ne sais pas vraiment pourquoi ils sont lancés en tant que threads séparés. Il semble y avoir un rapport avec les accès disque.

kflushd et kupdate Ces deux processus sont démarrés pour vider les buffers "sales" (c'est-à-dire modifiés) vers le disque. kflushd est lancé quand les buffers sont pleins et kupdate se lance périodiquement (5 secondes ?) pour synchroniser les disques avec les buffers en mémoire.

kpiod et kswapd Ils ont pour tâche l'exportation de pages (sections) de mémoire vers le fichier d'échange de façon à ce que la mémoire principale ne soit jamais remplie, ils ressemblent à kflushd et kupdate dans le sens où l'un est lancé en cas de besoin (kpiod) et l'autre (kswapd) est lancé périodiquement (à intervalles d'une seconde).

Autres démons du noyau

Sur une installation par défaut de RedHat 6, kupdate a disparu mais udpate tourne en tant que démon dans l'espace utilisateur, et il semble qu'il ait besoin d'être lancé. Il y a également un autre démon, mdrecoveryd, qui lui semble d'occuper du RAID. En regardant dans les sources du noyau, il semble que certains pilotes SCSI lancent eux aussi des processus séparés.

Je ne connais toujours pas la signification des parenthèses, mais elles semblent apparaître quand la RSS d'un processus est à zéro, ce qui signifie qu'il n'utilise pas de mémoire (?).

(fin de citation, merci David)

8.1 Configuration

Je n'ai pas connaissance d'une configuration de ces démons noyau.

8.2 Exercices

Trouvez à quoi servent ces processus, comment ils fonctionnent, et écrivez une nouvelle section 'Démons Noyau" pour ce document, et envoyez-la moi !

8.3 Plus d'informations

"Le Kernel Linux" du Projet de Documentation Linux (LDP). (voir section 4.3 (Le noyau Linux) pour l'url), et les sources du noyau, c'est tout ce que je vois pour le moment.

9 Le journal système (System Logger)

Init démarre les démons `syslogd` et `klogd`. Ils écrivent les messages à consigner dans le journal. Les messages du noyau sont pris en main par `klogd`, alors que `syslogd` gère les messages des autres processus. Le fichier journal principal est `/var/log/messages`. C'est un bon endroit où aller voir quand quelque chose tourne mal dans votre système. Vous y trouverez souvent de précieux indices.

9.1 Configuration

Le fichier `/etc/syslog.conf` indique au logger où mettre quels messages. Les messages sont identifiés par le service dont ils proviennent, et leur niveau de priorité. Ce fichier de configuration est constitué de lignes indiquant que les messages du service x avec une priorité y vont vers z, où z est un fichier, un terminal, une imprimante, une machine distante, ou autre chose encore.

NOTE : Syslog a besoin que le fichier `/etc/services` existe. Ce fichier alloue des ports (NDT : Ports des protocoles TCP et UDP). Je ne sais pas vraiment si syslog a besoin d'un port réservé pour faire de l'enregistrement de messages à distance, ou si même l'enregistrement local se fait à travers un port, ou même s'il se contente d'utiliser `/etc/services` pour convertir les noms de services indiqués dans `/etc/syslog.conf` en numéros de port.

9.2 Exercices

Jetez un oeil à votre journal système. Prenez un message que vous ne comprenez pas, et essayez de trouver ce qu'il signifie.

Redirigez tous les messages du journal vers un terminal. (Revenez à la normale une fois que c'est fait).

9.3 Plus d'informations

Le *miroir* `<http://mirror.aarnet.edu.au/pub/linux/metalab/system/daemons/>` australien de sysklogd.

10 Getty et Login

Getty est le programme qui vous permet de vous connecter à travers un périphérique série, comme une console virtuelle, un terminal en mode texte, ou un modem. Il affiche l'invite de login. Une fois que vous avez saisi votre nom d'utilisateur, getty le transmet à `login`, qui vous demande un mot de passe, le vérifie, puis vous donne le shell.

Il existe plusieurs getty disponibles. Certaines distributions, comme Red Hat, en utilisent un très petit appelé `mingetty` et qui ne gère que les terminaux virtuels.

Le programme `login` fait partie du package util-linux, qui contient aussi un getty nommé `agetty`, qui fonctionne bien. Ce package contient également `mkswap`, `fdisk`, `passwd`, `kill`, `setterm`, `mount`, `swapon`, `rdev`, `renice`, `more` et bien d'autres.

10.1 Configuration

Le message qui apparaît en haut de votre écran avec l'invite de login provient du fichier `/etc/issue`. Les getty sont en général démarrés depuis `/etc/inittab`. Login recherche les détails spécifiques à l'utilisateur dans `/etc/passwd`, et si vous avez le shadowing (une protection des mots de passe), dans `/etc/shadow`.

