De la mise sous tension à l'invite de commande de Bash

Greg O'Keefe

<gcokeefe@postoffice.utas.edu.au>

Dominique van den Broeck – Traduction française

Jean–Philippe Guérard – Relecture de la version française

v0.9, novembre 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 connectez et obtenez l'invite de commande de Bash. Comprendre ces mécanismes vous sera très utile lorsque vous aurez besoin de résoudre des problèmes ou de configurer votre système.

Table des matières

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, Les notions fondamentales d'Unix et d'Internet est un excellent endroit pour débuter. C'est un document 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 en apprendrez bien plus qu'en vous contentant de lire ce document.

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 « de l'expérience naît la compréhension » (NdT : traduction libre). Merci à Alex (voir références et remerciements) pour cette citation.

Si vous souhaitez vous réaliser votre propre installation Linux, je vous conseille d'aller voir le document de Gerard Beekmans Comment faire un système Linux à partir de zéro (*Linux from scratch − LFS*). 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 liste de discussion à la disposition des personnes qui construisent de tels systèmes. Les instructions jadis incluses à ce guide se trouvent maintenant dans un document distinct nommé « Construire un système Linux minimum à partir du code source » et qui peut être récupéré sur le site From PowerUp to Bash Prompt pour construire un système jouet, purement pour l'exercice.

Les paquets sont présentés dans l'ordre où ils apparaissent dans le processus de démarrage du système. Cela signifie que si vous installez les paquets 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

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 l'auto−test à l'allumage (*Power On Self Test − POST*). Ensuite, un programme nommé chargeur d'amorçage (*bootstrap loader*), situé dans le BIOS en ROM, recherche un secteur d'amorçage. Un secteur d'amorçage est le premier secteur d'un disque et contient un petit programme capable de charger un système d'exploitation. Les secteurs d'amorçage sont marqués par un nombre magique (i.e. une valeur fixe caractéristique) 0xAA55 = 43603 à l'octet 0x1FE = 510. Ce sont les deux derniers octets du secteur. C'est de cette façon que la partie matérielle peut déterminer s'il s'agit d'un secteur d'amorçage ou pas.

Le chargeur d'amorçage a une liste d'endroits où chercher un secteur d'amorçage. 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'amorçage sur un CD−ROM. S'il trouve un secteur d'amorçage, il le charge en mémoire et passe ainsi le contrôle au programme qui charge le système d'exploitation en mémoire. Sur un système Linux classique, ce programme sera la première étape du chargeur de Lilo. Il existe malgré tout plusieurs manières différentes de configurer l'amorçage de votre système. Voir le *Guide de l'utilisateur de Lilo* pour plus de détails. Voir la section liens sur Lilo pour l'url.

Évidement, il y a bien plus à dire sur ce que fait la partie matérielle du PC. Mais ce n'est pas l'objet de ce document. Lisez un des nombreux livres traitant de l'architecture matérielle des 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 de configuration, 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 **Suppr** avant qu'elle ne commence à charger le système d'exploitation.

2.2. Exercices

Une bonne façon d'en apprendre plus sur la partie matérielle 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, (ils avaient une page sur http://www.unios.org, mais elle a disparu) et téléchargez, compilez et fabriquez votre disquette amorçable. Ce n'est qu'un programme d'amorçage affichant « Hello World! », contenant à peine plus de 100 lignes d'assembleur. Il serait intéressant de le voir converti en un format exploitable par l'assembleur **as** de GNU.

Ouvrez l'image de la disquette d'amorçage 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 amorçable de votre propre ordinateur.

Vous pouvez utiliser la commande **dd** pour la copier dans un fichier : **dd if=/dev/fd0 of=*secteur.d.amorcage***. 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. Aller plus loin

* Les notions fondamentales d'Unix et d'Internet, par Eric S. Raymond, 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. Voir la section liens sur Lilo pour l'url.
* *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 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 une partie de Lilo, appelée chargeur d'amorçage de premier niveau (*first stage boot loader*). Il s'agit d'un mini−programme dont la seule tâche est de charger et d'exécuter le chargeur d'amorçage de deuxième niveau (*second stage boot loader*).

Le chargeur d'amorçage de deuxième niveau 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 l'outil de définition des localisations (*map installer*). Celui−ci lit le fichier de configuration `/etc/lilo.conf` et écrit le chargeur d'amorçage 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 amorçable. 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 méthodes d'amorçage. 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 pouvoir lancer. Pour une entrée Linux, cela inclut l'emplacement du noyau, et la partition à monter comme racine du système de fichier. Pour les autres systèmes, la principale information est la partition sur laquelle démarrer.

3.2. Exercices

DANGER : soyez prudent avec ces exercices. Il est assez facile de faire une erreur quelque part et de bloquer votre bloc de démarrage (*master boot record − MBR*, premier secteur du disque dur, qui contient le chargeur d'amorçage 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 ci−dessous 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 noyau − vous obtiendrez un kernel panic quand le noyau sera prêt à 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 « Comment faire une disquette d'amorçage » (url ci−dessous), et tomsrtbt (url ci−dessous) pour avoir des pistes.

Configurez Lilo afin qu'il lance Unios (voir section exercices sur la partie matérielle pour une url). Comme défi supplémentaire, voyez si vous pouvez le faire sur une disquette.

Faites une boucle d'amorçage. Configurez le Lilo du bloc de démarrage pour qu'il lance le Lilo du secteur d'amorçage d'une des partitions principales, puis configurez ce Lilo pour qu'il relance celui du bloc de démarrage. Ou alors utilisez le bloc de démarrage et vos quatre partitions principales pour faire une boucle en cinq points ! Marrant !

3.3. Aller plus loin

- La page de manuel de Lilo
- Le paquet Lilo (ftp://lrcftp.epfl.ch/pub/linux/local/lilo/) contient le « Guide l'utilisateur de Lilo » `lilo-u-21.ps.gz` (ou une version plus récente). Il se peut que vous disposiez déjà de ce document. Regardez dans `/usr/share/doc/lilo` ou à un endroit similaire. La version PostScript est meilleure que la version en texte brut, car elle contient des diagrammes et des tables.
- tomsrtbt : le Linux mono−disquette le plus cool ! Constitue une excellente disquette de secours.
- Comment faire une disquette d'amorçage (*Bootdisk HOWTO*).

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 matériel 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 quelconque. Cette tâche du noyau est appelée ordonnancement.

Si un programme ne fait rien, alors il n'a pas besoin d'être conservé en mémoire (RAM). Même un programme qui travaille peut avoir certaines parties inactives, qui n'ont donc pas besoin d'être en mémoire. L'espace adressable est divisé en pages. Le noyau garde une trace des pages les plus utilisées. Les pages qui sont moins souvent 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 une grande quantité de code spécifique pour interagir avec tous types de matériels, et pouvoir les présenter d'une façon propre et uniforme aux programmes.

Le noyau prend aussi en charge la gestion des fichiers, les communications entre processus, et une grande partie 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/` (là ou se trouvent les sources de votre noyau Linux). La commande **rdev** vous permet réinitialiser le mode vidéo par défaut, la racine du système de fichiers, le périphérique d'échange et la taille du disque virtuel (disque RAM). 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 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.

> Le fichier `System.map` est utilisé lors de l'écriture d'entrées dans le journal système pour déterminer les noms des modules générant les 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 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 non nécessaires.

Lisez « Le noyau Linux » (url ci–dessous) et ce faisant, trouvez les parties des sources auxquelles il se réfère. Le livre (au moment où j'écris ces lignes) se réfère au noyau version 2.0.33, qui commence à être 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 quoi que ce soit.

4.3. Aller plus loin

- Le fichier `/usr/src/linux/README` et le contenu du répertoire `/usr/src/linux/Documentation/`. Leurs emplacements peuvent varier selon votre système.
- Le Comment faire un noyau Linux (*Kernel HOWTO*).
- L'aide disponible quand vous configurez un noyau en utilisant **make menuconfig** ou **make xconfig**. Il existe une version française de cet aide disponible sur http://traduc.org/kernelfr/.
- « Le noyau Linux » et les autres guides du projet de documentation Linux (LDP)
- Pour le code source, suivre les hyperliens dans Construire un système Linux minimum à partir du code source

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. Pour un système GNU/Linux, il s'agit de glibc. Si vous les compilez sous Visual C++, alors il provient d'une mise en Suvre Microsoft de ces mêmes fonctions standard. Il existe des masses de ces fonctions standard, pour les mathématiques, la gestion des chaînes de caractères, de l'heure et de la date, des allocations de mémoire et ainsi de suite. Tout, dans Unix (y compris Linux) est soit écrit en C, soit doit faire de son mieux pour faire comme si, de sorte que tous les programmes utilisent ces fonctions.