10.2 Exercices

Créez un fichier `/etc/passwd` à la main. Les mots de passe peuvent être nuls, puis changés avec le programme `passwd` une fois connecté. Voir la page de manuel de ce fichier. Utilisez `man 5 passwd` pour obtenir la page de manuel du fichier plutôt que celle du programme.

11 Bash

Si vous donnez à `login` une combinaison valide de nom d'utilisateur et de mot de passe, il ira regarder dans `/etc/passwd` pour savoir quel shell vous donner. La plupart du temps, dans un système Linux, ce sera `bash`.

Le travail de `bash` consiste à lire vos commandes et voir ce sur quoi elles agissent. C'est à la fois une interface utilisateur, et l'interpréteur d'un langage de programmation.

Dans son rôle d'interface, il lit vos commandes, et les exécute lui-même si ces commandes sont "internes", comme `cd`, ou bien trouve et exécute un programme s'il s'agit de commandes "externes" comme `cp` ou `startx`. Bash propose également plusieurs options fort sympathiques comme un historique des commandes, ou le complètement automatique des noms de fichiers (avec la touche de tabulation).

Nous avons déjà vu `bash` à l'action dans son rôle de langage de programmation. Les scripts qu'`init` lance pour démarrer le système sont généralement des scripts shell, et sont exécutés par `bash`. Avoir un langage de programmation propre, parallèlement aux utilitaires systèmes disponibles depuis la ligne de commande forme une combinaison très puissante, si vous savez ce que vous faites. Par exemple ("séquence frime" !) j'ai eu besoin l'autre jour d'appliquer une pile entière de correctifs (patches) à un répertoire de codes source. J'ai été capable de le faire en une seule commande, la suivante :

```
for f in /home/greg/sh-utils-1.16*.patch; do patch -p0 < $f; done;
```

Ceci recherche tous les fichiers de mon répertoire personnel dont les noms commencent par `sh-utils-1.16` et finissent par `.patch`, puis affecte un par un ces noms à la variable `f` et exécute les commandes invoquées entre `do` et `done`. Il y avait en l'occurence 11 correctifs, mais il aurait pu aussi bien y en avoir 3000.

11.1 Configuration

Le fichier `/etc/profile` agit sur le comportement de bash au niveau du système entier. Ce que vous mettez dans ce fichier affectera toute personne qui utilise bash sur votre système. Cela sert par exemple à ajouter des répertoires dans la variable `PATH`, ou à définir celui de la variable `MAIL`.

Le comportement par défaut du clavier laisse souvent à désirer. En fait, c'est readline qui contrôle cela. Readline est un package distinct qui prend en main les interfaces de ligne de commande, en fournissant l'historique des commandes, et le complètement automatique de noms de fichiers (NDT : Touche TAB sous Bash), tout comme les facilités évoluées d'édition de ligne. Il est compilé dans bash. Par défaut, Readline est configuré à l'aide du fichier `.inputrc`, dans votre répertoire personnel. La variable INPUTRC peut être utilisée pour outrepasser les règles de ce fichier pour le bash. Par exemple, dans Red Hat 6, `INPUTRC` reçoit la valeur `/etc/inputrc` dans le fichier `/etc/profile`. Cela signifie que les touches Retour Arrière (Backspace), Suppr (Delete), Début (Home) et Fin (End) fonctionnent correctement et pour tout le monde.

Une fois que bash a lu le fichier de configuration général, commun au système entier, il recherche votre fichier de configuration personnel. Il teste l'existence des fichiers `.bash_profile`, `.bash_login` et `.profile` dans votre répertoire personnel. Il lance le premier qu'il trouve. Si vous voulez modifier le comportement de bash à votre égard, sans le changer pour les autres, faites-le ici. Par exemple, de nombreuses applications utilisent les variables d'environnement pour contrôler leur fonctionnement. J'ai une variable `EDITOR` contenant la valeur `vi` pour pouvoir utiliser vi sous Midnight Commander (un excellent gestionnaire de fichier orienté console) au lieu de son éditeur propre.

11.2 Exercices

Les bases de bash sont faciles à apprendre. Mais ne vous y limitez pas : on peut aller incroyablement loin avec. Prenez l'habitude de rechercher de meilleures façons de faire les choses.

Lisez des scripts shell, analysez les choses que vous ne comprenez pas.

11.3 Plus d'informations

– Téléchargement du code source, voir 13.1 (téléchargements)

– Il existe un “Manuel de Référence de Bash” avec, clair, mais assez lourd.
– Il existe un livre O'Reilly sur le Bash, je ne sais pas s'il est vraiment bon.
– Je ne connais pas de tutoriel bash gratuit et à jour. Si vous en connaissez un, merci de m'envoyer l'URL.