Si vous jetez un Sil dans `/lib` sur votre système Linux, vous verrez un grand nombre de fichiers appelés `libquelquechose.so` ou `libquelquechose.a` et cætera. Ce sont les bibliothèques de ces fonctions. Glibc est simplement la mise en Suvre GNU de ces fonctions.

Les programmes peuvent utiliser ces fonctions de deux manières. Si vous réalisez une édition de liens *statique*, ces fonctions seront copiées à l'intérieur de l'exécutable généré. C'est à cela que servent les bibliothèques `libquelquechose.a`. Si vous réalisez une édition de liens *dynamique* (cas par défaut), lorsque le programme aura besoin du code d'une bibliothèque, il l'appellera directement depuis le 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 des bibliothèques dépendent de la région géographique où vous vous trouvez. Par exemple, en français, on écrit les dates sous la forme jj/mm/aa, mais les américains les écrivent sous la forme mm/jj/aa. Ceci est configurable via un programme appelé **localdef** livré avec glibc.

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. Écrivez, compilez, et liez un programme utilisant cette bibliothèque.

5.3. Aller plus loin

- Pour le code source, suivre les liens dans Construire un système Linux minimum à partir du code source

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 correctement. 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, et cætera. Init démarre aussi les processus getty qui vous donnent l'invite de connexion 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. En général, 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, et cætera.

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 niveaux d'exécution (*runlevels*). 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 les scripts dont le nom débute par la lettre « S ». Il se peut qu'il tombe sur `S10syslog` en premier. Les chiffres indiquent au script de gestion des niveaux d'exécution l'ordre dans lequel il doit les lancer. En l'occurrence, `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 démon de gestion du journal système. Parce que le nom du lien commence par un « S », le script de gestion des niveaux d'exécution 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 qui doit être arrêter, et dans quel ordre, lorsque l'on entre dans ce niveau d'exécution.

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

La dernière chose importante qu'effectue init est de démarrer les getty. 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 faire tourner plus de choses à la fois, et passer rapidement de l'une à l'autre. Vous pourriez aussi avoir besoin de lancer un getty vers un terminal texte ou vers un modem. Vous devrez alors éditer `inittab`.

6.1. Configuration

`/etc/inittab` est le fichier de configuration principale d'init.

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 ce 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 getty 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 "Bonjour, 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 document de référence sur Bash à portée de la main.

6.3. Aller plus loin

- 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 Linux de l'administrateur système du projet de documentation Linux (LDP) contient une section intéressante sur init.
- Pour le code source, suivre les liens du Construire un système Linux minimum à partir du code source

7. Le système de fichiers

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 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 de Linux.

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 représenté par un *inode*, indiquant le fichier dont il s'agit, sa date de 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 répertoires contiennent. Si le système veut lire `/home/greg/groslolos.jpeg`, il commence par lire l'inode du répertoire racine `/` dans le « superbloc », 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 `groslolos.jpeg` qui lui dira quels blocs du disque il doit 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. La commande mount est ensuite utilisée pour le remonter en lecture−écriture après que fsck aie vérifié que tout est en ordre.

Linux prend aussi en charge d'autres types de systèmes de fichiers : msdos, vfat, minix, et cætera. Les détails d'un système de fichiers spécifique sont masqués par le système de fichier virtuel (Virtual File System − VFS), qui est une couche d'abstraction. Je ne rentrerai pas dans ces détails. Il existe une discussion sur ce sujet dans « Le noyau Linux » (voir la section aller plus loin − le noyau Linux pour l'url).

Un type de système de fichiers complètement différent est monté sur /proc. C'est une véritable projection de ce qui se passe dans le noyau. On y trouve un répertoire pour chaque processus existant sur le système, dont le nom correspond au numéro dudit processus. Il existe aussi des fichiers comme `interrupts` et `meminfo` qui donnent des informations sur l'utilisation du matériel. Vous pouvez découvrir énormément de choses en explorant `/proc`.

7.1. Configuration

Il est possible d'indiquer à **mke2fs**, la commande de création des système de fichiers ext2, des paramètres définissant la taille des blocs, le nombre d'inodes, et cætera. Voir la page de manuel 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 courrier électronique (en anglais) avec vos commentaires !

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

7.3. Aller plus loin

- 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 du projet de documentation Linux (LDP).

* La commande **mount** fait partie du paquet util−linux, il y a un lien vers celui−ci dans Construire un Système Linux Minimum à partir du Code Source
* Les pages de manuel de mount, fstab, fsck, mke2fs et proc.
* Le fichier `Documentation/proc.txt`, distribué avec les sources du noyau Linux, décrit le fonctionnement du système de fichier `/proc`.
* La page principale des utilitaires du système de fichier Ext2 ext2fsprogs. On y trouve également un document donnant une vue d'ensemble d'Ext2fs, bien qu'il ne soit plus à jour, et moins lisible que le chapitre 9 du livre « Le noyau Linux ».
* Le standard de système de fichier 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`, et cætera. C'est une bonne référence si votre objectif est un système minimal mais complet.

8. Les démons du noyau

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. Les informations proviennent du système de fichiers `/proc` que j'ai mentionné dans la section précédente. 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 ?

Il s'agit des démons propres au noyau. La majeure partie du noyau n'apparaît même pas dans la liste des processus, et le seul moyen de connaître la mémoire qu'il utilise est de soustraire la mémoire disponible à la quantité totale de mémoire installée. Les démons du noyau sont démarrés après init, et obtiennent de ce fait des numéros de processus normaux, mais leur code et leurs données n'existent qu'au sein de la zone de mémoire occupée par le noyau.

Les noms des démons du noyau dans la liste sont écrits entre crochets car le système de fichiers `/proc` ne contient pas d'information sur la ligne de commande utilisée pour lancer ces processus.

Alors, à quoi servent ces démons ? Les versions précédentes de ce document présentaient ici un appel à contribution, car mes connaissances dans ce domaine sont limitées. L'explication (partielle) qui suit est une fusion de différentes réponses à cet appel, pour lesquelles j'ai une grande reconnaissance. Toutes indications, références et corrections sont bienvenues.

Toutes les entrées et sorties sont effectuées via des *tampons* en mémoire, ce qui accélère grandement les choses. Tout ce qu'un programme écrit peut être conservé en mémoire, puis être écrit sur le disque par blocs plus grands et plus efficaces. C'est le travail des démons kflushd et kupdate.

Très souvent, les processus sont au repos, et ceux qui tournent n'ont pas besoin d'avoir l'intégralité de leur code et de leurs données chargée en mémoire. Cela signifie que l'on peut faire un meilleur usage de notre mémoire vive, en faisait glisser les pages inutilisées des programmes en cours d'exécution vers la ou les partitions d'échange du disque dur. Le transfert des données depuis et vers la mémoire selon les besoins est assuré par kpiod et kswapd. Toutes les secondes environ, kswapd se réveille et vérifie la situation de la mémoire, et, si une page devient nécessaire en mémoire ou que l'on commence à manquer de place, invoque kpiod.

Il peut aussi exister un démon kapmd si vous avez activé la gestion avancée de l'énergie (apm) dans votre noyau.

8.1. Configuration

Le programme **update** vous permet de configurer kflushd et kswapd. Essayez update −h pour avoir plus d'information.

L'espace d'échange (*swap*) est mis en service avec **swapon** et hors service par **swapoff**. Ce sont les scripts d'initialisation `/etc/rc.sysinit` ou `/etc/rc.d/rc.sysinit` qui appellent généralement **swapon**, lorsque le système démarre. J'ai entendu dire que **swapoff** était pratique pour économiser l'énergie des ordinateurs portables.

8.2. Exercices

Faites un **update −t**, notez les commentaires des dernières lignes parlant de « threshold for buffer fratricide » (seuil fratricides des tampons). Voilà un concept bien intriguant ! Enquêtez !

Entrez dans le répertoire `/proc/sys/vm` et faites un **cat** sur tous les fichiers s'y trouvant. Voyez ce que vous pouvez en tirer.

8.3. Aller plus loin

Le livre « Le noyau Linux » du projet de documentation Linux (LDP). Suivre les liens dans la section aller plus loin − le noyau Linux.

Le code source du noyau, si vous êtes courageux ! Le programme source de kswapd se trouve dans `linux/mm/vmscan.c`, ceux de klushd et kupdate se trouvent eux dans `linux/fs/buffer.c`.

9. Le journal système

Init démarre les démons syslogd et klogd. Ils écrivent les messages à consigner dans le journal système. 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 sur votre système. Vous y trouverez souvent de précieux indices.

9.1. Configuration

Le fichier `/etc/syslog.conf` indique au démon de gestion du journal système 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.