12 Les commandes

Vous effectuez la plupart des choses sous bash en saisissant des commandes comme `cp`. La majorité de ces commandes sont des petits programmes, bien que quelques unes, comme `cd` soient intégrées au shell.

Les commandes viennent de packages, la plupart de la Free Software Foundation (projet GNU). Plutôt que de dresser ici la liste des packages, je préfère vous renvoyer vers le *Linux From Scratch HOWTO* `<http://www.linuxfromscratch.org>`. Il contient une liste complète et à jour de tous les packages allant dans un système Linux, aussi bien que des indications pour les construire.

13 Construire un système Linux minimum à partir des sources.

Nous nous sommes concentrés jusqu'ici sur ce que les packages font. Je vais vous donner ici tout les indices que je peux pour fabriquer un système Linux de base à partir des sources. Si vous voulez monter un vrai système pour du vrai travail, lisez le *Linux From Scratch HOWTO* `<http://www.linuxfromscratch.org>`.

Il est possible d'obtenir une ligne de commande bash sans installer tout ce que je mentionne ici. Ce que je décris est un système de base, sans embûche, qui peut être monté facilement.

13.1 Ce qu'il vous faut

Nous installerons une distribution de Linux comme Red Hat sur une partition, et l'utiliserons pour construire un nouveau système Linux sur une autre partition. Je nommerai par la suite “cible” le système que nous construisons, et “source” le système que nous utilisons pour construire le système cible (à ne pas confondre avec *code source* que nous utiliserons aussi).

Vous allez donc avoir besoin d'une machine avec deux partitions libres dessus. Si vous le pouvez, utilisez une machine qui ne contienne rien d'important. Vous pouvez utiliser un système Linux déjà existant comme système source, mais je le déconseille. Si vous oubliez un des paramètres des commandes que nous allons saisir, vous pourriez accidentellement réinstaller des choses sur votre système source. Cela peut mener à des incompatibilités, et des conflits.

Les anciennes architectures PC, pour la plupart 486 et plus ancien, ont une limitation ennuyeuse de leur Bios. Il ne peuvent lire les disques durs passé les 512 premiers mégaoctets. Ce n'est pas vraiment un problème pour Linux, qui gère lui-même les disques une fois lancé. Mais pour que Linux soit chargé sur ces vieilles machines, le noyau doit résider quelque part en dessous de 512 mégaoctets. Si vous utilisez une de ces machines, vous devez créer une partition distincte en dessous de 512 Mo, à monter sur `/boot` pour toute partition au dessus de la limite des 512 Mo.

La dernière fois que je l'ai fait, j'ai utilisé Red Hat 6.1 comme système source. J'ai installé le système de base plus

– cpp
– egcs
– egcs-c++
– patch
– make
– dev86

– ncurses-devel
– glibc-devel
– kernel-headers

J'ai aussi installé X-window et Mozilla (NDT : Netscape) pour pouvoir lire les documentations facilement, mais ce n'est pas nécessaire. A la fin de mon travail, cela avait pris environ 350M d'espace disque (Cela semble un peu élevé, je me demande pourquoi).

Le système cible achevé prenait 650M, mais comprenait tout le code source et les fichiers intermédiaires. Si l'espace est limité, je vous conseille de faire un `make clean` après la construction de chaque package. Cela dit, c'est une source d'ennuis et d'hésitation.

Enfin, vous allez avoir besoin du code source du système que vous allez construire. Il y a les "packages" dont nous avons parlé dans ce document. On peut les obtenir depuis un CD, ou par l'Internet. Je donnerai les URL pour les sites américains et miroirs australiens.