> Syslog a besoin que le fichier `/etc/services` existe. Ce fichier alloue des ports (UDP et TCP). Je ne sais pas vraiment si syslog a besoin d'un port réservé uniquement pour pouvoir enregistrer les messages des machines distantes, ou si même l'enregistrement en local se fait au travers d'un port, ou même s'il ne se contente pas d'utiliser `/etc/services` pour convertir les noms de services indiqués dans

`/etc/syslog.conf` en numéros de port.

9.2. Exercices

Jetez un Sil 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. Aller plus loin

Le miroir français 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 connexion. 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 l'interpréteur de commandes (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 paquet util−linux, qui contient aussi un getty nommé agetty, qui fonctionne bien. Ce paquet 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 informations spécifiques à l'utilisateur dans `/etc/passwd`, et si vous utilisez un fichier de mot de passe ombre (*shadow password*), 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 interpréteur de commandes 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 la capacité de finir automatiquement les noms de fichiers que vous entrez (lorsque vous utiliser la touche **Tab**).

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 l'invite 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 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'occurrence 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 paquet distinct qui prend en main les interfaces de ligne de commande, en fournissant l'historique des commandes, et la capacité à terminer automatiquement de noms de fichiers, 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`. Ce qui signifie que les touches **Effacement arrière**, **Suppr**, **Début** et **Fin** 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 propre éditeur.

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. Aller plus loin

- Il existe le « Manuel de référence Bash », clair, mais assez lourd.
- Il existe également un livre O'Reilly sur Bash, je ne sais pas s'il est bon.
- Je ne connais pas de bon tutoriel bash gratuit et à jour. Si vous en connaissez un, merci de me faire connaître le lien.
- Le code source. Suivre les liens dans Construire un système Linux Minimum à partir du Code Source

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 à l'interpréteur de commandes.

Les commandes viennent de paquets, la plupart de la Free Software Foundation (projet GNU). Plutôt que de dresser ici la liste des paquets, je préfère vous renvoyer vers le Comment faire un système Linux à partir de zéro. Il contient une liste complète et à jour de tous les paquets allant dans un système Linux, aussi bien que des indications pour les construire.

13. 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.

14. Section administrative

14.1. Copyright

14.2. Page principale

Les mises à jour de ce document évoluent sur le site From Powerup To Bash Prompt avec son compagnon « Building a Minimal Linux System from Source Code ».

Il existe une traduction française sur From Powerup to Bash Prompt. Merci à Dominique van den Broeck. Une traduction japonaise par Yuji Senda est en cours et sera disponible sur le site du Projet des documentations et FAQ japonaises si elle ne s'y trouve pas déjà.

14.3. Réactions

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

14.4. Références et remerciements

Les noms de produits cités sont des marques déposées par leurs 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.

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 l'expression « rolling your own » (réaliser soi−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.

Dominique van den Broeck
: Pour avoir traduit cette doc en français. (NdT : Merci à Guillaume Allègre et Anthony Boureux pour la relecture ainsi qu'à tous les membres de projet de traduction linux français).

Matthieu Peeters
: Pour ses très bonnes informations sur les démons du noyau.

John Fremlin
: Pour ses très bonnes informations sur les démons du noyau.

Yuji Senda
: Pour la traduction en japonais.

Antonius de Rozari
: Pour avoir apporté une version en assembleur GNU de Unios (voir la section ressources sur la page principale).

14.5. Historique des changements

0.8 −> 0.9 (novembre 2000)
: Incorporation des informations de Matthieu Peeters sur les démons du noyau et le système de fichiers `/proc`.

0.7 −> 0.8 (septembre 2000)
: Suppression des informations sur la manière de construire un système, pour les placer dans un document distinct. Correction de quelques liens en conséquence.

  Changement de site internet: de learning@TasLUG vers mon propre site.

  Impossible d'incorporer toutes les bonnes informations reçues d'horizons variés. La prochaine fois, peut−être :(

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.

0.5 −> 0.6
: Ajout de l'historique des changements

Ajout de quelques éléments dans la liste des améliorations prévues.

14.6. Améliorations prévues

- 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.

14.7. Adaptation française

14.7.1. Traduction

La traduction française de ce document a été réalisée par Dominique van den Broeck `<dvandenbroeck@free.fr>`, mai 2000 (v0.7), février 2001 (v0.9).

14.7.2. Relecture

La relecture de ce document a été réalisée par Jean−Philippe Guérard `<jean-philippe.guerard@laposte.net>`. Les version précédentes ont été relues par Guillaume Allègre et Anthony Boureux.