– MAKEDEV *USA* <ftp://tsx-11.mit.edu/pub/linux/sources/sbin> Autre site aux *USA* <ftp://sunsite.unc.edu/pub/Linux/system/admin>
– Lilo *USA* <ftp://lrcftp.epfl.ch/pub/linux/local/lilo/>, *Australie* <ftp://mirror.aarnet.edu.au/pub/linux/metalab/system/boot/lilo/>.
– Noyau Linux Utilisez un des miroirs listés sur *home page* <http://www.kernel.org> plutôt que *USA* <ftp://ftp.kernel.org/pub/linux/kernel> car ils sont toujours en surcharge. *Australie* <ftp://kernel.mirror.aarnet.edu.au/pub/linux/kernel/>
– GNU libc La bibliothèque elle-même, ainsi que les add-on linuxthreads sont sur *USA* <ftp://ftp.gnu.org/pub/gnu/glibc> *Australie* <ftp://mirror.aarnet.edu.au/pub/gnu/glibc>
– Add-ons à la libc GNU Vous aurez aussi besoin des linuxthreads et des add-on libcrypt. Si libcrypt est absente à cause des lois américaines sur l'exportation, vous pouvez la récupérer sur *libcrypt* <ftp://ftp.gwdg.de/pub/linux/glibc> les add-ons linuxthreads sont au même endroit que libc proprement dite.
– GNU ncurses *USA* <ftp://ftp.gnu.org/gnu/ncurses> *Australie* <ftp://mirror.aarnet.edu.au/pub/gnu/ncurses>
– SysVinit *USA* <ftp://sunsite.unc.edu/pub/Linux/system/daemons/init> *Australie* <ftp://mirror.aarnet.edu.au/pub/linux/metalab/system/daemons/init>
– GNU Bash *USA* <ftp://ftp.gnu.org/gnu/bash> *Australie* <ftp://mirror.aarnet.edu.au/pub/gnu/bash>
– GNU sh-utils *USA* <ftp://ftp.gnu.org/gnu/sh-utils> *Australie* <ftp://mirror.aarnet.edu.au/pub/gnu/sh-utils>
– util-linux *Ailleurs* <ftp://ftp.win.tue.nl/pub/linux/utils/util-linux/> *Australie* <ftp://mirror.aarnet.edu.au/pub/linux/metalab/system/misc> Ce package contient `agetty` et `login`.

Pour résumer, il vous faut :
– Une machine avec deux partitions distinctes d'environ 400M et 700M respectivement, bien que vous puissiez sûrement vous en tirer avec un espace plus restreint.
– Une distribution de Linux (un CD de Red Hat par exemple), et de quoi l'installer (un lecteur de CD par exemple).
– Les archives (tarballs : fichier contenant plusieurs autres fichiers concaténés en un seul, puis compressé ou non ensuite) du code source, listées ci-dessus.

Je pars du principe que vous pouvez installer le système source vous-même, sans aide de ma part. A partir de maintenant, je considère que c'est fait.

Les premiers pas de ce projet consistent à faire démarrer le noyau, et le laisser 'paniquer' (panic) car il ne peut trouver le programme `init`. Cela signifie que nous allons devoir installer un noyau, et installer Lilo. Pour que Lilo fonctionne quand même correctement, nous avons besoin d'installer les fichiers spéciaux dans le `/dev` du système cible. Lilo en a besoin pour effectuer les accès bas niveau au disque, nécessaire pour

écrire le secteur d'amorce. MAKEDEV est le script qui crée ces fichiers spéciaux (Vous pourriez bien sûr les recopier depuis le système source, mais ce serait tricher !). Mais d'abord, il nous faut un système de fichiers pour les mettre dedans.

13.2 Le système de fichier (Filesystem)

Notre nouveau système a besoin d'un système de fichiers pour vivre. Donc, il nous faut tout d'abord créer ce système de fichiers en utilisant `mke2fs`. Ensuite il faut le monter quelque part. Je vous suggère `/mnt/target` (comme 'cible'). Dans ce qui va suivre, je considère que votre système se trouve à cet endroit. Vous pouvez gagner un peu de temps en ajoutant une entrée dans `/etc/fstab` de façon à ce que le montage de votre système destination soit automatique lorsque votre système source démarre.

Lorsque nous démarrerons le système cible, ce qui se trouve dans `/mnt/target` se trouvera alors dans `/` (à la racine).

Nous avons besoin d'une structure de sous-répertoires sur la cible. Jetez un oeil au Standard de la Hiérarchie des Fichiers (File Hierarchy Standard, voir section 7.3 (Système de Fichiers)). pour trouver vous même ce qu'elle devrait être, ou faites simplement un `cd` vers l'endroit où la cible est montée et tapez aveuglément :

```
mkdir bin boot dev etc home lib mnt root sbin tmp usr var
cd var; mkdir lock log run spool
cd ../usr; mkdir bin include lib local sbin share src
cd share/; mkdir man; cd man
mkdir man1 man2 man3 ... man9
```

Comme le FHS et la plupart des packages se contredisent en ce qui concerne l'endroit où les man pages doivent se trouver, nous avons besoin d'un lien symbolique :

```
cd ..; ln -s share/man man
```

13.3 MAKEDEV

Nous mettrons le code source dans le répertoire `/usr/src` cible. Aussi si votre système de fichiers cible est monté par exemple sur `/mnt/target`, et que vos archives sont dans `/root`, il faudra faire :

```
cd /mnt/target/usr/src
tar -xzvf /root/MAKEDEV-2.5.tar.gz
```

Ne vous comportez pas en amateur fini et pensez à copier vos archives à l'endroit où vous allez les décompresser ;-)

En principe, lorsque vous installez un logiciel, vous l'installez sur le système en fonctionnement. En l'occurence, ce n'est pas notre intention, nous souhaitons l'installer comme si /mnt/target était le système de fichiers racine. Les différents packages ont différentes manières de vous le laisser faire. Pour MAKEDEV, vous devez faire

```
ROOT=/mnt/target
make install
```

Vous devez rechercher ces options dans les fichiers README et INSTALL ou faire un `./configure -help`.

Explorez le `Makefile` de MAKEDEV pour voir l'usage qu'il fait de la variable ROOT, que nous avons définie dans cette commande. Ensuite jetez un oeil à la page de manuel en faisant un `man ./MAKEDEV.man` pour voir comment il fonctionne. Vous découvrirez que la méthode utilisée pour ces fichiers spéciaux consiste à faire un `cd /mnt/target/dev` puis un `./MAKEDEV generic`. Faites un `ls` pour découvrir tous les merveilleux fichiers spéciaux qu'il a créés pour vous !

13.4 Le noyau (kernel)

Ensuite, nous devons fabriquer un noyau. Je considère que vous l'avez déjà fait, aussi serai-je bref. Il est plus facile d'installer Lilo si le noyau censé être monté est déjà là. Retournez dans le répertoire `/usr/src` de la cible, et décompressez-y les sources du noyau linux. Entrez dans l'arborescence des sources (`cd linux`) et configurez le noyau, en utilisant votre méthode préférée, comme par exemple `make menuconfig`. Vous vous faciliterez grandement la vie si vous configurez un noyau sans module. Si vous configurez des modules, vous devrez éditer `Makefile`, trouver `INSTALL_MOD_PATH`, et lui affecter la valeur `/mnt/target`.

Vous pouvez maintenant taper `make dep`, `make bzImage`, et si vous avez configuré des modules : `make modules`, `make modules_install`. Copiez le noyau `arch/i386/boot/bzImage` et le plan système `System.map` vers le répertoire de boot de la cible `/mnt/target/boot`, et nous seront prêts à installer Lilo.

13.5 Lilo

Lilo est livré avec un très beau script nommé `QuickInst`. Décompressez les sources de Lilo dans le répertoire des sources du système cible, lancez ce script par la commande `ROOT=/mnt/target ./QuickInst`. Il vous posera plusieurs questions concernant la manière dont vous souhaitez que Lilo soit installé.

Souvenez-vous, comme nous avons affecté à la variable `ROOT` la partition cible, vos noms de fichiers s'y rapportent. Donc, lorsqu'il vous demandera le nom du noyau à lancer par défaut, répondez `/boot/bzImage`, *pas* `/mnt/target/boot/bzImage`. J'ai trouvé un bug mineur dans le script, qui lui fait dire :

```
./QuickInst: /boot/bzImage: no such file
```

Mais si vous vous contentez de l'ignorer, cela passe quand même.

Comment doit-on s'y prendre pour expliquer à `QuickInst` où installer le secteur de boot ? Quand nous redémarrons, nous voulons avoir le choix de démarrer le système source ou le système cible, ou encore n'importe quel autre système présent sur la machine. Et nous souhaitons que l'instance de Lilo que nous mettons en place maintenant lance le noyau de notre nouveau système. Comment est-ce que l'on réalise ces deux choses ? Ecartons-nous un moment du sujet et étudions la façon dont Lilo démarre DOS sur un système Linux en dual-boot. Le fichier `lilo.conf` d'un tel système doit sûrement ressembler à çà.

```
prompt
timeout = 50
default = linux

image = /boot/bzImage
        label  = linux
        root   = /dev/hda1
        read-only

other = /dev/hda2
        label = dos
```

Si la machine est configurée de cette façon, alors le Master Boot Record (MBR) est lu et chargé par le Bios, et lance le bootloader de Lilo, qui affiche une invite de commande. Si vous tapez `dos` dans cette invite, Lilo chargera le secteur de boot depuis hda2, qui lancera DOS.

Ce que nous allons faire est exactement la même chose, mis à part que le secteur d'amorce d'hda2 va être un autre secteur d'amorce Lilo - celui-là même que `QuickInst` va installer. Donc le Lilo de la distribution de linux chargera le Lilo que nous avons construit, qui chargera le noyau que nous avons bâti. Vous verrez alors deux invites de commande Lilo au redémarrage.

Pour raccourcir une longue histoire, lorsque `QuickInst` vous demande où placer le secteur de boot, indiquez-lui l'endroit où se trouve votre système de fichiers cible, par exemple `/dev/hda2`.

Maintenant modifiez le fichier `lilo.conf` de votre système source, de façon à ce qu'il ait une ligne ressemblant à :

```
other = /dev/hda2
        label = target
```

Lancez Lilo, et nous devrions être capables de faire notre premier démarrage sur le système cible.

13.6 Glibc

L'étape suivante consiste à installer `init`, mais comme la plupart des programmes qui tournent sous Linux, `init` utilise des fonctions issues de la bibliothèque C GNU, glibc. Aussi l'installerons-nous en premier.

Glibc est un package très gros et très compliqué. Il faut 90 heures pour le bâtir sur mon vieux 386sx/16 avec 8M RAM. Mais cela ne prend que 33 minutes sur mon Celeron 433 avec 64M. Je pense que la quantité de mémoire est le principal critère dans notre cas. Si vous n'avez que 8Mo de RAM (ou - j'en tremble - encore moins !), préparez vous à une très longue compilation.

La documentation d'installation de glibc recommande une construction dans un répertoire distinct. Cela vous permet de recommencer facilement, en supprimant simplement ce répertoire. Cela vous permet aussi d'économiser 265Mo d'espace disque.

Décompressez l'archive `glibc-2.1.3.tar.gz` (ou n'importe quelle autre version) dans `/mnt/target/usr/src` comme d'habitude. A présent, nous devons décompresser les "add-on" dans le répertoire de la glibc. Donc, faites un `cd glibc-2.1.3`, puis décompressez à cet endroit les archives `glibc-crypt-2.1.3.tar.gz` et `glibc-linuxthreads-2.1.3.tar.gz`.

Maintenant, nous pouvons créer le répertoire de construction, configurer, bâtir et installer glibc. Voici les commandes que j'ai utilisées, mais relisez vous-même la documentation et assurez-vous de faire ce qui est le plus approprié dans votre environnement. Toutefois, avant de faire tout cela, vous voudrez sans doute connaître l'espace disque qu'il vous reste par un `df`. Vous pouvez en faire un autre après avoir bâti et installé glibc pour en déduire son volume.

```
cd ..
mkdir glibc-build
../glibc-2.1.3/configure --enable-add-ons --prefix=/usr
make
make install_root=/mnt/target install
```

Remarquez que nous avons ici encore une autre façon de dire au package l'endroit où s'installer.

13.7 SysVinit

Bâtir et installer les binaires de SysVinit est assez direct. Je me contenterai d'être paresseux et de vous donner les commandes, en considérant que vous avez décompressé son code source, et que vous êtes entré dans son répertoire.

```
cd src
make
ROOT=/mnt/target make install
```

Il existe aussi beaucoup de scripts associés à `init`. Il y a des scripts d'exemple fournis dans le package de SysVinit, qui fonctionnent bien. Mais vous devez les installer manuellement. Ils sont organisés dans une hiérarchie sous `debian/etc` dans l'arborescence du code source de SysVinit. Vous pouvez recopier toute cette hiérarchie dans le répertoire `etc` du système cible, avec une commande du style `cd ../debian/etc; cp -r * /mnt/target/etc`. Evidement, vous explorerez ces scripts avant de tous les recopier.

Tout est désormais en place pour permettre au noyau cible de lancer `init` au redémarrage. Le problème, cette fois, viendra des scripts qui ne pourront être exécutés car `bash` ne sera pas là pour les interpréter. `init` tentera également de lancer des `getty`, qui sont inexistants eux aussi. Rebootez maintenant, et assurez-vous que tout le reste fonctionne correctement.

13.8 Ncurses

L'étape suivante consiste à mettre Bash en place, mais bash a besoin de ncurses, aussi devons-nous installer celui-ci en premier. Ncurses remplace termcap dans la manière de gérer les écrans texte, mais apporte également une compatibilité ascendante en prenant en charge les appels termcap. Dans l'objectif d'avoir un système moderne, simple et propre, je pense que le mieux est de désactiver l'ancienne méthode termcap. Vous pourriez par la suite rencontrer des problèmes avec des applications utilisant termcap, mais au moins vous connaîtrez les éléments qui l'utilisent. Si vous en avez besoin, vous pouvez recompiler ncurses avec prise en charge de termcap.

Les commandes que j'ai utilisées sont :

```
./configure --prefix=/usr --with-install-prefix=/mnt/target --with-shared --disable-termcap
make
make install
```

13.9 Bash

Il m'a fallu beaucoup de lecture, de réflexion, de tests, et d'erreurs pour que Bash s'installe là où je pensais qu'il devait aller. Les options de configuration que j'ai utilisées sont :

```
./configure --prefix=/mnt/target/usr/local --exec-prefix=/mnt/target --with-curses
```

Une fois que vous avez bâti et installé Bash, vous devez créer un lien symbolique comme ceci : `cd /mnt/target/bin; ln -s bash sh`. Cela est dû au fait que les scripts débutent généralement par une ligne comme celle-ci :

```
#!/bin/sh
```

Si vous n'avez ce lien symbolique, les scripts ne fonctionneront pas, car ils chercheront `/bin/sh` et non `/bin/bash`.

Vous pouvez redémarrer à ce point si vous le souhaitez. Vous devriez remarquer que les scripts peuvent maintenant s'exécuter, bien que vous ne puissiez vous loguer, car il n'y pas encore de programmes `getty` ou `login`.

13.10 Util-linux (getty et login)

Le package util-linux contient `agetty` et `login`. Nous avons besoin des deux pour nous loguer et obtenir la ligne de commande de bash. Après l'avoir installé, faites un lien symbolique depuis `agetty` vers `getty` de le répertoire `/sbin` du système cible. `getty` est un des programmes censés se trouver sur tous les systèmes de type Unix, donc faire un lien est une meilleure idée que de modifier `inittab` pour qu'il lance `agetty`.

Il me reste un problème avec la compilation d'util-linux. Le package contient également le programme `more`, et je n'ai pas été capable de persuader le processus `make` de placer le lien `more` sur la bibliothèque de ncurses 5 du système cible, plutôt que sur ncurses 4 du système source. Je regarderai cela de plus près.

Vous aurez aussi besoin d'un fichier `/etc/passwd` sur le système cible. C'est l'endroit où le programme `login` ira vérifier votre accréditation. Comme il ne s'agit que d'un système gadget à ce niveau, vous pouvez vous permettre des choses scandaleuses, comme ne définir que l'utilisateur root, sans mot de passe! Mettez le simplement dans le fichier `/etc/passwd` du système cible.

```
root::0:0:root:/root:/bin/bash
```

Les champs sont séparés par des deux-points, correspondent, de gauche à droite, à l'user id (nom de login), au mot de passe (crypté), au numéro d'utilisateur, au numéro de groupe, au nom de l'utlisateur, à son répertoire personnel, et à son shell par défaut.

13.11 Sh-utils

Le dernier package dont nous ayons besoin est sh-utils GNU. Le seul programme nécessaire à ce niveau est `stty`, qui est utilisé dans `/etc/init.d/rc`, lui-même utilisé pour changer de niveau d'exécution et entrer dans le niveau initial. En fait, je possède et ai utilisé un package qui ne contient que `stty` mais je ne peux me souvenir d'où il vient. Il vaut mieux utiliser le package GNU, car il contient d'autres choses dont vous aurez besoin si vous voulez les ajouter au système pour le rendre vraiment utilisable.

Eh bien ça y est. Vous devriez maintenant avoir un système qui doit démarrer et vous donner l'invite de login. Saisissez-y "root", et vous devriez avoir le shell. Vous ne pourrez pas faire grand chose avec, il n'y a même pas de commande `ls` pour voir votre travail. Tapez deux fois la touche tab pour voir les commandes disponibles. C'est la chose la plus intéressante que j'ai trouvée à faire avec.

13.12 Vers l'utilisabilité

Il semblerait que nous ayons là un système plutôt inutilisable. Mais en réalité, nous ne sommes pas très loin de pouvoir commencer à travailler avec. L'une des premières choses à faire est de rendre le système de fichiers racine accessible et lecture et écriture. Il y a un script issu du package, dans `/etc/init.d/mountall.sh` qui s'occupe de cela, et effectue un `mount -a` pour monter automatiquement tout ce qui est spécifié dans le fichier `/etc/fstab`. Mettez un lien symbolique du genre `S05mountall` vers lui dans le répertoire `etc/rc2.d` du système cible.

Il se peut que ce script utilise des commandes que vous n'avez pas encore installées. Si c'est le cas, trouvez le package qui contient ces commandes et installez-le. Voyez la section 13.13 (Random Tips) pour avoir des indications sur la marche à suivre pour trouver ces packages.

Regardez les autres scripts dans `/etc/init.d`. La plupart d'entre-eux doit être incluse dans tout système sérieux. Ajoutez-les un à un, et assurez-vous que tout se lance en douceur avant d'en ajouter d'autres.

Lisez le Standard de la Hiérarchie des Fichiers (voir section 7.3 (Système de Fichiers)). Il contient une liste de commandes qui devraient être dans `/bin` et `/sbin`. Assurez-vous que toutes ces commandes sont installées sur votre système. Mieux encore, trouvez la documentation Posix qui spécifie tout cela.

A partir de maintenant, il n'est plus question que d'ajouter de plus en plus de packages, jusqu'à ce que tout ce que vous souhaitez avoir se trouve sur votre système. Installez les outils de construction comme `make` et `gcc` le plus tôt possible. Une fois que cela est fait, vous pouvez utiliser votre système cible pour se construire lui-même, ce qui est bien moins compliqué.

13.13 Astuces diverses

Si vous avez une commande appelée `thingy` sur un système Linux avec RPM, et souhaitez avoir des indications sur l'endroit où trouver les sources, vous pouvez utiliser la commande :

```
rpm -qif `which thingy`
```

Et si vous avez un CD de sources Red Hat, vous pouvez installer le code source avec

```
rpm -i /mnt/cdrom/SRPMS/ce.qu.il.vient.de.dire-1.2.srpm
```

Ceci mettra l'archive, avec les patches Redhats éventuels dans `/usr/src/redhat/SOURCES`.

13.14 Plus d'informations

- Il existe un mini-howto sur la manière de construire les logiciels à partir de leurs sources, le *Software Building mini-HOWTO* <http://www.linuxdoc.org/HOWTO/Software-Building-HOWTO.html> (miroir français en *VO* <http://linuxfr.org/LDP/HOWTO/Software-Building-HOWTO.html>).
- Il existe aussi un HOWTO sur la manière de construire un système Linux depuis zéro. Il met l'accent sur la construction d'un système réellement utilisable, plutôt que pour le simple exercice. *The Linux From Scratch HOWTO* <http://www.linuxfromscratch.org>

14 Conclusion

L'un des meilleurs côtés de Linux, à mon humble avis, est que vous pouvez entrer dedans et voir réellement comment il fonctionne. J'espère que vous apprécierez cela autant que moi. Et j'espère que ces quelques notes vous y auront aidé.

15 Section administrative

15.1 Copyright

Ce document est copyright (c) 1999, 2000 Greg O'Keefe. Vous êtes libre de l'utiliser, le copier, le distribuer ou le modifier, sans obligation, selon les termes de la Licence Publique Générale (GPL : *GNU General Public Licence* <http://www.gnu.org/copyleft/gpl.html>). Merci de conserver les références à l'auteur si vous utilisez tout ou partie de ce document dans un autre.

15.2 Page principale

Les mises à jour de ce document évoluent sur *From Powerup To Bash Prompt* <http://learning.taslug.org.au/power2bash>.

15.3 Retours

J'aimerais recevoir vos commentaires, critiques et suggestions. Veuillez s'il vous plaît me les envoyer à *Greg O'Keefe* <mailto:gcokeefe@postoffice.utas.edu.au>

15.4 Références et remerciements.

Les noms de produits cités sont marques déposées de leur propriétaires respectifs, et considérés par cette note comme reconnus comme tels.

Il y a quelques personnes que je voudrais remercier, pour m'avoir aidé à réaliser tout ceci.

Tout les abonnés de la liste de discussion learning@TasLUG

Merci pour avoir lu tous mes mails et posé des questions intéressantes. Vous pouvez rejoindre cette liste en envoyant un message à *majordomo* <mailto:majordomo@taslug.org.au> avec la phrase

```
subscribe learning
```

dans le corps du message.

Michael Emery

Pour m'avoir rappelé Unios.

Tim Little

Pour de bonnes indications concernant `/etc/passwd`

sPaKr dans #linux sur efnet

Qui a soupçonné l'utilisation de `/etc/services` par syslog, et m'a fait connaître la phrase "rolling your own" ("roulez-la vous-même") pour décrire la construction d'un système à partir des sources.

Alex Aitkin

Pour avoir porté Vico et son "verum ipsum factum" (La compréhension découle de l'expérience) à mon attention.

Dennis Scott

Pour avoir corrigé mon arithmétique en hexadécimal.

jdd

Pour avoir mis en évidence quelques erreurs typographiques.

David Leadbeater

Pour avoir contribué aux "pérégrinations" dans les démons noyau.

15.5 Historique des changements

15.5.1 0.6 -> 0.7

- L'accent est plus porté sur l'explication, et moins sur la façon de monter un système, ces informations ayant été regroupées dans une section distincte, et le système une fois construit a été revu à la baisse, voir directement la documentation de Gerard Beekmans "Linux From Scratch" pour construire un système sérieux.
- Ajout de quelques hypothèses de la part de David Leadbeater

- Correction de deux URL, ajout d'un lien vers le téléchargement d'unios sur learning.taslug.org.au/resources
- Test et correction d'url.
- Grand nettoyage et réécriture générale.

15.5.2 0.5 -> 0.6

- Ajout de l'historique des changements
- Ajout de quelques "todos" ("A faire").

15.6 A faire (TODO)

- expliquer les modules noyau, depmod, modprobe, insmod et tout (il faut d'abord que je trouve moi-même).
- mentionner le système de fichiers /proc. Exercices potentiels.
- convertir en documentation sgml
- ajouter plus d'exercices, peut-être une section entière d'exercices plus poussés, comme créer un système de fichiers minimal fichier par fichier à partir de l'installation d'une distribution.